



UNIVERSITY OF BIRMINGHAM

DOCTORAL THESIS

Automatic Identification of Mechanical Parts
for Robotic Disassembly Using Deep Neural
Network Techniques

Author:

Senjing ZHENG

Supervisor:

Marco CASTELLANI

*A thesis submitted in fulfillment of the requirements
for the degree of*

DOCTOR OF PHILOSOPHY

in the

Department of Mechanical Engineering
University of Birmingham
UK, B15 2TT

03 September 2021

UNIVERSITY OF
BIRMINGHAM

University of Birmingham Research Archive

e-theses repository

This unpublished thesis/dissertation is copyright of the author and/or third parties. The intellectual property rights of the author or third parties in respect of this work are as defined by The Copyright Designs and Patents Act 1988 or as modified by any successor legislation.

Any use made of information contained in this thesis/dissertation must be in accordance with that legislation and must be properly acknowledged. Further distribution or reproduction in any format is prohibited without the permission of the copyright holder.

Acknowledgements

I am with mixed feelings writing this part in the thesis now, which means my journey of being a PhD student is coming to an end. This memorable journey can be derived from 2016, when I attended the "Intelligent System" module led by Dr. Marco Castellani, it was my first time came to the world of artificial intelligence, and I was immediately attracted by this amazing world of AI. Luckily Dr. Castellani then gave me a chance to work with him on a small research topic in artificial neural network for my MSc final project. And I can still remember that summer afternoon when I went to his office and asked for an opportunity to take a further study on PhD. With no doubt, Dr. Castellani was not only my supervisor but also a mentor guided me into the field of artificial intelligence, not to mention the efforts he devoted to me in the past years of my research journey. Hereby, I would like to give my sincere thanks to Dr. Castellani for his wise guidance which led my research journey and his endless supports which kept me from falling off.

My sincere thanks also goes to Prof. Duc Truong Pham for the insightful comments and inspirational encouragement he made regarding my research progress during these years.

Besides, I will give my thanks to two of my colleagues Dr. Luca Baronti and Mr. Feiying Lan, for the stimulating discussions we had in office S10, for the priceless advisories and ideas you have gave, and for the efforts we have made together for publications.

I thank the EPSRC-sponsored project (Grant No. EP/N018524/1) AUTOREMAN, which gave me access to the laboratory and research facilities. And I also thank the fellows in AUTOREMAN project: Dr. Yongjing Wang, Dr. Jun Huang, Dr. Mairi Kerin, Dr. Joey Lim, Mr. Mo Qu, and Ms. Kaiwen Jiang for the time we spent and worked together to support the community of remanufacturing.

I would also like to thank my best friends Congkai, Yezi and Jiaozi for their always support. Appreciation also goes to Tuma and Ganli who had my back during the pandemic, and to Weijia and Zhiwen who always inspiring me.

Last but not the least, I would like to give my sincere thanks to my parents for their endless love and support throughout my life, and to my wife for her love, support and companion.

Abstract

This work addressed the automatic visual identification of mechanical objects from 3D camera scans, and is part of a wider project focusing on automatic disassembly for remanufacturing. The main challenge of the task was the intrinsic uncertainties on the state of end-of-life products, which required a highly robust identification system. The use of point cloud models implied also the need to deal with significant computational overheads.

The state-of-the-art PointNet deep neural network was chosen as the classifier system, due to its learning capabilities, suitability to processing 3D models, and ability to recognise objects irrespective of their pose. To obviate the need for collecting a large set of training models, it was decided that PointNet was to be trained using examples generated from 3D CAD models, and used on scans of real objects. Different tests were carried out to assess PointNet ability to deal with imprecise sensor readings and partial views. Due to restrictions on access due to the pandemic, it was not possible to collect a sufficiently systematic set of scans of physical objects in the lab. Various tests were thus carried out using combinations of CAD models of mechanical and everyday objects, primitive geometric shapes, and real scans of everyday objects from popular machine vision benchmarks. The investigation confirmed PointNet's ability to recognise complex mechanical objects and irregular everyday shapes with good accuracy, generalising the results of learning from geometric shapes and CAD models. The performance of PointNet was not significantly affected by the use of partial views of the objects, a very common case in industrial applications. PointNet showed some limitation when tasked with recognising noisy scenes, and a practical solution was suggested to minimise this problem.

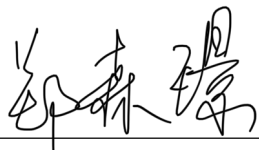
To reduce the computational complexity of training a deep architecture using large data sets of 3D scenes, a predator-prey coevolutionary scheme was devised. The proposed algorithm evolves subsets of the training set, selecting for these subsets the most difficult examples. The remaining training samples are discarded by the evolutionary procedure, which thus reduces the number of examples that are presented to the classifier. The experimental results showed that this economy of training samples allows reducing the execution time of the learning procedure,

without affecting the neural network recognition accuracy. This simplification of the learning procedure is of general importance for the whole deep learning field, since practical implementations are often hindered by the complexity of the training process.

Declaration

I, Senjing Zheng, hereby declare that this Ph.D. thesis entitled “Automatic Identification of Mechanical Parts for Robotic Disassembly Using Deep Neural Network Techniques” was carried out by my own for the degree of Doctor of Philosophy in the University of Birmingham. I confirm that:

- The presented work has never been previously included in a thesis or dissertation submitted for a degree or other qualifications.
- Where the thesis is based on joint works done by myself with others, a clear statement has been made to illustrate how the contribution was exactly distributed.
- Except where states otherwise by reference or acknowledgment, the work presented is entirely composed by myself.

Signed:  _____

Date: 12-04-2022

Contents

1	Introduction	1
1.1	Background	3
1.1.1	Robotic Disassembly for Remanufacturing	3
1.1.2	DNN Identification of Mechanical Parts	5
1.1.3	Evolutionary and Coevolutionary DNN Training	6
1.2	Aims of the Thesis	7
1.3	Outline of this Thesis	7
2	Literature Review	9
2.1	Automatic Disassembly for EOL Products	9
2.2	Object Recognition In Industry	12
2.3	Point Cloud Classification	14
2.3.1	Hand-crafted Feature Descriptors	14
2.3.2	DNN-based Point Cloud Classification	15
2.4	Evolving Artificial Neural Network	18
2.4.1	Evolving Network Weights	18
2.4.2	Evolving Network Architecture	19
2.4.3	Evolving Input Features and Sub-dataset	21
2.4.4	Co-evolving ANN Training	21
3	Automatic Identification of Mechanical Parts for Robotic Disassembly Using the PointNet Deep Neural Network	25
3.1	The Identification System	25
3.1.1	The PointNet Deep Neural Network	27
3.1.2	The Depth Camera Simulator	30
3.2	The Mechanical Objects Model Set	32
3.2.1	The Clean model set	34
3.2.2	Down-sampling	34
3.2.3	Normalisation	35
3.2.4	The Error model set	35
3.3	Experimental Design	36

3.3.1	Partial View and Sampling	36
3.3.2	Tolerance to Error	37
3.3.3	Part Specific Classifiers	38
3.4	Experimental Results	38
3.4.1	Partial View and Sampling	39
3.4.2	Tolerance to Error	42
3.4.3	Part Specific Classifiers	46
3.5	Discussion	47
3.6	Conclusions	49
4	A Detailed Evaluation of PointNet Capabilities	51
4.1	Data Sets Used	52
4.1.1	The Yale-CMU-Berkeley (YCB) Object and Model Set	52
4.1.2	Artificial Primitive Shapes	55
4.1.3	YCB-similar Artificial Primitive Shapes	56
4.1.4	All-in-One Set	58
4.2	Shallow Neural Network Architectures	58
4.2.1	Multi-Layer Perceptron	59
4.2.2	Radial Basis Function Network	60
4.2.3	Numerical Feature Generation	61
4.3	Experiments and Results	62
4.3.1	Hyper-parameters Optimisation for PointNet	64
4.3.2	Experimental Results - Artificial Model Sets	66
4.4	Discussion	68
4.5	Conclusions	69
5	Coevolutionary Deep Neural Network Training	71
5.1	Coevolutionary ANN Training	72
5.2	Coevolutionary Selected Training (CEST)	74
5.3	Experimental Model Sets	77
5.4	Experimental Results	79
5.4.1	Artificial Primitive Shapes and YCB-28	79
5.4.2	Effect of the Parameterisation on CEST Performance	82
5.4.3	MEch12 and ModelNet40	91
5.5	Discussion	92
5.6	Conclusions	93
6	Conclusions	95
6.1	Summary of Achievements	96
6.2	Limitations and Future Work	96
6.3	Publications Arisen from this Thesis	97

CONTENTS

ix

A Manually Measured Shape Features

99

List of Figures

3.1	Main structure of PointNet	27
3.2	Layout of designed scene (Zheng et al., 2022)	31
3.3	Two turbochargers used in experiments (Zheng et al., 2022)	32
3.4	Twelve mechanical parts used in experiments (Zheng et al., 2022), where: C-bearing represents compressor bearing; T-housing represents turbine housing	33
3.5	Visualisation of PCs of <i>C-housing-A</i> with 1000 points and corrupted with different level of error (visualised with Open3D (Q.-Y. Zhou et al., 2018))	36
3.6	Classification accuracy of experiments using PCs from <i>full-view</i> (12cam) and <i>partial-view</i> (3cam) scenes and with different number of sampled points: 1000 (1k) and 2000 (2k) points	40
3.7	Learning curve of networks trained by <i>partial-view clean</i> set with 1000 sampling points	41
3.8	Learning curve of networks trained by <i>partial-view clean</i> set with 2000 sampling points	41
3.9	Results on test sets of different error level, when the PointNet is trained using clean (zero error) scenes	43
3.10	Results on test sets of different error level, when the PointNet is trained using scenes with 5% error level	43
3.11	Plots of training and testing accuracies of PointNets trained with <i>clean</i> set and tested with 5%-error set	45
3.12	Plots of training and testing accuracies of PointNets trained with 5%-error set and tested with 5%-error set	45
3.13	Plots of training and testing curves of PointNets trained with 10%-error set and tested with 5%-error set	45
4.1	The the nine selected box-like objects (images and meshes) from YCB set	53
4.2	All the eight selected cylinder-like objects (images and meshes) in <i>YCB-28</i> from YCB set	53

4.3	All the eleven selected sphere-like objects (images and meshes) in <i>YCB-28</i> from <i>YCB</i> set	54
4.4	Example of created box-shape PC with 1000 points, from left to right: PC without any error, and PC with 5% error (visualised with Open3D (Q.-Y. Zhou et al., 2018))	56
4.5	Comparison of <i>YCB</i> meshes and hand-crafted PCs	57
4.6	Performance of PointNet on the <i>APS-clean</i> model set as the batch size setting is varied	64
4.7	Performance of PointNet on the <i>APS-clean</i> model set as the number of training epochs is varied	65
4.8	Performance on the <i>YCB-28</i> model set of PointNet trained using the <i>APS-all</i> set with different training epoch	66
4.9	Accuracy results obtained on the <i>YCB-28</i> model set by the three classifiers when trained using different sets: <i>APS-clean</i> (Clean), <i>APS-error</i> (Error) and <i>YCB-similar</i> (Similar)	67
4.10	Performance comparison on the <i>YCB-28</i> model set of PointNet trained using the <i>APS-error</i> (Error) and <i>APS-all</i> (All) set	68
5.1	Flow chart of CENNT proposed by Castellani (2018)	73
5.2	Flow charts of CEST	74
5.3	Box-plot of the classification results obtained training PointNet on different model sets, and testing its accuracy on the <i>YCB-28</i> set	82
5.4	Plots of number of PCs used per epoch to train and evaluate PointNet, in relation to different settings of population parameters	84
5.5	Plots of training samples utilised per training epoch for different mutation rates	87
5.6	Number of employed training patterns per training epoch in experiments with small populations and low mutation rates	87
5.7	Plots of number of training scenes used per epoch for different settings of the fitness weights	89
5.8	Box-plot of the classification accuracies obtained on the MEch12 and ModelNet40 sets with Adam and CEST learning algorithm, whilst median accuracies are shown within parentheses	92

List of Tables

3.1	Parameters of <i>Adam</i> optimisation algorithm in PointNet	28
3.2	Hyper-parameters of PointNet used in the experiments	37
3.3	Details of the computer used for experiments	39
3.4	Training time and accuracy obtained in the experiments where PointNet was trained using the <i>partial-view-1k</i> and <i>partial-view-2k</i> model sets. The training time was recorded from one sample training procedure with PC described in Table 3.3	40
3.5	Average (median) accuracies (%) obtained training the PointNet using the clean and 5%-error training sets, and tested on model sets with different levels of error. The last column reports the p-values of pairwise Mann-Whitney tests. Where statistical significance was found (p-value smaller than 0.01), superior accuracy results were highlighted in bold	44
3.6	Performance of classifiers trained using <i>clean</i> set, and tested on data with zero error (<i>clean</i> set)	47
3.7	Performance of classifiers trained using <i>clean</i> set, and tested on data with 5% error	48
3.8	Performance of classifiers trained using <i>clean</i> set, and tested on data with 10% error	48
4.1	IDs and names of the selected 28 objects from YCB set	54
4.2	Manually measured shape features of three mesh models from <i>YCB-28</i> set	58
4.3	Hyper-parameters of PointNet used in the experiments (please refer to Section 3.1.1 of a detailed description of these parameters)	63
4.4	Hyper-parameters of MLP and RBFN used in the experiments	63
5.1	Hyper-parameters related to the management of the genes in CEST	76
5.2	<i>GA</i> hyper-parameters of the prey module in CEST	77
5.3	Details of employed APS and YCB model sets	78
5.4	Details of MEch12 and ModelNet40 model sets	78

5.5	Hyper-parameters of PointNet used in the experiments (please refer to Section 3.1.1 of a detailed description of these parameters)	79
5.6	Setting of CEST hyper-parameters used for training PointNet using the <i>APS-error</i> model set and testing its performance on the YCB model set	80
5.7	Setting of CEST hyper-parameters used for training PointNet using the <i>APS-all</i> model set and testing its performance on the YCB model set	80
5.8	Experimental results obtained training PointNet on different model sets, and testing its accuracy on the <i>YCB-28</i> set	81
5.9	Matrix of p-values of pairwise Mann-Whitney tests performed on the learning results of different combinations of training sets and algorithms. The medians of the accuracy results are reported in Table 5.8	82
5.10	The three groups of parameters in CEST	83
5.11	Experimental design and results for varying population parameters, where the p-value was calculated against results made by PointNet trained with <i>APS-all</i> using Adam algorithm as shown in Table 5.8	83
5.12	Experimental design and results for varying mutation parameters, where p-value was calculated against results of error-Adam Table 5.8	85
5.13	Experimental results for low mutation rates and small population sizes	85
5.14	Basic hyper-parameters of experiments studying fitness weights	88
5.15	Experimental design and results for fitness weights	90
5.16	The best-so-far experimental results obtained training PointNet on <i>APS-error</i> and <i>APS-all</i> sets, and testing on the <i>YCB-28</i> set	90
5.17	Best-tuned CEST parameters setting for <i>APS</i> sets	91
5.18	CEST parameters setting for the experiments on the <i>MEch12</i> and ModelNet40 model sets	91
5.19	Results obtained by training PointNet using the CEST and <i>Adam</i> optimisers, on the <i>MEch12</i> and ModelNet40 model sets; where Time is the measured real execution time of one training epoch under RTX-3090 GPU	92
A.1	Manually measured shape features of the twenty-eight selected objects from YCB set	100

Glossary

ANN Artificial Neural Network. 2, 3, 6, 7, 9, 13, 18, 19, 21–23, 29, 51, 52, 69, 72, 73

APS Artificial Primitive Shapes. xii–xiv, 55, 56, 58, 62, 64–68, 77–83, 86, 88, 90, 91

CAD Computer-Aided Design. 2, 5, 7, 25–27, 32, 34, 37, 49, 51, 64, 95, 96

CANNT Co-Adaptive artificial Neural Network Training. 6, 72, 73

CENNT Co-Evolutionary artificial Neural Network Training. xii, 3, 6, 72–74

CEST Co-Evolutionary Selected Training. xii–xiv, 3, 6, 7, 74–84, 86, 88, 90–97

CGA Co-evolutionary Genetic Algorithm. 3, 6, 22, 23, 72, 73

CNN Convolutional Neural Network. 5, 13–18, 20

DNN Deep Neural Network. 1, 3–7, 9, 12, 15, 17, 18, 20, 26, 27, 58, 59, 64, 67, 69, 71, 72, 74, 79, 88, 92–97

EA Evolutionary Algorithm. 3, 6, 9, 18–21, 23, 72, 76, 93

EOL End Of Life. 1, 3, 4, 9, 10, 12, 95

GA Genetic Algorithm. xiii, 19, 21–23, 72, 73, 75–77

GPU Graphics Processing Unit. xiv, 40, 71, 92

IQR InterQuartile Range. 39, 81, 83, 85, 90, 91

MLP Multi-Layer Perceptron. xiii, 17, 18, 23, 27, 51, 59–61, 63, 64, 66–69

PC Point Cloud. xi, xii, 1, 2, 5–7, 9, 14–18, 26–28, 31, 32, 34–37, 40–42, 47, 49–52, 54–59, 61, 62, 65, 69, 72, 74, 75, 77, 78, 81, 83, 84, 86, 93, 95

RBFN Radial Basis Function Network. [xiii](#), [52](#), [59–61](#), [63](#), [64](#), [66](#), [67](#)

SNN Shallow Neural Network. [20](#), [58](#), [59](#), [62](#), [63](#), [66–68](#), [95](#)

YCB Yale-Cmu-Berkeley. [xi–xiv](#), [2](#), [3](#), [52–58](#), [62](#), [64](#), [66–69](#), [77–82](#), [90](#), [91](#), [95](#), [97](#), [100](#)

Chapter 1

Introduction

With the rising concerns over environmental issues and the depletion of natural resources, remanufacturing procedures have attracted increasing interest in engineering (Harper et al., 2019). A crucial step in the remanufacturing process is the disassembly of end-of-life (EOL) products. Due to the labour-intensive and sometimes hazardous nature of disassembly operations, since the late 1990's many studies focused on the creation of automatic or robotic disassembly procedures (Jovane et al., 1993; J. Li, Barwood, & Rahimifard, 2018).

The very first step in automatic disassembly is the identification of the objects, their pose, and their components. Unlike assembly, the disassembly task is fraught with uncertainties on the integrity and state of the objects and their parts (Vongbunyong & Chen, 2015). These can be stained, corroded, deformed, damaged, or missing. Devices of different model and manufacturer might have to be sorted. These uncertainties require robust sensing systems able to recognise objects, shapes, and sub-assemblies (Wegener et al., 2015; Bdiwi et al., 2016).

Machine vision is customarily a primary source of information on the environment. The increasing availability of affordable cameras, scanners, and related hardware has created the conditions for its widespread use in engineering and beyond. In parallel, deep neural network (DNN) techniques raised the accuracy of automatic recognition algorithms to levels never reached by traditional techniques (LeCun et al., 2015). DNNs owe their popularity not only to their accuracy, but also to the fact that they automate the labour-intensive and time-consuming steps of image segmentation, feature extraction, and selection (Z.-Q. Zhao et al., 2019).

This thesis details a study on the identification of object and shapes from 3D point cloud (PC) models. The application is the recognition of mechanical parts for disassembly and remanufacturing. PointNet (Qi, Su, et al., 2017), a recently developed DNN system for identification and segmentation of PC scenes, is the object of the study. PointNet was chosen due to its ability to recognise objects irrespective of their pose, whereas the performance of many DNNs is not orientation

invariant.

The first piece of research concerns the application of PointNet to the identification of mechanical parts of automotive devices. The first task in training any kind of artificial neural network (ANN) is to collect a large and diversified set of training examples. This task is time-consuming, and might need to be periodically repeated as new products are introduced in the market (Krueger et al., 2019; Börold et al., 2020). Therefore, it was decided to investigate the feasibility of training PointNet using artificial scenes generated from computer-aided design (CAD) models. For this purpose, a 3D camera simulator was developed to automatically generate the training examples. CAD models of twelve mechanical parts from two types of internal combustion engine turbo-charger were used in this study.

The performance of PointNet was evaluated in terms of its robustness to error (local sensor error in the simulated scanner readings), and partial view of the objects. The experimental tests confirmed the feasibility of the approach, and revealed some degree of sensitivity of PointNet to local depth scanning error. It was also found that, in presence of local error in the scans, the accuracy of PointNet could be improved by injecting some sensor error in the training scenes.

The second piece of research investigated the ability of PointNet to recognise primitive shapes in everyday objects. This study has direct application to many engineering problems, as is often the case that mechanical objects have fairly regular shapes (e.g. the head of a piston, the balls of a rolling bearing). In this case, the performance of PointNet was tested on real scans of everyday objects from the Yale-CMU-Berkeley (YCB) benchmark set (Calli et al., 2015), a popular robotics benchmark. The use of real scans of physical objects constitutes a more realistic setting than the CAD-generated images used in the previous tests, and an advancement on similar tests performed on artificial scans by the creators of PointNet (Qi, Su, et al., 2017). The fact that PointNet had never been evaluated on real-life 3D PCs classification set was first pointed out by Garcia-Garcia et al. (2017), and acknowledged by Uy et al. (2019) who manually built the ScanObjectNN set. ScanObjectNN contains camera scans of physical objects grouped in categories modelled on the popular ModelNet40 benchmark set. In their study, Uy et al. (2019) reported very poor classification accuracy (32.2%) when the the PointNet was trained using ModelNet40 and tested on the ScanObjectNN set.

In the work of this thesis, PointNet was trained using various sets of artificially generated PCs of primitive shapes, and tasked with recognising the irregular shape of the YCB models. These tests directly addressed the question on the possibility of training PointNet on artificial data, and use it in real-life applications. The YCB set contains 3D images captured from real-life objects using high-resolution RGB-D cameras. It was originally designed for robotic manipulation research, and contains PCs models of 77 daily-life objects divided in five main categories. From

these objects, 28 instances were selected based on their shape. They were grouped into three classes: boxes, cylinders, and spheres. Due to sensor imprecision and the intrinsic irregularity of the objects (e.g. an orange as an instance of sphere), the 28 models from the [YCB](#) set only approximated the ideal primitive shapes. The results obtained using PointNet were compared with those obtained using two shallow [ANNs](#), using purpose-designed feature extraction algorithms. The experimental results confirmed the overall good performance of PointNet, but also some limitation in generalising the learning results to the irregular [YCB](#) shapes. Also in this case, it was found that injecting imprecision (sensor error) in the training models the accuracy of PointNet could be improved.

The third piece of research aimed at reducing the computational overheads of training PointNet. Although targeted to the specific model considered in this thesis, this piece of research was relevant to the whole [DNN](#) field. At present, the cost of training large and complex architectures on computationally demanding 2D images or 3D models, is one of the main hindrances to the wider application of [DNNs](#) ([Z.-Q. Zhao et al., 2019](#)). Some authors tried to address this problem by heuristically selecting the training examples based on their evaluation from loss function ([Loshchilov & Hutter, 2015](#)). In this study, a meta-heuristic coevolutionary predator-prey approach was investigated. In this scheme, an evolutionary algorithm ([EA](#)) was used to select from the training set of examples, the most difficult instances (prey), that is the yet not learned examples. These instances were used to train PointNet (the predator) using the customary *Adam* optimiser ([Kingma & Ba, 2014](#)).

Modelled on the Coevolutionary Genetic Algorithm ([CGA](#)) proposed by [Paredis \(1995\)](#), and the CoEvolutionary [ANN](#) Training algorithm ([CENNT](#)) proposed by [Castellani \(2018\)](#), the CoEvolutionary Selected Training ([CEST](#)) algorithm could train PointNet using only a subset of the training models. It was demonstrated that this economy of training instances allowed speeding up the execution time of the PointNet learning process, without affecting its classification accuracy.

1.1 Background

In this section, the background of the research topics relevant to this thesis is presented.

1.1.1 Robotic Disassembly for Remanufacturing

Mass production and advances in automatic manufacturing have brought a great improvement into people's life. Unfortunately, the larger is the quantity of products manufactured by factories, the larger is the amount of [EOL](#) products that will

require disposal in the future. To address concerns on the environmental impact of waste disposal, and the depletion of natural resources, interest in remanufacturing of EOL products is rapidly rising (Harper et al., 2019).

Differently from recycling or refurbishing, remanufacturing is a more complex procedure, but also environmental more sustainable (Kin et al., 2014). Formally, remanufacturing is the process of disassembling, restoring, and re-assembling EOL products, reverting them back to at least their original performance, with the least environmental impact and financial cost (M. R. Johnson & McCarthy, 2014). In most cases, disassembly is the first step to remanufacturing.

Concerns about the management of EOL products generated the first remanufacturing studies in the 1990's (Dario et al., 1994), when the first instances of automatic disassembly systems were proposed. Since then, the literature in robotic disassembly gained momentum and attention. It will be discussed in Section 2.1.

Typical steps for automatic disassembly operations are in chronological order: the identification of the object to be disassembled, its localisation (position and orientation), the planning of robotic trajectories, and finally the manipulation and taking apart of the object's subsystems and components (Vongbunyong & Chen, 2015). The work reported in this thesis concerns the first step.

In traditional manufacturing procedures like automatic assembly lines, mechanical parts are newly manufactured without any damage or stain. However, in remanufacturing EOL products are usually in unknown state, and can be dirty, corroded, deformed or missing parts. Prior visual inspection is therefore of paramount importance, usually challenging, and poor performance from the vision system is a major limiting factor for the automation of disassembly processes (Vongbunyong & Chen, 2015). For example, recent work reported by Wegener et al. (2015) focused on the disassembly of the battery pack of Audi Q5 vehicles. A human-robot collaborative system was proposed, where the robotic component was employed for unscrewing parts. The main weakness of the system was the conventional vision module, based on separated feature extraction and identification steps, which only achieved 50% detection accuracy. A similar system for component unscrewing was proposed by Bdiwi et al. (2016), implementing a region growing algorithm over depth and colour information. Although in ideal conditions this system nearly reached 90% accuracy on true positives, and only 15% rates of false positives, Bdiwi et al. (2016) reported poor performances when the screws were rusty.

In summary, given the uncertainties and high variability of conditions of EOL products, state-of-the-art techniques are required for automated visual inspection and recognition. This thesis will study the PointNet DNN as a tool for recognition of mechanical objects and shapes.

1.1.2 DNN Identification of Mechanical Parts

In recent years, **DNN** technology has brought great improvements in machine vision tasks such as object identification and image segmentation. Some pioneering work has probed applications of deep learning to disassembly problems. For example, [Yildiz & Wörgötter \(2019, 2020\)](#) applied a convolutional neural network (**CNN**) for screw detection for automatic disassembly of electronic waste. This kind of **CNN** required a huge number of examples to train the network, and the authors collected in total 20,000 2D images from 500 samples of screw. The recognition system achieved a 97% detection accuracy on the test set of examples, and the whole pipeline (image pre-processing plus detection) an overall 75.7% recognition rate. The main limitation of this approach was its reliance on a very large number of training examples.

The requirement of a large training data set is a common problem in the implementation of **DNN** systems, and was reported by several other authors for implementations of 2D image classifiers in industrial applications ([Weimer et al., 2016](#); [He et al., 2016](#); [Krueger et al., 2019](#)). This requirement is often a limiting factor in the deployment of **DNN** systems in industry, where usually large number of parts are involved.

In this thesis, a different approach was investigated: the **DNN** was trained using artificial scenes created from **CAD** models of mechanical parts. The application was the identification of mechanical objects or shapes in 3D **PC** scenes. Differently from the real objects, 3D **CAD** models of mechanical parts are easily available in industry, either from the production process, or from reverse engineering via laser scanners. Using custom-made software, a large amount of training examples can be effortlessly produced from **CAD** models, including assemblies with missing or deformed parts. Compared with 2D images, 3D images allow more robust identification thanks to their depth information, particularly in the case of superficial blemishes or damage like stain or corrosion.

To the author's best knowledge, there is no previous work in the literature trying to apply **DNN** directly on 3D images to address the identification problems in industrial scenario like disassembly.

The main challenge of this approach was envisaged to be the accuracy of the classifier, in relation also to the representativeness of **CAD**-generated models. For the processing of 3D scenes, three main approaches can be identified in the **DNN** literature: multi-view, volumetric based, and **PC** based.

The multi-view approach ([Su et al., 2015](#)), uses multiple images from different angles of the object. These multiple views will be typically fed to a **CNN** for identification. The volumetric based approach firstly transforms the raw 3D images (typically **PCs**) into 3D volumetric images, and then applies purpose-designed convolutional layers for feature extraction ([Wu et al., 2015](#); [Maturana & Scherer,](#)

2015). The main drawback of this approach is the high computational overheads implied (Qi et al., 2016). Finally, the PC based scheme directly feeds the DNN with raw PC models from 3D camera scans (Qi, Su, et al., 2017).

In this thesis, the PC based scheme was adopted, due to its lower computational overheads and ease of implementation. The PointNet architecture, specifically designed for processing PC models by Qi, Su, et al. (2017), was used as recognition system. The main advantage of PointNet is its ability to recognise patterns independently of their pose. Like all DNNs, it performs both feature extraction and image identification tasks. A more detailed description of the DNN is given in Section 3.1.1.

1.1.3 Evolutionary and Coevolutionary DNN Training

EAs are a class of meta-heuristic optimisation procedures mimicking the process of natural evolution (Fogel, 2006). The main idea behind EAs is to think of candidate solutions as individuals, and evolve a population of them according to a desired optimality criterion. The defining traits (features) of a solution are typically encoded as a string of parameters, which can be numerical or other data structures. Applying mechanisms derived from biological evolution, such as selection, recombination, and mutation, an EA manipulates the population, mixing and altering the traits of the best solutions. EAs have been used in various ANN training schemes, which can be generally classified into three categories: evolving the network weights, evolving the network architecture, and evolving the training patterns (Yao, 1999).

