

MASTER THESIS

Deep Learning Based Classification of Clothes using Point Clouds

vorgelegt von

Niklas Fiedler

MIN-Fakultät

Fachbereich Informatik

Technische Aspekte Multimodaler Systeme

Studiengang: Master Informatik

Matrikelnummer: 6803451

Erstgutachter: Prof. Dr. Jianwei Zhang

Zweitgutachter: Dr. Chao Zeng

Abstract

In this work, a classifier for clothes was developed which solely relies on depth information. The task was approached using a neural network based on the PointNet architecture. The classification of clothes serves as a use case to investigate the usability of PointNet as a classifier of non-rigid objects. To train and evaluate the network, a new dataset was created which consists of samples of eight types of clothes grasped at a single random point. In the evaluation, diverse properties of the approach are shown and analyzed. The classifier was integrated into the ROS environment to allow its usage in various robot systems. Results of the evaluation indicate, that a sufficient classification accuracy can be reached when distinguishing general types of clothes. Furthermore, diverse tools were programmed which aid with the investigation of the recorded data and classification results.

Zusammenfassung

In dieser Arbeit wurde ein Klassifikator für Kleidung entwickelt, welcher ausschließlich auf Tiefeninformationen arbeitet. Dafür wurde ein neuronales Netz basierend auf der PointNet Architektur verwendet. Die Klassifizierung von Kleidung dient unter anderem als Anwendungsbeispiel, um die Verwendbarkeit von PointNet an nicht festen Objekten zu testen. Um das Netz zu trainieren und zu evaluieren, wurde ein neuer Datensatz erstellt, welcher Aufnahmen von acht an einem zufälligen Punkt gegriffenen Typen an Kleidungsstücken enthält. In der Evaluation wurden diverse Eigenheiten des Ansatzes aufgezeigt und analysiert. Der Klassifikator wurde in die ROS-Umgebung integriert, um in verschiedenen Robotersystemen verwendet werden zu können. Ergebnisse der Evaluation zeigen, dass eine ausreichende Klassifikationsgenauigkeit erreicht werden kann, um grundlegende Typen von Kleidung zu unterscheiden. Des Weiteren wurden diverse Werkzeuge programmiert, die der Untersuchung der aufgenommenen Daten und Ergebnisse dienen.

Contents

List of Figures	v
List of Tables	ix
List of Listings	xi
1. Introduction	1
2. Fundamentals	3
2.1. Robot Operating System	3
2.2. Point Clouds	7
2.2.1. Point Cloud Representation	9
2.2.2. Point Normal Estimation	9
2.3. PyTorch	10
2.4. PointNet Architecture	11
2.4.1. Point Order Invariance	12
2.4.2. Input and Feature Transformations	13
2.4.3. Classification Performance	14
2.4.4. Classification Robustness	15
2.5. Microsoft Kinect Camera	15
3. Related Work	19
3.1. Classification Datasets	19
3.2. Shape based Object Classification	20
3.3. Clothes Classification and Manipulation	25
4. Approach	27
4.1. Dataset Creation	27
4.1.1. Technical Aspects	28
4.1.2. Object Classes	34
4.1.3. Recorded Sets	34
4.2. Classifier	36
4.2.1. Data Loader	36
4.2.2. Training	37
4.2.3. Testing	37
4.2.4. Live Classifier	38
4.2.5. File Classifier	40

Contents

5. Experiment	43
5.1. Metrics	44
5.2. Number of Points	45
5.3. Grasp Positions	46
5.4. Similar Shape	48
5.5. Significant Points	49
5.6. Classification Performance and Generalization	51
6. Discussion	57
6.1. Performance in Experiments	57
6.2. Threats to Validity	59
6.3. Applicability in Robotic Systems	60
7. Conclusion	63
Bibliography	65
Appendices	75
A. Acronyms	77
B. Experiment Results	78
B.1. Grasp Positions	78
B.2. Similar Shape	82
B.3. Number of Points	87
C. Figures	88
D. Listings	89
D.1. ROS Messages	89
D.2. Live Classifier Configuration	90

List of Figures

1.1.	Exemplary image of a robot assisting a person in dressing.	1
2.1.	Exemplary representation of message passing in the ROS environment.	4
2.2.	Comparison of various depth information representations.	8
2.3.	The PointNet architecture.	11
2.4.	Overview of approaches to achieve point order invariance.	12
2.5.	Results of robustness tests on PointNet.	15
2.6.	Detail view of the Kinect v2 sensor.	16
3.1.	Overview of the PointNet++ architecture.	21
3.2.	Result of the point dropout test on PointNet++.	21
3.3.	Classification pipeline of DGCNN.	23
3.4.	Schematic representation of the edge convolution layers.	23
4.1.	Schematic representation of the approach of this work.	27
4.2.	Exemplary image of the data collection process and setup.	30
4.3.	Screenshot of the RQT recording setup.	31
4.4.	Screenshot of the point cloud viewer.	31
4.5.	Exemplary representation of grasp areas used for data recording.	36
4.6.	Schematic presentation of the live classifier integrated in the ROS environment.	38
4.7.	Screenshot of the result visualization tool.	39
5.1.	Visualization of classification accuracy with various numbers of input points.	45
5.2.	Exemplary representation of grasp areas used for evaluation.	47
5.3.	Significant points in point clouds of various classes.	51
C.1.	Significant points during rigid object classification.	88
C.2.	Significant points during rigid object orientation estimation.	88

List of Tables

2.1.	Performance comparison of approaches to achieve point order invariance. . . .	12
2.2.	Performance comparison of input and feature transforms.	13
2.3.	Classification performance of PointNet and related methods.	14
2.4.	Time and space complexity of PointNet.	15
2.5.	Comparison of Microsoft Kinect camera versions.	17
3.1.	PointNet++ non-rigid shape classification performance on SHREC15 bench- mark set.	22
4.1.	Excerpt of the key commands of the point cloud data visualization script. . .	32
4.2.	Overview of the object classes used in the dataset.	35
4.3.	Composition and size of the dataset developed in this work.	36
5.1.	Overview of the experiments performed in this work.	43
5.2.	Comparison of classification accuracies with four classes and different grasp positions.	47
5.3.	Comparison of classification accuracies with four classes and different grasp positions.	47
5.4.	Overview of the object classes used in the similar shape experiment.	49
5.5.	Comparison of classification accuracies with two similar or diverse classes. . .	50
5.6.	Comparison of classification accuracies of the seven-class classifier without and with point normals using validation and independent test data.	52
5.7.	Classification performance of the seven-class classifier without point normals on validation data.	53
5.8.	Confusion matrix of the seven-class classifier without point normals on vali- dation data.	53
5.9.	Classification performance of the seven-class classifier with point normals on validation data.	54
5.10.	Confusion matrix of the seven-class classifier with point normals on validation data.	54
5.11.	Classification performance of the seven-class classifier without point normals on independent test data.	55
5.12.	Confusion matrix of the seven-class classifier without point normals on inde- pendent test data.	55
5.13.	Classification performance of the seven-class classifier with point normals on independent test data.	56

List of Tables

5.14. Confusion matrix of the seven-class classifier with point normals on independent test data.	56
B.1. Classification performance for four classes grasped at the top without point normals on validation data.	78
B.2. Confusion matrix for four classes grasped at the top without point normals on validation data.	78
B.3. Classification performance for four classes grasped at the top with point normals on validation data.	79
B.4. Confusion matrix for four classes grasped at the top with point normals on validation data.	79
B.5. Classification performance for four classes grasped at the edge without point normals on validation data.	79
B.6. Confusion matrix for four classes grasped at the edge without point normals on validation data.	80
B.7. Classification performance for four classes grasped at the edge with point normals on validation data.	80
B.8. Confusion matrix for four classes grasped at the edge with point normals on validation data.	80
B.9. Classification performance for four classes grasped everywhere without point normals on validation data.	81
B.10. Confusion matrix for four classes grasped everywhere without point normals on validation data.	81
B.11. Classification performance for four classes grasped everywhere without point normals on validation data.	81
B.12. Confusion matrix for four classes grasped everywhere without point normals on validation data.	82
B.13. Classification performance for similar shapes grasped everywhere without point normals on validation data.	82
B.14. Confusion matrix for similar shapes grasped everywhere without point normals on validation data.	82
B.15. Classification performance for similar shapes grasped everywhere with point normals on validation data.	83
B.16. Confusion matrix for similar shapes grasped everywhere with point normals on validation data.	83
B.17. Classification performance for similar shapes grasped everywhere without point normals on test data.	83
B.18. Confusion matrix for similar shapes grasped everywhere without point normals on test data.	84
B.19. Classification performance for similar shapes grasped everywhere with point normals on test data.	84
B.20. Confusion matrix for similar shapes grasped everywhere with point normals on test data.	84

B.21. Classification performance for diverse shapes grasped everywhere without point normals on validation data.	85
B.22. Confusion matrix for diverse shapes grasped everywhere without point normals on validation data.	85
B.23. Classification performance for diverse shapes grasped everywhere with point normals on validation data.	85
B.24. Confusion matrix for diverse shapes grasped everywhere with point normals on validation data.	86
B.25. Classification performance for diverse shapes grasped everywhere without point normals on test data.	86
B.26. Confusion matrix for diverse shapes grasped everywhere without point normals on test data.	86
B.27. Classification performance for diverse shapes grasped everywhere with point normals on test data.	86
B.28. Confusion matrix for diverse shapes grasped everywhere with point normals on test data.	87
B.29. Classification accuracy over four classes with various numbers of input points during training and testing.	87

List of Listings

4.1. The definition of the <code>ClassificationResult</code> message.	39
4.2. The definition of the <code>Classify</code> action.	40
D.1. The definition of the <code>PointCloud2</code> message.	89
D.2. The definition of the <code>PointField</code> message.	89
D.3. The configuration of the live classifier.	90

1. Introduction

In recent years, robots designed to mow lawns and to vacuum or wipe floors became popular with customers and the market is expected to grow significantly in the future [Hä16, DT21]. Thus, robots assisting in household tasks can be considered a promising field of research, as consumers are interested in the technology. While relatively simple tasks such as vacuuming a floor are solved to a marketable degree, more complex tasks as for instance sorting and folding laundry or assisting in dressing (see Figure 1.1) still need more research. This work aims to contribute to this field of research.

Robots handling clothes need to be capable of detecting, classifying, grasping, and manipulating them. A major challenge in all these tasks is that clothes are non-rigid objects. This means that their shape is usually unknown and changes. The state of a rigid object can be defined by its pose in six dimensions (three dimensions for the position and three dimensions for the orientation). In contrast, non-rigid objects can change their shape within various constraints. Consequently, the pose of each of the infinite number of points that construct the objects are part of its state yielding an infinite dimensionality of the state.

Usually, models of the clothes are used to simulate their behavior to be able to plan steps to manipulate them. But before a model can be applied to predict the behavior of the clothes, the piece needs to be identified. Reinforcement learning architectures were developed which are capable of handling clothes [TMRS11, TCUM19]. These systems are usually trained for one specific model or use the clothes class as input. Thus, classification of clothes is also



Figure 1.1.: Exemplary image of a robot assisting a person in dressing. Image used with permission from the TAMS research group of the University of Hamburg.

1. Introduction

required for more sophisticated tasks such as manipulation as first, the clothes need to be identified.

When classifying clothes in a domestic environment, invariance to several changing conditions needs to be achieved. Those conditions include the lighting or background as well as the size, color, and pattern or print of the clothes. Image-based classification techniques are generally highly sensitive to these aspects. The severity of unexpected differences to the training data is demonstrated in other domains by adversarial samples [SZS⁺14, BHG⁺19]. Image-based information is inherently dependent on the mentioned changing conditions. Alternatively, depth information can be used. It is not directly dependent on the lighting, background, color, or pattern. However, the conditions can affect the measurement quality depending on the sensor. Still, relying on depth information is a reliable method to gather information about the shape and rough texture (e. g. wrinkle pattern) of clothes [JYT⁺17]. The most common format for raw depth information without knowing the actual object surfaces are depth images and point clouds. As the information density in depth images is generally lower compared to point clouds and their reduced flexibility, this work uses point clouds as input information. Point clouds without color information can be an option to classify clothes independently of the colors in the background as well as the color and patterns of the clothes. Thus, a high grade of generalization could be reached in the classification. This would mean that a training set with relatively small differences between the training samples is sufficient to train a classification model for various clothes.

This work aims to approach the problem of asserting a specific class to a set of 3D points captured by a depth camera of clothes grasped at a single position. The application of PointNet is a widely known and used method applied for similar problems. However, prior to this work, it was not evaluated in the field of classification of grasped clothes. Further, methods of classifying non-rigid objects are usually evaluated with samples of humans or animals which do not change their shape as significantly as clothes. Therefore, this work aims to investigate the performance of PointNet in that specific task. In doing that, the foundation for complex clothes handling and manipulation tasks is created. This is accomplished by recording a novel dataset with depth information of clothes grasped at a single point, training and evaluating a PointNet model on the dataset, and finally integrating the trained model into a live classifier for a full robotic system.

This work is structured as follows: Fundamentals about the tools and components used in this work are introduced in Chapter 2. Afterward, Chapter 3 presents work related to the thesis focusing on datasets and clothes classification as well as manipulation. The approach taken in this work is outlined in Chapter 4. This is followed by experiments conducted to evaluate the work. Their results are presented in Chapter 5 and discussed in Chapter 6. Finally, in Chapter 7, a conclusion is drawn and possible future work is described.

2. Fundamentals

The approach developed in this work builds upon several components, which are presented in the following sections. Section 2.1 presents the robot operating system, a middleware for robotic systems. The live classifier proposed in Section 4.2.4 is integrated into this ecosystem to be easily usable in an existing robot setup. The main type of data processed in the developed approach are point clouds. Section 2.2 presents their properties and methods used to represent and process them. PyTorch, the deep learning framework used to define, train and test the neural networks developed, is introduced in Section 2.3. The PointNet architecture used for the classification task posed in this work is presented in detail in Section 2.4. Finally, Section 2.5 shows the features of the Microsoft Kinect depth camera used for the collection of the point cloud data.

2.1. Robot Operating System

When developing software for robots, two patterns are very common: either the whole system is implemented as a single program that handles all tasks such as computer vision and motion, or each component is written as a separate program the sum of which communicate via shared memory or a middleware. In larger applications, the second approach is generally favorable. When developing small components, experts can focus on specific parts of the system and their interface without further regard towards the remainder of the pipeline. The strict code encapsulation also helps with code management, maintainability, and also reusability of code as one package might be usable in various robotic setups. The robot operating system (ROS) is such a middleware [QCG⁺09]. It was preferred for this work because it is especially popular in robotics research.

In the ROS ecosystem, small software components called nodes are developed individually. Usually, a system solving a complex task is composed out of multiple nodes each solving a specific subtask. Nodes share information by publishing and subscribing to topics. On these topics, an N:N communication can be achieved as multiple nodes can publish on and subscribe to the same topic. The ROS master connects the nodes and enables the message passing as shown in Figure 2.1. ROS2, the successor of ROS improves some of the features while it keeps the general structure. The main change over ROS is the transition from the ROS master to a data distribution service (DDS). However, it is not directly compatible with ROS. This work is developed for ROS because the majority of current robotic systems still rely on it and are not yet upgraded to ROS2.

In the following sections, the three inter-node communication channels messages, services, and actions are explained in detail. Further, other essential components of the ROS ecosystem are presented.

2. Fundamentals

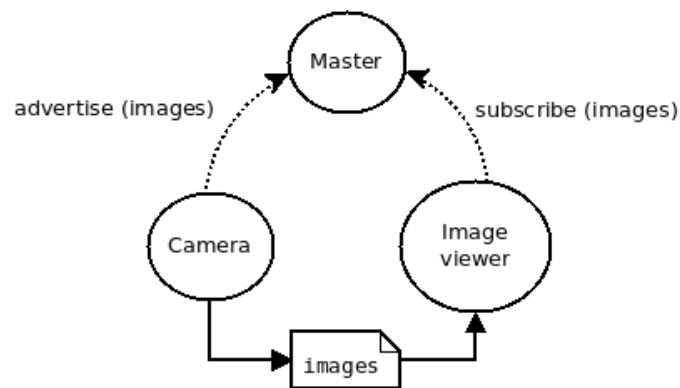


Figure 2.1.: Exemplary representation of message passing in the ROS environment. A node publishing to a topic advertises the topic to the ROS master. Another node (the image viewer in this example) subscribes to the topic. Then, messages on the topic (images in this case) are passed directly between the nodes. Figure source: [1] (modified)

Messages

As mentioned, the main communication between nodes is conducted by publishing and subscribing to topics on which ROS messages are sent. The messages are small data packages sent and received over the network by nodes. By using the network for communication instead of shared memory, it is possible to use multiple machines in the same ROS setup with nodes communicating with each other. Messages published on a specific topic are of a specific message type. The message type defines what kind of information is transferred. A message type consists of named fields each assigned a data type. These types can be primitive types such as boolean, int, float, or string or other message types. Also, lists of undefined length of a type are an option. Additionally to the primitive types, more complex message types are available in standard packages. The most relevant standard message type is the Header. It combines a timestamp with a frame id and thus can be used in a message type to specify when and where the data was ascertained. In the `common_msgs` repository [2], which includes several ROS packages, various messages are defined for diverse use cases. If a more specific message type is required in an application, the existing messages can be used as parts of a more complex type. Some of the types are available in stamped and non-stamped versions. Stamped indicates, that a field of the type Header is added to the definition. Thus, the stamped version can be used directly as a message while the non-stamped version is supposed to be used as part of another type.

Services

In contrast to asynchronous data transfer as implemented by messages, services offer a synchronous way of data transfer. A service client sends a request to the service provider. After processing the request, the service provider replies with a response to the client. Services

are designed in a similar manner as messages. However, their definitions are composed of two parts: the request and the response. Because of this, services are more specific to the domain they are used in. Still, the most simple and common service type, the Trigger service is provided by default. It combines an empty request with a response consisting of a boolean value indicating success and a text message. Besides this, an empty service, as well as a SetBool service, are also available.

Actions

Actions extend the capabilities of services with a feedback component. While services are designed to trigger tasks that are handled quickly such as setting parameters, actions are meant to trigger longer running processes. Also, a service call blocks further processing of the client, whereas an action client can send a request, continue with further processing, and is notified about feedback and the response via a callback function. Examples are triggering a robot motion or, as they are used in this work, the recording and classification of a point cloud. The additional feedback component is added in the action definition similar to the response in the service. Following the definition of the request and the response, the data type of the feedback message is defined in the same file.

Launch Files

In a large and complex system, a large number of ROS nodes are running simultaneously. Launch files define how nodes are started with an XML-like syntax. Parameters can be loaded for specific nodes. Further, when calling a launch file, arguments can be defined. Using these arguments, command groups can be selected and node parameters changed. With these groups of commands, parts of a large software stack can be activated or deactivated on launch as required. For example, this allows switching between a real-world and a simulation configuration of a system by adapting a single command-line argument. Additionally, it is possible to include other launch files. Thereby multiple layers of launch file abstraction can be implemented as a single launch file could combine launch files that handle smaller parts of the software stack and so on.

Packages

In the ROS ecosystem, the software is organized in packages. A package may contain one or multiple nodes, configuration files, launch files, and definitions of messages, services, or actions. The package concept adds a software management layer to the ROS environment. A package description in the form of a `package.xml`-file is required. In the description file, the name of the package, the name and contact information of the authors and maintainers, license information, and dependencies to other packages are mandatory. In conjunction with a `CMakeLists.txt`-file, dependencies can be resolved when the package is built.

2. Fundamentals

Tools

Additional to the general functionality, ROS provides several tools which help in developing, maintaining, and debugging robotic systems. Many of the tools focus on data visualization as it is an integral necessity for a user or researcher to understand what is happening in the system. The tools presented in the further sections are only exemplary and focus on the standards, as developers design various additional tools for the ecosystem as needed.

ROS Bags

When integrating software in robotic systems, complex setups may be required to achieve a desired system state. Thus, researchers need to be able to collect valuable data in such situations to inspect it in detail or use it later on in tests. In the ROS environment, the collection of such data is done by recording ROS bags. They provide an option to record all messages on all or a selection of topics. During recording, the time of receiving the messages is saved. Thus, a bag can be played back when needed. For example, a bag can be recorded on a large system during an experiment. Then, this bag can be replayed later on different systems to evaluate or test a single node. It is also possible to change the playback speed when required.

RQT

While the communication via messages, services, and actions is sufficient between nodes, it is not intuitive for a user. With RQT, ROS offers a framework for user interface design within a robotic system. It allows the user to interact with the robot setup using various graphical user interfaces. The general tool relies on various plugins each of which is designed to solve a single visualization or user interaction task. Both, the tool itself and its plugins are written using the QT framework [3]. In the following, some of the plugins available are presented.

- **Image View:** Visualizes Image messages and is therefore the de facto standard in introspecting any kind of image stream in the ROS environment. In this work, it is used to provide feedback during data recording (see Figure 4.3).
- **Dynamic Reconfigure:** Allows the user to change the parameters of nodes while they are running. Thus, the behavior and movements of the robot can be adapted on the fly. However, this requires a separate description of the parameters, their types, and value ranges. Further, a callback to handle the changed parameters has to be implemented by the node.
- **Node Graph:** Gives an overview of the whole system by visualizing the flow of information. A graph is shown indicating which nodes publish and subscribe to which topics.
- **Console:** Filters and sorts logging information in the system. Logging messages can be inspected filtered by their severity and the nodes which issued them.

