TECHNISCHE UNIVERSITÄT WIEN

ACIN
AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

3D Object Classification: From CAD Models to Reconstruction

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Dr. techn. Jean-Baptiste Weibel
Ao.Univ.Prof. Dipl.-Ing. Dr. techn. Markus Vincze

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by

Rainer Rohrböck
[redacted]

Wien, Februar 2023

Vision for Robotics Group
A-1040 Wien, Gußhausstr. 27, Internet: http://www.acin.tuwien.ac.at

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit gemäß dem Code of Conduct, Regeln zur Sicherung guter wissenschaftlicher Praxis, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, Februar 2023

Preamble

I would like to thank my supervisors Jean-Baptiste Weibel and Professor Vincze for their much appreciated support and great advice. Our participation in ARW21 and ICVS 2021 was particularly inspiring and insightful.

I would also like to extend my appreciation to the creators of the original neural networks MVCNN, VoxNet and PointNet for their significant contributions. Their work has been instrumental in making the research presented in this work possible.

Wien, Februar 2023

Abstract

In many areas of robotics, it is necessary to get a complete and correct understanding of the environment of a robot. Service robots in particular will operate in the same environment for extended period of time. An important part of that understanding is to correctly identify and understand the objects that are present. For this purpose, a scene in which the robot is usually located is scanned and mapped with a 3D scanner. The map can then be broken down into its individual elements either automatically by a segmentation procedure or by a human annotating it. Each object is assigned a class from a predetermined set of classes in order to be correctly processed by the robot.

Nowadays, such classification tasks are done with neural networks, which need a lot of data to be trained and achieve good results. Due to the effort and sometimes also the lack of capabilities, it is often not feasible to train a neural network with data from real scans. Another possibility is to train on computer-aided design (CAD) generated 3D models, which are often available in large numbers from libraries or production data. One problem that arises is the difference that exist between the real scanned data and the CAD data, known as the Sim2Real Gap.

We investigate this Sim2Real Gap in more detail by selecting different neural networks and performing different experiments. The neural networks are selected to cover different categories of input data. These input data are images, voxel grids (volume-based) and point cloud based. The selected algorithms are, in the same order, MVCNN, VoxNet and PointNet and its successor PointNet++. In order to simulate a depth scanner, a reconstruction pipeline is created, which, similar to sensors, compiles the necessary 3D data from generated depth images. Afterwards, this reconstructed 3D data is further modified in different experiments to simulate different defects that occur during 3D scans. These defects are, besides the change due to the reconstruction, random or unknown orientation and size, partial occlusions, limited visibility of objects and erroneous segmentation. Finally, a data set with real scans is evaluated. In all experiments, it was shown that training with reconstructed data can improve performance, in some cases significantly. Furthermore, it was shown that the image-based algorithm, MVCNN, performed best in almost all experiments, as it proved most robust to all types of noise applied.

Kurzzusammenfassung

Bei vielen Bereichen in der Robotik ist es notwendig, ein vollständiges und korrektes Verständnis der Umgebung eines Roboters zu bekommen. Das trifft insbesondere für Service Roboter zu, die in der gleichen Umgebung für einen längeren Zeitraum operieren. Ein wichtiger Teil dieses Verstehens, ist die vorkommenden Objekte korrekt zu identifizieren und verstehen zu können. Dazu wird eine Szene, in der sich der Roboter befindet üblicherweise mit einer Art von 3D-Scanner gescannt und kartographiert. Die Karte kann anschließend in ihre einzelnen Elemente zerlegt werden, entweder durch ein Segmentierungsverfahren oder durch einen Mensch, der sie annotiert. Im Folgenden wird jedes vorkommende Objekt einer Klasse aus einem vorbestimmten Set an Klassen zugeordnet, um korrekt von dem Roboter bearbeitet werden zu können.

Solche Klassifizierungsaufgaben werden heutzutage üblicherweise mit neuronalen Netzen gemacht, die in der Trainingsphase jedoch viel Material benötigen, um gute Ergebnisse zu erzielen. Aufgrund des Aufwands und manchmal auch mangelnder Möglichkeiten ist es oft nicht zielführend ein neuronales Netz mit Daten aus Scans zu trainieren. Eine andere Möglichkeit ist es mit Computer-aided Design (CAD) generierten 3D-Modellen zu trainieren, die oft in Vielzahl aus Bibliotheken oder Produktionsdaten vorhanden sind. Ein Problem, das daraus besteht, sind die generellen Unterschiede zwischen den in echt gescannten Daten und den CAD-Daten, die als Sim2Real Gap bekannt ist.

Um diese Sim2Real Gap näher zu untersuchen werden verschiedene neuronale Netze ausgewählt und damit verschiedene Experimente durchgeführt. Die neuronalen Netze werden so ausgewählt, dass verschiedene Kategorien von Eingangsdaten abgedeckt werden. Diese Eingangsdaten sind bild-, voxel- also Volumens- und Punktwolken-basiert und die dafür ausgewählten Algorithmen sind MVCNN, VoxNet und PointNet sowie dem Nachfolger PointNet++. Um einen Tiefenscanner zu simulieren wird eine eigene Rekonstruktions-Methode ("Reconstruction Pipeline") erstellt, welche ähnlich wie mit Sensoren die notwendigen 3D-Daten aus selbst erstellten Tiefenbildern zusammensetzt. Danach werden diese rekonstruierten 3D Daten in verschiedenen Experimenten weiter verändert um verschiedene Defekte zu simulieren, die bei 3D Scans entstehen. Diese Defekte sind neben der Veränderung durch die Rekonstruktion, zufällige bzw. unbekannte Orientierung und Größe, teilweise Verdeckungen, eingeschränkte Sicht auf Objekte und fehlerhafte Segmentierung. Schlussendlich wird noch ein Datensatz mit echten Scans ausgewertet. Bei allen Experimenten hat sich gezeigt, dass ein Training mit rekonstruierten Daten die Performance teils deutlich verbessern kann. Weiters hat sich gezeigt, dass der bildbasierte Algorithmus, MVCNN, bei fast allen Experimenten am besten funktioniert hat, also am robustesten auf die getesteten Arten von Störungen reagiert.

Contents

List of Figures

List of Tables

1 Introduction

With the development of robotics in recent decades, robots are no longer found exclusively in industrial environments. At home, the tasks of robots are also no longer limited to simple floor cleaning. Instead, as service robots, they are performing increasingly complex household chores.

To ensure safe and efficient operation, a robot must gain a good understanding of its environment before it becomes active, especially in people's homes. Whether it is about deciding which objects can be safely manipulated or identifying obstacles, machine vision is an essential component of the robot to understand its surroundings. Segmentation of the scene and 3D object classification are essential machine vision tasks to obtain semantic information. Later on, other useful information can be extracted as well, like pose estimation, part segmentation, etc. Scene segmentation's goal is to find out which data points belong to which object.

1.1 Challenge

After a scene has been fully captured by a 3D sensor, several tasks need to be executed in order to properly understand the content. The first one is to divide the scene into individual objects or object parts and is referred to as segmentation. Then the 3D object classification can be executed to assign those objects to a certain class as illustrated in Figure 1.1.

The classification can be achieved with a large variety of methods. Classical methods often rely on hand-crafted descriptors such as [2]. They are based on expert knowledge and require more adaptation of class shapes, input data, tweaking of parameters etc. [3]

The ever-increasing computing power and memory of general purpose GPUs or dedicated hardware combined with the creation of very large training datasets enabled the

Figure 1.1: 3D data processing steps: Original scene, segmentation and object classification [1]

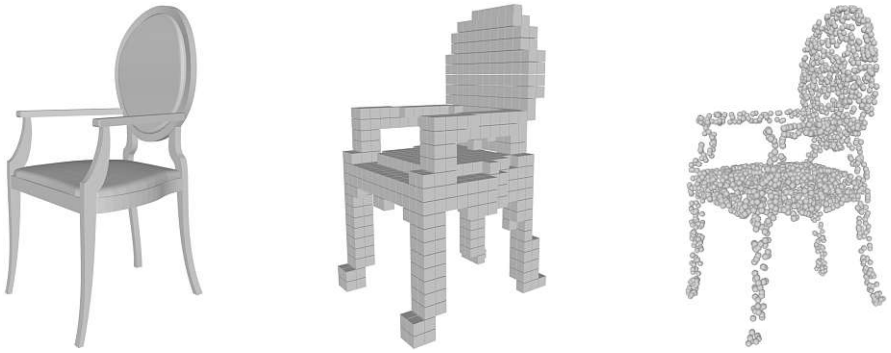


Figure 1.2: 3D data representations: view-, grid and point-based representation

rather old concept of neural networks to experience a revival in the last couple of years. State-of-the-art 3D object classification methods [4]–[8] are now dominated by deep learning-based methods, almost completely replacing classical methods [9].

For the best results with Deep Learning Algorithms, large training datasets are required. This means that for every object that the robot encounters during its operation, the robot must be provided with a large amount of annotated data. The process of manually capturing, segmenting and annotating 3D data for any object or class of objects is very costly. A better alternative is to use generic CAD models to alleviate this issue. They are widely available for all kinds of classes, because of the widespread use of CAD-based design and manufacturing. However, training neural networks with CAD models can lead to performance losses in operation due to the discrepancies between artificial models and recorded data. This is commonly referred to as the Sim2Real gap. Due to the many possible variations of that Sim2Real gap it is not easy to quantify the performance of each classification method. In general, these phenomena are occlusion, smoothed-out surfaces and over- or under-segmentation.

1.2 Contribution

In order to get a better understanding of the Sim2Real gap, its effect on the accuracy within a selection of state-of-the-art classification methods is checked. For each major 3D data representation, a characteristic classification method is selected. 3D data is commonly represented either as point clouds, voxel grids, or a set of rendered pictures/views in classification tasks. An example of these representations is shown in Figure 1.2. PointNet++ [4] and its predecessor PointNet [10] (which are referred to as PN++ and PN) are chosen as representatives for point-based methods, VoxNet [11] for grid-based methods and MVCNN [5], [12] for multi-views methods. These three methods were chosen because of their relatively good performance, availability and representativeness.

The 3D CAD model data set ModelNet [13] is chosen for training and evaluation. The ModelNet40 variant is widely used for evaluation and comparison by state-of-the-art 3D classification methods. It contains a sufficient amount of data with 12.311 artificial CAD

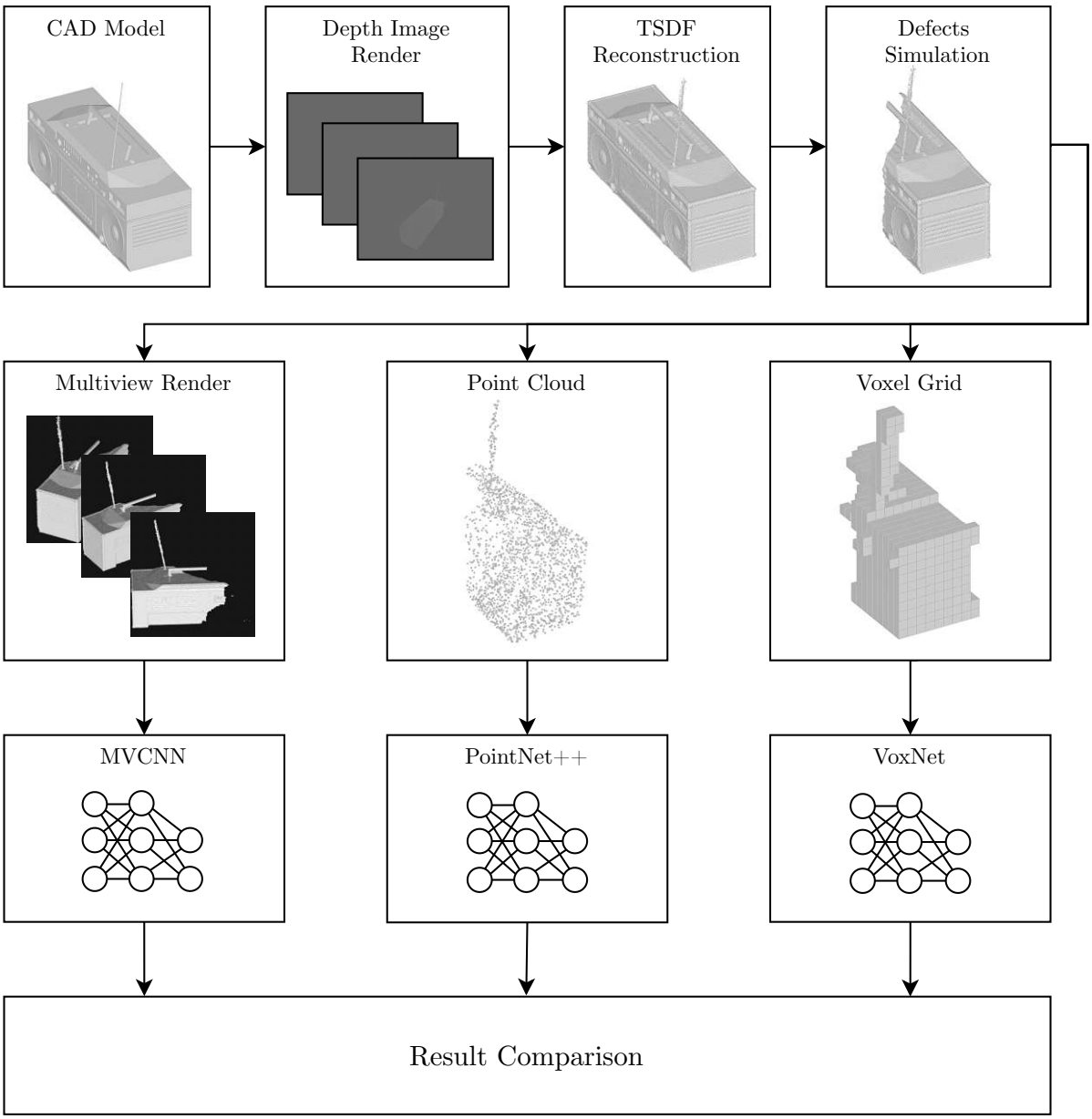


Figure 1.3: Procedure of an experiment, where artificial defects are introduced to a 3D model. It is then evaluated by three different kind of neural networks to investigate the behaviours. As an example a CAD model of a radio is shown with 50% of its surface removed.

models in 40 classes for the studies. The complete models of the dataset also enable the simulation of specific aspects of the Sim2Real gap to separately evaluate the impact of specific design elements on each deep learning model. In particular, the impact of the reconstruction of object models from a set of rendered depth maps, but also the impact of rotation, scale and occlusions on data representations is investigated. The steps of this experiments including the evaluation with each model are outlined in Figure 1.3. First a CAD model is rendered in depth images similar to the output of a sensor. This depth images are then combined and a mesh is recreated again with a Truncated Signed Distance Function (TSDF) procedure. Then the before mentioned defects are applied to simulate some challenges of the Sim2Real gap. The defective mesh is then converted to the individual input data type of the networks to be evaluated and compared.