Coevolutionary algorithms are based on two EAs (or other learning algorithms) optimising two populations (species) for complementary purposes. The idea is to mimic predator-prey interactions in a zero-sum game scheme (Hillis, 1990). A typical example is the co-evolution of ANN classifiers (predators) and training data (prey) implemented in various algorithms like CGA (Paredis, 1995), Co-adaptive ANN Training (CANNT) (Castellani, 2018), and CENNT (Castellani, 2018). The goal of the procedure is to evolve subsets of the most difficult training patterns, and use only them to train the classifier, thus reducing the computational overheads of the training process. Experimentally, it was found that coevolutionary algorithms are able to speed up the training process and increase the performance of ANN (Paredis, 1995; Castellani, 2018).

In this thesis, coevolutionary training was used to reduce the computational and time complexity of training PointNet. Indeed, training PointNet customarily takes the repeated presentation to the DNN of a large number of PCs, each of them containing thousands of points, and the periodic update of the very large number of parameters that characterise the network response. The novel CoEvolutionary Selected Training (CEST) algorithm was designed to minimise the number of

training samples fed to the algorithm. **CEST** is presented and its performance evaluated in Chapter 5

1.2 Aims of the Thesis

The aims of this thesis are the following:

- to implement a **DNN**-based automatic system for recognition of complex mechanical parts from 3D **PC** models;
- to investigate the possibility of training the **DNN** using **CAD**-generated or geometric models of objects and shapes, and use it on previously unseen noisy scenes, partial views, and real-life scans;
- to systematically investigate the accuracy of the **DNN**, the consistency of its learning procedure, and the effects of the parameterisation on the results of the learning procedure;
- to create a novel coevolutionary scheme to reduce the complexity of the **DNN** training procedure, by focusing the training effort on the most difficult examples;
- to evaluate the performance of the new coevolutionary procedure.

1.3 Outline of this Thesis

The remaining of this thesis is organised as follows:

- Chapter 2 reviews the literature on automatic disassembly system, object identification techniques in industry, feature-based and **DNN**-based **PC** classification, and evolutionary and coevolutionary **ANN** training;
- Chapter 3 presents the proposed **DNN**-based system for automatic identification of mechanical objects from **PCs**. The chapter hinges on a case study considering twelve mechanical parts from an automotive device;
- Chapter 4 presents a comprehensive examination of PointNet ability of recognising primitive shapes in everyday objects, using a popular benchmark set of **PCs** and three custom-made artificial model sets;
- Chapter 5 describes the proposed coevolutionary **CEST DNN** training algorithm, and its tuning and evaluation on three different model sets.

Chapter 2

Literature Review

In this chapter, literature related to the studies in this thesis will be reviewed. In Section 2.1, previous efforts over developing an automatic disassembly cell for remanufacturing are illustrated. Section 2.2 will review the 2D object recognition technologies used in industrial applications, from feature-based methods to deep neural network based methods. In Section 2.3, the feature-based methods and the deep neural network based methods for PC classification will be reviewed. Section 2.4 will review the literature over EA based training and designing for ANN and DNN. The coevolutionary algorithm based ANN training will also be reviewed.

2.1 Automatic Disassembly for EOL Products

Early concerns over EOL can be found since 1990's (Jovane et al., 1993), where the challenges and the benefits of developing an automatic disassembly cell were pointed out. Addressed by Dario et al. (1994), different from assembly, the main challenge in automatic disassembly is to make the system understand the unstructured scene. To tackle the challenge, they simultaneously used vision from camera and "pushing" motion of robot to recognise the target. The idea was derived from a cooperating robot system built by Vischer (1992), which Dario et al. (1994) recognised as the first trying to disassemble targets.

When it came to the late 1990's, Kopacek & Kopacek (1999) pointed out the urgency to have an automatic disassembly solution for massive number of electric wastes due to the great improvement in information technology and automatic manufacturing technology. They mentioned the challenge in automatic disassembly where the disassembly targets are in low volumes but multiple models. Thus, they proposed to build a modular flexible disassembly cell which can disassemble multiple models of one product. In their further research, Knoth et al. (2002) pro-

posed and built a semi-automatic disassembly cell for printed circuit boards. The printed circuit board was firstly disassembled from the product and fixed in the semi-automatic cell by human-operators. The purpose-built semi-automatic disassembly cell will finish the rest of the disassembly work. One of the key module in their work is a vision system, which capable of recognising and localising the target components with an accuracy of 0.1mm.

Similarly, a human-robot collaborative workstation was proposed by [Karls-son & Järrhed \(2000\)](#) to automatically examine and disassemble the EOL electrical motors. In their work, motors were firstly classified based on their electrical properties into three different classes: re-used, disassemble, and unknown. The developed vision sub-systems can automatically classify the type of motor with 95% accuracy, and localise the orientation of the motor with less than 10° error. Human-operators will then step in to remove the cover for disassembling targets. It can be found that there are lots of approaches in the literature to build the human-robot collaborative system ([R. Li et al., 2020](#)). One reason behind this is the limited level of automation can be achieved. Differently, [J. Huang et al. \(2020\)](#) proposed and established a human-robot collaboration system for high-load disassembly process. In their case, the system will focus more on reducing the safety risk to human operator by assigning the high-load operation to robotic system.

Another massive numbers of wastes in early 21st century is the personal computer. [Torres et al. \(2004\)](#) developed and presented an automatic system to disassemble personal computer. The developed vision sub-system includes a stereo-camera and a robot-armed camera. Using the feature based template matching techniques and the 3D modelling information of the target computer, the vision system can identify the model type of the target computer and also the relationship between components in the target. The disassemble sequence will then be planed and the robotic manipulation trajectory will also be generated. A further research based upon ([Torres et al., 2004](#)) was made by [Gil et al. \(2007\)](#), where a cooperative robot system for automatic disassembly was proposed. Researchers pointed out one of the challenge in disassembly, which is the high uncertainties of the target. To address the difficulty, the project fused multiple sources of sensors: force sensors and a cameras on robot arm, and a stereo-camera on the scene, to establish a robust environmental recognition and detection system. [Gil et al. \(2007\)](#) also proposed to use cooperative robots to increase the success rate of the procedure. A hierarchical disassembly planning algorithm was also developed based on [Torres et al. \(2003\)](#)'s work, taking account of the interaction between multiple robots in the system.

In the field of automotive, [Büker et al. \(2001\)](#) pointed out the increasing demands of building an automatic disassembly station while there are about 10 millions of cars were wrecked each year in the EU. In their understanding, there are

two important tasks for the disassembly automation, which are object recognition and localisation, and robotic manipulation respectively. They focused on wheel disassembly, and developed a stereo-camera based vision system with a special designed unbolting tool. The developed stereo-camera vision system can recognise and localise the target wheels and bolts, with a success rate of 98% within 15 seconds. Highlighted by [Büker et al. \(2001\)](#), with the robustness of the vision system and the special designed unbolting tool, their final developed automatic disassembly cell can be applied on various models of wheel.

Researchers also rose the concerning over the battery and motor in electric vehicles in recent decade. [Wegener et al. \(2014, 2015\)](#) proposed and developed a cooperative robotic cell for automatic vehicle batteries disassembly. Similarly, the challenges of uncertainties in disassembly scenario such as unpredictable screw type, and variants of battery designs were pointed out by the authors. To address the problems, the developed system was equipped with an automatic tool changing module and a vision module. Specifically, the vision system is based on 2D feature-based Haar Cascade technologies ([Viola & Jones, 2001](#)), and can only achieve 50% detection accuracy over target screws. [Bdiwi et al. \(2016\)](#) developed an automatic disassembly cell for motors of electric vehicles. Using two Kinect cameras, the developed system can ensure the safety for human activities, as well as the detection of screws on the target. The proposed feature-based screw detection algorithm will return the type and the location of the screw with no prior knowledge of the motor model. The algorithm can achieve about 90% accuracy with about 85% precision. Similarly, [DiFilippo & Jouaneh \(2017\)](#) proposed to combine vision with force data to identify screws on personal computer. Based on Gaussian blur and edge detection, the proposed system achieve 96.5% detection accuracy.

The above mentioned vision systems were based on traditional feature-based technology. In many cases, the use of this technology limited the performance of the vision system, and hence the whole disassembly procedure ([Büker et al., 2001](#); [Torres et al., 2004](#); [Gil et al., 2007](#); [Wegener et al., 2014, 2015](#); [Bdiwi et al., 2016](#); [DiFilippo & Jouaneh, 2017](#)).

In recent years, researchers applied deep neural network based vision technology to disassembly problems, attracted by their high accuracy and robustness. [Yildiz & Wörgötter \(2019, 2020\)](#) developed a screw detection and classification system based on a deep convolutional neural network, and demonstrated its effectiveness in a hard drive disk disassembly case study. Their algorithm was able to detect the screws in the scene with an 80.23% average accuracy, and identify their type and size with 75.7% average accuracy. The creation of the neural network training set of examples entailed a large effort, where 20,000 sample images of 500 screw elements were collected from 50 hard disk drives.

[Foo et al. \(2021\)](#) used deep learning for screw detection in an LCD monitor

disassembly application. The system used an image preprocessing procedure, an ontology reasoning module, and a Fast-RCNN network (Ren et al., 2015). The training dataset was built combining numerous images of screws acquired via an extensive Google search, plus 356 manually acquired images. A total of 1,496 bounding boxes around the screw samples had to be manually labelled in the images. The Fast-RCNN network was pre-trained using the COCO dataset (Lin et al., 2014), using transfer learning (Weiss et al., 2016) to speed up the screw detection training procedure. The trained system achieved 91.8% precision and 83.6% recall rates.

Similarly, X. Li et al. (2021) applied the fast Region-Convolution neural network (R-CNN) to detect screws on motherboards of mobile phones for disassembly. Trained with 72 samples, the vision system achieved a classification accuracy of 99.64% over another set of 72 previously unseen samples. Brogan et al. (2021) proposed a vision system based on the Tiny YOLO v2 (Tiny-You Only Look Once v2) pretrained DNN architecture, to identify screws on electrical waste for disassembly. The system achieved over 92% recognition accuracy using 900 manually collected training images. Rehnholm (2021) used a YOLO v4 architecture to build the vision system for a battery package disassembly application. The trained DNN obtained a final classification precision of 93.8%, with A recall rate of over 80%.

In summary, efforts towards automated recycling of EOL products started in the 1990's, and became a major research topic in the recent decade. Most researches addressed machine vision as a priority for building automatic disassembly cells, due to the uncertainties on EOL products. Vision is the first step in automatic disassembly, and should allow accurately recognising and localising objects to make sure the success of the full disassembly procedure. In recent years, some researchers proposed to apply DNN technology to address the machine vision challenge. However, their work required a significant effort for data preparation, image capture, and image labelling.

2.2 Object Recognition In Industry

In recent decades, most of the object recognition implementations in industry depends on template matching technologies over 2D images. In specific industrial case, special manual-designed algorithm will be applied to extract features from target images to ensure efficiency and reliability at the same time.

However, not every case has an efficient solution in feature extraction, not to mention the robustness of the solution. Consequently, researchers are looking for an efficient and reliable solution for the general feature extraction algorithm. Lowe (1999) proposed a feature extraction method namely Scale Invariant Feature Transform (SIFT) for 2D images. Specifically, the extracted features are in-

variant to the image rotation and scaling. Using nearest-neighbour indexing and some search algorithm, like best-bin-first search algorithm (Beis & Lowe, 1997) in Lowe (1999)'s original effort, the matched candidate class from templates can be determined. Proposed by Bay et al. (2006), SURF (Speeded Up Robust Features) feature descriptor brings an improvement to the SIFT, regarding computing speed and robustness. Although both SIFT and SURF brought a great leap in the robustness for novel feature descriptor algorithms, they are still far away from being implemented in the real-time application in industry. Further researches like FAST (Rosten & Drummond, 2006), BRIEF (Calonder et al., 2010), and ORB (Rublee et al., 2011), were all trying to improve the computing speed comparing SIFT and SURF.

ANN is a well-known machine learning technology for function modelling and pattern classification, especially after the propose of back-propagation learning algorithm (Rumelhart et al., 1985). Early approaches of ANN in object classification and pattern recognition require feature extraction in the pre-processing steps. The ANN only works as a classifier, which takes the extracted features as input, and outputs the recognised class (Pham & Liu, 1995; Pham & Alcock, 1996; Packianather, 1997; Pham & Alcock, 1999).

After the publication of AlexNet (Krizhevsky et al., 2012), the power of Convolutional Neural Networks (CNN) had then been found by researchers. A booming in researches in the CNN can be seen from AlexNet to GoogleNet (Szegedy et al., 2015), VGGNet (Simonyan & Zisserman, 2014), ResNet (He et al., 2016), DenseNet (G. Huang et al., 2017) and so on. Different from the traditional feature-based recognition methods, CNN can perform feature extraction in the hidden layer automatically. This property makes the CNN a great candidate to build a general, robust and accurate object recognition vision system in industry.

Weimer et al. (2016) proposed an application using CNN in defect detection for industrial manufacturing. The advantage of automatic feature extraction from CNN, and the high average detection accuracy (99.2%) were highlighted by the author. However, the drawback of using CNN can also be seen, requiring a large dataset. Weimer et al. (2016) constructed a dataset containing 1,299,200 examples for 12 different categories. Krueger et al. (2019) used CNN for mechanical and electrical parts recognition. The constructed dataset contains 144,000 images from 4,000 parts from automobile. Using pre-trained CNN structure ResNet-152 (He et al., 2016) and transfer learning method, the final performance of the network can achieve 82.0% average classification accuracy. Börold et al. (2020) pointed out the challenge of using CNN in the industry, which is the requirement of large training dataset. The research group used the free and open source 3D software Blender to generate artificial 2D images captured from 3D models of 24 automotive parts. Specifically, the 3D models of the targets were created based on 3D scanning from the real objects. The CNN were trained using artificial images

and tested with real captured images, which achieved about 80% average accuracy. Other trying of CNN in industry can also be found with work (Yildiz & Wörgötter, 2019, 2020; Foo et al., 2021; X. Li et al., 2021; Brogan et al., 2021; Rehnholm, 2021) which described in Section 2.1.

In summary, the main drawback of CNN for industrial applications can be identified: requiring a huge training dataset. Additionally, the original CNN can only work with 2D images but 3D images.

2.3 Point Cloud Classification

With the development of 3D sensors like RADAR (radio detection and ranging), LiDAR (light detection and ranging), and RGB-D (red, green, blue, and depth channels) camera, the 3D data can be easily obtained from field. The 3D data gives more information for object classification task, but also brings challenges to the current classification algorithms. Typically, the raw data captured from 3D sensors is in form of PC. PC is a 3D data structure contains a set of points represented in three-dimensional Cartesian coordinates (X, Y, Z). Specifically these coordinates in PCs are stored in unordered list and without structure (no fixed grid), and distributed irregularly in the space, which give a big challenge to the recognition algorithm.

2.3.1 Hand-crafted Feature Descriptors

Similar with 2D image recognition, most early approaches in PCs classification depends on hand-crafted features extracted from PCs. Researches were focusing on developing a general and robust feature descriptor for PCs (Guo et al., 2016; Han et al., 2018).

For example, work done by Chua & Jarvis (1997) extracted the information from a principle curvature based on the selected point, naming point signatures. The proposed method generates features which are invariant from the rotation and translation, though consumes huge computing resources. Similar approach can be found like the surface signatures descriptor proposed by Yamany & Farag (2002). Another popular used descriptor is the *spin-image* proposed by A. E. Johnson & Hebert (1998). In order to generate the spin-image, firstly a base point is selected. Coordinate and surface normal of the base point will then be used to generate a new cylindrical coordinate system. Lastly, all the remaining points in the PCs will be transfer into the generated cylindrical coordinate system. However, in their further work, A. E. Johnson & Hebert (1999) pointed out that spin-image is lack of robustness when encountering the noise. To address the problem of noisy and cluttered scenes in 3D recognition, Frome et al. (2004) proposed the 3D

shape contexts descriptor. The 3D shape context descriptor is derived from the 2D shape context descriptor proposed by [Belongie et al. \(2002\)](#). The main idea of 3D shape context is to use the a sphere shape support region to extract features from points in different bins. [Matei et al. \(2006\)](#) pointed out the drawback of 3D shape context descriptor when the input image has unknown rotation. Thus, they adopted spin-image descriptor for their 3D recognition system. [R. B. Rusu et al. \(2008\)](#) proposed a method called point feature histograms (PFH). The features of each point is calculated based on relation between its k neighbour-points. By introducing varying scalar to the neighbourhood size, a set of local features will be generated. The four geometric features for each point were split into two parts, and a histogram was then built. Based on the local feature histograms, a subset of important points will be computed and selected as representation for the PC. Using the selected subset features will reduce computing time, and on the other hand increase the significance of the extracted features. In their further research, [R. B. Rusu et al. \(2009\)](#) proposed fast point feature histograms (FPFH) method to increase computing speed.

2.3.2 DNN-based Point Cloud Classification

In recent years, the DNN has shown its outstanding performance in 2D image classification, thus, some researchers tried to applied DNN over 3D PC images. At the first place, researchers were facing the problem of PC's unstructured characteristic, while CNN can only be applied on structured data. To solve this challenge, [Wu et al. \(2015\)](#) proposed and tried to re-organise PCs into volumetric representation. Similar with pixels stored in a 2D image, volumetric image contains a set of 3D voxels stored in a regular 3D grid. With similarity between 2D images and 3D volumetric images, convolutional layer can be applied over the target 3D images. Their proposed 3D ShapeNet is a deep convolutional brief network, which achieved 77.32% accuracy over ModelNet40 benchmark ([Wu et al., 2015](#)). Some further researches are sharing the same idea, like the VoxNet proposed by [Maturana & Scherer \(2015\)](#). Followed by two 3D-convolutional layers, a max pooling layer, and a full connected layer, the VoxNet can achieve 83% accuracy on ModelNet40 benchmark.

Differently, [Su et al. \(2015\)](#) tried to solve the problem of unstructured characteristics of PCs in a brutal way. In their work, multiple simulated 2D cameras will be applied in the simulation to capture 2D images from the target 3D images. Later, their proposed multi-view CNN (MVCNN) architecture will be applied on 2D images using the state-of-art CNN method. Although MVCNN achieved a promising classification accuracy, it interacted with 2D images but 3D PCs. Similar work can also be seen in ([Shi et al., 2015](#)).

A full comparison between MVCNN and volumetric CNN was illustrated by

(Qi et al., 2016). They pointed out the over-fitting in 3D ShapeNet model set, and proposed a multi-task volumetric CNN architecture, which can learn to predict by partial image. They also proposed a multi-orientation volumetric CNN (MO-VCNN) architecture, which combined the idea of MVCNN and volumetric CNN. Both the proposed architecture gave outstanding performance compared with the original ones.

Y. Li et al. (2016) pointed out the drawback of volumetric CNN: an cubically increasing computational complexity with the growing of voxel resolution. To address the problem, Y. Li et al. (2016) proposed the field probing filter to reduce the unwanted computation. The field probing filter was implemented by the field probing layer in the network, which contains three different layers: Sensor Layer, DotProduct layer, and Gaussian Layer respectively. The field probing layer can detect the surface of the volumetric object and transform the image into an intermediate representation. Although the proposed FPNN (field probing based neural network) architecture solved the problem of booming computational complexity for volumetric CNN with increasing voxel resolution, it still restricted with volumetric representation but PC.

Klokov & Lempitsky (2017) proposed to use space-partitioning data structure to address this challenge. They proposed the Kd-Networks architecture based on the 3D indexing structure kd-tree. Using kd-tree data structure, the Kd-Networks can apply convolutional kernel over PCs without building an uniform 3D grids. This will increase the scalability of the network and avoid the booming computational complexity when dealing with large number of data. Similar approaches implemented with octree data structure can be found in OctNet proposed by Riegler et al. (2017), and O-CNN proposed by P.-S. Wang et al. (2017). However, the drawback of using convolutional network as network's skeleton is obvious, which is the non-invariance with data rotation.

Moreover, some approaches were proposed to apply CNN in PCs based on graph representation. Simonovsky & Komodakis (2017) proposed the ECC (edge-conditioned convolution) architecture based on graph convolutions. In ECC, the graph was constructed by downsampling the PCs using VoxelGrid algorithm R. Rusu & Cousins (2017). Then the proposed ECC kernel will be applied on the nearest neighbour points of each graph. Different from using static graph, Y. Wang et al. (2019) proposed the Dynamic Graph CNN (DGCNN) architecture based on dynamic graph. In DGCNN, a special designed EdgeConv kernel was proposed, which can take the coordinates from its neighbour points in the feature space and compute the output feature. Specifically, in every epoch, nearest neighbours method is applied in feature space to reconstruct the graph and compute the new neighbours, thus naming Dynamic Graph CNN. With this special characteristics, the DGCNN is invariant to permutation.

Then it comes to the remarkable work of Qi, Su, et al. (2017), where a pi-

oneer **DNN** architecture namely PointNet which directly interacts with **PCs** was proposed. Different from works like MVCNN which takes 2D images as input, or volumetric **CNN** should transform **PC** into volumetric representation, or works like KD-Networks which changes the data structure of **PCs**, PointNet takes the raw 3D **PCs** as input without any heavy-duty preprocessing. Specifically, two shared-weights deep multilayer perceptrons (**MLPs**) in PointNet were designed to perform the feature extraction function, and extract global features from coordinates of all points. A max-pooling layer was utilised to summarise the global features from the feature space. The last **MLP** in the network takes the extracted global features and perform the object classification. More details of PointNet will be illustrated in Section 3.1.1. Due to its special characteristics, PointNet can also be used for **PCs** segmentation, the point wise classification. Another great feature of PointNet is the rotation and translation invariant, which is quit important for **PC** classification in real scenario. All the special properties of PointNet makes it require less data preprocessing from raw 3D images, and also robust with unknown environment.

The propose of PointNet is an pioneer in extracting point-wise features from **PCs** using deep neural networks. However, the drawback of PointNet is obvious: being lack of local features, thus, in their further research, (Qi, Yi, et al., 2017) proposed an upgrade architecture PointNet++. Instead of simply extracting global features from **PCs**, PointNet++ used hierarchical structure to extract both global and local features from multiple neighbour size.

Inspired by the breakthrough achievement of PointNet, many researchers in the following years were focusing on the similar approaches: trying to propose a kernel which can extract both global and local features from points and their neighbours. For example, inspired by PointNet, J. Li, Chen, & Lee (2018) proposed the SO-Net architecture. SO-Net aims to extract hierarchical features from raw points in **PC** based on SOM (self-organizing map). SO-Net firstly uses SOM to learn the spatial distribution of the points in **PC**. Following by k-nearest neighbour search over the nodes of SOM, the SO-Net then gather batches of points for local feature extraction. Then an architecture similar with multiple deep **MLPs** in PointNet will apply to the found local batches, extract the global features, and classify the target **PC**. Y. Li et al. (2018) proposed the PointCNN architecture which based on a layer of **MLP** to transform **PCs** into a grid feature space, namely χ -transformation. The traditional **CNN** will then be applied over feature space for further classification. Xu et al. (2018) proposed the SpiderCNN architecture which applied a specially designed SpiderConv kernel over each points to extract features. In SpiderConv, neighbour points of target point were determined by nearest neighbour search. Different from PointNet which applies **MLPs** over the (X, Y, Z) coordinates, SpiderConv kernel used a designed filter based on Taylor expansions to extract more information from each point. After that, traditional

CNN will be applied over feature space for further classification.

In recent years, in order to boost the classification accuracy of the network, instead of simply extracting the local features from points, researchers try to extract the geometrical relation information between target point and its neighbours. One example trying is the RS-CNN architecture proposed by [Liu et al. \(2019\)](#). Highlighted by [Liu et al. \(2019\)](#), RS-CNN is focusing on extract relation information from the geometric topology constraints in the local batch. Specifically, a **CNN** layer was designed to learn the predefined low-level relation of each point. In their experiments, different types of low-level relations were tested, including 3D Euclidean distance and normal vectors. Similarly, [H. Zhao et al. \(2019\)](#) proposed the PointWeb architecture with a special designed module namely the Adaptive Feature Adjustment, which can extract relation information between paired points. In PointWeb, all the local neighbour points will be linked and a fully-linked web will be constructed. Utilising Adaptive Feature Adjustment module, the extracted information from **PCs** will also include the feature-space difference between the paired points. And a PointNet-like feature processing and classification **MLPs** will be applied to do the further work. [Qiu et al. \(2021b\)](#) tried to enrich the pre-defined low-level geometric information that fed into CNN, and proposed the GBNet architecture. Instead of using normal vector and Euclidean distance in [Liu et al. \(2019\)](#), [Qiu et al. \(2021b\)](#) used the information from triangular faces constructed by the target point and two nearest neighbour points. The constructed geometric descriptor is a 14-dimensional vector made by 6 geometric vectors from triangular face. [Qiu et al. \(2021a\)](#) pointed out the drawback of local searching techniques like Ball Query and K-nearest-neighbour algorithm: sensitive to the density of **PC** and the pre-defined parameters, thus, proposed a novel grouping algorithm, namely Adaptive Dilated Point Grouping. With Adaptive Dilated Point Grouping, the size of neighbourhood will be learnt and determined accordingly and automatically.

2.4 Evolving Artificial Neural Network

In this section, the literature of the previous work over **EA** based **ANN** training and designing will be reviewed. The **EA** based automatic **DNN** architecture designing will also be reviewed. Last but not the least, coevolutionary algorithm based **ANN** training will also be reviewed.

2.4.1 Evolving Network Weights

The popular used back-propagation training algorithm can boost the training speed for **ANN**, but also can easily lead to a local optimum network. While the train-

ing procedure of the ANN can be tackled as an optimisation problem of finding the best combination of weights, some researchers proposed to apply optimisation algorithm like EA into ANN training procedure to avoid the local optimum results.

Genetic Algorithm (GA) is a typically used evolutionary approach to evolve the weights for ANN (Goldberg & Holland, 1988; Montana & Davis, 1989; Whitley, 1989). Traditional GA can be summarised with the following steps: population initialisation, parent selection, off-spring generation (cross-over and mutation), and new population determination. Generally, the weights of the network will be encoded with either a binary representation or a real-number representation (Yao, 1999). The main challenge in evolving ANN weights is usually around choosing the appropriate representation schemes and operators. For example, if the weights are encoded with real-number, the cross-over step will bring a huge variation into the network, consequently cross-over will be abandoned in this case (Montana & Davis, 1989). To address the problem, Fogel et al. (1990) proposed to apply evolution programming (EP) algorithm. EP is a simplified GA, which has only mutation and selection operators. With such architecture, EP is suitable for optimising continuous problems. One drawback of using EA in evolving the weights of ANN is the high computing time. Some researchers applied hybrid training scheme, where back-propagation training was utilised as an operator within the evolution process to accelerate the convergence speed (Montana & Davis, 1989; Belew et al., 1991; Lee, 1996).

2.4.2 Evolving Network Architecture

Apart from the previous implementation, EAs can also be applied to evolve neural architecture (Yao, 1999; Floreano et al., 2008; Azzini & Tettamanzi, 2011).

Even until now, there is no systematic methods can explain or instruct the design of a well-performed neural network for a given problem. Thus, many researchers tackled the design of network architecture as an optimisation problem, and tried to use optimisation methods to solve it automatically (Azzini & Tettamanzi, 2011). The evolution of neural architectures usually defined as evolving the topological structure of the network. However, in some case, the definition can be extended to find the optimal transfer function between nodes, or learning hyper-parameters like learning rate, dropout rate, and so on. Typically there are two kinds of encoding scheme, that is, direct encoding scheme and indirect encoding scheme (Yao, 1999). With direct encoding scheme, all the information of the network will be encoded into the genome, including every nodes, connections, weights, transfer functions and so on. On the other hand, the indirect encoding scheme will only encode the most important parameters representing the architecture, like number of layers, and number of nodes. For example, proposed by Yao

& Liu (1997), the EPNet addressed the network architecture optimisation problem as a continuous optimisation problem with respect to topological structure, and used evolutionary programming scheme to solve the problem. In this work, the mutation operator was designed to search the optimal topological structure of network. Simultaneously, back-propagation and simulated annealing algorithm were introduced to accelerate the searching for connection weights. Specifically pointed out by Yao & Liu (1997), the proposed EPNet evolved the network architecture and weights at the same time which enhanced the performance of the network training. Similar conclusions about the benefit of simultaneous evolution can also be found in (Andersen et al., 2002; Castellani, 2006).

Evolving DNN Architecture

In recent years, DNNs have shown outstanding performance compared with traditional shallow neural networks (SNNs). Similar with researches in SNN, researches in the area of DNN also have the same challenge in network design. Different from situation in SNN, the training of DNN consumes more computing resources, and the architecture of a DNN is much more complex, which brings more challenges into DNN architecture design. Thus, a booming interest was arisen to automatically design the architecture of DNN using optimisation methods (X. Zhou et al., 2021). Some researchers proposed to extend the EA approach into DNN for automatic topological structure searching (Elsken et al., 2019; X. Zhou et al., 2021). In DNN, there are two main factors to determine the network architecture, which are the kernel unit and topological structure. The kernel unit will determine the type of kernel operation applied over the input data flow, and also determine the shape of output features by repeated times of kernels applied. On the other hand, the topological structure will determine the flow of the information in the network. For example, Real et al. (2017) encoded the CNN with graph representation. In the graph, each vertex represents a convolutional layer, while edges represent connections and activation functions between layers. Specifically, each vertex are encoded with hyper-parameters of a convolution kernel like dimensions of tensor and the channel of the kernel. With such presentation, the proposed method used the tournament selection and a mutation operator to evolve the population. In the experiments, the proposed EA based automatic CNNs architecture designing method achieved a state-of-art performance without any human efforts, but in a huge computing cost. Pointed by their further research (Real et al., 2019), the tournament selection will greatly waste the computing resource while less complex individuals will be idle and waiting. To address the problem, Real et al. (2019) introduced an age parameter into individual, which reduce the number of new individuals generated from old people.