RViz

In contrast to RQT, RViz focuses on data visualization in cartesian space. It features a 3D visualization of one or more robot models in their current state. The RViz window is generally separated into three areas. On the left, the visualized components of the system can be selected. This helps focus on a specific part of the system. Also, component-wise visualization parameters can be adjusted. On the right, the parameters of the view and perspective are configurable. In the center, the 3D visualization is located. Via the mouse, users can rotate, translate and zoom their current viewpoint.

While RViz is capable of visualizing various kinds of standard messages including Point-Cloud2, these are not sufficient for all domains. In those cases, RViz markers can be specified and published by a node as a message. The marker message (defined in the visualization message package) allows to define custom shapes that represent domain-specific information. Similar to RQT, its capabilities can be extended using plugins. However, it does not completely rely on them as RQT does.

Simulation

Simulators are a popular option to evaluate software components or a whole software stack. When a specific system state is required, it can be generated in a simulator. Also, depending on the requirements and computation performance available, applications can be simulated faster than real-time or in parallel. This is usable both in testing a large range of scenarios and in training applications that learn behavior by exploration such as deep reinforcement learning. Training and evaluating a system in simulation also significantly reduces the load on the robot's hardware. Especially with the increase of computing power available and the advancements in deep reinforcement learning over recent years [ADBB17], the relevance of simulation in robotics increased significantly. As photo-realistic image generation is possible, there are multiple examples of training data generation in a simulation environment [BEF⁺21].

While many general physics simulators exist, only a few are easily usable for robotic applications. And only a fraction of those offers an integration into the ROS environment. The Gazebo simulator [KH04] was originally designed for the Player/Stage framework [GVH03], the ideological predecessor of ROS. Later, it was adapted and integrated into the ROS environment. In recent years, the Webots simulator [Mic04] also gained popularity because of advantages in computation performance. This work does not utilize simulation for the dataset generation as detailed in Section 4.1.

2.2. Point Clouds

Together with depth images and voxel grids, point clouds are one of the most common representations of depth measurements. Depth images (see Figure 2.2 (c)) are a less dense representation than point clouds, as they only encompass rectangular areas in the sensors field of view (FOV) because they project depth measurements on a two-dimensional plane.

2. Fundamentals

Moreover, the distances between the points are fixed in image space and it is impossible to model measurements behind each other. Without knowing the characteristics of the camera, it is not possible to derive information in Cartesian space from the data. Despite their shortcomings, depth images are often preferred because, in many regards, they can be processed similar to images for example in convolutional neural networks.

Voxel grids (see Figure 2.2 (d)) divide the three-dimensional Cartesian space into voxels. A voxel is, as their name, a combination of the words *volume* and *pixel* indicates, a volumetric pixel. They can be used to describe the occupancy of volumes, but also include more information such as the color or type of the objects contained in the area. Voxel grids solve many of the problems occurring with depth images as they allow the representation of measurements behind each other and represent the volumes in Cartesian space. Similar to depth images, they can be processed by conventional methods such as computer vision filters or convolutional networks adapted for three-dimensional input. However, they are very restrictive in the resolution because the storage required grows in a cubical manner. This means that the information contained in a voxel grid is usually very sparse even compared to depth images.

In contrast, point clouds (see Figure 2.2 (b)) work the other way around: instead of describing what is in a predefined area, they represent measurements by describing where something is in the space. They are defined as a set of points in 3-dimensional Euclidian space.

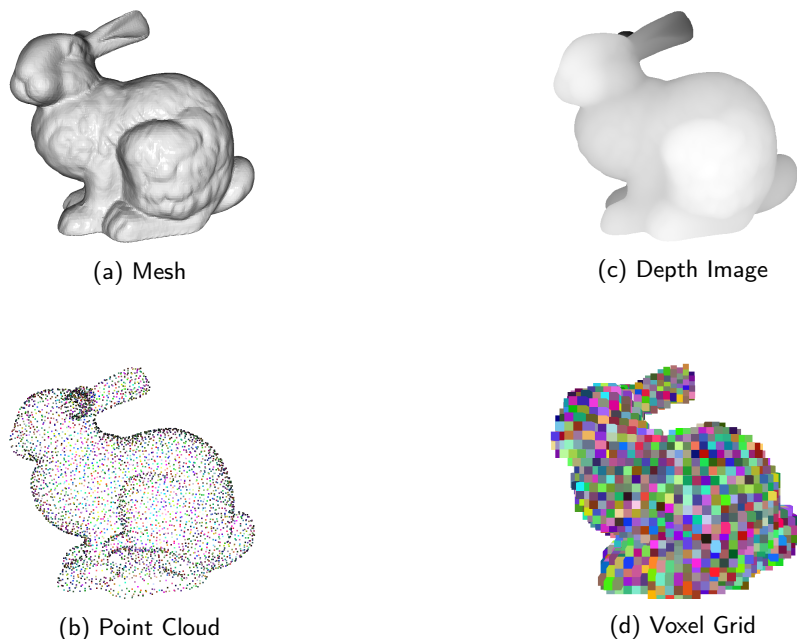


Figure 2.2.: Comparison of various depth information representations at the example of a single shape. The mesh in (a) is represented by a point cloud (b), a depth image (c), and a voxel grid (d).

Depending on the use case, additional information such as the color is included with each point. By describing specific points in space, positions and distances in the point cloud directly reflect the positions and distances in the real world. Compared to the other methods, point cloud processing is not possible with many conventional methods as there is no method of converting them to a canonical form available. This means that there is no way to sort a point cloud for example into a grid without massively decreasing the information density. This means that new forms of processing them in a neural network had to be developed when PointNet was introduced (see Section 2.4.1). Point clouds are for example generated by 3D scanning methods such as contact 3D scanners or contact-free methods such as time-of-flight (ToF) or structured light. Also, photogrammetry (and especially stereophotogrammetry) is a method of extracting 3D information from 2D data. RGB-D cameras such as the Microsoft Kinect (see Section 2.5) combine a 3D scanner with an RGB camera in a single device. The libraries PCL [RC11] and Open3D [ZPK18] are commonly used to process point cloud data.

2.2.1. Point Cloud Representation

Point clouds are represented by a set of points consisting of their positions and, as mentioned, in some cases by additional features such as color. Thus a point cloud consisting of n points can be formally represented by $\{P_i | i = 1, \dots, n\}$ with the position of each point P_i defined as (x, y, z) [QSKG17].

In the ROS environment (see Section 2.1), point cloud data is transferred in form of the `PointCloud2` message [4] (see Listing D.1), which is part of the `sensor_msgs` package. It is designed to be flexible and to be used for different kinds of point clouds. The message mainly contains a `Header`, the point cloud size, definition of the channels of the points, the length of a point and a row of points in bytes, and an array of bytes containing the raw data. Each channel is defined by a `PointField` message [5] (see Listing D.2). It specifies the name, datatype and element count of each field. Thus, `PointCloud2` messages can be interpreted without any additional information. All this information is used to allow the decoding of the array of bytes that contains the raw data in a compressed form. The `sensor_msgs` package also provides helper functions to convert and conveniently extract information from `PointCloud2` messages.

2.2.2. Point Normal Estimation

When a point cloud is used to represent an object, it is a collection of points sampled on the object's surface. However, no matter the point density used, the points are only able to capture the position of the surface at that point and not its orientation. The orientation of a surface is described by its surface normals. In three-dimensional space, a surface normal is a vector perpendicular to the tangent plane of the surface at a specific point in every axis. Due to this definition, the vector can point in two directions: inwards and outwards. Usually, the vectors are scaled to the unit length meaning that their length is 1. While the surface normals can be calculated based on the surface tangents when a full surface is given, this is not the case for point cloud measurements because the surface is not known and only described by the points. Therefore, point normals need to be estimated based on the points

2. Fundamentals

describing the surface. In that case, for each point, the N nearest neighbors are collected, and using them, a surface tangent is estimated followed by the calculation of the surface normal of that tangent. During this process, the density of sample points and the magnitude of random sensor error have a large impact on the quality of the estimated surface and thereby the estimated point normals. With a low point density, uneven parts of the surface might not be considered resulting in an estimated even surface. The opposite is caused by the random sensor error in the depth measurements as a random offset between the points can lead to estimating an uneven surface given an even one. This can be approached by adapting parameters such as N , the number of neighbors considered. But because no ground truth of the point normals for the scanned shapes is available, it is not possible to tune the parameters using a quantitative metric. Using point cloud measurements, it is not known, whether the individual point normals are pointing inwards or outwards. In PCL, point normals are oriented towards the view point [6]. Open3D does not enforce such a constraint [7]. However, it does offer a method to align the normals consistently given a number of nearest neighbor points to consider. Then, the correct direction is still not known, but they are aligned consistently.

2.3. PyTorch

When developing deep learning models, common structures and algorithms are used in various approaches. Deep learning frameworks provide an environment, general structure, algorithms, and tools necessary for the development, training, and usage of neural networks. PyTorch is such a deep learning framework mainly developed by Facebook AI researchers [PGM⁺19]. Its development focused on combining usability and processing speed. This was achieved by supporting an iterative programming style as it is commonly used in Python. Models can be defined in Python code which is intuitive for users and allows flexible experiments. As every component of PyTorch can be accessed as a regular Python program, the user retains control over every part even in complex architectures. The feature is not only beneficial in the definition of a model but also a significant help in debugging, as every aspect of a model can be inspected transparently and intuitively. However, to be competitive with other deep learning frameworks, PyTorch also has to provide a high runtime performance on both the CPU and GPU. This was achieved by implementing the core of the library in C++. The C++ components include the tensor data structure and GPU as well as CPU operators. Also, fundamental computation functions such as the automatic differentiation system are implemented in C++. Thereby, the global interpreter lock (GIL) of Python is bypassed. The GIL is the reason that it is not possible in Python to execute multiple threads in parallel [EE19]. The original multiprocessing module which bypasses the GIL by spawning multiple processes was extended in `torch.multiprocessing` to optimize parallel processes implemented in Python. Additionally features such as a separate control and data flow, a custom caching tensor allocator and reference counting were implemented. The combination of high usability and efficient processing makes PyTorch uniquely capable especially for researchers when compared to other deep learning frameworks such as Tensorflow [ABC⁺16]. Recently, the gap between Tensorflow and PyTorch was significantly reduced as Tensorflow 2 was introduced. Still, PyTorch is favored in many research environments.

2.4. PointNet Architecture

PointNet is a neural network architecture presented by Qi, Su et al. in 2017 [QSKG17]. It is designed to be capable of object classification, part segmentation, and scene semantic parsing on point cloud data. It was one of the first approaches which process raw point clouds and is similarly accurate in classification tasks as competing conventional approaches but significantly more efficient. The general architecture is shown in Figure 2.3. As this work focuses on the classification task, this overview of PointNet also prioritizes the classification aspects of the architecture.

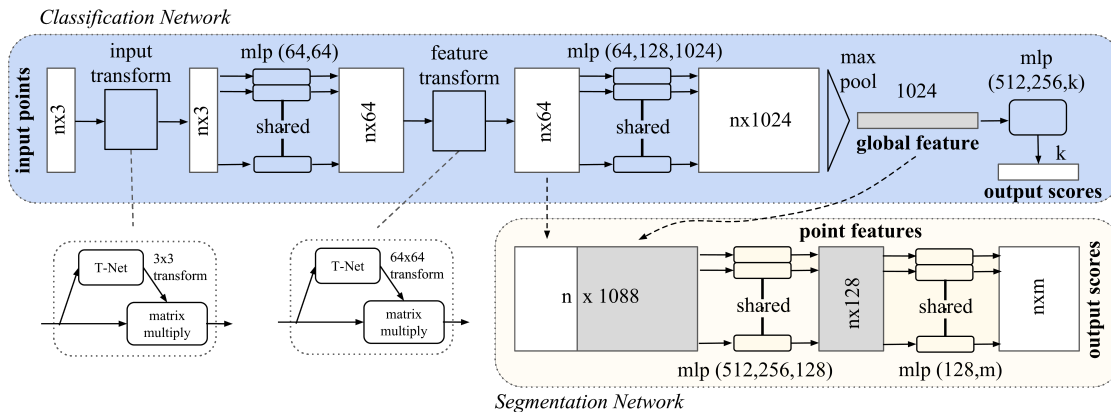


Figure 2.3.: The PointNet architecture [QSKG17].

When applied on rigid objects, methods processing point clouds need to consider permutations of the points in the set, and rigid motions (rotations and translations along three axes). The authors note that prior to their work, to use point cloud data in neural networks, the data was usually transformed into 3D voxel grids or collections of images. Due to the nature of point clouds, this results in very sparse input data for the neural networks and also quantization artifacts.

By default, the input of PointNet consists of 1024 three-dimensional points. However, depending on the application, the number of points and their dimensionality can be adapted. The authors state, that both the space and time complexity scale linearly to the input size. While the dimensionality of the input is fixed in the architecture, the number of input points is not. This is because the points are processed independently from each other until the max-pooling stage. The max-pooling itself is only dependent on a number of points $n \geq 1$. The shape of its output is independent of the number of input points. This allows for more flexibility in the input size both during training and testing.

PointNet is available implemented in Python using Tensorflow on GitHub [8]. The repository is maintained by the authors of the original paper. Since 2019, Xu Yang is implementing and refining a version of the network in PyTorch [10]. The latter is preferred in this work because PyTorch allows easier debugging and is more established in the research community (see Section 2.3).

2. Fundamentals

2.4.1. Point Order Invariance

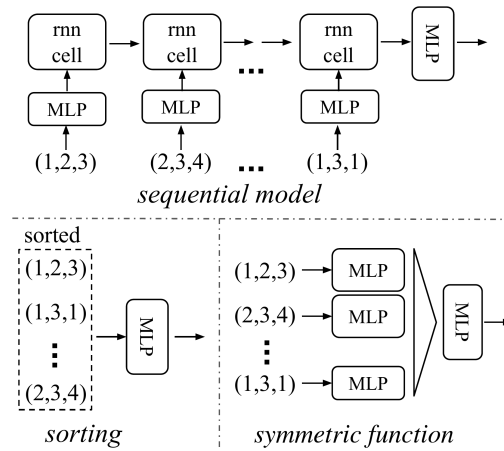


Figure 2.4.: Overview of approaches to achieve point order invariance. [QSKG17] (modified)

As mentioned in Section 2.2, point cloud measurements are an unordered set of points meaning that a classification approach has to achieve a point order invariance. The authors describe three approaches to achieve the invariance. Figure 2.4 sketches the methods and the classification performance of the approaches is compared in Table 2.1. The unsorted set of points is directly piped into a multi-layer perceptron (MLP) as a baseline resulting in an accuracy of 24.2%. While the accuracy is increased to 45% by sorting the set of points (see Figure 2.4 bottom left), there is no stable ordering for points in high dimensional space with respect to point perturbations.

The second approach is piping the points sequentially into a sequential model in form of a recurrent neural network such as an LSTM [HS97] (see Figure 2.4 top). However, while using this approach increases the accuracy to 78.5%, it is still not optimal, as the order of data fed into recurrent neural networks matters (shown by Vinyals et al. in [VBK16]).

The authors propose to solve the problem with the third method, using a symmetric function (see Figure 2.4 bottom right). A symmetric function has the property that its value stays the

Table 2.1.: Performance comparison of approaches to achieve point order invariance on the ModelNet dataset. [QSKG17]

Approach	Accuracy (%)
MLP (unsorted input)	24.2
MLP (sorted input)	45.0
LSTM	78.5
Attention sum	83.0
Average pooling	83.8
Max pooling	87.1

same for every order of the input variables. Using features extracted from the individually processed points as input of the symmetric function yields an output that is not only robust to different input orders but truly invariant. Three symmetric functions were tested and max-pooling performed best with an accuracy of 87%. Thus, it was used in PointNet.

However, processing the points completely independently from each other until their features are combined in a single stage, results in the network capturing mainly global features. This means that the network is not able to preprocess features in a local context.

2.4.2. Input and Feature Transformations

As mentioned, ambiguities due to geometric movements of the observed objects (translation and rotation) need to be taken into account. This is implemented in PointNet by applying a 3x3 transform matrix on the 3D input and a 64x64 matrix on the 64-dimensional features extracted from each point individually. The transformation matrices are generated by T-Nets, a structure that learns to generate a transformation matrix based on a single point (see Figure 2.3). To evaluate the effect of the transformation step, their classification performance is compared to the vanilla version of PointNet without any transform layers in Table 2.2. Introducing the input transformation to canonicalize the input data increased the classification performance from 87.1% to 87.9%. However, the accuracy drops to 86.9% when the feature transformation is applied. It was suspected that the reason for this might be that learning to produce a valid transformation matrix with such a high dimensionality was too complex. Therefore, the loss defined by equation 2.1 is introduced to approach the high complexity of generating the 64x64 transformation matrix by constraining it to be as close as possible to an orthogonal matrix. A matrix A is an orthogonal matrix if $AA^T = I$ with I being the identity matrix.

$$L_{reg} = \|I - AA^T\|_F^2 \quad (2.1)$$

Thus, L_{reg} penalizes a diversion from an orthogonal matrix. Using this loss, the classification performance was increased to 87.4%. The best result (89.2%) was achieved by combining the input transform with the feature transform including the regularization loss L_{reg} . Thus they are included in the PointNet architecture (see Figure 2.3).

Table 2.2.: Performance comparison of input and feature transforms. [QSKG17]

Transform	Accuracy (%)
none	87.1
input (3x3)	87.9
feature (64x64)	86.9
feature (64x64) + reg.	87.4
both	89.2

2. Fundamentals

2.4.3. Classification Performance

Especially as the PointNet architecture was one of the first approaches to make use of raw point cloud data, the authors compare the classification performance to existing approaches. The results of the analysis are listed in Table 2.3. Results of PointNet++, which is discussed in more detail in Section 3.2, are also included in the table. It was shown that PointNet performs better or competitively when compared to conventional approaches. The other approaches in the table use a mesh, volumes, or multiple images from various views as input. Considering robotic applications in which measurements of a depth camera are used so that using multiple views of the same object is often not possible especially with non-rigid objects, PointNet becomes the only viable option for shape-based classification.

Table 2.3.: Classification performance of PointNet and related methods on ModelNet40 [WSK⁺15]. Compiled from [QSKG17] and [QYSG17].

	Year	Input	# Views	Accuracy Avg. Class	Accuracy Overall
SPH [KFR03]	2003	mesh	-	68.2	-
3D ShapeNets [WSK ⁺ 15]	2015	volume	1	77.3	84.7
VoxNet [MS15]	2015		12	83.0	85.9
Subvolume [QSN ⁺ 16]	2016		20	86.0	89.2
LFD [CTSO03]	2003	image	10	75.5	-
MVCNN [SMKLM15]	2015		80	90.1	-
PointNet (vanilla) [QSKG17]	2017	point cloud	1	-	87.2
PointNet [QSKG17]				86.2	89.2
PointNet++ [QYSG17]				-	90.7
PointNet++ (with normals) [QYSG17]				-	91.9

Another advantage of PointNet is runtime performance because the architecture is processing the dense information more efficiently compared to conventional approaches. Table 2.4 gives an overview of the approaches competing with PointNet. The number of parameters of the model and required floating point operations (FLOPs) per sample are compared. It is evident that PointNet is significantly smaller and more efficient than its competitors. The low number of parameters is achieved with the MLP layers using shared weights. In comparison, convolutional layers slightly increase the number of parameters and significantly increase the FLOPs per sample.

These capabilities of competitively high classification precision and high runtime efficiency are essential for a live classifier approach. During live classification, the delay between data acquisition and the classification result should be as small as possible. This is especially crucial on a robotic system as many software components share the same hardware.

Table 2.4.: Time and space complexity of PointNet compared to competing approaches at the time it was released. [QSKG17]

	# Params	FLOPs/Sample
PointNet (vanilla) [QSKG17]	0.8M	148M
PointNet [QSKG17]	3.5M	440M
Subvolume [QSN ⁺ 16]	16.6M	3633M
MVCNN [SMKLM15]	60.0M	62057M

2.4.4. Classification Robustness

The authors performed robustness tests on the classification performance. PointNet was evaluated in three scenarios: missing data, adding outliers, and perturbation noise (see Figure 2.5). In the first scenario, a certain ratio of points is removed from the input data. The performance of furthest and random input sampling are compared. The second scenario tests resilience against outlier points by adding random points to the input data. An input of normalized X-, Y-, and Z-coordinates with and without the point density is tested. In the third scenario, the effect of perturbation noise is evaluated. The noise is simulated by applying offset sampled from a normal distribution to the original data. The X-axis of the plot defines the standard deviation of the noise added to the input data. It is not entirely clear, whether all three tests were performed with the architecture trained on the altered data, or if this just applies to the scenario of adding outliers. However, the results shown in the PointNet++ paper (see Figure 3.2) indicate that the latter applies.