As a result of all of these experiments, the weaknesses and strengths of each method are identified for different noise scenarios. This gives an overview of how well each type of classification algorithm performs but also which type is best suited for certain problem cases in the context of the Sim2Real gap. The results suggest that the multi-view render-based model MVCNN works best for most problems. However, there are also weaknesses that may not occur in the nearly equally well-performing point-based model PointNet++. In a further step, the performance on a data set with captured data ScanObjectNN [14] is evaluated as well. This test partially confirms the results from the previous experiments, although the overall performance is worse with the real recorded data. This is mainly due to the fact that in the experiments different phenomena are partly considered in isolation. In general, however, it can be shown that some further development is still required for state-of-the-art models. At least until an eventual final goal is reached to train with CAD data to evaluate measured data like from the ScanNet dataset. In addition, it could be shown that the performance of the CAD training could be further improved by a prior modification of the CAD training data. This is summarized in a set of guidelines to help improve the performance with real-world data.

Part of the results presented in this thesis were previously published in the following papers:

- "Analysis of 3D shape representations in presence of corrupted data" [15]
- "Measuring the Sim2Real Gap in 3D Object Classification for Different 3D Data Representation" [16]

1.3 Thesis outline

In the following chapters all the performed experiment are explained in detail. Chapter 2 gives a literature background on all the used methods. In chapter 3, the effects of the Sim2Real gap are further investigated. In the subsequent chapter 4, the experimental setups and results are then described in detail. Finally, chapter 5 gives a conclusion of all the results. Appendix A contains all the detailed results in tables and graphs.

2 Related Work

This chapter gives a brief overview of 3D data representations and existing work on the classification of 3D objects.

2.1 3D data representation

There are several ways in which 3D information can be stored and processed. Depending on the application and the origin of the data, one of them is selected. A lossless conversion between the 3D data representations is usually not possible.

2.1.1 Point Cloud

One of the most commonly used formats is point clouds. A point cloud is a set of points of a vector space that has a disorganized spatial structure. A point is usually described by a set of Cartesian coordinates. In some cases, additional information such as color or the surface normal for the respective points is also stored. Point clouds can be generated by 3D scanners using different methods. One example of this is RGB-D cameras which record a color image with additional depth information simultaneously, such as the Microsoft Kinect.

The point clouds can then be directly extracted from this depth information. For each pixel,the 3D position is calculated by an inverse projection using the camera's intrinsic parameters and position. Since the depth image is only taken from a single viewpoint, parts of the object or scene that are obscured for the camera are not included in the point cloud. To overcome this problem, multiple depth images can be captured from different viewpoints. Merging this depth information is a well-known problem for which several solutions exist.

Since the space represented is not bounded and there is no limit to the density and thus the number of points, the amount of data for high-resolution scans can be arbitrarily high. Manipulations such as resampling can limit the number to a practical amount. A popular toolset for processing point clouds is the so-called Point Cloud Library [17]. When processing point clouds, it must be taken into account that the position of the coordinate origin and the orientation of the object or scene can in principle be arbitrary, and can also differ from point cloud to point cloud.

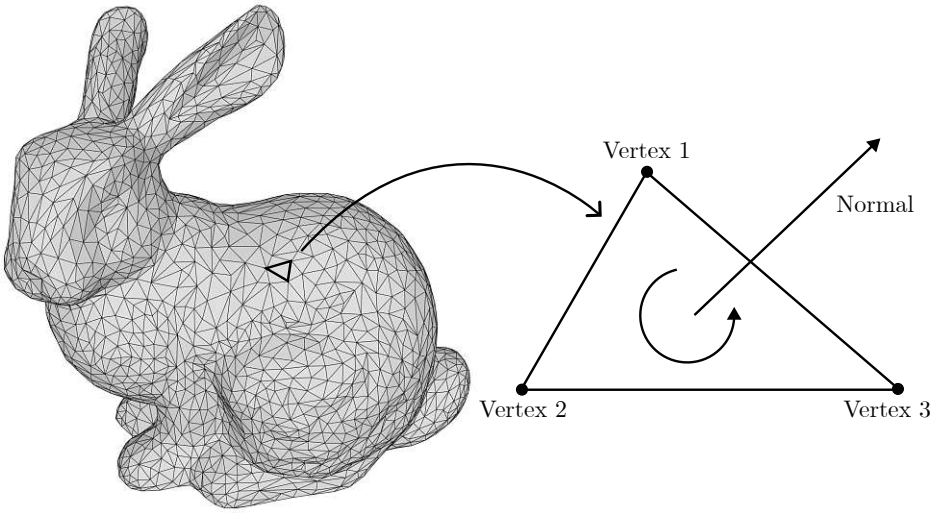


Figure 2.1: A mesh consists of multiple faces which each contain 3 vertices. The surface normal can be derived from the order of the vertices.

2.1.2 Mesh

Another prominent 3D data concept is the so-called mesh. In contrast to a point cloud, the data is presented in a more structured way. There are vertices, which represent spatial points similar to the point cloud, and faces, which form elements of the surface from these points. The faces are described by a set of vertices, whereas the vertices are usually only referenced by indices. The number of indices per face is in principle arbitrary, but triangles, i.e. faces with 3 indices, have become standard for simpler and uniform processing, especially for 3D graphics hardware. The order of the vertices in the face usually also determines the direction of the surface normals (see figure 2.1).

Meshes can be modeled directly by hand using 3D graphics software or alternatively exported from existing mechanical 3D CAD drawings. The latter has greatly contributed to the compilation of extensive data sets with meshes. Such data not created by sensor measurements is also referred to as synthetic data. Due to the large number of programs that can be used to create and edit meshes with different functionality and scope, there are also a large number (30+) of different file formats that are mostly not compatible with each other.

Meshes can also be generated from the point clouds or depth images acquired from scans using various algorithms such as Truncated Signed Distance Function (TSDF) to merge multiple scans from different viewpoints or the ball-pivoting algorithm [18] that can create a mesh out of a point cloud. The procedure with TSDF is explained in more detail in section 4.1.2. Ball-Pivoting Algorithm works by moving a virtual ball of a user-defined size across the space of the point cloud. If the ball touches exactly three points, a triangle is formed and saved. The ball is then moved across the edge of the triangle to find more connected triangles. This is repeated until all edges are tried and all points of the point cloud are examined. Conversely, a point cloud can be created

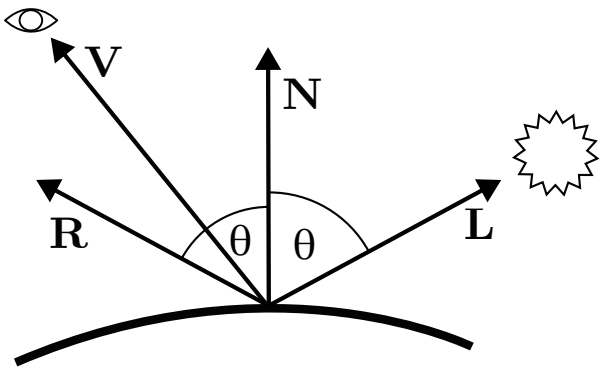


Figure 2.2: Phong reflection model vectors: surface normal **N**, direction to observer **V**, direction to light **L** and light reflection direction **R** [21]

quite easily from a mesh. For this purpose, only the vertices have to be stored as points. It should be noted, however, that, especially with synthetic meshes, the vertices can have large variations in density. Fixed density point clouds and meshes can be obtained e.g. by uniformly sampling the points over the surface. Such fixed density point clouds are then more similar to the result of a measurement.

Similar to point clouds, direct processing of non-regular and variable sized data is more complicated for neural networks. Standard neural network architectures usually rely on a regular data schema like images or similar, like spectrograms in the audio domain. However, there are also approaches to process mesh data directly with a neural network like MeshCNN [19]. In order to still take advantage of the advanced development in the field of image recognition, meshes can also be mapped to one or more 2D images.

When mapping to 2D images, the intrinsic and extrinsic camera parameters as well as the rendering must be taken into account. In the rendering phase, the 3D information is projected onto a 2D surface. In order to give the object a different appearance depending on the light, and give a visual sense of depth, shading is used to approximate the local behavior of light on the object's surface. Here, the light source used and the reflection model applied have a significant influence. An established and illustrative method for calculating reflection is the so-called Phong reflection model [20]. With this method, the intensity of the illumination is summed by three components: ambient, diffuse and specular.

One example with all three components is shown in figure 2.3.

The intensity is calculated from the original intensity of the light I_{in}, the ambient light intensity I_{ambient}, the material constants $k_{\text{ambient}}, k_{\text{diffuse}}, k_{\text{specular}}, n$ and the vectors $\mathbf{L}, \mathbf{N}, \mathbf{R}, \mathbf{V}$.

$$I_{\text{out}} = I_{\text{ambient}} + I_{\text{diffuse}} + I_{\text{specular}} \tag{2.1}$$

$$I_{\text{ambient}} = I_{\text{a}} \cdot k_{\text{ambient}} \tag{2.2}$$

$$I_{\text{diffuse}} = I_{\text{in}} \cdot k_{\text{diffuse}} \left(\mathbf{L} \cdot \mathbf{N}\right) \tag{2.3}$$

$$I_{\text{specular}} = I_{\text{in}} \cdot k_{\text{specular}} \left(\mathbf{R} \cdot \mathbf{V}\right)^{n} \tag{2.4}$$

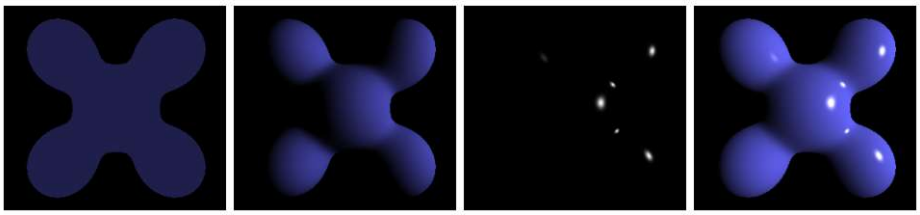


Figure 2.3: Example of Phong reflection components. For illustration purposes both the ambient and diffuse light components are colored blue [22]

with

$$k_{\text{diffuse}} + k_{\text{specular}} \leq 1, k_{\text{ambient}} \leq 1 \tag{2.5}$$

In this case, **L** is the incident vector of the light, **N** is the surface normal vector, **V** is the direction of the observer, and **R** is the direction of the light's reflection, calculated from **L** and **N** (see figure 2.2). All vectors are normalized.

This method can also be used to calculate several light sources simultaneously. For this purpose, the intensity of each light source is calculated individually and summed up. For different types of light sources such as point lights and parallel directional lights (sun), only the incidence vector **L** changes for the illumination calculation.

2.1.3 Voxel grid

In addition to point clouds and meshes, there is a third type of representation called voxel grids, which presents the data in a strict grid structure. A voxel can be represented by a grid point in a three-dimensional grid. The word voxel is a portmanteau of volume and pixel. In a dense voxel grid, usually the position of each voxel is not stored explicitly, but it is inferred from the save position in the data structure containing the grid. Depending on the type of grid, different categories of information can be stored for each voxel. To reduce data storage for large grids, it's also possible to compress the data by using a sparse grid representation using various methods.

A popular type of voxel grids are occupancy grids, which also have different subtypes, which are described in [11]. Often a so-called binary grid is used, where a Boolean value is used to indicate whether the voxel is occupied or free or if it belongs to an object or to the background. However, there are other forms which can provide more information per-voxel depending on the needs. For example, with limited perspective in 3D scans, a voxel grid can distinguish whether the voxel is free, occupied or only invisible to the recording system. The density grid is based on ray tracing calculation. The value of the voxel corresponds to the probability that the voxel would block the sensor ray. Thus, object parts that do not occupy a complete voxel in terms of volume generate a smaller density value.