2.4.3 Evolving Input Features and Sub-dataset

Evolving the subsets of training data or fed features is also one of the application of EA in ANN training, namely feature selection. Generally there are three types of feature selection approaches in the literature of machine learning: wrapper approach, filter approach, and embedded approach (Blum & Langley, 1997). The feature selection procedure in ANN training aims to accelerate the training speed and increase the network performance simultaneously, by inducting the irrelevant features or examples. The filter-based feature selection is the most common strategy, while it only require one step of preprocessing. Thus, the filter approach can easily benefit from fast computation speed. However, this manually determined algorithm can easily fail due to complexity of the dataset. Wrapper feature selection approach is the common used EA approach (Castellani, 2006). Specifically, all the candidate features will be encoded into the EA. Example can be found like work done by Zhang et al. (2005). In their work, 14 features were encoded in the chromosome of the population with a 14 dimensional binary string. GA was used as its EA scheme, with 1-point crossover and binary mutation operators. Pointed out by Castellani (2006), the wrapper approach has one drawback, that is, consuming huge computing efforts with lengthy training procedures. To address this problem, Castellani (2006) proposed a specially designed embedded approach. In his work, each individual contains two chromosome, representing both feature space and network weights. Similar with Zhang et al. (2005)'s work, the feature space chromosome was encoded with binary string and evolved with two-point crossover and binary mutation operators (Castellani, 2006). With the embedded approach, the network weights and selected features will be evolved at the same time within each individual with a better performance.

2.4.4 Co-evolving ANN Training

The coevolutionary approach was firstly proposed by Hillis (1990). He pointed out two types of limitation faced by evolutionary searching algorithm: local minimal trap and inefficient testing procedure. To overcome the limitation, some researchers tried to vary the testing cases during evolution, thus Hillis (1990) proposed the co-evolution idea to automatically perform data selection. In his research, the benefit of utilising co-evolution was derived from the biological aspect, where the core idea behind is the predator-prey interaction model. The predator-prey interaction model can increase the efficiency of searching procedure by eliminating easy individuals and focusing on the difficult individuals. On the other hand, by varying the interacting individuals, two species can easily escape from local minimal and achieve a better performance.

[Paredis \(1995\)](#) extended the co-evolution searching idea of [Hillis \(1990\)](#) and proposed Coevolutionary Genetic Algorithm ([CGA](#)) for [ANN](#) training. Similarly, the skeleton of [CGA](#) is the predator-prey model. The major difference between [GA](#) and [CGA](#) is the number of population. In [CGA](#), there will be two types of population, representing potential solutions and testing subsets. These two species represent the predator and prey in the nature. General [GA](#) method will be applied on each population individually. A specially designed interaction model will connect two species by selecting and encountering individuals from each side and assigning interaction fitness to each individual. The individuals in prey (population of testing subsets) which are difficult to be captured by predator (population of [ANN](#) solution) have higher chance to survive in the following generation. However, in [CGA](#), the population of testing dataset was not evolved during the procedure, but only selected based on fitness. In [CGA](#), [GENITOR \(Whitley, 1989\)](#) was used as the general [GA](#) method, including fitness ranking selection, adaptive mutation, two-point cross-over, and steady-state replacement. To address a continuously evolving fitness, the life-time fitness evaluation (LTFE) ([Paredis, 1994](#)) method was used to evaluate the fitness for predators. Specifically, in LTFE method, the fitness of each solution is calculated based on the results of last 20 interactions: the classification accuracy in the latest 20 tasks. Results of new classification tasks in the coming epoch will be inserted into the head of history list, while the oldest classification history will be eliminated.

Technically speaking, [CGA](#) is not a complete co-evolution process, only extending the idea of predator-prey model into [ANN](#) against test dataset. Similar approaches can be found like the coevolutionary between candidate neural networks to play checkers game ([Chellapilla & Fogel, 1999](#)). In this research, the predator-prey model was extended among neural networks. Each neural network represents an strategy agent who can play checkers in the population. Specifically, neural network is an intelligent agent who can evaluate all the possible steps found by minimax search strategy, thus, to guide the game. Different from [CGA](#), the predator-prey interaction is among two candidate agents. The won neural network agent will gain higher fitness compared with the loss one. The higher fitness the network agent gets in one training epoch, the higher the chance it will survive into next generation. On the other hand, network training is based on mutation operation over the weights. However, pointed out by [Watson & Pollack \(2001\)](#) there are few potential drawbacks when utilising coevolutionary: losing gradient, losing general ability, and red queen dynamics.

Based on [CGA](#), [Castellani \(2018\)](#) extended and full-filled more experiments over coevolutionary based [ANN](#) training. Highlighted and cleared by [Castellani \(2018\)](#), the purpose of coevolutionary based [ANN](#) training is to have [ANN](#) trained with most difficult sub-dataset, and as a consequence achieving higher training efficiency and accuracy. As mentioned above, in [CGA](#), the population of prey was

not evolved, thus, [Castellani \(2018\)](#) proposed CoEvolutionary ANN Training algorithm (CENNT), which evolved predator and prey module simultaneously with GA. [Castellani \(2018\)](#) then did a comprehensive comparison between standard back-propagation based training, EA based training, CGA based training, and CENNT based training on MLP. Experimental results show that coevolutionary based ANN training is benefit from faster training speed and higher classification accuracy especially with noise-corrupted and unbalanced dataset.

Chapter 3

Automatic Identification of Mechanical Parts for Robotic Disassembly Using the PointNet Deep Neural Network

The very first manipulation task in automatic disassembly and remanufacturing is the identification of the target mechanical parts. The challenges in this process amount to the uncertain conditions of the objects (e.g. dirty surfaces or missing parts), incomplete vision capturing due to the pose of objects, and error from sensors. Additionally, a typical problem in remanufacturing is that not only mechanical parts need to be identified from different types, but also from different models of the same type.

This chapter presents an identification system based on the deep neural network PointNet, and a developed depth camera simulator, which are described in Section 3.1. A case study on the identification of twelve parts from two turbochargers from automotive engines has been carried out, to examine the feasibility and performance of the proposed identification system. The generation of the experimental model sets is described in Section 3.2, whilst the design and results of the experiments is discussed in Section 3.3 and Section 3.4.

3.1 The Identification System

The proposed system was designed with the goal of carrying out accurate and reliable identification of mechanical parts for manipulation in disassembly processes. The objective is to train the identification system using 3D CAD models of the mechanical parts, and use it to identify real objects in real scenes. Fundamental

requisites of the system will be tolerance to error from the sensors, the ability to recognise the objects in previously unseen poses (rotations and positions), and the ability to recognise similar parts of different brands.

The identification system will constitute the very first module of the automatic disassembly system. It can be used to recognise entire objects to disassemble, the components of an object, and separate parts once the object has been taken apart. Once the desired object has been recognised, its pose can be understood (Besl & McKay, 1992) and robotic manipulation can be carried out. Additionally, once the parts of a product have been located, an optimal disassembly sequence can be sought and applied.

As the core of the proposed identification system, a DNN architecture was chosen. The benefits of using DNNs instead of traditional feature based recognition algorithms were discussed in the review on DNN-based object recognition by Z.-Q. Zhao et al. (2019). After careful consideration, the PointNet (Qi, Su, et al., 2017) DNN architecture was chosen as the recognition algorithm. The main benefit of PointNet is that it was explicitly designed for handling PC models. That is, it can be fed directly with the PCs captured from the scenes, without time-consuming data pre-processing. Additionally, PointNet is able to perform object recognition regardless of the rotation and position of the object, and is tolerant to error in the input data. Finally, once trained PointNet can also be used for scene segmentation, that is to locate different objects in one scene. This feature would be valuable to recognise the individual components of the object to be disassembled.

The main drawback of using DNN architectures (and applying machine learning in general) is the need of acquiring of a rich and variegated set of training data. The data preparation process is time- and labour-consuming, whilst the number of identification targets (object classes) can be very large. For example, it took half a year to Krueger et al. (2019) to create a model set of 4,000 mechanical parts. For each part, four sets of nine different perspectives were captured, for a total of 144,000 2D images.

To address the problem, a model set generation scheme based on 3D CAD models of the mechanical parts is proposed. These models can be easily acquired from the manufacturer, or from reverse engineering of the objects. A depth-camera simulator was developed and deployed to generate the training data. The simulator mimics the data capturing procedure in real-life, and thus provides a realistic set of training examples to the DNN. Using the 3D CAD models of the mechanical parts, the simulator software can be used to generate different kinds of models, varying the number of cameras, the sensor error level, and changing the rotation and position of the objects. Using the proposed data generation scheme, the training model set can be automatically generated for any number of mechanical parts. The details of the developed camera simulator are shown in Section 3.1.2.

In summary, the proposed identification system trains the **DNN PointNet** based on training data generated from 3D **CAD** models of objects, utilising the developed depth camera simulator.

3.1.1 The PointNet Deep Neural Network

PointNet was proposed by [Qi, Su, et al. \(2017\)](#) for the purpose of object classification and part segmentation for **PC** models. It is a deep neural network constructed by multiple neural layers as shown in Figure 3.1. It can be summarized as three key modules.

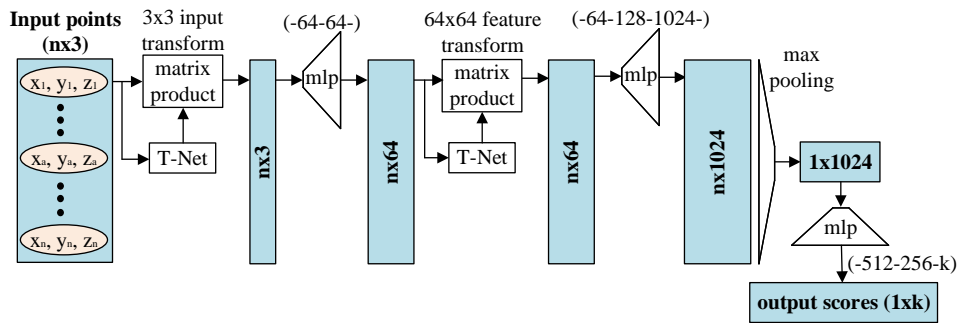


Figure 3.1: Main structure of PointNet

The first module is designed to map the input space to a higher-dimensional representation (embedding space), and makes the procedure invariant to rigid transformations of the object’s pose. Differently from the Spatial Transformer proposed by [Jaderberg et al. \(2015\)](#), a mini-network (T-Net) is used in PointNet ([Qi, Su, et al., 2017](#)). The T-Net takes all the points from the **PC** as input, and predicts the affine transformation matrix that aligns the object to a canonical space before feature extraction. Additionally, another T-Net (“feature transform” in Figure 3.1) is used to further align the embedding space.

The second module is the feature extraction module composed of a set of multi-layer perceptrons (**MLPs**) and a max pooling function ([Qi, Su, et al., 2017](#)). The **MLPs** are used as feature detectors that are applied to the higher-dimensional embedding space, whilst the max pooling layer is used to aggregate the feature detection result. The overall action of the first two modules is to transform the input information into a feature set. That is, it implements a symmetric function that maps the spatial information in the **PC** to the feature space, irrespective of the object pose.

Parameter	Value
β_1	0.9
β_2	0.999
$\hat{\epsilon}$	1×10^{-7}

Table 3.1: Parameters of *Adam* optimisation algorithm in PointNet

The third module of PointNet is a fully connected layer that takes the feature information and generates the identification result.

In summary, when a **PC** consisting of (n) points is fed to PointNet, the coordinates of all its ($n \times 3$) points are mapped into the feature space through the first and second modules of the network. The third module of the network is a standard classifier that takes the features extracted in the previous layers, and outputs the classification score for the input scene.

The customary stochastic gradient descent based *Adam* optimiser (Kingma & Ba, 2014) was used to train PointNet. *Adam* updates the network weights based on gradients calculated from randomly picked mini-batches of **PCs** of predefined size. The *batch-size* is an important hyper-parameter of the algorithm. The error gradients are calculated in parallel, and the calculations accelerated by graphic-processing-units (at the expense of precision). Thus, the *Adam* optimisation algorithm implements a trade-off between training speed and accuracy. The core idea behind *Adam* is to compute an adaptive learning rate for each weight, by estimating the first and second moments of the error gradient, in order to achieve faster training speed and improved accuracy (Kingma & Ba, 2014):

$$w_t = w_{t-1} - \alpha_t \cdot m_t / (\sqrt{v_t} + \hat{\epsilon}) \quad (3.1.1)$$

$$\alpha_t = \alpha \cdot \sqrt{1 - \beta_2^t} / (1 - \beta_1^t) \quad (3.1.2)$$

where t is the batch; w_t is the weight value at time t ; α_t is the adaptive learning rate. The parameter α_t is based on an initial learning rate α , and two parameters β_1 and β_2 . The parameters m_t and v_t are respectively the estimated first and second order moments; and $\hat{\epsilon}$ is a correction constant for numerical stability in case the denominator becomes zero. A recommended configuration for the three parameters m_t , v_t , and $\hat{\epsilon}$ is given by Kingma & Ba (2014) and adopted by Qi, Su, et al. (2017). It is reported in Table 3.1.

PointNet uses an exponential learning rate decay scheme with minimum clipping (Qi, Su, et al., 2017):

$$\alpha = \max \{ \alpha_{init} \cdot r_\alpha^{n/s_\alpha}, \alpha_{min} \} \quad (3.1.3)$$

where, \max is a maximum function; n is the learning step, corresponding to the total number of fed **PCs**; α is the calculated decayed learning rate; α_{init} is the

initial learning rate; α_{min} is the predefined minimum clipping learning rate; r_α is the decay rate parameter; and s_α is the decay step parameter.

Using the decay scheme, the learning rate in PointNet slowly decreases during the training, from α_{init} to a lower bound of α_{min} . By applying the learning rate decay scheme to the *Adam* optimiser, as shown in Equation (3.1.2) and Equation (3.1.3), the adaptive learning rate in *Adam* is constrained within a lower and upper bound, and can not become too large with the number of training steps.

Batch Normalisation (Ioffe & Szegedy, 2015) is also part of the standard *Adam* training procedure. Its purpose is to eliminate the internal covariate shift and speed up the training process. The *Adam* stochastic optimisation algorithm feeds mini-batches of inputs to the ANN, and updates the network weights at the end of the presentation of each batch. The main idea behind Batch Normalisation is to normalise the activation inputs in each layer based on the fed mini-batch inputs. For example, given a mini-batch input \mathcal{B} in a layer with m inputs $\{x_1, x_2, \dots, x_m\}$, the Batch Normalisation procedure transforms, scales, and shifts each activation into:

$$y_i = \text{BN}_{\gamma, \beta}(x_i) \quad (3.1.4)$$

where, γ, β are two trainable parameters for scaling and shifting; y_i is the activation after Batch Normalisation; **BN** is the Batch Normalisation module.

More specifically, the mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ of the batch inputs \mathcal{B} are calculated firstly:

$$\mu_{\mathcal{B}} = \frac{1}{m} \sum_{i=1}^m x_i \quad (3.1.5)$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (3.1.6)$$

Then, the original activation inputs x_i are normalised into \hat{x}_i :

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (3.1.7)$$

where ϵ is a constant for numerical stability.

Finally, the normalised outputs \hat{x}_i are scaled by γ , shifted by β and, generate the final outcome y_i :

$$y_i = \gamma \cdot \hat{x}_i + \beta \quad (3.1.8)$$

However, the mean $\mu_{\mathcal{B}}$ and variance $\sigma_{\mathcal{B}}^2$ cannot be calculated in the test phase when there is usually one set of inputs. Thus, these two variables are designed to

be updated and stored at each step during training, namely 'running mean' μ'_B and 'running variance' σ'^2_B . Specifically, they will be updated based on values calculated in the current batch step t and previous batch step $(t - 1)$ with momentum λ :

$$\mu'_B[t] = \lambda \cdot \mu'_B[t - 1] + (1 - \lambda) \cdot \mu_B[t] \quad (3.1.9)$$

$$\sigma'^2_B[t] = \lambda \cdot \sigma'^2_B[t - 1] + (1 - \lambda) \cdot \sigma^2_B[t] \quad (3.1.10)$$

Additionally, to achieve a more stable training, PointNet adopts a varying scheme based on exponential decay for the momentum λ :

$$\lambda[t] = \min \{1 - \lambda_{init} \cdot r_\lambda^{t \cdot m / s_\lambda}, \lambda_{max}\} \quad (3.1.11)$$

where, λ_{init} and λ_{max} are initial momentum and maximum momentum; r_λ and s_λ are decay rate and decay step; t is the batch step; and m is the batch size.

3.1.2 The Depth Camera Simulator

The purpose-built depth camera simulator was developed based on program developed by Bohg et al. (2014). The idea is to mimic the depth capturing procedure of a Time-of-Flight (ToF) depth sensor. The simulator emits artificial beams of light from the position of the virtual camera in the simulated scene. In real life, each beam is reflected by the object being scanned, and the returning light is captured by an array of photosensors. From the difference in time between the emitted and returned light, and knowing the speed of light, a 3D spatial model of the scene can be created.

In the simulator, the process is simplified by recording the coordinates (x,y,z) of the points where the simulated beams of light first hit the surface of the object. A fixed world Cartesian coordinate system was used to describe the location of the points in the cloud. In the simulations, the object is measured in terms of millimeter units, to reflect the size of the real objects as well as the parameters of the camera. The camera parameters mimic those of a real device, in this case a Microsoft Kinect camera. The simulator allows also building models out of partial scenes from multiple cameras with a user-defined layout. The source code of the simulator is available at the following Github repository: https://github.com/jerryzsj/ToF_camera_simulator.

To simplify the simulation process, the camera is always placed directly facing the origin of the coordinate system. Namely, the position of the camera, the center of the array of photosensors, and the origin of the coordinate system are always collinear. Additionally, the target object will always be placed with its center of gravity (COG) at the origin of the coordinate system with a user-defined orientation.

Due to the limited perspective view of one single camera, in real-life a full object model can only be obtained by placing multiple cameras to capture *partial-views* (PCs) from different angles, and merging these partial views. In practice, often only a small number of cameras can be placed around the object due to limitations in the work space, resulting in incomplete captures. To create the model sets, three cameras were employed in the simulation to mimic a realistic (*partial-view*) image capturing system. To determine the effect of the limited number of cameras on the recognition ability of PointNet, a second model set with *full-view* of the objects was generated, simulating the simultaneous capture from twelve cameras. In the *full-view* setting, complete 3D information of the target is obtained. In both cases, it was assumed that partial views of the mechanical parts could be perfectly merged. That is, that there was no uncertainty in the registration method.

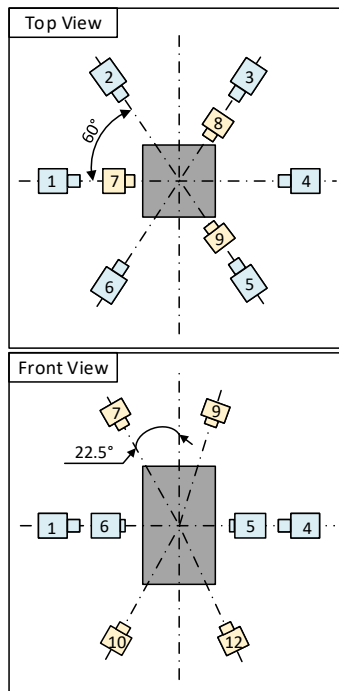


Figure 3.2: Layout of designed scene (Zheng et al., 2022)

The arrangement of the cameras (*partial-view* and *full-view*) is shown in the multiview projection in Figure 3.2, which contains a top view and front view of the simulated vision system. Figure 3.2 shows three cameras in yellow color, *Camera 7, 8, and 9*, placed above the target model, inclined 22.5 degrees with respect to the Z-axis and 120 degrees one from the other. These three cameras simulate a

real industrial scenario where only a *partial-view* of the scene is obtainable. Based on this arrangement, **PCs** of objects will be generated. These **PCs** will contain a number of points varying from 3,000 to 200,000, depending on the size of the objects.

The *full-view* setting contains all twelve cameras shown in Figure 3.2, including *Camera 7, 8, and 9* placed above the target, and another three cameras *Camera 10, 11, and 12* placed below the target, symmetrically to *Camera 7-9*. The last six cameras *Camera 1-6*, in light blue, were placed on the X-Y plane around the target at 60 degrees one from the other.

All cameras are placed at same distance from the target, which lays on the origin of the Cartesian system. This distance was set to 500mm for most of the mechanical parts, and 200mm for small items (socket, hex, and nut), in order to capture a reasonable number of points and mimic a real-life industrial scenario.

3.2 The Mechanical Objects Model Set

Two types of turbocharger (henceforth called *model A* and *model B*) were used to generate the model sets. **CAD** models of these two turbochargers are shown in Figure 3.3. These two turbochargers have a similar structure, with a main body containing from top down the compressor housing, turbine, and turbine housing. *Model A* has an additional wastegate. In total, twelve parts can be disassembled from the two turbochargers. They are shown in Figure 3.4. The goal of the experimental tests is to train the PointNet to correctly identify the 12 parts.

Although some parts share a similar shape, they can all be distinguished by peculiar features in their design, like for example *Blade A* and *Blade B* in Figure 3.4. Nonetheless, due to the similarity of their structure, there is potential for misclassification from the PointNet.

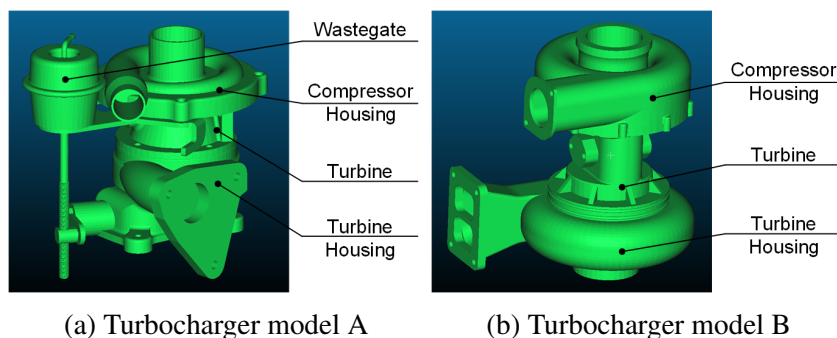


Figure 3.3: Two turbochargers used in experiments (Zheng et al., 2022)

Two model sets were generated from the **CAD** models of the twelve mechan-

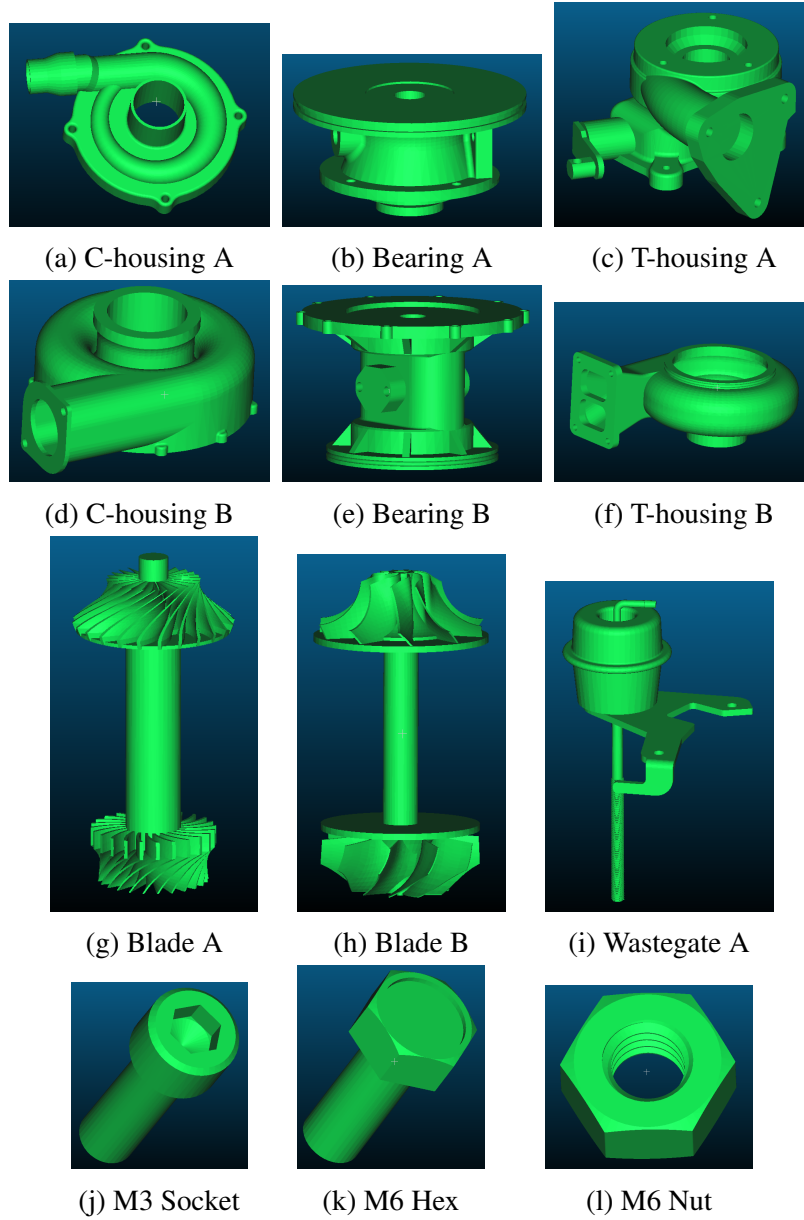


Figure 3.4: Twelve mechanical parts used in experiments (Zheng et al., 2022), where: C-bearing represents compressor bearing; T-housing represents turbine housing

ical parts, one constituted by *partial-views* and one by *full-views* of the objects as mentioned in section 3.1.2. Each part was represented in different poses, and for each pose one *partial-view* and one *full-view* was taken. The two data sets thus differed only from the completeness of the capture. The details of the model sets are described below.

3.2.1 The Clean model set

The designed scenes (*partial-view* and *full-view*) were based on an ideal situation where there is no sensor error. A typical case is when the PointNet is trained using synthetic *PCs* generated from the *CAD* models of the parts. The set of *PCs* captured without any sensor error will be henceforth called *clean data*. For each mechanical part, 300 *PCs* have been generated and saved, each representing a different pose of the object. These scenes were created rotating the *CAD* model randomly along the three coordinate axes (roll, pitch, yaw) with uniform distribution in the $[0, 365]$ range. Of these 300 *PCs*, 200 were used for training the PointNet, and the remaining 100 *PCs* to test the learning results.

In summary, the *clean model set* contains the following two training and test subsets:

- *partial-view* training set: $200 \text{ PCs} \times 12 \text{ objects} = 2400 \text{ PCs}$
- *partial-view* test set: $100 \text{ PCs} \times 12 \text{ objects} = 1200 \text{ PCs}$
- *full-view* training set: $200 \text{ PCs} \times 12 \text{ objects} = 2400 \text{ PCs}$
- *full-view* test set: $100 \text{ PCs} \times 12 \text{ objects} = 1200 \text{ PCs}$

3.2.2 Down-sampling

A very large number of points customarily characterises *PCs* acquired from real world scenes. Although this large number of points provides detail that may promote accurate recognition rates from the neural network, it does constitute a large input vector that may require a large architecture, and increases the *PC* processing time. Additionally, it should be noted that objects of different size will be described by a different number of points, and this variability is usually incompatible with the fixed structure of neural networks. For the above reasons, a random uniform down-sampling of the *PCs* was performed to standardise the size of the input vector fed to the neural network, and to obtain reasonably fast scene processing times.

PointNet uses weight-sharing MLPs as feature detectors for each cloud point, and a max-pooling layer to extract global feature information from the whole *PC*.

PointNet treats **PCs** as unordered sets of points, and regards the number of points that is fed to the input layer as a hyper-parameter set by the user. Originally, this hyper-parameter was optimised to handle the 2048 point **PCs** of the ModelNet40 benchmark set (Wu et al., 2015; Qi, Su, et al., 2017), or 1024 point **PCs**. In this study, following preliminary tests, this hyper-parameter was optimised for ease of implementation and performance of the classifier to 1000 points. The generated **PCs** were down-sampled accordingly. This setting will be consistently used throughout all the experiments reported in this thesis.

3.2.3 Normalisation

Normalisation was applied on the **PCs** before they were fed to the PointNet. The normalisation step re-scaled all **PCs** into a size 1 bounding box centered at the origin and aligned to the axes of symmetry of the objects. This procedure was introduced to crop out the empty space due to *partial-view*, and resize the **PCs** into a limited space. It is important to notice that the normalisation does not rescale the objects to a standard size, since the size after normalisation depends not only on the view but also the orientation of the objects. For example, the size of a cuboid will be largest when all its sides are aligned to the coordinate axes, and smallest when one of its diagonals is aligned to the coordinate axes. Most of the parts used in this study are between 12cm and 18cm long in all the three dimensions, except for the M3 Socket, M6 Hex and M6 Nut which are between 3cm to 5cm.

3.2.4 The Error model set

The normalisation process described in the previous section makes it also possible to introduce a consistent error level into the models of the data set. That is, the error level was set as a fraction of the side of the bounding box (which size is 1). For example, a 10% error level indicates that the coordinate of each point was perturbed of a random amount uniformly drawn within the interval $I = [-0.05, +0.05] \in \mathbb{R}$. Examples of **PCs** with different levels of error are shown in Figure 3.5.