2.5. Microsoft Kinect Camera

Recording depth information requires camera systems with more capabilities compared to RGB image information. Passive depth camera setups such as stereo cameras usually require extensive post-processing or complicated setups and are susceptible to the correspondence problem. When a low measurement delay and high precision are required from a compact setup, active camera systems are preferable. While passive cameras only perceive their environment using their sensors, active sensors send out a signal which is then received back

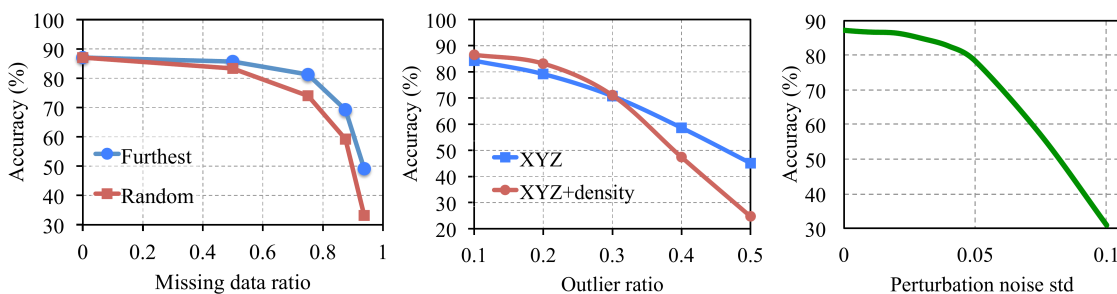


Figure 2.5.: Results of robustness tests on PointNet. [QSKG17]

2. Fundamentals

by the sensor. Depending on the altering of the signal emitted by the sensor, measurements about the environment are inferred. Despite its advantages, the measurement quality of active systems decreases significantly when the emitted signal is obstructed. In the case of most active depth cameras, this means that the cameras are sensitive to direct sunlight. However, at least in a laboratory setup, this can be prevented reliably. Often, depth cameras record both depth and image information simultaneously.

The point clouds recorded in this work were recorded using a Microsoft Kinect v2. It captures both depth and image data. Due to its widely usable driver support, precise measurements, and relatively low price, the Microsoft Kinect cameras are very popular in robotic research applications. In this work, the Kinect v2 was used because the newer and more capable Azure Kinect was sold out. Figure 2.6 shows a Kinect v2 camera mounted on a tripod. The sensors featured by the camera are labeled.

In their work, Tölgyessy et al. compared the Azure Kinect to its predecessors Kinect v1 and Kinect v2 [TDCH21]. They compiled the technical data of the cameras (see Table 2.5). While the Kinect v2 is sufficient in most regards for the task of clothes classification, its specified measuring distance is not optimal for classifying objects held close to the camera. There, the wide FOV depth recording capabilities of the Azure Kinect are a significant advantage. The Azure Kinect was originally developed as a component of the HoloLens 2. In the head-mounted display for augmented reality, the narrow FOV camera is used for navigation in space and to gain a general measurement of the environment of the user. The wide FOV is used to track the user's hands in a close range of the device. When the technology was packaged as a single camera unit, the two-FOV setup was kept. It might also be usable in the classification of objects held close (less than 0.5m) to the camera. However, while the system is generally designed for robotic setups, the train- and test data is recorded in a setup without a robot. Thus, the shortcomings of the Kinect v2 are not relevant to the result of this work.

The Kinect v2 measures depth data with the active ToF method. Depth information is inferred from measurements of the time it takes light to travel from an emitter in the

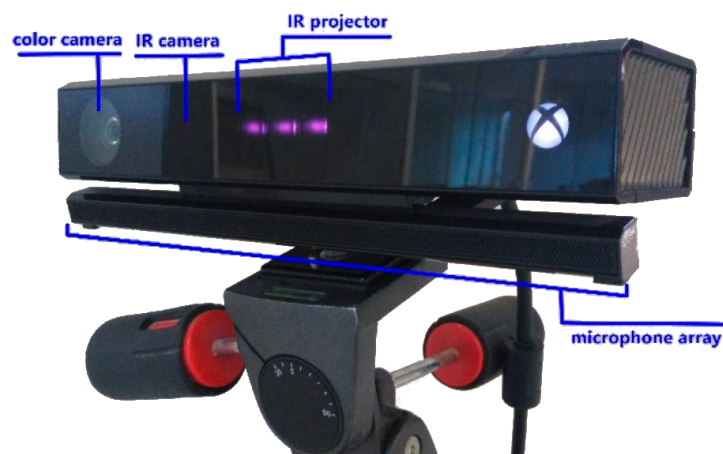


Figure 2.6.: Detail view of the Kinect v2 sensor. [JYT⁺17]

Table 2.5.: Comparison of Microsoft Kinect camera versions. [TDCH21]

	Kinect v1	Kinect v2	Azure Kinect
Color camera resolution	1280 × 720 px @ 12 fps 640 × 480 px @ 30 fps	1920 × 1080 px @ 30 fps	3840 × 2160 px @ 30 fps
Depth camera resolution	320 × 240 px @ 30 fps	512 × 424 px @ 30 fps	NFOV unbinned: 640 × 576 px @ 30 fps NFOV binned: 320 × 288 px @ 30 fps WFOV unbinned: 1024 × 1024 px @ 15 fps WFOV binned: 512 × 512 px @ 30 fps
Depth camera field of view	57° × 42° alt. 58.5° × 46.6°	70° × 60° alt. 70.6° × 60°	NFOV: 75° × 65° WFOV: 120° × 120°
Specified measuring distance	0.4 – 4m	0.5 – 4.5m	NFOV unbinned: 0.5 – 3.86m NFOV binned: 0.5 – 5.46m WFOV unbinned: 0.25 – 2.21m WFOV binned: 0.25 – 2.88m
Depth sensing method	Structured light pattern projection	ToF	ToF

measurement device to the object and back to the device. By measuring the time t in seconds between emitting and receiving the light, the distance d in meters can be inferred with the formula $d = \frac{ct}{2}$ with c being defined as the speed of light in meters per second. [BOL⁺05]

The camera was integrated into the ROS environment using the `iai_kinect2`-package [11] which is based on the `libfreenect2` driver [12]. The package includes a calibration tool, a library for depth registration, a bridge between the `libfreenect` driver and the ROS environment, and a viewer for images and point clouds. In this work, mainly the `kinect_bridge` is used to publish recorded point clouds and images in the ROS environment.

When calibrating the camera, three steps are conducted using a predefined grid and the calibration tool provided by the camera driver package. First, the depth image is calibrated. Second, the RGB image is calibrated. In the final calibration step, matching between the RGB image and the depth information is established. In this work, the third step was not necessary, as point clouds and RGB images are not processed in conjunction.

3. Related Work

In the following sections, approaches related to the work conducted in this thesis are presented. The works are separated into three groups: classification datasets similar to the one created in this work, methods for shape-based object classification, and approaches to clothes classification and manipulation.

3.1. Classification Datasets

At the time of writing, no dataset for point cloud based clothes classification is publicly available as part of a publication or on popular platforms such as Kaggle [13]. Therefore, this section focuses on datasets designed for image-based clothes classification as well as general object classification on point clouds. A new dataset for point cloud based clothes classification will be created in this work and presented in Section 4.1.

The most prominent example of a cloth classification dataset is Fashion-MNIST [XRV17]. However, it is rather designed to replace the MNIST dataset [Den12] as a generally simple and small image classification benchmark than as a benchmark for cloth detection specific systems. Therefore, it features a lot of similarities to the original MNIST such as greyscale images with a resolution of 28 by 28 pixels and 10 object categories. Additionally, it has the same size as MNIST (60.000 training images and 10.000 test images).

A popular dataset that is specifically designed to benchmark systems for the classification of clothes is Deepfashion [LLQ⁺16]. Their images were collected on the online shopping websites *Forever21* and *Mogujie* as well as entering clothing descriptions from online retailers as queries into the Google Image search. The authors note that their dataset includes both professional product photos as well as pictures uploaded by customers wearing the clothes in various situations. The resulting 1,320,078 images from the shopping websites and 1,273,150 images gathered from Google Images were filtered using AlexNet [KSH17] and the input of human annotators. This process resulted in a set of 800,000 images. Additionally to their class, images are each assigned a subset from 1000 attributes. The attributes are grouped into the five categories “texture”, “fabric”, “shape”, “part”, and “style”.

To evaluate their work in “3D ShapeNets: A Deep Representation for Volumetric Shapes”, Wu et al. created the ModelNet dataset [WSK⁺15]. It contains 151,128 3D CAD models of 660 unique object categories. For benchmarks of shape recognition systems, smaller subsets are selected. ModelNet40 consists of 48,000 CAD models (38,400 for training, 9,600 for testing) belonging to 40 classes. ModelNet10 is a subset of ModelNet40 consisting of 4899 CAD models (3991 for training, 908 for testing) which are designated to 10 classes [CCG⁺18]. In recent years, ModelNet became the de facto standard benchmark for evaluating 3D classification methods.

3. Related Work

The Computer Science Department of the University of Utrecht organizes a yearly 3D Shape Retrieval Contest (SHREC) [14]. In multiple tracks, various challenges are posed. Most notable in the context of this work is the track “Canonical Forms for Non-Rigid 3D Shape Retrieval” from SHREC’15 [PSR⁺15] [15]. For their challenge, Pickup et al. created a new dataset by selectively combining existing datasets from previous challenges. Specifically, the datasets from “Shape Retrieval of Non-Rigid 3D Human Models” from SHREC’14 [PSR⁺16] and “Shape Retrieval on Non-Rigid 3D Watertight Meshes” from SHREC’11 [LGB⁺11].

3.2. Shape based Object Classification

In this section, work covering object classification based on shapes is presented. As noted in the paper presenting PointNet [QSKG17], one of the major tasks of deep learning on point cloud data is to gain a point order invariance. Before the method of applying a symmetric function, this was approached in various ways such as using an occupancy grid or feeding multiple 2D views of the object into a convolutional neural network.

Maturana and Scherer approach the task of object classification based on point clouds with their VoxNet architecture in 2015 [MS15]. They generate a three-dimensional ($32 \times 32 \times 32$) occupancy grid from the input point clouds. These grids are used as input for 3D convolutional layers. Via multiple convolutional layers and pooling, the data is further compressed. These are followed by a fully-connected layer with 128 neurons. The size of the succeeding output layer is determined by the number of classes required for the task. On the ModelNet benchmark sets, the method outperforms ShapeNet [WSK⁺15]. While ShapeNet achieves an accuracy of 0.77 on ModelNet40, VoxNet achieves an accuracy of 0.83 (0.84 and 0.92 on ModelNet10 respectively).

When PointNet was developed, the state-of-the-art method for 3D shape classification was image-based and used multiple 2D views of a single shape which were fed into a series of CNNs [SMKLM15]. While this method does not directly process shapes, it was trained on the shapes from the ModelNet dataset. The shapes are rendered from multiple viewpoints into 2D images. The resulting images are fed in parallel into CNN layers with shared weights resulting in multiple descriptors. To aggregate the descriptors, a view-pooling layer is used, which performs an element-wise max-operation over all views. The output of the view-pooling layer is processed by further convolutional layers and then fed into a fully connected layer. Similar to other deep learning classification approaches, the output layer is sized to accommodate the number of classes supported. With pre-training on ImageNet [DDS⁺09] and fine-tuning on ModelNet40, a classification test accuracy of 88.6% using 12 views and 90.1% using 80 views was reached. However, the requirement of rendering the 3D shapes into 2D images makes this method unsuitable for processing point clouds recorded from a single view. It might be possible to reconstruct the surface of an object scanned from multiple angles, but this is impractical for live classification.

Since the presentation of PointNet, most approaches follow a similar structure or use PointNet itself as a part of a greater structure. The main reason for this trend is the runtime efficiency and realization of point order invariance (see Section 2.4.1). It was shown, that a symmetric function, (especially the max function) greatly improves the classification

3.2. Shape based Object Classification

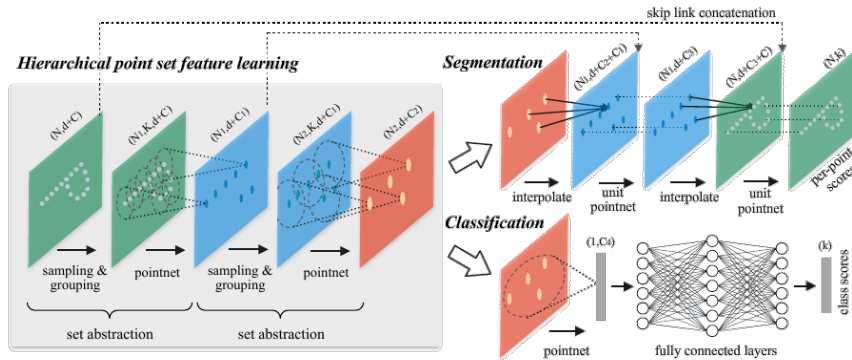


Figure 3.1.: Overview of the PointNet++ architecture. [QYSG17]

performance on raw point cloud data (see Table 2.1). As already mentioned in Section 2.4, PointNet++ improves upon the results of PointNet. This is done by using the original PointNet structure as a component in a greater system. Figure 3.1 gives an overview of the PointNet++ architecture. In multiple stages, points are sampled and grouped. These groups are processed separately by PointNet-like structures. Thereby, local features (features extracted at the beginning of the pipeline) are considered in a global context at the end of the pipeline. While the classification precision is improved compared to PointNet (see Table 2.3), the runtime efficiency is decreased significantly.

The resilience of PointNet++ to missing points was tested in various configurations (see Figure 3.2). The results were also compared to the vanilla version of PointNet. In the figure, DP indicates that a random input dropout was applied during training. The four figures in the left part of Figure 3.2 show examples of a chair represented by 1024, 512, 256, and 128 points. The graph shows a steady performance of both PointNet and PointNet++, as long as the input consists of at least 256 points when the input dropout was present during training. Also, the gap in performance between the two architectures is apparent. Without input dropout during training, both architectures perform significantly worse. Notably, PointNet performs considerably better in this scenario. When comparing the input dropout performance between Figure 2.5 and Figure 3.2, note the difference in their scale.

Further, especially relevant for this work, its performance in non-rigid shape classification was evaluated on the SHREC15 benchmark set [PSR⁺15]. The results of this analysis are

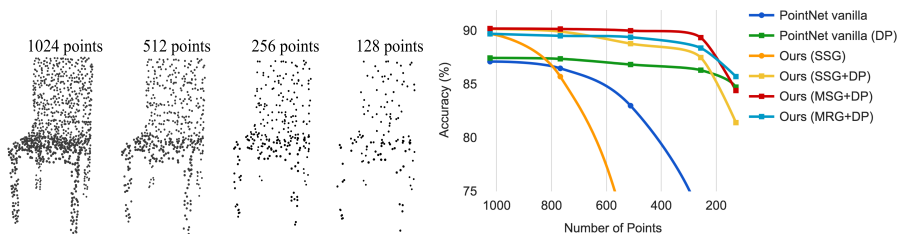


Figure 3.2.: Result of the point dropout test on PointNet++. [QYSG17]

3. Related Work

Table 3.1.: Pointnet++ non-rigid shape classification performance on SHREC15 benchmark set. [QYSG17]

	Metric space	Input feature	Accuracy (%)
DeepGM [LBH18]	-	Intrinsic features	93.03
PointNet++ [QYSG17]	Euclidean	XYZ	60.18
	Euclidean	Intrinsic features	94.49
	Non-Euclidean	Intrinsic features	96.09

shown in Table 3.1. DeepGM [LBH18] was used as a baseline. However, the non-rigid objects in the dataset are mainly humans and animals. They do not change their shape as radically as clothes when moved. Therefore, the test is an indication of the performance, but the challenge posed in this work is different from the SHREC15 benchmark.

In “Frustum PointNets for 3D Object Detection from RGB-D Data” [QLW⁺18], Qi et al. make use of RGB-D data, by using image data to select a segment of the point cloud which is then classified. The RGB-D input is split into an RGB image and the depth information. Based on the depth information, a point cloud is generated. Thus, an input of a high-resolution RGB image can be used in combination with a point cloud of lower resolution as this is a very common camera configuration. However, the system requires a camera projection matrix to be able to map pixels from the image to points in the point cloud. It is composed of three components: a frustum proposal generator, a 3D instance segmentation, and an amodal 3D box estimation. The frustum proposal generator selects a relevant frustum in the input point cloud. A convolutional neural network (CNN) detects an object in a region of the RGB image. Based on that region, the camera transformation matrix, and the depth information, a frustum in the input point cloud is defined. Then, the points inside the frustum are fed into the 3D instance segmentation component together with the class of the object detected by the CNN encoded as a one-hot vector. In this unit, the points in the frustum which do not belong to the object are removed using a PointNet++-like network. Afterward, the filtered points are fed into another PointNet++-like architecture to estimate box parameters for the 3D amodal bounding box enclosing the whole object and not only the directly visible parts of it.

Many approaches which improved upon PointNet and PointNet++ mainly focus on the selection of the points considered for further classification and the detection of bounding boxes in point clouds. These advantages are not utilizable with the task at hand, as the clothes are the only object in the point cloud (for details see Section 4.1.1) and their position is not relevant. Therefore, these advanced architectures are not used for the experiments in this work.

Qi et al. extend their PointNet++ in “Deep Hough Voting for 3D Object Detection in Point Clouds” [QLHG19] to output 3D bounding boxes. They demonstrated a simpler approach for the raw processing on point clouds with a PointNet-like architecture which is at least partially superior to conventional methods on benchmark datasets.

3.2. Shape based Object Classification

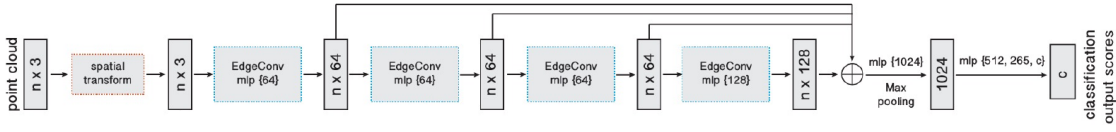


Figure 3.3.: Classification pipeline of DGCNN. The architecture makes extensive use of edge convolution layers. While this representation uses three-dimensional points, other configurations are possible. [WSL⁺19]

Similar to PointNet++, the dynamic graph CNN (DGCNN) developed by Wang et al. improves upon PointNet to capture local features [WSL⁺19]. Instead of applying MLPs with shared weights on each point individually, a new operation called “Edge Convolution” (EdgeConv) is used. Figure 3.3 shows its classification pipeline. The features extracted in all layers of the EdgeConv layers are concatenated at the end. Then, a max-pooling operation is applied and the features are processed further by MLPs.

In an EdgeConv layer with an input of n points ($X = x_1, \dots, x_n \subseteq \mathbb{R}^F$) with F being the dimensionality of the points), a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is computed to represent local point cloud structures. \mathcal{V} are the vertices $1, \dots, n$ and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ the edges of the graph. For example, the graph could be constructed as a k -nearest neighbor graph with self-loop meaning that each node also points on itself. The features of an edge are defined as $e_{i,j} = h_{\Theta}(x_i, x_j)$. $h : \mathbb{R}^F \times \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$ is a nonlinear function and Θ are its learnable parameters. A symmetric aggregation operation (\square) such as sum, max or mean is channel-wise applied on the edge features of all outgoing edges of a vertex. Thus, the output of the EdgeConv layer for a specific vertex is given by

$$x'_i = \square_{j:(i,j) \in \mathcal{E}} h_{\Theta}(x_i, x_j).$$

Therefore, an input of n F -dimensional points yields a point cloud with n F' -dimensional points. The authors discuss multiple possible configurations for h and \square . They implemented \square as the max function. The h -function is implemented by MLPs with shared weights. Both, the MLP implementation (left) as well as the calculation of edge features (right) are visualized in Figure 3.4.

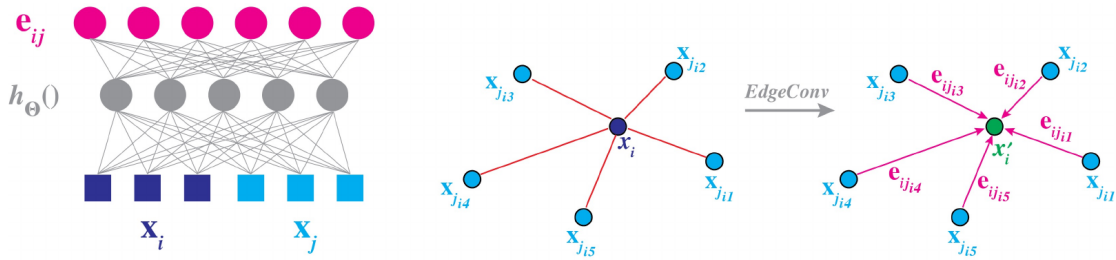


Figure 3.4.: Schematic representation of the edge convolution layers. In this example points are three dimensional. **Left:** MLP implementation of edge-feature extraction between x_i and x_j . **Right:** Visualization of edge feature computation. [WSL⁺19]

3. Related Work

As the graph is constructed new in each EdgeConv layer based on its input, it can be considered a dynamic graph. Contrary to a graph that is created at the beginning of the network and used throughout the processing pipeline, the neighbor relations in a dynamic graph change depending on the features extracted in later stages. This allows a single point at the end of a network a receptive field that is sparse and includes all points in the original point cloud.

After a spatial transform layer, similar to PointNet (see Section 2.4.2), points are fed into a series of EdgeConv layers. After four such stages, the features extracted at each stage are accumulated using a max pooling and MLPs. The output layer is sized depending on the number of classes to distinguish between. Similar to PointNet, the method also supports point cloud segmentation tasks. On ModelNet40, a higher classification accuracy of 92.9% was achieved with the trade-off of an increased processing time compared to PointNet. At the moment, the DGCNN is favored less than PointNet measured in the number of citations on the paper and stars on GitHub [16]. Considering this and the increased runtime which is crucial for a live classifier, this work will focus on the evaluation of PointNet.

