Graph-based Object Understanding

Dissertation

in order to obtain the doctoral degree
"Doctor rerum naturalium"
of the Georg-August-Universität Göttingen

in the Doctoral program
PhD Programme in Computer Science (PCS) of
the Georg-August University School of Science (GAUSS)

submitted by
Florian Teich

of Göttingen, Germany

Georg-August-Universität Göttingen
Göttingen, Germany
April 2021

Thesis committee

Prof. Dr. Florentin Wörgötter,
Georg-August-Universität Göttingen
Prof. Dr. Wolfgang May,
Georg-August-Universität Göttingen

Other members of the examination board:

Prof. Dr. Carsten Damm,
Georg-August-Universität Göttingen
Prof. Dr. Winfried Kurth,
Georg-August-Universität Göttingen
Prof. Dr. Stephan Waack,
Georg-August-Universität Göttingen
Prof. Dr. Ramin Yahyapour,
Georg-August-Universität Göttingen

Date of the oral examination: 1st of June, 2021

Graph-based Object Understanding

Abstract

Computer Vision algorithms become increasingly prevalent in our everyday lives. Especially recognition systems are often employed to automatize certain tasks (i.e. quality control). In State-of-the-Art approaches global shape characteristics are leveraged, discarding nuanced shape varieties in the individual parts of the object. Thus, these systems fall short on both learning and utilizing the inherent underlying part structures of objects. By recognizing common substructures between known and queried objects, part-based systems may identify objects more robustly in lieu of occlusion or redundant parts. As we observe these traits, there are theories that such part-based approaches are indeed present in humans. Leveraging abstracted representations of decomposed objects may additionally offer better generalization on less training data. Enabling computer systems to reason about objects on the basis of their parts is the focus of this dissertation.

Any part-based method first requires a segmentation approach to assign object regions to individual parts. Therefore, a 2D multi-view segmentation approach for 3D mesh segmentation is extended. The approach uses the normal and depth information of the objects to reliably extract part boundary contours. This method significantly reduces training time of the segmentation model compared to other segmentation approaches while still providing good segmentation results on the test data.

To explore the benefits of part-based systems, a symbolic object classification dataset is created that inherently adheres to underlying rules made of spatial relations between part entities. This abstract data is also transformed into 3D point clouds. This enables us to benchmark conventional 3D point cloud classification models against the newly developed model that utilizes ground truth symbol segmentations for the classification task. With the new model, improved classification performance can be observed. This offers empirical evidence that part segmentation may boost classification accuracy if the data obey part-based rules. Additionally, prediction results of the model on segmented 3D data are compared against a modified variant of the model that directly uses the underlying symbols. The perception gap, representing issues with extracting the symbols from the segmented point clouds, is quantified.

Furthermore, a framework for 3D object classification on real world objects is developed. The designed pipeline automatically segments an object into its parts, creates the according part graph and predicts the object class based on the similarity to graphs in the training dataset. The advantage of subgraph similarity is utilized in a second experiment, where out-of-distribution samples of

objects are created, which contain redundant parts. Whereas traditional classification methods working on the global shape may misinterpret extracted feature vectors, the model creates robust predictions.

Lastly, the task of object repairment is considered, in which a single part of the given object is compromised by a certain manipulation. As human-made objects follow an underlying part structure, a system to exploit this part structure in order to mend the object is developed. Given the global 3D point cloud of a compromised object, the object is automatically segmented, the shape features are extracted from the individual part clouds and are fed into a Graph Neural Network that predicts a manipulation action for each part.

In conclusion, the opportunities of part-graph based methods for object understanding to improve 3D classification and regression tasks are explored. These approaches may enhance robotic computer vision pipelines in the future.

Acknowledgements

First, I would like to thank my supervisor Prof. Dr. Florentin Wörgötter for his guidance on the topics of my dissertation. His constructive suggestions and valuable experiences enriched the research process immensely. Advice given by Prof. Dr. May has been a great help in facilitating this work and steering it into the right direction.

I am particularly grateful for the contribution given by my co-authors Dr. Tatyana Ivanovska, Dr. Shishan Yang, Dr. Simon Reich and Dr. Timo Lüddecke. I would like to offer my special thanks to my colleagues, who always provided constructive feedback. Namely, Carlo Michaelis, Dr. Christian Tetzlaff, Dr. David Kappel, Jannik Luboeinski, Dr. Micheal Fauth, Moritz Becker, Minghao Cheng, Dr. Minija Tamosiunaite, Osman Kaya, Dr. Tomas Kulvicius, Aisha Aamir, Erenus Yildiz, Dr. Fatemeh Ziaeetabar, Sebastian Ruiz, Dr. Sebastian Herzog, Shijia Li. Assistance provided by Ursula Hahn-Wörgötter, Thomas Geiling and Dr. Yvonne Lips was greatly appreciated.

Furthermore, I wish to thank my parents, Tanja and Christian, and my siblings Emma, Laurin, Leander, Lovis and Marlene for their support and encouragement throughout this journey. Finally, I want to thank Laura Mainitz without whom this work would not have been possible.

Contents

1 **Introduction** **1**

2 **Foundations** **7**
- 2.1 Machine Learning 7
- 2.2 Graphs 10
- 2.3 Neural Networks & Graph Neural Networks 12
- 2.4 3D Surface Meshes 15
- 2.5 Metrics 16

3 **3D Object Segmentation** **21**
- 3.1 Introduction 21
- 3.2 Related Works 24
- 3.3 Methods 27
- 3.4 Results 35
- 3.5 Conclusion 40

4 **Concept Learning in 3D** **43**
- 4.1 Introduction 43
- 4.2 Related Works 46
- 4.3 Methods 48
- 4.4 Experiments 55
- 4.5 Results 57
- 4.6 Conclusion 60

5 **Part-graph-based Object Classification** **63**
- 5.1 Introduction 63
- 5.2 Related Works 64
- 5.3 Methods 66
- 5.4 Experiment: Ordinary Object Classification 77

5.5 Experiment: Out-of-Distribution Object Classification 80
5.6 Conclusion 83

6 Assembly Repairment **85**
6.1 Introduction 85
6.2 Related Works 86
6.3 Methods 88
6.4 Experiments 98
6.5 Results 99
6.6 Conclusion 101

7 Conclusion **103**

Acronyms

CAD	Computer-aided design
CD	Chamfer distance
CNN	Convolutional Neural Network
CUDA	Compute Unified Device Architecture
DNN	Deep Neural Networks
DPM	Deformable Parts Model
EMD	Earth mover's distance
ESF	Ensemble of Shape Functions
GCN	Graph Convolutional Network
GH	Graph Hopper
GNN	Graph Neural Network
GPU	Graphics Processing Unit
HED	Holistically-nested edge detection
IoU	Intersection-over-Union
kNN	k Nearest Neighbour
LiDAR	Light Detection and Ranging
LSTM	Long short-term memory
mIoU	mean Intersection-over-Union
MVCNN	Multiview Convolutional Neural Network
MVRNN	Multiview Recurrent Neural Network
MLP	Multi layer perceptron
NN	Neural Network
PN	PointNet
RbC	Recognition-by-Components
RGB-D	Red-Green-Blue-Depth
RI	RandIndex
SDF	Shape-Diameter-Function
SotA	State-of-the-Art
SP	Shortest-Path
SVM	Support Vector Machine
VFH	Viewpoint Feature Histogram
WL	Weisfeiler-Lehman
WWL	Wasserstein Weisfeiler-Lehman

1

Introduction

Applications of Computer Vision can be found almost everywhere in our everyday life: from quality control in automated manufacturing systems [3], through autonomous driving [50] to Just Walk Out Shopping [78]. With an increasing amount of perception devices, there is a growing need for algorithms that are able to process, interpret and make use of raw visual data. 3D sensors such as Light Detection And Ranging (LiDAR) scanners or RGB-Depth (RGB-D) cameras became cheaper over the past years [99]. More and more datasets are offering raw 3D data in addition to 2D data, enabling to leverage depth and geometry information in their benchmarks [68, 104, 125]. Especially in robotics, spatial awareness with respect to real world positions is crucial for tasks such as navigation and motion planning.

Conventional 3D classification systems [73, 85, 97, 108] aim to extract global shape information from the object. Several of these approaches first extract low-level features per point [85], tuple [97] or pixels [108]. Subsequently, this low-level information is aggregated in order to obtain a single global shape feature vector. This feature vector is then used to classify the overall object. However, this process may discard specific shape information. As a single object shape may vary across its surface, the aggregation may lead to a less pronounced shape descriptor. This behaviour, in turn, may result in unreliable pipelines, as objects may be misclassified. This poses the question whether a fundamentally different approach may improve object understanding.

One alternative way to object understanding are part-based methods. Instead of working directly with the global shape, part-based methods address the problem in a bottom-up manner. First, the object is decomposed into its individual parts by using a segmentation algorithm or by employing a part-detection system. Relations between parts can be extracted and a part graph of the overall object can be obtained. For tasks like classification, the extracted and attributed part graph is ultimately used to reason about the object class [33]. A theoretical underpinning of part-based approaches can be found in the "Recognition-by-Components" (RbC) theory by Biederman [10]. In his RbC theory, Biederman suggests that humans might use an internal representation of compositions of primitives ("geons") to perceive objects. For a given object, several different instances of various of these geons would make up the final object, i.e. a mug could be composed of an arc and a cylinder, to represent the handle and the cup respectively. Of course, the (spatial) relations between components and their poses play a critical role in such compositions: If the arc is connected to the side of the cylinder, the resulting object closely resembles a mug, if the arc is connected to the upper flat side of the cylinder, the object will rather resemble a bucket.

The Deformable Parts Model (DPM) [28] is an example of such a part-based system. The DPM was introduced in 2008 by Felszenwalb et al. and uses a part-based model to identify pedestrians in images. Due to high variations across the pedestrians' appearance and poses, detection has often been difficult. In the DPM, object parts (bodies, legs, arms, and heads) were first identified inside the image. Afterwards, the spatial constellation of these components was checked for validity (i.e. "Are the legs below the arms?"). Conventional

State-of-the-Art (SotA) methods of the time had difficulties in dealing with high variation of the global appearance of the pedestrians due to the mentioned variance in appearance and poses, thus frequently failing to detect pedestrians. Through its part-based approach, the DPM successfully recognized many more pedestrian instances than these global approaches. More part-based models for highly structured objects followed, as e.g. for cars [47], animals, furniture [106], and entire rooms [34].

There are at least two important advantages that can be identified when working with part-based models. The first major advantage of part-based systems is that this allows for abstracting the objects in order to learn their underlying structure, similar to what the DPM method accomplished for pedestrian detection. Arguably, instances of a single object class have a high variance in shape. For example, there are many different cars "in the wild", all shaped differently, i.e. SUVs, cabriolets or station wagons. However, all these cars usually have four wheels in a specific spatial relation to each other and a body part on top. As a consequence, the entire spectrum of differently shaped instances of an object class may be abstracted by using a part graph description (containing parts and their relations). Learning - and working internally - with such graphs might reduce the amount of data needed for robust learning of object classes and might allow for easier generalization of classification systems. The second major advantage is that through the use of the part graph representation of the overall object, problematic cases such as occlusion or objects with additional redundant parts become easily classifiable by graph comparison methods between training graphs and query part graphs. Whereas global shape classifiers may be confused by missing or additional parts, part graph-based methods may recover from abundance or scarcity of shape information by identifying common subgraphs between the part graph of the queried object and part graphs from training objects.

An overview on the individual chapters of this work is visualized in Fig. 1.1. Chapter 2 offers explanations of methods and concepts that are referenced in multiple of the following chapters.

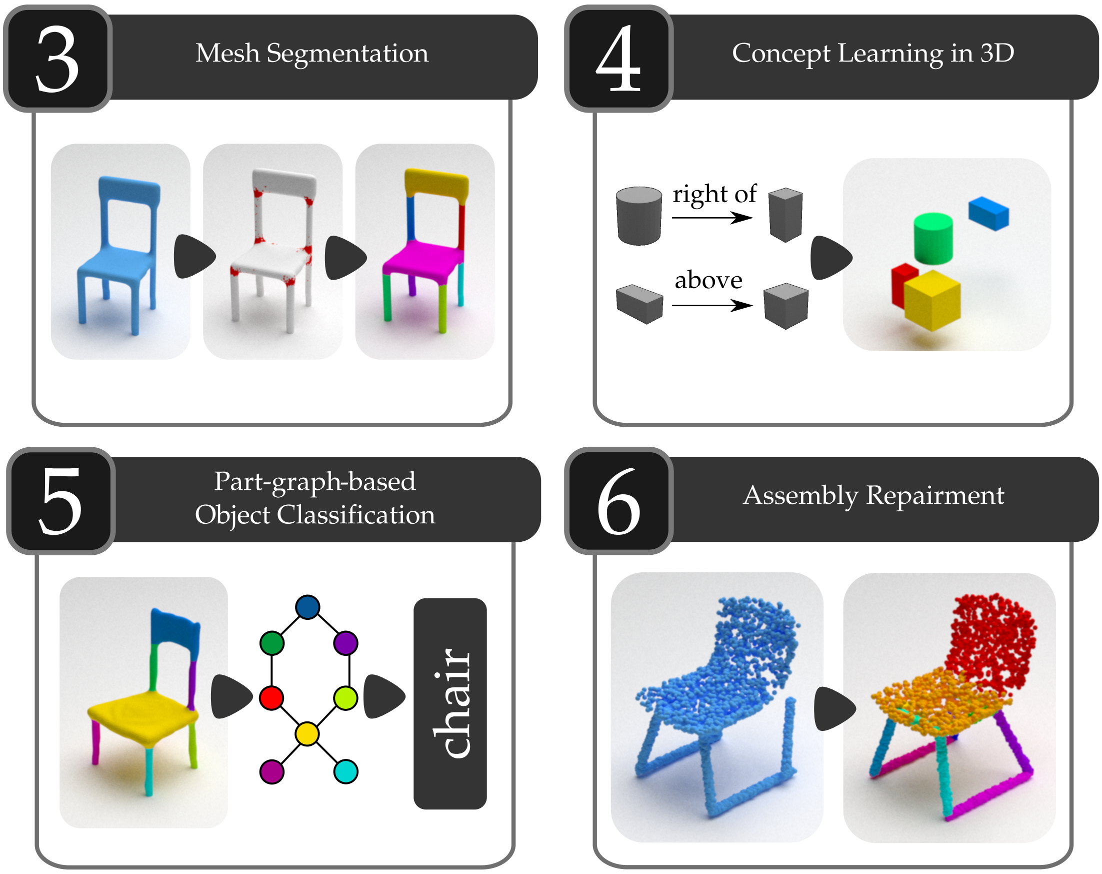


Figure 1.1: Overview on the individual Chapters. As decomposition is essential to part-based classification methods, the issue of 3D mesh segmentation on various objects is tackled in Chapter 3. In Chapter 4, an artificial dataset consisting of symbolic part structures is created and 3D classification methods on this data are tested. In Chapter 5, classification performance of global and part-graph methods on real world 3D object data is compared. Chapter 6 deals with object repairment via part-graphs.

For part-based systems to work, objects need to be segmented first into their individual components. In Chapter 3, a method for fast segmentation on 3D meshes is extended. The results are compared to State-of-the-Art approaches quantitatively, and qualitative segmentation results of the proposed model are shown.

In Chapter 4, a synthetic symbolic 3D dataset is developed that obeys to spatial rules between symbols. Various classification systems on this dataset are benchmarked in order to quantify how well symbols (abstract object parts) are learned in the designed model. Furthermore, it is measured whether providing symbol-level segmentation boosts performance on the employed 3D classifiers.

In Chapter 5, parts obtained by segmentation methods as presented in Chapter 3 are used for the task of 3D classification. The resulting segmentations are used to create part-graphs from 3D objects. By utilizing graph similarity measures, a novel part-graph-based object classification pipeline is developed which is compared to classical global shape approaches.

Object classification is not the only domain in which part-based object understanding might be useful to the system. Especially human-made objects adhere to an underlying structure of parts. For instance, conventional tables are composed of four legs and a tabletop. Whereas humans might be able to intuitively fix a table where one leg is lying on the ground, computer systems may face various difficulties in doing so. Chapter 6 considers the problem of object repairment. An automatic segmentation approach is utilized to obtain part-graphs of the object and densely predict manipulations for each part in order to repair the given object.

To summarize, in this work several possibilities for part-based methods are examined in the context of object understanding. Employing part-graph methods enriches the outcome of classification and regression tasks. This may lead to improved vision pipelines in the future.

Major sections of Chapter 5 contain figures and tables that are adapted or copied from the publication:

3D Object Classification via Part Graphs

Florian Teich (75% contribution), Timo Lüddecke, and Florentin Wörgötter.

VISAPP 2021.

2

Foundations

2.1 Machine Learning

The generic term "machine learning" covers many algorithms and methods but in general describes models that automatically provide a prediction as output to a given input data sample [11]. The prediction is usually affected by data seen during the training phase. Although there are different techniques to train these models, such as reinforcement learning, supervised learning, or unsupervised learning, only techniques from the field of supervised learning are considered in the following chapters. In contrast to other types of machine learning techniques, supervised learning relies on a labeled training dataset [11].

2.1.1 K-Nearest Neighbour

The nearest neighbour method is a widely applied classification strategy. It aims to offer accurate predictions of the target class of a given evaluation sam-

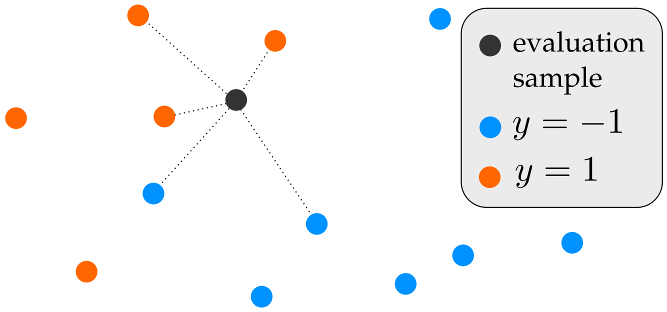


Figure 2.1: k-nearest neighbour visualization of training samples from two classes (orange and blue) and a query sample during evaluation (dark gray), embedded inside the 2D feature space. For $k = 5$, three neighbours are corresponding to class 1 (orange) and two neighbours correspond to class -1 (blue). Thus, the sample is predicted to be from class 1.

ple based on on its proximity to reference samples. The technique is popular for its simplicity and low requirements and computational effort as it avoids any training phase.

Based on n training samples $X = \{x_0, x_1, ...x_n\}$ and their according labels $Y = \{y_0, y_1, ...y_n\}$, a query sample $\hat{x}_i$ is evaluated by determining which label is most prominent in the training samples in the vicinity of the feature vector $\hat{x}$. Only the closest k neighbours are considered (cf. Fig. 2.1). For the distance, various metrics can be used. This method is especially accurate if the samples of the individual classes form separate segments in the feature space which are non-overlapping. Considering multiple neighbours instead of the trivial variant of the closest neighbour usually adds robustness to the method especially in case of samples that lie in between segments [11].

2.1.2 Support-Vector Machine

Support-vector machines (SVM) are models that aim to learn decision boundaries between samples of two classes.

The two classes are labeled as -1 and 1. The training samples are denoted as $x_1, ..., x_n$ with their corresponding targets $y_1, ...y_n$ and $y_i \in \{-1, 1\}$. For classifi-

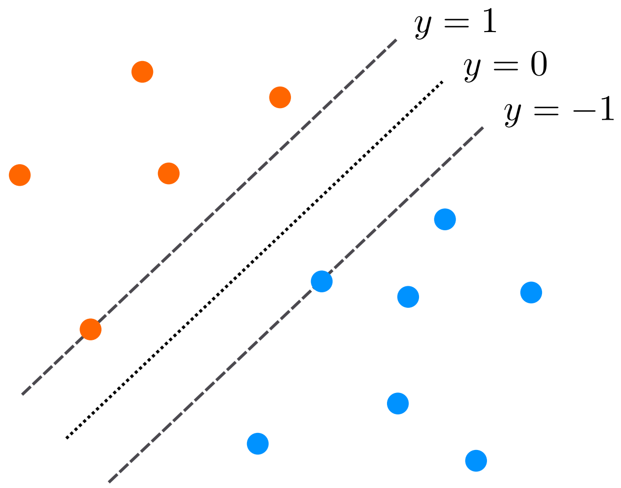


Figure 2.2: Visualization of the decision function, samples and margins of an SVM in feature space. Orange dots represent samples from class 1, whereas blue dots represent samples from class -1. Figure adapted from [11].

cation, the following decision function is be used:

$$h(x) = \text{sign}\left(w^T x + b\right), \tag{2.1}$$

with w being the normal vector to the hyperplane and sign being the sign function, switching value at 0.

The SVM tries to construct a hyperplane between the data samples of the two classes (cf. Fig. 2.2). Optimally, the created hyperplane has a maximum distance (margin) to the closest samples of any of the two classes involved. These close samples are also called "support-vectors". The margin maximization heuristic is important to reduce classification error during testing. This is due to the fact that test samples from one class may be even closer to training samples of the other class. The hyperplane parameters have to be adapted during training to obtain robust classification results during evaluation.

For scaling the binary classification method of SVMs to multiple classes, a one-vs-rest approach can be taken. Here, an individual SVM is trained for each class, where samples are either members of the class or not, thus reducing the problem again to the binary case. For n classes, this results in n SVMs. During evaluation, the test sample is evaluated by all n SVMs and classified according to the SVM that yielded the highest output result to the given input sample [11].

SVMs can be extended, i.e. if the samples of the two classes are overlapping, so-called "soft-margin" SVMs can be employed. In software frameworks [21, 82], kernel functions or precomputed kernel matrices ("Gram matrix") can be provided to the SVM instead of the specific data samples in feature space. The kernel function k can be applied to tuples of samples, thus resulting in the kernel matrix when applied to all combinations of samples. This kernel matrix can be considered as a novel feature space of the original samples. A popular choice for the kernel function is the dot-product $k(x_i, x_j) = (x_i \cdot x_j)$. Especially in Computer Vision, similarity measures such as Histogram Intersection [6] are frequently used:

$$k(x_i, x_j) = \sum_{k=1}^{m} \min(x_{ik} \cdot x_{jk}), \tag{2.2}$$

where m is the number of bins of each histogram. x_{jk} refers to the kth bin inside histogram x_j.

2.2 Graphs

The basic structure of a graph can be denoted by $G = (V, E)$. V represents the set of nodes of the graph, whereas E represents the set of edges inside the graph. Each edge is characterized by a source and a target node. Graphs may be directed or undirected.

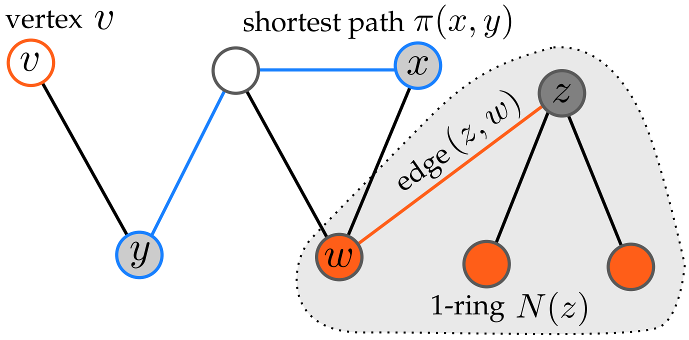


Figure 2.3: Different concepts inside graphs: vertices, edges, shortest path and 1-ring neighborhood. Figure adapted from [57].

For undirected graphs, $(v_i, v_j) \in E$ if and only if $(v_j, v_i) \in E$ (cf. Fig. 2.3). Nodes are attributed by numerical or categorical properties. If all node attributes are categorical, the node attributes are considered discrete and the graph is called "labeled". If all node attributes are numerical node attributes, the graph is considered to be "attributed". Additionally, edges may be attributed by weights, indicating the connection strength between nodes. To represent the set of edges inside a given graph G, the adjacency matrix A can be used:

$$A_{ij} = \begin{cases} 1 & \text{if } (v_i, v_j) \in E \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

Therefore, the adjacency matrix A has size $n \times n$, where $n = |V|$ is the number of nodes inside the graph. Another important graph matrix is the degree matrix D:

$$D_{ij} = \begin{cases} deg(v_i) & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

where deg is the degree function of the node, counting all edges that this node is a source or target of. Analogous to A, D is of size $n \times n$. From these two matrices, the Laplacian L of a graph can be obtained: $L = D - A$. The graph $G = (V, E)$ is usually attributed with node feature vectors $x_i, \forall i \in V$.

2.2.1 Weisfeiler-Lehman test

The Weisfeiler-Lehman test is usually used in order to determine graph isomorphisms. Satisfying this test is required for isomorphism, but not sufficient (there are indeed non-isomorph pairs of graphs that satisfy the WL-test, but these are usually rare). The Weisfeiler-Lehman (WL) test uses the concept of label propagation to test whether two labeled graphs are isomorph [101]. In this context, labeled means that a discrete label is assigned to each node of each graph.

Given a labeled graph, all nodes are updated in an iterative manner until no

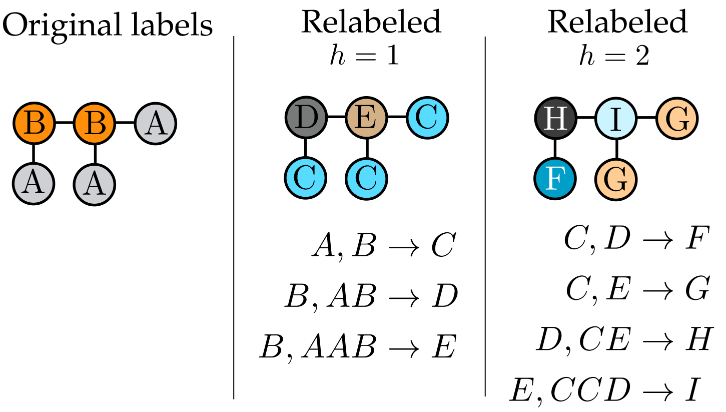


Figure 2.4: Visualization of the color refinement scheme via Weisfeiler-Lehman for discretely labeled graphs. For the attributed graphs, Eq. 5.11 is used as propagation scheme between iterations. Figure adapted from [57].

changes occur. In each update step, each node is relabeled by considering its own label and the multiset of labels in their neighbourhood; if the resulting label was never used before, a novel label is thus created (cf. Fig. 2.4). When considering labels as colors, this algorithm can be easily visualized as over the iterations, the node colors will change as neighbourhood information is propagated. Applying this coloring scheme to two graphs can then be used to check whether both will converge to the same composition of colors.

2.3 Neural Networks & Graph Neural Networks

Graph Neural Networks (GNNs) are a family of recently developed methods that extend the Neural Network (NN) framework and enable graph processing. In Neural Networks, neurons are the atomic units involved in the decision-making process of the model.

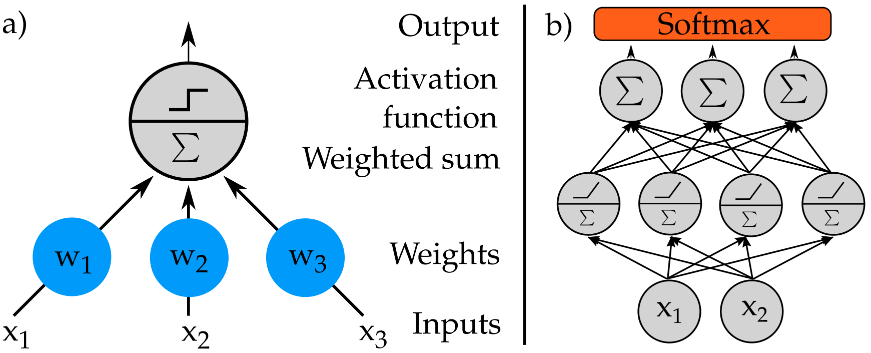


Figure 2.5: a) Simplified perceptron. b) Multi-Layer-Perceptron. Figures adapted from [38].

Neurons are characterized as computational nodes that are connected to (multiple) input nodes, each of which may have an individual connection strength to the target neuron (cf. Fig. 2.5). The neuron weighs each input signal according to the respective connection strength, sums up the result and - based on an activation function - returns an output signal.

During training, the connection strengths (or "weights") between input nodes and the neuron get adjusted to result in better output estimates. This adjustment is usually accomplished by backpropagation [93]. The backpropagation training algorithm requires an error signal in order to adjust all weights of the network appropriately. In the case of supervised learning, this error signal reflects the discrepancy between predicted output of the network and desired output ("ground truth"). Multiple neurons are often grouped together (cf. Fig. 2.5, b). These groups are called layers and are often evaluated sequentially.