Due to the normalisation of the **PC** models, the level of added error is not constant, and depends on the size and orientation of the objects. For example, the error applied on the three small parts (M3 Socket, M6 Hex and M6 Nut) simulates a level of sensor imprecision that is one order of magnitude smaller than the error on the other parts. This case is not realistic, since the scanning error in **PCs** largely depends on the sensor used and the pre-processing (registration) of the model. Yet, the main purpose of the experiments was to examine the ability of PointNet to deal with different levels of imprecision in the scans. For this purpose,

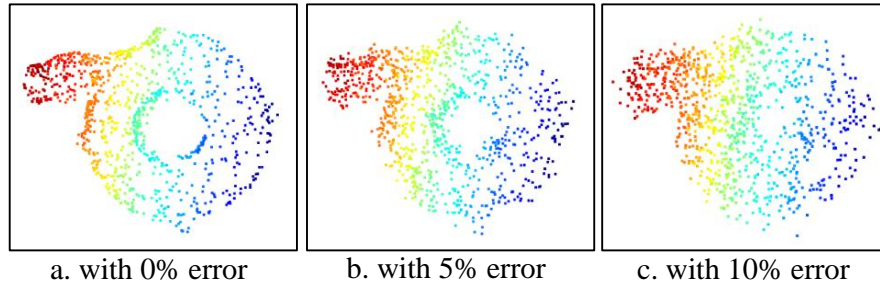


Figure 3.5: Visualisation of PCs of *C-housing-A* with 1000 points and corrupted with different level of error (visualised with Open3D (Q.-Y. Zhou et al., 2018))

the 12 mechanical components used in the tests could be as well thought of as dimensionless objects of complex shape.

3.3 Experimental Design

Three sets of experiments were designed to verify the performance of the proposed identification system. They were carried out using the open-source PointNet code made available by Qi, Su, et al. (2017) on Github ¹, keeping most of the hyper-parameters unchanged.

The full list of hyper-parameters used in the experiments is given in Table 3.2. Based on preliminary tests, it was decided to train the PointNet for 200 learning epochs, using a batch size of 100 samples, instead of the 250 learning epochs and 32 samples used by Qi, Su, et al. (2017). The above choice of parameters was observed to optimise PointNet accuracy in the fastest learning time. The remaining hyper-parameters were set according to the default values employed by Qi, Su, et al. (2017). The complete list of hyper-parameters is given in Table 3.2. For a full description of the hyper-parameters the reader is referred to Section 3.1.1.

3.3.1 Partial View and Sampling

The difference in classification accuracy when dealing with model sets generated by *partial-view* and *full-view* scenes was examined first. The experiments were designed to compare the performance of the PointNet when incomplete information from the object in the real scene was used (*partial-view*), and when full information (*full-view*) was used. Additionally, one further test was carried out to

¹<https://github.com/charlesq34/pointnet>.

Parameter	Value
learning epoch	200
batch size	100
initial learning rate	0.0001
minimum learning rate	0.00001
learning rate decay rate	0.7
learning rate decay step	200,000
initial Batch Normalisation momentum	0.5
maximum Batch Normalisation momentum	0.99
Batch Normalisation momentum decay rate	0.5
Batch Normalisation momentum decay step	200,000

Table 3.2: Hyper-parameters of PointNet used in the experiments

assess the variation in the classification performance when a different number of points was fed to the input layer. The results of this experiment will be used to fix the size of the input vector to the PointNet. In total, the proposed first set of experiments included the following three cases:

- *full-view clean* set with 1000 input points
- *partial-view clean* set with 1000 input points
- *partial-view clean* set with 2000 input points

To deal with the stochastic nature of the PointNet learning procedure, 10 independent learning trials were performed, and the results statistically analysed.

3.3.2 Tolerance to Error

A second set of experiments was carried out to identify the performance of the PointNet when dealing with sensor imprecision, that is, data with error. This experiment simulates the case where the neural network is trained using perfect images generated from the CAD models (*clean* model set), and used for recognition of real images acquired through imprecise sensors (*error* model set). That is, PointNet was trained using the *clean* model set and its performance tested using the *error* model set. In this second set of experiments, different levels of sensor error were simulated. The experiments were carried out using the *partial-view* images, mimicking a real industrial scenario.

This set of experiments tested also the possibility of increasing the robustness to error of PointNet by training it using noisy versions of the training PCs, where artificial error was injected.

In summary, the following cases were included in the second set of experiments:

- training set with zero error, and test set with error from zero to 10% in unitary steps
- training set with 5% error, and test set with error from zero to 10% in unitary steps

Also in this second set of experiments, 10 independent training trials were carried out for each experiment, and the results statistically analysed.

3.3.3 Part Specific Classifiers

The third set of experiments was performed to identify the performance of a set of 12 independent PointNet classifiers, each one trained to recognise one specific mechanical part. This set of classifiers could then be used independently to guide robots or robot operations, focusing on the manipulation of only one component, or they could be combined together to form an ensemble classifier (Rokach, 2010).

In this set of experiments, each PointNet was thus used as a binary classifier, and trained to output a value of '1' for the mechanical part it was tasked to recognise, or a '0' for any of the other 11 parts. The experiments were carried out using the *partial-view clean* model set, and tested on model sets containing different levels of error:

- training set with zero error, and test set with zero error
- training set with zero error, and test set with 5% error
- training set with zero error, and test set with 10% error

It should be noted that in this set of tests the classifiers had to face a data imbalance problem, since the training set is constituted of 200 positive instances, and $11 \times 200 = 2200$ negative instances of the sought mechanical part. As in the previous experiments, the classifiers were tested on the results of 10 independent learning trials.

3.4 Experimental Results

This section presents the results of the sets of experiments described in Section 3.3. The specifications of the hardware employed for the experiments are given in Table 3.3

Item	Detail
CPU	Intel i7-8700k
Motherboard	Asus Prime Z370A
RAM	Corsair CMK16GX4M2A2400C16 16GB
GPU-0	Nvidia GTX-1080-FE
GPU-1	Nvidia RTX-3090-FE

Table 3.3: Details of the computer used for experiments

The results of the 10 learning trials are summarised using the median of the classification accuracy. When useful, box-plots were used to visualise the results.

Each box-plot visualises the five-number summary of the accuracy results obtained in the 10 independent learning trials performed in the experiment. The five-number summary includes the following statistical values: minimum, first quartile, median, third quartile, and maximum. Using the box-plot, the spread of the results can be visualised as the size of the box in the plot, that is, the interquartile range (IQR):

$$IQR = Q3 - Q1 \quad (3.4.1)$$

where, Q3 is the third quartile of the distribution of the results, and Q1 is the first quartile.

To ascertain the statistical significance of the differences in the accuracy results obtained in different sets of learning trials, pairwise two-tailed Mann-Whitney tests were performed at a significance level of 1%. The choice of using non-parametric statistical measures (median or five-number summary, Mann-Whitney tests) was motivated by their robustness to outliers and skewed distributions.

3.4.1 Partial View and Sampling

The classification accuracies obtained in the first set of experiments are summarised in Figure 3.6 using box-plots. The accuracy of the classifier was calculated as the ratio of the correctly classified samples (true positives) to the total number of samples (true and false positives) in the test set (Grandini et al., 2020):

$$accuracy_{multi-class} = \frac{T_1 + T_2 + \dots + T_n}{T_1 + F_1 + T_2 + F_2 + \dots + T_n + F_n} \quad (3.4.2)$$

where n is the total number of classes in the dataset, and T_a and F_a are the true and false positives for class a .

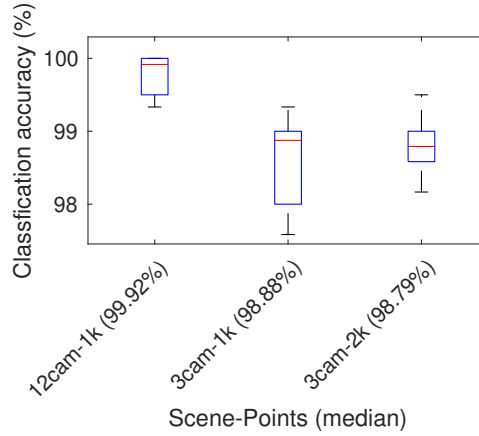


Figure 3.6: Classification accuracy of experiments using PCs from *full-view* (12cam) and *partial-view* (3cam) scenes and with different number of sampled points: 1000 (1k) and 2000 (2k) points

Model Set	Training Time		Accuracy(%)
	(in minutes using GPU)		
	(GTX-1080)	(RTX-3090)	
partial-view-1k	~80	~16	98.88%
partial-view-2k	~160	~32	98.79%

Table 3.4: Training time and accuracy obtained in the experiments where PointNet was trained using the *partial-view-1k* and *partial-view-2k* model sets. The training time was recorded from one sample training procedure with PC described in Table 3.3

Table 3.4 details the accuracy and total execution time of the learning procedure for the two experiments using PCs of different size (number of sampled points). To be noted, the execution time was measured based on codes running under two Graphics Processing Unit (GPU) models: GTX-1080 and RTX-3090 respectively. The learning curves obtained in the experiments where PointNet was trained using the *partial-view-1k* and *partial-view-2k* model sets are shown in Figure 3.7 and Figure 3.8. The curves were generated using the median training accuracy from 10 independent experiments. The plots show that the classification accuracy on the training set approaches 100% and stabilises after approximately 120 leaning cycles circa in both cases.

When the PointNet was fed using 1000 input points, Figure 3.6 shows that there were significant differences in accuracy between the networks trained and

tested using the *full-view* and *partial-view* model sets. The significance of these differences was confirmed by the p-value ($p = 0.0002$) of the pairwise Mann-Whitney test. However, it should be noted that in absolute terms the magnitude of the differences in accuracy is modest (1% to 2%), whilst the classification accuracy is in both cases close to 100%.

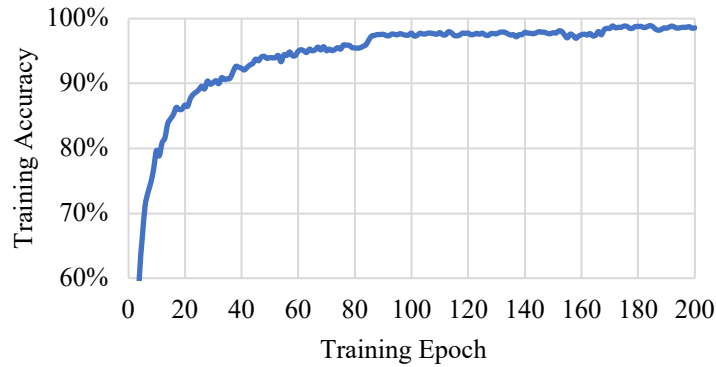


Figure 3.7: Learning curve of networks trained by *partial-view clean* set with 1000 sampling points

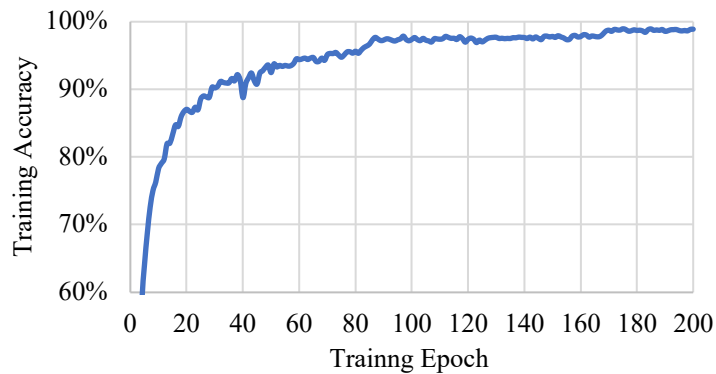


Figure 3.8: Learning curve of networks trained by *partial-view clean* set with 2000 sampling points

The learning results achieved by PointNet using 2000 sampled points are comparable to those obtained sampling 1000 points from the PCs (Figure 3.7 and Figure 3.8). The main difference in the test accuracy seems to be in a smaller spread of the results, which might indicate that an increase in the consistency of the learning results when more data points are used. The p-value ($p = 0.5708$) of the Mann-Whitney test confirmed there are no statistically significant differences

between the results of the learning trials carried out using 1000 and 2000 input points. Another difference between the two experiments is their computational complexity: the network trained with 2000 input points needed about twice the time needed for training the network using 1000 input points (Table 3.4).

Considering the trade-offs between accuracy and computational complexity, 1000 sampled points will be used for the remainder of the experiments in this thesis.

3.4.2 Tolerance to Error

Figure 3.9 shows the accuracy results obtained on test sets characterised by different levels of artificial additive error, when the PointNet was trained using the *clean* model set. The performance of PointNet remains close to 100% as long as the error level is within 4%. A significant decline in accuracy is seen as the additive error level increases, and falls below acceptable standards (less than 80%) as the error level reaches 7%.

Figure 3.10 is similar to Figure 3.9, with the difference that this time the PointNet was trained using PCs corrupted by a 5% level of error. The new figure shows clearly that the added error in the training set positively boosts the ability of the trained PointNet to deal with imprecise readings. Indeed, for nearly all levels of additive error (0% to 8%), a high accuracy (above 90%) was achieved. For the remaining two experiments where the error level is most severe (9% and 10% error level), the classification performance is still acceptable (over 80%).

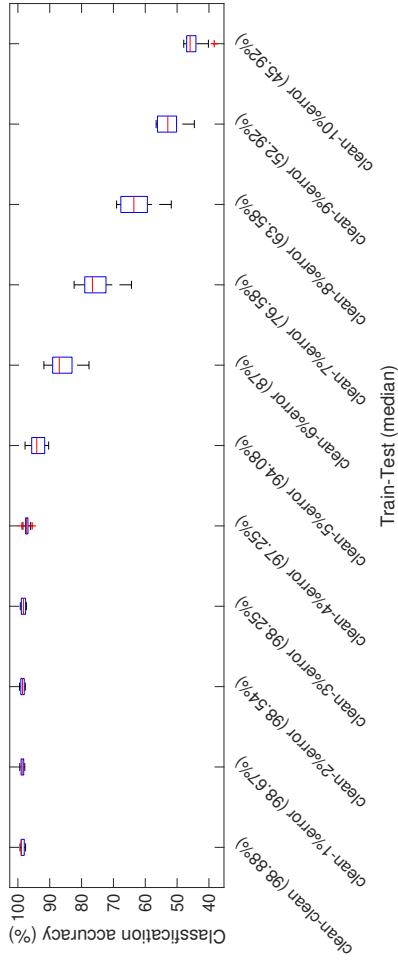


Figure 3.9: Results on test sets of different error level, when the PointNet is trained using clean (zero error) scenes

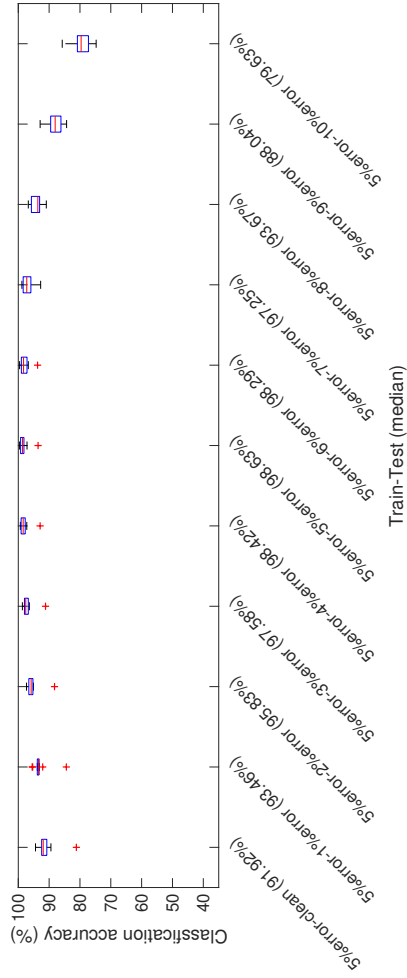


Figure 3.10: Results on test sets of different error level, when the PointNet is trained using scenes with 5% error level

Error level in test set	Training Set		P-value
	clean	5%-error	
0%	98.88	91.92	0.0002
1%	98.67	93.46	0.0002
2%	98.54	95.83	0.0002
3%	98.25	97.58	0.0284
4%	97.25	98.42	0.0343
5%	94.08	98.63	0.0009
6%	87.00	98.29	0.0002
7%	76.58	97.25	0.0002
8%	63.58	93.67	0.0002
9%	52.92	88.04	0.0002
10%	45.92	79.63	0.0002

Table 3.5: Average (median) accuracies (%) obtained training the PointNet using the clean and 5%-error training sets, and tested on model sets with different levels of error. The last column reports the p-values of pairwise Mann-Whitney tests. Where statistical significance was found (p-value smaller than 0.01), superior accuracy results were highlighted in bold

Table 3.5 compares the performance of the networks trained using the *clean* and *5% error* model sets, for each of the 10 the test sets of different error level. As above, Mann-Whitney tests were conducted to reveal statistically significant differences in the results. The results of the tests confirm that if sensor imprecision is expected, the performance of PointNet in the recall phase can be significantly improved by perturbing the training set with error. As Figure 3.10 shows, the improvement is maximum when the level of the error that is injected in the training set is closest to the level of error in the test set (i.e. the level of sensor error).

Finally, Figure 3.11, Figure 3.12, and Figure 3.13 show the evolution of the training and testing accuracy against the training epochs. The plots refer to three different experiments where PointNet was trained using model sets containing different levels of error and tested with the *5%-error* set. The curves were generated based on the median values of the training and testing accuracy from 10 independent learning trials. The three curves show that the ability of PointNet to generalise to unseen examples is maximum when the level of error in the training set corresponds to the level of error in the test set.

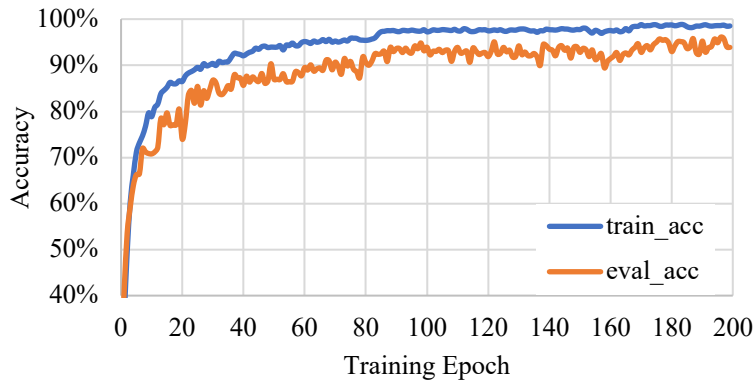


Figure 3.11: Plots of training and testing accuracies of PointNets trained with *clean* set and tested with *5%-error* set

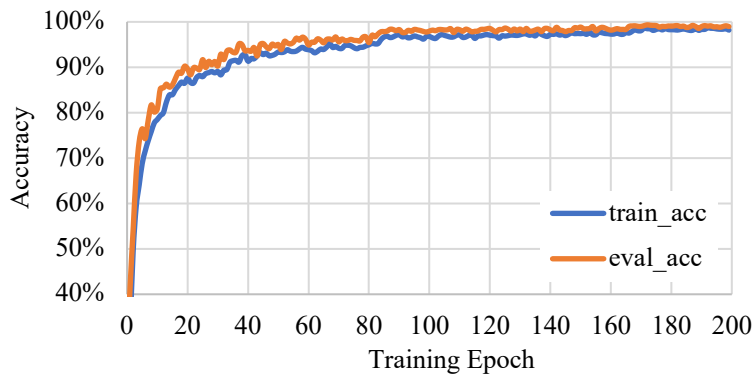


Figure 3.12: Plots of training and testing accuracies of PointNets trained with *5%-error* set and tested with *5%-error* set

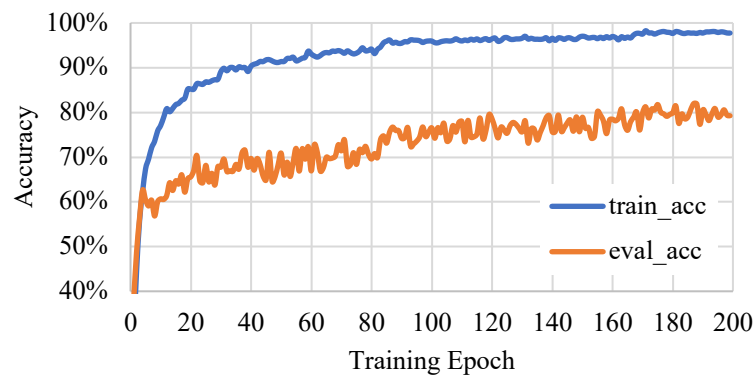


Figure 3.13: Plots of training and testing curves of PointNets trained with *10%-error* set and tested with *5%-error* set

3.4.3 Part Specific Classifiers

The results of the third set of experiments are shown in Tables 3.6 to 3.8. The tables include the classification accuracy, precision and breakdown of the classification results on the test set for each mechanical part. The breakdown of the classification results in Tables 3.6 to 3.8 shows the average counts (statistical median from the 10 independent learning trials) of the true positives (TP), false positives (FP), true negatives (TN), false negatives (FN). Keeping in mind the composition of the test set, a perfect classifier would score a true value $TP = 100$, a true negative $TN = 1100$, and zero false positives and false negatives ($FP = FN = 0$).

In detail, the accuracy of each binary classifier was calculated as the ratio of all true positives and negatives (TP+TN) versus the total classification attempts (TP+FP+TN+FN) (Grandini et al., 2020):

$$Accuracy_{two-class} = \frac{TP + TN}{TP + FP + TN + FN} \quad (3.4.3)$$

The precision represents the ratio of true positives versus all positive predictions (TP+FP):

$$Precision = \frac{TP}{TP + FP} \quad (3.4.4)$$

Name of classifier	Breakdown of results				Precision	Accuracy
	TP	FP	TN	FN	(%)	(%)
Compressor Housing A	99.7	17.3	1082.7	0.3	85.21	98.53
Compressor Housing B	94.4	60.4	1039.6	5.6	60.98	94.50
Bearing A	100.0	10.0	1090.0	0.0	90.91	99.17
Bearing B	99.9	0.5	1099.5	1.0	99.50	99.88
Turbine Housing A	97.2	11.6	1088.4	2.8	89.34	98.80
Turbine Housing B	94.3	61.7	1038.3	5.7	60.45	94.38
Blade A	99.5	7.1	1092.9	0.5	93.34	99.37
Blade B	99.3	4.3	1095.7	0.7	95.85	99.58
Wastegate	99.9	2.4	1097.6	0.1	97.65	99.79
M6 Hex	97.9	13.4	1086.6	2.1	87.96	98.71
M6 Nut	100.0	3.2	1096.8	0.0	96.90	99.73
M3 Socket	99.9	6.7	1093.3	0.1	93.71	99.43
Median	-	-	-	-	92.13	99.27

Table 3.6: Performance of classifiers trained using *clean* set, and tested on data with zero error (*clean* set)

In consideration of the imbalanced distribution of the class instances (see Section 3.3.3), the overall classification accuracy was mainly determined by the performance on the largest class (the negative instances). For example, when tested with 10% error corrupted PCs, part specific classifiers show a good classification accuracy (92.68% in median) in Table 3.8. This good performance was mostly limited to the ability to recognise negative instances (TN) of the desired parts. This limitation can also be identified from a low precision in the results. For example, Compressor Housing B and Turbine Housing B have precision lower than 61% in both three set of tests in Tables 3.6 to 3.8.

3.5 Discussion

This chapter aimed to evaluate the ability of PointNet to identify complex mechanical parts from PCs. The experiments showed that PointNet achieved a very good performance when trained with partial-view images of the desired parts. This ability of identifying targets based on partial-views suggests that PointNet may also be capable of recognising objects with missing parts, a common occurrence in re-manufacturing. Additionally, the PointNet needed only a small randomly sampled fraction of the points in the PCs to achieve successful identification.

The addition of error to the test set of models affected the performance of

Name of classifier	Breakdown of results				Precision (%)	Accuracy (%)
	TP	FP	TN	FN		
Compressor Housing A	92.6	13.6	1086.4	7.4	87.19	98.25
Compressor Housing B	98.2	175.6	924.4	1.8	35.87	85.22
Bearing A	97.9	23.0	1077.0	2.1	80.98	97.91
Bearing B	100.0	2.3	1097.7	0.0	97.75	99.81
Turbine Housing A	96.8	80.9	1019.1	3.2	54.47	92.99
Turbine Housing B	66.9	94.8	1005.2	33.1	41.37	89.34
Blade A	90.9	1.5	1098.5	9.1	98.38	99.12
Blade B	90.2	0.1	1099.9	9.8	99.89	99.18
Wastegate	96.9	0.3	1099.7	3.1	99.69	99.72
M6 Hex	78.8	67.0	1033.0	21.2	54.05	92.65
M6 Nut	87.6	0.6	1099.4	12.4	99.32	98.92
M3 Socket	87.7	0.0	1100.0	12.3	100.00	98.98
Median	-	-	-	-	92.47	98.59

Table 3.7: Performance of classifiers trained using *clean* set, and tested on data with 5% error

Name of classifier	Breakdown of results				Precision (%)	Accuracy (%)
	TP	FP	TN	FN		
Compressor Housing A	24.9	9.1	1090.9	75.1	73.24	92.98
Compressor Housing B	86.7	284.3	815.7	13.3	23.37	75.20
Bearing A	74.3	61.2	1038.8	25.7	54.83	92.76
Bearing B	99.9	10.1	1089.9	0.1	90.82	99.15
Turbine Housing A	90.6	381.1	718.9	9.4	19.21	67.46
Turbine Housing B	8.4	250.3	849.7	91.6	3.25	71.51
Blade A	21.2	0.1	1099.9	78.8	99.53	93.43
Blade B	15.5	0.0	1100.0	84.5	100.00	92.96
Wastegate	60.9	0.0	1100.0	39.1	100.00	96.74
M6 Hex	8.8	96.3	1003.7	91.2	8.37	84.38
M6 Nut	5.3	0.0	1100.0	94.7	100.00	92.11
M3 Socket	11.1	0.0	1100.0	88.9	100.00	92.59
Median	-	-	-	-	82.03	92.68

Table 3.8: Performance of classifiers trained using *clean* set, and tested on data with 10% error

PointNet. The identification accuracy was satisfactory (above 95%) until a 5% or more level of error was applied to the models of the test set. In a factory environment, the error level of the captured images will be determined by camera quality, lighting status, denoising algorithms, etc. For situations with appreciable levels of error (more than 5%), the experiments in this study indicated that the identification performance can be significantly improved by training PointNet with PCs corrupted by artificial error.

The experimental results also find that training one PointNet to identify each mechanical part can achieve a good recognition accuracy. However, a low precision was also observed. Specifically, the classification accuracy was mainly limited to the ability of recognising false negatives. In general, it can be suggested that part-specific classifiers can be a useful solution for situations where one specific mechanical object needs to be picked up from a batch of different objects. Further work should be done to systematically evaluate the possibility of forming an identification system based on an ensemble of classifiers. For example, a multi-machine voting system can be designed to summarise results from all the classifiers for decision making.

Additionally, if one unique classifier is used for all mechanical parts, any addition or removal of parts requires the re-training of the whole system. Using an ensemble of classifiers only one new classifier would have to be trained to identify a new object, or one classifier would have to be removed if identification of an old object is no longer needed, whilst the rest of the system would remain unchanged. Further study can perform to identify if an ensemble of classifiers is easier to add new objects or remove old ones to the recognition task.

3.6 Conclusions

This chapter proposed an identification system to recognise complex mechanical parts for remanufacturing applications. The proposed system was based on the recently introduced deep neural network PointNet.

The preparation of the training data for deep neural network applications is often time- and labour-intensive. This chapter evaluated the viability of using artificially generated PCs from 3D CAD models to train the PointNet, and perform object identification from previously unseen and possibly imprecise 3D scans. A purpose-built depth camera simulator was developed to generate the PointNet training and test model sets. The developed simulator is capable of building scenes with objects in desired orientation and position, and with a pre-defined number and layout of cameras. The model sets included *partial-view* scenes taken by simulating three cameras placed above the target object, which mimicked a real industrial scenario.

Experimental evidence showed that the PointNet was capable of identifying mechanical parts with high accuracy, although the results tended to deteriorate as simulated sensor error was increased. However, on objects of size larger than 100mm (as it was the case for most of the turbocharger parts), and a simulated camera precision better than ± 2 mm, viz. for an error level less than 2%, the identification system would be able to perform with satisfactorily good accuracy. Additionally, if the PointNet is trained using PCs where error is artificially added, the performance can be significantly raised on corrupted scenes (i.e. real-life images affected by sensor error).

This chapter explored also the viability of using one separate classifier to identify each of the twelve mechanical parts. Experimental results suggested that, if implemented in an ensemble of classifiers, the recognition system can show a good classification accuracy but limited precision. Further work is required to systematically evaluate the performance of an ensemble of classifiers.

Chapter 4

A Detailed Evaluation of PointNet Capabilities

Reliable object manipulation procedures are a fundamental prerequisite for the robotic handling of parts in disassembly and remanufacturing. The literature on grasping and manipulation includes methods based on properties of the objects like their geometry (Kopicki et al., 2016) or dynamics (Mavrakis et al., 2016). Regardless of the method used, the shape of the target object must be estimated.

In many industrial applications, it is possible to approximate the shape of mechanical parts with geometric primitives such as spheres, boxes, and cylinders. This chapter reports a thorough investigation on the ability of PointNet to recognise shape primitives in real-life objects. The main difficulty in this task comes from the fact that the shape of everyday objects is often not perfectly regular. Error and occlusion (*partial view*) contribute to the difficulty of the recognition task. In the case of PointNet, the results of Chapter 3 indicated that the performance of PointNet is sensitive to the level of error in the PCs.

Most of the literature on PointNet focuses on the recognition of objects from clean PCs generated from CAD models. For example, Qi, Su, et al. (2017) tested PointNet on PCs generated from ModelNet40, a large benchmark containing 12,311 CAD models from 40 object categories (Wu et al., 2015). In their study, Qi, Su, et al. (2017) only cursorily examined PointNet ability to deal with error corrupted scenes.

This chapter reports a thorough investigation on the performance of the PointNet classifier. In particular, the aim is to assess the feasibility of training PointNet using a set of primitive shapes (cylinders, spheres, etc.), and use the trained network to recognise real-life objects of similar shapes. The study aims also to evaluate whether shape recognition can be performed using simpler classifiers like shallow ANNs. To answer this question, the performance of PointNet was compared to the performance of two popular shallow neural networks: a MLP

(Rumelhart et al., 1985) and a radial basis function network (RBFN) (Broomhead & Lowe, 1988).