Besides object classification, PointNet is also used in other contexts. In the work of Gao et al. [GLZF18], the usability of a modified version of the vanilla PointNet to regress an object's orientation from point cloud segments is investigated. The input of the PointNet consists of 256 points. Each point is defined by three dimensions of spatial coordinates and three dimensions for color information. Points are randomly selected from a point cloud segment of the object, the orientation of which is to be estimated. The spatial coordinates of the input points are normalized based on the estimated position of the object. The network is trained to provide a three-dimensional output which is interpreted as an axis-angle orientation. They built upon that work in 2020 [GLW⁺20] and regress both orientation and translation given a point cloud segment and a one-hot vector indicating the object class. In this case, points have $3 + k$ dimensions with k being the number of supported classes, as each point is defined as its coordinates in 3D space and the one-hot vector defining its class. The input points are fed into two "base nets" the architecture of which is very similar to a vanilla PointNet. One base net is tasked with estimating the orientation of the object, while the other estimates the translation. In contrast to their previous work, the input point coordinates of the orientation estimation base net are not normalized. The input point coordinates of the position estimation base net are normalized with the mean of all points. To still achieve an accurate position estimation, the mean is added to the results of the network. Thus, the output of the architecture is a pose estimation of the object in form of the coordinates in 3D space and the orientation in axis-angle representation given the input points and their class. Their most recent work [GLH⁺21] accomplishes a similar result following a different route. The architecture is based on an augmented autoencoder. A structure employing EdgeConv layers is used as an encoder with an average-pooling layer as its latest stage. The code in form of 1024 features is decoded with multiple MLP layers. The autoencoder is trained to generate a noise and occlusion free point cloud from a noisy input point cloud with occlusions and the class label. Based on the code generated by the encoder, the translation, as well as the rotation of the object, is estimated. Thus, additionally to the translation and rotation of the object, this approach removes noise and occlusions from the input point cloud.

3.3. Clothes Classification and Manipulation

As this work evaluates the usability of PointNet for clothes classification to allow clothes manipulation by robots, approaches to clothes classification and manipulation are presented in this section. However, as Fashion-MNIST (see Section 3.1) is generally designed to evaluate vision systems without a focus on clothes classification, approaches which do not especially consider clothes besides using that dataset such as the work of Bhatnagar et al. [BGK17], Duan et al. [DYZL19], Meshkini et al. [MPG20], or Kayed et al. [KAM20] are not regarded in detail.

Because clothes do not have a distinct shape to be identified by, methods leveraging their wrinkle characteristics are a logical consequence. In 2009, Yamazaki and Inaba presented a method to detect wrinkled objects in images and use it to direct daily assistive robots towards clothes [YI09]. Besides others, they use features extracted by Gabor filters as input of a Support Vector Machine (SVM). Their system was integrated into the software stack of an assistive robot and enables it to find and collect clothes dispersed in a room to put them in a washing machine. The authors follow up on this work with “Clothing Classification Using Image Features Derived from Clothing Fabrics, Wrinkles and Cloth Overlaps” in 2013 [YI13]. They evaluate how the usage of diverse features affects the cloth classification accuracy of a multi-class SVM. By combining features extracted about clothing fabric and wrinkle density, the existence of cloth-overlaps, and scale-space extrema, they achieved an accuracy of 99.07% using 10-fold cross-validation.

Maitin-Shepard et al. use a Willow Garage PR2 robot to fold towels [MSCTLA10]. The borders of the cloth are detected and then, corners are fitted into the detections. The detected corners of the towels are used as grasp points for the robot. When the robot has grasped the towel in a predefined manner at its corners, it folds them on a table. In 28 out of 50 trial runs, the robot succeeded in its task without complications. The robot could recover from all errors which occurred in the remaining 22 runs. Videos in the additional material of the paper [17] show the process of the robot folding towels. It takes roughly 20 minutes for each cloth.

Bersch et al. also present a method to fold cloth with the PR2 robot platform [BPK11]. However, as they fold a t-shirt, the complexity of the task is much higher. Solely relying on corners is not applicable in this scenario. The authors approach this task with fiducial markers printed all over the t-shirt. While this significantly eases the perception of the current state, they focus on the internal representation of the state and selecting useful manipulation actions.

In his work, Pablo Jiménez Schlegl gave an overview on the field of visual grasp point localization, classification, and state recognition in robotic manipulation of cloth in 2017 [JS17].

4. Approach

In this thesis, the task of developing a live classifier for clothes based on point cloud data is approached. To achieve a system capable of the task, a dataset of depth and RGB image recordings of various types of clothes was created and a processing pipeline, which classifies clothes based on point clouds was set up. The overall approach is schematically outlined in Figure 4.1. The presentation of this work is split into two parts. First, the dataset generation is explained in Section 4.1 including the tools developed and its object types. Second, the implementation, training, and testing of the clothes classifier is detailed in Section 4.2.

4.1. Dataset Creation

To be able to classify clothes held at a single point based on depth information, a dataset encompassing measurements of such samples is needed. As shown in Section 3.1, there is currently no dataset available that fits the requirements of this work. The classification task posed in this work is very hard, as it requires the network to tolerate the changing shapes of the non-rigid clothes and to focus on the small features which distinct classes from each other. Deep learning methods focus on features extracted from the input data. Often these features are not intuitively understandable for humans. When approaching such a task, great attention has to be paid to preventing overfitting of the deep learning approach.

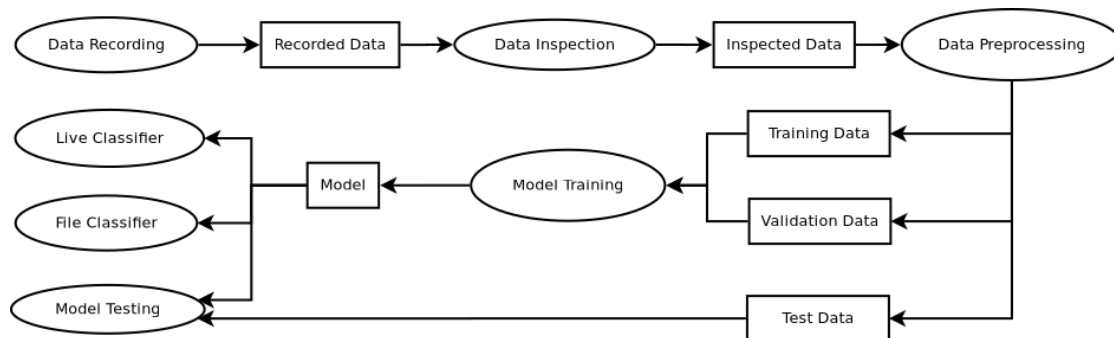


Figure 4.1.: Schematic representation of the approach of this work. First, data is recorded. Second, it is inspected and filtered if needed. Afterward, the recorded data is preprocessed. The preprocessed data is used for model training and testing. A model is trained on the training data. Based on the validation data, the end of training is determined. Later on, the model can be tested on test data and used in the live- or file classifier.

4. Approach

As mentioned when discussing simulation in Section 2.1, it is possible to generate many types of data in a simulator. Doing this would allow a very efficient generation of data in various scenarios. However, in this work, data generation in simulation is deliberately avoided. The main reason for this is fear of overfitting on simulation artifacts. When modeling clothes for simulation, they are commonly represented by a loosely connected mesh. The way the clothes behave is therefore highly dependent on how they are modeled. This includes characteristics such as resolution, stiffness, and shape of the segments of the mesh. While the modeling is sufficiently close to real clothes behavior such as, for example, simple manipulation tasks in reinforcement learning, this does not warrant an authentic replication of the behavior of real clothes in every detail. Thus, it is likely that the classifier focuses on differences in the modeling of the clothes which are not observable in real applications. This would distort the results of the work significantly. Consequently, the data was collected manually by holding clothes in front of the Kinect v2 camera presented in Section 2.5 to eliminate this overfitting possibility. As one of the main goals of this work is to achieve a well generalizing classifier, this generalization capability is tested with the relatively unconventional constraint of using only a single piece of clothes to represent each class in the training data. Usually, various examples of the same object type are utilized to achieve a good generalization. In this case, however, due to the type of information regarded and measures were taken against overfitting at every step of the processing pipeline, the system is designed to be able to extract the defining features of the classes in learning from a single piece. In the following sections, the creation of the dataset is detailed.

4.1.1. Technical Aspects

As the creation of such a domain-specific and multimodal dataset and the management and inspection of its samples also requires specialized tools, these were developed as part of this work. The point cloud data needs to be recorded in a data format that is both widely usable and efficiently readable.

The data processing pipeline created in this work consists of a data recording component, which produces point cloud data files and corresponding images, a data prepare script, which preprocesses the data for the neural network and splits the data in a train and a validation set. Additionally, a data inspection tool was created which can visualize both the raw as well as the prepared data.

Data Format

Point clouds used in this work are processed and saved by two libraries: NumPy [HMvdW⁺20] and Open3D [ZPK18]. To accommodate this, they are stored in two separate file formats. NumPy offers to save arrays in a compressed manner as .npz-files. In that case, point clouds consisting of n points are processed in the form of $n \times 3$ matrices stored as two-dimensional NumPy arrays. Each point of the point cloud is a row in the matrix with x , y , and z defined in the columns. When point normals are available, they are included in the form of three additional columns in the matrix. The NumPy format was used as it contains less overhead information and is easier to handle, because NumPy is the only dependency required to process

them. The input of PointNet consists of a matrix as defined in the NumPy format which is easily converted to a PyTorch Tensor. Therefore, the arrays can be directly fed into the neural network. Additionally, the format allows to save sequences of point clouds in a single file by adding another dimension to the matrix. This might be useful for future work exploring the usage of temporal features.

Open3D relies on the point cloud data format (.pcd-files) which is also used by the Point Cloud Library (PCL) [RC11]. Point cloud data files can be directly imported as Open3D PointCloud objects. These objects can be used to reconstruct the object surface and calculate the point normals. Also, they are natively supported by the Open3D visualization which is used for data inspection. The format was chosen for the recorded point clouds, as it is widely used for various point cloud applications.

Data Recording

While approaches of simulating ToF camera readings in artificial environments exist [GKUP11], it was decided to take the measurements in the real world with real clothes to generate more meaningful results. As the neural network has to focus on small details to solve the tasks posed in the experiments (see Section 5), it was expected that the network might overfit on artifacts induced in the simulation of non-rigid objects (see Section 2.1 “Simulation”). To record the point cloud data, a tool was developed specifically for the ROS environment. The `point_cloud_recorder` is a script written in Python which offers a console based user interface for data collection. It subscribes to the point cloud and image topics of the `kinect_bridge` presented in Section 2.5. Via command line parameters, the directory of the created dataset is defined. Additional options include whether images are recorded along with the point clouds, whether single point clouds or sequences are recorded, and the length of sequences recorded. On startup, it asks the user to enter a class name. Recorded point clouds and image will be stored in a directory named after the class in the dataset directory. Then, the user can trigger a recording by pressing **Enter**. In normal mode, a single point cloud is recorded (with an image if the option is set). When the sequence mode is activated, multiple sequences are recorded in series. The frequency of recorded point clouds is defined by setting the `max_frames`-option of the `kinect_bridge`. The user can change the current class by entering `c` triggering the recorder to ask for the new class name. The names of the stored files are composed of the class name, a time stamp and their number in the sequence they are recorded in, if sequence mode is activated. Additional commands are available to switch between sequence and normal mode and visualize the last captured point cloud by entering `v`.

During tests, it was found that the `pc2.read_points` method induces a major delay. Its task is to unpack the `PointCloud2`-message (see Section 2.2.1) generating a NumPy array consisting of the point coordinates. This caused issues when recording sequences, as the recording frequency became unstable. The goal of the recording of sequences is to allow future work on the dataset to use the movement of clothes over time as a feature for classification. This would be complicated by inconsistent time intervals between recordings. Caching the sequence in the form of `PointCloud2`-messages in the RAM and converting them to NumPy arrays after a whole sequence is recorded solves the issue.

4. Approach



Figure 4.2.: Exemplary image of the data collection process and setup. The user holds the piece grasped at a single position in front of the Kinect camera. A distance of 0.6m to 1m between the camera and the piece is maintained with the help of markers on the floor.

The data collection setup shown in Figure 4.2 is composed of a Microsoft Kinect v2 camera on a tripod and markers on the floor. The camera is connected to a computer, which handles the data recording and gives the user direct feedback on the data recorded. The user can see the output of the data recording script, an overview of the files collected (including their sum), and a live view of both the RGB and depth images recorded by the Kinect camera. The data recording script informs the user whether it is currently recording data, processing a sequence, or ready for the next recording. To further improve the recording efficiency, the user may activate a “speak”-option as a command line parameter which causes the script to announce the states *recording*, *processing*, and *ready* via a text to speech engine. This is intended to prompt the user to record more data as effectively as possible. The file overview is necessary to keep track of how many samples are collected so the user can monitor their progress. In this work, the hand of the user is not included in the point cloud because the type of grasp for example could cause overfitting. This is achieved by holding the clothes with the hand always being held as low as possible but above the cameras FOV. Based on the live view of the camera images, the user gets immediate feedback about the holding position (see Figure 4.3). The holding position is further restricted by the distance to the camera. As mentioned in Section 2.5, the Kinect v2 camera only supports a minimal distance of 0.5 m. With a safety margin, a minimal distance to the camera of 0.6 m was chosen. To be able to separate the object from its background reliably (see Section 4.1.1), a maximal distance of 1m was decided upon.

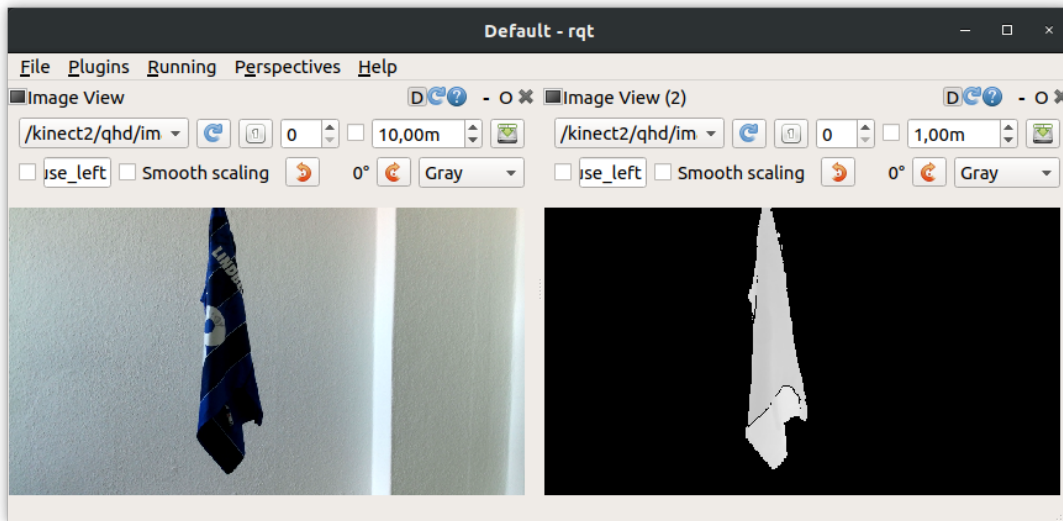


Figure 4.3.: Screenshot of the RQT recording setup. Using the live view of the RGB image and depth image, the user can assess whether the holding position fits the requirements. The depth image was used as a representation instead of a point cloud because it provides a better reflection of the camera's FOV.

Data Inspection

Data inspection is vital for the management of a dataset. A user can check whether errors in the data collection occurred and qualitatively assess the quality of the recordings. Visualized samples help in explaining the challenge posed to the neural network classifier by the task. It

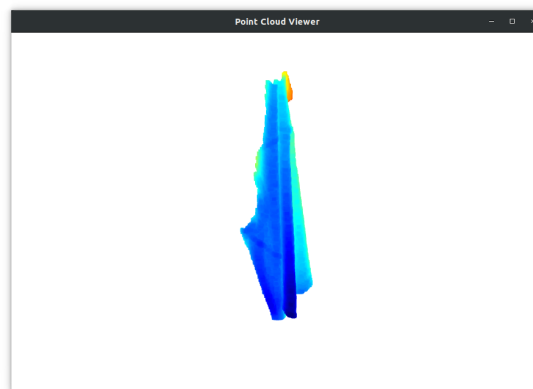


Figure 4.4.: Screenshot of the point cloud viewer showing a sample recorded of a jersey. The color correlates to the distance to the viewer. Red indicates points in the back and blue signals a point close to the camera.

4. Approach

Table 4.1.: Excerpt of the key commands of the point cloud data visualization script. When not noted otherwise, the key commands are natively supported by the Open3D viewer.

Key	Function
S	View alphabetically last file in directory (added in this work)
D	View alphabetically next file in directory (added in this work)
N	Change number of visualized points (added in this work)
H	Print help information
Q, Esc	Exit window
Alt + Enter	Toggle fullscreen view
L	Toggle lightning in renderer
N	Toggle point normal rendering
R	Reset view point

also helps new users to get an overview of the data collected in the set. Thus, a tool similar to an image viewer is required. It should be capable of displaying the point cloud files created in this work and feature a customizable viewpoint as well as convenient navigation between files. As mentioned, the `point_cloud_recorder` provides the option to visualize the last point cloud recorded. But as in that case, only the point cloud recorded most recently is shown, an additional visualization program was written to allow users to inspect a specific point cloud from a dataset. A screenshot of the tool is shown in Figure 4.4. Compressed NumPy arrays (.npz files), as well as point cloud data (.pcd) files, are supported. The Open3D library [ZPK18] is used to visualize the point clouds. While Open3D natively offers a method to load .pcd files, the .npz files are loaded using NumPy and the values are loaded into an Open3D point cloud object later on. When available, point normals are also visualized. The mouse can be used to move, rotate and zoom the viewpoint. Table 4.1 lists an excerpt of the key commands available in the visualization script. Most notably, the tool offers commands to print help (all available key commands explained), toggle point normal rendering, activate color coding, and exit the tool. In this work, the option to iterate through the supported files in the directory was added to allow an easier overview of multiple files. While iterating through files, the viewpoint is kept. Thus, multiple point clouds can be compared without changing the perspective. Also, the number of points visualized can be changed. This can also be done using a command-line parameter. When the number of points available is below the limit, points are selected by random sampling.

Data Preprocessing

It would not be optimal to use the recorded data directly for training as it might contain too many points some of which do not belong to the object to be classified, does not include estimated point normals, and is in the wrong format. A data preprocessing script was written to prepare the datasets for training and testing of PointNet by addressing these issues. Additionally, it is tasked with separating the datasets into a train and a test set. Generating

separate train and test sets allows a fair comparison of multiple approaches on the same challenge with the same train and test samples. Also, other performance aspects can be assessed later on using the dataset without the need to redo the whole experiment including training. The ratio of training to test data can be defined as required using a command-line parameter with the default being a test fraction of 0.2. In this work, the test set generated in this way was used as a validation set while the test set was generated based on a dataset recorded completely independently and only for testing. Thus, especially with the sequence-wise data collection, the validation set is very similar to the training set. To use all data in a set as test data, another command line parameter was introduced. When it is set, all data in the input set is processed as test data. This option is used to generate the test data for the experiments.

Not all points in the samples necessarily belong to the piece to be classified. Other objects in the background (for example a wall) might be included in the sample. To approach this issue, the fact that the distance between camera and object is known can be leveraged. By defining a maximal distance from the camera, the recorded points can be filtered. Thereby, measures of objects in the background are reliably removed if their distance to the sensor is greater than the threshold. Also, it was observed that the Kinect camera produced some erroneous point measurements at the edge of the FOV. However, these points were always more than ten meters away from the camera. Thus, also these erroneous measurements are removed from the samples. To give the user control over the threshold, it is also definable as a command-line parameter.

Another essential function of the prepare script is the estimation of point normals. The estimation of point normals explained in detail in Section 2.2.2 is computationally expensive but necessary for a wide range of experiments. It cannot be integrated into the data recording or in the train- and testing process because it would cause a significant delay while computing. Therefore it is performed in the preprocessing step because it is usually only performed once for a dataset and does not have time-sensitive requirements. To optimize the efficiency during point normal estimation, the CPU usage of both the normal estimation and alignment processes provided by Open3D [ZPK18] were analyzed and it was found that they only run on a single processing core. In an effort to still make use of all processing cores available, the Python library joblib [18], which simplifies the usability of the multiprocessing library was used to distribute multiple point normal estimation and alignment operations across all processing cores.

After the estimation of point normals, the point cloud size is reduced by randomly sampling a subset of the points. The number of sampled points is set by a command-line parameter with a default of 1024. When performing the sampling after the point normal estimation, the point normals can contain meaningful information about the shape described by the point cloud which would be lost when estimating the normals on sampled data.

As explained in Section 4.1.1 “Data Format”, the preprocessed point clouds are stored as compressed NumPy arrays (.npz-files) to increase the time and space efficiency during training and testing. When unpacking the NumPy array, the data is already available in a format that can be fed into the neural network as no unpacking of the points and possibly point normals from the point cloud data structure is necessary.

4. Approach

4.1.2. Object Classes

The dataset created features 8 object classes, which were used in various configurations (see Section 4.1.3). Depending on the experiments, the clothes were chosen because of their shape, their material, or the fact that they are very similar to other clothes evaluated. The object classes were Jeans, T-Shirt, Sweatpants, Sweater, Underpants, Sock, Shortpants, and Jersey.

In Table 4.2, exemplary images and point clouds of each object class are shown to give an overview of the general classification challenge posed by the set used in the evaluation (see Section 5). Note, that the point clouds shown in the table are not the full point clouds of the recorded clothes but the more sparse sampled point clouds preprocessed for PointNet. The point of view of the point cloud renderings is the same they are recorded in. Lines in the visualized point clouds represent the point normals estimated in the data preprocessing. The classes were selected to represent the subset of most commonly worn clothes in a domestic environment. The Jersey was added for experiments evaluating the classification performance of similar shapes (when compared to the T-Shirt). Samples used during recording are used clothes that were washed but not ironed since they were last worn.