Unlike point cloud or mesh, an object in a voxel grid cannot have an arbitrarily large size. Therefore, before converting to a voxel grid, the object must be scaled, positioned, and optionally aligned to fit the voxel grid. In contrast to point clouds or meshes, voxel

grids are very sensitive to changes in rotation and scaling. This means, for example, that point clouds rotated by a few degrees can result in completely different voxel grids after conversion. The higher the resolution of the voxel grid, the smaller this effect.

Due to the three-dimensional expansion, the memory needed increases cubically with the grid resolution. However, it is also possible to reduce this significantly with sparse data structures, such as octrees, reducing the memory footprint but making queries more complicated.

2.2 Datasets

In addition to the ever-improving hardware, the many available ever-growing datasets have also contributed greatly to the advancement of machine vision methods in recent decades. A dataset generally refers to a collection of data that is collected or managed for a specific purpose. For machine learning, a dataset is needed for unsupervised and supervised learning. For the latter, there needs to be separate training and test sets from one or two different data sources. As an essential statistical property, a dataset should satisfy the independent and identically distributed (i.d.d) assumptions [23]. This means that all examples in the dataset are independent of each other. Additionally, training and test sets should be identically distributed. Due to the considerable effort to create or assemble datasets and also to verify these properties, a limited number of datasets are used as standards in a given field. Results can therefore easily be compared. With popular datasets, the results are sometimes also tracked and benchmarked in global hosted challenges.

The datasets for 3D data available for scientific purposes fall into different categories depending on their intended use. The most popular ones at the moment are (Human) Pose Estimation, 3D Semantic Segmentation (implicitly including 3D object classification), Object Detection, 3D Reconstruction, Autonomous Driving, etc. as listed on [24]. For this work with a robotic background, indoor datasets are of particular interest. The available datasets can be distinguished between two different types. On the one hand, there are those based on (synthetic) man-made CAD data like ShapeNet [25] and ModelNet [13]. While ModelNet has fixed categories, ShapeNet uses a hierarchy based on the WordNet lexical database. Smaller subsets have been spun off from ShapeNet, such as ShapeNetCore, ShapeNetSem, ShapeNet Parts or PartNet with a smaller number of clearly delimited categories, which, like ModelNet40, have undergone manual annotation verification. ModelNet contains 127.915 objects in 662 categories. The subset ModelNet40 has 12.311 objects.

In contrast, there are datasets consisting of 3D scans like ScanNet [26], Matterport3D [27], Stanford 3D Indoor Scene Dataset (S3DIS) [28] and the derived ScanObjectNN [14]. Most of these scans were captured as video using RGB-D sensors and include entire scenes with many typical indoor objects such as furniture. A mesh of the entire scene was then generated offline from the videos using a TSDF-based method with additional camera pose estimation such as [29]. This mesh was then segmented and annotated manually or with auxiliary tools. The annotation here represents a much

larger effort than the actual acquisition (500 to 20 workers for ScanNet). There are also technical differences in the annotation between the data sets. For example, for ScanNet, voxel-based annotation was used, while for S3DIS, additionally generated Point Clouds were annotated. The entire ScanObjectNN dataset consists of 15 classes with a total of 2902 unique objects. These objects were extracted from 700 unique scenes from SceneNN and ScanNet.

In practice, datasets are split into two or three non-overlapping and independent subsets. A training set, a test set and an optional validation set. The validation set is used for evaluating models when tuning hyperparameters and data preparation. The test set is used for rating the performance of a model after training is complete and can be used to compare different models with each other. Some recommend a 70%, 20% and 10% split [30] or a 80%, 20% split without validation part [31].

2.3 3D object classification

Statistical classification of data can be described as determining which category a data point, observation, or variable should be placed in. The algorithm that performs a classification process is called a classifier. This is usually done on the basis of a training dataset that contains observations (or instances) whose association to a category is known. This type of learning is referred to as supervised or inductive learning and is also the best studied approach.

Fundamentally, a distinction is made between binary and multi-class problems, i.e. two or more classes which usually have different solutions. Most problems always deal with a finite number of classes [32]. A common approach to classification problems in computer vision is to extract certain features of objects. These features are basically a set of quantifiable properties and are used for training via machine learning in order to create a classifier. Machine learning is a subfield of computer science that studies methods trained to make predictions or perform behaviors based on data or, in particular, relationships within that data. The predictions are made based on recognized patterns in the data that are extracted during the training phase, and, for a subset of the algorithms, are continuously updated to make increasingly better predictions. Popular machine learning methods used in the context of classification are linear classifiers, quadratic classifiers, support vector machines (SVMs), decision trees and neural networks.

In the context of this work, 3D object classification is discussed, which is the process of assigning a semantic label to 3D data. This chunk of 3D data is expected to contain only one single object. This means that for real scanned data, each object has to be isolated as well as possible with segmentation first. The term "object recognition" is also sometimes used in the same context as classification, but it can also describe a broader context. Object classification is to be distinguished from the discipline of object detection, which describes procedures that not only identify objects but also localize them.

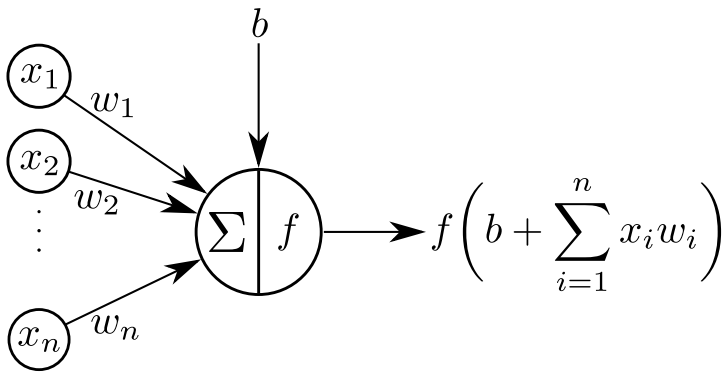


Figure 2.4: single perceptron with inputs (x_i), bias b, weights w_i and an activation function f

2.3.1 Deep Learning

One important subfield of machine learning is the study of artificial neural networks (ANNs), which are inspired by the neurons of a human brain. The very basic element of an ANN is a node, perceptron or neuron. Figure 2.4 shows a typical single node which has a number of inputs (x_i), a bias value (b), connection weights (w_i), an activation function (f) and an output. For the output of a neuron, the sum of all the input values multiplied by the weights and the bias is calculated. The sum is then passed to a certain activation function and the output is generated. The activation function can be any linear or nonlinear function. The most common ones are:

- Rectified Linear Units (ReLU) [33]: $f(z) = max(0, z)$
- Tanh: $f(z) = tanh(z)$
- Sigmoid activation: $f(z) = \frac{1}{1+e^{-z}}$

If many of such nodes are connected and structured in multiple layers an entire neural network can be constructed. Figure 2.5 shows the separation of these layers into one input layer, multiple hidden layers and one output layer. In the hidden layers, all nodes are connected to all nodes in the previous layer, which makes them so called fully connected (FC) layers.

For computer vision, another special architecture has proven to be very effective. With high-dimensional inputs such as raw images or other visual data, ANNs using only fully connected layers quickly reach their limits, because of the high number of nodes and weights. When using images as input, the natural two-dimensional arrays have to be transformed to one-dimension, which destroys the two-dimensional locality of images. The same applies for images with multiple (color-) channels or for fixed 3D data such as voxel grids.

Convolutional neural networks (CNNs) have one or multiple convolutional layers to overcome these problems. Instead of a connection to all previous inputs or previous nodes, only a certain number of local nodes are used for the calculation. The structure of

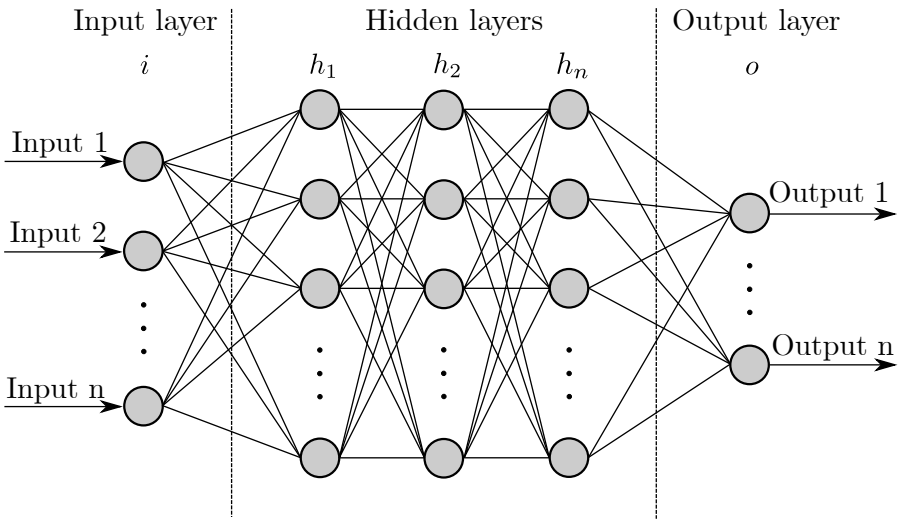


Figure 2.5: Structure example of a network with n full connected layers

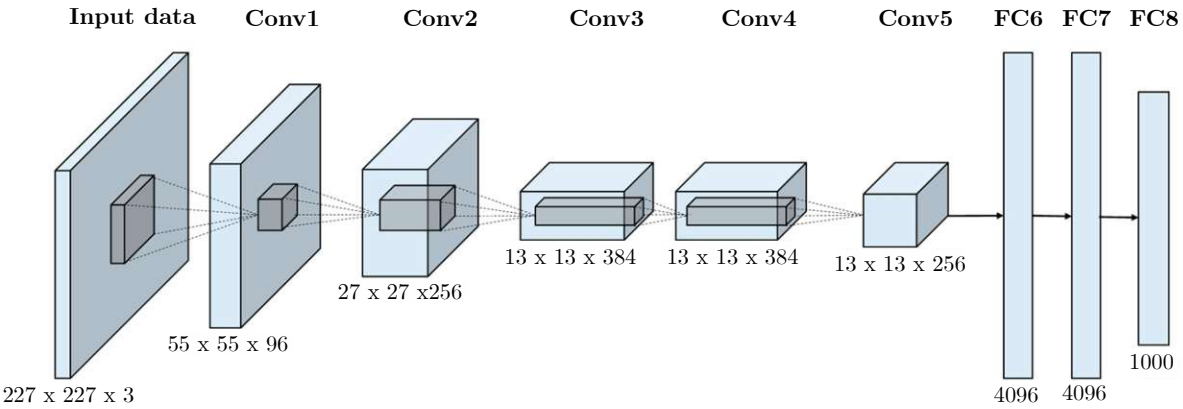


Figure 2.6: AlexNet architecture with the dimension of each layer [36]. It consists of 5 convolutional (Conv) and 3 fully connected (FC) layers.

this assignment acts similar to a learnable filter or kernel which extracts useful features for the rest of the network. After the convolutional layer, a pooling layer is often used. In these layers, the data of the previous layer are again aggregated multidimensionally and locally. The output is generated by a simple function such as the maximum or average of all the input data. This allows the dimension for the next layer to be reduced. A convolutional layer is defined via kernel size (e.g. 3x3 for 2D), stride and padding.

The first popular convolutional neural network was AlexNet [34], which uses 5 convolution layers for feature extraction and 3 fully connected layers for classification. Figure 2.6 shows all the layers with their respective node sizes. With LeNet-5 [35], there was actually the first published use of CNN for handwritten character recognition as early as 1998, but CNNs did not find wider use until the success of AlexNet in the ImageNet 2015 contest.

For training a neural network, the definition of a loss function is necessary. The loss function result indicates how well or poorly the network performs. One simple example of a loss function is a mean square error function such as

$$L(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.6}$$

where $\hat{\mathbf{y}}$ is the desired and $\mathbf{y}$ is the actual output of our network of all the training examples n. At the start of the training, the weights are either set to a random value or to a defined value, e.g. for pre-trained networks. In order to reduce the value of the loss function, the weights have to be adjusted in the correct way. One popular method is to calculate or approximate the gradient of the loss function for the weights. The corresponding algorithm is known as gradient descent or stochastic gradient descent if only a subset of the data is considered. With the gradient function, the weights can be iterative calculated with a defined learning rate (l_r), which is one of the hyperparameters.

$$w_j = \hat{w}_j - l_r \frac{\partial L}{\partial w_j} \tag{2.7}$$

Because of the complex network structure, the calculation of the gradient is not done analytically but rather with a separate method called backpropagation. Instead of using a stochastic gradient descent with a constant learning rate, more sophisticated optimizer methods with adaptive learning rate are used. One of the optimizers named Adam [37] has turned out to be a de-facto standard due to its properties and is utilized for many networks. The name is derived from the phrase "adaptive moments" and it is often considered robust and therefore easy to handle.

2.3.2 View-based representation

The idea behind view-based methods is to make 2D image recognition techniques usable for tasks in the 3D domain. Being a very popular research area on its own, 2D image recognition is an established subject with a large number of well performing methods and algorithms. Some well-known tasks include recognizing certain things like faces, handwriting or objects e.g. in traffic or in the medical domain.