Especially in Computer Vision tasks, convolutional layers show promising results and are widely employed in many different models [58, 110, 127]. In convolutional layers, a convolution operation over the input signal is applied, where input as well as output may consist of multiple planes, so called "channels". The weights of the convolutional layer are called the kernel, which is

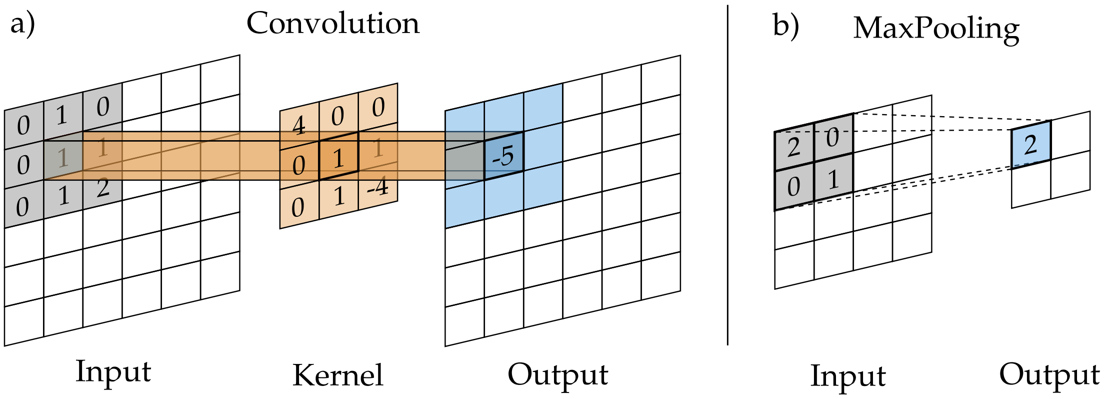


Figure 2.6: a) Convolution in 2D: element-wise multiplication of the input by the kernel and subsequent aggregation by a summation operation. b) MaxPooling in 2D: the maximum element inside the group of 2×2 activations is selected as output.

element-wise multiplied by the input and subsequently aggregated by a summation operation (cf. Fig. 2.6, a). This convolutional operation is executed over the entire input in a sliding window manner, where the parameters "stride" and "dilation" influence the step size and the input elements that are considered. As the same kernel is applied to all regions of the image, the weights are considered to be "shared". Per convolutional layer, multiple of these kernels are usually maintained in order to capture multiple important patterns. Each convolutional layer may also lead to a smaller output than its input. In image classification, a pooling strategy called "Max-Pooling" (cf. Fig. 2.6, b) is frequently used to reduce a group of activations to a single scalar by applying the max operator. For the final classification, the intermediate activation matrix is usually reorganized into vector form and processed by a Multi-Layer-Perceptron (MLP) classification head. This MLP head will reduce the extracted n-element feature vector to a dense prediction: a scalar for each output class, called "logits". To obtain the respective class probabilities from the logits, a Softmax layer can be employed:

$$\text{Softmax}(\hat{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, ..., K \text{ and } \hat{z} = (z_1, ..., z_K). \tag{2.5}$$

The concept of convolutional layers was later adopted for Graph Neural Networks (GNNs). In GNNs, the input to the network is provided as graph data, which might be highly heterogenous compared to CNNs used for image classification. Where CNNs often expect the input image to have a specific size, GNNs are able to handle various graphs of different node and edge sizes. For these GNNs, the adjacency matrix A and the feature matrix X are used as input. A graph layer at depth $l+1$ is generally formalized as

$$H^{(l+1)} = f(H^{(l)}, A), \tag{2.6}$$

with $H^{(0)} = X$ and f being a specific propagation function. An example of a propagation function would be:

$$f(H^{(l)}, A) = \sigma\left(AH^{(l)}W^{(l)}\right), \tag{2.7}$$

where $W^{(l)}$ is the l-th layer weight matrix and $\sigma(\cdot)$ represents an activation function. After extending this rule to account for self loops (such that the current nodes' own internal state is leveraged as well as its neighbours) and normalizing A to avoid an explosion of the scaling in deeper layers of the network, the following formula is obtained [56]:

$$f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right), \tag{2.8}$$

with $\hat{A} = A + I$, I being the identity matrix and the diagonal node matrix of $\hat{A}$:

$$\hat{D}_{ii} = \sum_j \hat{A}_{ij}. \tag{2.9}$$

There are different graph convolutional layers, differing in propagation or aggregation behaviour [75, 129]. Others make use of edge weights between the nodes of the graph [30, 32] or employ attention-mechanisms [113, 118].

2.4 3D Surface Meshes

Meshes are one of several modalities for describing objects in 3D. Digital 3D mesh models are composed of polygons. Polygons can be described by a set of vertex coordinates

$$V = \{v_1, ..., v_V\}, v_i \in \mathbb{R}^3 \tag{2.10}$$

and a set

$$F = \{f_1, ..., f_F\}, f_i \in V \times V \times V \tag{2.11}$$

of lists of indices referring to vertex indices and thus creating the polygons. Such a collection of polygons is called a "polygon soup". Triangle meshes only consist of triangular polygons. As multiple triangles may share common vertices, they can create more complex surfaces and topologies (cf. Fig. 2.7). These

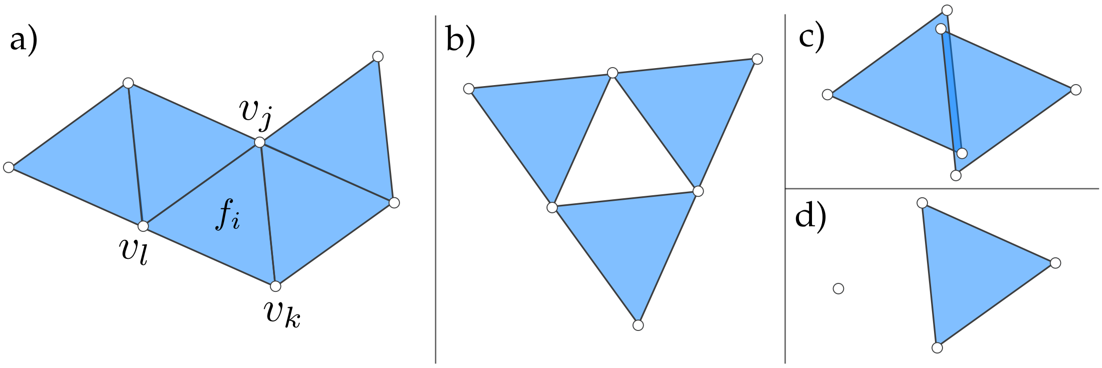


Figure 2.7: a) Example of triangles (faces) forming a mesh. Examples of degeneracies in meshes: b) A single hole (missing face definition). c): Overlapping faces. Here, no common topology can be extracted, as the two faces do not share a common edge. d) Non-referenced vertex. The unreferenced vertex cannot be reached via any path as no incident edge to the vertex exists. Figures adapted from [14].

surface meshes often contain degeneracies. Such degeneracies can be holes, which result from missing surfaces, or overlapping faces, which are hard to locate by visual inspection. In case of overlapping faces, pairs of vertices might be close to each other but each is referenced by one of the faces involved. Another problem are unreferenced vertices, which can never be reached if only traversing the face-topology. All these degeneracies need to be repaired, as most algorithms on 3D meshes assume watertightness (mesh creates a closed volume) or at least 2-manifoldness (surface is locally homeomorphic to a disk) [14].

2.5 Metrics

To objectively compare or rank various approaches, metrics can be applied on a given test set. These metrics often reduce multiple results to a small amount of values, or - in some cases - single scalars.

2.5.1 Accuracy

Accuracy is most frequently used for classification tasks and measures the methods' success rate. Given a sequence of predictions $\hat{y}$ and ground truth y of length n_{samples}, the accuracy of the classifier can be calculated as:

$$\text{Accuracy}(y, \hat{y}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \delta(\hat{y}_i, y_i), \tag{2.12}$$

with δ being the Kronecker delta:

$$\delta(i, j) = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases} \tag{2.13}$$

A downside of this metric is that possible imbalances between the various classes inside the dataset are not considered. In case of the dataset containing significantly more samples of specific classes compared to other underrepresented classes, this dataset is considered imbalanced. In this scenario, high accuracy values are misleading as they can be achieved by always predicting classes of high representation. One approach to improve the meaningfulness of the accuracy metric on imbalanced data is to use a weighting scheme for individual classes according to their frequency.

2.5.2 Rand Index (RI)

Rand Index is a widely used [4, 39, 54] metric for segmentation tasks. Given two segmentations, the Rand Index is represented as a scalar reflecting the similarity between the two segmentations. In [23], the Rand Index is formalized as:

$$\text{RI}\left(S_1, S_2\right) = \binom{2}{N}^{-1} \sum_{i,j,i<j} \left[C_{ij}P_{ij} + \left(1 - C_{ij}\right)\left(1 - P_{ij}\right)\right], \tag{2.14}$$

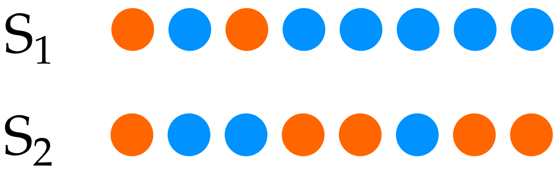


Figure 2.8: Example of two binary segmentations S_1 and S_2. The original entity consists of eight primitives. S_1 and S_2 both assign each of these eight primitives to one of two segments. There is no limitation on how many segments (here represented as colors) are used in any segmentation.

where S_1, S_2 represent two given segmentations, containing sequences $s_1^1, ..., s_n^1$ and $s_1^2, ..., s_n^2$ indicating which segment the ith element is assigned to. $C_{ij} = \delta(s_i^1, s_j^2)$ is the Kronecker delta comparing element s_i^1 and s_j^2, whereas $P_{ij} = \delta(s_i^2, s_j^2)$ is the Kronecker delta comparing elements of the same segmentation: s_i^2 and s_j^2. It is important to note that Funkhouser [23] defined the RI score as $1 - \text{RI}$ to reflect a measure of dissimilarity instead of similarity. In the following chapters, this convention will be used. Lower RI scores are desired in segmentation approaches as these represent low dissimilarities between predicted and reference segmentation.

2.5.3 Mean Intersection-over-Union (mIoU)

The mean Intersection-over-Union (mIoU) is a metric to capture semantic segmentation performance often used to monitor fully-convolutional segmentation networks. The Intersection-over-Union (IoU) or Jaccard-Index [49] can be calculated by:

$$\text{IoU}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \tag{2.15}$$

For each semantic class, an individual IoU can thus be calculated between a predicted semantic segmentation and the ground truth. If the average of all semantic classes is taken, this metric is considered the mIoU.

In the example shown in Fig. 2.8, two sets of segmentations of eight entities are presented. Calculating the IoU for each individual class (orange & blue), yields:

$$\mathrm{IoU}_{\mathrm{o}}(S_1, S_2) = \frac{1}{6}, \qquad \mathrm{IoU}_{\mathrm{b}}(S_1, S_2) = \frac{2}{7}. \tag{2.16}$$

Averaging over both classes results in mIoU ≈ 0.226, where mIoU $= 0$ would mean no agreement in segmentation and mIoU $= 1$ would indicate identical segmentation.

For IoU and mIoU concrete labeling of the entities is required, whereas the Rand Index does not make use of this information.

3

3D Object Segmentation

3.1 Introduction

Segmentation is an essential component of many Computer Vision processes, especially tasks related to scene understanding. In 3D mesh segmentation, input objects are decomposed into their parts. Part segmentation is quite challenging as often part boundaries between segments are not easily detectable. Moreover, specific formal criteria (such as ones regarding concavity and curvature) for parts or part boundaries may not be applicable to all possible scenarios. For instance, one might consider the forearm and the upper arm two separate parts of a human body, even when the arm is stretched and only very small visual cues - such as creases - offer explanation to the existence of the part boundary. Both geometrical properties as well as semantic criteria impact our understanding of what we consider as parts of an object. Providing explicit semantic knowledge (i.e. an arm is composed of upper and forearm) about the world is often unfeasible in practice.

It is important to distinguish Segmentation from Semantic Segmentation and Instance Segmentation - two other popular tasks in Computer Vision. For segmentation, the input is only separated into - often disjunct - regions, where-as in Semantic Segmentation the input data gets densely labeled in addition. In Instance Segmentation, additionally to dense semantic labeling, entity instances are separated from each other.

Instead of explicit semantic knowledge, thanks to recent advances in Machine Learning, systems can be trained on collected data, such as ground truth segmentations of objects. Generated by humans, the ground truth can be used as a supervision signal to a model that tries to reproduce such ground truth by predictions based on the original input data (supervised learning). Especially in Computer Vision, these supervised and data-driven methods are often employed, as image data acquisition and annotation is usually cheap. The goal is to avoid formalizing concepts explicitly and to train the model on data that implicitly contains the concepts. Deep Neural Networks succeed in many of these Computer Vision tasks such as classification and segmentation. Segmentation algorithms serve the purpose of partitioning the input into disjoint clusters. Such algorithms map each input primitive to a specific cluster index. For point cloud input, a segmentation algorithm will assign a specific cluster index to each point. For mesh input, segmentation will assign cluster indices to faces on the mesh surface.

Various segmentation approaches exist for the different 3D modalities (cf. Fig. 3.1). These approaches differ in the criteria they employ to partition the input object into multiple clusters. Popular criteria make use of insights from perception and psychology [44, 67, 105], especially the concepts of convexity/concavity [24, 53, 80, 98] and the minima-rule [23, 51]. Through the use of CNNs, systems can be trained to segment given 2D images, especially important for robotics and autonomous driving. In recent years, focus shifted more and more towards 3D data as 3D sensors are getting cheaper and systems may benefit from the depth information gained by 3D data compared to 2D images. But

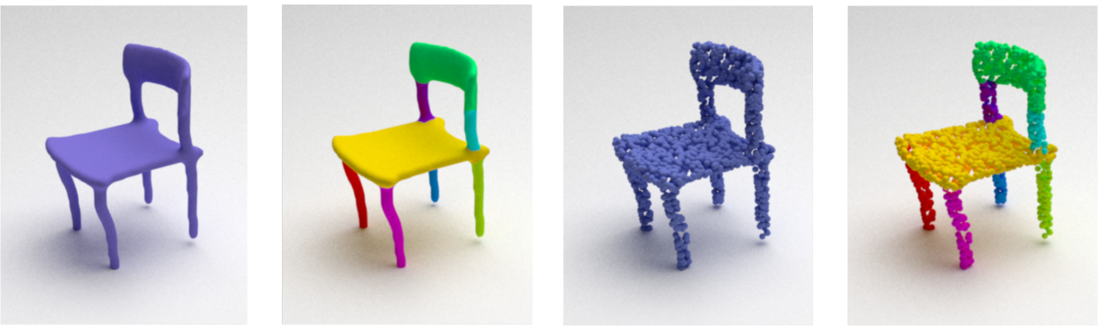


Figure 3.1: Different 3D modalities and respective segmentations. From left to right: Original mesh, segmented mesh, original point cloud, segmented point cloud.

more data - in general - also requires more space, more computational power and often more time to train a system. Since the field of 3D machine learning is still young, many of its methods have been lent from 2D machine learning. Results in [84] on 3D data classification tasks suggest that 2D methods may often outperform techniques that explicitly work on the original 3D data. In these 2D techniques, the 3D shapes - usually provided as mesh surfaces - are projected onto a virtual camera and these rendered images are subsequently classified [108].

A conventional rendering technique for 2D image projections of 3D data is the Phong shading [83], resulting in a gray scale image of the mesh. The pixel intensity is determined by the angle of the surface normal, the camera pose and the light sources' location. Using such rendered images of the 3D shapes, a CNN can be employed in order to obtain a mesh classification model. In [108], Su et al. combine images taken from multiple positions around a canonical pose of the object. The authors combine these views by means of a pooling layer. The max-pooled activations result in a single feature vector that can be fed to the classification head of the network.

In [62], analogous to the classification task, a method was developed to segment 3D shapes based on multiple 2D projections. Their pipeline consists of two parts: first, the rendered multiple views of the 3D model are individually

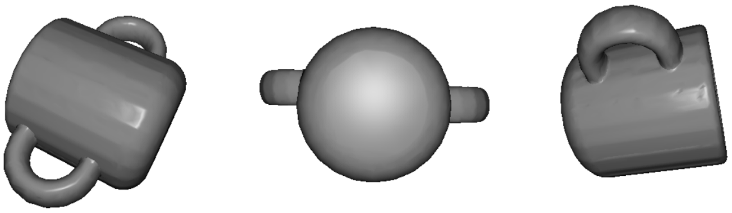


Figure 3.2: Phong shading renderings of a 3D mug object from various different views.

fed into a Fully-Convolutional MVCNN, meaning all layers are Convolutional/Pooling/Upsampling layers such that the output of this model will have the same size as the input. The goal is to train this model to detect salient regions inside the image that may indicate part boundaries.

In the following, the data-driven segmentation approach from [62] (MVCNN) is extended ("MVCNN++") by leveraging different surface properties such as normal and depth information. Furthermore, various possible design choices for the proposed method are discussed and tested in order to identify the optimal model architecture. The performance of the investigated method is quantitatively compared to other popular segmentation algorithms via a benchmark dataset. Advantages and disadvantages are later discussed, focusing on qualitative segmentation results and comparing them to ground truth segmentations.

3.2 Related Works

Classical 3D segmentation techniques often rely on concepts such as finding creases and concavities on the shapes' surface, whereas newer methods usually use supervised data-driven approaches to learn the features that are important for segmentation implicitly.

Clustering techniques such as K-means [71] were one of the first methods applied in the field of mesh segmentation [102]. Based on randomly chosen seed faces, the mesh can easily be clustered by assigning each face to the closest prototype. Weighting of the dihedral angle between faces even allows for a more advanced clustering that will be sensitive to concavities and convexities. However, this 3D segmentation method requires user input about the number of final clusters and is therefore not usable as an automatic approach. In the ShapeDiam [100] approach, the Shape Diameter Function, measuring the length of rays inside the mesh is used in order to find salient part boundary regions. This approach is able to automatically choose a fitting amount of segmentation clusters on its own based on a heuristic that reasons about the SDF distribution on the global shape. Especially 3D objects that are accurately described by their skeleton (medial axis transform) can be well segmented with this method. However, objects which contain a lot of unsmooth creases - such as box-shaped furniture created from CAD software - are often poorly segmented by the SDF approach. In CoreExtra [54], the authors use Multi Dimensional Scaling (MDS) [59] to transform the input mesh into a canonical pose in order to extract feature points for final the segmentation. Their method leverages geodesic distances between the points of the mesh. NormCuts [39] is a face clustering technique that leverages face area and concavity information in a cost function to hierarchically merge adjacent face clusters. Normalized Cuts are a widely employed technique in 2D Computer Vision for image segmentation. However, the approach [39] does not include any heuristic to infer a fitting number of segments, thus rendering it as a manual approach. The RandWalks [60] method employs Random Walks on the mesh surface, relying on the dihedral angle between faces for appropriate traversal costs. Again, this method has to be initialized with a desired number of final clusters. Attene et al. [4] developed a method that approximates the given input shape as a collection of primitives. The proposed method uses spheres, cylinders and boxes as the set of primitives used for fitting. Whereas this segmentation approach performs very well on “Computer-aided design” models (“CAD”), the segmentation results for other objects such as *animals* and *humans* are often inadequate. Benhabiles et al. [9] extract rich features related to curvature and concavity for the edges between vertices of the mesh. With the help of a SVM they are able to train a model on predicting boundary and non-boundary edges on novel meshes, based on the aforementioned features. After a region thinning step, contours between parts are obtained and optimized in order to retrieve the final segmentation [4, 100].

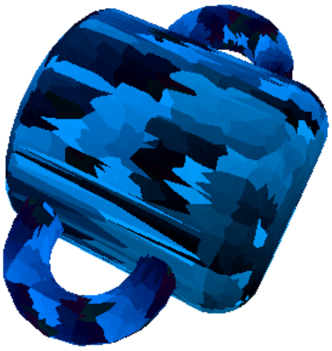


Figure 3.3: Example rendering used for backprojection. Each pixel is color coded by the face ID that was projected to the 2D plane to obtain this rendering. From the RGB value, the original faces ID can be obtained.

Since the first mesh segmentation papers contained rather qualitative results showing particular output of their methods, Funkhouser et al. [22] created a dataset of 380 meshes from 19 different object classes such that existing methods can be compared to one another more easily. In order to also compare to and analyze human performance, multiple human annotations of manual segmentations of each model into its parts was provided. Interestingly, the segmentations from different users do not always agree, i.e. some users segmented human models into 6 parts (head, body, 4 extremities), others segmented the same models into 10 parts (hands and feet separately).

In [62], Truc et al. explored the task of 3D mesh segmentation by using projections in 2D. By providing multiple views of the same 3D object to the network, possible part boundary regions from various perspectives are obtained. These resulting "edge probability maps" are then fed to a Long-Short-Term-Memory [43] ("LSTM"), whose task is to correlate boundary regions across multiple maps in order to obtain more consistent edge probability maps. The order in which the multiple edge probability maps are fed into the LSTM is fixed, as a canonical ordering of views is initially defined such that the maps create a time series (a video) which the LSTM processes. The output of the LSTM will still be an ordered series of boundary probability maps. Afterwards, via a technique called backprojection (cf. Fig. 3.3), the boundary candidates can be recovered inside the original mesh. A mapping between the original mesh faces and the pixels' coordinates inside the rendered views (and the boundary probability maps) is maintained. Finally, a region growing algorithm is employed

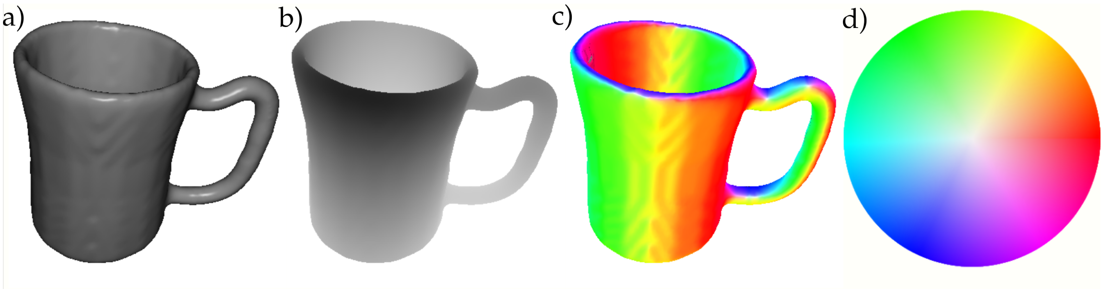


Figure 3.4: Different rendering methods used: a) Phong shading. b) depth map. c) color coded normal vectors relative to the camera. d) normal map used for color coding.

on the original mesh in order to obtain the mesh segmentation of the 3D model. The reported training time of the MVRNN model is three days for the MVCNN and additional three days for the LSTM stage, both ran on an Nvidia TITAN X. This does not include the preprocessing steps of rendering all training and testing objects, resulting in a very time- and computation-expensive method. The authors of MVRNN justify the need for the LSTM step by suboptimal segmentation results of the first stage alone - the Fully-Convolutional MVCNN.

3.3 Methods

In this work, similar to [62], a fully-convolutional network is employed. The multiple rendered views serve as input to this network. In contrast to [62], the original MVCNN which uses the "holistically-nested edge detection" (HED) architecture [126] is substituted by a Fully-Convolutional ResNet architecture [41] ("DenseResNet"). This new architecture will create output that is nearly on par with the original MVRNN two-stage method, making the refinement step nearly redundant (as it provides diminished return) and saving a lot of training time and computations along the way. Secondly, the input size of the rendered images is increased to 256 as premature experiments showed that higher image resolutions resulted in better segmentations in general, possibly since in the backprojection step boundary faces may not be visible inside the 2D projection, leading to "bleeding" effects in the final region growing stage. Two

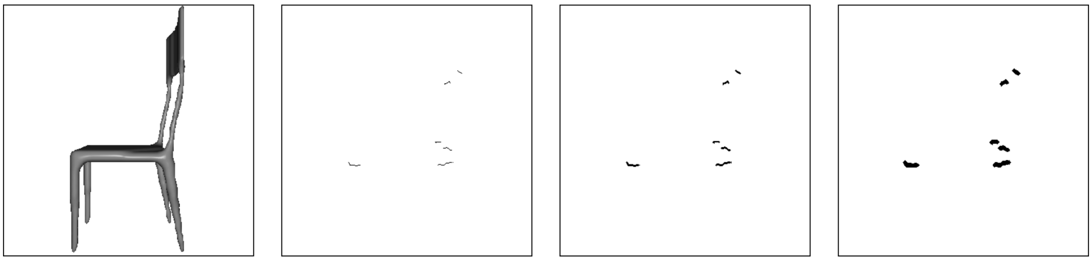


Figure 3.5: Original 3D object rendering and dilation results of the ground truth segmentation pixels. From left to right: phong shading, ground truth part boundary mask, weakly dilated (1 dilation iteration) part boundaries, strongly dilated (2 dilation iterations) part boundaries.

more modalities are provided for each rendered view to the network: a normal rendering and a depth map. The motivation behind this is that part boundaries are often observable at creases between two parts. These creases may be easier detectable by the respective depth map or normal map as both normals as well as depth strongly varies around these regions. Furthermore, true part boundary masks are dynamically changed during the training stage: as the true boundary pixels occupy only a small subset of pixels in the rendered images, the masks are dilated with a big kernel at the first epochs and this dilation kernel is reduced over time (cf. Fig. 3.5). This strategy may lead to faster training convergence as the model will start learning to identify fuzzy regions that contain boundaries first and in later epochs it will shift to exact boundary localisation.

The overview of the resulting method is illustrated in Fig. 3.6. First, 60 2D projections from different poses around the meshes are rendered using different modalities (phong shading, depth- and normal maps). Also, ground truth part boundary maps are rendered in this step. Next, the novel MVCNN++ evaluates all 60 views of a given mesh to densely predict locations of the part boundaries in each of these views. Afterwards, the part boundary predictions from all 60 views are combined and the respective regions on the mesh are marked via backprojection. Using an automatic clustering technique (GraphCut [15]), the final segments are identified on the object surface. In the following sections, these steps are explained in greater detail.

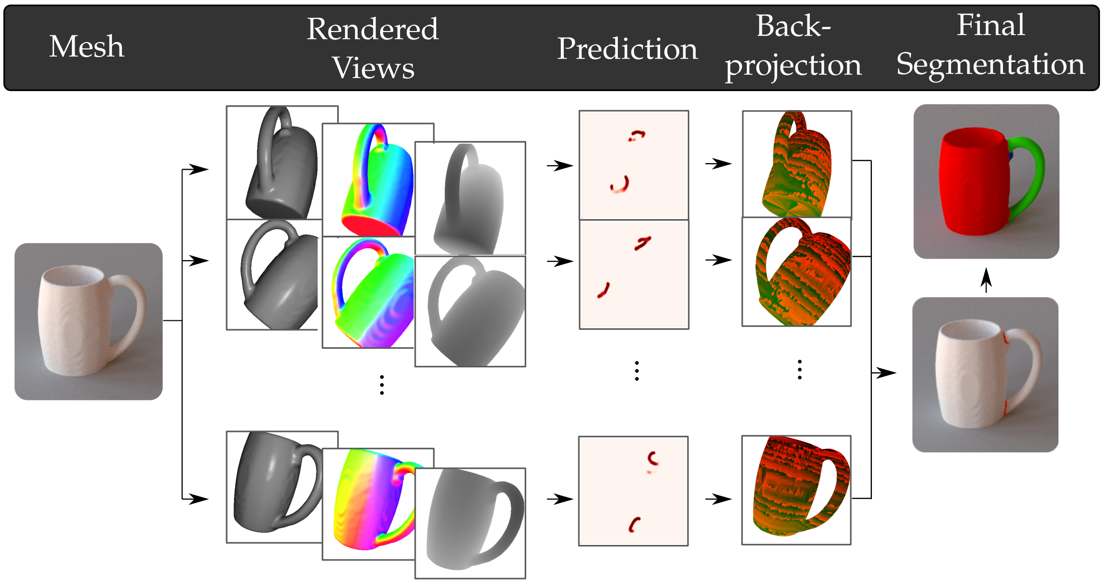


Figure 3.6: Pipeline of the proposed segmentation method. 60 views are rendered from the original input mesh, each view creating images for phong shading, normals and depth maps. The different modalities of a single view are concatenated and fed into the DenseResNet architecture, which predicts per-pixel part boundary probabilities. Aggregating all predictions from the 60 views, the pixels can be reprojected to the original mesh faces. Using the GraphCut algorithm [15], the final segmentation is obtained.

Preprocessing

Similar to the MVRNN preprocessing, all 3D objects were rendered from 60 different poses (fixed positions of the camera around the unit sphere). The objects are scaled in order to capture their full extent in each image. The camera is always oriented towards the coordinates $(0|0|0)$. In total, 380 3D objects from the Princeton MeshSeg dataset were used across 19 object classes with each 20 instances. Similar to Truc [62], 16 objects per category were used as training samples and 4 objects were used for testing respectively. For the training instances, the canonical (upward) object pose and 15 random initial rotations of the object were used in order to augment the data. For each object rotation, 7 different renderings were created for each of the 60 views:

- Phong shading: grayscale image (single channel), as in [83] (cf. Fig. 3.2)
- Normal map: RGB image (three channels), where the color represents

the values of the surface normal at the specific location relative to the camera orientation. To map the vectors to color values, the difference between the surface normal and the camera orientation in polar coordinates is calculated and the image in Fig. 3.4 d) is used as the normal color map.

- Depth map: (single channel), indicates the distance of the surface to the camera. The values are clipped between 0 and 0.9 for each image, where 0.9 indicates the closest distance between camera and the unprojected object surface and 0 indicates farthest distance respectively. Clipping at 0.9 was done in order to still be able to distinguish the object from the background (which has value 1.0).
- Ground truth part boundary mask: (single channel, only provided to training instances) indicates which pixels represent regions between two parts. As mentioned above, the boundary mask is dilated in earlier training steps.

Model and Training

In this section, the design of the proposed model is described. The purpose of the model is to retrieve a 2D boundary prediction from a given 2D image input. Such a model requires a fully-convolutional architecture: the output size needs to be the same as the input size. Output of convolutional filters usually tends to be smaller than their input. Therefore, an autoencoder-like structure was imitated, where in the first part of the network the input image is processed through a cascade of convolutions which extract important features. This will result in a spatially small vector or matrix that afterwards needs to be inflated again to obtain an output matrix that has the same dimensions as the original input image. For this inflation, bilinear upsampling operations are used to increase the current feature matrix.