4.1.3. Recorded Sets

A dataset of point clouds and RGB images was created for the experiments presented in Section 5. Based on this large dataset, various small datasets were created as required in a specific experiment. In the overall dataset, for all clothes except Underpants and Sock presented in Table 4.2, samples are collected in each of the three grasp areas shown in Figure 4.5. The T-Shirt in the Figure is only an example. the pattern was applied to the other clothes classes as well. The top part is defined for the T-Shirt, Sweater, and Jersey as the area between and above the shoulders. For all kinds of trousers, it is the waistband. The edge is everything but the top which is at the border of the piece when viewed directly from the front or back. The Inside collection encompasses all other grasp positions. Openings in the pieces such as the collar or waistband were held shut in the Top and Edge collections but left open in the Inside collection. Later on, these collections can be combined as necessary.

A training and a test dataset were recorded independently from each other. Most importantly, different clothes of the same types were used. When possible, a different color, cut, and print were chosen. Additionally, the sets differ in the background, the pitch of the camera angle was changed slightly and a different person held and rotated the clothes to avoid a similar grasp or holding position. Thus, test results reflect the performance of the classifier on unknown clothes given they can be assigned to one of the classes learned by the classifier.

The composition and sample counts for each class in the training and test data are listed in Table 4.3. Note that all sample counts in the table (except total) are per-class values. In this work, the preprocessing step splits the training data in a train and a validation portion (see Section 4.1.1).

4.1. Dataset Creation

Table 4.2.: Overview of the object classes used in the dataset. For each class, an image depicting the piece as well as an exemplary RGB image and point cloud are shown. The point clouds are depicted after performing a random sampling of 1024 points. Additionally, estimated point normals are visualized.



4. Approach

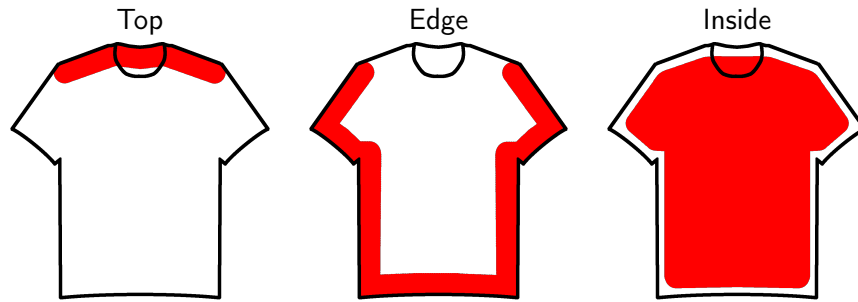


Figure 4.5.: Exemplary representation of grasp areas used for data recording. During data collection, clothes were grasped at a single point inside the areas marked. The grasp areas are applied to the other types of clothes in a similar manner.

Table 4.3.: Composition and size of the dataset developed in this work. For the training and test set, the number of samples per class is given. The total numbers sum up all classes and all grasping positions.

Class	Training Set		Test Set	
Jeans				
T-Shirt	Top	1500	Top	300
Sweatpants	Edge	1500	Edge	300
Sweater	Inside	1500	Inside	300
Shortpants				
Jersey				
Underpants		1500		300
Sock				
Total		30000		6000

4.2. Classifier

The classifier used in this work is the PointNet in the classification configuration. The number of classes is adapted to the specific dataset and experiment. As mentioned in Section 2.4, a PyTorch based PointNet implementation is available [10]. This implementation was used as the groundwork of the framework used in this work. While the architecture of the original PointNet remains original, a new data loader for network training and testing was developed and the training and testing scripts were extended.

4.2.1. Data Loader

A data loader is an essential part of the PyTorch test- and training pipeline [PGM⁺19]. It acts as an interface that makes the prepared datasets accessible to the DataLoader class of PyTorch. The new data loader called `KinectDataLoader` reads a text file generated by the data prepare script. The file contains multiple lines of file names and class IDs. As an iterable

object, it offers the option to load a specific compressed NumPy array from the file together with its class ID. To allow using the same dataset for multiple different experiments, the constructor of the `KinectDataLoader` offers to limit the number of points loaded from the files and to select whether point normals are loaded if available. To prevent overfitting during training, additional parameters are offered to move the coordinate system of the object into the middle of the point cloud and to apply a random scaling factor in the range of $\pm 10\%$.

4.2.2. Training

The training script already available in the PointNet implementation was extended for the needs of this work. It features a train and a test component. The test component is designed to test the trained model on a validation dataset. After the first epoch, the model is saved and tested using the validation set. The classification accuracy is saved. When a later model trained in further epochs achieves a better result, the stored model and classification accuracy are overwritten. Thus, the model performing best on validation data is selected after the predefined amount of training epochs was carried out. In this work, models were trained for 50 epochs, after tests with 100 epochs did not yield improved results. Additional to the point cloud centering in the data loader, a random point dropout, scale, and shift are applied on each sample to prevent overfitting. Also, the random point dropout is supposed to improve training in general and make the resulting model more robust against point dropout when classifying real-world samples. Similar effects were shown in the robustness test by Qi et al. [QYSG17] shown in Figure 3.2.

For training, the Adam optimizer [KB15] was used in a default configuration with a learning rate of 0.001, $\beta_1 = 0.9$, and $\beta_2 = 0.999$, and a smaller than usual $\epsilon = 10^{-8}$. Usually, a batch size of 24 was used. Via command-line options, various additional settings such as CPU mode, a specific GPU to be used, the number of points, and whether or not to use point normals can be set.

4.2.3. Testing

The testing script is in many aspects similar to the training script and its testing component. During training, point clouds are still centered around the zero-point of the coordinate system because this step is mandatory for successful classification as the models are trained with it. The other measures against overfitting, however, are dropped as they do not affect learning or the evaluation results in a meaningful manner. The main difference is that the original testing script was extended in this work to produce detailed classification metrics. During the evaluation, the result and the label for each sample are saved. Afterward, several class-specific and overall metrics, as well as confusion matrices, are calculated using scikit-learn [PVG⁺11] functions. The metrics used are defined and explained in greater detail in Section 5.1. Results of the metrics are printed into the console and saved to a .txt-file. Optionally, a .tex file with tables presenting the results can be rendered. To generate these files, the class names have to be provided as a command-line parameter of the script. This work makes heavy use of the L^AT_EX-file generation capability as nearly all tables for the classification performance experiments (see Section 5.6) and in the appendix (see Section B) were generated this way.

4. Approach

4.2.4. Live Classifier

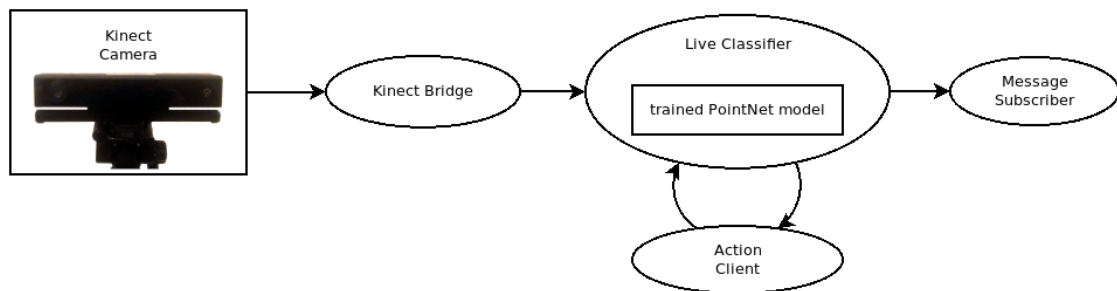


Figure 4.6.: Schematic presentation of the live classifier integrated in the ROS environment. The Kinect bridge publishes images and point clouds taken by the Kinect camera into the ROS environment. They are processed in the live classifier using the trained model. Classification results are made available via a message to all subscribers or as response to an action request from a client.

To integrate the clothes classifier into the ROS environment, the live classifier node was written. An exemplary classification pipeline using the live classifier node is sketched in Figure 4.6. As outlined in Section 2.5, the Kinect Bridge [11] is used as an interface between the Kinect v2 camera and the ROS environment. Thus, all measurements of the camera can be inspected using common ROS tools such as RQT or RViz. The live classifier node subscribes to a topic, on which a `PointCloud2`-message is published. It can be started using a launch file, which loads the configuration file (see Listing D.3) and optionally also launches the Kinect Bridge. The node acts as a wrapper around a trained PyTorch classifier model. On startup, a model is loaded similar to the testing script (see Section 4.2.3). During preprocessing, the points are filtered by their depth (distance to the camera), a predefined number of points is sampled randomly and finally, the points are transformed such that the coordinate system is in the center of the point cloud. As the point clouds are just captured by the camera, no point normals are already available. They are not estimated in the live classifier, because it would introduce a significant delay in the processing pipeline. The classifier can be run in a *live mode* or a *request mode*. In live mode, the node enters a loop of classifying the most recent `PointCloud2` message and publishing the result as `ClassificationResult` message (see Listing 4.1). When processing, the most recent incoming message is queued. Afterward, the loop is reiterated. The `ClassificationResult` message is composed of a header, lists specifying the class names, IDs, and ratings as well as the name, ID, and rating of the class with the highest rating. Whereas including the information for the highest-rated class twice is redundant, it improves the readability of the raw data.

```

1  # Message describing a classification result
2  std_msgs/Header header
3
4  string[] class_names # names of all classes
5  uint8[] class_ids # IDs of all classes
6  float32[] class_ratings # ratings of all classes
7  string max_class_name # name of the class with the maximal rating
8  uint8 max_class_id # ID of the class with the maximal rating
9  float32 max_class_rating # maximal rating of any class

```

Listing 4.1: The definition of the ClassificationResult message.

A visualization node was written to give a better impression of the classification results to the user. It subscribes to a configurable topic on which messages of the type `ClassificationResult` are published. Then, a visualization that uses a bar chart to display the ratings of each class is rendered and displayed. When a new message is received, the visualization is updated. The visualization is implemented in Python using the `matplotlib` [Hun07] library. This provides future users with various well-documented customization options. These are convenient when using screen captures of the result visualization for demonstration purposes. However, due to the design of the `ClassificationResult` message, it is not necessary to adapt the visualization node for a certain domain. The number of classes and their names are inferred from the message. The result of the live classifier to an exemplary sample is shown in Figure 4.7. An RGB image and a depth image are shown to give an understanding

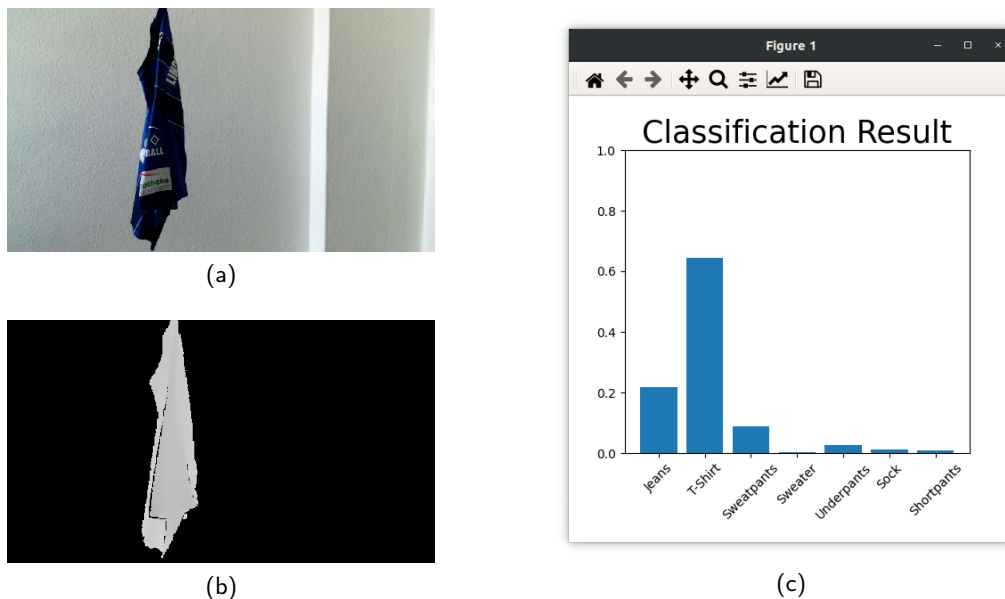


Figure 4.7.: Screenshot of the result visualization tool given an exemplary sample of a jersey. The sample is shown both as an RGB image (a) and depth image (b). In (c), the corresponding classification result is visualized in the tool. Similar to all other classifications, the RGB image data was not used for the classification.

4. Approach

of the sample. Still, a point cloud without color information is used as input of the classifier. The classification result is displayed by the live visualization tool.

While the live mode is intended for demonstrations to give users an impression of the system's performance especially in combination with the visualization node, the request mode is supposed to be used in an integrated robot system. In request mode, the live classifier offers an action interface with which a classification can be requested. On receiving a request, the node waits for the next `PointCloud2` message. When the message is received, the preprocessing is applied and the data is fed into the neural network. Finally, the classification result is returned. The node also returns information about the process in between steps. The definition of the action is listed in Listing 4.2. Its request is empty because the classification parameters are defined in the configuration file of the live classifier. On returning the classification result, a boolean is added to signal a successful classification. When an error occurs in the process, for example when no point cloud was received, the success field is set to false and the result is left undefined. To provide users with information about the progress and ease the analysis of possible errors, the node returns feedback on the current state by publishing whether it is currently waiting for the data or processing it in the neural network.

```
1  # Action triggering a point cloud recording and classification
2
3  # Empty request
4  ---
5  # Result
6  bool success
7  ClassificationResult result
8  ---
9  # Feedback
10 uint8 NONE = 0
11 uint8 WAIT_FOR_DATA = 1
12 uint8 PROCESS_DATA = 2
13 uint8 state
```

Listing 4.2: The definition of the Classify action.

In a robotic scenario, the robot could grasp the clothes, position them in front of the camera and then, request a classification. When the classification ended successfully, its result can be used for further processing of the clothes. The configuration file of the live classifier is shown in the appendix (see Listing D.3).

4.2.5. File Classifier

The live classifier approaches the task posed in this work directly. However, it is not used to analyze the inner workings of the learned model. In live classifier applications, the most recent message is processed continuously. This aggravates a detailed analysis of a classification result and comparisons between specific samples. While it is not directly relevant for the live classification task, such functionality is for example required to analyze the processing results of the classifier for specifically selected samples. Thus, it is vital when analyzing and improving the model used in the live classifier.

The file classifier extends the point cloud visualization tool used for data inspection (see Section 4.1.1) with the capability of classifying the currently visualized point cloud. This was implemented as a separate script to avoid requirements such as PyTorch and a trained model to run the general visualization tool. Therefore, it also implements the features and key commands presented in Section 4.1.1 “Data Inspection”.

Similar to other functions, the added file classification capability is also accessible via a key command. When `c` is entered, preprocessing is applied on the point cloud and it is fed into the loaded model. Various parameters of the preprocessing and the model path can be defined as command line parameters. Notably, the number of points used for the classification is the same number as visualized points. After the classification, the points are colored in blue and red, with red signaling that the points were significant points during the classification process (for more information on significant points see Section 5.5).

Thus, this tool allows the user to inspect point clouds and the network’s reaction to specific situations. The focus and errors of the network can be analyzed using various examples. When using the file navigation features, the network’s reactions to different samples can be compared conveniently, especially because the view parameters (camera position, orientation, and zoom) are kept between files and during classification. Exemplary samples classified and visualized with highlighted significant points using the file classifier are shown in Figure 5.3 in the Experiment section.

5. Experiment

During this work, multiple experiments were conducted to evaluate the performance of PointNet on clothes. The results of the experiments mostly consist of a series of metrics representing the classification performance in a numerical manner. Section 5.1 explains and defines the metrics used in this work. Table 5.1 gives an overview of the experiments conducted in this work. First, the effect of the number of points on the classification performance was evaluated. Due to the structure of the PointNet architecture, the computation effort of a sample scales linearly with its size. With computation time being a premium in live classification tasks, a middle ground between accuracy and performance has to be determined. Second, the grasp point used for holding the clothes was evaluated to evaluate whether clothes grasped at a specific point can be classified significantly better. To analyze the impact of the material, two classes of clothes with a generally similar shape (T-Shirt and Jersey) were used to train a binary classifier. With the material, the way the clothes hang changes. The experiment is set up to determine whether the network can make sufficient use of these changes. An understanding of what the neural network is focusing on is gained in the following experiment. It investigates which points in a sample were significant in the classification process. Finally, the performance of the classifier is evaluated to assess the general capability of the system when applied to a task. As shape measurements in the form of point clouds were chosen to achieve a good generalization over the clothes types, this is also investigated in detail.

Table 5.1.: Overview of the experiments performed in this work.

Experiment	Classes	Evaluation Set
5.2 Number of Points	T-Shirt Jeans	Validation Set
5.3 Grasp Positions	Sweatpants Sweater	
5.4 Similar Shape	T-Shirt Jersey	Validation Set Test Set
5.5 Significant Points	T-Shirt Jeans Sweatpants	Validation Set
5.6 Classification Performance and Generalization	Sweater Underpants Sock Shortpants	Validation Set Test Set

5. Experiment

When not stated otherwise, the models trained for the experiments are PointNets with the input transform layer, but without the feature transform layer as explained in Section 4.2.2. By default, an input of 1024 points is used. For training, the Adam optimizer [KB15] was used with a learning rate of 0.001 and a batch size of 24 over 50 epochs. Over the 50 epochs, the model with the best performance on the validation data is selected. For testing, the script presented in Section 4.2.3 was used.

5.1. Metrics

In the following sections, multiple classification experiments are presented. The evaluation of the experiment results is mainly done by discussing classification metrics and confusion matrices. As mentioned in Section 4.2.3, the test script generates an evaluation of the trained model using the *scikit-learn* Python library [PVG⁺11]. Therefore, similar definitions of the metrics are used. Additionally to the overall accuracy, class specific metrics such as *precision*, *recall* and the *f1-score* are used. As only one class is considered at once, measures of binary classifiers such as the number of true positive (*TP*), false positive (*FP*), true negative (*TN*), and false negative (*FN*) detections of the class can be used. The overall accuracy of a test run is ascertained by dividing the number of correct classifications by the number of overall classified samples as shown in equation 5.1.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (5.1)$$

The precision is defined in equation 5.2.

$$Precision = \frac{TP}{TP + FP} \quad (5.2)$$

To calculate the precision, the number of true positive detections is divided by the overall number of positive detections. Thus, the precision is the measure of when a class is detected, it actually is correct. In contrast, the recall defined in equation 5.3 measures the proportion at which the class is detected when it is present in the sample as it divides the number of true positive detections of a class by the number of its occurrences.

$$Recall = \frac{TP}{TP + FN} \quad (5.3)$$

The f1-score is the harmonic mean of the precision and recall metrics. It is calculated as shown in equation 5.4.

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \quad (5.4)$$

In confusion matrices, the ground truth is represented by the row of the table and the matrices are normalized by the ground truth. Thus, a value in the table reflects the portion of all samples of the class denoted by the row, which were identified as the class denoted by

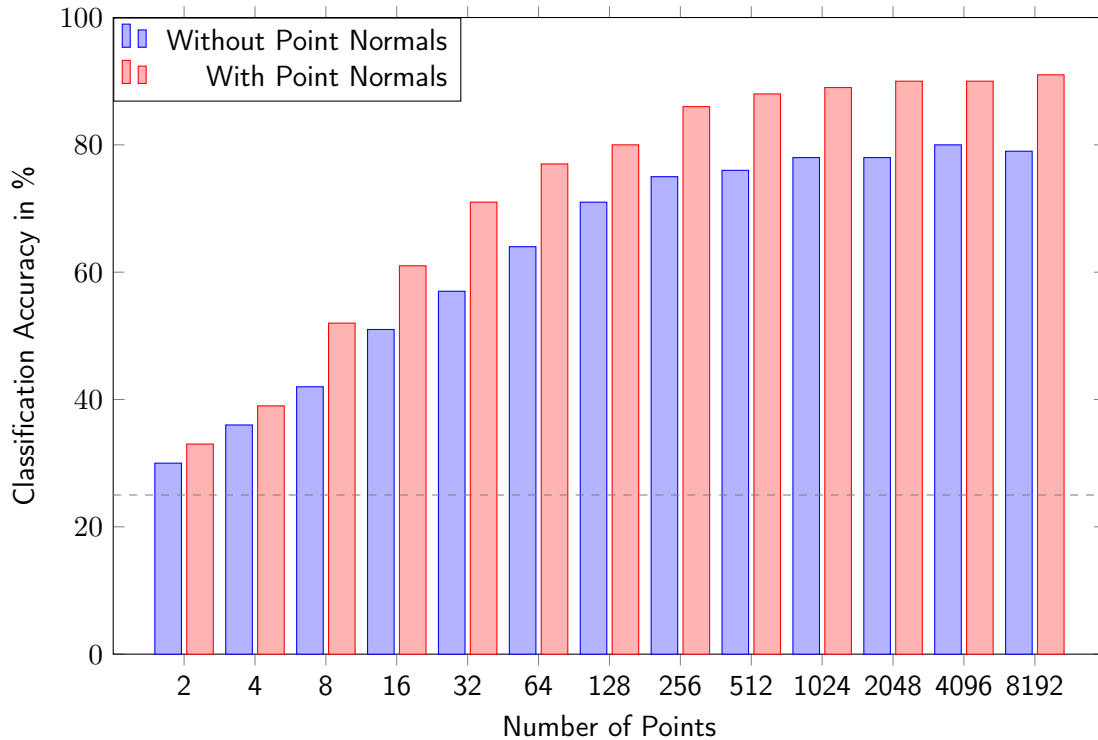


Figure 5.1.: Visualization of classification accuracy with various numbers of input points. The number of input points was used for both training and testing. The validation set, which is very similar to the training set, was used for evaluation. In the plot, results for both experiments, without (blue) and with (red) point normals are shown. A dashed, gray line marks the classification accuracy expected from a random classifier (25% on four classes).

the column. Note, that the order of the axis used by scikit-learn may be different than in other works. This is caused by a general confusion about the default. For example, at the time of writing, the German and the English article in the Wikipedia are not consistent in this regard [19, 20]. To increase the readability, the matrix cells are colored with a saturation reflecting their value.