The model sets used in the experiments are described in Section 4.1. The two shallow ANNs take as input a vector of numerical descriptors of the scenes. The shallow neural network architectures and the feature extraction scheme are described in Section 4.2. The experimental set up and results are reported in Section 4.3. The outcomes of the tests are discussed in Section 4.4, whilst Section 4.5 concludes the Section.

4.1 Data Sets Used

In this section, the details of the model sets used in the experiments are presented. The goal of the work described in this chapter was to evaluate the ability of PointNet to recognise shape primitives in real objects, after having been trained on samples of artificial primitive shapes. For this purpose, three model sets were used. All the PCs in the model sets were normalised before being fed to the PointNet and feature extraction routine (for the shallow neural networks). At present, there is no available benchmark of real 3D scanning of mechanical objects like those studied in Chapter 3. For this study, a popular benchmark of 3D models of real-life objects was used: the Yale-CMU-Berkeley (YCB) model set (Calli et al., 2015). Twenty-eight models from this set were selected, namely YCB-28, and used to evaluate the performance of the trained PointNet and shallow ANNs. The YCB model set was originally created for robotic manipulation instead of classification. For this reason, for each of the models used in the experiments, a target classifier output was created. This output corresponded to one of three basic primitive shapes: box, cylinder, and sphere. The full details of the YCB scenes used are given in Section 4.1.1. To train the classifiers, two artificial model sets were used. They are presented in Section 4.1.2 and Section 4.1.3.

4.1.1 The Yale-CMU-Berkeley (YCB) Object and Model Set

The Yale-CMU-Berkeley object and model set was created by Calli et al. (2015) for research in robotic manipulation. Calli et al. (2015) used two series of depth cameras (BigBIRD Object Scanning Rig and Google Scanners) to capture PCs from several real-life objects from multiple angles of views. PCs captured from each object were then merged and de-noised to create mesh models. Only one mesh model was created for each object.

Differently from large classification sets like ModelNet40 (Wu et al., 2015), which contains 12,311 items from 40 different categories, the YCB set contains PCs and mesh models from only 77 daily-life objects. These objects were broadly

classified by [Wu et al. \(2015\)](#) in 5 main categories: food items, kitchen items, tool items, shape items, and task items. In this study, the objects were grouped by their shape, and samples of appearance reasonably close to the following three primitive shapes were picked: *box*, *cylinder*, and *sphere*. In total, 28 samples were chosen, namely 9 models of *box*-shaped objects, 8 models of *cylinder*-shaped objects, and 11 models of *sphere*-shaped objects. The names and IDs of the twenty-eight models are shown in Table 4.1, and their pictures and mesh models are shown in Figures 4.1 to 4.3. The test model set for all the experiments in this chapter was generated from these twenty-eight models.

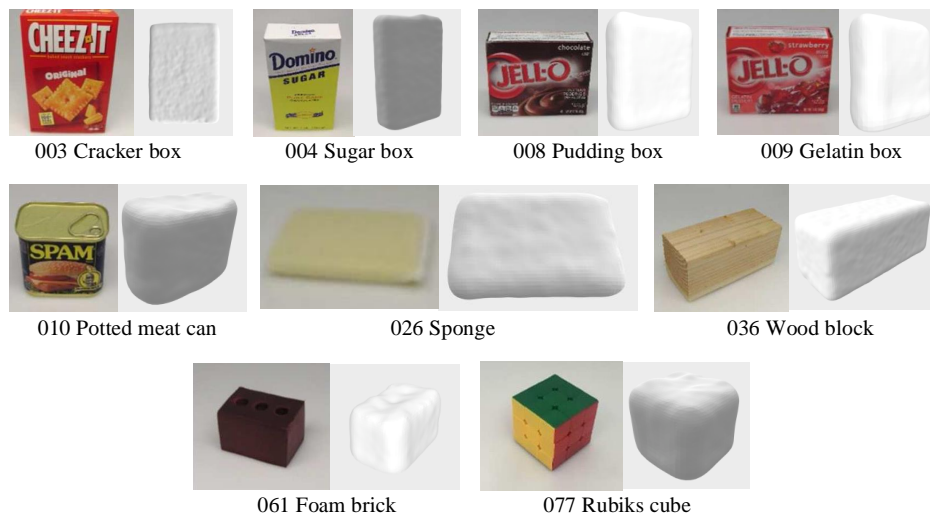


Figure 4.1: The the nine selected box-like objects (images and meshes) from [YCB](#) set

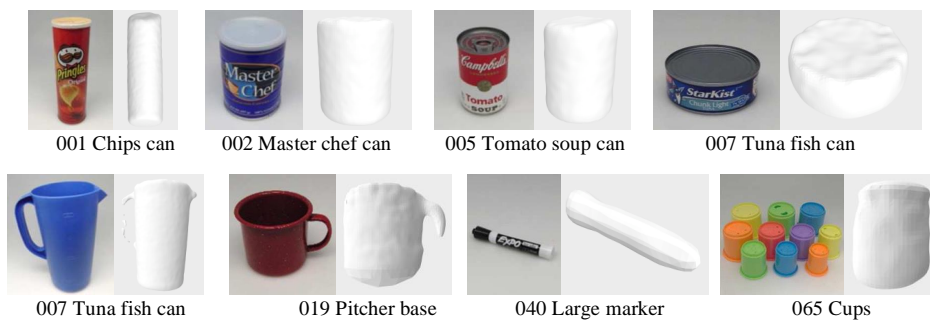


Figure 4.2: All the eight selected cylinder-like objects (images and meshes) in [YCB-28](#) from [YCB](#) set



Figure 4.3: All the eleven selected sphere-like objects (images and meshes) in *YCB-28* from *YCB* set

Box	Cylinder	Sphere
003-cracker box	001-chips can	012-strawberry
004-sugar box	002-master chef can	014-lemon
008-pudding box	005-tomato soup can	015-peach
009-gelatin box	007-tuna fish can	017-orange
010-potted meat can	019-pitcher base	018-plum
026-sponge	025-mug	054-softball
036-wood block	040-large marker	055-baseball
061-foam brick	065-a-cups	056-tennis ball
077-rubiks cube		057-racquetball
		058-golf ball
		063-a-marble

Table 4.1: IDs and names of the selected 28 objects from *YCB* set

The original *PCs* in the *YCB* model set were raw data captured by depth sensors, which contain sensor error and environment noise. The authors of the *YCB* model set created de-noised mesh models out of the raw *PCs* of the objects, using the Truncated Signed Distance Function method (Curless & Levoy, 1996) and Poisson reconstruction (Kazhdan et al., 2006). Despite the de-noising and reconstruction steps, the mesh models still contain a certain level of sensor error, which is visible in Figures 4.1 to 4.3.

The test set was generated using the mesh models of the 28 selected objects, and is henceforth named *YCB-28* model set. The procedure used to generate it

consisted of the three steps detailed below.

The first step was to randomly sample with uniform probability $1,000,000$ points from the mesh model of each object. This initial large point set was called the *point pool*. The second step was to create 20 PCs by randomly sampling 1,000 out of the $1,000,000$ points from the *point pool*. Each of the 20 PCs created from one object model contained a different sample of points. Given that the sampling rate was $(1/1000)$, it is reasonable to think that any two of the 20 PCs had very little sampled points in common. Finally, in the third and last step each PC was centred on the origin and the shape randomly rotated (roll-pitch-yaw rotation). In detail, the YCB-28 model set contained the following PCs:

- *box*: $9 \text{ objects} \times 20 \text{ PCs} = 180 \text{ PCs}$
- *cylinder*: $8 \text{ objects} \times 20 \text{ PCs} = 160 \text{ PCs}$
- *sphere*: $11 \text{ objects} \times 20 \text{ PCs} = 220 \text{ PCs}$
- *Total*: 560 PCs

In summary, the YCB-28 model set contains 560 PCs sampled from 28 mesh models generated from real scenes, and was created for final performance test.

4.1.2 Artificial Primitive Shapes

The Artificial Primitive Shapes (APS) model set was chosen to train the classifiers. It was originally created by Baronti et al. (2019) for research on primitive shape fitting, and can be found in the first author’s GitHub repository ¹.

The APS model set contains objects of the following three shapes: *box*, *cylinder*, and *sphere*. Baronti et al. (2019) created 591 different artificial shapes by changing their height (H), width (W), breadth (B), and diameter (D). In detail, the model set was created from a full-factorial combination of the parameters defining each shape:

- *box*: 220 PCs with $H, W, B \in \{1, 2, \dots, 10\}$ ($H \geq W \geq B$)
- *cylinder*: 190 PCs with $D \in \{0.5, 0.75, \dots, 5\}$ and $H \in \{1, 2, \dots, 10\}$
- *sphere*: 181 PCs with $D \in \{1, 1.05, 1, 10, \dots, 10\}$
- *Total*: 591 PCs

¹https://github.com/lucabaronti/BA-Primitive_Fitting_Dataset.

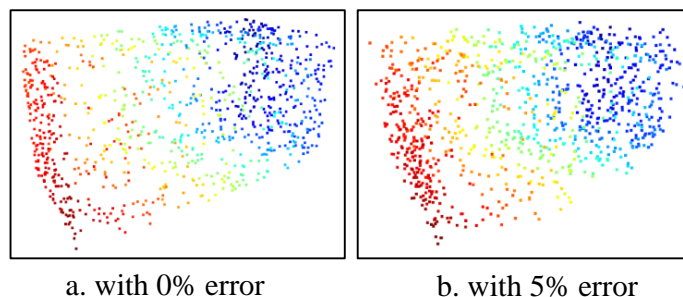


Figure 4.4: Example of created box-shape **PC** with 1000 points, from left to right: **PC** without any error, and **PC** with 5% error (visualised with Open3D (Q.-Y. Zhou et al., 2018))

Given that the **PCs** represent artificial objects, the H, W, B, and D parameters are dimensionless. It is worth mentioning that the elements of the above model set are not corrupted by any sensor error. Henceforth this set will be called *APS-clean*.

Baronti et al. (2019) made also available a tool to inject error (local imprecision simulating sensor inaccuracy) into the **PCs**, as shown in Figure 4.4. In the experiments, 5% of error was added into the *APS-clean* model set to create the *APS-error* model set.

All the shapes were placed with their centers in the origin, and with random orientations. Finally, a test set containing 200 **PCs** for each primitive shape was created. These shapes had random dimensions (H,W,B,D) and contained no sensor error. Henceforth, this set will be called *APS-clean-test*.

In summary, three model sets were created for the experiments: *APS-clean* and *APS-error* for training the classifiers, and *APS-clean-test* for test purposes.

4.1.3 YCB-similar Artificial Primitive Shapes

One last artificial **PCs** model set was created based on the features of the *YCB-28* set. This model set contains artificial primitive shapes of features (H,W,B,D) similar to the objects in *YCB-28*. The motivation for building this set is to simulate the case where some knowledge about the expected shape of the objects is available.

Specifically, the Open3D open-source library (Q.-Y. Zhou et al., 2018) was obtained to enquire the shape features from the mesh models of the twenty-eight objects selected from the *YCB* set. Each mesh model was firstly visualised using visualiser in Open3D. The shape features (H,W,B,D) of the objects were measured from the coordinates of manually picked key points from the visualised model. Three examples of the manually measured shape features are shown in Table 4.2,

whilst the complete table of all the measured shape features from twenty-eight selected YCB mesh models is detailed in Appendix A.

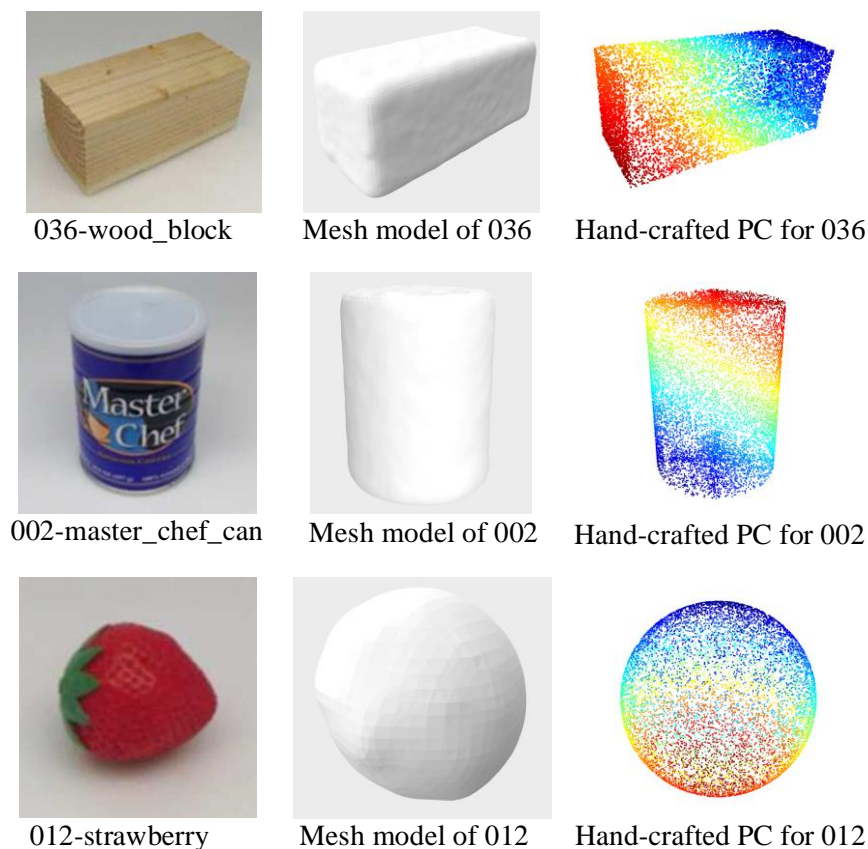


Figure 4.5: Comparison of YCB meshes and hand-crafted PCs

Afterwards, twenty PCs without additive sensor error, were generated for each of the twenty-eight selected objects based on their measured shape features, using the code developed by Baronti et al. (2019). For each of the twenty PCs created for each object, a randomly generated modification was made to each of the shape features (H,W,B,D) independently. Specifically, for each PC model, each feature $K \in (H, W, B, D)$ received a randomly generated modification in the range of $(-5\%, +5\%)$ of its size:

$$K' = [1 + \text{Uniform_Random}(-5\%, +5\%)] \times K \quad (4.1.1)$$

where, $\text{Uniform_Random}(-a, a)$ is a random number generator with uniform probability in the range $(-a, a)$. Thus, each element of the set of generated PCs was similar in shape, although not the same, to the twenty-eight selected objects

ID-Name	Shape	H	W	B	D
036-wood block	box	190.35	80.52	75.81	-
002-master chef can	cylinder	129.79	-	-	96.09
012-strawberry	sphere	-	-	-	48.52

Table 4.2: Manually measured shape features of three mesh models from *YCB-28* set

in *YCB-28*. Figure 4.5 shows examples of artificially created *PCs* near the *YCB-28* mesh models that they simulate. Henceforth, this new model set will be named *YCB-similar*.

The structure of the *YCB-similar* model set is as below:

- *box*: $9 \text{ objects} \times 20 \text{ PCs} = 180 \text{ PCs}$
- *cylinder*: $8 \text{ objects} \times 20 \text{ PCs} = 160 \text{ PCs}$
- *sphere*: $11 \text{ objects} \times 20 \text{ PCs} = 220 \text{ PCs}$
- *Total*: 560 PCs

In summary, the generated *PCs* of the *YCB-similar* model set simulate the shapes of objects in *YCB-28*. They contain no sensor error, and will be used for training the classifiers.

4.1.4 All-in-One Set

In the previous subsections, three artificial *PC* model sets for network training have been created and described for the *YCB-28* classification problem. They include generic geometric shapes (*APS-clean* and *APS-error*), and shapes mimicking the features of the real objects (*YCB-similar*). These three data sets have been grouped into one unique model set called *APS-all*. This experiment simulates a situation where only a limited batch of real shapes is available, and the training examples are beefed up with generic synthetic shapes.

4.2 Shallow Neural Network Architectures

Shallow neural networks (*SNNs*) have a longer history than *DNNs*. Compared to *DNNs*, *SNNs* are known to be faster to train, and are less likely to overfit the training data because they use a much smaller number of parameters (weights). Their main limitation is that they need a pre-processing step to extract the vector of input

features (variables). In **DNNs**, feature extraction is performed by the first layers of the architecture, and is optimised by the learning procedure together with the classifier proper (the last layers of the architecture). Nonetheless, when fed with a descriptive set of features, **SNNs** are known to reach accuracies comparable to those obtained by **DNNs** (Dominguez et al., 2018) in **PC** classification problems.

In this study, the performance of two classical **SNN** models will be compared to the performance of PointNet. The first **SNN** is the widely used Multi-Layer Perceptron (**MLP**), which is described in Section 4.2.1. The second is the Radial Basis Function Network (**RBFN**), which is described in Section 4.2.2. Both **SNNs** were fed features extracted using the method described in Section 4.2.3.

4.2.1 Multi-Layer Perceptron

The Multi-Layer Perceptron (**MLP**) is a popular feed-forward and versatile neural network used for classification and modelling problems (Andina & Pham, 2007). The **MLP** is usually trained using the back-propagation learning algorithm, which was firstly proposed by Rumelhart et al. (1985). The versatility of the **MLP** stems from its ability to approximate any function to any desired degree of accuracy (Hornik et al., 1989).

In its standard configuration, an **MLP** contains one input layer (which acts as a buffer), one or two hidden layers, and one output layer. Each layer contains a number of neuron units, which are fully connected to the neurons of the previous (incoming connections) and next (outgoing connections) layer. Each neuron gathers the outputs of the neurons of the previous layer (or the input vector of variables) via the incoming connections, combines and processes them, and broadcasts to all the neurons of the next layer (or the user) its output. Each connection between two neurons modulates the magnitude of the signal via an adjustable weight. These weights are changed by the learning algorithm, and encode the 'knowledge' of the network. The mapping of a standard neuron unit j can be fully described by the following equation:

$$y_j = f(h_j) = f\left(\sum_{i=1}^N w_{ji}x_i + w_{j0}\right) \quad (4.2.1)$$

where y_j is the response (output) of neuron j ; x_i is the output of the i^{th} neuron in the previous layer; N is the number of units in the previous layer; w_{ji} is the weight of the connection between neuron n_j and n_i ; w_{j0} is a parameter called the bias term; and f represents the transfer function.

The number of units in the input and output layers correspond to the number of input variables to the neural network and the dimensionality of the output.

The number of hidden layers and neuron units in each hidden layer are hyper-parameters which depend on the problem domain, and are often experimentally defined. The details of the hyper-parameters used in the experiments reported in this chapter are described in Section 4.3.

4.2.2 Radial Basis Function Network

The Radial Basis Function Network (**RBFN**) was firstly proposed by [Broomhead & Lowe \(1988\)](#) in 1988. Like the **MLP**, the **RBFN** is a popular feed-forward neural network used for modelling and classification problems ([Andina & Pham, 2007](#)). the **RBFN** has a strictly defined architecture, featuring one input layer, one hidden layer, and one output layer. The activation function of the hidden layer is the radial basis function. The input layer of an **RBFN** acts as a buffer, and broadcasts the input vector to each neuron in the hidden layer. The hidden neurons process the input vector via the activation function (radial basis function), whilst the output neurons perform a linear summation of the weighted outputs of the hidden neurons.

The radial basis function is a real-valued function with output: $[0, \infty) \rightarrow \mathbb{R}$. Specifically, the radial basis function uses a radial kernel to transfer the Euclidean distance between the input vector (\mathbf{x}) and a predefined center point (\mathbf{c}) into a strictly positive real-value. A general definition of a radial basis function can be described by the following equation:

$$\varphi_{\mathbf{c}}(\mathbf{x}) = \hat{\varphi}(\|\mathbf{x} - \mathbf{c}\|) \quad (4.2.2)$$

where, $\varphi_{\mathbf{c}}(\mathbf{x})$ is a basis function with predefined center point \mathbf{c} over input vector \mathbf{x} ; $\hat{\varphi}$ is a radial kernel, typically a non-linear strictly positive definite function; $\|\mathbf{x} - \mathbf{c}\|$ calculates the Euclidean distance between input vector \mathbf{x} and predefined center point \mathbf{c} .

In this work, a Gaussian function was chosen as the applied radial kernel. If the Euclidean distance between the input vector \mathbf{x} and predefined center point \mathbf{c} is denoted as r , the radial basis function with Gaussian kernel is defined as follows:

$$\varphi_{\mathbf{c}}(\mathbf{x}) = \hat{\varphi}(r) = e^{-(\sigma r)^2} \quad (4.2.3)$$

where, σ is a parameter that scales the kernel input r .

Thus, the mapping performed by the whole **RBFN** can be described by the following equation:

$$y_j = f(\mathbf{x}) = \sum_{i=1}^N w_{ji} \varphi_{\mathbf{c}_i}(\mathbf{x}) \quad (4.2.4)$$

where, y_j is the output of output neuron j ; \mathbf{x} is the input to the neural network; N is the number of hidden neurons; w_{ji} is the weight of the connection between hidden neuron i and output neuron j ; and function $\varphi_{\mathbf{c}_i}$ is a radial basis function applied on hidden neuron i with predefined center point \mathbf{c}_i .

Combining Equations (4.2.2) to (4.2.4), the output of an output neuron in an **RBFN** defined with Gaussian radial kernel becomes:

$$y_j = f(\mathbf{x}) = \sum_{i=1}^N w_{ji} \cdot e^{-[\sigma_i(\|\mathbf{x}-\mathbf{c}_i\|)]^2} \quad (4.2.5)$$

where \mathbf{c}_i and σ_i are respectively the centre and spread of the basis function $\varphi_{\mathbf{c}_i}$. The parameters \mathbf{c}_i , σ_i , w_{ji} are changed by the learning procedure. The details of the **RBFN** used in experiments reported in this chapter are described in Section 4.3.

4.2.3 Numerical Feature Generation

A **PC** is a data structure containing an unordered list of x-y-z coordinates. Differently from PointNet, the **MLP** and **RBFN** treat the input as vector, and are thus sensitive to the ordering of its elements. Thus, the **PCs** cannot be fed directly into the **MLP** or **RBFN**. In this section, a feature generation scheme to extract numerical features from the **PCs** is described.

In the tests, it is assumed that the objects are in unknown orientation. Thus, the extraction process starts with aligning the shapes with the coordinate axes. For this purpose, principal component analysis (Pearson, 1901; Hotelling, 1933) was used to extract the eigenvectors of the **PC**. The **PC** is then placed with its centroid in the origin, and its eigenvectors are aligned with axes of the Cartesian coordinates. Since the eigenvectors broadly correspond with the main axes of the shapes, this method was proven to align with reasonable accuracy the **PC**.

In real-life, a human eye can recognise a primitive shape from its orthogonal projections onto the three planes $x = 0$, $y = 0$, and $z = 0$. Namely, a cube with its sides aligned with the Cartesian axes will create three rectangular shapes (one per plane), a sphere will generate three disks, and a cylinder will create two rectangular shapes and one disk. In summary, recognising the three 3D primitive shapes boils down to recognising two 2d shapes (rectangle and disk) in their projections. This idea is exploited as follows.

After principal component analysis alignment, the points in the cloud are projected onto the three planes $x = 0$, $y = 0$, and $z = 0$. For example, the projection of a point of coordinates $z = (x, y, z)$ onto the $z = 0$ plane is $z = (x, y, 0)$. For each 2D projection on a plane:

1. The coordinates of all the points are transformed into 2D polar-coordinates: (r, θ)
2. The plane is divided into 64 sectors (intervals of θ).
3. For each of the 64 sectors, the radius r of the most distant point from the origin is taken as representative of the interval. Namely, the representative of interval $1 \leq j \leq 64$ is $m_j = \max(r_i)$ where $1 \leq i \leq N$ indicates one of the N points of the cloud.
4. the arithmetic mean and standard deviation of the 64 representatives m_j of the interval are calculated for each plane

The features extracted from one **PC** form a 6-dimensional vector:

$$(m_x, \delta_x, m_y, \delta_y, m_z, \delta_z)$$

where m_k and δ_k ($k = x, y, z$) are respectively the mean and standard deviation of the 64 representatives on the planes $x = 0$, $y = 0$, and $z = 0$. In a perfect **PC** without error, all the most distant points of a disk will be at the same distance from the origin, hence $m_x = m_y = m_z$ and $\delta_x = \delta_y = \delta_z = 0$. Also, the most distant points of a rectangle will not be at the same distance from the origin, $\delta_{x,y,z} \neq 0$, and in general $m_x \neq m_y \neq m_z$. This will hold as long as the error level is reasonable, namely that it won't completely blur the shapes of the projections, and when the model has been cleaned of sensor error.

4.3 Experiments and Results

This section will describe the experimental set up and the results obtained by PointNet and the two **SNNs**. Three artificial model sets, *APS-clean* model set, *APS-error* model set, and *YCB-similar* model set, were used for network training. The final performance of trained network was examined using *YCB-28* model set. In all experiments 10 independent learning trials were performed, and the results were statistically analysed.

All the experiments were run using the hardware described in Table 3.3. The experimental results will be displayed using the multi-class accuracy and box-plots described in Section 3.4.

The PointNet architecture used in the experiments was obtained from open-source code made available by Qi, Su, et al. (2017) in their Github repository ². Following some preliminary tests, the structure and most of the hyper-parameters

²<https://github.com/charlesq34/pointnet>

Parameter	Value
training algorithm	Adam
decay rate β_1 in Adam	0.9
decay rate β_2 in Adam	0.999
numerical constant $\hat{\epsilon}$ in Adam	1×10^{-7}
initial learning rate	0.0001
minimum learning rate	0.00001
learning rate decay rate	0.7
learning rate decay step	200,000
initial Batch Normalisation momentum	0.5
maximum Batch Normalisation momentum	0.99
Batch Normalisation momentum decay rate	0.5
Batch Normalisation momentum decay step	200,000

Table 4.3: Hyper-parameters of PointNet used in the experiments (please refer to Section 3.1.1 of a detailed description of these parameters)

of PointNet were kept as originally designed by Qi, Su, et al. (2017), or manually optimised. They are shown in Table 4.3. The MLP and RBFN SNNs were implemented using the C++ code made available by Castellani (2013). The MLP was trained using the standard BP algorithm with momentum term (Pham & Liu, 1995), whilst the RBFN was trained using first a KNN-based algorithm for a broad brush optimisation of the RBF parameters, and then a fine tuning of the whole network parameters using the BP algorithm. The hyper-parameters of the SNNs, and their learning procedure were optimised by trial and error. They are detailed in Table 4.4.

Parameter	MLP	RBFN
training algorithm	back propagation	back propagation
number of inputs	6	6
number of hidden layers	1	1
number of hidden neurons	15	20
number of outputs	3	3
learning rate	0.01	0.01
training epoch	2,000	5,000

Table 4.4: Hyper-parameters of MLP and RBFN used in the experiments

Two learning hyper-parameters, the batch size and number of training epochs,

have the largest effect on the results. Section 4.3.1 describes the procedure followed for their experimental optimisation. This procedure was carried out using the artificial *APS-clean* model set. After optimisation, since the PointNet was always tested on the same benchmark problem (*YCB-28*), the hyper-parameters were fixed for all the experiments.

Once trained using one of the *APS-clean*, *APS-error*, *YCB-similar*, and *APS-all* sets, depending on the experiment, the classifiers (PointNet, MLP and RBFN) were tested using the *YCB-28* model set, and the experimental results are reported in Section 4.3.2.

4.3.1 Hyper-parameters Optimisation for PointNet

PointNet was trained using the *APS-clean* model set, and the results validated using *APS-clean-test* set (Section 4.1.2). This corresponds to training the DNN with perfect shapes as for example those obtained from CAD models.

Batch Size Optimisation

To optimise the batch size of PointNet, tests were performed with batch size varied from 10 to 100 in steps of 10, using a fixed training epoch of 200. Ten learning trials were performed for each batch size setting.

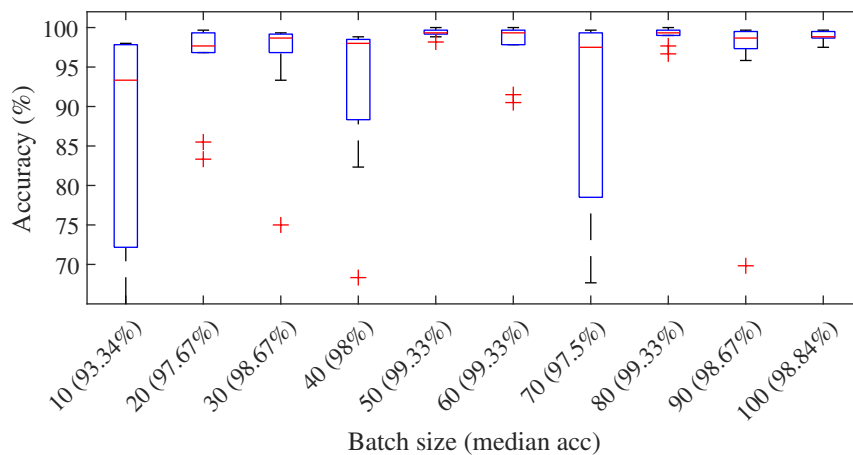


Figure 4.6: Performance of PointNet on the *APS-clean* model set as the batch size setting is varied

The experimental results are shown using box-plots in Figure 4.6. The red line within the box indicates the median result of the 10 independent learning trials. In terms of median accuracy, the performance improves noticeably when the batch

size is increased from 10 (93.34% median accuracy) to 20 (97.67%), and from 40 (98.0%) to 50 (99.33%). Both improvements are statistically significant at an $\alpha = 0.01$ confidence level: the p-value is 0.0343 for the difference between the results obtained using batch sizes of 10 and 20, and $p = 0.0006$ for the difference between the results obtained using batch sizes of 40 and 50.

Beyond a size of 50, the statistical analysis suggests there are no benefits in any further increase of the batch size. However, as the batch size increases, the training procedure appears to become more consistent (smaller width of the box plots), and for this reason a batch size 100 was chosen. Although this choice is the most computationally intensive, the training process can be sped up using GPU acceleration.