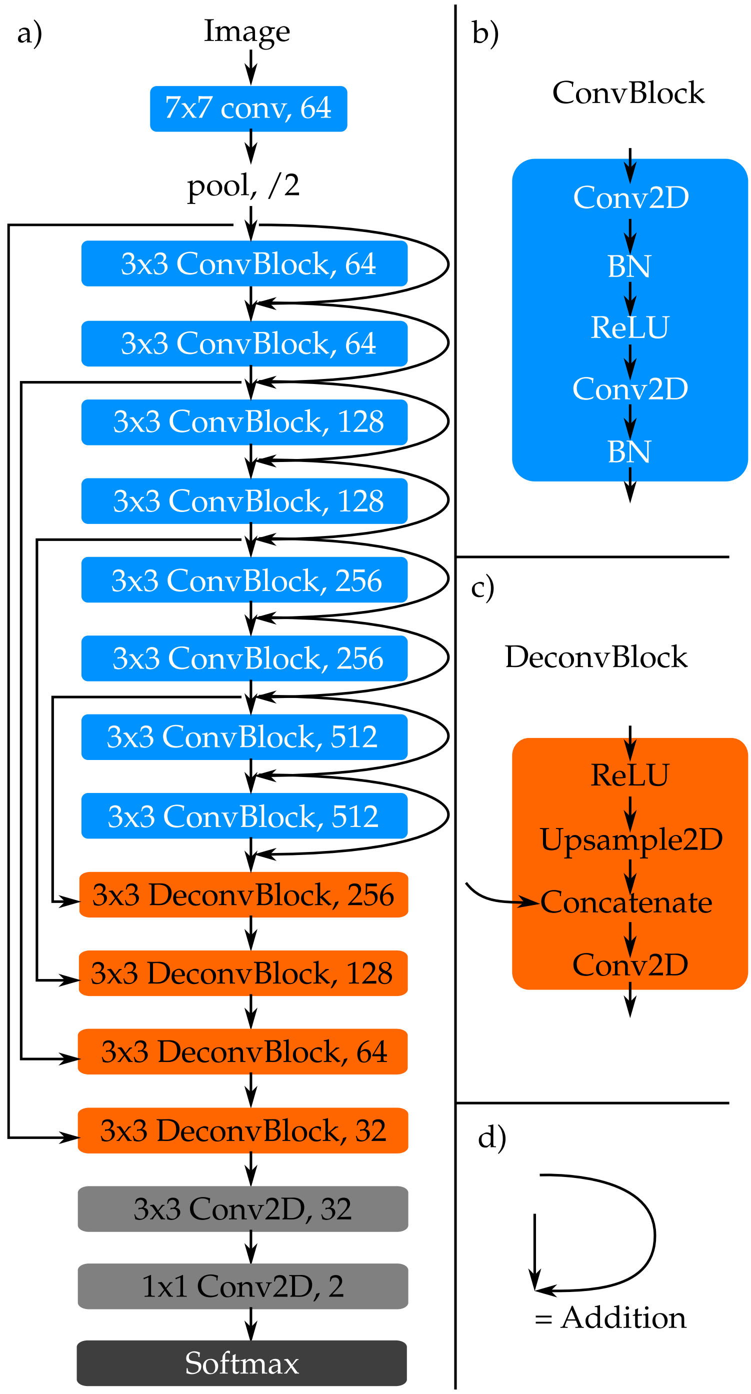


Figure 3.7: a) The fully-convolutional DenseResNet architecture. b) ConvBlocks consist of mainly two sequential convolutional layers, each followed by BatchNorm [48]. After the first BN, a ReLU activation [40] is used. c) a DeconvBlock obtains the input from the previous layer/block which is upsampled and concatenated with output from earlier ConvBlocks. d) illustrates that residual connections are added to block outputs at specific points.

A dense ResNet18 [41] (a fully-convolutional variant of ResNet, in the following called “DenseResNet”) is used as the proposed model for this boundary-detection task and trained in mini-batches of 32 view instances per batch. The ResNet [41] architectures contain blocks (sequences of layers) where the input of each block is added to its output before feeding it to the next layer or block. These shortcut connections force the network to fit a residual mapping. The authors of ResNet argue that fitting this mapping requires less effort than fitting the underlying desired mapping. Eight “ConvBlocks” are used to first extract important features from the input image (cf. Fig. 3.7). The amount of kernels that are used inside the convolutional layers is successively increased from 64 up to 512. For the size of the convolutional kernels, 3×3 was chosen as in the original ResNet work [41].

In order to densely predict the likelihood of part boundaries, bilinear upsampling is used. After the upsampling, the result and the input from the residual connection are concatenated and are fed into another convolutional layer for each “DeconvBlock”. The last two “DeconvBlocks” in the model skip the concatenation as they do not receive any residual input. Using this sequence of deconvolutions, the original image size is reached. The penultimate layers’ output is a binary map of logits representing background (no part boundary) and foreground (part boundary). With its skip-connections, this architecture highly resembles the overall structure of fully-convolutional networks such as UNet [90] or FCN [69].

Model predictions are visualized in Fig. 3.8. Analogue to MVRNN [62], every training object is sampled from 16 random rotations so that in total 291840 image instances are obtained (16 objects × 19 object classes × 16 random rotations × 60 renderings). 10% of the training instances are randomly sampled as validation set that is evaluated after each epoch in order to measure the boundary edge detection performance on held back, unseen instances. The ADAM optimizer [55] was used with a learning rate of 0.001, the models were trained for 50 epochs and early stopping was employed to stop the training process if the validation performance (measured in mIoU) did not improve for the last five

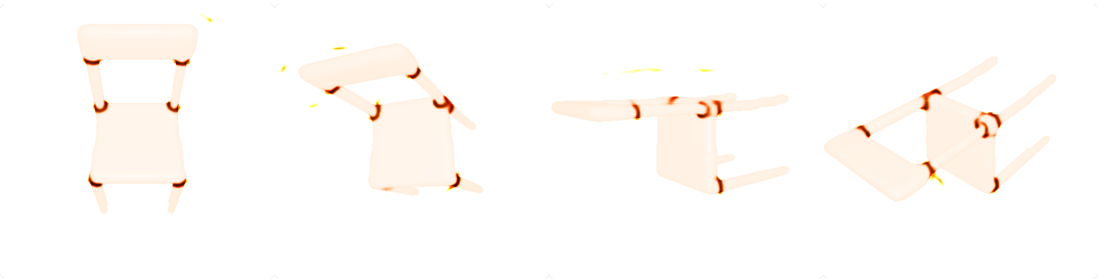


Figure 3.8: Predictions of the DenseResNet model on a chair overlaid on Phong rendering of the view.

epochs.

Post-processing and segmentation evaluation

After the model evaluated all 60 views of a given object, 60 prediction maps from different poses are obtained. In combination with the backprojection renderings, the faces that occupy predicted part boundaries can be identified. For each of the 60 views, the respective part boundary faces are stored and subsequently aggregated: If any of the 60 predictions estimates a face as a part boundary face, this face will be considered a part boundary candidate (cf. Fig. 3.9). Still, the identified regions are big and more thinning needs to be done in order to obtain closed paths representing sets of edges between parts. Such paths can be easily extracted if all faces are completely labeled (each one assigned to a specific cluster), as boundary edges are then identified by having two neighbouring faces from different labels. Graph cut optimization [15] is employed to obtain this final face labeling. Graph cut optimization is a popular method in Computer Vision for tasks such as smoothing, segmentation or the correspondence problem. Essential to graph cut is that all entities (here: faces) will have costs associated to them for the explicit assignment to labels (unary term) and also edges (connections between neighbouring faces) will have costs based on whether the faces involved share the same label (smoothness term). Thus, the overall energy is formalized as

$$E(\overline{x}) = \sum_{f \in F} e_{\text{data}}\left(f, x_f\right) + \sum_{f,g \in N} e_{\text{smooth}}\left(x_f, x_g\right), \tag{3.1}$$

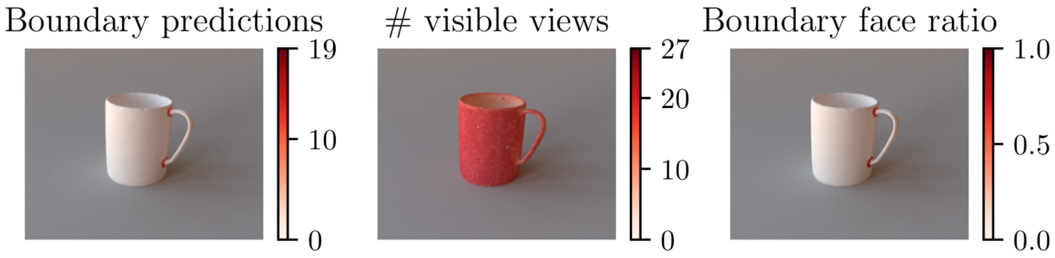


Figure 3.9: Left: Number of total boundary predictions for all faces after aggregating all 60 predictions. Middle: number of views where a particular face is visible. Right: Ratio of boundary predictions to number of visible views.

where x_f indicates the label assigned to face f. Based on the regions identified as in the previous paragraph, initially one (random) face of each of these regions is labeled with a unique cluster label. The unary term can be formalized as:

$$e_{\text{data}}(f, x_f) = \begin{cases} 0.0 & \text{if } x_f = x_{f_{init}} \\ 1000.0 & \text{else,} \end{cases} \tag{3.2}$$

where $x_{f_{init}}$ is the initial cluster label (if present). This term enforces that neighbouring clusters do not merge to a single one, as the cost for changing the label of the cluster prototype face is very high. The second term on the other hand enforces smoothness by punishing not identically labeled adjacent faces with a high cost.

$$e_{\text{smooth}}\left(x_f, x_g\right) = \begin{cases} 0.025 & \text{if } x_f = x_g \\ 1.000 & \text{if } x_f \neq x_g \end{cases} \tag{3.3}$$

The method of Boykov et al. [15] is used in order to obtain a solution to the graph cut optimization problem by means of an energy function (cf. Eq. (3.1)). This solution is a set of cuts through the edges of the constructed graph. These cuts will result in n isolated graphs, each representing a cluster. Thus, the final partition of all faces is achieved.

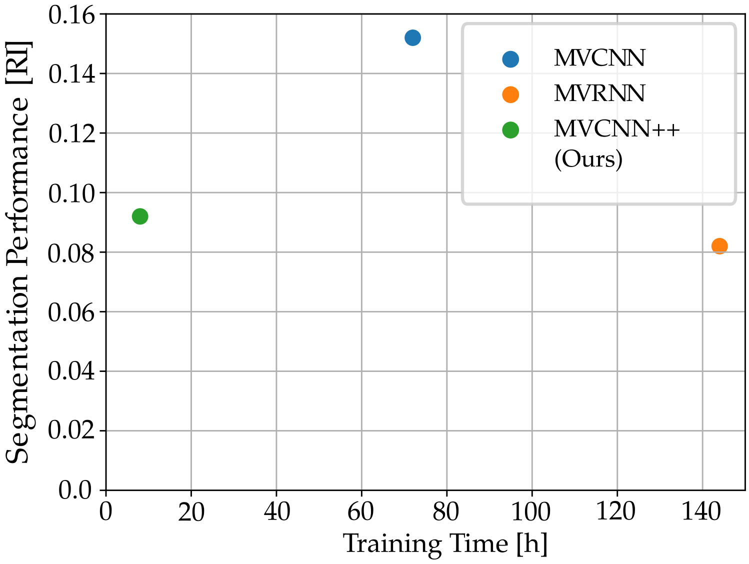


Figure 3.10: Training time and segmentation performance of the proposed MVCNN++ method compared to MVRNN & MVCNN as reported in [62]. Segmentation-Performance: RandIndex after [22] (lower is better).

3.4 Results

Benchmark

The presented approach is compared to other 3D mesh segmentation methods. The results are visualized in Fig. 3.12. It is important to note that several of these methods are supervised methods (MVRNN, MVCNN++, Shu2016, MVCNN) that use individual training and test splits. Therefore, the resulting RI may not be the result of all meshes but only a subset or an average over multiple sets via cross validation. Furthermore, many methods (namely RandCuts, NormCuts, FitPrim, RandWalks, KMeans) are not completely automatic but require the number of total segments as input. These methods are evaluated using a variety of possible values for the number of total segments and the results are averaged in order to compare them to fully automatic segmentation approaches.

Fig. 3.12 shows that recent approaches seem to even outperform average human segmentation results. This may be due to high variation in the level-of-detail in the segmentation task that was conducted for the original benchmark[22]. The performance of the proposed method is only surpassed by MVRNN. Due to its similarity to MVRNN and MVCNN, the proposed models' performance

Model	R	D	N	Res.	Rot.	Pre-trained	RI
DenseResNet18	✓	✓	✓	400x400			9.72
DenseResNet18	✓	✓	✓	128x128			12.74
DenseResNet18	✓	✓	✓	256x256			9.63
DenseResNet18	✓			256x256			9.92
DenseResNet18		✓		256x256			9.68
DenseResNet18			✓	256x256			9.45
DenseResNet18	✓	✓		256x256			9.65
DenseResNet18		✓	✓	256x256			9.16
DenseResNet18	✓		✓	256x256			9.58
DenseResNet18	✓	✓	✓	256x256	✓	✓	9.31
DenseResNet50	✓	✓	✓	256x256			9.29

Table 3.1: Results for different configurations of the MVCNN++. The best single-input configuration is achieved by normals, whereas the best paired input performance is achieved by depth and normals. Rotation augmentation during training and the use of a pre-trained feature extractor improved results in case of all three modalities used for input (9.63 to 9.31). The RI values here are multiplied by 100 for reporting more precision.

and training time is compared to these two approaches in Fig. 3.10. It can be seen that the novel method reduces training time significantly (8 hours) thus rendering it an attractive alternative to the slightly better-performing MVRNN (training time: 36 hours) and the MVCNN (training time: 72 hours).

Experiments

Experiments with varying model and training configurations have been conducted in order to measure the impact of the different features on the overall segmentation result. Particularly, the impact of varying image resolutions, different combinations of input features as well as various model designs is analyzed.

Image Resolution

Many of the meshes inside the MeshSegBenchmark dataset consist of several thousands of faces. When applying the boundary prediction technique to the rendered input images, the following condition is essential to the success of the technique: each face should occupy at least one pixel in at least one of the 60 views in order to be classified either as boundary or non-boundary face. When using low resolutions for the input images backprojection, not all faces may get the opportunity to be classified. This might lead to the well-known segmentation problem of "cluster bleeding": if the boundary contour between clusters is not completely closed (due to a face on the contour that was not classified as a boundary face), the involved regions may get merged to a single supercluster. For the original input image resolution in the MVRNN (and MVCNN), 128x128 was chosen by the authors. Low resolutions were found to lead to many open creases in the predicted boundary regions and subsequently to suboptimal segmentations. Therefore two further resolutions are employed: 256x256 and 400x400. As shown in Tab. 3.1, the overall segmentation performance is heavily impacted by the chosen resolution. Interestingly, there seems to exist a local optimum around 256x256 pixels, since the respective model performance (9.63) is slightly higher than the 400x400 variant (9.72).

Input Features

To investigate how well the individual 3D modalities may help in predicting the part boundary regions, the experiment was ran with different input feature configurations, namely R (*phong shading*), D (*depth*), N (*normals*) and all combinations of these three modalities: RN, RD, ND and RND. Comparing the performance of the resulting architectures may offer insights on how much information overlap there is between these modalities. Also, a group of features that may contain complementary information is identified, which is beneficial to provide in combination to the network. Regarding single modalities, the data in Tab. 3.1 support the conclusion that the normal map is the most helpful modality when it comes to predicting part boundaries. Interestingly, the phong shading seems to be the least informative modality, even outperformed by the depth map. Looking at the two-modality models, a combination of depth and normals lead to the best overall result (9.16), which may indicate that these two modalities hold relatively much complementary information that is important

	Benchmark	RandCuts	ShapeDiam	NormCuts	CoreExtra	RandWalks	FitPrim	K-Means	ConcAware	MV-RNN	MV-CNN	Shu2016 [103]	WC-Seg	MV-CNN++
Average	0.103	0.157	0.176	0.178	0.211	0.230	0.216	0.255	0.094	**0.082**	0.154	0.118	0.123	0.092
Human	0.135	0.158	0.179	0.182	0.225	0.295	0.171	0.203	0.119	**0.106**	0.196	0.116	0.128	0.127
Cup	0.136	0.224	0.358	0.236	0.307	0.334	0.409	0.439	0.099	0.100	0.100	0.096	0.171	**0.077**
Glasses	0.101	0.097	0.204	0.142	0.301	0.316	0.230	0.178	0.136	**0.066**	0.115	0.173	0.173	0.081
Airplane	0.092	0.115	0.092	0.186	0.256	0.261	0.166	0.227	0.079	**0.085**	0.157	0.150	0.089	0.095
Ant	0.030	0.025	0.022	0.047	0.065	0.068	0.086	0.131	0.019	0.021	0.044	**0.001**	0.021	0.016
Chair	0.089	0.189	0.111	0.093	0.187	0.167	0.214	0.213	0.054	0.051	0.078	**0.040**	0.103	0.051
Octopus	0.024	0.067	0.045	0.063	0.051	0.069	0.103	0.100	0.018	**0.022**	0.060	0.036	0.029	0.028
Table	0.093	0.374	0.184	0.098	0.244	0.139	0.200	0.379	0.062	0.072	0.091	**0.040**	0.091	0.049
Teddy	0.049	0.045	0.057	0.121	0.114	0.127	0.132	0.181	0.031	0.035	0.055	**0.024**	0.056	0.040
Hand	0.091	0.097	0.202	0.156	0.155	0.222	0.206	0.164	0.104	0.076	0.122	0.135	0.116	**0.070**
Plier	0.071	0.109	0.375	0.183	0.093	0.230	0.169	0.263	0.054	**0.054**	0.143	0.151	0.087	0.066
Fish	0.155	0.297	0.248	0.399	0.273	0.406	0.421	0.414	0.129	0.146	0.253	0.288	0.203	**0.125**
Bird	0.062	0.114	0.115	0.212	0.124	0.280	0.206	0.199	0.104	**0.059**	0.119	0.171	0.101	0.071
Armadillo	0.083	0.081	0.090	0.120	0.141	0.107	0.094	0.116	0.080	**0.060**	0.120	0.073	0.081	0.115
Bust	0.220	0.251	0.298	0.332	0.315	0.335	0.327	0.352	0.214	**0.162**	0.351	0.275	0.266	0.292
Mech	0.131	0.283	0.238	0.175	0.387	0.244	0.272	0.426	0.100	0.121	0.369	0.073	0.182	**0.065**
Bearing	0.104	0.129	0.119	0.179	0.398	0.271	0.234	0.260	0.097	0.080	0.104	**0.056**	0.122	0.065
Vase	0.144	0.160	0.239	0.268	0.226	0.287	0.289	0.401	0.160	**0.106**	0.216	0.212	0.161	0.140
FourLeg	0.149	0.177	0.161	0.189	0.191	0.208	0.183	0.195	0.133	**0.135**	0.213	0.140	0.152	0.195

Table 3.2: Comparison by RandIndex of different mesh segmentation methods by category. Note that *Benchmark* is the Human annotation.

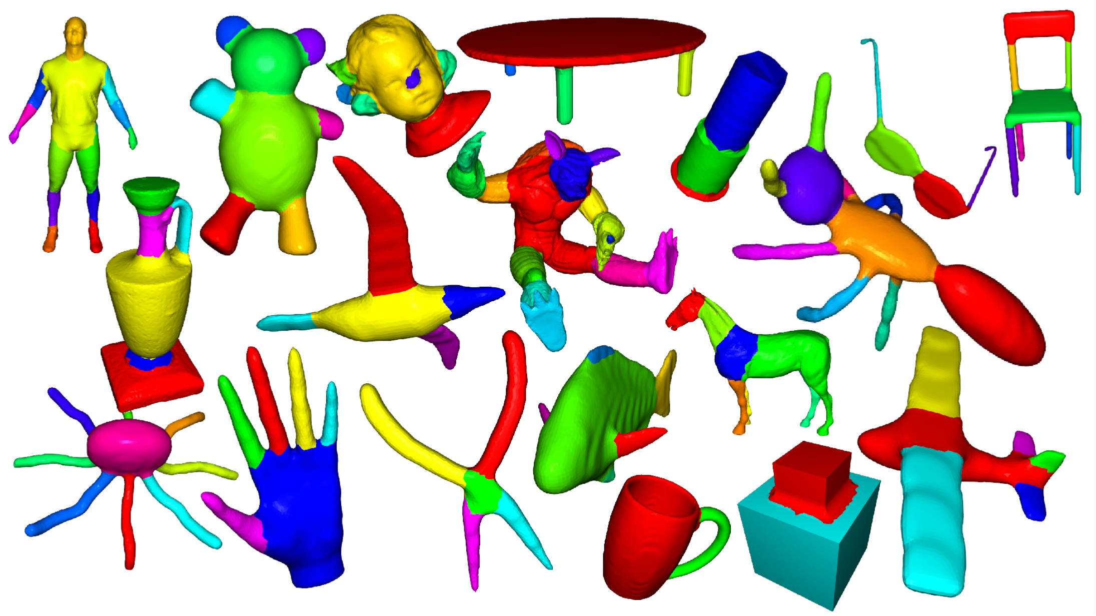


Figure 3.11: Qualitative results of the best model (Fully-Convolutional ResNet18, 256x256 pixel input, depth and normal data as input).

for part boundary prediction.

Models

Different sizes of the DenseResNet architecture were tested: DenseResNet18 and DenseResNet50 and also measure the impact of using pre-trained weights (provided by ImageNet) and data augmentation via random rotation of the training meshes. However, as shown in Tab. 3.1, these choices seem to only have a minor impact on the overall segmentation results.

Qualitative Results

In Fig. 3.11, segmentation results of the proposed method on the test data are shown qualitatively. The segmentations resemble the ground truth and major problems are usually encountered in object classes where other segmentation algorithms struggle as well: i.e. *fish* and *bust* often contain unintuitive partitions, which have a negative impact on the RI score. Also, the part boundaries of CAD-like models such as cubes or cylinders are not perfectly refined but sometimes cross neighboured part regions. The *teddy* class is often a challenge for

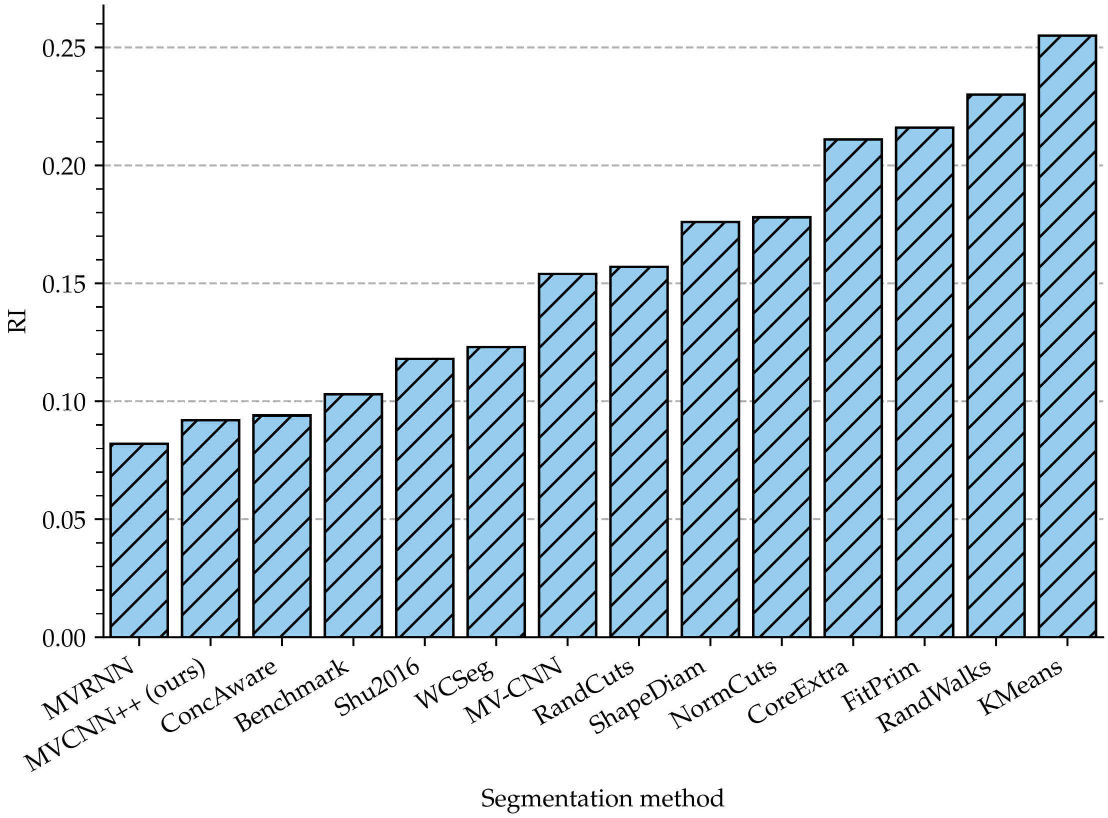


Figure 3.12: Segmentation performance comparison of various methods. Lower RI is better.

completely non-data-driven methods as the part boundaries are occasionally traversing convex regions (upper arms), but the proposed method manages to still identify part boundaries there. In Fig. 3.13, ground truth and predictions are compared. As visible in the *mug* and *octopus* mesh, due to the graphcut algorithm [15] applied on a very fuzzy boundary prediction, additional clusters may be predicted, leading to bad segmentation.

3.5 Conclusion

In this work, a 2D multi-view segmentation approach for 3D mesh was developed by identifying and addressing three main concerns of the existing MV CNN [62] approach. First, input resolution is crucial to the segmentation pro-

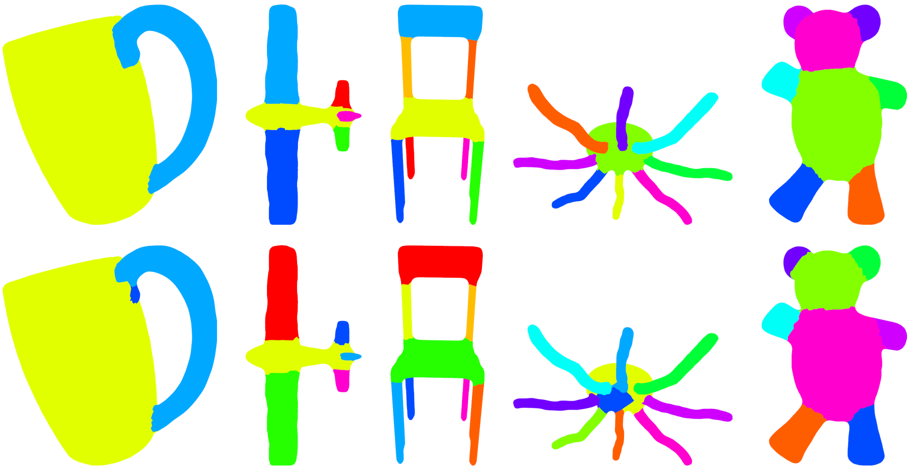


Figure 3.13: Ground truth segmentations and qualitative results. Upper row: GT, Bottom row: MVCNN++ (ours).

cess in the sense that higher resolution (up to a certain point) increases the performance heavily. Second, the choice of the HED architecture was shown to be a bottleneck, training time-wise and performance-wise. Employing the DenseResNet further enhances the model performance and lowers the training time dramatically. Third, enriching the input by providing additional modalities containing complementary information boosts the model prediction even further. Through these enhancements, the proposed method can achieve acceptable segmentation results without extensive training time. In the future, extending this approach by providing multiple segmentation proposals together with the already tested input modalities to the network can be investigated. We aim for a system capable of fusing multiple segmentation approaches this way, as these are often complementary. For instance, the SDF approach works well for humanoid objects and suboptimally on CAD-like objects; the FitPrim approach performs complementary to this. By considering multiple segmentation approaches, a best-of-both-worlds compromise may be found to obtain good segmentation results on all different object classes.

This chapter dealt with 3D mesh segmentation. The topic of segmentation reappears in the subsequent chapters as it is essential to bottom-up understanding of objects and the creation of part-graphs. In the next chapter, part segmentation information is utilized for object classification. Instead of an automatic segmentation approach as in this chapter, focus is on perfect segmentation information and the question whether it provides benefits for the classification task.

4

Concept Learning in 3D

4.1 Introduction

Humans are able to learn concepts - “building blocks of thoughts” as formulated in [72]. Learning such concepts has various advantages: often, only small amounts of examples are required in order to learn the representation of entities by means of concepts, which in turn generalize well to unseen data later on [61, 112]. For instance, a child may easily learn to perceive and recognize hand-written digits - without the need of thousands of examples. Contrary, modern data-driven deep learning approaches are considered data hungry and often require thousands of samples for accurate classification. A reason for this might be that current computer systems are often instructed to find discriminatory features in the input data instead of explicitly learning concepts.

In this work, a concept is defined as a set of rules. Each rule is made of symbols and their spatial relationship to each other. To remain with the handwritten

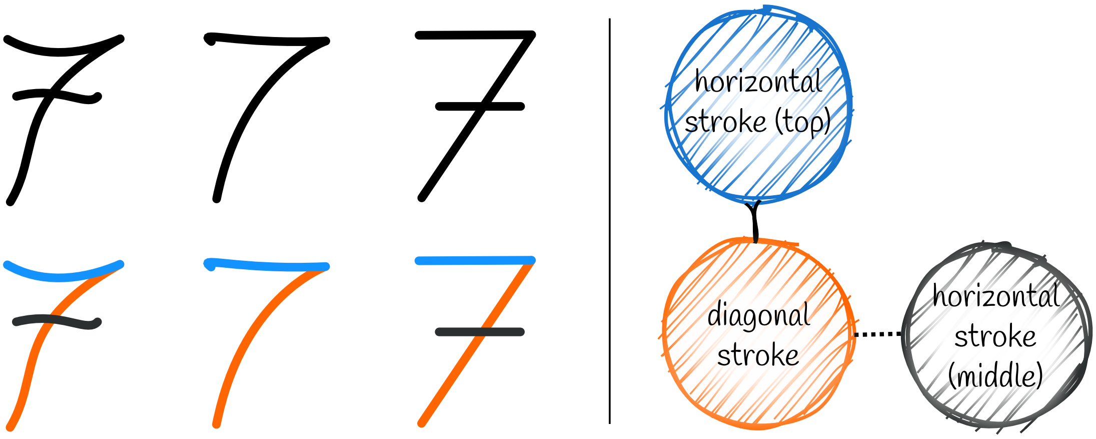


Figure 4.1: Instances of handwritten digit 7 (top), segmented (bottom) and entity graph as a prototype of a ruleset (right).

digit example: *seven* is usually represented by two or three strokes. Namely, a horizontal stroke on top, a diagonal stroke from top right to bottom left and optionally another smaller horizontal stroke in the middle (cf. Fig. 4.1). Each stroke in this example would be considered a symbol. The concept of the digit *seven* is composed of these symbols, represented by strokes, and their relation to each other. These kinds of concepts can be applied to many human-made objects. This is because human-made objects are often highly structured, following such rules, e.g. "A chair is a seat with a back and at least one leg.".