All values are rounded to three digits behind the comma, as more precise values are not reproducible due to the random initialization of the neural networks and reduce the readability of the results.

5.2. Number of Points

Contrary to most common neural networks such as YOLOv4 [BWL20], where the size of the input image is fixed by the architecture, PointNet is capable of handling various sizes of point clouds. As both the sample classification runtime as well as training duration increase linearly

5. Experiment

with the number of points taken as input, an evaluation of the number of points required for a reliable classification performance needs to be conducted. While it is possible to train and test a PointNet with different numbers of input points, in this experiment, the same number of points was used for training as well as testing. For the experiment, a subset of the dataset presented in Section 4.1 which includes the classes T-Shirt, Jeans, Sweatpants, and Sweater was used. The evaluation set is the validation data which is very similar to the training data. Contrary to other evaluation sets, in the preprocessing stage, 8192 points were sampled from each original point cloud instead of the 1024 points, which are used in the other experiments. The availability of 8192 points in each sample allows the user to make use of the point sampling functionality of the train- and test scripts. Utilizing this option, 13 models trained with different amounts of points as sample size for both, training and evaluation were evaluated. Each was trained and tested with and without the usage of point normals.

Figure 5.1 depicts the results of the experiment runs performed. The experiment results are visualized for the configuration with and without the usage of point normals. A numerical listing of the classifier performance is available in the appendix (see Table B.29). In a dataset with four classes, a classification accuracy of 25% is equally precise as a random class assignment and means that the network did not learn successfully. In the experiment, an input size of two points already surpasses that mark. While the inclusion of point normals generally increases the accuracy, this effect becomes more pronounced with 8 or more input points.

5.3. Grasp Positions

In a robotic setup, it might not be possible to grasp clothes reliably at a specific position such as the collar or the waistband. Therefore, the effect of the grasp position on the classification performance of PointNet is investigated in this experiment. Three grasp areas were evaluated with a dataset for each area. In each sample, the clothes were held at a single point in a specified area. A dataset consisting of 1500 point clouds for each of four object classes grasped in each of the three grasp areas was used. The separate datasets (see Figure 4.5) were combined resulting in collections of point clouds recorded in the grasp positions presented in Figure 5.2. Grasp areas shown in Figure 5.2 are exemplary and carried over to the other clothes types, too. They are explained in detail in Section 4.1.1. The chosen object classes are T-Shirt, Jeans, Sweatpants, and Sweater. A train/validation split of 80:20 was used. The validation set which is relatively similar to the training data was used as evaluation data in this experiment. 1024 points per sample were used as input of PointNet. As before, each experiment run was conducted separately with and without point normals. Table 5.2 gives an overview of the accuracies measured in the experiments. Detailed experiment results are listed in the appendix in Section B.1.

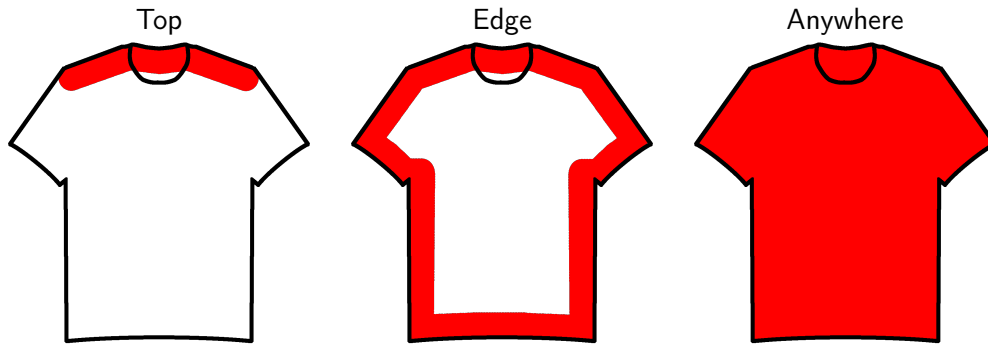


Figure 5.2.: Exemplary representation of grasp areas used for evaluation. The grasp areas are achieved by combining smaller sample collections recorded with disjoint areas (see Figure 4.5).

Table 5.2.: Comparison of classification accuracies with four classes and different grasp positions.

	Grasp Position		
	Top	Edge	Anywhere
Without Point Normals	91.4%	83.9%	81.8%
With Point Normals	95.8%	91.7%	91.3%

Table 5.3.: Comparison of classification accuracies with four classes and different grasp positions.

	Grasp Position		
	Top	Edge	Anywhere
Accuracy on validation data	91.4%	83.9%	81.8%
With Point Normals	95.8%	91.7%	91.3%

5. Experiment

	Grasp Position		
	Top	Edge	Everywhere
Accuracy on validation data	91.4%	83.9%	81.8%

5.4. Similar Shape

To investigate the significance of the clothes' material, a classifier was trained to differentiate between a t-shirt and a jersey. For comparison, another model which distinguishes between a t-shirt and a jeans was trained. All three classes (only two of which are used at once in both cases) classified in the experiment are shown in Table 5.4, an excerpt from Table 4.2. Again, the first row shows the pieces used in the experiment, and additionally, exemplary RGB images and point clouds with estimated point normals are shown. While the general shape of the classes T-Shirt and Jersey is very similar, they mainly differ in color and material. In the training data, the t-shirt consists of dark blue cotton which is not very reflective. In contrast, the jersey in the train and validation sets is made from synthetic material and colored red. Also, the synthetic material glares slightly. In the test dataset, different pieces in different colors of the same type were used. Because the color is not included in the input information of the network, it will not be regarded.

Similar to other experiments, results are collected with and without calculating point normals beforehand. Again, the experiment was conducted sequentially for the three grasping areas as the grasp positions experiment. As this is not the focus of this evaluation, only the results for all grasp positions are discussed.

Table 5.5 provides an overview of the results measured. The similar shape classifier performed relatively well on the validation data (78.8% accuracy without and 89.6% with point normals). On independent test data, however, it performs worse than a pure guess which would achieve an accuracy of about 50% when distinguishing two classes. This pronounced decline in classification accuracy indicates severe overfitting on the training and validation data. In contrast, for the classifier applied to diverse shapes, no such effect can be reported. More detailed results are listed in the appendix (see Section B.2).

Table 5.4.: Overview of the object classes used in the similar shape experiment. Excerpt from Table 4.2. The classes Jeans and T-Shirt are used as diverse shapes and T-Shirt and Jersey are used as similar shapes. For each class, an image depicting the piece as well as an exemplary RGB image and point cloud are shown. The point clouds are depicted after performing a random sampling of 1024 points. Additionally, estimated point normals are visualized.





5.5. Significant Points

When the max-pooling is applied in the PointNet, the maximal activation for each of the 1024 features is selected. While for some of the points multiple features gain a maximum activation, others have none. Because the PointNet classification pipeline does not take any information besides the resulting global feature vector into account, the points without any feature with a maximal activation are not regarded in further processing. Thus, features of a subset of the input points are processed further and the basis for the class rating. In this work, the subset of points regarded further is called *significant points*. In other work such as [Gao21], it is also referenced as *active points*. To understand what kind of features the network is focusing on, analyzing the significant points provides valuable insight.

The seven-class classification models which are evaluated in detail in Section 5.6 were used for the experiments. The input point clouds consist of 1024 points. Again, similar to the other experiments, it was performed with and without considering point normals. The difference is especially interesting as point normals add local information to the extracted features. As can be seen in the classification performance results in the other experiments, these additional

5. Experiment

Table 5.5.: Comparison of classification accuracies with two similar or diverse classes. The experiments were conducted without and with point normals in the input samples. Both, validation and independent test data were used as evaluation set.

	Similar Shape		Diverse Shape	
				
	Validation Data	Test Data	Validation Data	Test Data
Without Point Normals	78.8%	24.3%	97%	97.7%
With Point Normals	89.6%	30.1%	99.7%	96.1%

features are useful for the classification process. Thus, the possible difference in the selection of significant points needs to be examined.

Exemplary visualizations of input point clouds with significant points marked in **red** are shown in Figure 5.3. The experiments were performed with the T-Shirt, Jeans, and Sock classes. Examples selected are subjectively representational of the overall results. In the images, the perspective is oriented along the z-axis. This means that the viewing angle is similar to that of the camera. For the experiment without point normals, it is evident that most of the significant points are arranged along the edge of the clothes. Most of the points which are located closer to the center of the point cloud are close to the edge of the cloud on the z-axis. Also, consistently, a large portion of the points are at the bottom of the point cloud for all classes. In the case with point normals, the result is similar with the exception of a higher rate of points distributed seemingly randomly throughout the point cloud.

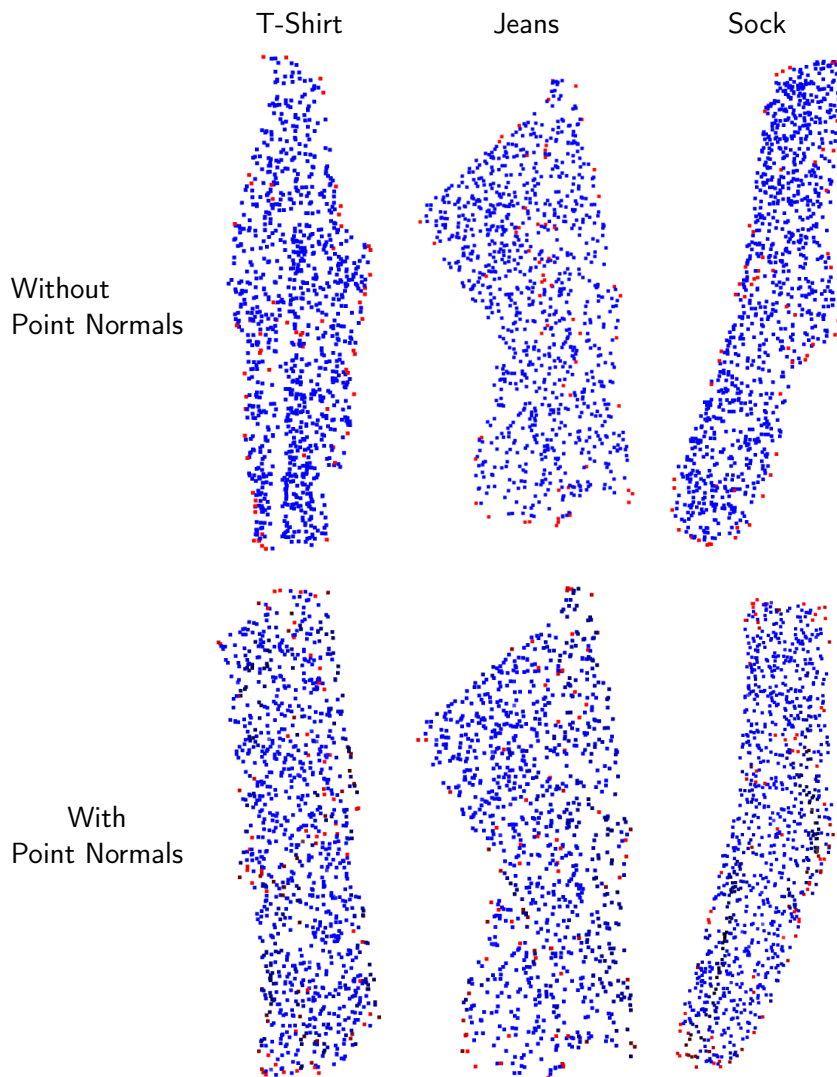


Figure 5.3.: Significant points (marked in red) in point clouds of various classes with and without point normals. The samples depicted are exemplary and reflect a qualitative review of the results. Some points in the examples with point normals are depicted in a darker tone as the visualization applies a shading on the points when point normals are available.

5.6. Classification Performance and Generalization

As the overall goal of this work is to design a system that can classify a live stream of point clouds, a PointNet model is required. This model was trained with and without using point normals and is evaluated in detail. Because in the design of the pipeline, an emphasis was put on preventing overfitting and achieving a generalizing classifier using relatively monotone

5. Experiment

training data, this experiment was conducted both with validation data and independently recorded test data used as evaluation set. Similar to all other experiments, this analysis is performed without and with point normals in the input point cloud. Thus, each of the two models was trained once but tested and evaluated twice.

Table 5.6 gives an overview of the experiments performed based on the classification accuracy achieved on the evaluation sets. Most notable is the difference in the impact of the point normal based classification between the two evaluation methods. While the model using point normal information achieves an accuracy of 92.2% on validation data, which is a 10.1 percentage points improvement over the model without point normals, with an accuracy of 72.5% on test data, it performs 1.9 percentage points worse than the model without point normals.

Table 5.6.: Comparison of classification accuracies of the seven-class classifier without and with point normals using validation and independent test data.

	Validation Data	Test Data
Without Point Normals	82.3%	74.4%
With Point Normals	92.2%	72.5%

The following tables present the detailed results of the four experiments. For each experiment, various metrics are listed on a class-specific scale and overall values are calculated. The weighted average is required because the support (number of samples in the evaluation set) varies between the classes. This is regarded by the weighted average value by weighting the individual scores based on the support of the class. Additionally to the confusion matrices of the models are provided. Their structure follows the definition given in Section 5.1.

5.6. Classification Performance and Generalization

First, the model trained without point normals is evaluated using the validation set. As shown in Table 5.7, an accuracy of 82.3% was reached. The classes Underpants, Sock, and Shortpants are detected best. In the confusion matrix (see Table 5.8), the most inter-class confusion is observable between the classes Sweater and Sweatpants. Also notable is the relatively high rate at which Sweatpants are mistaken for Jeans. This does not occur as commonly the other way around.

Table 5.7.: Classification performance of the seven-class classifier without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.782	0.813	0.797	870
T-Shirt	0.890	0.888	0.889	872
Sweatpants	0.685	0.706	0.696	895
Sweater	0.761	0.713	0.736	963
Underpants	0.885	0.936	0.910	312
Sock	0.986	0.976	0.981	288
Shortpants	0.928	0.913	0.920	900
Accuracy			0.823	5100
Average	0.845	0.849	0.847	5100
Weighted Average	0.823	0.823	0.823	5100

Table 5.8.: Confusion matrix of the seven-class classifier without point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater	Underpants	Sock	Shortpants
Jeans	0.813	0.006	0.06	0.084	0.0	0.0	0.038
T-Shirt	0.01	0.888	0.06	0.017	0.013	0.0	0.013
Sweatpants	0.118	0.038	0.706	0.135	0.0	0.0	0.002
Sweater	0.061	0.028	0.186	0.713	0.0	0.0	0.011
Underpants	0.0	0.022	0.003	0.003	0.936	0.013	0.022
Sock	0.01	0.0	0.0	0.0	0.014	0.976	0.0
Shortpants	0.022	0.026	0.007	0.007	0.026	0.0	0.913

5. Experiment

Second, the network was trained with point normals. As already mentioned, it outperforms the model trained in the first experiment by a significant margin as can be seen in Table 5.9. It is also evident, that it performs better in every aspect over all classes. Class-specific strengths and weaknesses are also apparent in this model. In the confusion matrix shown in Table 5.10, the improvements over the model without point normals are visible. The inter-class confusions are relatively low throughout the whole table, with the comparably high misclassification rate of Sweatpants as Sweaters being an exception. While the confusion between the two classes is bidirectional in the first experiment, it is unidirectional in this run.

Table 5.9.: Classification performance of the seven-class classifier with point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.905	0.957	0.931	870
T-Shirt	0.971	0.956	0.964	872
Sweatpants	0.917	0.775	0.840	895
Sweater	0.829	0.918	0.871	963
Underpants	0.962	0.965	0.963	312
Sock	0.993	0.990	0.991	288
Shortpants	0.969	0.967	0.968	900
Accuracy			0.922	5100
Average	0.935	0.933	0.933	5100
Weighted Average	0.924	0.922	0.921	5100

Table 5.10.: Confusion matrix of the seven-class classifier with point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater	Underpants	Sock	Shortpants
Jeans	0.957	0.003	0.006	0.022	0.0	0.0	0.011
T-Shirt	0.011	0.956	0.009	0.01	0.006	0.0	0.007
Sweatpants	0.037	0.015	0.775	0.169	0.0	0.0	0.004
Sweater	0.021	0.006	0.052	0.918	0.001	0.0	0.002
Underpants	0.0	0.01	0.0	0.0	0.965	0.006	0.019
Sock	0.0	0.0	0.0	0.0	0.01	0.99	0.0
Shortpants	0.027	0.0	0.0	0.003	0.003	0.0	0.967

5.6. Classification Performance and Generalization

Following the evaluations on validation data, both models are evaluated using the independent test data. In doing this, the generalization capabilities of the network are tested as the test data was recorded independently from the training and validation data. This test is the closest representation of real-world performance, as the live classifier does not estimate the point normals and has to handle clothes different from those in the training data. Table 5.11 presents the test results of the test run without point normals. In contrast to the results on validation data, the differences between the class scores are more pronounced. In Table 5.12 generally more inter class confusion is evident. Most notably, Sweatpants and Sweaters are often mistaken for Jeans and Socks, a recall of 1.0 is achieved.

Table 5.11.: Classification performance of the seven-class classifier without point normals on independent test data.

	Precision	Recall	F1-score	Support
Jeans	0.569	0.693	0.625	900
T-Shirt	0.809	0.868	0.838	900
Sweatpants	0.702	0.528	0.602	900
Sweater	0.731	0.571	0.641	900
Underpants	0.761	0.880	0.816	300
Sock	0.926	1.000	0.962	300
Shortpants	0.846	0.929	0.886	900
Accuracy			0.744	5100
Average	0.763	0.781	0.767	5100
Weighted Average	0.745	0.744	0.738	5100

Table 5.12.: Confusion matrix of the seven-class classifier without point normals on independent test data.

	Jeans	T-Shirt	Sweatpants	Sweater	Underpants	Sock	Shortpants
Jeans	0.693	0.006	0.106	0.12	0.0	0.0	0.076
T-Shirt	0.013	0.868	0.007	0.004	0.087	0.009	0.012
Sweatpants	0.22	0.171	0.528	0.079	0.0	0.0	0.002
Sweater	0.232	0.008	0.112	0.571	0.0	0.0	0.077
Underpants	0.003	0.057	0.0	0.0	0.88	0.053	0.007
Sock	0.0	0.0	0.0	0.0	0.0	1.0	0.0
Shortpants	0.058	0.001	0.0	0.007	0.006	0.0	0.929

5. Experiment

In the final experiment of this series, the model trained with point normals was evaluated on the independent test set. As already mentioned, it performed inferior compared to the model trained without point normals. Tables 5.13 and 5.14 indicate that the lower accuracy is caused by a slightly worse performance on most classes. The confusion pattern is generally comparable to the previous experiment run with minor differences.

Table 5.13.: Classification performance of the seven-class classifier with point normals on independent test data.

	Precision	Recall	F1-score	Support
Jeans	0.556	0.616	0.584	900
T-Shirt	0.809	0.688	0.744	900
Sweatpants	0.739	0.611	0.669	900
Sweater	0.694	0.622	0.656	900
Underpants	0.709	0.903	0.795	300
Sock	0.905	0.950	0.927	300
Shortpants	0.790	0.957	0.865	900
Accuracy			0.725	5100
Average	0.743	0.764	0.749	5100
Weighted Average	0.728	0.725	0.722	5100

Table 5.14.: Confusion matrix of the seven-class classifier with point normals on independent test data.

	Jeans	T-Shirt	Sweatpants	Sweater	Underpants	Sock	Shortpants
Jeans	0.616	0.004	0.129	0.179	0.0	0.0	0.072
T-Shirt	0.086	0.688	0.0	0.002	0.106	0.012	0.107
Sweatpants	0.141	0.15	0.611	0.093	0.0	0.0	0.004
Sweater	0.223	0.004	0.087	0.622	0.0	0.0	0.063
Underpants	0.0	0.01	0.0	0.0	0.903	0.063	0.023
Sock	0.0	0.0	0.0	0.0	0.05	0.95	0.0
Shortpants	0.042	0.0	0.0	0.0	0.001	0.0	0.957

6. Discussion

In the following sections, the work and experiment results are discussed. First, Section 6.1 discusses the results of the experiments conducted to evaluate the PointNet clothes classifier developed in this work. Second, the validity of the work conducted is debated in Section 6.2. Finally, Section 6.3 discusses the applicability of the developed system in a robot setup.

6.1. Performance in Experiments

The experiments conducted in Chapter 5 provide various measurements about and insights into the PointNet classifier when applied to non-rigid objects. As shown in the overview of the experiments (see Table 5.1), multiple aspects of the classifier are inspected independently.

In the experiments evaluating the optimal number of points required for the task of clothes classification, the general expectation that an increase in the number of input points used leads to an improved classification accuracy was confirmed. The increase of the classification accuracy in relation to the number of points used can be approximated by a logarithmic function while the computational effort grows linearly in relation to the number of points.