Since the number of **PCs** constituting the model sets used in the experiments is not a multiple of 100, the number of training examples in the last batch fed to the *Adam* optimiser had to be brought to 100. This was achieved by duplicating randomly picked **PCs** from the whole model set. For example, the *APS-error* set contained 591 **PCs**, which were fed in 6 batches of 100 **PCs** each. The last batch was formed by the remaining unused 91 examples, and 9 randomly picked duplicates.

Training Epoch Optimisation

After the batch size had been fixed at 100, the number of training epochs was optimised by trial and error. Experiments were performed increasing the number of epochs from 20 to 200 in steps of 20. The results are shown in Figure 4.7.

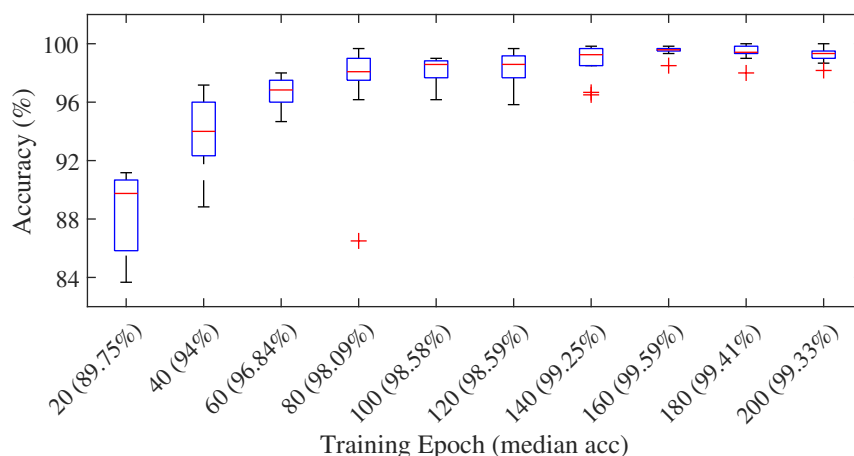


Figure 4.7: Performance of PointNet on the *APS-clean* model set as the number of training epochs is varied

A progressive improvement in the performance of PointNet as the number of training epochs is increased until 160 can be seen in Figure 4.7. Pairwise Mann-Whitney statistical tests indicated that, the performance of PointNet trained with 160 epochs is significantly superior to the performance obtained using any smaller number of training epochs (from 200 to 140 epochs). Further increases of the number of training epochs beyond the 160 epochs did not yield any significant improvement in performance. Consequently, the number of training epochs was fixed to 160.

A similar experiment was done to optimise the number of training epochs for the case where PointNet is trained using the *APS-all*. Slightly different from previous experiments, this group of experiments were tested using *YCB-28*. The results are visualised in Figure 4.8. From examination of Figure 4.8, a training duration of 160 epochs seems satisfactory in terms of accuracy and consistency.

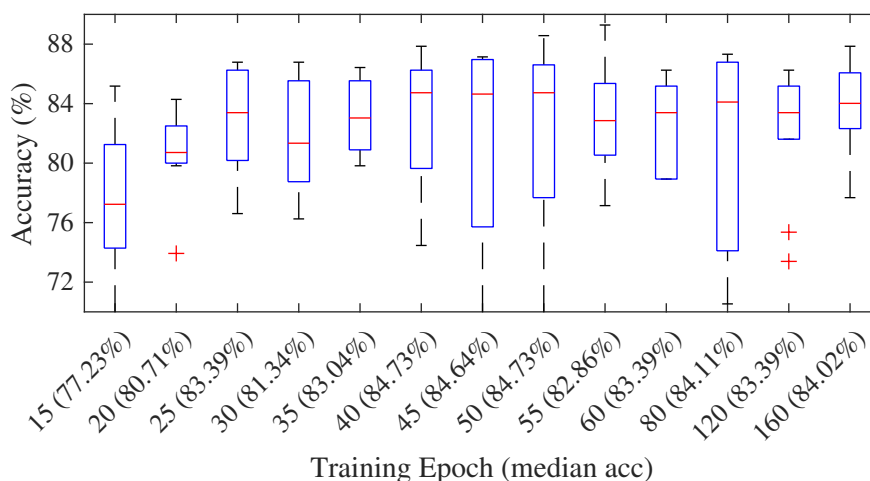


Figure 4.8: Performance on the *YCB-28* model set of PointNet trained using the *APS-all* set with different training epoch

4.3.2 Experimental Results - Artificial Model Sets

As mentioned in Section 4.3, three instances of PointNet were trained using three artificial model sets: *APS-clean*, *APS-error*, and *YCB-similar*, using the hyperparameters listed in Table 4.3 and discussed in Section 4.3.1. The performance of the trained PointNets was evaluated on their accuracy on the *YCB-28* model set, and compared to that of two *SNNs*: an *MLP* and an *RBFN*. The results of the experiments are visualised using box-plots in Figure 4.9. Each box visualises the five-number summary of the accuracy results attained in the 10 independent learning trials, each performed using a given classifier and training set.

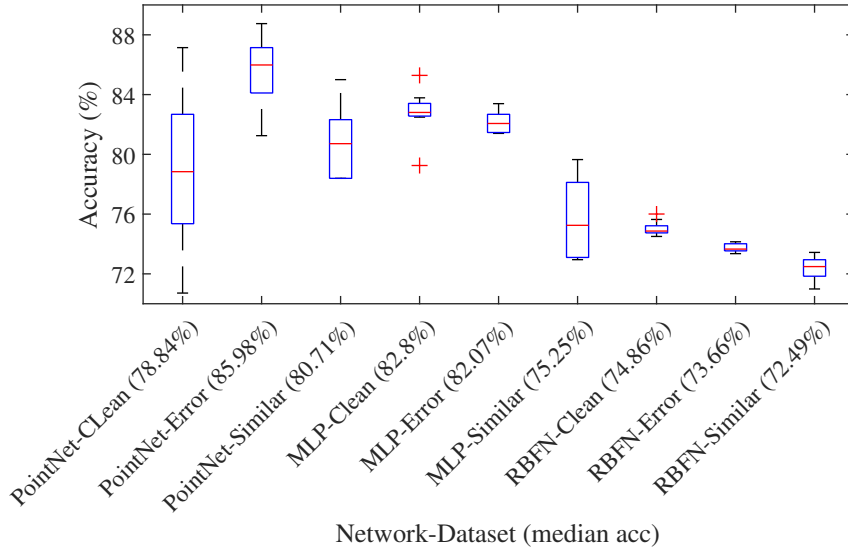


Figure 4.9: Accuracy results obtained on the *YCB-28* model set by the three classifiers when trained using different sets: *APS-clean* (Clean), *APS-error* (Error) and *YCB-similar* (Similar)

Figure 4.9 shows that PointNet can be trained to achieve an average (median) accuracy of 86% circa training the *DNN* on the *APS-error* model set. Pairwise Mann-Whitney tests indicated that this result is significantly superior to the accuracy results obtained using any other combination of classifier and training set. PointNet did not perform equally well when trained using the *APS-clean* and *APS-similar* model sets, although the performance on the latter (81% circa average accuracy) was still adequate.

Despite the unsophisticated feature extraction method used, the *MLP* performed remarkably well. Trained using the *APS-clean* model set, it achieved nearly 83% training accuracy, and slightly less (82%) when trained using the *APS-error* set. Compared to PointNet, the *MLP* obtained more consistent learning results, as shown by the width of the box plots. The *RBFN* was the clear underperformer of the three tested classifiers. Finally, training the classifiers on the *YCB-Similar* did not provide any visible benefit, particularly for the two *SNNs*.

The accuracy obtained training the PointNet on the *APS-error* set represents the best-so-far result. This result was compared with the accuracy attained when PointNet was trained using the *APS-all* model set (Section 4.1.4). The comparison is visualised using box-plots in Figure 4.10. Each box visualises the five-number summary of the accuracy results attained in the 10 independent learning trials.

Despite using a roughly three times larger training set, the PointNet trained

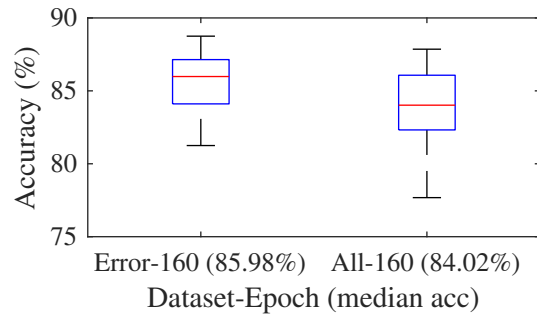


Figure 4.10: Performance comparison on the *YCB-28* model set of PointNet trained using the *APS-error* (Error) and *APS-all* (All) set

using the *APS-all* set did not improve the performance obtained by the PointNet trained using the *APS-error* set. A pairwise Mann-Whitney suggests there is no significant difference ($p = 0.257$) between the results obtained training PointNet using the *APS-all* and *APS-error* model sets.

4.4 Discussion

This chapter presented a study aiming to evaluate the ability of PointNet to correctly recognise objects from real-life scenes, after being trained on artificial geometric models. The hyper-parameters of PointNet were tuned using artificial models. The study aimed also at testing whether simple *SNNs* were able to obtain results comparable to those obtained by the much more complex PointNet.

The experimental tests showed that, in terms of accuracy, PointNet had indeed an edge, albeit small, on a standard shallow *MLP* classifier. However, the *MLP* showed more consistent training results. The tests also confirmed that PointNet performs best when trained on scenes that were perturbed with some level of random error. Training PointNet using images of features (size, proportions between dimensions) similar to those to the real images, instead of training it with more general artificial shapes, did not improve the accuracy of the classifier.

The above results can be interpreted remembering the outcomes of the experiments in the previous chapter Section 3.5: namely that the recall accuracy of PointNet is sensitive to error in the scenes. If the irregularities of real-life scans are interpreted as error, then a PointNet trained on uncorrupted shapes is bound to perform poorly. Training PointNet on models of features that are closer to those of the real-life objects (*YCB-Similar*) does not make a difference, because the *YCB-Similar* shapes are still clean geometric shapes. The *YCB-Similar* model set did not generate any improvement in accuracy even when combined with the other two data sets. The only solution that brought improvements to the accuracy of

PointNet was to perturb the artificial shapes with some local error.

The overall conclusion of this set of experiments is that PointNet indeed provides some moderate improvement in performance respect to a basic **MLP** shallow architecture, although at the price of slightly less consistent learning results.

The main advantage of PointNet is that the data from the raw point clouds do need only minimal pre-processing (normalisation and down sampling). In particular, they do not need the feature extraction process required by the **MLP**. The feature extraction process is embedded in the first block of layers of the PointNet, and optimised simultaneously to the classifier block by the **DNN** learning algorithm. In particular, the concurrent optimisation of the feature extraction and classification procedures is likely to account for the superior accuracy of PointNet respect to the **MLP**. In the latter case, the feature extraction process has been carried out prior to the classifier training procedure, and it is possible that the criterion of the former did not perfectly match the inductive and representational biases of the latter. Compared to other **DNNs**, the recognition accuracy of PointNet is also invariant to rotation and translation of the shapes.

The above properties make PointNet preferable for object shape recognition, although its sensitivity to sensor error must be taken into account given the high level of variability and potentially imprecise sensing in remanufacturing applications.

4.5 Conclusions

This chapter presented a study on the possibility of training PointNet to recognise primitive geometric shapes in **PCs**, and use it to identify similar shapes in 3D scans of everyday objects. The performance of PointNet was evaluated on a subset of 28 models selected from **YCB** model set. The tests focused on three types of primitive shapes: boxes, cylinders, and spheres.

The experimental results confirmed the feasibility of the task. If trained on perfect geometric primitives, PointNet confirmed its difficulty to generalising to noisy or moderately irregular shapes. However, if the training set is perturbed with some noise, PointNet is able to learn the task with high accuracy. Compared to a shallow **MLP ANN**, PointNet performed with higher accuracy but slightly lower consistency. Overall, the main advantage of PointNet is its embedded feature extraction ability, irrespective of the pose of the objects. It was concluded that PointNet is indeed a good candidate for object primitive shape recognition, provided that the samples of the training set take into account possible sensor error in real-life scenarios.

Chapter 5

Coevolutionary Deep Neural Network Training

The success of the **DNN** structure *AlexNet* (Krizhevsky et al., 2012) had a huge impact on the research community, contributing to nowadays' popularity of deep learning in machine vision. *AlexNet* obtained dramatically superior accuracy in object recognition tasks, compared to traditional shallow and other early **DNN** structures. The strength of *AlexNet* was in the depth and complexity of its structure, and the use of **GPUs** to make its high computational complexity manageable in real time.

Most of the research on **DNNs** in the last decade focused on the architecture and learning algorithms, to improve the network performance and execution speed, and on the **GPU** implementation to further accelerate computations. Fast execution is of particular concern in **DNN** practice during the training phase, which often entails many cycles of repeated processing of a large amount of complex training examples, and the periodic update of millions of connection weights. For example, in their seminal paper, (Krizhevsky et al., 2012) trained *AlexNet* for 90 cycles, each cycle consisting of the presentation of 1.2 million high-resolution images and the update of 60 million weights (Krizhevsky et al., 2012).

To exacerbate the problem, due to the increasing availability and sophistication of sensing and imaging equipment, the size and complexity of machine vision tasks is constantly growing. For example, the 2D *ImageNet* benchmark repository was firstly established with 3.2 millions of images in 2009 (Deng et al., 2009). By 2015, its number of images had increased to around 14 millions (Russakovsky et al., 2015). Increasing the size of the *ImageNet* set did facilitate the attainment of increasingly robust performances from the classifiers, but also introduced redundant data and increased the computational complexity of the learning process.

The computational complexity of training a **DNN** makes it also problematic to use alternative training methods to the standard gradient-based algorithms. The

literature on learning algorithms based on computationally intensive global optimisation techniques is so far minimal. Implementations of EAs for deep learning are scarce, and often limited to simple data structures like the 28×28 pixel 2D images of the MNIST data set (LeCun, 1998) used by Martín et al. (2018). Particle Swarm Optimisation, an instance of swarm optimisation algorithm, was used by M. Li et al. (2019), but in this case the complexity of the DNN was very low. Perhaps the only instances of evolutionary DNN training that are comparable in complexity to PointNet, are the systems created by Young et al. (2017) and Real et al. (2017). Despite being massively parallelised, the authors of both evolutionary DNNs reported long training times. Sun et al. (2019) also reported execution times in the order of several days for their evolutionary DNN implementation.

In this chapter, it will be investigated how to reduce the computational complexity of the learning task for a PointNet classifier in an evolutionary scheme, without compromising its final accuracy. The idea is to reduce the number of PCs at each training cycle, adaptively focusing on yet unlearned examples.

5.1 Coevolutionary ANN Training

In the CGA proposed by Paredis (1995), shallow ANN classifiers were evolved in a predator-prey scheme. Namely, the ANN classifiers were considered as *predators*, whilst the training samples were seen as the *preys*. The fitness of the predators was determined by the amount of prey that they were able to capture (i.e. correctly classify), whilst the fitness of the prey was calculated on their ability to avoid predation (i.e. how difficult it was for the classifiers to correctly identify them). Predators and preys were randomly picked and paired, and the outcome of the encounter was used to incrementally update their fitness based on the life time fitness evaluation (Hillis, 1990) procedure. Complementary fitness is an important concept in predator-prey schemes: success for the predator (correct classification of the prey) corresponds to failure for the prey, and vice versa.

The CGA evolved the population of predators using the GENITOR, a GA proposed by Whitley (1989). The preys were not evolved, but their fitness was used to determine their chances to be picked up and paired to a predator (Paredis, 1995). The aim of the procedure is to create a tug of war between the best classifiers and the most difficult training examples.

Castellani (2018) further developed and investigated the coevolutionary approach to ANN training, developing two new coevolutionary algorithms: the co-adaptive CANNT algorithm, and the coevolutionary CENNT algorithm.

CANNT was inspired by the CGA, and evolved only the predator population. Like in CGA, the prey population was not evolved but the fitness of the individuals was used to determine their likelihood of being picked and paired to a

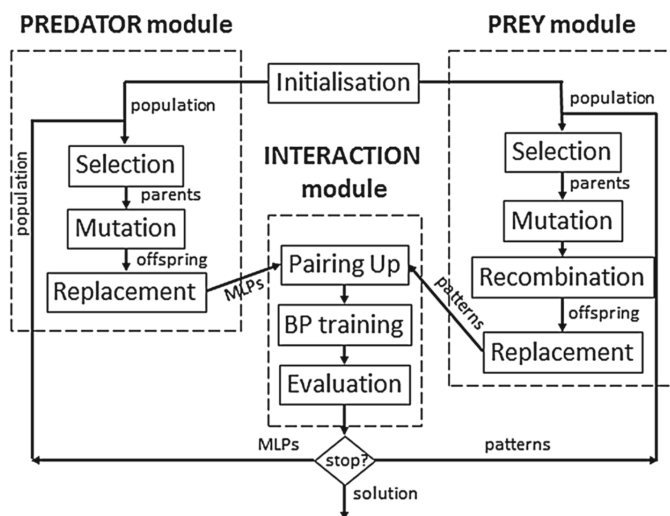


Figure 5.1: Flow chart of CENNT proposed by Castellani (2018)

predator. Differently from the CGA, CANNT matched each predator to a batch of randomly picked preys at each generation, and evolved the predator population using an evolutionary scheme based on generational replacement, mutation, and the fitness ranking selection scheme. CANNT did not feature genetic crossover, whilst employed Lamarckian learning (Cortez et al., 2002) to accelerate the training process.

CENNT evolved both predators and preys, respectively ANN classifiers and subsets of the training set. CENNT used the same algorithm used in CANNT to evolve the predator population. The prey population was encoded as a binary vector, where the value of each element defined whether the corresponding learning example was selected for a given training subset or not. The prey population was evolved using a standard GA. Complementary fitness was again used: a predator (ANN) and a prey (training examples subset) were randomly paired up, and whilst the fitness f_p of the predator was equal to its accuracy a on the prey subset of examples, the fitness of the prey was equal to $f_v = 1 - a$.

Trained on a number of benchmarks, CANNT and CENNT brought improvement to the classifier accuracy and training speed, in comparison to traditional GA-based training. Due to the large reduction of used data samples, the speed of the co-evolutionary algorithms was comparable to that of the much faster back-propagation algorithm on the largest training sets. In this chapter, a new coevolutionary approach is proposed to speed up the PointNet training procedure.

5.2 Coevolutionary Selected Training (CEST)

This section introduces the Co-Evolutionary Selected Training (**CEST**) algorithm, a novel coevolutionary scheme aiming to increase the speed and accuracy of the PointNet training procedure.

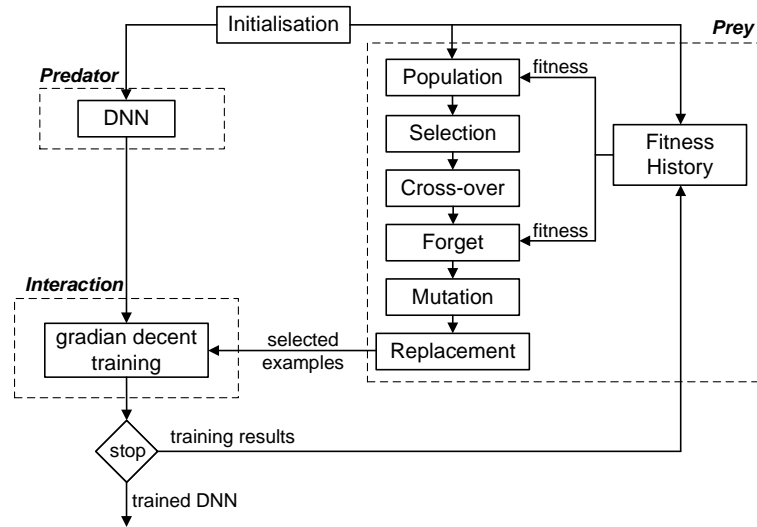


Figure 5.2: Flow charts of **CEST**

CEST is based on one *Predator* (DNN) against a population of *Prey* (subsets of training data). The flowchart of the proposed **CEST** algorithm is shown in Figure 5.2. The *Predator* module of **CEST** is in reality the standard PointNet with *Adam* optimiser. In all experiments, the PointNet implementation by Qi, Su, et al. (2017) was used.

The *Prey* module of **CEST** is where the true evolutionary process takes place. It takes the classification results (fitness evaluations) from the *Interaction* module, and uses them to evolve subsets of the most difficult examples. Very much like the interaction module of **CENNT**, the *Interaction* module of **CEST** takes a set of training examples from the *Prey* module, and uses it to train the PointNet and evaluate its classification accuracy. This set of training examples is built concatenating the subsets evolved in the *Prey* module. Since the most difficult examples are likely feature in many of the subsets forming the population of the *Prey* module, and the most easy examples in none, the set of **PCs** fed to PointNet is likely not to contain all the examples of the training set, and several duplicate examples. The training patterns from the *Prey* module are shuffled before being fed into the **DNN** for training.

CEST can be thought of as two learning algorithms opposed one another. Whilst the *Predator* module uses the standard *Adam* optimiser to improve the accuracy of the classifier, the *Prey* module uses evolutionary optimisation to generate subsets of difficult examples.

The *Prey* module is based on a traditional **GA**. Candidate solutions (subsets of the training set of **PCs**) are encoded as binary vectors (*chromosomes* in **GA** terminology), where each element of the vector (*gene* in **GA** terminology) takes the value of '1' if the associated **PC** is used to train and evaluate the PointNet, '0' otherwise. These strings are evolved via standard genetic operators (**Goldberg & Holland, 1988**): fitness proportionate selection with elitism, 2-point crossover, bit-flip mutation, and generational replacement. The size N_e of the elite set of individuals that will be copied into the next generation is a user-set system parameter.

A particularity of **CEST** is that, in addition to the fitness of the individuals (sets of **PCs**), it keeps also track of the fitness of each gene (the individual **PCs**). At the end of each training cycle, the fitness of a gene is calculated as follows:

$$f_i(t) = w_c \cdot \frac{m_i(t)}{a_i(t)} + w_o \cdot f_i(t-1) \quad (5.2.1)$$

where, i is the gene index; t is the training cycle; $a_i(t)$ is the number of times the classifier (predator) tried to classify the i_{th} **PC**; $m_i(t)$ is the number of times the i_{th} **PC** was misclassified, and w_c and w_o are two user-defined hyper-parameters which respectively define the weight of the current evaluation and the trace of the previous evaluations. The default values for w_c and w_o are respectively 0.2 and 0.8. If $a_i(t) = 0$, $\frac{m_i(t)}{a_i(t)}$ is set to 1.

The fitness of an individual (a subset of **PCs**) is equal to the average fitness of its 'on' genes (those of value '1'). Namely, the fitness $F_p(t)$ of individual p , defined by the chromosome string $g_p = (g_{p1}, \dots, g_{pN_g})$ of length N_g (total number of **PCs** in the training set), at training cycle t , is calculated as follows:

$$F_p(t) = \frac{\sum_{j=1}^{j=N_g} g_{pj} \cdot f_j(t)}{\sum_{j=1}^{j=N_g} g_{pj}} \quad (5.2.2)$$

where

$$g_{pj} = \begin{cases} 0 & \text{gene index } j \text{ is 'off'} \\ 1 & \text{gene index } j \text{ is 'on'} \end{cases} \quad (5.2.3)$$

In addition, the history of the last L_h fitness evaluations is stored for each gene in a 2D matrix \mathcal{FH} of size $N_g \times L_h$. The default value for L_h is 12. The history matrix \mathcal{FH} bears similarities with the life time fitness evaluation used by **Paredis**

(1995), and is used by a novel *forget* operator designed for CEST. This operator is applied to a new individual after it has been created via crossover by two parents. The *forget* operator first calculates for each gene the average fitness A_j in the last L_h training cycles:

$$A_j = \sum_{i=1}^{i=L_h} \frac{\mathcal{FH}_{ji}}{L_h} = \sum_{i=0}^{i=L_h-1} \frac{f_j(t-i)}{L_h} \quad (5.2.4)$$

Then the *forget* operator sets to '0' all the values of the genes in the chromosome that have an average history A_j below a predefined threshold Θ . The purpose of the operator is to weed out (forget) training samples that are not frequently misclassified. At the beginning of the evolutionary process, the elements of the matrix \mathcal{FH} are set to '1'.

The hyper-parameters related to the management of the genes are listed in Table 5.1 with example default values used in the experiments. Specifically, the genes initial fitness f_{gene}^{init} is the initial fitness for each gene in the history fitness matrix \mathcal{FH} .

Name	Symbol	Default
number of genes	N_g	-
genes initial fitness	f_{gene}^{init}	1.0
weight of fitness trace	w_o	0.8
weight of current fitness	w_c	0.2
history span	L_h	12

Table 5.1: Hyper-parameters related to the management of the genes in CEST

The remaining evolutionary hyper-parameters are listed in Table 5.2. The evolutionary algorithm is a typical GA using fitness proportionate selection and generational replacement with elitism. These procedures are defined by hyper-parameters like the population size N_p , number of offspring N_c , and elite size N_e . At the end of each evolution epoch (generation, in EA parlance), the new population is made of the N_e elite individuals from the last epoch, and the $N_p - N_e$ fittest individuals of the N_c offspring population. There are two types of $SC_{individual}$ initialisation procedure for the individuals of the prey module: *full* initialisation sets all the gene values to '1', *random* initialisation sets all the gene values to randomly drawn values. Preliminary tests indicated that the former supports early exploration of the solution space, and hence reduces the likelihood of sub-optimal convergence. As previously mentioned, the *forget* operator in a new individual

Name	Symbol	Default
population size	N_p	-
number of children	N_c	-
elite size	N_e	-
individual init scheme	$SC_{individual}$	full
selection scheme	$SC_{selection}$	roulette wheel
crossover scheme	$SC_{crossover}$	two-point crossover
<i>forget</i> operator threshold	Θ	0.1
mutation rate for individuals	m_i	-
mutation rate for genes	m_g	-
learning cycles	E	-

Table 5.2: *GA* hyper-parameters of the prey module in *CEST*

switches off all the genes of average fitness $A_j < \Theta$, based on the history matrix \mathcal{FH} . Mutation is implemented via the standard flip-bit operator, and depends on two parameters: the probability m_i that an individual is selected for mutation (mutation rate for individuals), and the probability m_g that a gene of an individual selected for mutation is changed (mutation rate for genes).

5.3 Experimental Model Sets

The learning task in the experiments will be *PC* classification over three different model sets: Artificial Primitive Shapes (*APS*, Chapter 4), Mechanical Objects (Chapter 3), and the Princeton ModelNet40 set (Wu et al., 2015). As mentioned in the previous chapters, the three model sets differ by size (respectively 600, 2,400, and 9,840 *PCs*) and number of classes (respectively 3, 12, and 40).

In the experiments discussed in Chapter 4, the best results were obtained training the PointNet using the *APS-error* model set, where the shapes are perturbed with a 5% error. This model set was retained for the first set of experiments reported in this chapter. In addition, the *APS-all* model set was retained too, because of its size and the competitive learning results obtained when employed to train PointNet. Like in Chapter 4, the *YCB-28* set was used to test the performance of the trained PointNet.

A noticeable feature of ModelNet40 is that it is an unbalanced model set, that is, the 40 classes are represented by a different number of samples. In previous study, Castellani (2018) demonstrated that coevolutionary approaches are intrinsically suited to tackle the problems created by such sets. The experiments in this

	APS-error	APS-all	YCB-28
Training/Test	training	training	test
Artificial/Real	artificial	artificial	real
Clean/Error	error	mix	-
Set Size	591	1,742	560
Classes	3	3	3

Table 5.3: Details of employed APS and YCB model sets

	MEch12-clean	MEch12-error	ModelNet40	ModelNet40
Training/Test	training	test	training	test
Artificial/Real	artificial	artificial	artificial	artificial
Clean/Error	clean	error	clean	clean
Set Size	2,400	1,200	9,843	2,468
Classes	12	12	40	40

Table 5.4: Details of MEch12 and ModelNet40 model sets

chapter will also test this hypothesis.

The main details of the model sets used in the first set of experiments are summarised in Table 5.3. The experiments contain also a study on the effect of the parameterisation on the functioning and performance of CEST.

For the second set of experiments, the training set containing 2,400 PCs of clean partial views of mechanical objects described in Section 3.2.1 was used to train PointNet. Henceforth, this model set will be called *MEch12-clean*. The 1,200 error-corrupted partial-views test set described in Section 3.3.2 was used for testing. Henceforth, this second model set will be called *MEch12-error*.

Finally, in the third set of experiments the well-known 3D CAD model model set ModelNet40 was used. ModelNet40 contains 12,311 artificial CAD models from 40 object categories. The training set contains 9,843 models, whilst the test set contains 2,468. From the surface of the CAD models, Qi, Su, et al. (2017) uniformly sampled points to build PCs. In this study, the scenes with 1,024 points made by Qi, Su, et al. (2017) were used.

Parameter	Value
decay rate β_1 in Adam	0.9
decay rate β_2 in Adam	0.999
numerical constant $\hat{\epsilon}$ in Adam	1×10^{-7}
initial learning rate	0.0001
minimum learning rate	0.00001
learning rate decay rate	0.7
learning rate decay step	200,000
initial Batch Normalisation momentum	0.5
maximum Batch Normalisation momentum	0.99
Batch Normalisation momentum decay rate	0.5
Batch Normalisation momentum decay step	200,000

Table 5.5: Hyper-parameters of PointNet used in the experiments (please refer to Section 3.1.1 of a detailed description of these parameters)

5.4 Experimental Results

In this section, the results of the experimental work are presented. All the experiments are conducted based on PointNet DNN architecture developed by (Qi, Su, et al., 2017) in his Github repository ¹. The applied PointNet will be in its original architecture and with most of its original parameters as shown in Table 5.5.