As mentioned before, current computer models will unlikely learn any concepts explicitly. Additionally, these systems can often be tricked by adversarial attacks [1, 16]. By only manipulating a small set of input elements (i.e. pixels), deep neural networks prone to these attacks may misclassify objects seen in 2D images. As the manipulation created by these attacks are often subtle, humans may not even recognize them. Similar to 2D models, 3D models can also be impacted by adversarial attacks.

One of the most widely employed 3D point cloud classification architectures is PointNet [85]. The authors of PointNet [85] found that often, only a small number of points inside the input cloud (called "critical points") is shaping the fi-

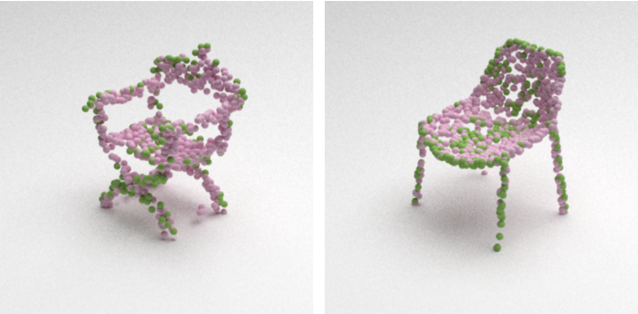


Figure 4.2: Visualization of critical points (green) and non-critical points (pink) of two point clouds evaluated on a trained PointNet architecture. Only the green points contribute to the final feature vector. Obfuscation of pink points would thus result in the same feature vector and classification result.

nal class prediction (cf. Fig. 4.2). This behaviour shows yet again the weakness of discriminative sub-symbolic learners: perturbation of small input regions (e.g. points in 3D) may lead to significantly altered predictions. The aforementioned practice may be detrimental to the overall classification performance of the model.

To summarize, methods that are able to learn concepts have at least the following two major advantages. First, the learned concepts may be utilized to improve the robustness of the classification system and - in general - may decrease data required for training. Second, systems with symbolic reasoning may provide more human-like explanation for decisions. For instance, attributing the classification prediction of an image on specific object parts instead of scattered pixels. One approach to steer current models into the direction of learning concepts is to make it easier to recognize symbols. A bottom-up approach can be used that first segments the input data in order to extract the individual symbols. If the model successfully learned the underlying class concepts, these symbols can later be utilized for the classification.

There are several questions that arise from this theoretical approach. Does segmentation indeed improve classification? Are current 3D classification methods already exploiting reoccuring subpatterns - symbols - to the fullest extent?

These two questions are tackled by creating a dataset that different models are benchmarked on. First, a symbolic dataset for the emulation of class concepts is developed. These concepts consist of spatial relationship rules between symbols. Our dataset provides not only symbolic data but also offers 3D point clouds (raw, and segmented) based on these symbols. Four classification models are employed on these three data modalities. The already existing PointNet and PointNet++ are chosen for raw point cloud classification. Additionally, two new models are designed: Graph-PointNet and GraphNet, working on segmented point clouds and symbolic data respectively. By comparing the performance of these four approaches, the impact of segmentation and efficiency of symbol extraction from 3D data is quantified. Lastly, how strongly certain parameters (i.e. size of symbol alphabet) may impact the overall classification is quantified as well.

4.2 Related Works

Symbolic & subsymbolic datasets
Symbolic data is especially present in Computer Graphics. As for games and simulations, systems need to keep track of objects, agents and other entities. Maintaining entity instantiation parameters such as position, orientation, pose and size for each dynamic object allows for renderings of entire 3D scenes. Recently, the machine learning community started to use games and game engines for creating synthetic datasets. Particularly for semantic segmentation tasks, game engines that render (instantiate) symbolic data are a major boon to current models [87, 89, 91], as the rendering looks more and more realistic.Secondly and more importantly: The process of automatic rendering of scenes makes the - otherwise required - task of human annotations of semantic labels unnecessary. Semantic labels can be trivially obtained by reprojecting the image pixels to the represented object, rendering costly human annotators redundant and saving hours of work. Third, the resulting semantic segmentations can be much more accurate compared to human semantic segmentation annotations which often suffer from simplified shapes or contours

such as polygons consisting of only several anchor points. With automatic semantic labeling of synthetic data on the other hand, a pixel-level granularity can be achieved. For classification tasks, there are few datasets that are using variations of subsymbolic representation. In [32], oversegmentation was applied to the popular MNIST dataset [63]. Thus, new data samples containing graphs of superpixels were created in order to use the obtained irregular data structure on Graph Convolutional Networks. Compared to pixel-level explanations, through the oversegmentation, more human-like explanations can be constructed. This stands in stark contrast to per-pixel scalars when using explanation techniques on CNNs, where consistency is often not achieved and neighbouring pixels may indicate opposing classes. Different from the here presented dataset, the Graph-MNIST dataset does not explicitly contain the underlying symbolic rules to generate new data but only the 2D instantiations.

Concept learning

In [61], Lake et al. introduce Omniglot - a dataset of handwritten characters. The dataset consists of over 1600 characters from 50 alphabets, drawn by 20 different people. The provided stroke data enables complete segmentation of individual strokes. Using this Omniglot dataset, Lake et al. compare one-shot classification performance of deep learners and their proposed Bayesian Program learner. Different from this approach, the here presented dataset is dynamically created based on randomly created concepts in 3D. In [18], Cao et al. develop a meta-learning strategy to create models that identify human-understandable concepts (here: animals) inside images. The identified parts are correlated with known prototypes and leveraged for few-shot learning on image classification. The authors selected the UCS Bird dataset [121], containing multiple instances of different birds. Different from their approach, the here presented dataset is working on artificially created concepts instantiated in 3D point clouds instead of 2D images.

3D classification using segmentation

In recent years, multiple approaches were developed that are making use of rudimentary segmentation information in order to more robustly classify given

3D data. PointNet++ [86] uses a kNN approach to group multiple points, thus creating local patches. Through multiple, hierarchical groupings, PointNet++ uses these local patches of varying granularity for point cloud classification. This stands in stark contrast to its predecessor PointNet, which instead focused on individual points instead of point clusters. However, this kNN clustering method only considers distances between points for the grouping. This may be problematic as desired local patches may contain outlier points that are emitted by other patches in local vicinity. The approach of Vincze et al. [122] uses a region growing approach to cluster 3D meshes into a fixed number of segments. Afterwards, a Graph Neural Network [56] is employed in order to classify the sampled 3D point clouds of the given mesh by considering the point clouds of the extracted segments. Vincze et al. report that their approach generalizes well from artificial to real-world data, as they benchmark their part-based approach against the classical PointNet [85] method. Although, Vincze et al. only use one cluster granularity instead of a more hierarchical approach, their grouping resembles the kNN strategy of PointNet++. Similar to PointNet++, semantic part boundaries are ignored and extracted clusters have no semantic meaning.

The aforementioned Computer Vision datasets are using 2D images as not much research on concepts in 3D is conducted yet. Here, we want to fill the void and create an abstract and adjustable dataset with the additional feature to instantiate individual samples in 3D.

4.3 Methods

As the dataset will serve as input to the models, the dataset generation process is described in this section. Afterwards, the four individual models are introduced: PointNet, PointNet++ and two proposed models - GraphNet and Graph-PointNet.

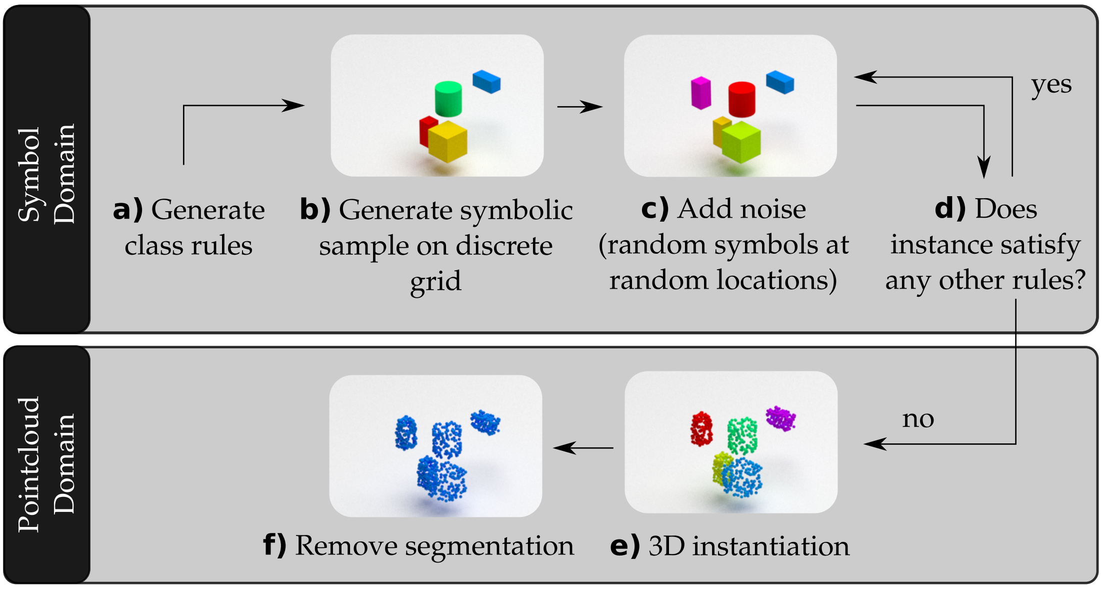

Symbol Domain
a) Generate class rules
b) Generate symbolic sample on discrete grid
c) Add noise (random symbols at random locations)
d) Does instance satisfy any other rules?
yes
no
Pointcloud Domain
f) Remove segmentation
e) 3D instantiation

Figure 4.3: The process of creating the dataset. a) Rules are generated by sampling random pairs of symbols and one random spatial relationship (“above”, “below”, “left”, “right”, “in front of”, “behind”). b) On a discrete grid, a class concept can be instantiated by sampling coordinates of the symbols involved on a discrete 3D grid. c) Noise can be added by inserting random symbols at non-occupied coordinates. d) Through noisy symbols, other class concepts may also be satisfied by this sample. The sample has to be newly generated in case it satisfies any other class concept. e) Each symbol is transformed into a point cloud and added to the global cloud of the current sample. f) After removing the symbol instance segmentation, a raw cloud is obtained.

4.3.1 Dataset Generation

The dataset developed here contains two important features. First, the designed dataset offers multimodal data. In addition to symbolic data samples, the dataset is able to generate 3D point clouds from these samples. Two different types of point clouds can be obtained: raw, unsegmented clouds as well as instance segmented clouds, where each point is mapped to its respective symbol. The second major feature of the dataset is its parameterization: the most important variables inside the dataset are adjustable. This enables individually tuned datasets for different desired scenarios. Parameters of the dataset include variables such as: number of symbols used, number of rules used for each class

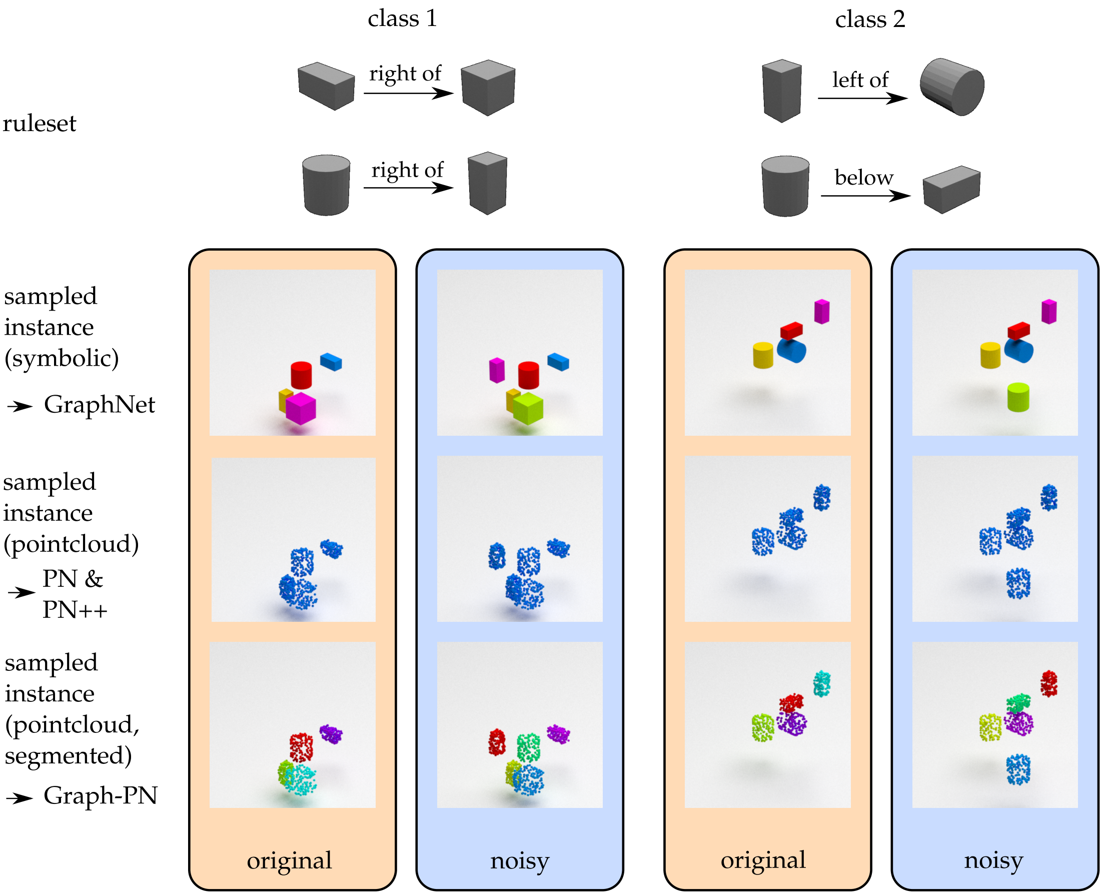


Figure 4.4: Visualization of randomly created class concepts and instantiations thereof. First row: ruleset (concepts) of class 1 and class 2. Second row: symbolic data instances that adhere to the rules. Third row: raw point cloud generated from rules, possible input to PointNet and PointNet++. Fourth row: instance-segmented point clouds, input to Graph-PointNet. "Noisy" columns contain one additional random symbol that is added to the scene.

concept, number of class concepts, and the resolution of the discrete grid. The entire dataset generation process is compartmentalized into an initial symbolic class concept generator (cf. Fig. 4.3 a and b), a noise adding module (cf. Fig. 4.3 c) and a 3D instantiator module (cf. Fig. 4.3 e and f). Additionally, an instance validation module (cf. Fig. 4.3 d) is needed to test whether given instances indeed only satisfy the concepts of the target class and no other class.

Symbolic class concepts

In the dataset, any class concept can be represented by a set of rules. A rule is composed of a pair of symbols and their spatial relationship. For each rule, the pair of symbols is randomly chosen from all possible symbols - the symbol alphabet. The spatial relationship is randomly sampled as well from the set "above", "below", "right", "left", "in front of", "behind". Given the desired number of classes and the desired size of the symbol alphabet, rules for each class have to be generated. Provided with these parameters, a specific number of rules is added to each of the classes. Next, it has to be checked whether through the above mentioned process of rule aggregation, identical concepts in different classes were created. If this is the case, class concepts have to be resampled until all classes have disjoint sets of rules.

Symbolic data instantiation

From the created class concepts, concrete symbolic data can be instantiated. For this, a discrete 3D grid is used to assign the involved symbols to the coordinates of the grid. To instantiate a given class concept, all symbols' coordinates are sampled from the discrete 3D grid. In case any of the class concept rules cannot be satisfied by the choice of the randomly sampled symbol coordinates, these symbol coordinates are resampled. Furthermore, it has to be validated whether other class concepts may be satisfied by the symbol coordinates. In this case, all symbol coordinates have to be resampled. Additionally, a predefined number of randomly chosen symbols can be added to the sample during this step to emulate noise in the data.

3D point cloud instantiation

After sampling symbolic data instances from each class concept, 3D point clouds can be generated from these symbolic data instances. First, the symbols are transformed to point clouds for later use as input in 3D classification models. Therefore, a 3D point cloud representation of each individual symbol is required. Here, distinct primitive 3D shapes are chosen: a sphere, a cuboid, three cylinders and three boxes. The cylinders and boxes differ in orientation and thus vary in appearance, such that models have the opportunity to distin-

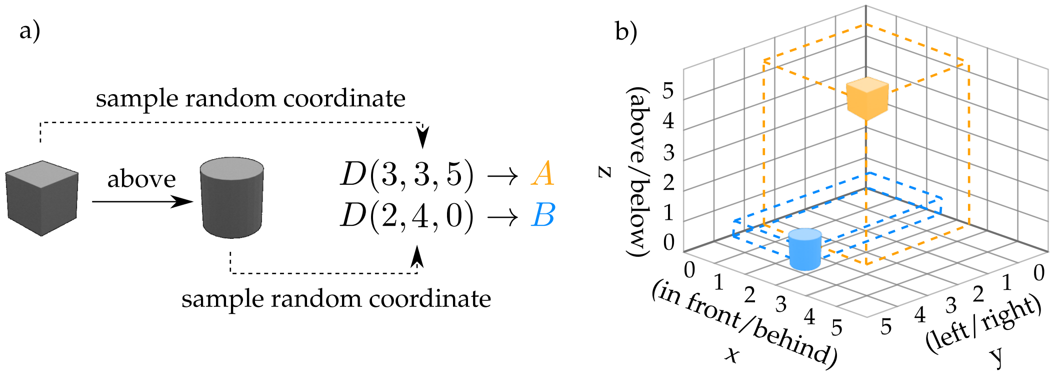


Figure 4.5: a) rule and concrete rule instantiation. b) 3D instantiation.

guish them based on their shape. For each symbolic data sample, the abstract symbols are substituted by the aforementioned shapes and 256 points are sampled from each of these shapes to obtain the final 3D point cloud transformation of the original symbolic data sample. The designed dataset offers both raw, unsegmented global 3D point clouds for each symbolic data sample, as well as instance segmented 3D point clouds.

4.3.2 Benchmark models

Here, the four individual models are introduced: PointNet, PointNet++ and two proposed models - GraphNet and Graph-PointNet.

PointNet

PointNet is one of the most widely employed 3D classification models in recent years. The model uses 3D point clouds as input. In the original work [85], the authors propose two architectures: a classification network and a semantic segmentation network. This section focuses on the classification model proposed by Qi et al. [85]. One of the issues with supervised 3D point cloud classification is that the resulting process should be permutation independent. This means that the order in which the points are fed into the system should not impact the final prediction. Therefore, PointNet uses a Max-Pooling operation on extracted features of all individual points, as the *max* operator is considered a symmetric function. For symmetric functions, the order of input elements

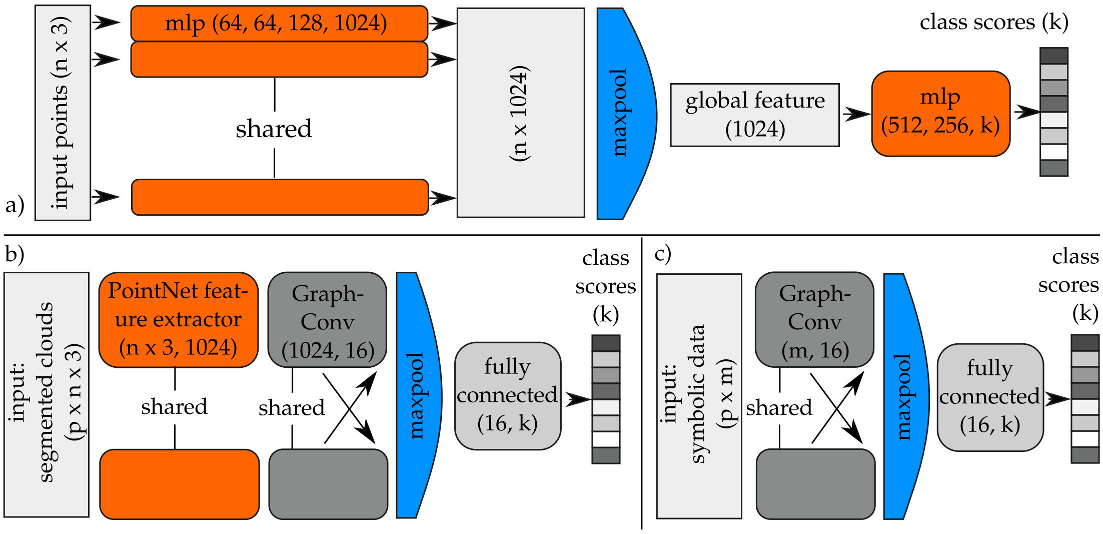


Figure 4.6: Architectures of a) PointNet, b) Graph-PointNet and c) Graph-Net.

does not impact the results of the function. Starting with X, Y and Z coordinates of the cloud, each point is first fed into a sequence of MLPs of varying sizes (cf. Fig 4.6). These MLPs share their weights over all input points, meaning that each point of the input cloud is fed into the same MLP network. After a 1024 element feature vector of each point has been extracted, the aforementioned Max-Pooling reduces the point cloud feature matrix to a global feature vector. For classification, an MLP is used to compute the final class predictions of the point cloud based on the extracted global feature vector. Due to high scores on many benchmark datasets [19, 124, 131] and easy implementation, PointNet was quickly adapted into many Computer Vision pipelines and further improved by numerous extensions [2, 79, 86].

PointNet++

PointNet++ adds several improvements to the classic PointNet in order to boost classification and segmentation performance further. Firstly, one of the biggest issues of PointNet is that each point is processed individually without considering its local neighbourhood. If the point cloud is sampled from a 3D surface, points in proximity of a specific point often describe the local surface and may contribute to important local geometric information. In PointNet++, prototype points sampled in a farthest-point manner are selected and points in proximity of these prototypes are collected, creating multiple small clusters on the input cloud. Then, each of these clusters is fed into a PointNet backbone. Although PointNet usually captures global shape features, through multiple local point set queries, patch-wise, local shape information is acquired. Whereas PointNet++ shows superior performance compared to PointNet in many benchmarks [19, 117, 124], there is still room for improvement. The neighbourhood clustering around prototype points in PointNet++ can be seen as a segmentation of the overall object, similar to the methods discussed in Chapter 3. However, compared to part segmentation, the clustering method in PointNet++ does not consider any semantically meaningful information, but only uses the criteria of proximity to create clusters. This will result in points influencing (feature vectors of) clusters from neighbouring *parts* of the object. As outlined in Chapter 5, suboptimal segmentation (including under- or over-segmentation) might have a negative impact on the classification performance. Therefore, one way to possibly boost performance would be to exchange the clustering method with a more advanced segmentation technique.

Graph-PointNet (G-PN)

To investigate the benefit of working on symbols instead of subsymbolic entities, a pipeline for symbol extraction is needed. A PointNet backbone is used, which runs on the 3D input, but additionally leverages a segmentation mask that clusters the point cloud into separate entities. For segmentation, the ground truth segmentations of the symbols are used. It is not trivial to obtain such perfect segmentations in real world scenarios (i.e. object classification). However using these segmentations allows for measuring the performance gap between suboptimal segmentations (in PointNet++ via kNN) and optimal symbolic segmentations. Due to the irregular nature of the symbolic representation of the dataset (data samples may vary in the number of symbols they contain), Graph Convolutional Networks [56] are used to process the acquired symbol-like entities and to reason about the final class. In fact, a single GraphConv layer [75] is employed to propagate the individual node information to all neighbours. The output of this layer is a 16-element vector for each part. After aggregating these vectors via Max-Pooling, the output class is predicted using a single fully connected layer.

GraphNet (GN)
Since the designed dataset implicitly works directly with symbolic entities, the upper limits of Graph-PointNet are investigated, given the correct symbol is already extracted. For this particular model, raw 3D data or 3D data with additional segmentation information is not provided but rather the underlying symbolic state. First, the occupied cells are selected and represented as nodes of the input graph. Each node thus may be represented as a (one-hot) symbol vector. In a one-hot vector, all entries are 0 except for the element at index p. With these vectors, categorical data can be represented. Additionally, the discrete coordinates of the symbol is appended to the state vector of each node. A graph of all nodes is constructed by creating edges between all possible pairs of nodes. The developed model contains a GraphConv layer [75] to process the symbols of the grid, which are represented as nodes of the input graph. After the GraphConv layer, the results of all symbol nodes of the graph are aggregated via Max-Pooling. The resulting global feature descriptor is leveraged for classification, utilizing a single fully connected layer.

4.4 Experiments

Several experiments are performed, differing in the dataset creation parameters on all four methods. Default values for the datasets are: 4 symbols, 6 classes, grid resolution of 6 and 2 rules per class. The number of additionally introduced noise symbols per data sample is set to 1. For each individual experiment, 5 random instances of the artificial dataset are created. For each dataset in each class, 1100 instances are sampled. 1000 instances per class are used for testing and 100 instances per class for training. The training data is provided to each network in a batch size of 32.

10% of the training data was separated as a validation set during training to monitor the models performance. Each model was trained for 100 epochs or

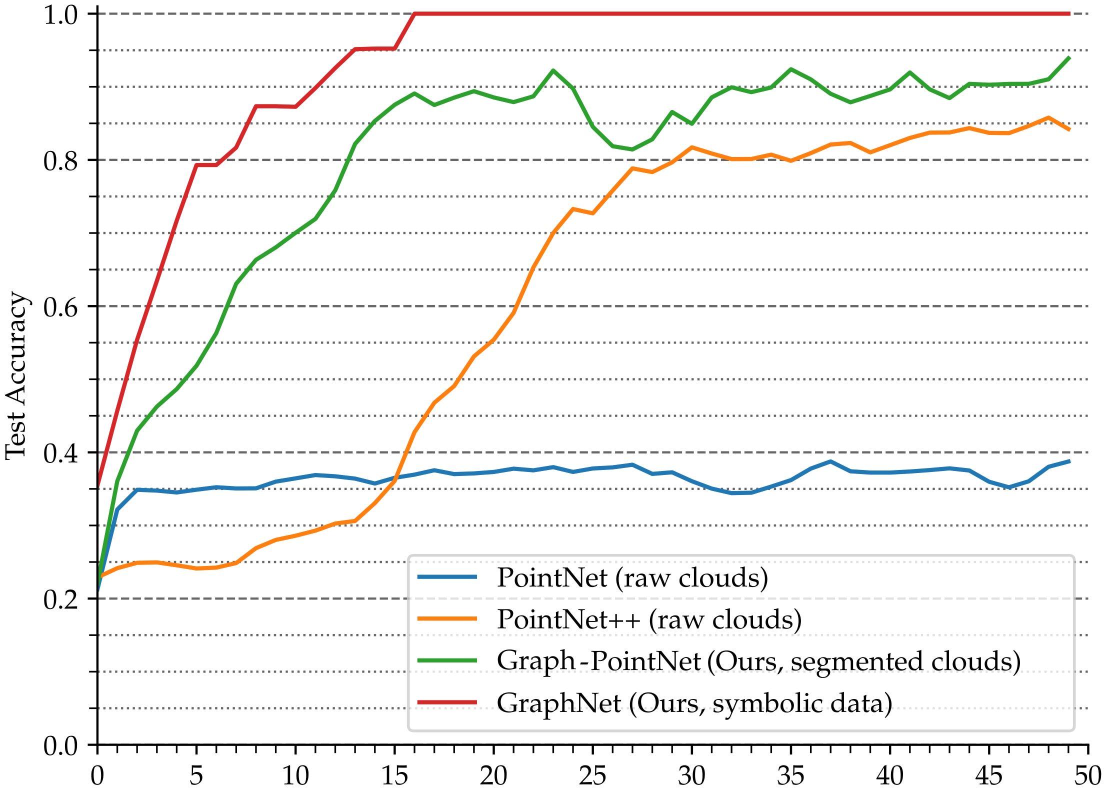


Figure 4.7: Performance of the benchmarked models on the artificial dataset for single run. As expected, GN quickly saturates at 100% accuracy, as purely symbolic data is used as input and the only challenge is to filter out noise. PointNet stagnates below 40% accuracy. PointNet++ needs many epochs to reach an accuracy around 85%, whereas Graph-PointNet - provided with ground truth segmentations - saturates around 90%.

until the loss on the validation data did not increase anymore for 10 consecutive epochs. Default values for each parameter were set such that in each run of the experiment, only a single parameter is changed to observe its impact on the model performances. All models were implemented via PyTorch and PyTorch Geometric [31, 81] and trained on an Nvidia GeForce GTX 1080 Ti GPU using CUDA [76]. For easy comparison, the Adam optimizer [55] was chosen with a learning rate of 0.001 and the negative log-likelihood as the loss function. To gain insights into which entities each model relied on for the final class prediction, the Integrated Gradients (IG) method [109] was employed. With this method, important attributes of the input data on the final prediction can be

parameter	values	PointNet	PointNet++	GraphPointNet	GraphNet
symbols	4	45.3	86.1	92.4	99.3
	6	41.7	83.9	94.2	99.1
	8	47.8	79.2	89.9	98.4
classes	4	57.2	91.6	96.0	99.8
	6	42.6	83.4	89.9	98.7
	8	44.1	83.2	92.7	99.3
grid resolution	**6**	39.0	83.4	92.1	99.8
	8	40	86.9	83.3	100.0
	10	41.6	71.1	78.6	97.9
rules	1	39.2	96.7	86.5	100.0
	2	45.2	90.6	93.6	98.9
	3	36.1	68.1	95.5	99.0

Table 4.1: Accuracy (in percent) on test data for different configurations of the dataset for the benchmarked methods.

visualized.