In order to make 2D image recognition usable for 3D object classification, the object must be mapped as completely as possible. This can be done by using multiple views from different angles. The images of those different angles can then be individually evaluated and the result can be combined, for example, with average or by taking the maximum output value. However, as shown in [12] a more sophisticated solution is to combine those results inside the network with certain layers. For this network Multi-View-CNN (MVCNN) the outputs for each individual image are connected with a max-pooling layer followed by a few fully connected layers. This makes the network able to learn how to combine the results in the best way possible. MVCNN's work has also been updated in [5], which is the version referred to as MVCNN furthermore in this thesis.

For rendering images, the author of MVCNN uses the Phong reflection model with perspective projection. However, it is stated that the network is rather invariant to

illumination. For a complete mapping, the author uses either 12 views around the z-axis or 20 views around the object in all directions. In order to avoid the assumption of an always up-right orientation of the meshes, each view is again rotated 90° four times to get a total of 80 views. However, since the rendered meshes are still oriented the same, this approach is still only able to evaluate a limited number of orientations. Yet, from the experiment results, it can be seen that the difference between 12 and 80 views is not very significant (<1% for classification accuracy).

The image classification network used is from the Oxford Visual Geometry Group (VGG), which was first introduced at the ILSVRC challenge in 2014 [38]. The first used variant, VGG-M, is similar to AlexNET as it also contains 5 convolutional and 3 fully connected layers with additional dropout and softmax layers. In the updated version VGG-11 [39], a slightly bigger variation with 8 convolutional and 3 fully connected layers could improve the result by a few percent. Subsequent experiments with even more complex networks from the ResNet family (ResNet 18, ResNet 34, ResNet50), however, could not enhance the performance significantly further.

The image classification networks are also pre-trained with ImageNet images. This means that this part of the network has already been trained with a generic data set with 1000 categories. It gives the network the ability to generate classifiers for generic features found in most of the images. During the actual training of the whole network, which is often called fine-tuning, the weights already start with certain initial values. A general term for this strategy is Cross-Modal Transfer Learning. A comparison without using this pre-training shows a drop in performance of a few percent.

The original paper also describes another application called sketch recognition, i.e. the categorization of simple hand drawings. For this application, the models were rendered differently in line drawing style. For further improvement, dataset augmentation and jittering on the rotation of the input images is used to improve the accuracy.

In later works, the structure of MVCNN was further refined. One of such is GVCNN [40] which improves the view combination with an extra group-based pooling, while 3DSeqViews [41] takes advantage of the sequence in which views are observed to improve the classification results. Another direction is to use different projections such as panorama [42] to cover more of the object in a single view. These variations are not considered in this work, because the basic properties that are important for the experiments still remain the same.

2.3.3 Grid-based representation

A different approach is utilizing 3D convolutional neural layers instead of 2D ones in combination with 3D grids. In contrast to the view-based methods, the existing 3D data can directly be used as input. The use of voxel grids is investigated in 3D ShapeNets [43] and VoxNet [11]. Yet, with the additional dimension, the exponential growth in the number of parameters and memory has been a limiting factor in the accuracy of early models. Further works overcome this limit like OctNet [44] by using an octree-based learning architecture or binVoxNet [45] by using binary representations. More recent improvements to the VoxNet approach have been made by using sparse representations

like in [46]. In this work, the basic VoxNet [11] is chosen as a representative due to its simplicity and to better show the strengths and weaknesses of the grid-based representation.

Similar to MVCNN, with VoxNet the input data Z-axis is assumed to be aligned with the gravity axis. To allow for rotation, the input data is augmented by several twisted poses. This means that each model is included in the data set with several different poses. In the evaluation, all poses are also evaluated and the maximum score is taken. Only by augmenting training data and taking it into account in the evaluation could a per-class accuracy of 83% be achieved for ModelNet40. Without this, the performance for this dataset was significantly worse at 61%.

The original paper also compares different types of occupancy grids. Here, however, no clear difference has been shown. For the experiments, data from very different sources were used. Besides the ModelNet40 dataset with CAD models, also datasets from LiDAR data (LiDAR data Sydney Urban Objects [47]) and a dataset from RGBD data (NYUv2 [48]) were considered.

2.3.4 Point-based representation

The third kind of approach is using point-clouds without intermediate representation. With the input being a subset of 3D points lying on the surface of an object, point clouds are closer to the output of depth sensors than the other two representations. The challenge of using point clouds is handling the specific properties of this input data. In general, point clouds are unordered. That means that, unlike pixel arrays in images or voxel arrays, the data is not inherently sorted. Another important aspect of point clouds is that a large part of the information content is contained in the interaction between points. Since no structural or neighbor information is available, the information is contained in the distance metric of the points. So the network has to be able to retrieve this structural information on its own. Finally, the points must also be able to be modified by transformations such as translation, rotation and scaling without this having an influence on the information content. This means that all points undergo the same transformation while preserving the relational euclidean distances between the points. The network must therefore not consider the absolute position of individual points in any way.

With PointNet [10], all these distinctive characteristics are dealt with. In order to get independent of the order of the input data, symmetric functions are used to combine the information from all the points. For this case, the max-pooling function is used in the initial layers. In the paper, other options that utilize permutation or sorting of the input data are investigated and do not provide as good results. The problem with a possible unknown transformation is handled with an input dependent matrix. The values of this transformation matrix are also calculated using a separate neural network with several layers, which are also trained in the training phase. The neural network of the matrix also uses the point cloud as input. This means, depending on the point cloud, the transformation matrix adjusts itself to achieve a transformation as independently as possible. In addition to classification, the network is also able to do segmentation tasks

with an adaptation of the architecture on the output side.

A further improvement was made with PointNet++ [4] by using hierarchy layers to make it more robust to scale and other variations. Other approaches rely entirely on neighborhood information instead of coordinates information. Some like SpiderCNN [49] and PointCNN [7] even go so far as to adapt conventional operators which have been used for regular domains such as images to irregular ones. Other approaches like KPConv use convolution weights that are located in Euclidean space by kernel points. Only input points close to them are considered with a distance function. Another algorithm with extended grouping is investigated in DGCNN [6] where, instead of individual points, local geometric structures are used by constructing a local neighborhood graph. For this work, PointNet and PointNet++ are chosen as representatives, because of their popularity. In general, point-based methods are also probably the most researched of these 3 representations at the time of writing.

Table 2.1 gives an overview of some size metrics of all the networks such as the layer counts and calculation effort. The total layer count includes all layers and activating functions including Convolution, Fully Connected, MaxPool, ReLU, Dropout, BatchNorm Layers and calculation. The parameter number is the total number of weights and biases of all the layers. MAC is short for Multiply-accumulate operation which is a fundamental procedure for most of the calculations in neural networks.

	Convolution Layers	**Fully Connected Layers**	**Total Layers**	**Parameters (M)**	**MACs (G)**
MVCNN	8	3	26	128.93	89.94
VoxNET	2	2	11	0.92	15.27
PointNet	9	7	23	3.47	0.45
PointNet++	27	6	49	1.48	0.87

Table 2.1: Comparison of network size parameters. Parameters are written in Mega (10^6) and Multiply-accumulate operations (MACs) are written in Giga (10^9).

2.3.5 Sim2Real Gap

The idea of Sim2Real is to transfer observations acquired through simulations to real world systems. These can cover different areas in computer vision, like Automated Driving [50], Robot Ego-Pose Estimation [51] or 3D Object Classification [52].

The motivation is that the use of simulation data is usually cheaper and safer. In addition, there are experiments that can not be performed at all or much more easily in the simulation domain. An example are 3D objects, which are more difficult to obtain in reality, but can be easily produced with CAD data. In the final application, however, these rare objects should be reliably recognized just like the others. To close the Sim2Real gap, this must first be properly understood, i.e. how systems react differently to simulation and the real world. For object classification in literature [52] there is rather a qualitative estimation of what these differences, often called noise, look like.

The sources of error can essentially be divided between the acquisition and the post-processing. On the one hand, the sensor causes errors due to sensor noise. Another important point, however, is the usually limited viewing angle. In order to obtain a complete object representation, it must be scanned from all sides. This is either simply not possible because the object is not accessible from all sides or not practical because it does not correspond to the function of the system in the situation. In such cases, an incomplete model must therefore be expected. With the dataset of ScanObjectNN, it has been shown that with common indoor scans, the objects of the scenes often contain only 50% of their surface [14].

During post-processing there are errors with under- or over-segmentation, whereby parts of other objects are wrongly assigned or parts are missing because they are assigned to other objects. The superfluous or missing data further complicates detection. When converting the sensor data to the input data of the network, there may also be distortions of the surfaces, alteration of details or artifacts, which can be seen with TSDF in section 4.1.2.

2.4 Performance metrics

For comparing the performance of classification algorithms, different metrics can be used.

The most basic and most widely used one is accuracy. Accuracy is the number of correct predictions divided by the total number of predictions. For a multi-class problem statement like in this work, an accuracy value can be evaluated for every class individually. For this, the number of correct predictions for this class is divided only by the number (predicted and actual) of that class. The accuracy values per class can be combined in two ways. Either the average of all values can be calculated, which is the Mean Class Accuracy. Otherwise, all correct predictions of all classes can be summed up and divided by the number of all predictions. The latter is more often called Overall Accuracy. If the number of objects in all classes is equal, these two accuracy values are equal.

One popular concept of visualizing classification results is using confusion matrices. For these matrices, one axis represents the predicted class and the other the actual class.

Figure 2.7 shows an example with MVCNN and ScanObjectNN results. Here accuracy values are used, which means that each value is normalized by the total number of actual class entries. This means that each column sums up to 1. It is also sometimes practical to show the actual amount of each prediction or to normalize it by predicted classes, which makes it essentially recall values.

With the confusion matrix, problematic classes can be identified conveniently. In the example it shows that almost every shelf is correctly identified as such, but many other objects are identified as shelf as well, which includes most of the desks.

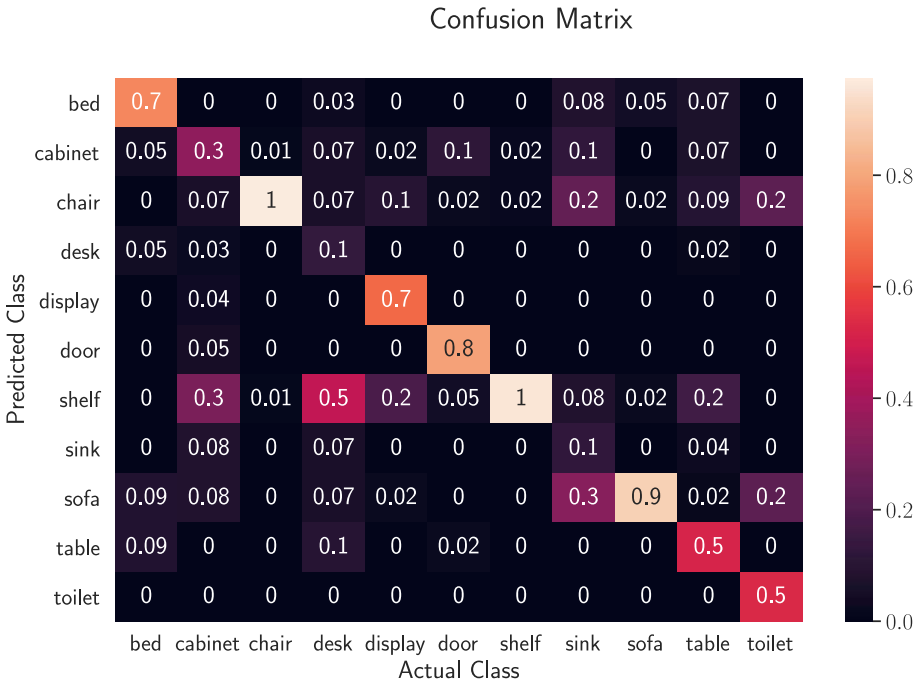


Figure 2.7: Example of a confusion matrix for 11 classes of ScanObjectNN

3 Understanding the Sim2Real gap causes

For a better understanding of the Sim2Real gap, a few experiments similar to [14] will be performed. In this paper, objects are extracted from real-world scans and the performance of common point-based networks is compared with that dataset. The networks were trained using either objects from the ModelNet40 dataset or from the same dataset. In addition, the same scan dataset was also tested with artificially induced undersegmentation. This means that background parts were also present with the objects. To test this in a consistent way, the objects were separated from the scene with an axis-aligned bounding box.

Although only point-based networks were compared, there were clear performance differences and the authors were also able to create a new, optimized network. So, based on the findings, a new, partly better network could be created.

In general, however, the tests showed that the absolute performance of all tested networks is rather poor when training with ModelNet40 and testing the ScanObjectNN data set. Here the overall accuracy was between 49.3% and 30.9%. PointNet, also considered in this work, was about in the upper third with 42.3%. Compared to most of the results in this work, however, it must be clarified here that this data set consists of only 11 classes. This new data set was formed from the common classes of ModelNet40 and the 15 classes of ScanObjectNN. A further explanation with own experiments is described in section 4.4.