All the experiments were run using the hardware described in Table 3.3. The results were summarised using the multi-class accuracy described in Section 3.4. The experimental results are analysed using the median accuracy value of 10 independent learning trials, and in some cases displayed using box-plots (Section 3.4). The statistical significance of the difference in the results obtained in different experiments is evaluated using pairwise Mann-Whitney tests at a 1% level of significance. The next three sections present the results obtained respectively the APS and YCB, MEch12, and ModelNet40 model sets.

5.4.1 Artificial Primitive Shapes and YCB-28

PointNet was trained using the CEST algorithm using the APS-error and APS-all model sets, and tested on the YCB-28 model set. The hyper-parameters of the CEST algorithm were experimentally set as reported in Tables 5.6 and 5.7.

As discussed in Section 4.3.1, the Adam algorithm trained PointNet employing

¹<https://github.com/charlesq34/pointnet>

Name	Symbol	Value
number of genes	N_g	591
genes initial fitness	f_{gene}^{init}	1.0
weight of fitness trace	w_o	0.8
weight of current fitness	w_c	0.2
history span	L_h	6
population size	N_p	16
number of children	N_c	80
elite size	N_e	4
<i>forget</i> operator threshold	Θ	0.1
mutation rate for individuals	m_i	0.5
mutation rate for genes	m_g	0.7
learning cycles	E	10

Table 5.6: Setting of [CEST](#) hyper-parameters used for training PoinNet using the [APS-error](#) model set and testing its performance on the [YCB](#) model set

Name	Symbol	Value
number of genes	N_g	1,742
genes initial fitness	f_{gene}^{init}	1.0
weight of fitness trace	w_o	0.8
weight of current fitness	w_c	0.2
history span	L_h	10
population size	N_p	8
number of children	N_c	40
elite size	N_e	2
<i>forget</i> operator threshold	Θ	0.1
mutation rate for individuals	m_i	0.5
mutation rate for genes	m_g	0.7
learning cycles	E	10

Table 5.7: Setting of [CEST](#) hyper-parameters used for training PoinNet using the [APS-all](#) model set and testing its performance on the [YCB](#) model set

a batch-size of 100 for 160 training epochs. When the size of the training set was not a multiple of 100, the number of training examples in the last batch size was brought to 100 using the procedure described in Section 4.3.1. Accordingly, the *APS-error* set (591 *PCs*) was fed to the *Adam* optimiser in 6 batches of 100 examples each (the last containing 9 duplicates), for a total of $6 \times 100 \times 160 = 96,000$ shapes during the 160 epochs of the learning trial. Likewise, the *APS-all* set (1742 *PCs*) was fed to the *Adam* optimiser in 18 batches of 100 examples each (the last containing 58 duplicates), for a total of $18 \times 100 \times 160 = 288,000$ shapes in the 160 training epochs.

The learning results obtained using *CEST* were compared with those obtained using the standard *Adam* optimiser. The latter are described in detail in Section 4.3.2. The results are detailed in Table 5.8 and visualised in Figure 5.3. Table 5.9 reports the p-values of pairwise Mann-Whitney tests performed on the accuracy results obtained using the various combinations of training algorithms and model sets.

	Training Set			
	<i>APS-error</i>	<i>APS-error</i>	<i>APS-all</i>	<i>APS-all</i>
Training Algorithm	<i>Adam</i>	<i>CEST</i>	<i>Adam</i>	<i>CEST</i>
Training Epochs	160	10	160	10
Fed Examples	96,000	54,950	288,000	79,900
Fed Examples (%)	100%	57.24%	100.0%	27.74%
Training Time (minutes under RTX-3090)	3.34	2.02	9.86	2.83
Median Accuracy	85.98%	82.85%	84.02%	87.59%
Interquartile Range (IQR)	2.59%	8.26%	4.29%	6.25%

Table 5.8: Experimental results obtained training PointNet on different model sets, and testing its accuracy on the *YCB-28* set

Although *CEST* did not improve in statistically significant fashion the results obtained using the *Adam* optimiser, it obtained a clearly superior median accuracy. The most noticeable result is the reduction in the number of *PCs* (fed shapes) employed in the training procedure. Despite training PointNet using a population of *prey*, where several *PCs* are potentially duplicated, *CEST* significantly reduced the sampling of the training set, whilst obtaining accuracy results at least comparable to those obtained by the *Adam* optimiser. This result has a major impact

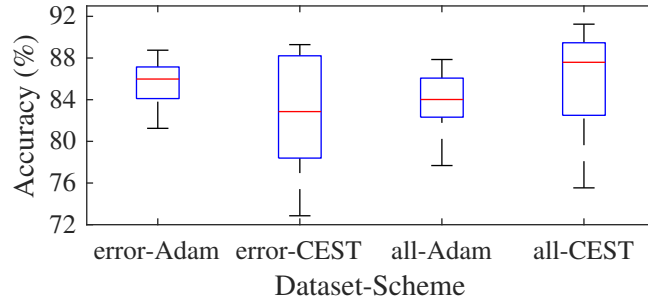


Figure 5.3: Box-plot of the classification results obtained training PointNet on different model sets, and testing its accuracy on the *YCB-28* set

	error-Adam	error-CEST	all-Adam	all-CEST
error-Adam	-	0.241	0.257	0.385
error-CEST	0.241	-	0.650	0.151
all-Adam	0.257	0.650	-	0.131
all-CEST	0.385	0.151	0.131	-

Table 5.9: Matrix of p-values of pairwise Mann-Whitney tests performed on the learning results of different combinations of training sets and algorithms. The medians of the accuracy results are reported in Table 5.8

on the execution time of the training procedure. [Castellani \(2018\)](#) estimated the sampling of the training set to be roughly proportional to the execution time. The results in Table 5.8 roughly confirm this estimate.

5.4.2 Effect of the Parameterisation on CEST Performance

To understand the effect on the performance of the parameterisation of *CEST*, and to better understand the functioning of the algorithm, a number of further tests was carried out. For this purpose, the main hyper-parameters of *CEST* were grouped into *population parameters*, *history fitness parameters*, and *mutation rates*, and within each group different combinations of parameters were tested. The three groups of parameters are shown in Table 5.10. All the parameterisation experiments used *APS-all* for training, and *YCB-28* for performance testing. Each experiment was repeated 10 times, and compared with the results obtained using the original *Adam* training algorithm as in Table 5.8.

Population	History Fitness	Mutation
N_p	L_h	m_i
N_c	w_o	m_g
N_e	w_c	

Table 5.10: The three groups of parameters in CEST

Population Parameters

The total number of PCs employed in a learning trial is equal to the population size, i.e. the number of training subsets that are fed to the classifier at each training cycle, times the complexity of the individuals, i.e. the size of the training subsets. The latter parameter will be highest at the beginning of the coevolutionary process, since the initial population is seeded to include all the samples of the training set (full initialisation scheme).

Experimentally (Table 5.8 and Table 5.9), the best performing combination of parameters featured a total number of offspring per learning cycle equal to 40 (five times the population size), and the elite size was set to $N_e = 2$ (one quarter of the population size). The optimal number of training epochs was empirically determined as: $E = 80 \div N_p$. It is important to notice that the number of training epochs is inversely proportional to the population size, keeping the overall time complexity constant. The trade-off is therefore whether to perform the evolutionary process with a small population for many iterations, or a large population for a few iterations. Finally, the history length was determined as 60% of the duration of the algorithm: $L_h = 0.6 \times E$.

Name	N_p	N_c	N_e	L_h	E	Fed Examples	Accuracy	IQR	P-value
P4-R1-E20	4	20	1	12	20	26.22%	84.11%	3.13%	0.850
P6-R1-E15	6	30	1	9	15	30.42%	85.71%	7.68%	0.791
P6-R2-E15	6	30	2	9	15	30.61%	80.80%	10.45%	0.496
P8-R2-E10	8	40	2	6	10	27.74%	87.59%	6.25%	0.131
P10-R2-E8	10	50	2	5	8	29.17%	83.30%	8.30%	0.821
P10-R3-E8	10	50	3	5	8	29.18%	82.95%	14.02%	0.910

Table 5.11: Experimental design and results for varying population parameters, where the p-value was calculated against results made by PointNet trained with APS-all using Adam algorithm as shown in Table 5.8

A new set of 6 different **CEST** parameterisations was tested, keeping the above empirically determined ratios. For each parameterisation, 10 independent learning trials were performed. The results of the tests are reported in Table 5.11: for each setting they include the median of the 10 obtained classification accuracies, the spread of the classification accuracies (upper minus lower quartile), and the median of the number of sampled **PCs**. Additionally, the total number of used **PCs** per training epoch is plotted in Figure 5.4.

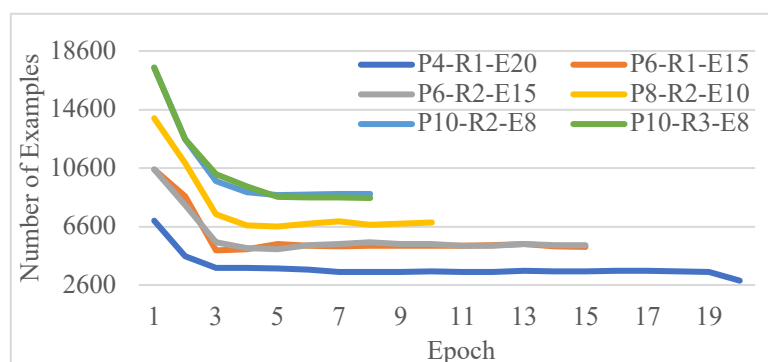


Figure 5.4: Plots of number of **PCs** used per epoch to train and evaluate PointNet, in relation to different settings of population parameters

The experiments confirm the optimality of the settings previously found. The fact that the population size is inversely proportional to the number of training epochs effectively constrains the exploration capability of the algorithm. To support exploration, high mutation rates could be used. This scenario will be tested in the following of this section.

Finally, the plots in Figure 5.4 clearly show that **CEST** quickly reduces the usage of the training examples. Given the high classification accuracies obtained, this result suggests that **CEST** is able to quickly narrow down the training process to a small number of difficult scenes.

Mutation Parameters

The sampling opportunities for the **CEST** algorithm are severely constrained due to the need of reducing as much as possible the number of used scenes. It is thus important to understand the consequences of the exploration-exploitation trade-off (Berger-Tal et al., 2014) on the quality of the solutions. The mutation operator plays a major role in addressing this trade-off, and its role will be investigated.

The population parameters were fixed to the four best performing settings from Table 5.11, that is P4-R1-E20, P6-R1-E15, P8-R2-E10, and P10-R2-E8. The proposed investigation will focus on the mutation rates m_i and m_g , testing a number

Name	N_p	m_i	m_g	Fed Examples	Accuracy	IQR	P-value
P4-I3-G3	4	0.3	0.3	29.10%	82.50%	2.59%	0.064
P4-I3-G5	4	0.3	0.5	27.24%	85.80%	8.53%	0.162
P4-I5-G3	4	0.5	0.3	27.93%	87.05%	9.87%	0.521
P4-I5-G5	4	0.5	0.5	27.07%	83.84%	7.05%	0.850
P4-I5-G7	4	0.5	0.7	26.22%	84.11%	3.13%	0.850
P6-I5-G5	6	0.5	0.5	30.94%	83.57%	7.32%	0.791
P6-I7-G5	6	0.7	0.5	29.79%	81.25%	5.85%	0.427
P6-I5-G7	6	0.5	0.7	30.42%	85.71%	7.68%	0.791
P6-I7-G7	6	0.7	0.7	29.36%	74.64%	8.75%	0.096
P8-I5-G5	8	0.5	0.5	30.73%	82.95%	5.62%	0.199
P8-I7-G5	8	0.7	0.5	28.87%	82.59%	5.00%	0.186
P8-I5-G7	8	0.5	0.7	27.74%	87.59%	6.25%	0.131
P8-I7-G7	8	0.7	0.7	26.75%	84.29%	4.33%	0.345
P10-I5-G7	10	0.5	0.7	29.17%	83.30%	8.30%	0.821
P10-I7-G7	10	0.7	0.7	27.14%	85.71%	4.60%	0.850
P10-I5-G9	10	0.5	0.9	25.64%	81.43%	9.82%	0.496

Table 5.12: Experimental design and results for varying mutation parameters, where p-value was calculated against results of error-Adam Table 5.8

Name	N_p	M_i	M_g	Fed Examples	Accuracy	IQR	P-value
P4-I1-G1	4	0.1	0.1	38.77%	88.66%	5.67%	0.082
P4-I1-G3	4	0.1	0.3	36.42%	85.36%	5.71%	0.571
P4-I3-G1	4	0.3	0.1	34.70%	84.29%	7.81%	0.678
P4-I3-G3	4	0.3	0.3	30.02%	82.50%	4.06%	0.496
P2-I05-G05	2	0.05	0.05	36.88%	85.45%	12.95%	0.597
P2-I05-G1	2	0.05	0.1	36.75%	85.09%	4.11%	0.364
P2-I1-G05	2	0.1	0.05	36.56%	86.43%	11.38%	0.597
P2-I1-G1	2	0.1	0.1	35.02%	87.14%	4.20%	0.049
P2-I3-G3	2	0.3	0.3	27.92%	78.93%	8.21%	0.089

Table 5.13: Experimental results for low mutation rates and small population sizes

of combinations of the parameters as shown in Table 5.12. The statistics were calculated over 10 independent learning trials. The plots of the number of PCs utilised per epoch for training and testing PointNet are shown in Figure 5.5 for different combinations of parameters.

The most noticeable effect in the plots of Figure 5.5 is that a high mutation rate does speed up the convergence of the evolutionary process towards a small subset of PC samples. This setting should help when the chosen strategy is to perform a small number of training cycles with a large population. Vice versa, when a small population was used and let to evolve for several cycles, the best results were obtained using a medium or low mutation rate.

One more set of experiments was performed further decreasing the mutation rate, and keeping a small ($N_p = 4$, $N_c = 20$, $N_r = 1$, and $E = 20$) or very small ($N_p = 2$, $N_c = 20$, $N_r = 1$, and $E = 30$) population size. The results of these last experiments are detailed in Table 5.13 and visualised in Figure 5.6. The statistics were calculated over 10 independent learning trials, and the p-values were calculated against the results obtained by PointNet when trained on the *APS-all* set using the Adam algorithm (Table 5.8). In this latter case, the mutation rate was too small to play a major part in the reduction of the number of training examples, and the sharp late drop in the plots is due to the action of the *Forget* operator.

Due to the late drop in the number of utilised training patterns, in many of the tests in Figure 5.6, the CEST algorithm used a total number of training examples slightly superior to the previous experiments. However, the number of fed examples is still very low, roughly one third of the examples used by the Adam optimiser. Despite using less examples, the CEST algorithm can still train PointNet to reach a classification accuracy comparable to the accuracy obtained using the Adam optimiser.

Fitness Weights

This set of experiments evaluated the impact of varying the contribution of the last fitness evaluation to the determination of an individual overall fitness. Namely, the effect of varying the w_c and w_o parameters in Equation (5.2.1) is the object of the experiments.

The CEST algorithm was run using various settings which could be summarised as: medium level mutation rates for large populations (P8-R2-I5-G5-E10 and P10-R2-I6-G6-E8), and low level mutation rates for small populations (P4-R1-I1-G1-E20 and P2-R1-I1-G1-E40) as shown in Table 5.14. The settings are slightly different from those used in the previous experiments, to allow for slightly higher use of the examples in the training set. The parameterisation of CEST and the experimental results are detailed in Table 5.15 and visualised in

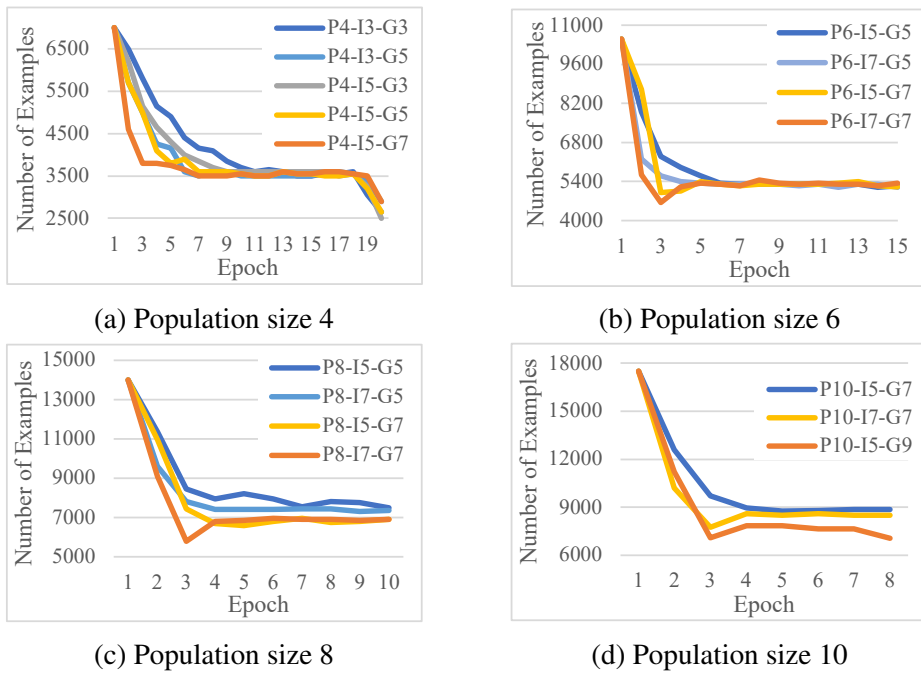


Figure 5.5: Plots of training samples utilised per training epoch for different mutation rates

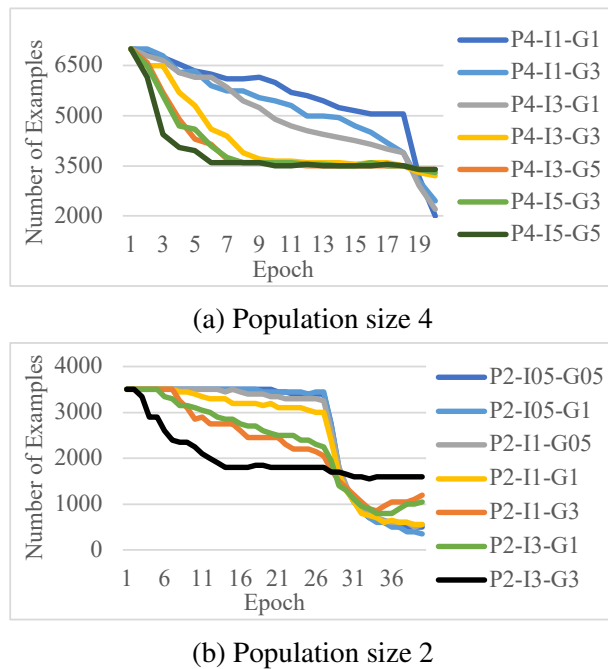


Figure 5.6: Number of employed training patterns per training epoch in experiments with small populations and low mutation rates

Name	N_p	N_c	N_r	m_i	m_g	E
P10-R2-I6-G6-E8	10	50	2	0.6	0.6	8
P8-R2-I5-G5-E10	8	40	2	0.5	0.5	10
P4-R1-I1-G1-E20	4	20	1	0.1	0.1	20
P2-R1-I1-G1-E40	2	20	1	0.1	0.1	40

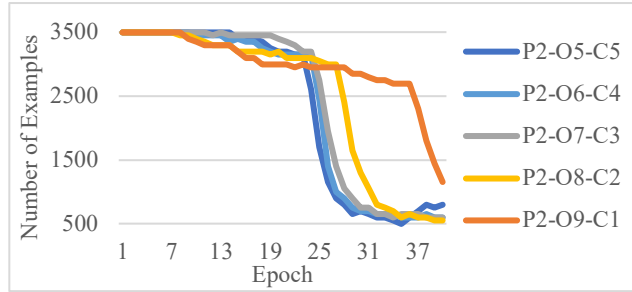
Table 5.14: Basic hyper-parameters of experiments studying fitness weights

Figure 5.7. The statistics were calculated over 10 independent learning trials, and the p-values were calculated against results made by PointNet trained with *APs-all* using Adam algorithm as shown in Table 5.8. The plots indicate that when more emphasis is placed on the current fitness evaluation, *CEST* is able to react more quickly to the learning results, and the *Forget* operator is triggered earlier.

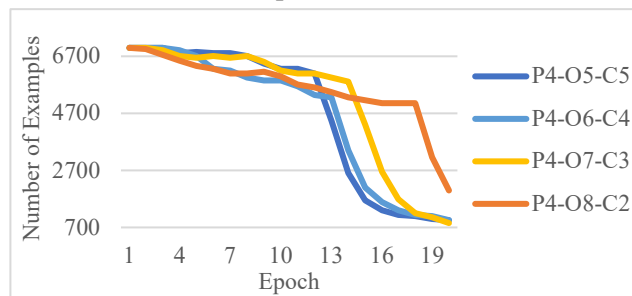
Overall, the results confirm that the *CEST* algorithm is capable of training the PointNet *DNN* to achieve a comparable or even significantly better classification accuracy than the *Adam* optimiser, using less training examples. The best results in terms of accuracy and consistency of the learning procedure, were obtained using low population sizes, and values of w_c and w_o close to the default values indicated in Section 5.2.

Results Achieved by Tuned *CEST* over APS and YCB-28 Sets

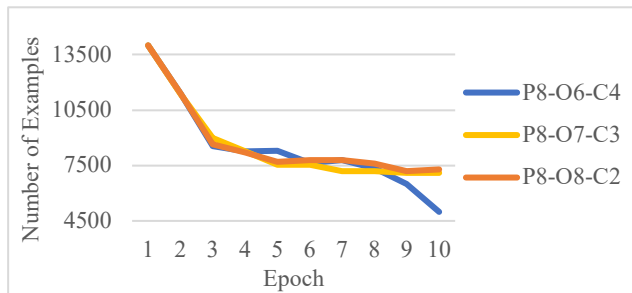
This set of experiments are conducted based on expertise gained from the parametrisation experiments above, and trail and error. A comparison of the best experimental results of PointNet trained with both *CEST* and *Adam* are illustrated in Table 5.16, and the obtained parameters for *CEST* are shown in Table 5.16.



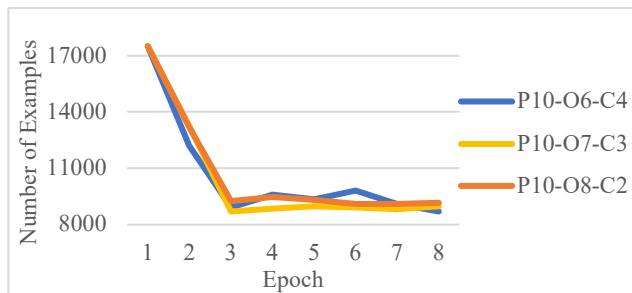
(a) Population size 2



(b) Population size 4



(c) Population size 8



(d) Population size 10

Figure 5.7: Plots of number of training scenes used per epoch for different settings of the fitness weights

Name	N_p	L_h	w_o	w_c	Fed Examples	Accuracy	IQR	P-value
P2-O5-C5	2	24	0.5	0.5	32.48%	83.39%	5.80%	0.910
P2-O6-C4	2	24	0.6	0.4	32.80%	87.05%	5.98%	0.226
P2-O7-C3	2	24	0.7	0.3	33.96%	88.21%	1.52%	0.001
P2-O8-C2	2	24	0.8	0.2	35.02%	87.14%	4.20%	0.049
P2-O9-C1	2	24	0.9	0.1	41.27%	85.54%	7.72%	0.326
P4-O5-C5	4	12	0.5	0.5	32.71%	86.70%	5.58%	0.257
P4-O6-C4	4	12	0.6	0.4	32.14%	83.30%	10.18%	1.000
P4-O7-C3	4	12	0.7	0.3	35.66%	83.57%	6.56%	0.791
P4-O8-C2	4	12	0.8	0.2	38.77%	88.66%	5.67%	0.082
P8-O6-C4	8	6	0.6	0.4	29.44%	88.39%	9.11%	0.151
P8-O7-C3	8	6	0.7	0.3	28.97%	87.68%	3.17%	0.034
P8-O8-C2	8	6	0.8	0.2	30.43%	85.63%	7.32%	0.257
P10-O6-C4	10	5	0.6	0.4	29.57%	78.13%	9.46%	0.450
P10-O7-C3	10	5	0.7	0.3	29.13%	85.98%	6.29%	0.385
P10-O8-C2	10	5	0.8	0.2	29.88%	82.14%	9.87%	0.226

Table 5.15: Experimental design and results for fitness weights

	Training Set			
	APS-error	APS-error	APS-all	APS-all
Training Algorithm	<i>Adam</i>	CEST	<i>Adam</i>	CEST
Training Epochs	160	40	160	40
Fed Examples	96,000	75,750	288,000	97,800
Fed Examples (%)	100%	78.91%	100.0%	33.96%
Training Time (minutes under RTX-3090)	3.34	2.69	9.86	3.41
Median Accuracy	85.98%	89.20%	84.02%	88.21%
IQR	2.59%	2.50%	4.29%	1.52%
P-value	0.019		0.001	

Table 5.16: The best-so-far experimental results obtained training PointNet on *APS-error* and *APS-all* sets, and testing on the *YCB-28* set

Model Set	N_p	N_c	N_r	m_i	m_g	L_h	Θ	w_o	w_c	E
<i>APS-error</i>	4	20	2	0.05	0.05	24	0.1	0.8	0.2	40
<i>APS-all</i>	2	20	1	0.1	0.1	24	0.1	0.7	0.3	40

Table 5.17: Best-tuned **CEST** parameters setting for *APS* sets

As shown in Table 5.16, the **CEST** algorithm was able to largely reduce the number of model samples used in the training procedure, and consequently executed in much faster times. A significantly better classification accuracy ($p \leq 0.01$) was achieved on the *APS-all* set compared to the network trained using the standard *Adam* optimiser. On the *APS-error* set the improvement in performance was noticeable but not significant at a 1% level of confidence. **CEST** also gave more consistent learning results (smaller **IQR**) than *Adam* on the *APS-all* set, and a comparable spread of the results on the *APS-error* set.

5.4.3 MEch12 and ModelNet40

In this last set of experiments, the **CEST** algorithm was tested on the *MEch12* and *ModelNet40* model sets. As detailed in Section 5.3, these two sets are more complex than *YCB-28* in terms number of classes and set size. The **CEST** parameters were set based on the expertise gained in the previous tests, and trial and error.

Model Set	N_p	N_c	N_r	m_i	m_g	L_h	Θ	w_o	w_c	E
MEch12	16	48	1	0.6	0.6	12	0.125	0.8	0.2	20
ModelNet40	16	48	1	0.6	0.6	12	0.1	0.8	0.2	20

Table 5.18: **CEST** parameters setting for the experiments on the *MEch12* and *ModelNet40* model sets

The accuracy results obtained using **CEST** and the standard *Adam* optimiser are detailed in Table 5.19, and visualised Figure 5.8, whilst the used **CEST** parameters are illustrated in Table 5.18. In both sets of experiments, the PointNets will use parameters shown in Table 5.5.

The experiments reported in Chapter 3 indicated that a high classification accuracy (94%) was attainable on the *MEch12* set. On *ModelNet40*, a preliminary test was performed reproducing the experimental settings reported by Qi, Su, et al. (2017). The test confirmed that, trained using the *Adam* optimiser, PointNet is able to reach a median accuracy of 88% circa.

Model Set	Algorithm	Fed Examples	Time (minutes)	Accuracy	P-value
MEch12	<i>Adam</i>	480,000 (100%)	~16	94.16%	0.5453
MEch12	CEST	419,500 (87.40%)	~14	95.00%	
ModelNet40	<i>Adam</i>	2,460,000 (100%)	~89	88.37%	0.9097
ModelNet40	CEST	1,658,688 (67.32%)	~62	88.43%	

Table 5.19: Results obtained by training PointNet using the **CEST** and *Adam* optimisers, on the MEch12 and ModelNet40 model sets; where **Time** is the measured real execution time of one training epoch under RTX-3090 GPU

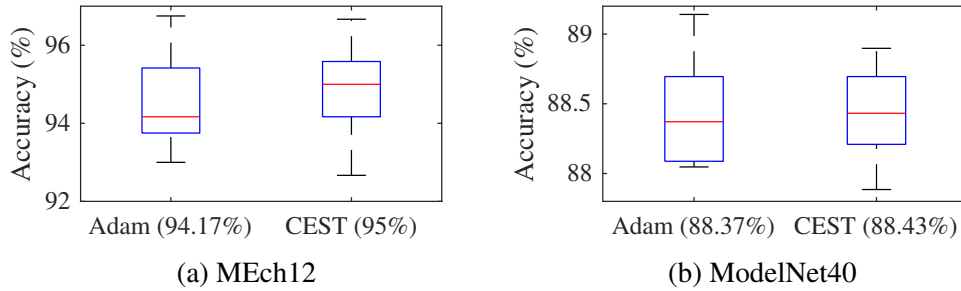


Figure 5.8: Box-plot of the classification accuracies obtained on the MEch12 and ModelNet40 sets with Adam and **CEST** learning algorithm, whilst median accuracies are shown within parentheses

On both model sets, **CEST** obtained statistically indistinguishable results from those obtained by *Adam*. However, **CEST** used less training set samples, the difference between the two algorithms being particularly noticeable for the tests involving the largest ModelNet40 set (saving one third of training time), as shown in Table 5.19.

In conclusion, the **CEST** algorithm has the potential to increase the training speed of **DNN** without affecting its original performance by evolving the subsets of training data.

5.5 Discussion

The main research question in this study was whether the computational overheads of training a complex **DNN** on a set of 3D models could be reduced via coevolutionary learning. One of the primary reasons for the complexity of the

task, is the large amount of **PCs** that needs to be processed by the **DNN** structure. The current availability of cheap 3D scanners is creating an increasing need to deal with large sets of **PCs**.