Possibly, the performance using a very low number of points could be improved by using a distance-based sampling method instead of random sampling. Using random sampling, points close to each other might be selected leading to a poor representation of the clothes. In case of larger numbers of points, random sampling is faster and sufficient as the probability of selecting a sub-optimal set of input points decreases significantly.

Especially for applications in which a robot grasps a sample and classifies it afterward, the effect of the grasp positions is relevant. When the classification accuracy achieved while grasping the clothes anywhere in the pieces is not sufficient, the results in Table 5.2, indicate that implementing a system that grasps the pieces at the edge is not worth the effort. To achieve a significant improvement in classification performance, a method to reliably grasp the samples at the top is required. Based on the results, the classification performance reached by the model handling grasps anywhere was deemed sufficient and the grasp position is used for all other models trained. Also, the training data generated by grasping the clothes anywhere includes a larger variety in the data, which could generally yield a more robust model. For the experiments, the datasets with samples grasped from the top edge and inside the clothes (see Figure 4.5) were combined resulting in the datasets used in the experiments (see Figure 5.2). Thereby, a larger dataset in the evaluation for more diverse grasping positions is caused. This might have lessened the effect of the diverse grasp positions. But as the disparity in dataset size between the grasp positions Top and Edge is similar to the difference between Edge and Anywhere, the difference in classification accuracy is still comparable.

6. Discussion

To investigate whether the classifier is able to capture the characteristics of the clothes material, the similar shapes experiment was conducted (see Section 5.4). As the clothes types classified did not differ significantly in their shape but in their material (T-Shirt: cotton, Jersey: polyester). It was assumed that the different materials lead to different shapes when the clothes are grasped at a single point. The results achieved on validation data indicate that there are indeed differences between the types significant enough to differentiate between the classes. On the independent test data, however, the classifier drastically failed as the accuracy achieved was significantly worse than a random guess. This is a strong indication for overfitting. It means that the model focused on differences between the pieces in the training set which are specific to the individual pieces or the data collected and do not apply to the type of clothing in general. These results indicate that either the assumption made about the change of the shapes due to the material is wrong in general or that the PointNet is not able to capture the features. Possibly, a method capable of processing local features such as PointNet++ [QYSG17] or DGCNN [WSL⁺19] is able to learn to differentiate between similar shapes. Also, on the diverse shapes used as a baseline in the experiments, a significantly less pronounced overfitting effect can be observed for the experiment run with point normals.

A deeper insight into the inner workings of the PointNet classifier was gained in the significant points experiment in Section 5.5. For exemplary samples, the points selected for the classification are highlighted in the input point cloud. The experiment showed, that the model trained without point normals in the input mainly focuses on points on the outer border of the pieces on the x-y-plane. Some apparently randomly selected points in the center of the pieces are in fact on the outer border in the z-direction. The model trained with point normals shows a relatively similar pattern to the one without point normals. The main difference between the two is that the samples with point normals show a slightly larger portion of seemingly randomly selected significant points. It can be assumed that some of these points are selected because their point normals include especially large parameters. These observations could be caused by the PointNet property of only regarding global features. This would mean that the network is unable to focus on smaller parts of the samples individually and only considers the sample as a whole. This hypothesis would also explain the inability of the network to generalize on similar shapes made from different materials evident in the similar shape experiment. When only regarding the sample as a whole, smaller details such as wrinkle patterns of the clothes cannot be considered in the classification process.

The results are especially interesting when compared to the significant points identified in other applications of PointNet. A similar experiment was conducted in the original PointNet paper [QSKG17] and in the dissertation of Gao [Gao21]. Exemplary samples of both experiments shown in the publications are also available in the appendix of this work (see Figures C.1 and C.2). The PointNet paper focuses on the classification of rigid objects. It is observed that the significant points form a skeleton of the objects. While Gao still utilizes rigid objects, the objects are not classified, but their orientation in space is estimated. In contrast to the PointNet paper, no such skeleton-like structures were observable in the selection of significant points. The significant points are spread out evenly across the sample point cloud. The reason for the higher density of significant points is that the same point selection process as

used in this work and the original PointNet is applied on only 256 points, which is a quarter of the 1024 points used in the other approaches.

The seven-class classifier designed to be used in the live-classifier is evaluated in detail in Section 5.6. Analogical to the similar shape experiment, the classification performance is regarded both on the validation set and the independent test set. This was done again to investigate whether the accuracy achieved on test data is transferable to real-world applications. While the performance achieved on the independent test data is significantly worse than on validation data, it cannot be considered a total failure as in the similar shape experiment. Also, no fatal failures of any per-class score can be reported. Most notably, whereas the model trained with point normals outperformed the model without point normals on validation data, it performed inferior on test data. This reinforces the impression that point normals can lead to overfitting of a model which was already apparent in the baseline tests of the similar shapes experiment.

In general, the classification performance is significantly worse than the performance achieved by PointNet and other approaches achieved on benchmark data such as ModelNet40 (see Table 2.3). However, as in this work, the network is applied to classify non-rigid objects instead of rigid objects, the task is much harder and a decrease in classification accuracy was expected despite using fewer classes. The classifier outperforms PointNet++ without intrinsic features which achieved a classification performance of 60.18% on the SHREC15 benchmark (see Table 3.1). Due to less challenging shapes the SHREC15 benchmark, a different number of classes, and significantly more restrictions for the recording of train and test data, the results are not directly comparable. Probably, the restrictions on the object position, size, and distance to the camera (in conjunction with the preprocessing steps) compensate for the higher flexibility of classified objects.

The results achieved with the seven-class classifier (see Table 5.6) can be considered to be sufficient for simple tasks. Note, that only the classification accuracy of the model trained without point normals measured on test data (74.4%) is transferable to a real-world scenario. To further improve the results, regarding clothes from multiple viewing angles might help. Even sequences of point clouds could be processed in a classification pipeline with a recurrent component such as an LSTM [HS97]. This would allow capturing temporal features and multiple viewing angles when the samples are rotated during sequence recording. Also, using a classification pipeline capable of capturing local features might help with the classification of similar shapes and possibly also with the other classes.

6.2. Threats to Validity

In the following, threats to the validity of the results gathered in the experiments are discussed. The results gathered in one experiment were assumed to apply in other experiments while changing other variables. For example, the optimal number of points was evaluated using four classes, but that does not imply that it is still optimal in a case with seven classes or for similar shapes. However, the experiments in this work are supposed to provide a general impression of the effect and order of magnitude of the variables investigated and not to yield the best results possible. Also, the number of different clothes is very restricted. The

6. Discussion

selection of clothes types was a compromise. The clothes commonly worn by males in western societies were selected because they were available and are relatively uniform. In contrast, in women's clothing, more variation in styles is common. There, the types of clothes merge into each other. Also, the general shape is not always sufficient for classification, as for example sweatpants, a pair of leggings, or tights are comparatively similar to the clothes used in the similar shape experiment in Section 5.4. Situations in a real-world application, especially when sorting laundry, clothes might be dirty, wet, and partially or completely inside-out. This was not tested in this work, but as this technology is still in the early stages of development, general usability in a laboratory with constraints on the clothes such as being clean and dry is still given. Additional to the concerns detailed in this section, when applied to a robot, more problems could surface. These are listed in the following section.

6.3. Applicability in Robotic Systems

When the live classifier is deployed on a robot, several implications of the new setup need to be considered. A depth camera with a wide field of view and the capability of taking measurements in a field relatively near to the sensor is required. The recording setup used in this work (see Section 4.1.1) allows to select the position of the person holding the clothes without a connection to the camera as can be seen in Figure 4.2. When using a robot such as the PR2 service robot, the maximal distance between camera and clothes is constrained by the length of the robot's arms. This constraint causes the requirement for a low minimal measurement distance. The larger field of view is required to capture as much information about the clothes as possible even if they are held close to the sensor. Another influence of the fixed relation between camera and robot arms might be that the position of the robot's camera is not optimal to observe objects hanging below the robot's grippers. This is the case when the robot's arms can not be lifted above the camera. As mentioned, the Microsoft Azure Kinect presented in Section 2.5 offers both, a wide FOV as well as capabilities for measuring objects relatively close to the sensor. However, it is not absolutely certain how the results might be affected by using a different camera. The changes could be positive, as a better camera with less noise in the measurements can improve the classification accuracy.

Also, the position at which the clothes are grasped might differ from the current state. While this might improve the results, as the clothes are held at a more consistent position, the robot arm might be included in the depth information. Nonetheless, there are pitfalls of using a robotic arm, as it could be captured in the point cloud measurements. In such cases, a system removing the robot's links from the point cloud based on its forward kinematic such as the `robot_self_filter` [21] can address the issue.

To achieve an actually autonomous process, the task of locating and grasping clothes needs to be solved. Otherwise, a person is required to hand the clothes to the robotic gripper. The challenge of grasping clothes is especially demanding considering that only a single piece can be classified at a time. So the robot needs a procedure that results in a single piece of clothing grasped and ready for classification. One could achieve this by including information on the grippers opening angle and tactile sensor measurements.

6.3. Applicability in Robotic Systems

In the future, many robotic systems will be upgraded to the ROS2 middleware. Still, most of this work such as the train and testing pipeline, the file classifier, and data inspection do not depend on ROS and do not need to be changed. Only the package definition, live classifier, and result visualization have to be adapted to the new ecosystem. As only standard components and capabilities are used for which drop-in replacements are available in ROS2, this can be considered to be an easily solvable issue.

Altogether, a good integration of the live classifier in a robotic system can be expected. The live classifier is already well integrated into the ROS environment. Potential issues as outlined can be approached with relatively low effort. The greatest adaption possibly required would be re-recording the training dataset when the set created in this work can not be adapted to the viewing angle of the robot. But even in that case, the general structure of the deep learning model, the train and test pipeline, and the live classifier itself do not require adaptations. Especially when considering the classification performance and generalization capabilities demonstrated in the experiments (see Section 5.6), the results of this work can be considered to be promising.

7. Conclusion

In this work, the applicability of point cloud data based classification of non-rigid objects was assessed on the example of the classification of clothes grasped at a single point. A dataset of images and point clouds was collected to allow testing and training of a deep learning based model approaching the task. An existing pipeline for training and testing was extended to the needs of the approach and experiments presented in this work. Various experiments were conducted to evaluate diverse aspects of the classification performance of the trained PointNet models on clothes. Finally, the trained models were integrated into the ROS environment to be usable as a software component in a robotic setup. Throughout the whole system, a strong emphasis was placed on generalization by taking several measures against overfitting and performing analyses with independent test data. This goal of a generalizing model was achieved as it was shown in the experiments. In the similar shapes experiment, the limitations of the approach were demonstrated. Also, the analyses of the experiment results concluded, that while the inclusion of point normals in the input data yielded significant improvements on data similar to the training set, it seemingly caused the model to overfit as the performance on independent test data was inferior to models trained without point normals.

As outlined in the introduction, this work can be utilized as the basis of several greater robotic applications in the field of handling, perceiving, and manipulating clothes. Also, the dataset produced is usable to train and evaluate future approaches to non-rigid object classification.

Possible future work could evaluate the usage of a runtime efficient network architecture that captures local features such as the DGCNN for the task. Also, recurrent model architectures could process sample sequences to make use of the temporal dimension as well as multiple viewing angles. While this work approaches the issue of overfitting mainly by solely relying on point cloud input data, methods of multimodal object classification could be evaluated for this task. For example, RGB image information could be used. But also less conventional approaches such as tactile sensors in the fingers grasping the clothes or a force sensor that measures the weight of the grasped object would be possible.

Bibliography

- [ABC⁺16] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: A system for large-scale machine learning. *12th USENIX symposium on operating systems design and implementation*, 2016.
- [ADBB17] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep Reinforcement Learning: A Brief Survey. *IEEE Signal Processing Magazine*, 34(6):26–38, November 2017. Conference Name: IEEE Signal Processing Magazine.
- [BEF⁺21] Marc Bestmann, Timon Engelke, Niklas Fiedler, Jasper Güldenstein, Jan Gutsche, Jonas Hagge, and Florian Vahl. TORSO-21 Dataset: Typical Objects in RoboCup Soccer 2021. In *RoboCup Symposium*. Springer, June 2021.
- [BGK17] Shobhit Bhatnagar, Deepanway Ghosal, and Maheshkumar H. Kolekar. Classification of fashion article images using convolutional neural networks. In *Fourth International Conference on Image Information Processing (ICIIP)*, pages 1–6, Shimla, December 2017. IEEE.
- [BHG⁺19] Adith Boloor, Xin He, Christopher Gill, Yevgeniy Vorobeychik, and Xuan Zhang. Simple Physical Adversarial Examples against End-to-End Autonomous Driving Models. In *International Conference on Embedded Software and Systems (ICESS)*, pages 1–7. IEEE, June 2019.
- [BOL⁺05] Bernhard Büttgen, T. Oggier, M. Lehmann, R. Kaufmann, and F. Lustenberger. CCD / CMOS Lock-In Pixel for Range Imaging : Challenges , Limitations and State-of-the-Art, 2005.
- [BPK11] Christian Bersch, Benjamin Pitzer, and Sören Kammel. Bimanual robotic cloth manipulation for laundry folding. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1413–1419. IEEE, September 2011.
- [BWL20] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, April 2020.

BIBLIOGRAPHY

- [CCG⁺18] Xuzhan Chen, Youping Chen, Kashish Gupta, Jie Zhou, and Homayoun Najjaran. SliceNet: A proficient model for real-time 3D shape-based recognition. In *Neurocomputing*, volume 316, pages 144–155. Elsevier, November 2018.
- [CTSO03] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On Visual Similarity Based 3D Model Retrieval. *Computer Graphics Forum*, 22(3):223–232, 2003.
- [DDS⁺09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *Conference on Computer Vision and Pattern Recognition*, page 2, Miami, FL, USA, 2009. IEEE.
- [Den12] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29:141–142, November 2012.
- [DT21] Olga Digilina and Irina Teslenko. The Robotics Market: Development Prerequisites, Features and Prospects. *SHS Web of Conferences*, 101:02029, 2021.
- [DYZL19] Chao Duan, Panpan Yin, Yan Zhi, and Xingxing Li. Image Classification of Fashion-MNIST Data Set Based on VGG Network. In *2nd International Conference on Information Science and Electronic Technology (ISET)*, page 4. IEEE, 2019.
- [EE19] Roger Eggen and Maurice Eggen. Thread and Process Efficiency in Python. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Athens, 2019.
- [Gao21] Ge Gao. *Learning 6D Object Pose from Point Clouds*. Dissertation, Universität Hamburg, 2021.
- [GKUP11] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. BlenSor: Blender Sensor Simulation Toolbox. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Song Wang, Kim Kyungnam, Bedrich Benes, Kenneth Moreland, Christoph Borst, Stephen DiVerdi, Chiang Yi-Jen, and Jiang Ming, editors, *Advances in Visual Computing*, Lecture Notes in Computer Science, pages 199–208, Berlin, Heidelberg, 2011. Springer.
- [GLH⁺21] Ge Gao, Mikko Lauri, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. CloudAAE: Learning 6D Object Pose Regression with On-Line Data Synthesis on Point Clouds. In *arXiv:2103.01977*, 2021.
- [GLW⁺20] Ge Gao, Mikko Lauri, Yulong Wang, Xiaolin Hu, Jianwei Zhang, and Simone Frintrop. 6D Object Pose Regression via Supervised Learning on Point Clouds. In *International Conference on Robotics and Automation (ICRA)*, pages 3643–3649, Paris, France, January 2020. IEEE.

- [GLZF18] Ge Gao, Mikko Lauri, Jianwei Zhang, and Simone Frintrop. Occlusion Resistant Object Rotation Regression from Point Cloud Segments. In *European Conference on Computer Vision*. Springer, 2018.
- [GVH03] Brian P Gerkey, Richard T Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *11th International Conference on Advanced Robotics (ICAR)*, 2003.
- [HMvdW⁺20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997.
- [Hun07] John D. Hunter. Matplotlib: A 2D Graphics Environment. *Computing in Science Engineering*, 9(3):90–95, May 2007. Conference Name: Computing in Science Engineering.
- [Hä16] Martin Hägele. Robots Conquer the World. *IEEE Robotics Automation Magazine*, 23(1):120–118, March 2016.
- [JS17] Pablo Jiménez Schlegel. Visual grasp point localization, classification and state recognition in robotic manipulation of cloth: An overview. *Robotics and Autonomous Systems*, 92:107–125, June 2017.
- [JYT⁺17] Jichao Jiao, Libin Yuan, Weihua Tang, Zhongliang Deng, and Qi Wu. A Post-Rectification Approach of Depth Images of Kinect v2 for 3D Reconstruction of Indoor Scenes. *ISPRS International Journal of Geo-Information*, 6:349, November 2017.
- [KAM20] M. Kayed, A. Anter, and H. Mohamed. Classification of Garments from Fashion MNIST Dataset Using CNN LeNet-5 Architecture. In *2020 International Conference on Innovative Trends in Communication and Computer Engineering (ITCE)*, pages 238–243, February 2020.
- [KB15] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. 2015. arXiv: 1412.6980.
- [KFR03] Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164, Goslar, Germany, June 2003. Eurographics Association.

BIBLIOGRAPHY

- [KH04] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154 vol.3. IEEE, September 2004.
- [KSH17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017.
- [LBH18] Lorenzo Luciano and A. Ben Hamza. Deep learning with geodesic moments for 3D shape classification. *Pattern Recognition Letters*, 105:182–190, April 2018.
- [LGB⁺11] Zhouhui Lian, Afzal Godil, Benjamin Bustos, Mohamed Daoudi, Jeroen Hermans, Shun Kawamura, Yukinori Kurita, Guillaume Lavoué, Hien Nguyen, Ryutarou Ohbuchi, Yuki Ohkita, Yuya Ohishi, Fatih Porikli, Martin Reuter, Ivan Sipiran, Dirk Smeets, Paul Suetens, Hedi Tabia, and Dirk Vandermeulen. SHREC '11 Track: Shape Retrieval on Non-rigid 3D Watertight Meshes. In *Eurographics Workshop on 3D Object Retrieval*, January 2011.
- [LLQ⁺16] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. DeepFashion: Powering Robust Clothes Recognition and Retrieval with Rich Annotations. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1096–1104, Las Vegas, NV, USA, June 2016. IEEE.
- [Mic04] Olivier Michel. Cyberbotics Ltd. Webots™: Professional Mobile Robot Simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, March 2004.
- [MPG20] Khatereh Meshkini, Jan Platos, and Hassan Ghassemian. An Analysis of Convolutional Neural Network for Fashion Images Classification (Fashion-MNIST). In Sergey Kovalev, Valery Tarassov, Vaclav Snasel, and Andrey Sukhanov, editors, *Proceedings of the Fourth International Scientific Conference “Intelligent Information Technologies for Industry” (IITI'19)*, Advances in Intelligent Systems and Computing, pages 85–95, Cham, 2020. Springer International Publishing.
- [MS15] Daniel Maturana and Sebastian Scherer. VoxNet: A 3D Convolutional Neural Network for real-time object recognition. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, September 2015.
- [MSCTLA10] Jeremy Maitin-Shepard, Marco Cusumano-Towner, Jinna Lei, and Pieter Abbeel. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In *International Conference on Robotics and Automation (ICRA)*, pages 2308–2315. IEEE, May 2010.

- [PGM⁺19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach 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. *arXiv:1912.01703 [cs, stat]*, December 2019.
- [PSR⁺15] David Pickup, Xianfang Sun, Paul L. Rosin, Ralph Robert Martin, Zhiqian Cheng, Sipin Nie, and Longcun Jin. SHREC'15 Track: Canonical forms for non-rigid 3D shape retrieval. In *Eurographics Workshop on 3D Object Retrieval*, pages 316–327, Zurich, Switzerland, May 2015.
- [PSR⁺16] D. Pickup, X. Sun, P. L. Rosin, R. R. Martin, Z. Cheng, Z. Lian, M. Aono, A. Ben Hamza, A. Bronstein, M. Bronstein, S. Bu, U. Castellani, S. Cheng, V. Garro, A. Giachetti, A. Godil, L. Isaia, J. Han, H. Johan, L. Lai, B. Li, C. Li, H. Li, R. Litman, X. Liu, Z. Liu, Y. Lu, L. Sun, G. Tam, A. Tatsuma, and J. Ye. Shape Retrieval of Non-rigid 3D Human Models. In *International Journal of Computer Vision*, volume 120, pages 169–193, November 2016.
- [PVG⁺11] Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, and David Cournapeau. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:6, 2011.
- [QCG⁺09] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, and others. ROS: an open-source Robot Operating System. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [QLHG19] Charles R. Qi, Or Litany, Kaiming He, and Leonidas J. Guibas. Deep Hough Voting for 3D Object Detection in Point Clouds. *arXiv:1904.09664 [cs]*, August 2019.
- [QLW⁺18] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. *arXiv:1711.08488 [cs]*, April 2018.
- [QSKG17] Charles R. Qi, Hao Su, Mo Kaichun, and Leonidas J. Guibas. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 77–85, Honolulu, HI, July 2017. IEEE.
- [QSN⁺16] 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. pages 5648–5656, 2016.