A challenge in this test with real data is, on the one hand, the limited data and, on the other hand, that all phenomena of the Sim2Real Gap occur simultaneously. For the ScanObjectNN mentioned above, a large number of scans were used. A total of 700 indoor scenes were selected from the SceneNN and ScanNet mesh datasets. From these, 2902 objects could be manually selected from 15 categories. Despite the large effort, this is little compared to ModelNet40 and makes it difficult to find many matching classes. With the attempts with ScanObjectNN, only the behavior of incorrect segmentation could be examined better in detail. The other lack of objects cannot be regarded or only with difficulty separately from each other. There are, on the one hand, incompleteness due to occlusion, errors of reconstruction or oversegmentation. On the other hand, there are other phenomena due to scan or reconstruction, like low frequency noise on the surface. One way to do this with the existing data would be to sort the objects manually according to how pronounced the phenomena are. However, since this is not a practical approach and it would also reduce the data set very much, the different phenomena are reproduced based on the generic data.

3.1 Data representations

In the following, we investigate how the different classification algorithms might behave for the previously mentioned problems.

3.1.1 Effects on rendered images

For view-based algorithms, the object is first imaged in two dimensions and this 2D image is then evaluated. In advanced algorithms such as MVCNN, several images from different viewpoints are also combined during the evaluation. For different occlusion effects, this leads to a higher probability of an image where the object is possibly almost complete. Depending on the scanning method, there may be more or less noise on smooth surfaces. These imperfections can have a negative influence on the result when it is picked up by the renderer. For example, even though in MVCNN the silhouette is likely to have a greater influence than the surface itself as explored in the later paper [5], the reconstruction also changes the silhouette. Thus, for straight objects, the silhouette is also changed upon distortion of the surface, at least for certain viewing angles. Another problem is caused by undersegmentation. Similar to missing parts, certain viewing angles can be unchanged here. However, depending on the characteristics, the leftover parts are often visible from several viewing angles. E.g. remnants of the floor, which appear on almost all images, because of the 45° viewing angle of the renderer.

3.1.2 Effects on point cloud

In contrast to the view-based methods, the point-based methods use much less information from the actual object. For example, in the ScanObjectNN dataset, the objects have 50000 - 500000 vertices. For the point-based methods, only a random fraction or a fraction determined by sampling methods is used. Usually, these are 1000 to 2000 points, depending on the setting. In contrast to view-based, the behavior is more difficult to estimate due to internal complexity. In general, nonetheless, it can be assumed that surface distortions have less effect, since the surfaces are resolved less precisely anyway due to the small number of points. Based on the neighborhood hierarchy, it can be assumed that the methods are less robust against under- and over-segmentation effects.

3.1.3 Effects on voxel grids

Similar to the point-based methods, grid-based also significantly reduces the information of the input data. The meshes often have only a fraction of the volume of the bounding boxes and thus, depending on the resolution of the voxel grid, many data points are combined into a single voxel. As a result, surface imperfections are also less heavily transferred to the input data and are therefore not picked up by the network. This also applies to smaller holes in the objects that can occur during segmentation or due to occlusion. E.g. with the binary occupation network, the corresponding voxel is still marked as occupied by the surrounding faces. In the case of under-segmentation, however,

this leads to the opposite effect, that even small remnants lead to an unintentionally occupied voxel when captured.

3.2 Occlusions

When manually viewing scanned datasets such as ScanObject, it is noticeable that almost all objects have been captured incompletely. This incompleteness generally happens due to occlusions during scanning. Basically, two types of causes of occlusions can be distinguished. On the one hand, the scanners often have only a limited range of motion. In addition, objects are often located on or near walls. This results in large-scale occlusions that often affect an entire side or sometimes even two entire sides of an object. In real scenarios, this often depends on the object class. There are objects that are rather freestanding and accessible from all sides, like office chairs, dining tables, doors and objects that are almost always placed against a wall like sofas, TVs, beds or even structurally connected to walls like bathtubs and toilets. A view from below, which is often present in CAD data, is always missing in scans. The other cause is objects in the room which are obscuring others. These obscured areas are usually smaller and do not cover an entire side of an object. Also, for this, there are tendencies in the classes of real scenarios e.g. office chairs which are often in front of desks and they obscure each other.

In the following, experiments are presented to simulate the described occlusion phenomena in an analytical way for the Sim2Real Gap. The goal of these experiments is to modify CAD data as if an occlusion happens. For the restricted viewing angle, the CAD data is recorded at restricted camera angles and then reconstructed. To get a consistent result, the camera angles are arranged in a circle around the object. The camera positions are at a regular distance of exactly 50% the height of the object. The orientation points exactly at the object and the distance is set so that the object is always completely captured by the cameras. A total of 12 camera perspectives are to be used for the experiment. To investigate the different characteristics of the occlusion behavior, different subsets of the cameras are used for the reconstruction.

Simulating occlusion by other objects can also be done using camera reconstruction. Here a virtual object, for example a cube, can be placed in front of the object, which partially obscures the object. During reconstruction, the cube is removed and a hole remains. This method works in principle, but has the major disadvantage that the occlusion varies greatly in size depending on the object shape and thus class. For this reason, the reconstruction is not suitable for this type of experiment. Therefore, another method is used that directly modifies the mesh data. This is done by randomly selecting a vertex in the object and removing all adjacent faces. Then, other adjacent faces are randomly removed until a desired percentage of the surface has been removed. Thus, a hole is generated in the object and it grows until it reaches a well-defined share of the total surface area. In Figure 3.1 this is visualized. This method has the advantage that it is highly controllable and therefore has the same effect on all classes of objects. There is, however, the disadvantage that the choice of a random starting point may lead to unrealistic or obscuring effects that are unlikely to occur with real objects. Nonetheless,

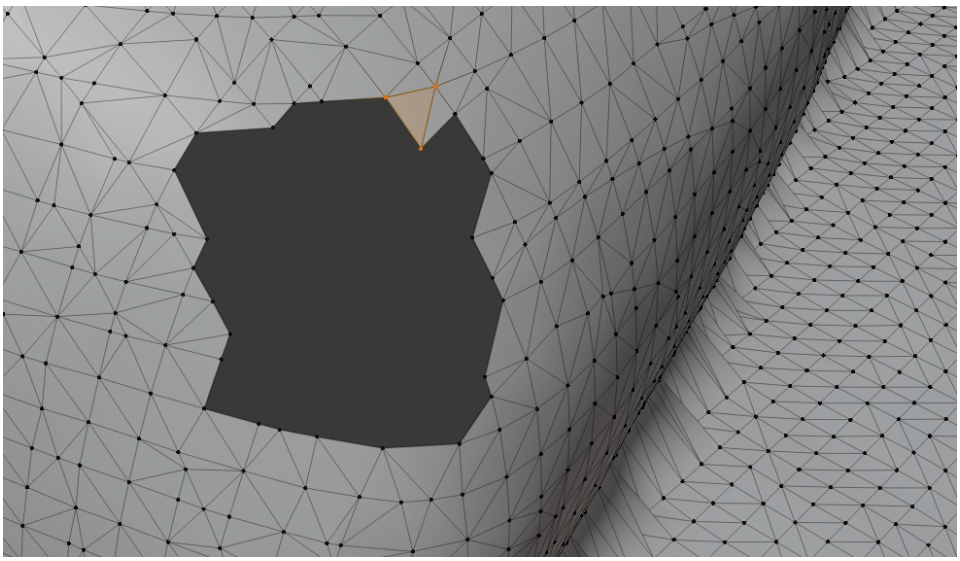


Figure 3.1: Artificial hole grown on the object surface. It's further extended by removing a random adjacent face (orange).

this cannot be corrected without a deeper understanding of the object.

3.3 Segmentation errors

When processing a scanned scene, it is broken down into its individual components. During this segmentation, defects can appear just like during the scan. If the objects are not separated correctly, the object will either become larger or smaller due to the unrelated parts. Both have different effects on the classification performance. A common error can occur when separating the floor to which many indoor objects are directly connected.

Semantic errors can also happen, so that e.g. the pillows are not assigned to the bed, although it is handled that way in a matching CAD dataset. Or small objects that are on a table or shelf and are not recognized as independent objects by the segmentation algorithm.

There are several possibilities to simulate the misbehavior of the segmentation. Over-segmentation often cannot be distinguished from occlusion, so it makes no sense to conduct separate experiments for it. In contrast, under-segmentation is much more difficult to simulate. One possibility might be to assemble a complete realistic scene with CAD models and apply a segmentation algorithm. However, a good and consistent way without human intervention may only be possible through complicated procedures with AI. A straightforward way is to consider only the segmentation of the floor, as this is also a very common error. This can be easily done by attaching the floor as a surface to the object at the bottom with a size depending on the object's dimensions.

4 Experiments

4.1 Experimental setup

In this section, first the used datasets are introduced, then creation of the experiment input data is explained and finally the setup of each network, MVCNN, VoxNet and PointNet/PointNet++, is presented.

4.1.1 Datasets

For the experiments, two 3D object datasets, ModelNet40 and ScanObjectNN, were used.

The original division of the objects into training and test split is used in this thesis. Because the number of available objects is not the same between the classes, the dataset is not balanced. This means that it makes a difference in the accuracy values whether the overall accuracy or the mean class accuracy is considered, which is explained in section 2.4. Table A.1 lists all classes of ModelNet40 with the object count for training and test.

To compare the results between ModelNet40 and ScanObjectNN dataset, an intersection of the common classes of both datasets was created. The newly formed datasets with 11 classes are named ModelNet11 and ScanObjectNN11. Table 4.1 shows how the classes correspond to each other.

4.1.2 Reconstruction Pipeline

In order to generate evaluation data from the ModelNet dataset that is closer to scanned real-world data, a custom reconstruction pipeline was created. In the first step, it is necessary to scale the models uniformly. Although real scanned data is often scaled uniformly by the scanning and reconstruction process, this cannot be assumed for man-made objects. Especially if the data originates from different libraries, there will be different scaling. This means that with man-made objects or objects from different scanning databases it is necessary to scale them uniformly before further processing. Each object was centred based on the bounding box and then resized so that the object fits exactly into a unit sphere around the object.

The object is then rendered from different camera angles in depth images. The depth images have a resolution of 1 mm. To get a comprehensive image, 26 camera positions were selected. Since the objects should still be as complete as possible in this step, a sufficiently large number of camera positions are chosen. Especially for concave-shaped objects like bowls, it is difficult to capture the object completely with only a few camera angles. These cameras are arranged in two circles on the top and bottom half of the

Class #	ScanobjectNN Class	ModelNet Classes	# training MN11	# test MN11	# test SO11
1	bed	bed	515	100	22
2	cabinet	dresser, wardrobe	287	106	75
3	chair	bench, chair, stool	1152	140	78
4	desk	desk	200	86	30
5	display	monitor	465	100	42
6	door	door	109	20	42
7	shelf	bookshelf	572	100	49
8	sink	sink	128	20	24
9	sofa	sofa	680	100	42
10	table	table	392	100	54
11	toilet	toilet	344	100	17

Table 4.1: New datasets ModelNet11 (MN11) and ScanObjectNN11 (SO11) as intersection between ScanobjectNN and ModelNet40 with object count

model. In addition, there is a top and a bottom view. In Figure 4.1 camera positions are illustrated.

A 3D model is then created from the depth images, just as is possible with a depth camera. For the purpose of reconstruction, a truncated signed distance function (TSDF) [53][54] based algorithm is used which is provided by [55].

A signed distance function is a mathematical term which is the distance from a given point to the boundary of a set such as a shape. With one of the conventions, the sign is a positive value for all points inside the set and decreases with proximity to the boundary. The result is zero when the point is exactly on the boundary and negative on the outside of the set. In addition, if the values are clamped between 1 and -1 it is a truncated distance function.

As a basis, the TSDF algorithm uses a volume where each voxel contains a numerical value. The whole process is also referred to as RGB-D integration. In this volume, the depth images are projected and the distance function is calculated. Each pixel in the depth image is projected through the volume with a ray cast. Equation 4.1 shows the calculation of the TSDF. For each voxel with position $\mathbf{x}$ hit by the ray, the distance to the camera is calculated (d_{cam}) and the depth value of the pixel (d_{depth}) is subtracted. Then the value is scaled by the truncation distance (d_{TD}). An increased truncation distance helps smooth out noise from the depth images and sensor position, but thin structures might not be detected correctly. The resulting value is then clamped between -1 and 1. For each voxel hit by the ray, the TSDF values of the voxel grid D_t and the weights W_t are recalculated. The weighting function $w_t(\mathbf{x})$ can be set as a function depending on the voxel camera distance. Ideally, the weighting function should match the sensor noise model. In [56] some functions are evaluated. In the implementation of Open3D the weighting function is set to $w_t = 1$.