4.5 Results

In Tab. 4.1, the results for the four different methods with different dataset configurations are shown. In general, the performance of each model varies strongly for each run. One underlying condition that enables this behaviour is the fact that the randomly created class prototypes share varying numbers of common rules with each other across the different runs. More disjoint rulesets lead to less classification confusion in general, whereas the more sets of rules overlap, the higher the confusion. In most cases, the GraphNet achieves nearly 100%. This is expected, as - by design - the input data of the GraphNet method is already symbolic and each symbol's position is provided explicitly as well. In Fig. 4.7, the test accuracy is tracked over the training epochs for each method. As expected, when provided with the symbols directly, the model (GraphNet)

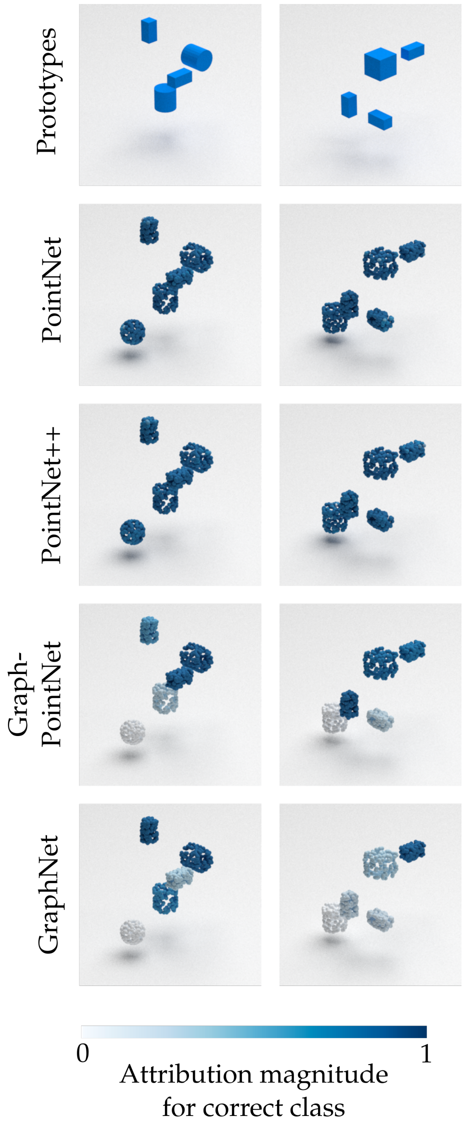


Figure 4.8: Explainability via Integrated Gradients. Here, two different classes are used, having four symbols as prototypes each. A single noise symbol is added in each instance. For consistency, results of GraphNet are mapped to point clouds, although symbolic data is used here. In both Graph-PN and GraphNet, the noise symbol has less impact on the final class compared to PointNet and PointNet++.

quickly reaches perfect accuracy. The performance of PointNet here saturates at around 35-40%. Whereas PointNet++ needs a long time (approximately 20-30 epochs) to saturate, Graph-PointNet reaches accuracies of 80% and beyond much faster (already after 15 epochs).

Performance on varying Symbol size

Increasing the number of symbols that are considered may have at least the following two effects. On the one hand, individual classes might be easier to distinguish from others due to discriminatory symbols that uniquely exist in only one/few of all classes. This might intuitively ease the task of classification. On the other hand, when instantiating additional symbols, the chance of confusing any of these symbols with each other increases. The number of symbols seems to have only a minor impact on the overall classification performance. PointNet performs worst, as it correctly classifies only 41.7%-47.8% of the samples across the three settings. PointNet++ performs better than PN, however a performance gap compared to G-PN exists. The performance gap between PN++ and G-PN increases with growing alphabet size. As mentioned before, the negative impact of an increased number of symbols in case of PN++ might be due to more likely confusions of these entities. G-PN is provided with correct segmentation of all entities, thus reducing the negative impact of an increased alphabet size.

Performance on varying number of classes

Principally, increasing the number of classes will make the classification problem harder, as more classes are introduced that can be confused and may even share some common rules with other classes. Whereas performance of PN++ strongly decreases for 6 and 8 classes, the resolution impact on Graph-PointNet is lower. GraphNet consistently achieves high classification accuracy.

Performance on varying grid resolution

Varying grid resolution is an important parameter, as increasing grid size enables more instances for each class to sample from. In smaller grids - with appropriate training samples - all possible spatial configurations between the prototypes might be extracted and learned. In more complex grids, the number of possibilities explodes. To succeed, classification methods need to leverage the spatial relations between entities to generalize to the test data. Similar to varying numbers of symbols and classes, superior performance using Graph-PointNet is observed compared to PointNet++ and PointNet.

Performance on a varying number of rules per class

For this setting, the change in performance is significant for PN++ and Graph-

PointNet. Additionally, while the increasing number of rules seems to improve the classification performance of Graph-PointNet, PointNet++ performance decreases strongly. An increased number of rules per class implies more entities overall inside the grid. Thus, with an increasing number of symbols inherent to the prototype, the signal-to-noise ratio increases as the total number of noise symbols is fixed.

Attribution
Fig. 4.8 shows the original prototypes of two samples without any noise (first row). In order to qualitatively check the predictions made by the different systems, Integrated Gradients are used to visualize important attributes of the input for each method. As Fig. 4.8 shows, the attribution of PointNet and PointNet++ is very diffuse: a lot of points - dispersed over all entities - seem to be attributed to the network's prediction. On the other hand, as Graph-PointNet uses ground truth segmentations, each symbol will be attributed isolated from each other. This leads to the opportunity to detect and visualize the noisy symbol inside each sample in an "odd-one-out" manner. Similarly, GraphNet is able to achieve this detection of noisy symbols as well (cf. Fig. 4.8 bottom row).

4.6 Conclusion

In this chapter, a novel classification dataset generation method has been introduced, consisting of symbols adhering to specific spatial rules. The dataset is easily tunable to create new data and it provides methods to transform the discrete, symbolic data into 3D point clouds. Additionally, the dataset provides multiple modalities such as raw point clouds and part-segmented point clouds, where points emitted by the same symbol obtain a common label. With these modalities the impact of perfect segmentation compared to a raw point cloud on 3D classification approaches was investigated. PointNet only considers the global point cloud and showed shortcomings in the classification task. This is consistent with the expectation that if symbols are not learned, small amounts of training data will not lead to good overall generalization in this classic approach. PointNet++, which uses a primitive automatic segmentation strategy improves the performance further, offering empirical evidence that segmentation indeed helps in this classification task. The proposed Graph-PointNet method uses ground truth segmentations and improves on PointNet++ results in most of the scenarios. To summarize the findings, most importantly, a strong accuracy boost in classification is observed when leveraging the perfect symbol segmentations. These results show that designing accurate segmentation algorithms in the future for employing in classification tasks may be critical for tasks where the underlying data is known to adhere to components in specific spatial relations.

From the observations in this chapter, an important question emerges: are real-world objects structured enough such that part-based methods may improve classification compared to global methods? This question is explored in Chapter 5 by means of part-graph decompositions of more realistic 3D objects compared to the abstract rule-based data in this chapter.

5

Part-graph-based Object Classification

5.1 Introduction

Object classification is crucial for interacting with our environment in our everyday life. But also many robotic applications depend on the recognition and subsequent classification of objects. In autonomous driving and autonomous robots, classification methods are employed to categorize (parts of) the visual input. If classification fails, e.g. a traffic sign is recognized but misclassified as a tree, the consequences may be dramatic when considering autonomous cars. Light detection and ranging (LiDAR) devices have become cheaper in recent years, and with 3D sensors offering additional depth information compared to 2D camera systems, these devices become more and more attractive for the task of object classification. Due to the shift in input modality from 2D to 3D data, new approaches for classification have to be found, optimally approaches that leverage the depth information and geometry. Feature extraction mechanisms are a popular method to reduce objects to a fixed vector or descriptor, on which subsequently classification algorithms can be employed. Earlier variants of features were hand-designed, i.e. specific algorithms were developed to obtain desired shape characteristics [97, 123]. Whereas at the start of these approaches, smaller regions of the overall shape are evaluated, at the end, all information is aggregated in order to create one single fixed-size vector or matrix (i.e. by using histograms). This aggregation stands in stark contrast to the fact that the shape (and its characteristics) may vary from region to region on a single object. Neglecting this intra-object shape-variance may lead to poor classification performance due to less nuanced shape descriptors. In the field of psychology, Biederman postulated his Theory of Recognition-By-Components [10]. Biederman postulates that object recognition by humans starts with separating the target into its parts and reducing each part to a certain primitive volume called "geon" such as cubes, spheres and cylinders. Biederman suggests that subsequently, using these geons and their spatial relation to each other, the final object class is inferred.

In this chapter, a pipeline is proposed that follows this process of object classification. The proposed method will first segment the 3D mesh into its parts and extract feature descriptors from each part individually. Afterwards, graph kernels are used to train a model that can handle the unstructured data at hand. The method is tested on different segmentation methods, feature descriptors and graph kernels. Additionally, the method is benchmarked against classification via global 3D feature extraction.

5.2 Related Works

Initial 3D classification methods used hand-engineered feature descriptors that make use of certain properties of the geometry of the input [97, 115, 123]. Most methods focus on point clouds as the 3D input modality. Using randomly sampled combinations of points from the cloud, distance profiles of point pairs as well as areas and angles between point triplets can be calculated, as in the case of ESF [123]. These profiles are usually aggregated by histograms [97, 115, 123]. Thus, mapping each measurement to a discrete bin and normalizing the resulting frequencies, a final global feature descriptor can be obtained. These feature descriptors can be used in a supervised manner in combination with a classification model such as k-nearest neighbour (kNN) [25], decision trees [88] or SVMs [13].

More recent methods [84, 85, 86] leverage the power of Deep Learning to avoid predefined explicit feature descriptors and instead try to dynamically learn a compact representation of the input. Maturana et al. [73] discretized the input into volumetric primitives - "voxels" - in order to apply 3D CNNs to the data, a method which was very successful in 2D image classification. In PointNet [85], the model extracts a 1024-element feature vector from each point of the point cloud individually and subsequently applies Max-Pooling to obtain a global point cloud descriptor. This representation is used inside a classification head that mostly consists of Multi-Layer-Perceptrons (MLPs) that map the global descriptor to an output vector where each element represents one class. These networks are trained via Backpropagation [93] to minimize the prediction error on the training data.

The MVCNN approach [107] aims to render images from the input mesh from different perspectives and subsequently employing a 2D CNN on each view. This way, for each view a descriptor could be extracted. Descriptors from all views are then Max-Pooled to obtain a single descriptor that can be processed by a classification header - similar to the one in PointNet. Recently, FoldingNet [130] was proposed. FoldingNet follows an auto-encoder structure which consist of an encoder and a decoder module. The input point cloud is passed through the encoder network to be reduced to a 1024-element descriptor. In the decoding stage, based on this descriptor, the original point cloud is tried to be reconstructed. Based on the reconstruction error given as e.g. Chamfer Distance [7] or Earth Mover's Distance [92], the two modules are optimized during training, such that the intermediate representation holds a lot of shape information and characteristics about the input. This method does not explicitly contain classification capabilities but can rather be seen as a learnable feature extraction method. However, the authors propose the use of a SVM which is trained on the extracted feature descriptors of the point clouds inside the training set. Thus, the method is extended to 3D object classification in a two-stage manner.

5.3 Methods

Instead of direct global aggregation of the 3D shape to obtain a feature descriptor for each object, a multi-stage pipeline is proposed: First, the object, represented as a mesh is segmented into its parts. For each part, a part-mesh is automatically obtained at the end of this stage. Second, an individual feature descriptor is extracted from each part of the object. Part meshes are uniformly sampled to obtain part-specific point clouds to extract features from. Next, a part graph is constructed with nodes representing parts and edges representing spatial neighbourhood between parts. Subsequently, graph kernels are employed to calculate similarities between graphs from the training data and the currently evaluated object's part graph. Last, the similarity of the part graph at hand to all training samples is fed to the SVM to obtain a final prediction of the object class. In the following, each step will be explained individually in more detail.

5.3.1 Preprocessing

First, the mesh is transformed such that it is centered at the coordinates $(0|0|0)$ and it is completely contained inside a unit sphere. The segmentation method that is employed later on requires the mesh to be watertight. This means that the mesh needs to be clean, enclose a volume and not contain any topological inconsistencies. To make the input mesh watertight, the method from Huang [46] is used. Similar to the popular Marching Cubes method [70], Huang's approach [46] voxelizes the input space and extracts isosurfaces. After applying this method, the input mesh is a watertight mesh without degeneracies.

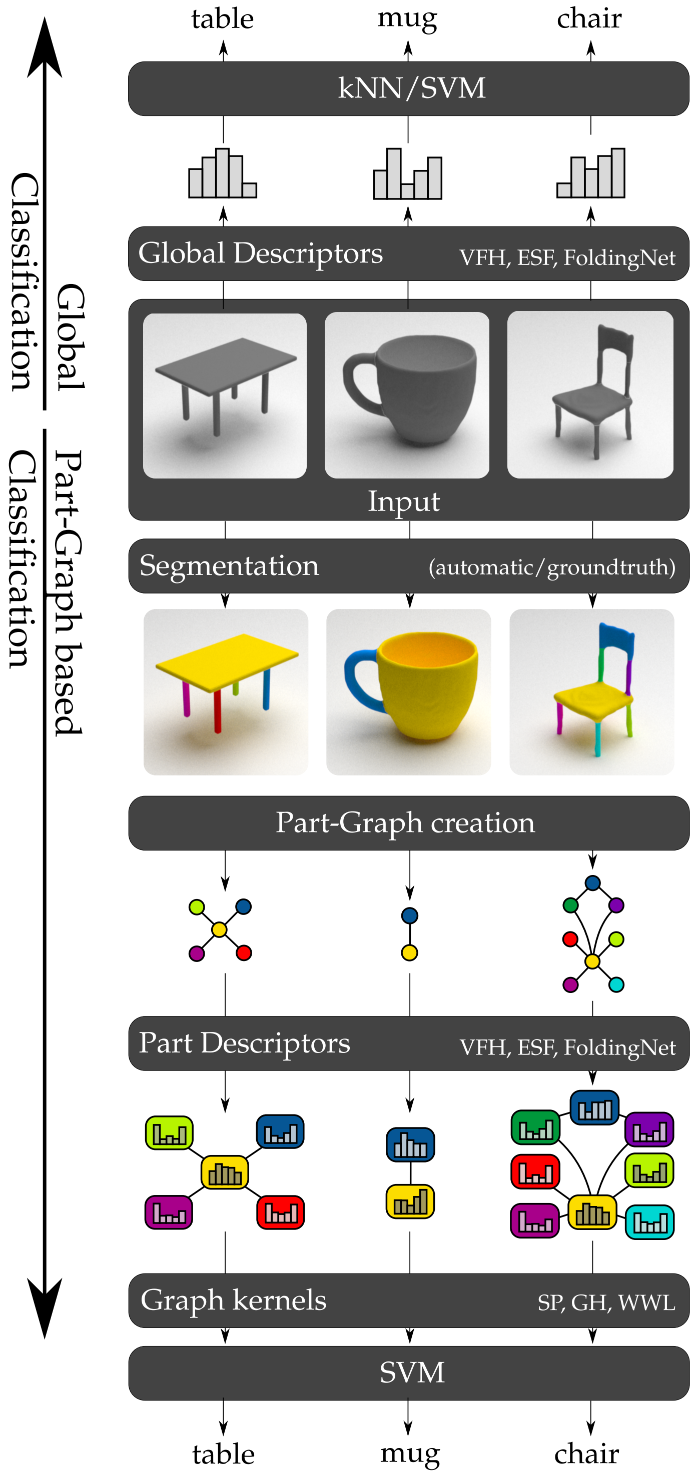


Figure 5.1: Overview for global descriptor pipeline and the part graph pipeline. For the former, global descriptors are extracted from the input shapes (here either VFH, ESF or FoldingNet), which are fed to the SVM. For the part graph pipeline, the input shape is first decomposed into its parts, using ground truth or automatic segmentation. Subsequently, the part graph is created and the parts' descriptors are extracted (again, either VFH, ESF or FoldingNet descriptors). Finally, graph kernels (ShortestPath, GraphHopper or Wasserstein Weisfeiler-Lehman) are used to compute the kernel matrix between all graphs to train a SVM. Figure adapted from Teich et al. [111].

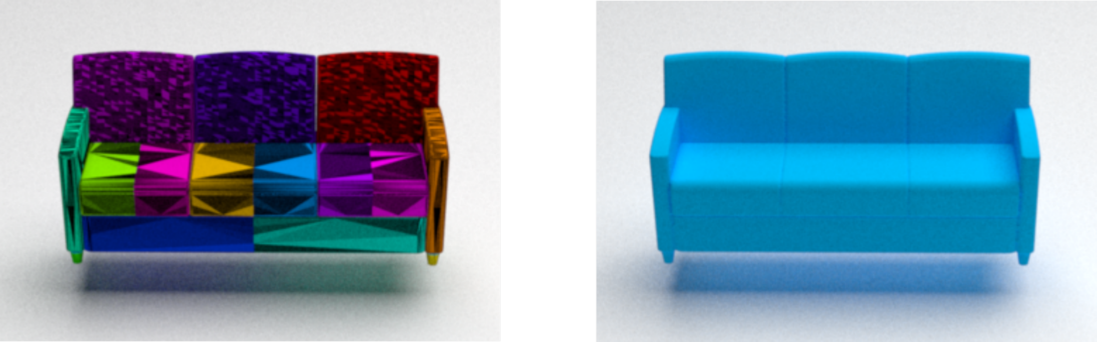


Figure 5.2: Left: Rendering of raw 3D data from the ShapeNet dataset. Individual components have different colors. Different shades indicate degeneracies such as inconsistent face orientations of adjacent triangles. Right: processed mesh, cleaned, watertight and only consisting of a single component.

5.3.2 Segmentation

For the segmentation algorithm, the Concavity-Aware method from Au et. al [5] has been implemented. The Concavity-Aware method analyses features of isolines on gradient fields across the 3D mesh shape in order to select several of them as part boundaries. The idea behind this approach is to first identify important points on the shape's surface that are far away from the central region of the surface. Then, the problem of finding part boundaries can be reformulated into a heat diffusion problem. Assuming that convex regions on the shape's surface propagate heat efficiently and concave regions inhibit heat propagation, a gradient field between pairs of the above mentioned critical points can be created. Isolines across these gradient fields represent possible part boundaries such that heuristics can be used to identify the most likely part boundaries and obtain the final segmentation of the mesh in turn.

5.3.3 Feature descriptors

Two hand-engineered descriptors [97, 123] as well as one data-driven feature descriptor [130] are considered. In this section, the individual descriptors are explained. As all three feature descriptors are only applicable to point clouds, point cloud extraction for each part individually is needed. Therefore, 1000 points are uniformly sampled from each parts' surface.

VFH
The Viewpoint Feature Histogram (VFH) is an extension of the FPFH descriptor [96], which in turn builds on top of the PFH [95]. The PFH additionally requires normal information of the points inside the point cloud. For all points of the cloud, the neighbourhood is analyzed by always considering the current point and one of its neighbours. Their normals' angular distance can be described by three parameters, each of which is binned into a separate histogram. The fourth histogram captures the euclidean distance between the coordinates of the two points. FPHF prunes the number of considered neighbours for each point to make the algorithm faster. For the VFH descriptor, viewpoint information is additionally considered for the histogram creation. The implementation of the VFH inside the Point Cloud Library [94] returns a 308 element feature vector for each point cloud.

ESF
The ensemble of shape functions (ESF) [97] captures the following point cloud properties: distance, angles, and area. For the area property, three random points are sampled from the point cloud and the triplet is categorized as either lying completely inside the surface, completely outside/on the surface or partly inside and partly outside. For each of these three categories, an individual histogram is used. The squared surface area that the currently sampled triangle creates is then added to the appropriate histogram. This sampling of triplets is repeated either for a fixed number of iterations, which should be a high number to capture representative values for the area characteristic of the point cloud. Analogous to the area property, the same method is employed for distances (where pairs of points are sampled) and angles (where triplets of points are sampled). Finally, a tenth histogram is created that captures the relative length (of each straight line between two points of the cloud) that lies inside the surface. These ten histograms are normalized and appended to result in one feature descriptor. As each histogram contains 64 bins, the final feature vector will have a length of 640 elements. The ESF is implemented in the Point

Cloud Library [94] in C++, allowing for fast feature extraction of point clouds. The ESF is robust against outliers and against occlusion [97].

FoldingNet

Different from VFH and ESF, FoldingNet is a data-driven approach, which means that the feature extraction process has to be learned on training data first. FoldingNet is an autoencoder [93] structure that uses a PointNet [85] backbone inside the encoder to obtain a 1024-element descriptor of the input point cloud. During training this descriptor will also be passed through the decoder network which tries to reconstruct the original point cloud. The Chamfer distance [7] is used as the error signal that captures the distance between original and reconstructed cloud during training.

Graph creation

Each part obtained by the segmentation algorithm is represented as an individual node inside the part graph. Since the watertight and clean mesh introduces a complete topology on the mesh, neighbouring parts can easily be identified and edges between them can be added to the part graph. For the node attributes, the extracted feature vectors are used, whereas edges are kept unlabeled. Other techniques for graph creation would also be possible, e.g. via kNN.

5.3.4 Graph kernels

The attributed graphs obtained by the previous steps will highly differ in their number of nodes, connectivity and node attributes. Therefore a method is needed which is able to handle this type of irregular data. A popular approach for this are graph kernels, which have proven to be useful in various tasks involving irregular data such as protein classification [12] and room classification [34]. Three different graph kernels are employed: the Shortest-Path kernel [12], the Graph-Hopper kernel [29] and the Wasserstein Weisfeiler-Lehman kernel [114]. In general, graph kernels define distances between smaller entities, i.e. pairs of nodes (each from one of the two graphs). Based on these

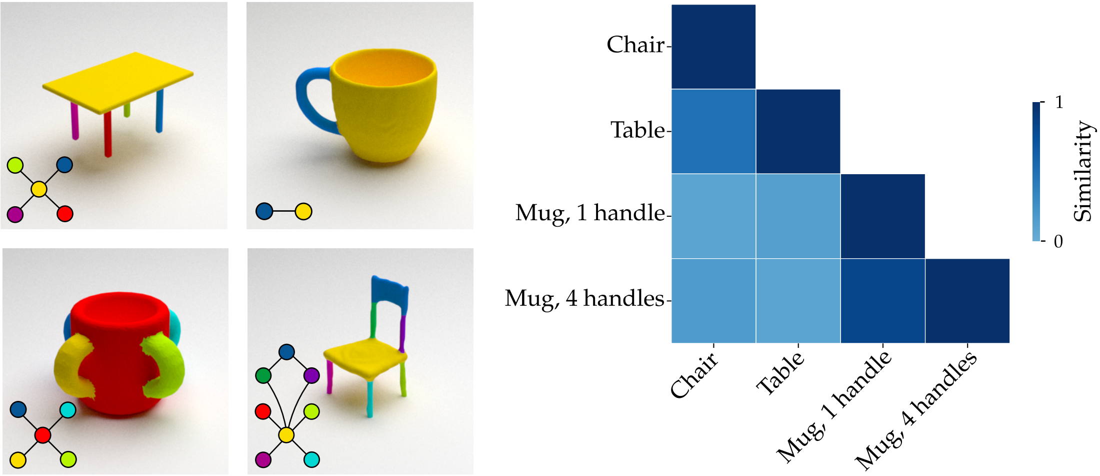


Figure 5.3: Objects are automatically segmented and part graphs are created. 3D shape descriptors (here: ESF) of parts serve as node attributes in the graph. Using these node embeddings, graph kernels (here: Shortest Path kernel) can be used to estimate the similarity between objects represented as graphs. Graph node colors and part colors are selected randomly across the objects. Figure adapted from Teich et al. [111]

atomic distances, distances for more complex entities such as paths can be created (cf. Fig. 5.3).

Shortest-Path kernel

The Shortest-Path kernel [12] is one of the earliest graph kernels developed and was successfully applied to bioinformatical data. Given any graph G with vertices V and edges E (expressed as pairs (v_i, v_j)), path are defined as sequences of vertices: $\pi = v_0, v_1, ..., v_n$. Every two subsequent vertices inside the sequence of the path must be adjacent, i.e.:

$$(v_i, v_{i+1}) \in E. \tag{5.1}$$

Additionally, paths should not contain cycles:

$$v_i \neq v_j \text{ if } i \neq j \tag{5.2}$$

The Shortest-Path kernel will analyze the two input graphs and compare all pairs of shortest paths inside the two graphs. First, a Shortest-Path graph can be constructed from each of the two individual input graphs G_i and G_j to make further processing easier. To transform a graph G_i into a Shortest-Path graph, a copy of the original graph is created, called S_i and edges are added such that each edge in S_i indicates that there exists a path between the two vertices in G_i. After transforming both input graphs into their respective shortest-path graphs S_i, S_j, the Shortest-Path kernel is then defined as:

$$k\left(S_i, S_j\right) = \sum_{e_i \in E_i} \sum_{e_j \in E_j} k_{\text{path}}\left(e_i, e_j\right) \tag{5.3}$$

where $e_i = (u_i, v_i)$ and $e_j = (u_j, v_j)$. The path distance is calculated as:

$$\begin{aligned} k_{\text{path}}\left(e_i, e_j\right) =& k_v\left(l\left(v_i\right), l\left(v_j\right)\right) k_e\left(l\left(e_i\right), l\left(e_j\right)\right) k_v\left(l\left(u_i\right)\right) + \\ & k_v\left(l\left(v_i\right), l\left(u_j\right)\right) k_e\left(l\left(e_i\right), l\left(e_j\right)\right) k_v\left(l\left(v_j\right)\right), \end{aligned} \tag{5.4}$$

where $k_e(l(e_i), l(e_j))$ compares the lengths of the two paths via the Dirac function:

$$k_e\left(l\left(e_i\right), l\left(e_j\right)\right) = \begin{cases} 1 & \text{if } l\left(e_i\right) = l\left(e_j\right) \\ 0 & \text{otherwise} \end{cases} \tag{5.5}$$

For the node distance kernel, two different kernels are experimented with; the linear kernel:

$$k_v\left(u, v\right) = u^T \cdot v \tag{5.6}$$

and the histogram-intersection kernel:

$$k_v\left(u, v\right) = \sum_{k=1}^{m} \min\left(u_k, v_k\right). \tag{5.7}$$

Normalization is applied to the calculated graph distances of the training data:

$$\frac{k\left(G_i, G_j\right)}{\sqrt{k\left(G_i, G_i\right) k\left(G_j, G_j\right)}}. \tag{5.8}$$

GraphHopper kernel

Since the Shortest-Path kernel is computationally very expensive with a complexity of $\mathcal{O}(n^4)$ [57], the more recent GraphHopper kernel was also tested, as it promises lower computational cost [57]. The building block of the GraphHopper kernel are the paths on the individual graphs G_i and G_j again.

$$k\left(G_i, G_j\right) = \sum_{\pi \in P} \sum_{\pi \in P'} k_{path}\left(\pi, \pi'\right), \tag{5.9}$$

with sets of shortest paths P, P' from G_i and G_j. Path kernel k_{path} is only calculated if the length of both paths π_i, π_j is identical:

$$k_{path}\left(\pi, \pi'\right) = \begin{cases} \sum_{j=1}^{|\pi|} k_v\left(\pi(j), \pi'(j)\right), & \text{if } |\pi| = |\pi'|, \\ 0, & \text{otherwise,} \end{cases} \tag{5.10}$$

where $\pi(j)$ represents the jth node inside the path π. As the vertex kernel, the linear kernel (Eq. (5.6)) is used.

Wasserstein Weisfeiler-Lehman kernel

The Wasserstein Weisfeiler-Lehman (WWL) graph kernel provides a framework for attributed graphs to estimate distances between each other. The WWL can be seen as an amalgamation of the Weisfeiler-Lehman test (for continuous attributes in our case) and the Wasserstein distance. The WWL graph kernel first computes feature vectors for all nodes of the involved graphs and then uses the Wasserstein distance for matching nodes between the two graphs.

Given an attributed graph, node attributes can be described by $a^0(v)$ for all vertices $v \in G$. In this work, the node attributes are the part descriptors, obtained by the feature extraction from the individual parts of the object. Toginalli et al.

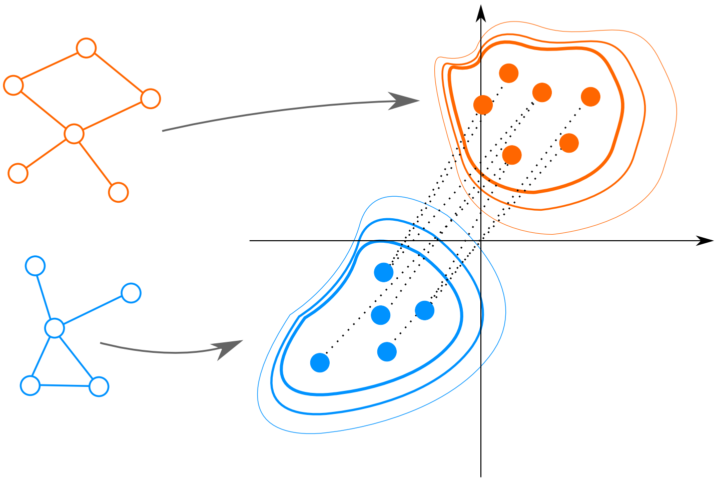


Figure 5.4: Schematic of the Wasserstein Weisfeiler-Lehman transform of graphs into a shared embedding space. Correspondence between nodes of the involved graphs is solved via the sinkhorn algorithm. Figure adapted from [114].

[114] propose a novel propagation scheme in this continuous node label case:

$$a^{h+1}(v) = \frac{1}{2}\left(a^h(v) + \frac{1}{\deg(v)} \sum_{u \in \mathcal{N}(v)} w(v,u) \cdot a^h(u)\right) \tag{5.11}$$

with $w(v,u)$ being the edge weights between node u and v (always 1 in our case). This formula basically updates the current node attribute by considering all neighbors of a specific node, calculating their average and finally averaging the nodes' current attribute and the aforementioned neighborhood attribute. A specific amount of iterations can be chosen in order to create multiple node attributes for each node. All these node attributes for can later be concatenated to obtain a single node attribute per node inside the graph.