The envisaged answer to the above research question was **CEST**, a predator-prey system where a standard gradient-based optimisation algorithm (*Adam*) was fed training examples selected by an **EA**. The goal of the **EA** was to pick the most difficult examples of the training set, namely those which hadn't been yet learned by **DNN**. By focusing the learning process on a small subset of critical examples, **CEST** was able to obtain top learning accuracies whilst reducing the computational overheads and hence the time of the training procedure. The answer to the research question was therefore positive.

In terms of classification accuracy, **CEST** obtained superior and more consistent learning accuracies. This result is likely due to the ability of **CEST** to focus the learning process on problematic examples, which towards the end drive the training procedure.

Although **CEST** gives emphasis during the training process to a limited subset examples, its performance was found to be robust to error in the training patterns.

On the imbalanced ModelNet40 model set, the classification accuracy attained training PointNet via **CEST** was undistinguishable from the accuracy obtained using the standard *Adam* optimiser. In terms of accuracy, the coevolutionary approach didn't seem to give any advantage respect to the standard training method on the unbalanced model set. However, **CEST** employed only about two-thirds of the training examples used by *Adam*. Further tests should evaluate whether increasing the sampling of the training set (the prey population size or the number of evolution cycles), the performance of **CEST** can be improved beyond the current accuracy.

Investigation of the effects on the learning process of **CEST** parameterisation suggested that very small prey population sizes can still provide acceptable results, when the mutation rate is increased to support the exploration capabilities of the algorithm. Overall, **CEST** reacted predictably to changes in the parameter settings. If confirmed by further studies, this latter feature would make **CEST** easy to optimise to the desired problem domain.

5.6 Conclusions

In this chapter, the novel **CEST** algorithm was presented. Experimental evidence showed that **CEST** was able to attain top classification accuracies, whilst reducing the use of the training set of examples. These savings in training examples implied a reduction in the computational overheads from running the procedure, and hence a reduction in execution times. A number of tests were also performed to in-

investigate the effects of the parameterisation of **CEST** on the evolutionary learning process, and yielded some improvements in performance.

The proposed algorithm is expected to increase the applicability and affordability of **DNN** systems, with particular regard to computationally intensive machine vision applications.

Chapter 6

Conclusions

The work of this thesis investigated the automatic identification of objects in **PC** models, with a view on its application to visual inspection of **EOL** mechanical devices for disassembly. A system based on the PointNet **DNN** was proposed. To reduce the effort of gathering a large set of scans of objects for the **DNN** training procedure, a 3D model generation scheme was proposed. This scheme was based on 3D **CAD** models, and a purpose-built depth camera simulator. A first set of experiments based on artificial model sets indicated that PointNet is able to recognise with good accuracy complex mechanical parts. The ability of the **DNN** to deal with partial views and noisy sensors was tested. The study revealed some limitation of PointNet in dealing with error in the scans. The extent of this limitation could be greatly reduced by injecting the training patterns with some level of local noise.

A second set of experiments investigated the ability of PointNet to recognise primitive shapes in objects. This work has important applications in the field of robotic pick-up and handling. The performance of PointNet was tested on three artificial model sets, and the **YCB** benchmark model set featuring scans of real-life everyday objects. The results suggest that PointNet can be trained on geometric or **CAD**-generated models, and recognise with good accuracy the real-life **YCB** scans. The performances obtained by PointNet were equal or better than those obtained using a **SNN** classifier, justifying thus the use of the more complex deep architecture. The experimental study showed also the effect of the parameterisation of the learning procedure on the accuracy of the **DNN** classifier. Finally, the tests revealed that, also in this case, the performance of PointNet can be improved by injecting some local noise in the training **PCs**.

Finally, the last set of experiments demonstrated the accuracy of the novel **CEST** coevolutionary procedure, and its ability to quickly narrow down the training procedure to a subset of mainly not yet learned examples. **CEST** addressed the problem of the computational complexity of **DNN** training algorithms, which

makes their implementation reliant on powerful and fast hardware. The work has potential impact on the whole [DNN](#) field.

6.1 Summary of Achievements

The presented work was done by the author of this thesis for the degree of Doctor of Philosophy. The following scientific contributions have been made:

- the feasibility of applying the state-of-the-art PointNet [DNN](#) to the recognition of complex mechanical parts was proven. The system was able to distinguish the different parts of an engine turbocharger, as well as similar parts of different maker;
- a novel training data generation scheme based on 3D [CAD](#) models and a purpose-built 3D camera simulator was created;
- the possibility of training PointNet to recognise primitive shapes in real life scans of everyday objects using models created from geometrical shapes or [CAD](#) models, was proven;
- the ability of PointNet to recognise objects and shapes from imprecise scans and partial views was evaluated;
- the novel coevolutionary [CEST](#) algorithm was proposed. [CEST](#) sped up the training procedure of PointNet by focusing it on the most difficult training examples;
- the performance of [CEST](#) was tested on several benchmark model sets of various origin, complexity, and size. Despite using less the instances of the training set, [CEST](#) excelled in terms of accuracy and consistency of the learning results.

6.2 Limitations and Future Work

This thesis addressed the challenge of building an automatic identification system for robotic disassembly in remanufacturing. Various problems ranging from the cost of acquiring training data, to the computational complexity of the training procedure were addressed. The study included an in-depth evaluation of the ability of PointNet to cope with imprecision in the scans and partial view of the objects. This evaluation was made on a combination of custom-made model sets of simulated scenes, and real scans of everyday (non mechanical) objects.

Due to the lack of access to the robotic lab during the pandemic, a detailed evaluation on real scans of mechanical objects could not be performed. For the experiments of Chapter 4, the YCB dataset was used as benchmark for the shape approximation challenge (Kopicki et al., 2016; Mavrakis et al., 2016). Although the results of the performed tests were promising, the proposed system has not been tested yet on lab conditions that closely reproduce a real-life industrial scenario. Further work should fill this gap, extending the experiments to scans acquired from industrial-grade 3D cameras over real mechanical parts.

Also, the proposed work was conceived in 2018 when PointNet was the state-of-the-art architecture for processing 3D scenes. There are now more DNN architectures which showed higher classification accuracies than PointNet on benchmark model sets. The proposed study should be extended to the new DNN architectures that emerged in the last three years.

Chapter 5 presented the new CEST algorithm for DNN training. Tests performed during the development of the algorithm suggest that the *Forget* operator plays a major role in the convergence of the algorithm to the set of most difficult training examples. Further work should be carried out to investigate when a pattern can be safely discarded, and optimal ways of taking into account of the history of the pattern's fitness evaluations.

Finally, further work should be done to investigate the performance and generalisation of CEST on other data sets and problem domains.

6.3 Publications Arisen from this Thesis

Part of the work described in this thesis has already been presented in conferences, and resulted in the following publications:

- Zheng, S., Lan, F., Baronti, L., and Castellani, M., 2018. "Automatic Identification of Mechanical Parts for Robotic Disassembly Using Deep Neural Networks." II International Workshop on Autonomous Remanufacturing (IWAR) Wuhan, China. (DISTINGUISHED INDUSTRIAL APPLICATION PAPER AWARD)
- Zheng, S. and Castellani, M., 2019. "Identification of mechanical parts for robotic disassembly from point cloud scenes." III International Workshop on Autonomous Remanufacturing (IWAR) Albacete, Spain.
- Zheng, S., Lan, F., Baronti, L., Pham, D., and Castellani, M., 2022. "Automatic identification of mechanical parts for robotic disassembly using the PointNet deep neural network." International Journal of Manufacturing Research (IJMR), Volume 17, Issue 1, Page 1-21.

- Zheng, S., Castellani, M., 2021. “Primitive Shape Recognition from Real-Life Scenes Using the PointNet Deep Neural Network”, Submitted to The International Journal of Advanced Manufacturing Technology (IJAMT).

Appendix A

Manually Measured Shape Features

ID-Name	Shape-Type	H	W	B	D
003-cracker box	box	216.23	149.82	57.04	-
004-sugar box	box	162.50	82.58	32.32	-
008-pudding box	box	97.80	79.34	29.38	-
009-gelatin box	box	73.70	65.62	23.73	-
010-potted meat can	box	80.16	76.26	39.73	-
026-sponge	box	95.09	60.03	14.20	-
036-wood block	box	190.35	80.52	75.81	-
061-foam brick	box	63.68	54.29	43.36	-
077-rubiks cube	box	57.35	48.56	46.12	-
001-chips can	cylinder	237.27	-	-	69.33
002-master chef can	cylinder	129.79	-	-	96.09
005-tomato soup can	cylinder	92.34	-	-	64.63
007-tuna fish can	cylinder	38.44	-	-	80.25
019-pitcher base	cylinder	226.75	-	-	128.13
025-mug	cylinder	101.24	-	-	84.78
040-large marker	cylinder	103.09	-	-	14.81
065-a-cups	cylinder	57.36	-	-	43.53
012-strawberry	sphere	-	-	-	48.52
014-lemon	sphere	-	-	-	63.59
015-peach	sphere	-	-	-	73.05
017-orange	sphere	-	-	-	74.81
018-plum	sphere	-	-	-	52.29
054-softball	sphere	-	-	-	97.03
055-baseball	sphere	-	-	-	75.46
056-tennis ball	sphere	-	-	-	69.18
057-racquetball	sphere	-	-	-	60.41
058-golf ball	sphere	-	-	-	47.36
063-a-marble	sphere	-	-	-	39.92

Table A.1: Manually measured shape features of the twenty-eight selected objects from [YCB](#) set

References

- Andersen, T., Stanley, K. O., & Miikkulainen, R. (2002). *Neuro-evolution through augmenting topologies applied to evolving neural networks to play othello*. Citeseer.
- Andina, D., & Pham, D. T. (2007). *Computational intelligence: For engineering and manufacturing*. Springer.
- Azzini, A., & Tettamanzi, A. G. (2011). Evolutionary anns: a state of the art survey. *Intelligenza Artificiale*, 5(1), 19–35.
- Baronti, L., Alston, M., Mavrakis, N., Ghalamzan, E., Amir, M., & Castellani, M. (2019). Primitive shape fitting in point clouds using the bees algorithm. *Applied Sciences*, 9(23), 5198.
- Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404–417).
- Bdiwi, M., Rashid, A., & Putz, M. (2016). Autonomous disassembly of electric vehicle motors based on robot cognition. In *2016 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 2500–2505).
- Beis, J. S., & Lowe, D. G. (1997). Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 1000–1006).
- Belew, R., McInerney, J., & Schraudolph, N. (1991). *Evolving networks: using genetic algorithms with connectionist learning in proceedings of second artificial life conference*, 511–547. New York: Addison-Wesley.
- Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, 24(4), 509–522.

- Berger-Tal, O., Nathan, J., Meron, E., & Saltz, D. (2014). The exploration-exploitation dilemma: a multidisciplinary framework. *PloS one*, 9(4), e95693.
- Besl, P. J., & McKay, N. D. (1992). Method for registration of 3-d shapes. In *Sensor fusion iv: control paradigms and data structures* (Vol. 1611, pp. 586–606).
- Blum, A. L., & Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial intelligence*, 97(1-2), 245–271.
- Bohg, J., Romero, J., Herzog, A., & Schaal, S. (2014). Robot arm pose estimation through pixel-wise part classification. In *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 3143–3150).
- Börold, A., Teucke, M., Rust, J., & Freitag, M. (2020). Recognition of car parts in automotive supply chains by combining synthetically generated training data with classical and deep learning based image processing. *Procedia CIRP*, 93, 377–382.
- Brogan, D. P., DiFilippo, N. M., & Jouaneh, M. K. (2021). Deep learning computer vision for robotic disassembly and servicing applications. *Array*, 12, 100094.
- Broomhead, D. S., & Lowe, D. (1988). *Radial basis functions, multi-variable functional interpolation and adaptive networks* (Tech. Rep.). Malvern: Royal Signals and Radar Establishment Malvern (United Kingdom).
- Büker, U., Drüe, S., Götze, N., Hartmann, G., Kalkreuter, B., Stemmer, R., & Trapp, R. (2001). Vision-based control of an autonomous disassembly station. *Robotics and Autonomous Systems*, 35(3-4), 179–189.
- Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., & Dollar, A. M. (2015). The ycb object and model set: Towards common benchmarks for manipulation research. In *2015 International Conference on Advanced Robotics (ICAR)* (pp. 510–517).
- Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010). Brief: Binary robust independent elementary features. In *European conference on computer vision* (pp. 778–792).
- Castellani, M. (2006). Anne—a new algorithm for evolution of artificial neural network classifier systems. In *2006 IEEE International Conference on Evolutionary Computation* (pp. 3294–3301).

- Castellani, M. (2013). Evolutionary generation of neural network classifiers—an empirical comparison. *Neurocomputing*, 99, 214–229.
- Castellani, M. (2018). Competitive co-evolution of multi-layer perceptron classifiers. *Soft Computing*, 22(10), 3417–3432.
- Chellapilla, K., & Fogel, D. B. (1999). Evolving neural networks to play checkers without relying on expert knowledge. *IEEE transactions on neural networks*, 10(6), 1382–1391.
- Chua, C. S., & Jarvis, R. (1997). Point signatures: A new representation for 3d object recognition. *International Journal of Computer Vision*, 25(1), 63–85.
- Cortez, P., Rocha, M., & Neves, J. (2002). A lamarckian approach for neural network training. *Neural Processing Letters*, 15(2), 105–116.
- Curless, B., & Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques* (pp. 303–312).
- Dario, P., Rucci, M., Guadagnini, C., & Laschi, C. (1994). An investigation on a robot system for disassembly automation. In *Proceedings of the 1994 IEEE international conference on robotics and automation* (pp. 3515–3521).
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).
- DiFilippo, N. M., & Jouaneh, M. K. (2017). A system combining force and vision sensing for automated screw removal on laptops. *IEEE Transactions on Automation science and engineering*, 15(2), 887–895.
- Dominguez, M., Dhamdhere, R., Petkar, A., Jain, S., Sah, S., & Ptucha, R. (2018). General-purpose deep point cloud feature extractor. In *2018 IEEE winter conference on applications of computer vision (wacv)* (pp. 1972–1981).
- Elsken, T., Metzen, J. H., & Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1), 1997–2017.
- Floreano, D., Dürr, P., & Mattiussi, C. (2008). Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1), 47–62.
- Fogel, D. B. (2006). *Evolutionary computation: toward a new philosophy of machine intelligence* (Vol. 1). John Wiley & Sons.

- Fogel, D. B., Fogel, L. J., & Porto, V. (1990). Evolving neural networks. *Biological cybernetics*, 63(6), 487–493.
- Foo, G., Kara, S., & Pagnucco, M. (2021). Screw detection for disassembly of electronic waste using reasoning and re-training of a deep learning model. *Procedia CIRP*, 98, 666–671.
- Frome, A., Huber, D., Kolluri, R., Bülow, T., & Malik, J. (2004). Recognizing objects in range data using regional point descriptors. In *European conference on computer vision* (pp. 224–237).
- Garcia-Garcia, A., Orts-Escolano, S., Oprea, S., Villena-Martinez, V., & Garcia-Rodriguez, J. (2017). A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*.
- Gil, P., Pomares, J., Diaz, S. v. P. C., Candelas, F., & Torres, F. (2007). Flexible multi-sensorial system for automatic disassembly using cooperative robots. *International Journal of Computer Integrated Manufacturing*, 20(8), 757–772.
- Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine Learning*, 3(2-3), 95–98.
- Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*.
- Guo, Y., Bennamoun, M., Soheli, F., Lu, M., Wan, J., & Kwok, N. M. (2016). A comprehensive performance evaluation of 3d local feature descriptors. *International Journal of Computer Vision*, 116(1), 66–89.
- Han, X.-F., Jin, J. S., Xie, J., Wang, M.-J., & Jiang, W. (2018). A comprehensive review of 3d point cloud descriptors. *arXiv preprint arXiv:1802.02297*, 2.
- Harper, G., Sommerville, R., Kendrick, E., Driscoll, L., Slater, P., Stolkin, R., . . . others (2019). Recycling lithium-ion batteries from electric vehicles. *Nature*, 575(7781), 75–86.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770–778).
- Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3), 228–234.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5), 359–366.

- Hotelling, H. (1933). Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24(6), 417.
- Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).
- Huang, J., Pham, D. T., Wang, Y., Qu, M., Ji, C., Su, S., ... Zhou, Z. (2020). A case study in human–robot collaboration in the disassembly of press-fitted components. *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, 234(3), 654–664.
- Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning* (pp. 448–456).
- Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks. *arXiv preprint arXiv:1506.02025*.
- Johnson, A. E., & Hebert, M. (1998). Surface matching for object recognition in complex three-dimensional scenes. *Image and Vision Computing*, 16(9-10), 635–651.
- Johnson, A. E., & Hebert, M. (1999). Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on pattern analysis and machine intelligence*, 21(5), 433–449.
- Johnson, M. R., & McCarthy, I. P. (2014). Product recovery decisions within the context of extended producer responsibility. *Journal of Engineering and Technology Management*, 34, 9–28.
- Jovane, F., Alting, L., Armillotta, A., Eversheim, W., Feldmann, K., Seliger, G., & Roth, N. (1993). A key issue in product life cycle: disassembly. *CIRP annals*, 42(2), 651–658.
- Karlsson, B., & Järred, J.-O. (2000). Recycling of electrical motors by automatic disassembly. *Measurement science and technology*, 11(4), 350.
- Kazhdan, M., Bolitho, M., & Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the fourth eurographics symposium on geometry processing* (Vol. 7).
- Kin, S. T. M., Ong, S., & Nee, A. (2014). Remanufacturing process planning. *Procedia Cirp*, 15, 189–194.

- Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- Klokov, R., & Lempitsky, V. (2017). Escape from cells: Deep kd-networks for the recognition of 3d point cloud models. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 863–872).
- Knoth, R., Brandstotter, M., Kopacek, B., & Kopacek, P. (2002). Automated disassembly of electr (on) ic equipment. In *Conference record 2002 IEEE International Symposium on Electronics and the Environment (cat. no. 02ch37273)* (pp. 290–294).
- Kopacek, B., & Kopacek, P. (1999). Intelligent disassembly of electronic equipment. *Annual Reviews in Control*, 23, 165–170.
- Kopicki, M., Detry, R., Adjigble, M., Stolkin, R., Leonardis, A., & Wyatt, J. L. (2016). One-shot learning and generation of dexterous grasps for novel objects. *The International Journal of Robotics Research*, 35(8), 959–976.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 1097–1105.
- Krueger, J., Lehr, J., Schlueter, M., & Bischoff, N. (2019). Deep learning for part identification based on inherent features. *CIRP Annals*, 68(1), 9–12.
- LeCun, Y. (1998). The mnist database of handwritten digits. <http://yann.lecun.com/exdb/mnist/>.
- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, 521(7553), 436–444.
- Lee, S.-W. (1996). Off-line recognition of totally unconstrained handwritten numerals using multilayer cluster neural network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6), 648–652.
- Li, J., Barwood, M., & Rahimifard, S. (2018). Robotic disassembly for increased recovery of strategically important materials from electrical vehicles. *Robotics and Computer-Integrated Manufacturing*, 50, 203–212.
- Li, J., Chen, B. M., & Lee, G. H. (2018). So-net: Self-organizing network for point cloud analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 9397–9406).

- Li, M., Lian, S., Wang, F., Zhou, Y., Chen, B., Guan, L., & Wu, Y. (2019). prediction model of organic molecular absorption energies based on deep learning trained by chaos-enhanced accelerated evolutionary algorithm. *Scientific reports*, 9(1), 1–9.
- Li, R., Pham, D. T., Huang, J., Tan, Y., Qu, M., Wang, Y., ... others (2020). Unfastening of hexagonal headed screws by a collaborative robot. *IEEE Transactions on Automation Science and Engineering*, 17(3), 1455–1468.
- Li, X., Li, M., Wu, Y., Zhou, D., Liu, T., Hao, F., ... Ma, Q. (2021). Accurate screw detection method based on faster r-cnn and rotation edge similarity for automatic screw disassembly. *International Journal of Computer Integrated Manufacturing*, 34(11), 1177–1195.
- Li, Y., Bu, R., Sun, M., Wu, W., Di, X., & Chen, B. (2018). Pointcnn: Convolution on x-transformed points. *Advances in neural information processing systems*, 31, 820–830.
- Li, Y., Pirk, S., Su, H., Qi, C. R., & Guibas, L. J. (2016). Fpnn: Field probing neural networks for 3d data. *Advances in Neural Information Processing Systems*, 29, 307–315.
- Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740–755).
- Liu, Y., Fan, B., Xiang, S., & Pan, C. (2019). Relation-shape convolutional neural network for point cloud analysis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 8895–8904).
- Loshchilov, I., & Hutter, F. (2015). Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*.
- Lowe, D. G. (1999). Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision* (Vol. 2, pp. 1150–1157).
- Martín, A., Lara-Cabrera, R., Fuentes-Hurtado, F., Naranjo, V., & Camacho, D. (2018). Evodeep: a new evolutionary approach for automatic deep neural networks parametrisation. *Journal of Parallel and Distributed Computing*, 117, 180–191.
- Matei, B., Shan, Y., Sawhney, H. S., Tan, Y., Kumar, R., Huber, D., & Hebert, M. (2006). Rapid object indexing using locality sensitive hashing and joint

- 3d-signature space estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(7), 1111–1126.
- Maturana, D., & Scherer, S. (2015). Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (pp. 922–928).
- Mavrakis, N., Stolkin, R., Baronti, L., Kopicki, M., & Castellani, M. (2016). Analysis of the inertia and dynamics of grasped objects, for choosing optimal grasps to enable torque-efficient post-grasp manipulations. In *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)* (pp. 171–178).
- Montana, D. J., & Davis, L. (1989). Training feedforward neural networks using genetic algorithms. In *Ijcai* (Vol. 89, pp. 762–767).
- Packianather, M. S. (1997). *Design and optimisation of neural network classifiers for automatic visual inspection of wood veneer*. (Unpublished doctoral dissertation). University of Wales. Cardiff.
- Paredis, J. (1994). Classification neural networks. In *Artificial life iv: Proceedings of the fourth international workshop on the synthesis and simulation of living systems* (Vol. 4, p. 102).
- Paredis, J. (1995). Coevolutionary computation. *Artificial life*, 2(4), 355–375.
- Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11), 559–572.
- Pham, D., & Alcock, R. (1996). Automatic detection of defects on birch wood boards. *Proceedings of the Institution of Mechanical Engineers, Part E: Journal of Process Mechanical Engineering*, 210(1), 45–52.
- Pham, D., & Alcock, R. (1999). Recent developments in automated visual inspection of wood boards. In *Advances in manufacturing* (pp. 79–88). Springer.
- Pham, D., & Liu, X. (1995). *Neural networks for identification, prediction and control*.
- Qi, C. R., Su, H., Mo, K., & Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 652–660).

- Qi, C. R., Su, H., Nießner, M., Dai, A., Yan, M., & Guibas, L. J. (2016). Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 5648–5656).
- Qi, C. R., Yi, L., Su, H., & Guibas, L. J. (2017). Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems* (pp. 5099–5108).
- Qiu, S., Anwar, S., & Barnes, N. (2021a). Dense-resolution network for point cloud classification and segmentation. In *Proceedings of the ieee/cvf winter conference on applications of computer vision* (pp. 3813–3822).
- Qiu, S., Anwar, S., & Barnes, N. (2021b). Geometric back-projection network for point cloud classification. *IEEE Transactions on Multimedia*.
- Real, E., Aggarwal, A., Huang, Y., & Le, Q. V. (2019). Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence* (Vol. 33, pp. 4780–4789).
- Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., . . . Kurakin, A. (2017). Large-scale evolution of image classifiers. In *International conference on machine learning* (pp. 2902–2911).
- Rehnholm, J. (2021). *Battery pack part detection and disassembly verification using computer vision*.
- Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 91–99.
- Riegler, G., Osman Ulusoy, A., & Geiger, A. (2017). Octnet: Learning deep 3d representations at high resolutions. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 3577–3586).
- Rokach, L. (2010). Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2), 1–39.
- Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. In *European conference on computer vision* (pp. 430–443).
- Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. R. (2011). Orb: An efficient alternative to sift or surf. In *Iccv* (Vol. 11, p. 2).

- Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1985). *Learning internal representations by error propagation* (Tech. Rep.). California: California Univ San Diego La Jolla Inst for Cognitive Science.
- Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., ... Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3), 211–252.
- Rusu, R., & Cousins, S. (2017). 3d is here: point cloud library. *Point Cloud Library* <http://pointclouds.org/>. Accessed, 15.
- Rusu, R. B., Blodow, N., & Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation* (pp. 3212–3217).
- Rusu, R. B., Blodow, N., Marton, Z. C., & Beetz, M. (2008). Aligning point cloud views using persistent feature histograms. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3384–3391).
- Shi, B., Bai, S., Zhou, Z., & Bai, X. (2015). Deeppano: Deep panoramic representation for 3-d shape recognition. *IEEE Signal Processing Letters*, 22(12), 2339–2343.
- Simonovsky, M., & Komodakis, N. (2017). Dynamic edge-conditioned filters in convolutional neural networks on graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 3693–3702).
- Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- Su, H., Maji, S., Kalogerakis, E., & Learned-Miller, E. (2015). Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 945–953).
- Sun, Y., Xue, B., Zhang, M., & Yen, G. G. (2019). Evolving deep convolutional neural networks for image classification. *IEEE Transactions on Evolutionary Computation*, 24(2), 394–407.
- Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 1–9).
- Torres, F., Gil, P., Puente, S., Pomares, J., & Aracil, R. (2004). Automatic pc disassembly for component recovery. *The International Journal of Advanced Manufacturing Technology*, 23(1-2), 39–46.

- Torres, F., Puente, S., & Aracil, R. (2003). Disassembly planning based on precedence relations among assemblies. *The International Journal of Advanced Manufacturing Technology*, 21(5), 317–327.
- Uy, M. A., Pham, Q.-H., Hua, B.-S., Nguyen, T., & Yeung, S.-K. (2019). Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1588–1597).
- Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001* (Vol. 1, pp. I–I).
- Vischer, D. (1992). Cooperating robot with visual and tactile skills. In *Proceedings 1992 IEEE International Conference on Robotics and Automation* (pp. 2018–2019).
- Vongbunyong, S., & Chen, W. H. (2015). Disassembly automation. In *Disassembly automation* (pp. 25–54). Springer.
- Wang, P.-S., Liu, Y., Guo, Y.-X., Sun, C.-Y., & Tong, X. (2017). O-cnn: Octree-based convolutional neural networks for 3d shape analysis. *ACM Transactions On Graphics (TOG)*, 36(4), 1–11.
- Wang, Y., Sun, Y., Liu, Z., Sarma, S. E., Bronstein, M. M., & Solomon, J. M. (2019). Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5), 1–12.
- Watson, R. A., & Pollack, J. B. (2001). Coevolutionary dynamics in a minimal substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)* (pp. 702–709).
- Wegener, K., Andrew, S., Raatz, A., Dröder, K., & Herrmann, C. (2014). Disassembly of electric vehicle batteries using the example of the Audi Q5 hybrid system. *Procedia CIRP*, 23, 155–160.
- Wegener, K., Chen, W. H., Dietrich, F., Dröder, K., & Kara, S. (2015). Robot assisted disassembly for the recycling of electric vehicle batteries. *Procedia Cirp*, 29, 716–721.
- Weimer, D., Scholz-Reiter, B., & Shpitalni, M. (2016). Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Annals*, 65(1), 417–420.

- Weiss, K., Khoshgoftaar, T. M., & Wang, D. (2016). A survey of transfer learning. *Journal of Big data*, 3(1), 1–40.
- Whitley, L. D. (1989). The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best. In *Icga* (Vol. 89, pp. 116–123).
- Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1912–1920).
- Xu, Y., Fan, T., Xu, M., Zeng, L., & Qiao, Y. (2018). Spidercnn: Deep learning on point sets with parameterized convolutional filters. In *Proceedings of the European conference on computer vision (eccv)* (pp. 87–102).
- Yamany, S. M., & Farag, A. A. (2002). Surface signatures: an orientation independent free-form surface representation scheme for the purpose of objects registration and matching. *IEEE transactions on pattern analysis and machine intelligence*, 24(8), 1105–1120.
- Yao, X. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9), 1423–1447.
- Yao, X., & Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. *IEEE transactions on neural networks*, 8(3), 694–713.
- Yildiz, E., & Wörgötter, F. (2019). Dcnn-based screw detection for automated disassembly processes. In *2019 15th international conference on signal-image technology & internet-based systems (sitis)* (pp. 187–192).
- Yildiz, E., & Wörgötter, F. (2020). Dcnn-based screw classification in automated disassembly processes. In *Robovis* (pp. 61–68).
- Young, S. R., Rose, D. C., Johnston, T., Heller, W. T., Karnowski, T. P., Potok, T. E., . . . Miller, J. (2017). Evolving deep networks using hpc. In *Proceedings of the machine learning on hpc environments* (pp. 1–7).
- Zhang, P., Verma, B., & Kumar, K. (2005). Neural vs. statistical classifier in conjunction with genetic algorithm based feature selection. *Pattern Recognition Letters*, 26(7), 909–919.
- Zhao, H., Jiang, L., Fu, C.-W., & Jia, J. (2019). Pointweb: Enhancing local neighborhood features for point cloud processing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 5565–5573).

- Zhao, Z.-Q., Zheng, P., Xu, S.-T., & Wu, X. (2019). Object detection with deep learning: A review. *IEEE transactions on neural networks and learning systems*, 30(11), 3212–3232.
- Zheng, S., Lan, F., Baronti, L., Pham, D. T., & Castellani, M. (2022). Automatic identification of mechanical parts for robotic disassembly using the pointnet deep neural network. *International Journal of Manufacturing Research*, 17(1), 1–21.
- Zhou, Q.-Y., Park, J., & Koltun, V. (2018). Open3d: A modern library for 3d data processing. *arXiv preprint arXiv:1801.09847*.
- Zhou, X., Qin, A., Gong, M., & Tan, K. C. (2021). A survey on evolutionary construction of deep neural networks. *IEEE Transactions on Evolutionary Computation*.