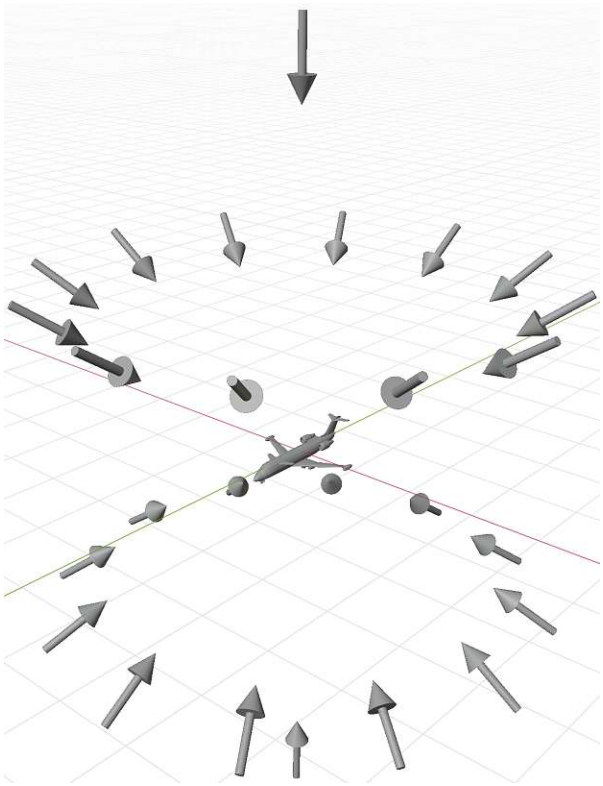
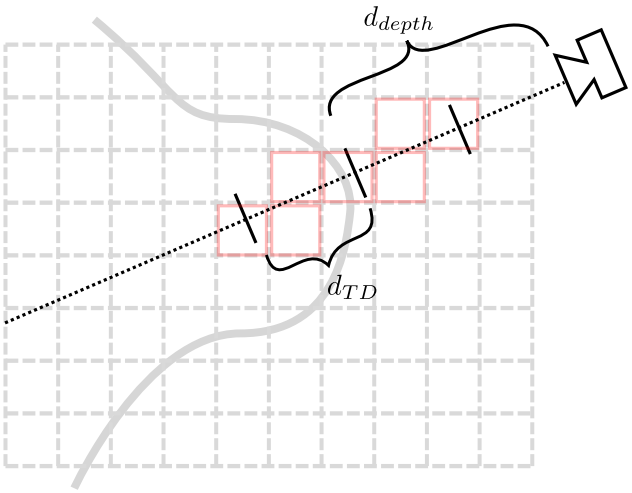


Figure 4.1: Placement of 24 depth cameras around an object for a nearly complete TSDF reconstruction



(a)

0.7	0.4	0.1	-0.3	-0.4	-0.5	-0.7	-0.9	-1	-1
1	0.6	0.4	0.2	0.0	-0.2	-0.3	-0.6	-1	-1
1	1	0.7	0.5	0.4	0.3	0	-0.4	-0.9	-1
1	1	1	0.9	0.9	0.5	0	-0.4	-0.8	-1
1	1	0.7	0.5	0.5	0.3	-0.1	-0.5	-0.9	-1
1	0.7	0.4	0.1	0.0	-0.1	-0.3	-0.7	-1	-1
0.7	0.4	0.1	-0.3	-0.4	-0.5	-0.7	-1	-1	-1
0.6	0.2	-0.2	-0.6	-0.8	-0.9	-1	-1	-1	-1

(b)

Figure 4.2: Example of TSDF for an object in 2D. (a) shows the raycast and distances for a single depth value with the truncation distance d_{TD} and the stored depth value of the pixel d_{depth}. All red marked voxel are being evaluated for that depth value. (b) shows the final grid values from which the zero crossings are determined.

$$d_{sdf}(\mathbf{x}) = \frac{d_{cam} - d_{depth}(\mathbf{x})}{d_{TD}} \tag{4.1}$$

$$d_{tsdf}(\mathbf{x}) = \min(-1, \max(1, d_{sdf}(\mathbf{x}))) \tag{4.2}$$

$$D_{t+1}(\mathbf{x}) = \frac{W_t \cdot D_t + w_t(\mathbf{x}) \cdot d_{tsdf}(\mathbf{x})}{W_t(\mathbf{x}) + w_{t+1}(\mathbf{x})} \tag{4.3}$$

$$W_{t+1}(\mathbf{x}) = W_t(\mathbf{x}) + w_{t+1}(\mathbf{x}) \tag{4.4}$$

Figure 4.2 illustrates the calculation of the distance between the camera and the depth measurement. For simplification reasons, only a slice of the voxel grid is shown. For performance reasons, usually only voxels inside truncation distance d_{TD} are updated (marked red) at each step. After the calculation is done for each voxel and each depth map, the surface of the object is determined by the zero crossing of the voxel values.

Finally, a marching cube algorithm [57] is then used to create a correct mesh from the voxel grid. With this algorithm, for each 8 neighboring voxels a small mesh piece is added based on whether they are occupied or not. For 8 voxels, a total of $2^8 = 256$ combinations are possible. When ignoring all rotational and mirrored duplicates, there are only 15 unique combinations. Figure 4.3 shows these 15 combinations. Following this principle, the entire voxel grid is traversed and the mesh is successively assembled. Finally, the mesh is cleaned up by removing duplicated vertices.

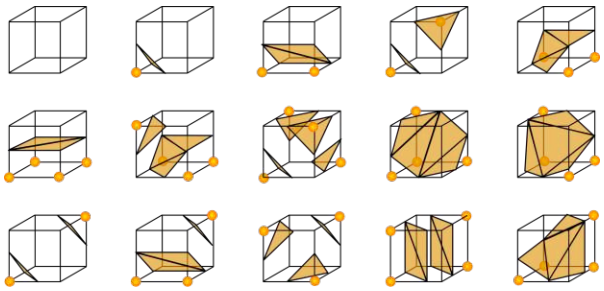


Figure 4.3: Marching cube: Conversion to mesh parts based on the occupation of each voxel. Occupied voxel are displayed with orange dots. [58]

In order to achieve a similarity with real applications, the same resolution and image format is chosen for the depth images as for the original Microsoft Kinect sensor, which is 640x480 with 16 bits.

In visual comparison to CAD models, the reconstructed ones show smoothed out object edges. At the same time, planar surfaces become less smooth. Small details such as doorknobs are either simplified or missing completely. Very thin structures like doors can also be problematic when the truncation distance value is set too large, or when the CAD design is using unreasonable fine structures.

There are also considerable differences in the mesh structure, as the vertices are now more equally distributed evenly on the surfaces. Also, the total count of the vertices and faces is much higher. A side-by-side comparison is shown in Figure 4.4.

In real recordings, the depth data is also overlaid with sensor noise. This noise was measured and investigated, for example, in [59] for the Microsoft Kinect sensor. In general, a lateral and an axial noise component can be expected, which have approximately a Gaussian distribution. These components also depend on the distance, whereby the standard deviation for the noise in the axial direction increases quadratically with the distance. For this work, artificial noise directly in the depth data was omitted, as in the range we work in the noise model, influence was mostly removed by the integration done over all the images.

4.1.3 Neural Network Setup

A total of four neural networks are used for the following experiments. They are parametrized with standard values according to their publications. If those parameters are fine-tuned, the accuracy values could be increased even further, but the target is to get a better understanding of the characteristics of degradation of each method, instead of comparing the absolute maximum values.

MVCNN

For MVCNN an implementation with PyTorch [60] is used, which is provided by the original author of [5]. As described in the original paper [12] the quality of the input dataset plays a significant role in the performance of MVCNN in terms of image properties

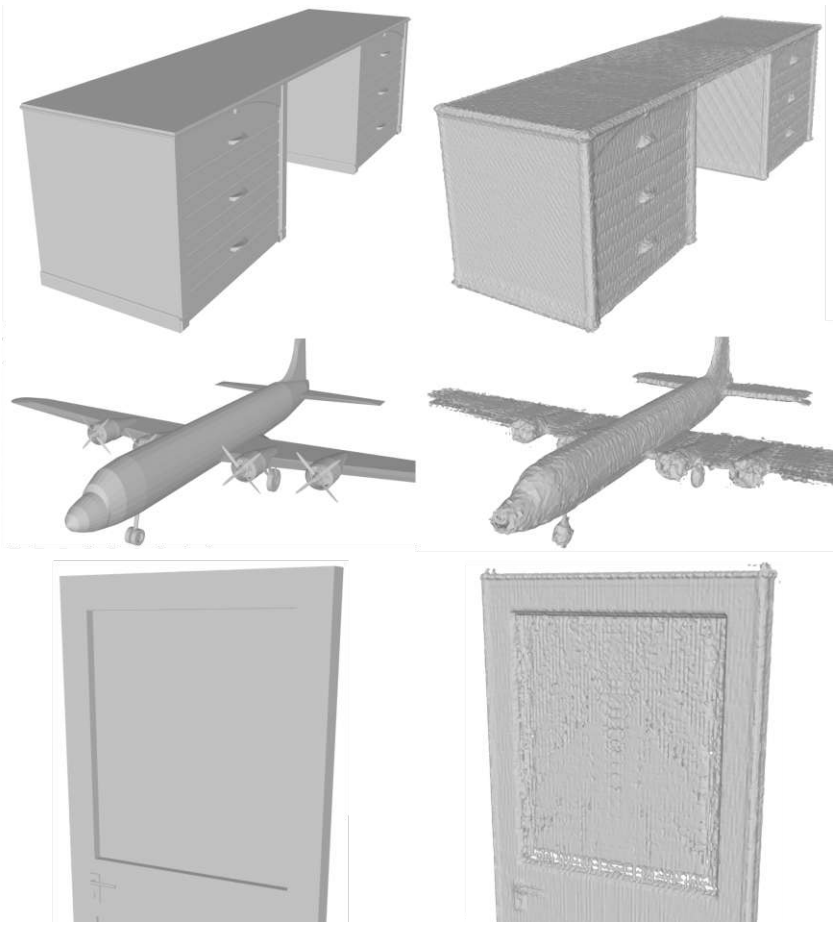


Figure 4.4: CAD Model (left) and TSDF reconstruction (right)

Figure 4.5: Camera placement for the MVCNN dataset generation [12]

and view coverage. In order to generate the dataset from the ModelNet40 objects, a blender script was used. This script as well as an already full rendered ModelNet40 dataset is used, which is also provided by the author [61].

Like in the original paper and the subsequent publication [5], 12 views per object are used for the experiments. These views are taken by 12 virtual cameras that are arranged in a circle around the object. As in the original, these cameras are also arranged above the ground plane at an angle of approx. 30° to the ground plane as shown in figure 4.5. Since the models of ModelNet40 are always aligned with the upper axis, an additional transformation is not necessary. The distance from the object to the camera is always adjusted so that the image is completely filled by the object. The generated images have a resolution of 224x224 and are rendered with the Phong reflection model with a parallel light source from a fixed position. The phong reflection model means that the final rendered color of each pixel is the addition of an ambient, diffuse and specular highlight component. The used image is gray scale and saved with RGBA8 encoding and a uniform background color.

In order to test the MVCNN implementation, the dataset with the rendered images was recreated with the original ModelNet data and a Blender script provided by the author. When training and evaluating with the recreated dataset, the 94.21 % overall could not be achieved as with the original dataset also provided by the author. The deviations are shown in Table 4.2.

Dataset	**Overall Accuracy**	**Mean Class Accuracy**
Original rendered	94.21%	90.59%
Script rendered	91.53%	89.38%

Table 4.2: Comparison of training and evaluation with original dataset provided by the author and self-rendered dataset with blender

A further analysis with a per-pixel comparison of the render images shows that the camera is rotated around its roll axis by a few degrees. It is also noticeable that this angle difference is not constant across all classes. By training with the original dataset and modified datasets with various angles, it can also be noticed that the network with the original dataset is sensitive to changes in angle, where the datasets with a constant angle are not. If the class and angle matched during the evaluation, a significantly higher

accuracy could be achieved for this class than if only the class matched. In addition, the differences between similar classes are artificially increased by changing the angle. A concrete example is 'desk' and 'table' with an angle difference of approx. 0.5°. With the original dataset files, both classes achieve 96% and 95% accuracy compared to 82% and 76% with the script rendering. If the entire test dataset is now rendered with either those two angles, an apparent increase can be noticed if the angles match. Table 4.3 shows the change in class accuracy when evaluation is done with datasets at different angles.

Object rotation	**desk**	**table**
0.8°	95.3%	82%
1.31°	40.6%	93%

Table 4.3: Class accuracy when training with original dataset and evaluation with datasets rendered with different angles

In general, it can be stated that the trained model with the original dataset reacts much more sensitively to changes in angle. A test with several evaluation data sets at different angles shows a clear fluctuation in accuracy (0° 71.2 %; 0.8° 90.1 % and 5°80.1 %). If, on the other hand, the network has only been trained with a constant angle, the accuracy remains almost constant at 91.5% even when evaluated with different angles.

It can be concluded from this that MVCNN can react very sensitively to inconsistencies in the dataset with overfitting, even if these are not noticeable during a visual inspection or seem unimportant.

The datasets for MVCNN are created with Blender version 2.79b and the original provided script for all experiments. As for the training settings, vgg11 was used as CNN and the learning rate was set to $5 \cdot 10^{-5}$ as recommended by the author. The batch size was kept at 32 with a total of 30 epochs.

VoxNet

For VoxNet, the implementation which is provided by the original author uses outdated neural network frameworks. That is why for this work an alternative implementation is used which also utilizes PyTorch. It gets similar performance results and the neural network structure matches the one described in the paper. As suggested by the paper author, a 32x32x32 binary occupancy grid is chosen as input format. A bigger grid size has not shown any significant increase in accuracy. For the preparation of the dataset, the voxel grids are created with an universal rasterization tool called binvox [62]. The binary occupancy grid works in a way that every voxel is considered occupied as soon as some part of any face is inside its volume. Since the models in ModelNet40 are of different sizes, they are uniformly scaled according to their bounding sphere before conversion. In contrast to a bounding box, the bounding sphere has the advantage that it can be rotated without having to be rescaled to fit into the same voxel grid. The provided Pytorch implementation is modified to be able to handle different rotations. As suggested in the original paper [11], each ModelNet40 object was trained with 12 copies

in 30°off rotation poses around the vertical axis. Conversely, during the evaluation, the voting takes place over these 12 rotation poses, with the pose with the greatest activation winning. In the paper, this is referred to as "Training Augmentation" and "Test Voting". Both are necessary to reach the published 83% accuracy with ModelNet40 which is assumed to be the mean class accuracy. However, the accuracy drop without both is not that significant as described in the original paper with the used implementation (-22% vs. -4.8%, absolute difference). For training, the learning rate is set to 10^{-4}, the batch size to 128 and a few hundred epochs are used.