BIBLIOGRAPHY

- [QYSG17] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. *arXiv:1706.02413 [cs]*, June 2017.
- [RC11] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *International Conference on Robotics and Automation*, pages 1–4. IEEE, May 2011.
- [SMKLM15] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-View Convolutional Neural Networks for 3D Shape Recognition. In *International Conference on Computer Vision*, pages 945–953. IEEE, 2015.
- [SZS⁺14] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.
- [TCUM19] Yoshihisa Tsurumine, Yunduan Cui, Eiji Uchibe, and Takamitsu Matsubara. Deep reinforcement learning with smooth policy update: Application to robotic cloth manipulation. *Robotics and Autonomous Systems*, 112:72–83, February 2019.
- [TDCH21] Michal Tölgyessy, Martin Dekan, Lubos Chovanec, and Peter Hubinský. Evaluation of the Azure Kinect and Its Comparison to Kinect V1 and Kinect V2. *Sensors*, 21:413, January 2021.
- [TMRS11] Tomoya Tamei, Takamitsu Matsubara, Akshara Rai, and Tomohiro Shibata. Reinforcement learning of clothing assistance with a dual-arm robot. In *International Conference on Humanoid Robots*, pages 733–738. IEEE, October 2011.
- [VBK16] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order Matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2016.
- [WSK⁺15] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3D ShapeNets: A Deep Representation for Volumetric Shapes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920. IEEE, 2015.
- [WSL⁺19] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. Dynamic Graph CNN for Learning on Point Clouds. *ACM Transactions on Graphics*, 38, June 2019.
- [XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *arXiv:1708.07747 [cs, stat]*, September 2017.

BIBLIOGRAPHY

- [Y109] Kimitoshi Yamazaki and Masayuki Inaba. A Cloth Detection Method based on Image Wrinkle Feature for Daily Assistive Robots. In *IAPR Conference on Machine Vision Applications*, page 4, Yokohama, 2009.
- [Y113] Kimitoshi Yamazaki and Masayuki Inaba. Clothing classification using image features derived from clothing fabrics, wrinkles and cloth overlaps. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 2710–2717. IEEE, November 2013.
- [ZPK18] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847 [cs]*, January 2018.

Online References

- [1] "ROS Master." http://wiki.ros.org/robot_self_filter, 2008 – 2018. last accessed: 2021-08-21.
- [2] "common_msgs." https://github.com/ros/common_msgs, 2009 – 2021. last accessed: 2021-07-22.
- [3] Q. Group, "Estimating Surface Normals in a PointCloud." <https://www.qt.io/>, 2021. last accessed: 2021-08-21.
- [4] ROS Documentation, "PointCloud2 Message." http://docs.ros.org/en/api/sensor_msgs/html/msg/PointCloud2.html. last accessed: 2021-08-28.
- [5] ROS Documentation, "PointField Message." http://docs.ros.org/en/api/sensor_msgs/html/msg/PointField.html. last accessed: 2021-08-28.
- [6] "Estimating Surface Normals in a PointCloud." https://pointclouds.org/documentation/tutorials/normal_estimation.html. last accessed: 2021-07-31.
- [7] "Normal estimation." http://www.open3d.org/docs/release/tutorial/geometry/surface_reconstruction.html#Normal-estimation. last accessed: 2021-07-31.
- [8] Qi, Charles R., "pointnet (GitHub)." <https://github.com/charlesq34/pointnet>. last accessed: 2021-08-28.
- [9] Qi, Charles R., "pointnet2 (GitHub)." <https://github.com/charlesq34/pointnet2>. last accessed: 2021-08-28.
- [10] Xu Yan, "Pointnet_Pointnet2_pytorch (GitHub)." https://github.com/yanx27/Pointnet_Pointnet2_pytorch. last accessed: 2021-08-28.
- [11] T. Wiedemeyer, "IAI Kinect2." https://github.com/code-iai/iai_kinect2, 2014 – 2015. last accessed: 2021-08-28.
- [12] J. Blake, F. Ehtler, C. Kerl, and L. Xiang, "libfreenect2." <https://github.com/OpenKinect/libfreenect2>, 2013 – 2020. last accessed: 2021-08-28.
- [13] K. Inc., "Kaggle." <https://www.kaggle.com/>, 2021. last accessed: 2021-08-21.
- [14] C. S. Department, "SHREC." <https://shrec.net>, 2013 – 2021. last accessed: 2021-05-18.

ONLINE REFERENCES

- [15] D. Pickup, X. Sun, P. L. Rosin, R. R. Martin, Z. Cheng, S. Nie, and L. Jin, "SHREC'15 - Canonical Forms for Non-Rigid 3D Shape Retrieval ." <http://www.cs.cf.ac.uk/shaperetrieval/shrec15/index.html>, 2015. last accessed: 2021-05-18.
- [16] Y. Wang, "DGCNN." <https://github.com/WangYueFt/dgcnn>, 2019 – 2020. last accessed: 2021-05-22.
- [17] J. L. Jeremy Maitin-Shepard, Marco Cusumano-Towner and P. Abbeel, "Cloth Grasp Point Detection based on Multiple-View Geometric Cues with Application to Robotic Towel Folding - Material." <https://rll.berkeley.edu/pr/icra10/>, 2010. last accessed: 2021-05-24.
- [18] J. developers, "Joblib." <https://joblib.readthedocs.io>, 2008 – 2018. last accessed: 2021-08-28.
- [19] "Beurteilung eines binären Klassifikators." https://de.wikipedia.org/wiki/Beurteilung_eines_bin%C3%A4ren_Klassifikators. last accessed: 2021-05-28.
- [20] "Evaluation of binary classifiers." https://en.wikipedia.org/wiki/Evaluation_of_binary_classifiers. last accessed: 2021-05-28.
- [21] E. Marder-Eppstein, "robot_self_filter." http://wiki.ros.org/robot_self_filter, 2015 – 2020. last accessed: 2021-07-01.

Appendices

A. Acronyms

ToF time-of-flight

MLP multi-layer perceptron

CNN convolutional neural network

FLOP floating point operation

GIL global interpreter lock

FOV field of view

B. Experiment Results

Additional results of experiments, which were not discussed further in Section 5 and 6 are presented in this section.

B.1. Grasp Positions

Top

Table B.1.: Classification performance for four classes grasped at the top without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.935	0.894	0.914	292
T-Shirt	0.957	0.970	0.963	296
Sweatpants	0.896	0.836	0.865	298
Sweater	0.875	0.955	0.913	314
Accuracy			0.914	1200
Average	0.916	0.914	0.914	1200
Weighted Average	0.915	0.914	0.914	1200

Table B.2.: Confusion matrix for four classes grasped at the top without point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.894	0.014	0.038	0.055
T-Shirt	0.014	0.97	0.014	0.003
Sweatpants	0.047	0.03	0.836	0.087
Sweater	0.0	0.0	0.045	0.955

Table B.3.: Classification performance for four classes grasped at the top with point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.997	0.986	0.991	292
T-Shirt	0.970	1.000	0.985	296
Sweatpants	0.915	0.936	0.925	298
Sweater	0.950	0.911	0.930	314
Accuracy			0.958	1200
Average	0.958	0.958	0.958	1200
Weighted Average	0.958	0.958	0.957	1200

Table B.4.: Confusion matrix for four classes grasped at the top with point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.986	0.007	0.003	0.003
T-Shirt	0.0	1.0	0.0	0.0
Sweatpants	0.0	0.017	0.936	0.047
Sweater	0.003	0.006	0.08	0.911

Edge

Table B.5.: Classification performance for four classes grasped at the edge without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.904	0.835	0.868	589
T-Shirt	0.840	0.955	0.894	584
Sweatpants	0.849	0.748	0.795	618
Sweater	0.775	0.824	0.799	609
Accuracy			0.839	2400
Average	0.842	0.841	0.839	2400
Weighted Average	0.842	0.839	0.838	2400

Table B.6.: Confusion matrix for four classes grasped at the edge without point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.835	0.032	0.044	0.088
T-Shirt	0.007	0.955	0.021	0.017
Sweatpants	0.026	0.091	0.748	0.136
Sweater	0.053	0.051	0.072	0.824

Table B.7.: Classification performance for four classes grasped at the edge with point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.912	0.985	0.947	589
T-Shirt	0.977	0.947	0.962	584
Sweatpants	0.909	0.856	0.882	618
Sweater	0.877	0.887	0.882	609
Accuracy			0.917	2400
Average	0.919	0.919	0.918	2400
Weighted Average	0.918	0.917	0.917	2400

Table B.8.: Confusion matrix for four classes grasped at the edge with point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.985	0.002	0.005	0.008
T-Shirt	0.014	0.947	0.026	0.014
Sweatpants	0.032	0.01	0.856	0.102
Sweater	0.046	0.01	0.057	0.887

Everywhere

Table B.9.: Classification performance for four classes grasped everywhere without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.916	0.768	0.836	870
T-Shirt	0.905	0.961	0.932	872
Sweatpants	0.766	0.708	0.736	895
Sweater	0.722	0.837	0.775	963
Accuracy			0.818	3600
Average	0.827	0.819	0.82	3600
Weighted Average	0.824	0.818	0.818	3600

Table B.10.: Confusion matrix for four classes grasped everywhere without point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.768	0.017	0.098	0.117
T-Shirt	0.01	0.961	0.009	0.019
Sweatpants	0.026	0.051	0.708	0.215
Sweater	0.03	0.028	0.105	0.837

Table B.11.: Classification performance for four classes grasped everywhere without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.923	0.964	0.943	870
T-Shirt	0.965	0.974	0.969	872
Sweatpants	0.886	0.845	0.865	895
Sweater	0.879	0.874	0.877	963
Accuracy			0.913	3600
Average	0.913	0.914	0.914	3600
Weighted Average	0.912	0.913	0.912	3600

Table B.12.: Confusion matrix for four classes grasped everywhere without point normals on validation data.

	Jeans	T-Shirt	Sweatpants	Sweater
Jeans	0.964	0.008	0.009	0.018
T-Shirt	0.008	0.974	0.01	0.008
Sweatpants	0.038	0.013	0.845	0.104
Sweater	0.03	0.012	0.083	0.874

B.2. Similar Shape

Similar Shapes Performance

Table B.13.: Classification performance for similar shapes grasped everywhere without point normals on validation data.

	Precision	Recall	F1-score	Support
T-Shirt	0.782	0.792	0.787	890
Jersey	0.794	0.785	0.789	910
Accuracy			0.788	1800
Average	0.788	0.788	0.788	1800
Weighted Average	0.788	0.788	0.788	1800

Table B.14.: Confusion matrix for similar shapes grasped everywhere without point normals on validation data.

	T-Shirt	Jersey
T-Shirt	0.792	0.208
Jersey	0.215	0.785

B. Experiment Results

Table B.15.: Classification performance for similar shapes grasped everywhere with point normals on validation data.

	Precision	Recall	F1-score	Support
T-Shirt	0.913	0.872	0.892	890
Jersey	0.880	0.919	0.899	910
Accuracy			0.896	1800
Average	0.896	0.895	0.895	1800
Weighted Average	0.896	0.896	0.895	1800

Table B.16.: Confusion matrix for similar shapes grasped everywhere with point normals on validation data.

	T-Shirt	Jersey
T-Shirt	0.872	0.128
Jersey	0.081	0.919

Table B.17.: Classification performance for similar shapes grasped everywhere without point normals on test data.

	Precision	Recall	F1-score	Support
T-Shirt	0.266	0.293	0.279	900
Jersey	0.214	0.192	0.202	900
Accuracy			0.243	1800
Average	0.24	0.243	0.241	1800
Weighted Average	0.24	0.243	0.241	1800

Table B.18.: Confusion matrix for similar shapes grasped everywhere without point normals on test data.

	T-Shirt	Jersey
T-Shirt	0.293	0.707
Jersey	0.808	0.192

Table B.19.: Classification performance for similar shapes grasped everywhere with point normals on test data.

	Precision	Recall	F1-score	Support
T-Shirt	0.346	0.447	0.390	900
Jersey	0.218	0.154	0.181	900
Accuracy			0.301	1800
Average	0.282	0.301	0.285	1800
Weighted Average	0.282	0.301	0.285	1800

Table B.20.: Confusion matrix for similar shapes grasped everywhere with point normals on test data.

	T-Shirt	Jersey
T-Shirt	0.447	0.553
Jersey	0.846	0.154

Diverse Shapes Baseline

Table B.21.: Classification performance for diverse shapes grasped everywhere without point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.976	0.963	0.969	297
T-Shirt	0.964	0.977	0.970	303
Accuracy			0.97	600
Average	0.97	0.97	0.97	600
Weighted Average	0.97	0.97	0.97	600

Table B.22.: Confusion matrix for diverse shapes grasped everywhere without point normals on validation data.

	Jeans	T-Shirt
Jeans	0.963	0.037
T-Shirt	0.023	0.977

Table B.23.: Classification performance for diverse shapes grasped everywhere with point normals on validation data.

	Precision	Recall	F1-score	Support
Jeans	0.998	0.996	0.997	890
T-Shirt	0.996	0.998	0.997	910
Accuracy			0.997	1800
Average	0.997	0.997	0.997	1800
Weighted Average	0.997	0.997	0.997	1800

Table B.24.: Confusion matrix for diverse shapes grasped everywhere with point normals on validation data.

	Jeans	T-Shirt
Jeans	0.996	0.004
T-Shirt	0.002	0.998

Table B.25.: Classification performance for diverse shapes grasped everywhere without point normals on test data.

	Precision	Recall	F1-score	Support
Jeans	0.976	0.979	0.977	900
T-Shirt	0.979	0.976	0.977	900
Accuracy			0.977	1800
Average	0.977	0.977	0.977	1800
Weighted Average	0.977	0.977	0.977	1800

Table B.26.: Confusion matrix for diverse shapes grasped everywhere without point normals on test data.

	Jeans	T-Shirt
Jeans	0.979	0.021
T-Shirt	0.024	0.976

Table B.27.: Classification performance for diverse shapes grasped everywhere with point normals on test data.

	Precision	Recall	F1-score	Support
Jeans	0.949	0.974	0.962	900
T-Shirt	0.974	0.948	0.961	900
Accuracy			0.961	1800
Average	0.961	0.961	0.961	1800
Weighted Average	0.961	0.961	0.961	1800

Table B.28.: Confusion matrix for diverse shapes grasped everywhere with point normals on test data.

	Jeans	T-Shirt
Jeans	0.974	0.026
T-Shirt	0.052	0.948

B.3. Number of Points

Table B.29.: Classification accuracy over four classes with various numbers of input points during training and testing. Validation data was used as evaluation set.

	Number of Points												
	2	4	8	16	32	64	128	256	512	1024	2048	4096	8192
Without Point Normals	0.3	0.36	0.42	0.51	0.57	0.64	0.71	0.75	0.76	0.78	0.78	0.80	0.79
With Point Normals	0.33	0.39	0.52	0.61	0.71	0.77	0.8	0.86	0.88	0.89	0.9	0.9	0.91

C. Figures

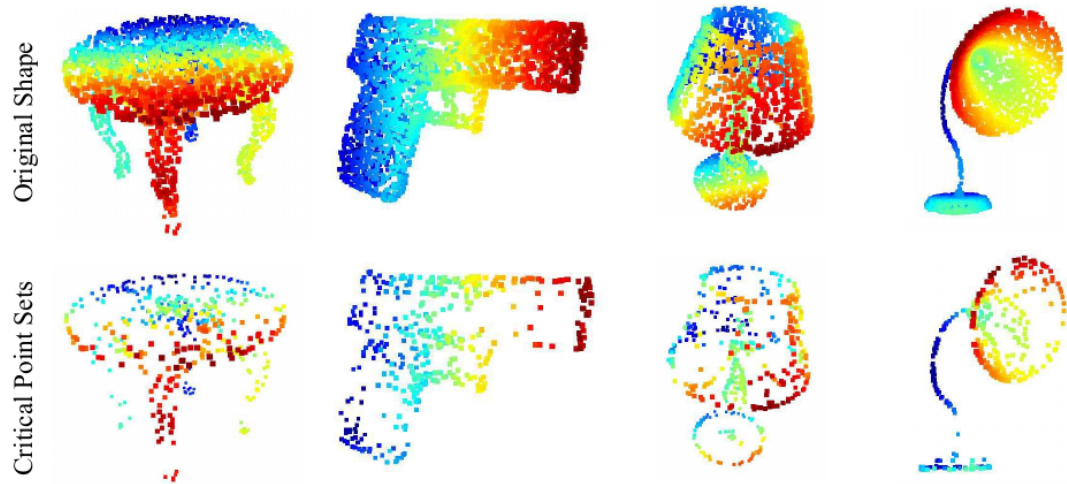


Figure C.1.: Significant points during rigid object classification identified in the original PointNet paper. [QSKG17]

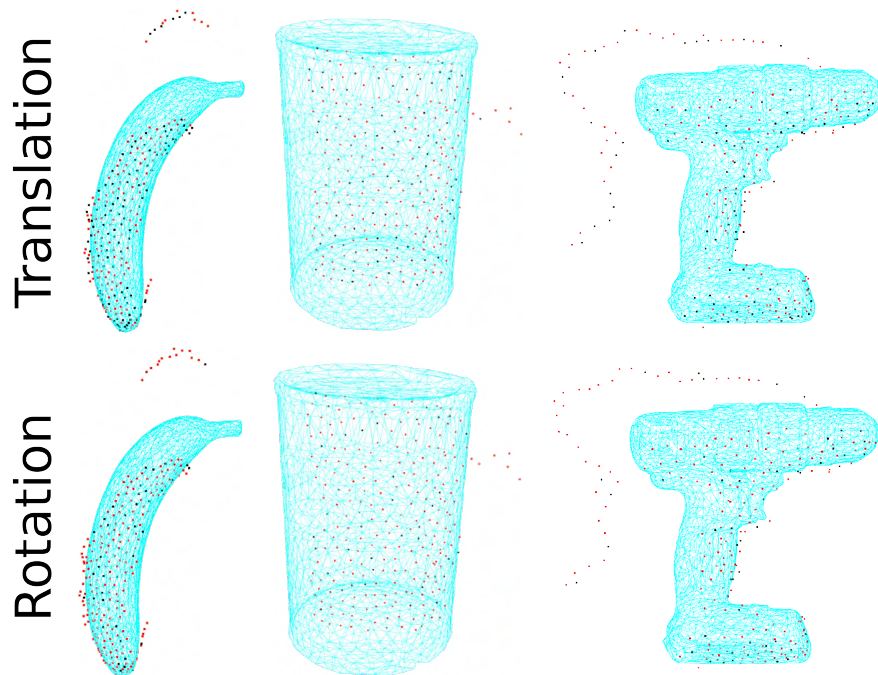


Figure C.2.: Significant points during rigid object orientation estimation identified in the dissertation of Gao. [Gao21]

D. Listings

D.1. ROS Messages

```

1  # This message holds a collection of N-dimensional points, which may
2  # contain additional information such as normals, intensity, etc. The
3  # point data is stored as a binary blob, its layout described by the
4  # contents of the "fields" array.
5
6  # The point cloud data may be organized 2d (image-like) or 1d
7  # (unordered). Point clouds organized as 2d images may be produced by
8  # camera depth sensors such as stereo or time-of-flight.
9
10 # Time of sensor data acquisition, and the coordinate frame ID (for 3d
11 # points).
12 Header header
13
14 # 2D structure of the point cloud. If the cloud is unordered, height is
15 # 1 and width is the length of the point cloud.
16 uint32 height
17 uint32 width
18
19 # Describes the channels and their layout in the binary data blob.
20 PointField[] fields
21
22 bool    is_bigendian # Is this data bigendian?
23 uint32  point_step   # Length of a point in bytes
24 uint32  row_step     # Length of a row in bytes
25 uint8[] data         # Actual point data, size is (row_step*height)
26 bool is_dense       # True if there are no invalid points

```

Listing D.1: The definition of the PointCloud2 message.

Source: [4]

```

1  # This message holds the description of one point entry in the
2  # PointCloud2 message format.
3  uint8 INT8      = 1
4  uint8 UINT8     = 2
5  uint8 INT16     = 3
6  uint8 UINT16    = 4
7  uint8 INT32     = 5
8  uint8 UINT32    = 6
9  uint8 FLOAT32   = 7
10 uint8 FLOAT64   = 8
11
12 string name      # Name of field
13 uint32 offset    # Offset from start of point struct
14 uint8 datatype   # Datatype enumeration, see above
15 uint32 count     # How many elements in the fields

```

Listing D.2: The definition of the PointField message.

Source: [5]

D.2. Live Classifier Configuration

```
1 # topic the pointcloud is published on
2 pointcloud_topic: '/kinect2/qhd/points'
3
4 # name of the classification action
5 action_name: 'classify_action'
6
7 # topic on which the classification result is published
8 result_pub_topic: 'classification_result'
9
10 # whether the camera input is directly classified
11 live_mode: true
12
13 # whether to calculate point normals
14 use_normals: false
15
16 # number of points fed into the network
17 network_input_points: 1024
18
19 # max depth of a pointcloud in z-direction
20 max_depth: 1.0
21
22 # number of classes output by the neural network
23 class_number: 7
24
25 # the class names with their position in the list corresponding to the
26 class_mapping: ['Jeans', 'T-Shirt', 'Sweatpants', 'Sweater', 'Underpants',
27               ', 'Sock', 'Shortpants']
28
29 # the model checkpoint used for classifications
30 model_checkpoint: '2021-06-05_13-37'
31
32 # whether to run PointNet in CPU mode
33 cpu_mode: true
34
35 # whether to visualize the classified pointcloud
36 visualization: false
37
38 # number of pointclouds dropped at beginning of each scanning sequence
39 drop_num: 3
```

Listing D.3: The configuration of the live classifier.

Eidesstattliche Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Masterstudiengang Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel — insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen — benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Hamburg, den 28.08.2021

Niklas Fiedler

Veröffentlichung

Ich stimme der Einstellung der Arbeit in die Bibliothek des Fachbereichs Informatik zu.

Hamburg, den 28.08.2021

Niklas Fiedler