In traditional graph kernels [12, 29], the graphs are decomposed into smaller entities of which all combinations between the two involved graphs are later compared and aggregated. Instead, the authors implement a matching approach based on the Wasserstein distance (or "Earth mover's distance" (EMD)). Given two probability distributions, the EMD can help constructing a distance between them by considering the effort of transporting smaller amounts of one of the distributions such that the result will be the second distribution. This transportation problem is executed on the nodes of the two graphs (cf. Fig. 5.4). As the distance between two nodes, Togninalli et al. [114] propose the euclidean

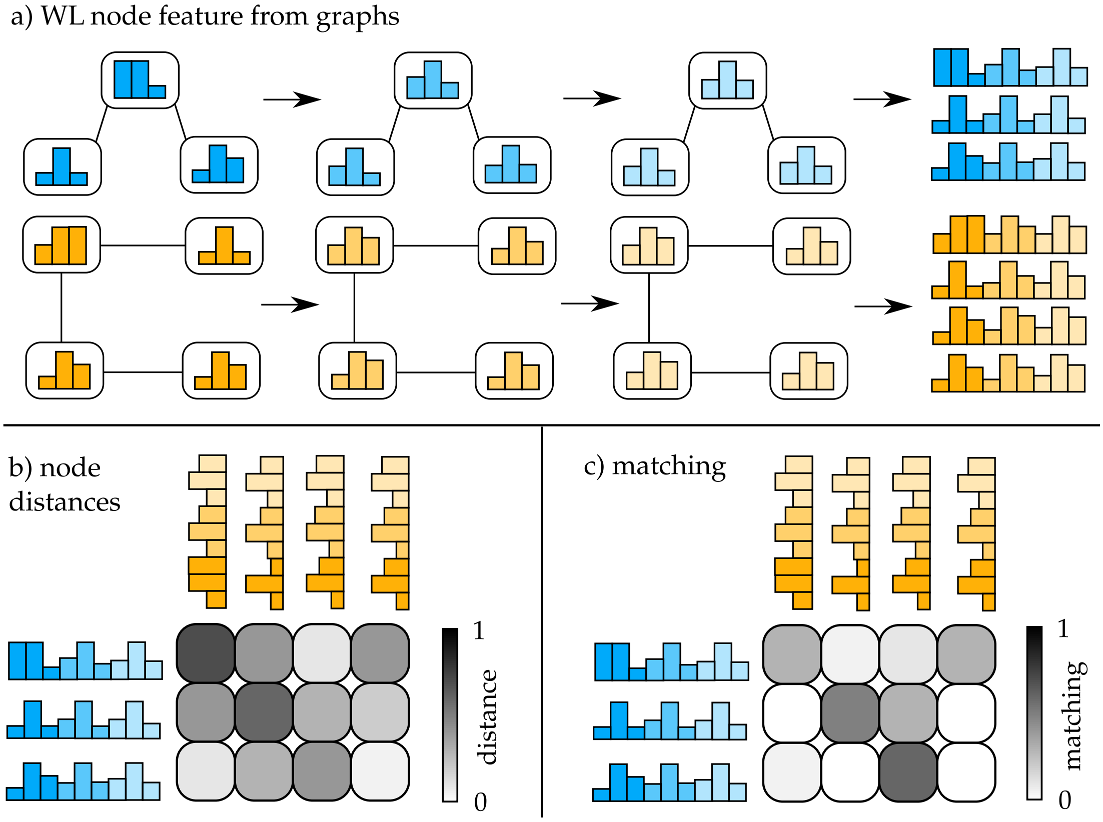


Figure 5.5: a) Illustration of two initial attributed graphs and subsequent Weisfeiler-Lehman iterations (cf. Eq. 5.11) to result in new nodewise attributes. b) distance matrix using euclidean distance between pairs of nodes. c) sparse matching between the nodes.

distance:

$$d_E(v, v') = \| v - v' \| \tag{5.12}$$

Given the distance matrix between all nodes from the two graphs, a nearly optimal solution minimizing the overall transportation cost can be found. For this, the Sinkhorn method is used, as this approach approximates the exact EMD with great precision and simultaneously reduces the computational cost needed [26]. Input to the Sinkhorn algorithm is the node distance matrix of all node pairs between the two graphs as calculated by Eq. 5.12 and visualized in Fig. 5.5 b). The output of the Sinkhorn algorithm is a soft assignment or

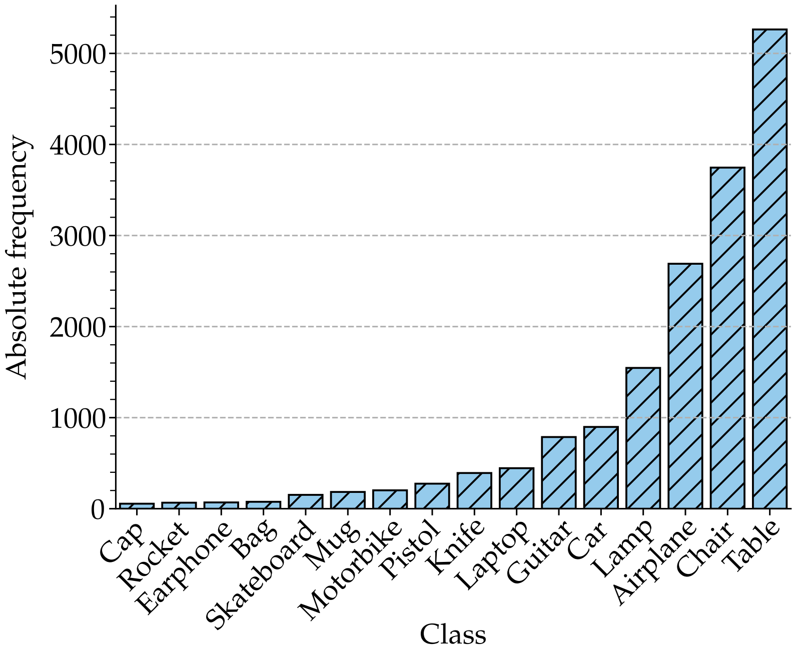


Figure 5.6: Class representation in the original subset of part annotations of ShapeNet. As representation of different classes varies strongly, a balanced subset of 880 objects (55 from each class) was sampled and used for the experiment.

matching between the nodes of the graphs. Each entry inside the matching lies between 0 and 1 and each row as well as each column has the sum 1, as visualized in Fig. 5.5 c). The overall WWL distance between the two graphs is then obtained by multiplying the node distance and the matching matrix and taking the sum of the resulting matrix. Additionally, the graph distances $D_W^{f_{\mathrm{WL}}}$ obtained by the EMD matching are preprocessed as [114] propose to use a Laplacian kernel.

$$K_{\mathrm{WWL}} = e^{-\lambda D_W^{f_{\mathrm{WL}}}}. \tag{5.13}$$

λ is another hyper parameter that can be tuned, similar to the regularization parameter C that is used in SVMs. This procedure results in a kernel matrix for the given set of graphs, which can be used inside an SVM for graph classification.

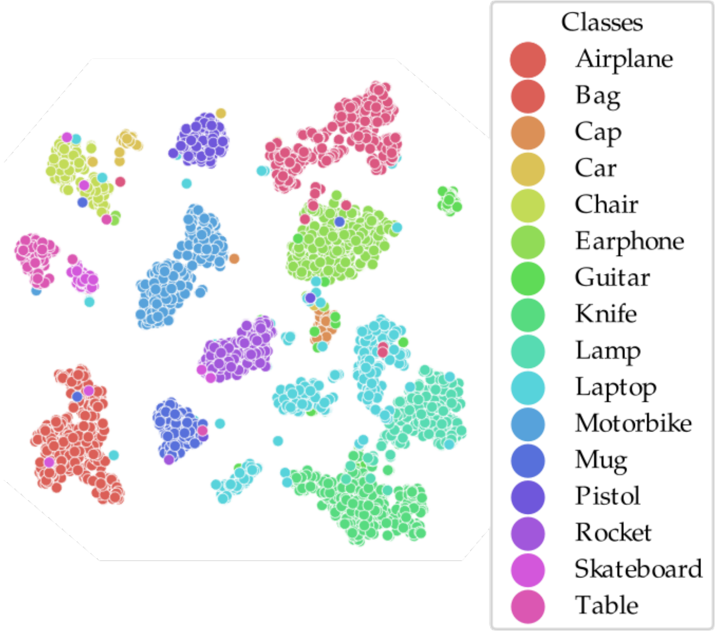


Figure 5.7: t-SNE embedding of the global FoldingNet descriptor extracted from several 3D objects of the ShapeNet dataset. Although class specific clusters can be identified visually, there are outlier instances that are located close to the wrong cluster. This may eventually lead to inferior classification results. Ideally, non-overlapping clusters are desired.

5.4 Experiment: Ordinary Object Classification

5.4.1 Experiment

The novel part-based 3D object classification pipeline is tested on a subset of the Shape-Net dataset [20]. The ShapeNet dataset consists of over 51.000 3D objects from 55 different categories. A subset of this dataset is richly annotated, including consistent segmentation and labeling of parts of the 3D objects. From this dataset of over 16.000 objects from 16 classes, 880 objects were selected (55 instances of each class) randomly since the original dataset is highly imbalanced, as visualized in Fig 5.6. For the part-based pipeline, two segmentations are available: the above-mentioned ground truth and the part segmentation obtained by the automatic approach described in Section 5.3.2. Each variation of the segmentation is benchmarked in combination with each of the three graph kernels (SP, GH, WWL). To compare against global methods, the respective global features are extracted from the meshes and kNN as well as SVMs are employed for the final class predictions on these global approaches. For both the global SVM as well as for the graph kernel SVMs, Grid Search Cross Valida-

Classifier		VFH	ESF	FoldingNet
Global descriptor-based Pipeline				
kNN		62.26 ± 2.86	73.73 ± 2.95	92.43 ± 1.87
SVM		65.32 ± 3.46	79.89 ± 2.92	95.84 ± 1.58
Graph-based Pipeline				
GH	automatic seg.	71.96 ± 3.00	78.32 ± 3.04	95.43 ± 1.57
SP	automatic seg.	75.38 ± 3.13	78.68 ± 3.12	92,81 ± 1.82
WWL	automatic seg.	**76.86 ± 2.85**	**82.46 ± 2.82**	**96.49 ± 1.41**
GH	ground truth seg.	82.31 ± 2.78	85.88 ± 2.51	96.60 ± 1.41
SP	ground truth seg.	84.16 ± 2.28	85.67 ± 2.76	94.86 ± 1.48
WWL	ground truth seg.	84.33 ± 2.77	89.26 ± 2.58	97.38 ± 1.11

Table 5.1: Classification accuracy of tested methods. The columns refer to different descriptors. Especially for the VFH and ESF descriptors, classification accuracy of the graph-based approaches are significantly higher than the respective global-pipeline. The WWL graph kernel performs always best, whereas GH and SP are often still performing better than their global-descriptor counterparts. Table taken from Teich et al. [111].

tion is used only on the training data for parameter tuning. $k = 10$ is chosen as the number of neighbours in the kNN classifier, based on which the prediction is made.

5.4.2 Results

Quantitative results of the experiment are summarized in Tab. 5.1. Looking at the individual descriptors, a big gap between hand-engineered (VFH, ESF) features and the data-driven FoldingNet descriptor is observable: both global descriptor pipelines (kNN and SVM) only achieve about 60-80% accuracy on VFH and ESF, whereas FoldingNet accuracy lies around 92 - 96%. However, it is important to emphasize that the FoldingNet descriptors' improved accuracy comes with the downside of training the autoencoder structure for several

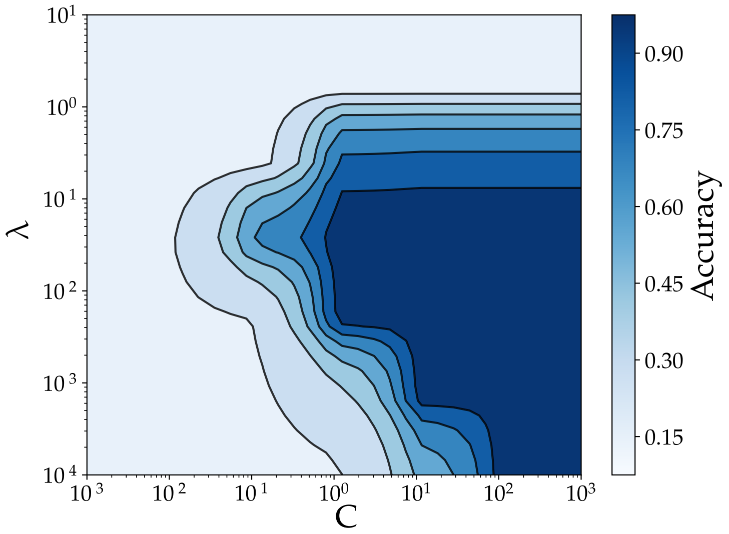
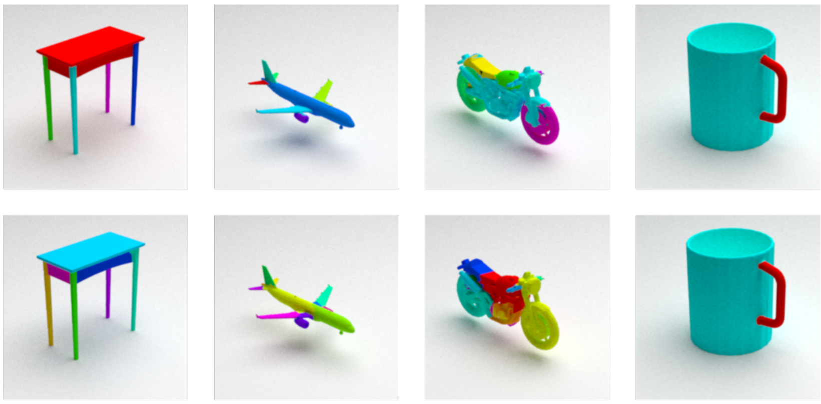


Figure 5.8: Visualization of Grid Search (with cross validation) on the training data for hyper parameter tuning of λ (see Eq. 5.13) and C (SVM regularization parameter). These two parameters are used in the SVM and will significantly impact the prediction result. Descriptor: FoldingNet, graph kernel: WWL.

epochs, disqualifying it for plug-n-play solutions in e.g. robotic pipelines.

Comparing the results of the graph-based pipeline, significant improvements in classification seem to occur when the ground truth segmentation is provided. This observation seems intuitive, since ground truth segmentations create a consistency in the segmentation scheme, i.e. legs are always separated from the table, whereas in the automatic segmentation approach, parts may get merged due to inadequate segmentation and the descriptor of the resulting graph-node might be far away from typical prototypes of the two original parts involved. This deficient segmentation behaviour subsequently leads to flawed class predictions, as similarities between graphs will be impacted by these inconsistent segmentations. Furthermore, the impact of providing correct segmentation to the part-based pipeline seems to differ for the three descriptors: VFH and ESF greatly benefit from the canonical ground truth segmentation, whereas accuracy on FoldingNet improves by a smaller margin. Furthermore, GraphHopper and Shortest-Path kernels show inferior performance than the WWL kernel overall.



knn (global)	Table	Rocket	Airplane	Mug
SVM (global)	Chair	Airplane	Skateboard	Mug
SP Graph Kernel	Table	Skateboard	Motorbike	Mug
GH Graph Kernel	Knife	Airplane	Motorbike	Mug
WWL Graph Kernel	Table	Airplane	Motorbike	Mug

Figure 5.9: Qualitative classification results of global classifiers and graph classifiers based on automatic segmentations and using the VFH descriptor. Ground truth segmentation is shown for comparison of segmentations. Note that GH, SP and WWL predictions are based on automatic segmentation, not ground truth segmentations. Figure adapted from Teich et al. [111].

5.5 Experiment: Out-of-Distribution Object Classification

5.5.1 Experiment

Additional to the classification task described in Section 5.4, an experiment on out-of-distribution data was performed. The motivation here is due to the design of the graph kernels: since $\mathcal{R}$-Convolutions work on decompositions of the involved graphs, in general close distances are assigned to graphs that share a significant substructure with each other. This behaviour can be used to easier recognize e.g. a *chair* with five *legs* since many of its subgraphs resemble graphs inside the training data, usually *chairs* consisting of four *legs*. To demonstrate this advantage of graph kernels, artificial data (50 objects) was created from three classes (*chairs*, *tables*, and *mugs*). For each class, a part label was chosen

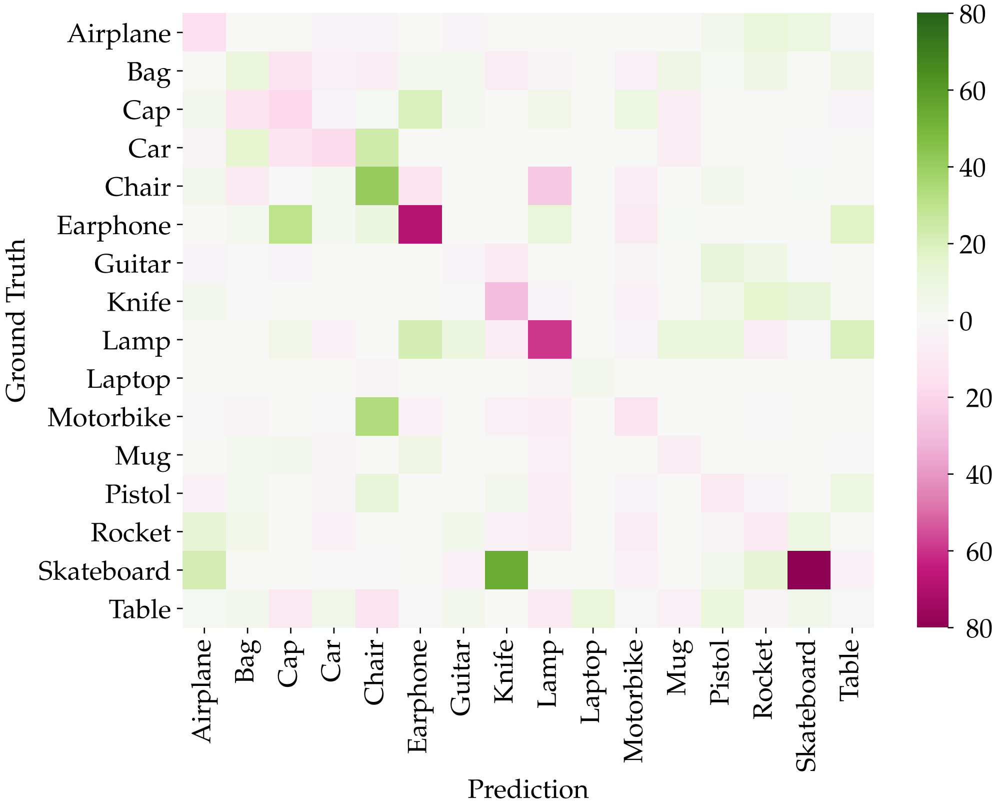


Figure 5.10: Difference between confusion matrices of the global VFH-SVM and the VFH-GH graph kernel SVM using the automatic segmentation approach over 25 runs. Numbers indicate how many more samples were predicted as Class X in the global SVM compared to the graph-based method. Thus, on the diagonal, pink regions indicate superior performance of the graph-based VFH-GH, whereas outside of the diagonal, green spots indicate superior performance of VFH-GH.

that - if present multiple times on the same object - would not change the overall object class. By adding such redundant parts, the out-of-distribution dataset is created. Between one and up to six legs were added to all tables and chairs and also one to six handles to all mugs. All redundant parts were pasted on appropriate regions to not change the final object class. Examples of this artificial dataset are visualized in Fig. 5.11.

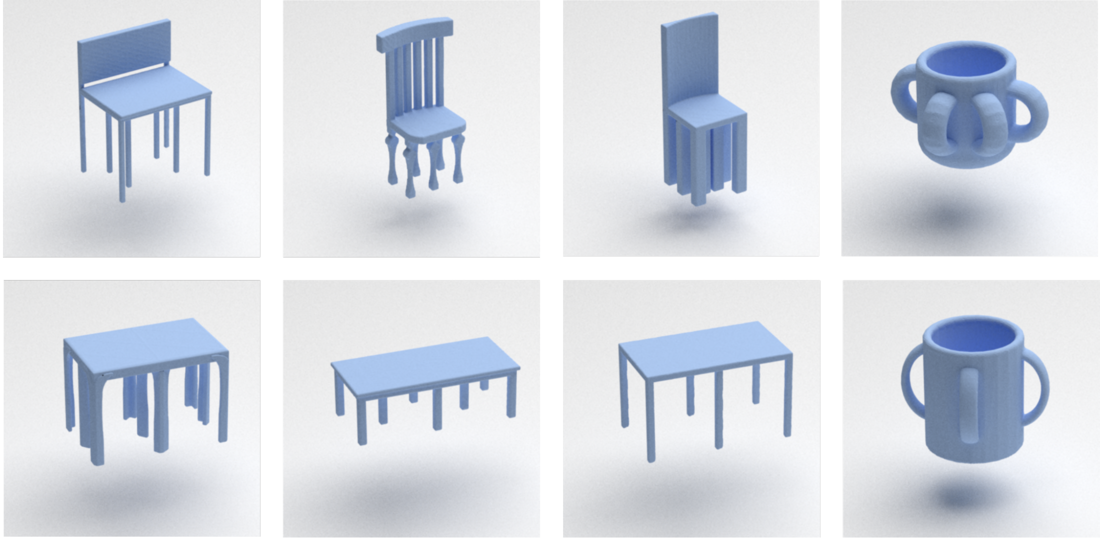


Figure 5.11: Artificial object examples. Tables and chairs contain redundant legs and mugs contain additional handles. These objects are taken from the out-of-distribution dataset.

5.5.2 Results

Consistent with the results from Section 5.4.2, graph-based methods outperform the global classification approach in this experiment. Interestingly, the WWL kernel is not the best choice in all settings: in case of using the VFH descriptor, the two other graph kernels (GH and SP) both outperform WWL (although in case of GH only by a slight margin). Another important observation is the wide performance gap between the global SVM and the graph-based methods in case of the FoldingNet descriptor: the redundant parts seem to have a significant impact on the global shape descriptor (SVM: 84%, graph-based methods: 96-98%). The results show that the graph-based methods may be able to exploit the known substructures in the training dataset to re-identify them when confronted with objects containing redundant parts.

Method	VFH	ESF	Folding-Net
Global descriptor			
SVM	72	80	84
Graph-based			
GH, autom. seg.	78	76	96
SP, autom. seg.	**90**	84	**98**
WWL, autom. seg.	76	**88**	**98**

Table 5.2: Classification accuracy of tested methods on out-of-distribution objects. The columns refer to different descriptors. The global SVM classification results are inferior to the respective graph-based SVM result (except for the GH kernel when using the ESF descriptor). Table taken from Teich et al. [111].

5.6 Conclusion

In this chapter, classic global 3D shape classification was analyzed and an alternative part-based pipeline was proposed. The novel approach was evaluated in an experiment, also accounting for different shape descriptors, segmentations and graph kernels. The result on the out-of-Distribution data supply more empirical evidence that one of the advantages of graph-based methods is that the final predictions are based on common subgraphs between training and evaluation data. Extending existing robotic applications - that make use of classical descriptors like VFH and ESF - by the here introduced graph-based method may increase their performance further and eventually lead to more reliable vision pipelines. One bottleneck of the graph-based technique seems to be the automatic segmentation subprocess, as processes of the pipeline are relying on a correct segmentation. The performance gap between the automatic segmentation and ground truth segmentation indicate that there is room for improvement for future segmentation algorithms. Thanks to the modularity of the approach, newly developed segmentation methods (and graph kernels as well) can easily be exchanged to boost the classification accuracy towards the theoretical limit of the ground truth segmentation. In the future, investigating possibilities of adapting the graph-based approach on Graph Neural Networks [56] as a substitute for the SVM is important, as SVMs introduce practical limitations to the size of training data (as kernel matrices grow quadratically). Graph Neural Networks may alleviate this issue and may offer similar classification performance.

The next chapter focuses on the task of object repairment. Here, it is essential to entangle the different parts of the object and to individually evaluate whether a part needs to be fixed or not. For this, an automatic segmentation approach as well as a part-graph-based model are used - similar to the developed approaches in Chapters 3, 4 and 5.

6

Assembly Repairment

6.1 Introduction

As novel algorithms are being developed and systems evolve, robots are more and more integrated into society [36]. In various application areas, ranging from assistive elderly care [27, 119], over industrial use cases [120] to human-centered services [42, 116, 132], robots are already used. To further support humans, more sophisticated robotic systems have to be designed. One task that is currently still outside the realm of possibility for robots is object repairment. Given a defect or manipulated instance of an object, the task is to repair the object and restore its canonical structure. This task is significantly different from predefined conveyor-belt-like robotic assembly tasks where a specific set of instructions is already hardcoded and executed. In object repairment, the main issues are twofold. First, it is not explicitly stated which part of the object is manipulated and thus has to be changed. Second, even if the compromised object part is known, it is not clear where to assemble it without prior knowledge about the object's function or structure. In the following, the task of assembly repairment is formalized and tackled from a Computer Vision viewpoint where additional annotation information is limited or non-existent and only a "raw" point cloud is used as input. This stands in stark contrast to current object assembly pipelines where part segmentations and labels are assumed to be provided from the start - an assumption that is hardly satisfied in real-world scenarios.

6.2 Related Works

In recent years, many different works in the field of Computer Graphics enabled working on highly-detailed and annotated 3D point clouds of man-made objects. From object assembly [45], over object generation to object interpolation [74], many of these methods focus on datasets that are arguably hard to obtain, to create and to annotate in the wild. The task of object assembly is very fundamental in robotics and already has a lot of applications. In bigger industry settings, robots are often employed to assemble parts into entire objects in a predefined way. But humans can also benefit from robotic object assembly in their everyday life: given a package of parts and tools, assistive robots may be able to construct furniture bought from retailers that offer unassembled products such as IKEA [8, 52, 64]. Of course, these products often come with manuals containing step-by-step instructions for assembling the final product. However, when there are not many parts to consider, humans are sometimes able to assemble them "intuitively" and thus do not need to rely on any explicit set of instructions. On the other hand, robots still seem to lack this skill of autonomously assembling the product without much prior knowledge.

Generative Models: A large body of work in the field of Computer Graphics focuses on object understanding via generative models. In [128], the authors acquired 3D mesh models of household objects with consistent semantic segmentations. Through evolutionary algorithms [35], their model is able to create novel objects of the same class, resulting in interesting combinations of object parts from multiple object instances. This process might be of great benefit for designers as the qualitative results often look plausible yet still *artificial*. Random part combinations obtained through this process do not consider typical ingredients of man-made design: cultural context and function [77]. In [65], an autoencoder model was designed that enables encoding of the objects' structure, part layout and spatial part relationships. Using this model, object blending/interpolation is possible. Given two instances of the same object class, instances *in between* can be sampled, e.g. to visualize a step-by-step transformation from the source instance to the target instance. Similarly, Mo et al. [74] use supervised learning on Graph Neural Networks [56] to train models that are able to generate novel object instances, respecting the typical object structure and spatial part relations. Their work [74] can be seen as an improvement over GRASS [65] as the resulting models resemble realistic objects, whereas objects generated by GRASS may often lack realistic appearance.

In [45], Huang et al. develop a Graph Neural Network for 3D object assembly. The process involves multi-stage message passing in a coarse-to-fine manner. Their proposed model takes advantage of the semantically labeled input data to alternatingly propagate information between part instances of the same class and all part instances. This way, all instances of a given class can first reorganize themselves and afterwards refine their orientations and relations based on the global part structure. Different from [45], the presented approach does not require semantic labels or even segmentations as input during evaluation. In fact, the proposed approach will internally make use of an automatic segmentation method in order to obtain knowledge of the individual object parts. This relaxation allows for an easier use of the proposed method in new scenarios, as no segmentation ground truth or semantic labels are required for evaluation. Furthermore, no leveraging spatial relations such as adjacency or symmetry are leveraged explicitly.

A more relaxed setting of the assembly problem is *object repairment*. Given that the overall object is already assembled and only a single part of the object was previously somehow manipulated, the task is to find out how to transform

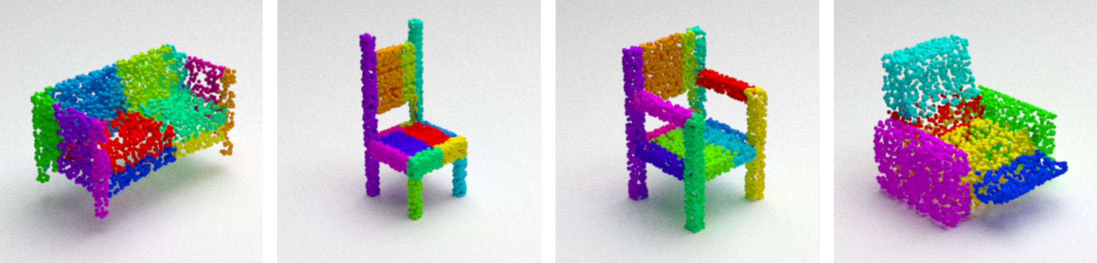


Figure 6.1: Qualitative results of the ACD [66] segmentation method.

which part to restore the overall object. In this chapter, object repairment is seen as part pose prediction on an unlabeled 3D point cloud input.

6.3 Methods

Dataset & Segmentation

The input to the proposed pipeline is a point cloud of the object that requires repairment. We use a subset of the ShapeNet dataset [19] for training and evaluation. The dataset contains objects from classes such as *chairs* and *tables*. During evaluation, first the object is segmented into individual parts. For automatic segmentation, Approximate Convex Decomposition (ACD) [66] is used. This algorithm clusters the 3D input into groups, each covering its members by a convex hull. The results are often slight oversegmentations of the original objects. This approach works very well for CAD-like objects constructed from 3D primitives but may be suboptimal in cases of high details inside the object (such as engravings or jittery object surfaces).