PointNet/PointNet++

For PointNet and PointNet++, implementations by the original author are used.

When generating the input data, the models are also scaled according to their bounding sphere with zero mean. Then random 1024 vertices with their absolute coordinates are selected from each mesh. With CAD data the vertices are generally not evenly distributed. More detailed parts, for example, a door handle contains substantially more vertices than the other parts of the door, while having a significantly smaller share of the total surface. Depending on the design artist, exporter or scanner, the vertex distribution can also be very unreasonable in some places. In order to get the best information content needed for classification a vertices are evenly resampled on the surface of the object before selecting the 1024.

For the experiments, the standard configuration with the three-level hierarchical network with three fully connected layers was selected. No additional normals were used as input.

4.2 Impact of TSDF reconstruction

To better understand the difference between artificial and real data, the entire ModelNet40 dataset [13] was reconstructed using the pipeline described in the previous chapter. As a comparison, each of the classification methods was trained and evaluated with the original and the TSDF reconstructed dataset. The results are shown in table 4.4. For this experiment, the original parameters were taken by the authors as best as possible. A detailed description of the individual parameters is given in section 4.1.3

Train	Test	MVCNN	VoxNet	PN++
CAD	CAD	89.4	82.9	88.0
CAD	TSDF	83.4 (-6.0%)	82.2 (-0.7%)	85.6 (-2.4%)
TSDF	TSDF	88.5	83.1	87.1
TSDF	CAD	85.8 (-2.7%)	80.8 (-2.3%)	85.0 (-2.1%)

Table 4.4: Impact of the TSDF reconstruction of object models, mean per class accuracy is reported in percent with the absolute percentage decrease in parenthesis

In general, it can be seen that all CAD trained networks have quite a loss when evaluated with TSDF and vice versa. For MVCNN, this is most likely due to the different surface structure. The small imperfections make a considerable difference to the surfaces of rendered images, but also on the outline of the renders. For VoxNet, these imperfections have almost no influence at all, because during the conversion to the voxelgrid this information is lost and has no influence on the classification result. All 3 networks have a small drop in training and evaluation with TSDF data of a few percent. It is possible that with TSDF data it is more difficult for the networks to detect subtle differences between classes that may be unintentionally present in the CAD data.

4.2.1 Impact of random rotation

In the original ModelNet40 dataset, all objects have a correct orientation on the global vertical axis (model Z-axis). Objects with a specific preferred direction (e.g. aircraft) are aligned to one of the other axes, although there is no clear consistency even within classes. The vertical axis is also often known in real-world scenarios, as it can be determined relatively easily via an accelerometer, for example. The orientation of an object around the global vertical axis cannot be easily determined by sensors or a simple evaluation by software.

To determine the sensitivity of each classification method to this rotation, each model in the dataset was rotated by a random angle. To obtain a consistent result, each model was evaluated at 3 different angles. The result is summarised in table 4.5. The reconstructed TSDF dataset was used as the basis for training and evaluation.

Method	Overall accuracy	Mean class accuracy
MVCNN	91.5 (-0.0%)	88.8 (+0.3%)
VoxNet	85.9 (-0.6%)	82.5 (-0.6%)
PointNet	88.0 (-0.4%)	85.0 (+0.1%)
PointNet++	89.1 (-0.3%)	86.6 (-0.5%)

Table 4.5: Impact of random rotation, absolute percentage decrease in parenthesis. The models are trained and tested with the TSDF reconstructed dataset

As can be seen, none of the methods shows a significant difference. This can be attributed to the respective techniques of the methods to deal with rotations. For MVCNN, the angle dependency is automatically compensated by the dataset of 12 images with different rotation poses per object. According to the experiment, the error margin in the rotation angle left at 30° seems to be too small to show any difference.

Since the grid-based format is very angle-dependent, similar to the view-based format, a similar principle was used for VoxNet. In VoxNet, the training data set is enriched with different angles in order to obtain a rotation dependency. This means that each object is present several times in the training data set with different angles. As suggested in the original paper, a number of 12 angles were selected here. During the evaluation,

each angle is also evaluated separately and the angle with the best score is used for the classification via vote.

According to the author, PointNet and PointNet++ show a small rotational dependency. In order to achieve a certain degree of rotational independence, the models are randomly rotated with a uniform distribution during training.

A rotation around one of the other axes is not dealt with here, as there are too many possibilities due to three degrees of freedom to get a reasonably meaningful result. For various scenarios, however, such a rotation could certainly occur, e.g. a service robot which wants to classify a fallen bottle.

4.3 Impact of segmentation and occlusion errors

In contrast to real recorded data, CAD models cannot be viewed from every angle or they are obscured by other objects. Another problem that can be observed with real data is incorrect segmentation after further processing of the data. Here it can happen that either parts of the objects are completely missing because they are assigned to another object or foreign parts of another object are assigned to the target object. This problem is often described as over-segmentation. Since faulty segmentation and occlusion often have almost the same effects, no further distinction is made between them in the following experiments.

4.3.1 Random holes

One way to simulate such occlusions as consistently as possible across all classes is to generate artificially growing holes in the objects. Although this is not quite the most realistic approach, it gives the most control over occlusions and allows a fair comparison across all classes. In the method used, a random face is selected by the model and the nearest faces are removed in a random manner until a certain percentage of the total surface area of the model is reached. If no nearby faces are left and the criterion is not yet reached, the result is discarded and the process is restarted for the model. This ensures that there is always only 1 hole per model with the correct surface ratio.

This experiment also starts with the TSDF reconstructed models and then evaluates the deformed models with all classification algorithms. Figure 4.6 shows the mean class accuracy for different occlusion levels.

As can be seen, all 3 methods have a similar downward trend. For small occlusion levels (5% and 10% occluded), VoxNet shows almost no degradation. Due to the comparably large voxel size, almost no changes are made to the input here. For the Occupancy Grid, which VoxNet uses, the voxel is still considered to be occupied unchanged, even if some vertices have been removed.

For medium sized occlusions (of up to 20%), PointNet++ shows the smallest loss. MVCNN overall shows the best performance even for the larger holes. This can most likely be attributed to the fact that of the 12 views, even with 50% missing surface, some still have a recognizable and distinguishable shape.

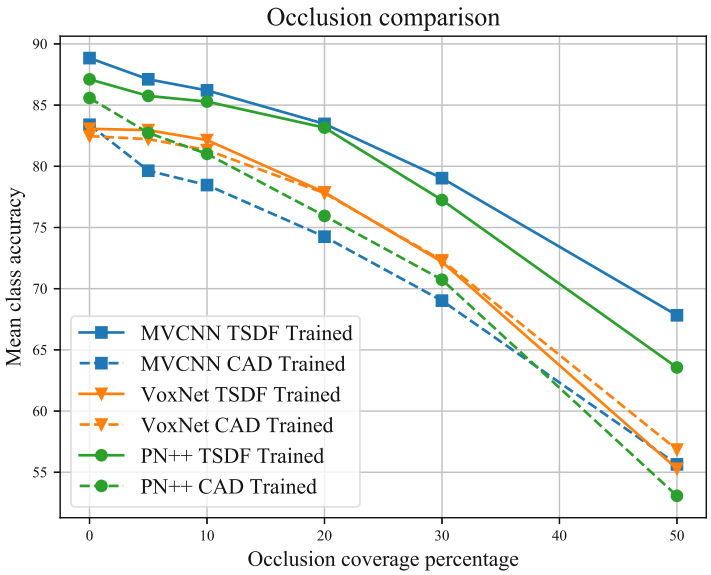


Figure 4.6: Average class accuracy at different degrees of occlusion for CAD and TSDF trained MVCNN, VoxNet and PN++ networks

4.3.2 Circle camera with full, half and quarter circle coverage

Another experiment deals specifically with the limitations of the available camera angles for the scanner. For this purpose, the models were reconstructed similarly to the TSDF pipeline, but only with a limited camera angle. In contrast to the original pipeline, the camera is located at about 50% of the object height, i.e. exactly between the two circles used for training. For the experiment, either all (full), only the cameras from the front or from the back (half) and the sides (quarter) were used. For each step, 12 regularly spaced views were placed. To ensure a correct orientation of the objects, the provided 'Aligned 40-Class ModelNet' dataset was used. However, since only the orientation per class is consistent here, but not among the classes, this orientation was still corrected for some classes.

The results are shown in table 4.6. Figure 4.7 shows the arrangement of the cameras with reference to the table. The left and right quarters differ only slightly, so in each case the average value for both was given.

At the class level, the degradation is also very dependent on the general shape of the objects. Some objects like 'guitar', 'bottle', 'laptop' keep their unique shape even when scanned from only a few angles. For other concave shaped objects, the low viewing angle makes it much harder to distinguish them at fewer angles. Therefore, even at the full circle, a clear degradation is apparent.

Method	Full [a]	Half (Front) [b]	Half (Back) [c]
MVCNN	78.14	73.45 (-6.0%)	67.63 (-13.5%)
VoxNet	71.41	67.33 (-5.7%)	62.00 (-13.2%)
PointNet	75.27	67.45 (-10.4%)	64.10 (-14.8%)
PointNet++	70.58	64.23 (-9.0%)	56.79 (-19.5%)

Method	Quarter (Front) [d]	Quarter (Back) [e]
MVCNN	69.10 (-11.6%)	61.85 (-20.8%)
VoxNet	60.68 (-15.0%)	55.92 (-21.7%)
PointNet	51.62 (-31.4%)	49.14 (-34.7%)
PointNet++	57.73 (-18.2%)	49.39 (-30.0%)

Table 4.6: Impact of limited viewpoints during model reconstruction. Mean class accuracy and relative change in percent, TSDF trained model.

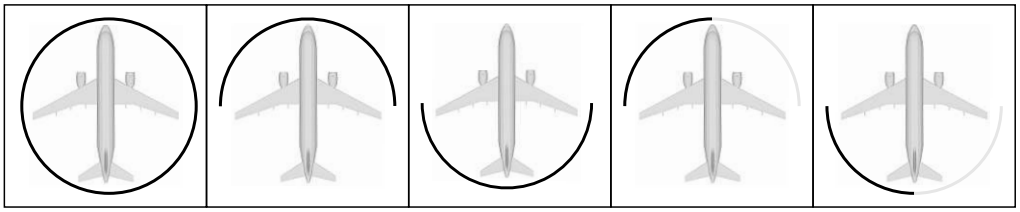


Figure 4.7: Visualization of the restricted viewing angles used for the experiments. (a)-(e) are referenced in Table 4.6

4.3.3 Impact of Scale

Since the actual size of the objects is not known or cannot be taken into account either in ModelNet40 or in the sensor recording, the models are scaled uniformly before they are processed further. This happens as described, on the basis of a unit sphere. If parts of the object are missing due to over-segmentation or missing blink angles, the models are scaled and positioned differently before processing than if they were whole. This behavior is examined separately with an experiment by cutting the models. In contrast to the experiment with the random holes, the clear cuts only resulted in scaling changes that depend on the object and the orientation. The percentages can also not be compared, because here the volumes of the unit sphere and not the ratio of the surface are decisive.

For the experiment, each model was divided according to the TSDF reconstruction. With 2 gradations, either 50% or 30% of the unit sphere volume was removed by a straight horizontal cut. Figure 4.8 shows the cut of such a model. Subsequently, the models were evaluated accordingly. Here, once a correct scaling was performed for the respective incomplete model and once the (incomplete) model was scaled as if it was still whole. The results are summarized in table 4.7.

As can be seen, MVCNN is almost unaffected by scaling. The slight increase in rescaling could be explained by the fact that more of the image area of the input images

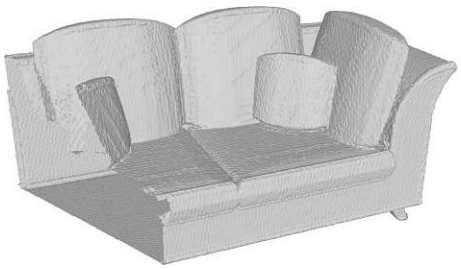


Figure 4.8: Clear cut of an object with 30% unit sphere volume

Method	30% cut (orig scale)	30% cut (rescaled)	50% cut (original)	50% cut (rescaled)
MVCNN	83.1 (-11.1%)	86.0 (-8.0%)	71.0 (-24.1%)	72.7 (-22.2%)
VoxNet	73.5 (-18.4%)	73.0 (-19.0%)	73.0 (-19.0%)	45.2 (-49.8%)
PointNet	79.7 (-11.7%)	76.4 (-15.4%)	26.0 (-71.2%)	40.4 (-55.3%)
PointNet++	84.3 (-9.7%)	83.3 (-10.7%)	71.1 (-23.8%)	68.8 (-26.3%)

Table 4.7: Impact of re-scaling due to incomplete models. Mean class accuracy and relative change in percent. TSDF trained model.

can be utilized and thus more information can be used by the network.