Point cloud reconstruction

As the repair task can be formulated as a per-part pose estimation, supervised learning techniques can be applied - given an appropriate error signal. However, the idea is to not directly supervise the predicted transformation signal itself. Instead, the predicted transformation is applied to a reference point cloud and the disparities in reconstruction are used as error signal. Thus, the transformed point cloud becomes a proxy for the underlying transformation that it

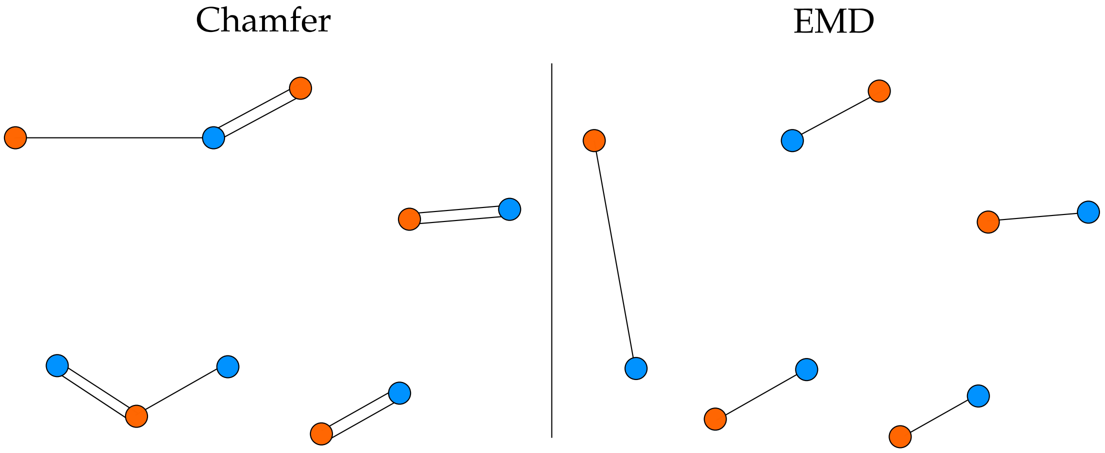


Figure 6.2: Illustration of matchings used for Chamfer and EMD distances between two point clouds.

was subjected to. There are typically two different distances that can be used for point clouds: Earth mover's distance and Chamfer distance.

Earth mover's distance

For computing the Earth mover's distance, a matching between points of the two clouds is required. This matching needs to minimize the sum of all pairs of euclidean distances between a point in cloud S_1 and its matched point in cloud S_2.

$$d_{\text{EMD}}(S_1, S_2) = \min_{\Phi : S_1 \rightarrow S_2} \sum_{x \in S_1} ||x - \Phi(x)||_2 \tag{6.1}$$

where $\Phi : S_1 \rightarrow S_2$ is a bijection.

Chamfer distance

Comparing two point clouds by means of the Chamfer distance is often computationally less expensive than by means of EMD. This is due to the simpler definition of the Chamfer distance, as no advanced matching technique has to be used. The distance is defined as the sum of two terms, each of which aggregates the (euclidean) distance of each point of one of the two clouds to its closest neighbour inside the other cloud.

$$d_{\mathrm{CD}}(S_1, S_2) = \sum_{x \in S_1} \min_{y \in S_2} ||x - y||_2^2 + \sum_{y \in S_2} \min_{x \in S_1} ||x - y||_2^2 \tag{6.2}$$

During preliminary experiments on both the Chamfer and EMD loss, several observations are made. First, the Chamfer and EMD distance between clouds will grow quadratically, when one of the two point clouds is moved away from the other in one dimension (cf. Fig. 6.4). However, in case of the Chamfer distance, this growth is stronger as the second cloud moves away from the reference cloud. Additionally, when using the Chamfer distance, a bigger portion of the overall loss may be due to the reconstructed clouds' translation instead of its orientation.

6.3.1 Part-level pose prediction

The proposed method will predict a transformation for each of the identified parts of the object individually. In Fig. 6.3 b), this pipeline is visualized. We start with the point cloud of the part as input to the PointNet [85] backbone. The purpose of this backbone is to process the 3D input in an appropriate manner (order invariant) and extract meaningful features of the global shape. Similar to [45], after a 1024-element feature vector is extracted, two MLPs are used in order to predict the required transformation: one MLP will regress the translation (3 elements: x,y,z) and the second MLP will regress the rotation (4 elements: rotation quaternion). We opted for the use of quaternions due to two major advantages over their alternatives: First, compared to rotation matrices, quaternions are much more compact (4 elements vs 9 elements) as well as numerically stable. Second, compared to euler angles, quaternions avoid the gimbal lock problem [17]. Quaternion rotations are 4 element vectors that can be represented as:

$$q = s + xi + yj + zk \qquad s, x, y, z \in \mathbb{R}, \tag{6.3}$$

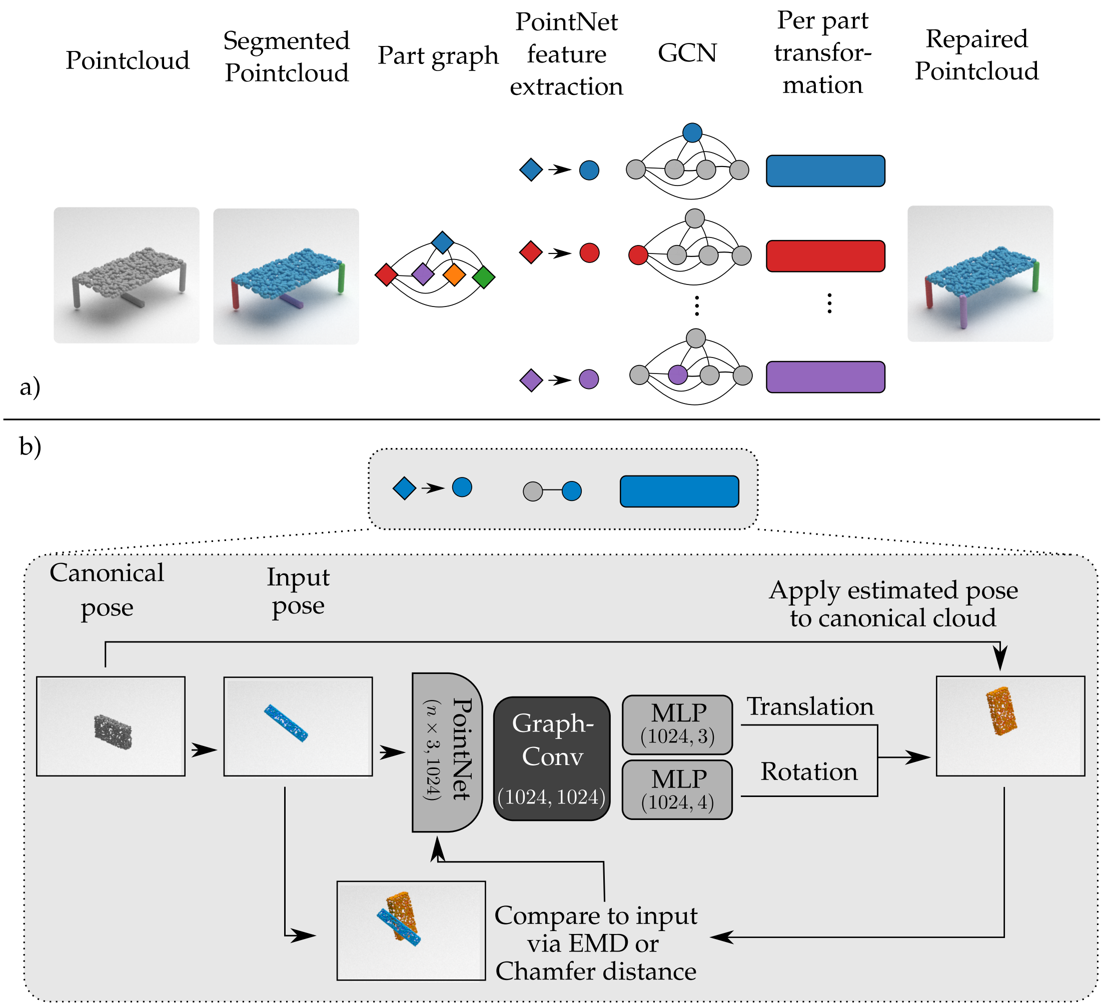


Figure 6.3: a) Assembly repairment pipeline. Given a point cloud of a compromised object, ACD [66] is used to split the individual parts. After creating a part graph, individual features are extracted from each part's point cloud. These features are then propagated to all other parts via the GraphConv layer. Subsequently, two MLPs extract rotation and translation for each part. b) Per part pipeline. During training, correct part configurations are known and can be used for computing the loss.

inspired by the notation of complex numbers. To test whether the above-mentioned network is able to capture spatial properties of a given shape, a preliminary experiment is conducted on customized point clouds. Investigation on whether the PointNet backbone indeed captures such spatial properties is crucial for the overall system that will eventually tackle the assembly repair-

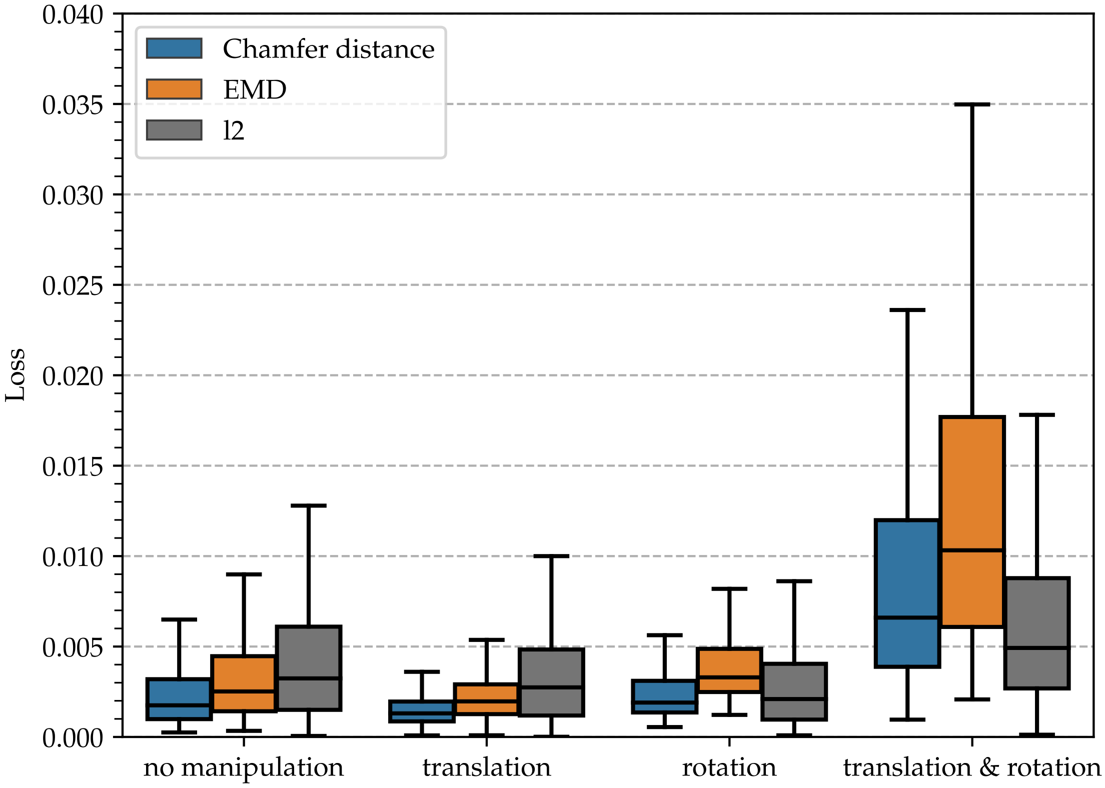


Figure 6.4: Reconstruction of a single part (box), captured by different metrics. Translation as well as rotation seem to partially improve the regression performance compared to "no manipulation". However, when applying both transformations to the input point cloud, the reconstruction error strongly increases.

ment task, as it relies on accurate spatial information about the respective object parts. The process of this intermediate experiment is visualized in Fig. 6.3 b). We sampled a single point cloud ("reference point cloud") and during training we applied random transformations to this point cloud. Subsequently, the manipulated point cloud is fed to the model as input. The model creates a predicted transformation vector (7 elements, 3 for translation, 4 for rotation) that can be applied to the reference point cloud. Using EMD or Chamfer distance, the reconstruction error can be calculated and the weights of the model can be adapted accordingly. In Fig. 6.4, the results of the experiments are visualized. A point cloud of 1024 points, sampled from a box mesh with extents $[0.2, 0.4, 1.0]$ and center coordinates $(0, 0, 0)$ were used as the reference point

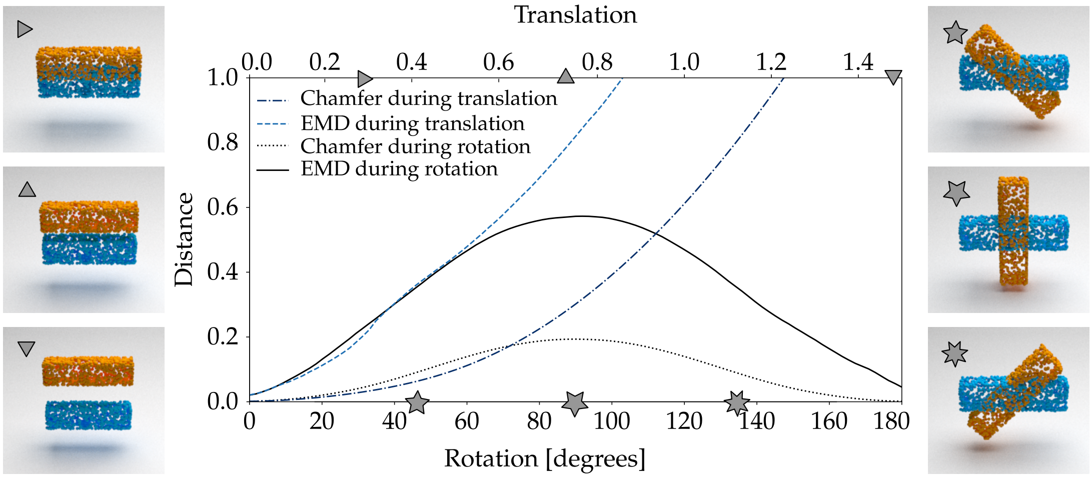


Figure 6.5: EMD and Chamfer distance for rotation and translation.

cloud. We ran the experiment for different configurations, for 50000 training iterations per configuration:

- no manipulation: here, the transformation is not changing the point cloud. Thus, the desired output of the model is supposed to be $(0, 0, 0, 1, 0, 0, 0)$.
- translation: the point cloud was randomly translated in each dimension based on a uniform random distribution between $[-0.5, 0.5]$
- rotation: the point cloud was rotated randomly.
- translation & rotation: the point cloud was first translated randomly as detailed above and subsequently randomly rotated.

We use the Chamfer distance and EMD for measuring the reconstruction error. Additionally, the distance between the mean of the original point cloud and the reconstruction in euclidean space is measured ("l2"). It is worth mentioning that the l2 metric only captures one aspect of the reconstruction (rough spatial location) and thus may not capture the rotational differences between the two clouds. Looking at the results in Fig. 6.4, several observations can be made. First, across the four different configurations, values for all three measured metrics change similarly. Interestingly, the reconstruction error is consistently

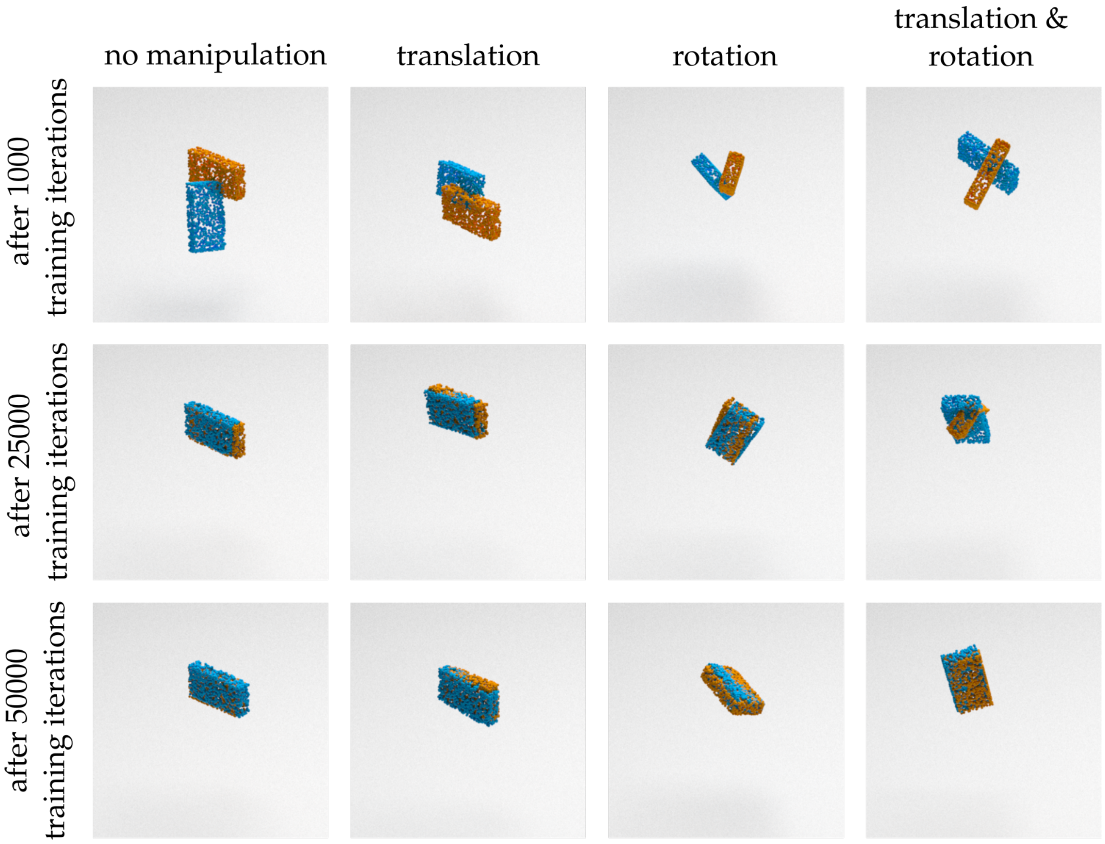

no manipulation
translation
rotation
translation & rotation
after 1000 training iterations
after 25000 training iterations
after 50000 training iterations

Figure 6.6: Qualitative reconstruction results based on the predicted pose of the PointNet backbone during training for all four manipulation configurations.

reduced when changing the configuration from "no manipulation" to "translation". A possible explanation for this behaviour is that for the "no manipulation" action, only a very limited subset of the input space is actually presented to the system during training. On the other hand, when employing translation, many more regions inside the input space are covered during training, from a statistical perspective. This input variety translates into learning more useful convolutional kernels compared to the "no manipulation" setting, observable by the lower reconstruction errors. Second, rotation - similar to translation - reduces the reconstruction error, compared to the "no manipulation" setting, at least in the case of Chamfer distance and l2. When using both manipulations in conjunction, the reconstruction error increases, compared to all other

Translation	Rotation	mean CD input	mean CD reconstruction
		0.000	0.0024
✓		0.0832	0.0018
	✓	0.0370	0.0031
✓	✓	0.1624	0.0101

Figure 6.7: Comparison of mean Chamfer distance between input cloud and reference clouds ("mean CD input") and mean Chamfer distance between reconstructed cloud and reference cloud ("mean CD reconstruction").

settings due to many more possible poses that have to be captured. To offer more perspectives for the results of the single point cloud reconstruction by pose estimation, the mean Chamfer distance for randomly manipulated input clouds and their reconstruction for all four settings was analyzed. The results are shown in Tab. 6.7. This table illustrates that input point clouds subjected to both translation and rotation are - on average - orders of magnitude stronger perturbed than their single-manipulation counterparts (0.1624 vs. 0.0832 and 0.0370).

6.3.2 Object-level pose prediction

Since objects usually differ in the number of parts they are composed of, the segmentation output is highly irregular. Thus, Graph Convolutional Networks [56] are used to process the object by means of a part graph. These graphs allow for rich representations of individual parts (nodes) and relations between these parts (edges). Graph Convolutional Networks allow propagating information contained in nodes to their neighbours as well as subsequent updating of each node's state based on the newly aggregated neighbourhood information. Similar to Huang et al. [45], each part's initial feature vector is propagated to all other parts of the object. Afterwards, each part's transformation is predicted individually. For this prediction, the aforementioned PointNet backbone is used. Note, that the learned weights of this network module are shared across all nodes, meaning each node is evaluated using the same network. An illustration of this pipeline is shown in Fig. 6.3.

During training, the single-node backbone is provided with a point cloud as input and the predicted transformation is applied to this input point cloud and compared to the reference cloud in an auto-encoder manner. The distance between the reconstructed point cloud and the reference cloud is then used as the error signal during backpropagation to adapt the individual weights of the network. Instead of the Chamfer loss used in [45], the Earth mover's distance is used due to better reconstruction performance in preliminary experiments. For the convolutional graph layer, *GraphConv* [75] is employed, as this layer performed best compared to other convolutional layers such as GCN [56], GAT [118] or GIN [129]. For the GraphConv layer, the node features are updated as follows:

$$\tilde{x_i} = W_1 x_i + W_2 \sum_{j \in \mathcal{N}(i)} x_j, \tag{6.4}$$

with weight matrices W_1, W_2.

We identified two critical hypotheses that are relevant for the employed approach.

Context improves pose prediction accuracy

The reconstruction performance of single parts was investigated in Sec. 6.3.1. It is important to quantify whether correctly placed parts will improve the reconstruction performance. If this hypothesis does not hold true, there is little justification for using techniques such as message passing to propagate part information to all parts involved. To empirically investigate this hypothesis, an experiment is ran on a custom *table* subset of the 3D dataset for two different configurations:

- The tabletop is already located at the correct position and only a single leg now has to be placed accordingly.
- The tabletop and three legs are already located at the correct positions and the fourth leg has to be placed accordingly.

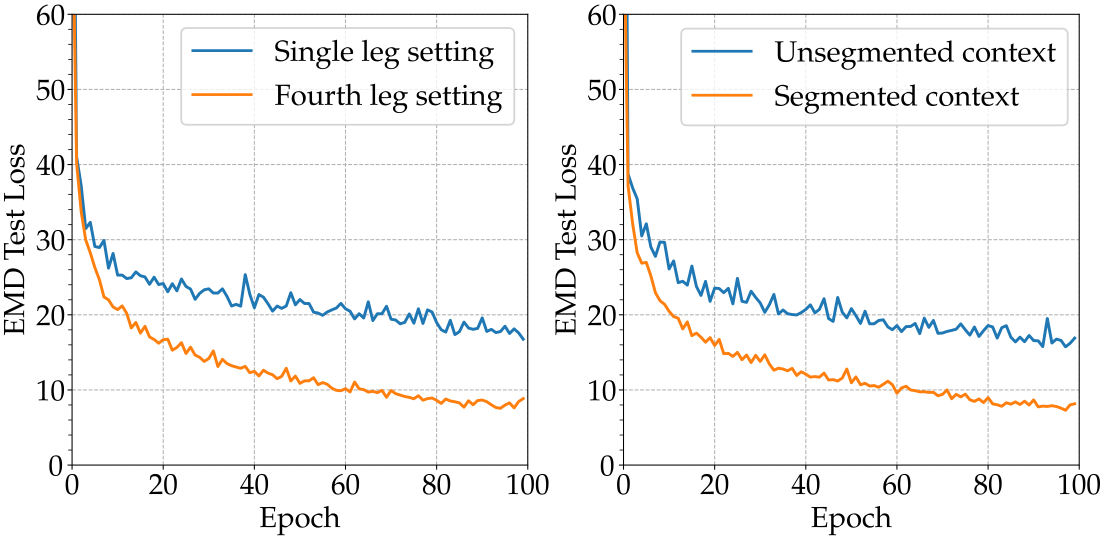


Figure 6.8: Test loss during training for the settings of the two experiments.

Intuitively, for humans these two tasks are both easily solvable. However, the intuition here is that the GCNs will use redundant parts to their advantage, i.e. by exploiting symmetry information. In the first configuration, there is no redundant information about the other three legs that can be exploited. In the second configuration however, point- and axes symmetry relations may help to consistently predict the desired pose of the fourth leg.

Segmentation improves context

If indeed context helps to improve the reconstruction, why not provide it (the already correctly placed parts) as one single entity? Here it is investigated whether deconstruction of the complex object may increase the performance of the reconstruction further. Again, a custom table subset of the 3D dataset is used and two different scenarios are constructed. In both scenarios, the tabletop and three legs are already located at the correct positions and the fourth leg has to be placed accordingly.

- Provide the tabletop and the three legs as one aggregated part to the network.
- Provide the tabletop and the three legs as segmented parts to the network.

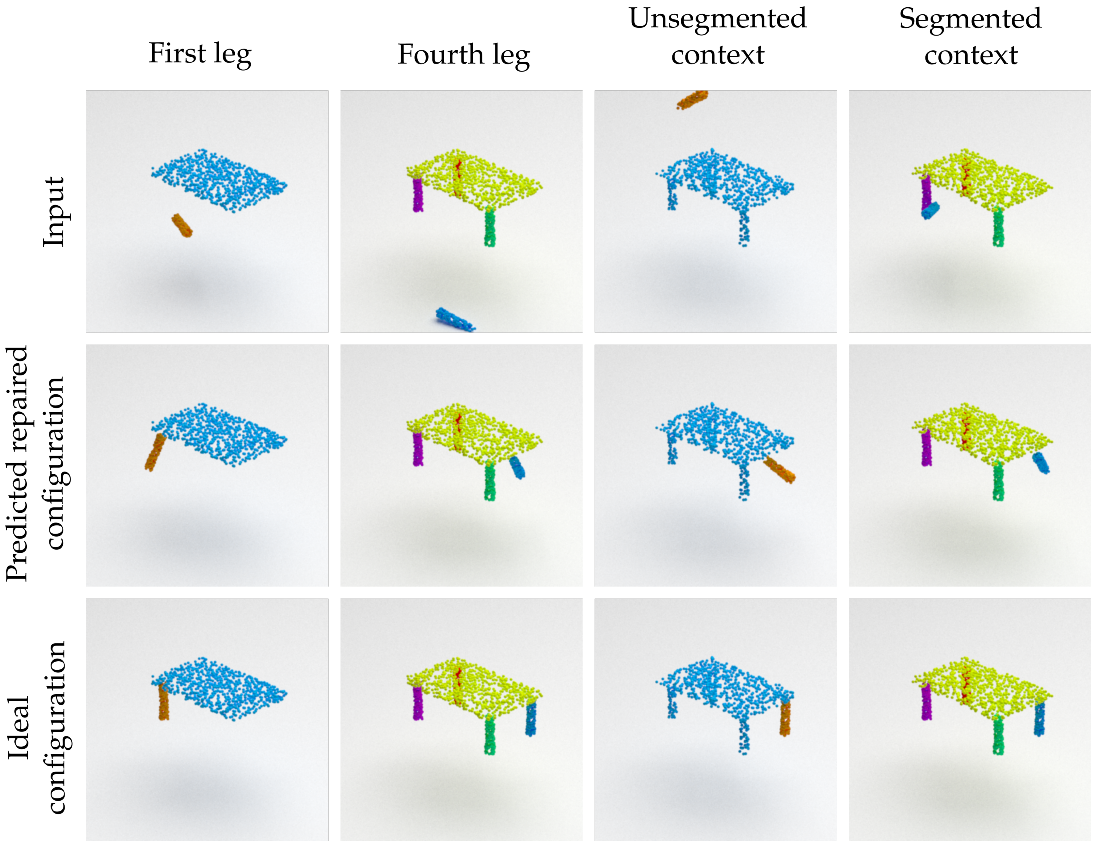


Figure 6.9: Qualitative results for the different settings.

6.4 Experiments

As mentioned in Sec. 6.3, a subset of the ShapeNet dataset is used for the experiments. Specifically, 4096 chairs and tables are sampled from the dataset. The automatic ACD segmentation of the respective objects was taken from [37]. Ground truth semantic segmentations of these objects were obtained from the original PointNet [85] repository. As objects first need to be manipulated in order to create scenarios where repairment is needed, the ground truth segmentations for this particular task were used. After transforming the semantic segmentations to instance segmentations, individual parts of each object can be manipulated. In the experiment, all objects were first centered and scaled to fit into a unit sphere. Afterwards, a random part from the ground truth segmentation was selected. The selected part was translated randomly using a uniform

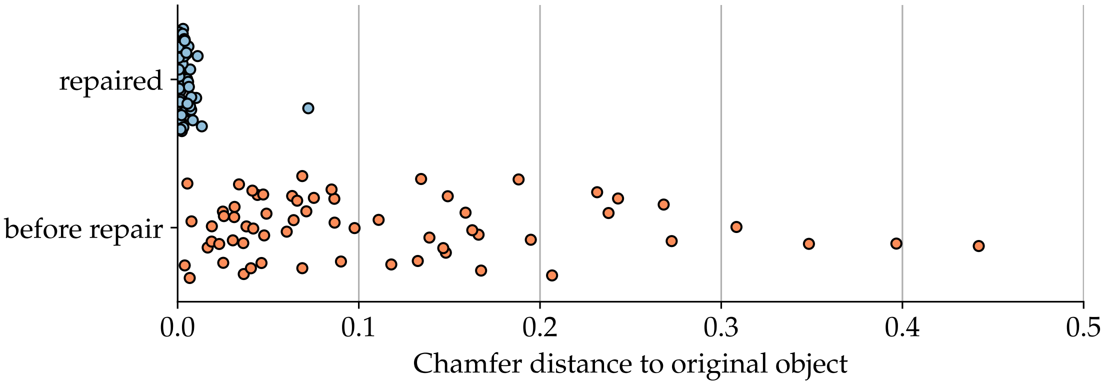


Figure 6.10: Absolute distances of scenarios before and after repairment. Quantified by evaluating Chamfer distance to ideal object configuration. 64 samples considered.

distribution of $\mathcal{U}(-0.5, 0.5)$ for all three dimensions. For the rotation manipulation, a random quaternion was sampled from a uniform distribution $\mathcal{U}(0, 1)$ and normalized to reflect only rotation and no change in scale.

$$q_{\text{norm}} = \frac{q}{||q||}, \text{ where } ||q|| = \sqrt{w^2 + x^2 + y^2 + z^2}, \text{with } q = [w, x, y, z]. \quad (6.5)$$

After retrieving the automatic ACD segmentation for the cloud, a new segment for the points that were affected by the manipulation action was created. This is reasonable, as the manipulated part usually creates a novel connected component which is spatially disconnected from the rest of the object, such that this connected component can be clustered into a single segment. We split the data objects into a training and test set, using 10% of the objects for testing. The pipeline was implemented in PyTorch Geometric [31]. The training consisted of 50 epochs and was executed on an Nvidia GTX 1080 Ti. A batch size of 4 objects was used and the learning rate of the Adam optimizer [55] was set to 0.001. As loss during training, EMD was used.

6.5 Results

Fig. 6.10 shows quantitative results of the approach on 64 random objects, where always a single part was manipulated ("before repair") in order to create a sce-

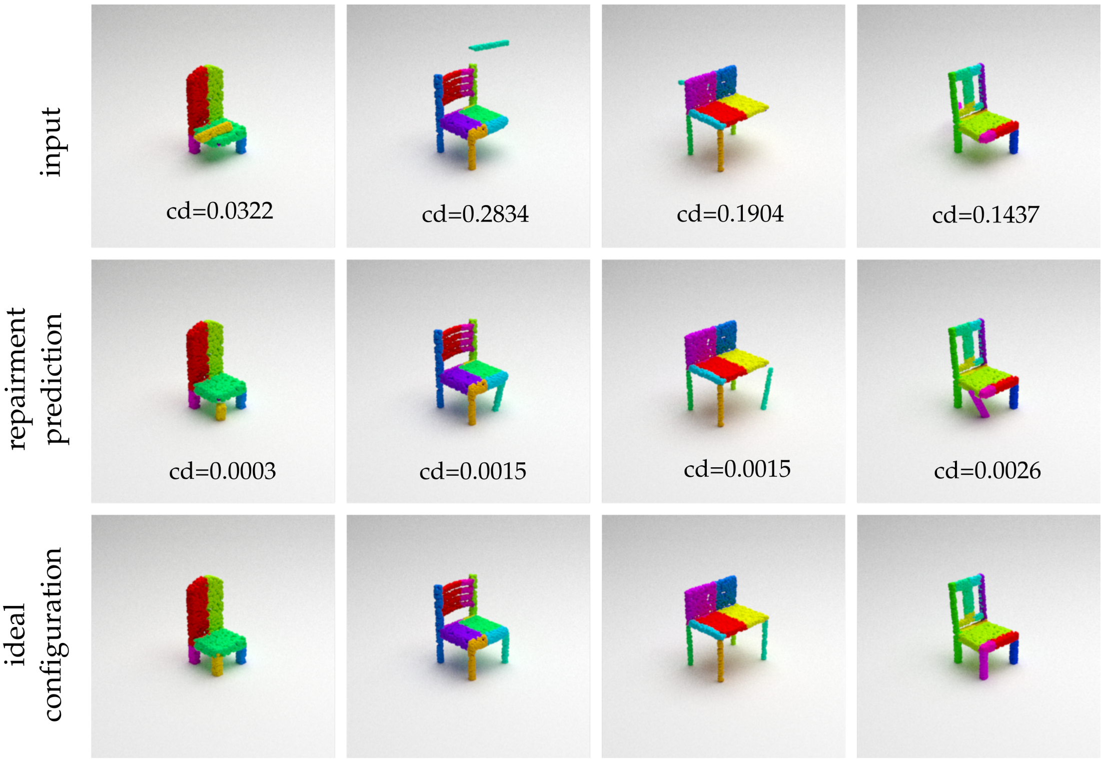


Figure 6.11: Qualitative results for four chairs.

nario where the object is defect. The dispersion inside the input configuration ("before repair") reflects the variance of the individual manipulations: the uniformly distributed translation and the random rotation of the part. On average, the mean Chamfer distance between the original point cloud and the defect object point cloud is 0.45. The designed approach greatly reduces the distance to the optimal point cloud by entire orders of magnitudes. This is also illustrated in Fig. 6.12. Here, the Chamfer distance to the ideal cloud was normalized by dividing the distance of the prediction by the initial (defect) input configuration. As it can be seen, the proposed approach reduces the distance to the ideal configuration to maximally 10% in most cases.

As strong drops in the l2 center loss can be observed, most of the improvement in Chamfer distance can be attributed to precise prediction of the translation of the defect part. On the other hand, as visualized in Fig. 6.11, orientation

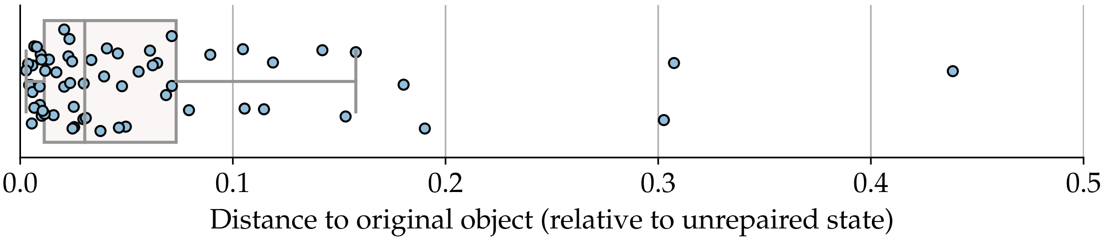


Figure 6.12: Distances of reconstructed assembly to ideal configuration, relative to pre-repairment state. Only 64 samples considered.

prediction may sometimes lack precision. The repaired chairs in column 2, 3 and 4 of Fig. 6.11 are not completely orthogonally oriented in relation to the seat.

6.6 Conclusion

In this chapter, a pipeline for the novel task of object repairment was designed. The introduced approach tackles multiple issues by employing automatic 3D segmentation and dense pose regression of the individual object parts. In the experiments, empirical evidence was found for two important hypotheses. First, in contrast to non-segmented object scenarios, the segmentation of the object improved the prediction of corrected poses on the test dataset. Second, the approach makes use of the various parts of the object. Concretely, the proposed method was more accurate in predicting the correct pose of the fourth leg of a table compared to scenarios where other legs were missing. As the method does not require any prior semantic segmentation or classification of the objects' parts, this is a step towards potential real-world applications, e.g. in robots. While unsegmented and unlabeled, the used dataset may not completely capture real world scenarios, where regions of the object might be occluded, background noise is present and 3D data is often only obtained by single-view cameras. Thus, investigating more realistic 3D domestic environment data in the future is very important.

7

Conclusion

In this work, part-based object understanding by means of graphs was used for classification tasks on artificial and real 3D data, as well as for object repairment.

As segmentation is crucial to any bottom-up reasoning approach, the topic of 3D object segmentation is explored. The MVCNN [62] approach was extended by defining a fully-convolutional ResNet architecture for more accurate part-boundary prediction. The presented model significantly cuts training time of the boundary-prediction network, making the RNN submodule of the original MVRNN [62] unnecessary. Most importantly, it was found that leveraging alternative rendering modalities may boost segmentation results of multi-view approaches. Especially the normal shading seems to contain more critical information about part-boundaries than the original phong-shading. Additionally, higher resolution of the input data and the use of augmentation techniques such as dilation on ground truth improve the robustness of the final part segmentations. For future extensions of the method, we want to provide automatic segmentations of other segmentation approaches as input to the model. This approach of segmentation-fusion may boost performance even further, as current methods are often complementary: one approach performs well on humanoid objects [100] and the other on CAD-created objects [4]. By providing both results to the network, a best-of-both-worlds result might be obtainable, if the model learns a heuristic when to rely on which segmentation aspect.

Secondly, symbolic and artifical data is explored by designing a rule-based dataset. Rules are comprised of symbols and their spatial relations to each other. As samples of the symbolic dataset can be transformed into 3D point cloud data, benchmarking of different classification systems on the various modalities is made possible. Two important aspects are focused on, namely the perception gap and the symbol gap. By comparing two models, one using segmented point clouds (indicating the represented symbols) and one using the symbolic data directly as input, we are able to quantify the perception gap, i.e. how much accuracy is due to imperfect extraction of symbolic data from the segmented point clouds. Good classification results are observable in both cases. The symbolic-input model nearly achieves maximum accuracy, whereas the performance drop of the segmented point cloud model is about 20% on average. The second important aspect is the impact of the correct symbolic segmented point cloud compared to its completely unsegmented point cloud counterpart. Here, the results show that providing the segmentation increases classification performance, outperforming standard methods such as PointNet and PointNet++. These results offer empirical evidence that - at least as long as the data adheres to underlying spatial rules and object classes are highly structured - better segmentation strategies may boost classification results in the future when employed in such models.

Furthermore, graph kernels are transferred to the 3D domain for object classification. By employing an automatic segmentation approach, part graphs are obtained that are attributed using 3D feature descriptors. The employed graph kernels enable us to compare graphs to each other, allowing for a robust classification pipeline. The results show that the novel approach is often superior to methods that directly extract global shape descriptors for classification, specifically kNN and SVMs. The class predictions achieved on the test data show that objects that share common structures are easier to associate with each other, resulting in higher classification accuracy. In a second experiment the exploitation of important substructures by the part-based approach is demonstrated as artificial data is used in which redundant parts are added to the objects. Differences in the various kernels are observable, as the Wasserstein Weisfeiler-Lehman kernel dominates over the Shortest-Path and GraphHopper kernel. Additionally, the theoretical upper bound of the part-based approach is explored by providing ground truth segmentations to the pipeline. Here, big performance gaps compared to automatic segmentation can be observed. This implies that with better segmentation, further improvements to the part-based approach can be achieved. As the pipeline is highly modular, all three main components can easily be exchanged: feature descriptor and graph kernel as well as the segmentation method.

Lastly, a pipeline for object repairment via part-graphs is created. The proposed pipeline automatically segments a given object point cloud into its parts, and predicts manipulation actions for each part individually in order to repair the overall object. First, during initial experiments, it was found that the already correctly placed parts enrich the scene, ultimately improving the predictions of the model. This might indicate that the model is able to learn concepts such as symmetry, however, additional evidence is required before a definitive statement can be made. Second, the automatic segmentation increases repairment results compared to non-segmented input clouds. As a subnetwork was used for feature extraction from the point clouds, it was found that less complex point clouds are better covered by their extracted feature descriptor than more complex point clouds. By employing an automatic segmentation approach, these more complex clouds get separated into primitive, more simple clouds, leading to better regression results. The approach was demonstrated on a heterogeneous dataset containing both chairs and tables that need to be repaired. The results show that approximating the translation of all parts involved works to a great extent. The biggest contributor to the overall reconstruction error are the part orientations of the compromised part. Thus, there is still room for improvement for orientation prediction in the future, e.g. through better feature extractors. In the future, more meaningful aspects - such as anchoring - of the object assembly have to be considered and need to be reflected in metrics for more accurate quantitative measurements.

Bibliography

[1] Naveed Akhtar and Ajmal Mian. Threat of adversarial attacks on deep learning in computer vision: A survey. *IEEE Access*, 6:14410–14430, 2018.

[2] Yasuhiro Aoki, Hunter Goforth, Rangaprasad Arun Srivatsan, and Simon Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7163–7172, 2019.

[3] Ehsan Asoudegi and Zhibing Pan. Computer vision for quality control in automated manufacturing systems. *Computers & Industrial Engineering*, 21(1):141 – 145, 1991. ISSN 0360-8352. doi: https://doi.org/10.1016/0360-8352(91)90078-K. URL `http://www.sciencedirect.com/science/article/pii/036083529190078K`.

[4] Marco Attene, Bianca Falcidieno, and Michela Spagnuolo. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer*, 22 (3):181–193, 2006.

[5] Oscar Kin-Chung Au, Youyi Zheng, Menglin Chen, Pengfei Xu, and Chiew-Lan Tai. Mesh segmentation with concavity-aware fields. *IEEE Transactions on Visualization and Computer Graphics*, 18(7):1125–1134, 2011.

[6] A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *Proceedings 2003 International Conference on Image Processing (Cat. No.03CH37429)*, volume 3, pages III–513, 2003. doi: 10.1109/ICIP.2003.1247294.

[7] HG Barrow, JM Tenenbaum, RC Bolles, and HCf Wolf. Parametric correspondence and chamfer matching: Two new techniques for image matching. In *Proceedings: Image Understanding Workshop*, pages 21–27. Science Applications, Inc Arlington, VA, 1977.

[8] Yizhak Ben-Shabat, Xin Yu, Fatemeh Saleh, Dylan Campbell, Cristian Rodriguez-Opazo, Hongdong Li, and Stephen Gould. The ikea asm dataset: Understanding people assembling furniture through actions,

objects and pose. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 847–859, 2020.

[9] Halim Benhabiles, Guillaume Lavoué, Jean-Philippe Vandeborre, and Mohamed Daoudi. Learning boundary edges for 3d-mesh segmentation. In *Computer Graphics Forum*, volume 30, pages 2170–2182. Wiley Online Library, 2011.

[10] Irving Biederman. Recognition-by-components: a theory of human image understanding. *Psychological review*, 94(2):115, 1987.

[11] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[12] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *Fifth IEEE international conference on data mining (ICDM'05)*, pages 8–pp. IEEE, 2005.

[13] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

[14] Mario Botsch, Leif Kobbelt, Mark Pauly, Pierre Alliez, and Bruno Lévy. *Polygon mesh processing*. CRC press, 2010.

[15] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[16] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *International Conference on Learning Representations*, 2018.

[17] Danail S Brezov, Clementina D Mladenova, and Ivaïlo M Mladenov. New perspective on the gimbal lock problem. In *AIP Conference Proceedings*, volume 1570, pages 367–374. American Institute of Physics, 2013.

[18] Kaidi Cao, Maria Brbic, and Jure Leskovec. Concept learners for few-shot learning. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=eJIJF3-LoZO`.

[19] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv*

preprint arXiv:1512.03012, 2015.

[20] Angel X. Chang, Thomas A. Funkhouser, Leonidas J. Guibas, Pat Hanrahan, Qi-Xing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. *CoRR*, abs/1512.03012, 2015. URL `http://arxiv.org/abs/1512.03012`.

[21] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3), May 2011. ISSN 2157-6904. doi: 10.1145/1961189.1961199. URL `https://doi.org/10.1145/1961189.1961199`.

[22] Xiaobai Chen, Aleksey Golovinskiy, and Thomas Funkhouser. A benchmark for 3D mesh segmentation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 28(3), August 2009.

[23] Zhi-Quan Cheng, Kai Xu, Bao Li, Yan-Zhen Wang, Gang Dang, and Shi-Yao Jin. A mesh meaningful segmentation algorithm using skeleton and minima-rule. In *International Symposium on Visual Computing*, pages 671–680. Springer, 2007.

[24] Simon Christoph Stein, Markus Schoeler, Jeremie Papon, and Florentin Worgotter. Object partitioning using local convexity. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 304–311, 2014.

[25] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967. doi: 10.1109/TIT.1967.1053964.

[26] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.

[27] Miroslawa CYLKOWSKA-NOWAK, Slawomir Tobis, Claudia Salatino, Adriana Tapus, and Aleksandra Suwalska. The robots in elderly care. In *2nd International Multidisciplinary Scientific Conference on Social Sciences and Arts SGEM2015*, Albena, Bulgaria, 2015. URL `https://hal.archives-ouvertes.fr/hal-01762539`.

[28] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively

trained, multiscale, deformable part model. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[29] Aasa Feragen, Niklas Kasenburg, Jens Petersen, Marleen de Bruijne, and Karsten Borgwardt. Scalable kernels for graphs with continuous attributes. In *Advances in neural information processing systems*, pages 216–224, 2013.

[30] Matthias Fey. Just jump: Dynamic neighborhood aggregation in graph neural networks. *arXiv preprint arXiv:1904.04849*, 2019.

[31] Matthias Fey and Jan Eric Lenssen. Fast graph representation learning with pytorch geometric. *arXiv preprint arXiv:1903.02428*, 2019.

[32] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 869–877, 2018.

[33] Martin A Fischler and Robert A Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on computers*, 100(1): 67–92, 1973.

[34] Matthew Fisher, Manolis Savva, and Pat Hanrahan. Characterizing structural relationships in scenes using graph kernels. In *ACM Transactions on Graphics (TOG)*, volume 30, page 34. ACM, 2011.

[35] Lawrence J Fogel, Alvin J Owens, and Michael J Walsh. *Artificial intelligence through simulated evolution*. Wiley, New York, NY, 1966. URL `https://cds.cern.ch/record/107769`.

[36] Susanne Frennert. *Older People Meet Robots: Three Case Studies on the Domestication of Robots in Everyday Life*. PhD thesis, Certec - Rehabilitation Engineering and Design, August 2016. Defence details Date: 2016-09-09 Time: 09:15 Place: Stora Hörsalen, IKDC, Sölvegatan 26, Lunds Tekniska Högskola External reviewer(s) Name: Jaeger, Birgit Title: Professor Affiliation: Roskilde Universitet, Danmark —.

[37] Matheus Gadelha, Aruni RoyChowdhury, Gopal Sharma, Evangelos Kalogerakis, Liangliang Cao, Erik Learned-Miller, Rui Wang, and Subhransu Maji. Label-efficient learning on point clouds using approximate convex decompositions. In *European Conference on Computer Vision*

(*ECCV*), 2020.

[38] Aurelien Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow.* O'Reilly, Beijing, first edition edition, 2017.

[39] Aleksey Golovinskiy and Thomas Funkhouser. Randomized cuts for 3d mesh analysis. In *ACM SIGGRAPH Asia 2008 papers*, pages 1–12. 2008.

[40] Richard HR Hahnloser, Rahul Sarpeshkar, Misha A Mahowald, Rodney J Douglas, and H Sebastian Seung. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789): 947–951, 2000.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[42] Wei He, Zhijun Li, and CL Philip Chen. A survey of human-centered intelligent robots: issues and challenges. *IEEE/CAA Journal of Automatica Sinica*, 4(4):602–609, 2017.

[43] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[44] Donald D Hoffman and Manish Singh. Salience of visual parts. *Cognition*, 63(1):29–78, 1997.

[45] Jialei Huang, Guanqi Zhan, Qingnan Fan, Kaichun Mo, Lin Shao, Baoquan Chen, Leonidas Guibas, and Hao Dong. Generative 3d part assembly via dynamic graph learning. In *The IEEE Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[46] Jingwei Huang, Hao Su, and Leonidas Guibas. Robust watertight manifold surface generation method for shapenet models. *arXiv preprint arXiv:1802.01698*, 2018.

[47] Daniel Huber, Anuj Kapuria, Raghavendra Donamukkala, and Martial Hebert. Parts-based 3d object classification. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II. IEEE, 2004.

[48] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[49] Paul Jaccard. The distribution of the flora in the alpine zone. *New Phytologist*, 11(2):37–50, 1912. doi: https://doi.org/10.1111/j.1469-8137.1912.tb05611.x. URL `https://nph.onlinelibrary.wiley.com/doi/abs/10.1111/j.1469-8137.1912.tb05611.x`.

[50] Joel Janai, Fatma Guney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *Foundations and Trends in Computer Graphics and Vision*, 12, 04 2017. doi: 10.1561/0600000079.

[51] Hui Jia and Jiangang Zhang. Extract segmentation lines of 3d model based on regional discrete curvature. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 9(1):265–274, 2016.

[52] Jonathan D Jones, Cathryn Cortesa, Amy Shelton, Barbara Landau, Sanjeev Khudanpur, and Gregory D Hager. Fine-grained activity recognition for assembly videos. *arXiv preprint arXiv:2012.01392*, 2020.

[53] Oliver Van Kaick, Noa Fish, Yanir Kleiman, Shmuel Asafi, and Daniel Cohen-Or. Shape segmentation by approximate convexity analysis. *ACM Transactions on Graphics (TOG)*, 34(1):1–11, 2014.

[54] Sagi Katz, George Leifman, and Ayellet Tal. Mesh segmentation using feature point and core extraction. *The Visual Computer*, 21(8):649–658, 2005.

[55] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[56] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. 2017.

[57] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *Applied Network Science*, 5(1):1–42, 2020.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[59] Joseph B Kruskal. *Multidimensional scaling*. Number 11. Sage, 1978.

[60] Yu-Kun Lai, Shi-Min Hu, Ralph R Martin, and Paul L Rosin. Fast mesh segmentation using random walks. In *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pages 183–191, 2008.

[61] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[62] Truc Le, Giang Bui, and Ye Duan. A multi-view recurrent neural network for 3D mesh segmentation. *Computers & Graphics*, pages 103–112, August 2017.

[63] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`.

[64] Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. Ikea furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

[65] Jun Li, Kai Xu, Siddhartha Chaudhuri, Ersin Yumer, Hao Zhang, and Leonidas Guibas. Grass: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics (TOG)*, 36(4):1–14, 2017.

[66] Jyh-Ming Lien and Nancy M Amato. Approximate convex decomposition of polyhedra. In *Proceedings of the 2007 ACM symposium on Solid and physical modeling*, pages 121–131, 2007.

[67] Ik Soo Lim and E Charles Leek. Curvature and the visual perception of shape: Theory on information along object boundaries and the minima rule revisited. *Psychological Review*, 119(3):668, 2012.

[68] Joseph J. Lim, Hamed Pirsiavash, and Antonio Torralba. Parsing IKEA Objects: Fine Pose Estimation. *ICCV*, 2013.

[69] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[70] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[71] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[72] Eric Margolis and Stephen Laurence. Concepts. In Edward N. Zalta,

editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, summer 2019 edition, 2019.

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

[74] Kaichun Mo, Paul Guerrero, Li Yi, Hao Su, Peter Wonka, Niloy Mitra, and Leonidas Guibas. Structurenet: Hierarchical graph networks for 3d shape generation. *ACM Transactions on Graphics (TOG), Siggraph Asia 2019*, 38(6):Article 242, 2019.

[75] Christopher Morris, Martin Ritzert, Matthias Fey, William L Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and leman go neural: Higher-order graph neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4602–4609, 2019.

[76] John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda: Is cuda the parallel programming model that application developers have been waiting for? *Queue*, 6(2): 40–53, 2008.

[77] Don Norman. *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[78] Nemer Odeh and Cem Direkoglu. Automated shopping system using computer vision. *Multimedia Tools and Applications*, pages 1–11, 2020.

[79] Anshul Paigwar, Ozgur Erkent, Christian Wolf, and Christian Laugier. Attentional pointnet for 3d-object detection in point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[80] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2027–2034, 2013.

[81] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein,

Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems* 32, pages 8024–8035. Curran Associates, Inc., 2019.

[82] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[83] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[84] Charles R Qi, Hao Su, Matthias Nießner, Angela Dai, Mengyuan Yan, and Leonidas J Guibas. Volumetric and multi-view cnns for object classification on 3d data. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5648–5656, 2016.

[85] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[86] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in neural information processing systems*, pages 5099–5108, 2017.

[87] Weichao Qiu, Fangwei Zhong, Yi Zhang, Siyuan Qiao, Zihao Xiao, Tae Soo Kim, and Yizhou Wang. Unrealcv: Virtual worlds for computer vision. In *Proceedings of the 25th ACM international conference on multimedia*, pages 1221–1224, 2017.

[88] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[89] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Play-

ing for data: Ground truth from computer games. In *European conference on computer vision*, pages 102–118. Springer, 2016.

[90] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[91] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3234–3243, 2016.

[92] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99–121, 2000.

[93] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[94] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[95] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.

[96] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[97] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux, and John Hsu. Fast 3d recognition and pose using the viewpoint feature histogram. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2155–2162. IEEE, 2010.

[98] Markus Schoeler, Jeremie Papon, and Florentin Worgotter. Constrained planar cuts-object partitioning for point clouds. In *Proceedings of the IEEE*

Conference on Computer Vision and Pattern Recognition, pages 5207–5215, 2015.

[99] Julius Schöning and Gunther Heidemann. Taxonomy of 3d sensors-a survey of state-of-the-art consumer 3d-reconstruction sensors and their field of applications. In *VISIGRAPP (3: VISAPP)*, pages 194–199, 2016.

[100] Lior Shapira, Ariel Shamir, and Daniel Cohen-Or. Consistent mesh partitioning and skeletonisation using the shape diameter function. *The Visual Computer*, 24(4):249–259, 2008.

[101] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[102] Shymon Shlafman, Ayellet Tal, and Sagi Katz. Metamorphosis of polyhedral surfaces using decomposition. In *Computer graphics forum*, volume 21, pages 219–228. Wiley Online Library, 2002.

[103] Zhenyu Shu, Chengwu Qi, Shiqing Xin, Chao Hu, Li Wang, Yu Zhang, and Ligang Liu. Unsupervised 3d shape segmentation and co-segmentation via deep learning. *Computer Aided Geometric Design*, 43: 39–52, 2016.

[104] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012.

[105] Manish Singh and Donald D Hoffman. Part-based representations of visual shape and implications for visual cognition. In *Advances in psychology*, volume 130, pages 401–459. Elsevier, 2001.

[106] Richard Socher, Cliff Chiung-Yu Lin, Andrew Y Ng, and Christopher D Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, 2011.

[107] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[108] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik G. Learned-Miller. Multi-view convolutional neural networks for 3d shape recogni-

tion. In *Proc. ICCV*, 2015.

[109] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International Conference on Machine Learning*, pages 3319–3328. PMLR, 2017.

[110] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[111] Florian Teich, Timo Lüddecke, and Florentin Wörgötter. 3d object classification via part graphs. In *VISAPP 2021*, 2020.

[112] Joshua B. Tenenbaum, Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011. ISSN 00368075. doi: 10.1126/science.1192788.

[113] Kiran K Thekumparampil, Chong Wang, Sewoong Oh, and Li-Jia Li. Attention-based graph neural network for semi-supervised learning. *arXiv preprint arXiv:1803.03735*, 2018.

[114] Matteo Togninalli, Elisabetta Ghisu, Felipe Llinares-López, Bastian Rieck, and Karsten Borgwardt. Wasserstein weisfeiler-lehman graph kernels. In *Advances in Neural Information Processing Systems*, pages 6436–6446, 2019.

[115] Federico Tombari, Samuele Salti, and Luigi Di Stefano. Unique signatures of histograms for local surface description. In *European conference on computer vision*, pages 356–369. Springer, 2010.

[116] Iis P Tussyadiah and Sangwon Park. Consumer evaluation of hotel service robots. In *Information and communication technologies in tourism 2018*, pages 308–320. Springer, 2018.

[117] Mikaela Angelina Uy, Quang-Hieu Pham, Binh-Son Hua, Thanh Nguyen, and Sai-Kit Yeung. Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1588–1597, 2019.

[118] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero,

Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[119] Alessandro Vercelli, Innocenzo Rainero, Ludovico Ciferri, Marina Boido, and Fabrizio Pirri. Robots in elderly care. *DigitCult-Scientific Journal on Digital Cultures*, 2(2):37–50, 2018.

[120] Valeria Villani, Fabio Pini, Francesco Leali, and Cristian Secchi. Survey on human–robot collaboration in industrial settings: Safety, intuitive interfaces and applications. *Mechatronics*, 55:248–266, 2018.

[121] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011.

[122] Jean-Baptiste Weibel, Timothy Patten, and Markus Vincze. Addressing the sim2real gap in robotic 3-d object classification. *IEEE Robotics and Automation Letters*, 5(2):407–413, 2019.

[123] Walter Wohlkinger and Markus Vincze. Ensemble of shape functions for 3d object classification. In *2011 IEEE international conference on robotics and biomimetics*, pages 2987–2992. IEEE, 2011.

[124] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[125] Yu Xiang, Wonhui Kim, Wei Chen, Jingwei Ji, Christopher Choy, Hao Su, Roozbeh Mottaghi, Leonidas Guibas, and Silvio Savarese. Objectnet3d: A large scale database for 3d object recognition. In *European conference on computer vision*, pages 160–176. Springer, 2016.

[126] S. Xie and Z. Tu. Holistically-nested edge detection. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1395–1403, 2015. doi: 10.1109/ICCV.2015.164.

[127] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016. URL `http://arxiv.org/abs/1611.05431`.

[128] Kai Xu, Hao Zhang, Daniel Cohen-Or, and Baoquan Chen. Fit and diverse: Set evolution for inspiring 3d shape galleries. *ACM Transactions*

on Graphics, (Proc. of SIGGRAPH 2012), 31(4):57:1–57:10, 2012.

[129] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? 2019.

[130] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 206–215, 2018.

[131] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[132] Georgios A Zachiotis, George Andrikopoulos, Randy Gornez, Keisuke Nakamura, and George Nikolakopoulos. A survey on the application trends of home service robotics. In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 1999–2006. IEEE, 2018.

Curriculum Vitae – Florian Teich

Address	Christophorusweg 14, 37075 Göttingen	**Date of Birth**	12th December 1992
		Place of Birth	Göttingen
Email	fteich1@gwdg.de	**Nationality**	German/American

Education

since 2017 Research Assistant
Georg-August University of Göttingen
III. Physics Institute
Department of Computational Neuroscience
Topic:"Graph-based Functional Object Understanding"

2015-2017 M.Sc. in Applied Computer Science
Georg-August University of Göttingen
Specialization: Computational Neuroscience
Thesis:"Multiple Extended Target Tracking using Multiplicative Error Shape Model and Network Flow Labeling"

2011-2015 B.Sc. in Applied Computer Science
Georg-August University of Göttingen
Specialization: Computational Neuroscience
Thesis: "Multichannel electrotactile biofeedback for predictive control of myoelectric prostheses"

2003-2011 General qualification for university entrance
Grotefend-Gymnasium Münden

Employment

since 2017 University of Göttingen, III. Physics Institute
Research Assistant

2015 - 2017 IVDK, Göttingen
Software-& Hardware-Administration, Software Development,
System Hardening

2014 - 2017 Institute of Computer Science, Göttingen
Web representation of the Institute, Software Development

Publications

2021 3D Object Classification via Part Graphs
Teich, Lüddecke, Wörgötter
VISAPP (2021)

2019 A Data-driven Approach for General Visual Quality Control
in a Robotic Workcell
Reich, Teich, Tamosiunaite, Wörgötter, Ivanovska
Journal of Physics: Conference Series (2019)

2017 Network Flow Labeling for Extended Target Tracking PHD filters
Yang, Teich, Baum
IEEE Transactions on Industrial Informatics (2017)

2017 GM-PHD filter for Multiple Extended Object Tracking based on
the Multiplicative Error Shape Model and Network Flow Labeling
Teich, Yang, Baum
IEEE Intelligent Vehicles Symposium (2017)

2016 Electrotactile EMG feedback improves the control of prosthesis
grasping force
Schweisfurth, Markovic, Dosen, Teich, Graimann, Farina
Journal of Neural Engineering 13.5 (2016)