VoxNet shows a clear difference in scaling at the 50% cut. As one can imagine, scaling and centering on the voxel grid gives a significant difference in occupancy. This means that very incomplete models would still perform comparably well with VoxNet. However, due to scaling and positioning, there is a significant degradation in VoxNet compared to MVCNN and PointNet++. When comparing PointNet and PointNet++, it becomes clear that the former generally has a clear disadvantage of 50% due to the lack of hierarchical neighborhood information. PointNet++ seems to perform much better here, as expected, and also seems to have only a slight dependency on scaling.

4.3.4 Floor segmentation error

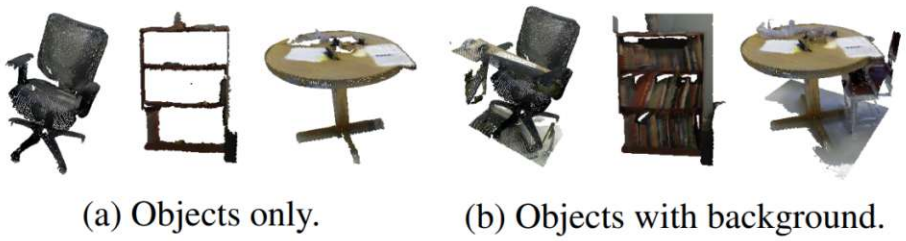
Since many objects in the household area are located on a flat base surface, be it the floor or e.g. a table surface, it can happen that parts of the surface are also incorrectly assigned to the object. Such a failure of the segmentation function is also called under-segmentation. To mimic this behavior, a rectangular plane was attached to the base of the model with a size of 105% of the original object size. The models are then run through the TSDF reconstruction pipeline including the plane and are evaluated.

The results are summarized in table 4.8.

In this experiment, MVCNN again has a clear lead over the other methods. The large drop-off in PointNet and PointNet++ can be explained by the fact that, due to the fixed number of input points, not only unwanted outliers are sampled, but also fewer correct points can be sampled. Since in the experiment the plane depends only on the total size of the objects and not on the actual ground contact area, some objects contain

Method	Bottom plane
MVCNN	78.3 (-16.2%)
VoxNet	66.5 (-26.2%)
PointNet	23.2 (-74.3%)
PointNet++	25.3 (-72.9%)

Table 4.8: Impact of a plane sticking to bottom. Mean class accuracy and relative change in percent. TSDF trained model.



(a) Objects only. (b) Objects with background.

Figure 4.9: Example of objects from the ScanObjectNN dataset [14]

a disproportionate amount of plane (e.g. airplanes). Due to the higher camera angle used in MVCNN, the ground area for many classes does not make much difference for MVCNN.

4.4 Evaluation with ScanObjectNN Dataset

In order to be able to compare the results with real world sensor data, tests were also made with ScanObjectNN, a dataset which is acquired from sensor data. As described in section 4.1.1, both ModelNet11 and ScanObjectNN11 datasets were created from a set of common classes.

For the source data of ScanObjectNN there are two variants for all objects. One is using an exact segmentation and one is using an imprecise one, where also larger parts of the background are located at the objects. The latter was created by placing an axis aligned bounding box over the exactly segmented object and then cutting out the object together with the contents of the bounding box. A few examples of the included objects are shown in figure 4.9.

For the experiments, the networks were trained with ModelNet11 and the reconstruction pipeline. Then they were evaluated with both datasets. As can be seen in table 4.9, there is a significant drop for all networks on the real data, whereas MVCNN still dominates. The differences here are not as clear as usual. Unexpectedly, the accuracy of data with background improves for VoxNet, PointNet and PointNet++. This can possibly be described by the otherwise observed robustness of VoxNet. Because the bounding box is used for the segmentation, there is practically no change in scale, which is also very positive for VoxNet. For PointNet, the background may not provide enough false points

to matter in random sampling. For this case, only every 50^{th} to 150^{th} point is sampled.

Since only 11 classes were considered here instead of the usual 40, the accuracy values cannot be directly compared with the previous experiments.

Method	ModelNet11	ScanObjectNN without BG	ScanObjectNN with BG
MVCNN	93.49	62.69 (-33.0%)	58.58 (-37.3%)
VoxNet	90.08	49.77 (-44.8%)	54.36 (-39.7%)
PointNet++	93.31	52.49 (-43.8)	55.78 (-40.2%)
PointNet	90.31	54.22 (-40.0%)	55.71 (-38.3%)

Table 4.9: Results when training on ModelNet and evaluating on ScanObjectNN with and without baackground (BG). Mean class accuracy in percent and relative change in parenthesis

4.5 Impact of test data degradation

Since MVCNN performed much better than the others in the previous tests, the behavior of MVCNN will be further investigated with additional analysis. An essential property of MVCNN is the ability to find the best result from different perspectives. This makes it robust against a wide range of noise. To investigate this behavior further, from the 12 images per object, a different number of images are replaced by black ones. The images which are left out are always next to each other. Table 4.10 shows the results for the TSDF trained and evaluated model with objects at a random orientation. This means that always one random orientation is left out.

Number of black images	Overall accuracy	Mean class accuracy
1	90.9	88.2
2	91.2	88.3
3	91.1	88.2
6	91.2	87.9
8	90.8	87.8
9	90.3	87.2
10	88.5	86.7
11	83.5	81.7

Table 4.10: Impact of manipulated images on accuracy of the evaluation.

This result shows that even with a small number of usable images, a rather good performance can be achieved. Black images do not produce any activation in the network, so they do not affect the result in a negative way compared to images with other colors.

For the next experiment, the robustness against wrong inserted images is investigated. For that, one random image of the 12 for each object is replaced by a random image from another class. First, a more distinctive object, an airplane, is inserted. This results in an overall accuracy of 91.4% which means a rather small drop. Secondly, a cup, a more generic object, is inserted, which results in a slightly worse drop to 90.1% overall accuracy.

This result shows that as soon as a view resembles a supposedly different object due to noise, it can very well lead to a wrong classification. However, the probability of this seems to be comparably low, which makes MVCNN still very robust.

5 Conclusion

In this work, three 3D classification methods are compared and their performance is analyzed with real sensor data. The central problem with using state-of-the-art methods like deep neural networks is that huge amounts of data training data is necessary in order to achieve the best results. Thanks to computer-aided design (CAD), a huge amount of 3D data is available, but it lacks some similarities with real captured data. In order to further examine this so-called Sim2Real gap, several experiments with three selected classification methods were done. The selection of those methods was based on the type of input data they use. To gain a comprehensive understanding of the algorithms, a view-based, a voxel-based and a point-based algorithm were selected. The particular selected methods were MVCNN [5], VoxNet [11], PointNet [10] and its successor, PointNet++ [4]. Due to the lack of the huge amounts of curated captured data, the experiments were done with CAD data which was altered in several ways. The first step was to setup a reconstruction pipeline which acts similarly to real world scans. The CAD models are rendered in depth images from several perspectives and then reassembled to 3D models again via TSDF. Starting from this data, several other experiments are carried out, like giving the models a random orientation, creating holes of different sizes to simulate occlusion and segmentation errors, cutting away certain parts with clear cuts and experimenting with resulting size changes. To investigate restricted views further, a camera on a circular trajectory with varying coverage was used. Finally, a comparison with real world data from ScanObjectNN dataset was performed.

5.1 Understanding the Sim2Real gap

Based on the analysis of the experiments, especially with the ScanObjectNN dataset, it was shown that it is advantageous for all networks to modify the CAD-based datasets before training. In this case, artificial depth mapping and subsequent TSDF reconstruction improved the outcome and also made the models more robust to other noise. This behavior could be observed with all four methods.

With neural networks acting as a black box, it is complex to explain the cause of certain behaviours. However, with the experiments shown and own assumptions made, it can be shown that it is still possible to predict certain properties. The sensitivity of MVCNN to surface perturbations by the TSDF pipeline as opposed to the other networks, or the vulnerability of VoxNet to changes in the scaling of data are good examples, as they are directly connected to the data representation more so than the network itself. On the other hand, the robustness of MVCNN in the presence of limited camera views can also be well understood by the network combination of multiple views.

When looking at the actual performance, it can be seen that all the networks are performing really well on CAD data. Even with very similar classes like desk and table, the overall accuracy of all networks is close to or over 90%. When doing experiments which heavily modify the data, like creating holes with 50% of the total surface area or doing a reconstruction with a very restricted view, the accuracy quickly drops from 70% to 50%. Depending on the application, this is very likely not acceptable and shows that the use of these methods is only practical if the scans are as complete or more complete than that. Conversely, this means that with such experiments it may also be possible to estimate how complete the scans in the applications actually need to be in order to provide reliable results.

With an overall accuracy of just 67% to 54% all the networks performed rather poorly with the ScanObjectNN dataset. Considering only 11 classes were used instead of the 40 classes for the other experiments, this means for this particular dataset the Sim2Real gap is still too big for these four methods and the data preparation in the form of TSDF reconstruction.

5.2 Outlook

To improve the less than ideal results for the ScanObjectNN dataset, the Sim2Real gap could be reduced in several ways or the methods as a whole could be improved even further.

One direction could be to further investigate how to prepare for better training. As was done with the reconstruction pipeline, even more defects and imperfections could be introduced to make the training data even more similar to the actual scanned data. This approach is usually referred to as data augmentation. In this work the pipeline reconstructed models with complete visibility. A natural extension would be to examine scanned data to best implement the limited view. From the results of the original ScanObjectNN paper [14], we can see that methods like PointNet++ achieve about 77.9% overall accuracy when also trained with the ScanObjectNN data and using 15 classes. Therefore, with a suitable modification of the CAD training data, a similarly good result should also be achievable.

Other feasible experiments could be to further test occlusion behavior, which is one of the main problems. This could possibly be done by setting up artificial room scenes similar to the ones in SceneNN [63] or ScanNet [26]. Performing similar experiments when using more realistic constraints imposed by realistic room layouts and a real sensor would create realistic occlusion and help refine the intuition built in this work. Another interesting experiment could be the simulation of an inaccurate camera position sensor which is a common source of noise and leads to certain artifacts in the reconstructed data.

A completely different direction could be to further investigate the behavior of the networks at a lower level to get even more insights for the Sim2Real Gap. This could be done by visualizing the neural activations and getting a better understanding of e.g. why certain objects are wrongly categorized.

A Appendix

Table A.1 gives an overview of all objects counts for the ModelNet40 dataset. The numbers represent the default test and training split which is also used in this work.

No.	Class	# train	# test
1	airplane	626	100
2	bathtub	106	50
3	bed	515	100
4	bench	173	20
5	bookshelf	572	100
6	bottle	335	100
7	bowl	64	20
8	car	197	100
9	chair	889	100
10	cone	167	20
11	cup	79	20
12	curtain	138	20
13	desk	200	86
14	door	109	20
15	dresser	200	86
16	flower_pot	149	20
17	glass_box	171	100
18	guitar	155	100
19	keyboard	145	20
20	lamp	124	20
21	laptop	149	20
22	mantel	284	100
23	monitor	465	100
24	night_stand	200	86
25	person	88	20
26	piano	231	100
27	plant	240	100
28	radio	104	20
29	range_hood	115	100
30	sink	128	20
31	sofa	680	100
32	stairs	124	20
33	stool	90	20
34	table	392	100
35	tent	163	20
36	toilet	344	100
37	tv_stand	267	100
38	vase	475	100
39	wardrobe	87	20
40	xbox	103	20

Table A.1: ModelNet40 classes and object count with train and test split

A.1 Results for TSDF reconstruction

Refering to section 4.2, the results of training and testing the original and TSDF reconstructed dataset are listed in Tables A.2 and A.3.

	MVCNN				VoxNet			
Training	**CAD**	**CAD**	**TSDF**	**TSDF**	**CAD**	**CAD**	**TSDF**	**TSDF**
Test	**CAD**	**TSDF**	**TSDF**	**CAD**	**CAD**	**TSDF**	**TSDF**	**CAD**
Overall acc.	0.915	0.863	0.915	0.889	0.858	0.854	0.865	0.835
Mean class acc.	0.894	0.834	0.888	0.857	0.829	0.822	0.831	0.808
airplane	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
bathtub	0.940	0.800	0.940	0.920	0.720	0.767	0.820	0.800
bed	1.000	0.990	1.000	0.990	0.980	0.933	0.980	0.970
bench	0.800	0.750	0.750	0.800	0.700	0.600	0.600	0.650
bookshelf	0.910	0.800	0.940	0.930	0.970	0.930	0.950	0.980
bottle	0.990	0.990	0.990	1.000	0.930	0.950	0.950	0.910
bowl	0.900	0.850	1.000	0.900	0.950	0.900	0.950	0.950
car	1.000	1.000	1.000	0.940	0.960	0.957	0.970	0.960
chair	0.970	0.970	0.980	0.980	0.970	0.970	0.970	0.960
cone	0.950	0.900	0.900	0.900	0.950	0.900	0.900	0.900
cup	0.750	0.400	0.700	0.750	0.700	0.750	0.600	0.650
curtain	0.950	1.000	0.950	0.800	0.750	0.750	0.800	0.800
desk	0.826	0.756	0.849	0.767	0.709	0.643	0.698	0.686
door	0.950	1.000	1.000	0.950	0.850	0.883	0.850	0.850
dresser	0.872	0.744	0.791	0.721	0.616	0.601	0.709	0.395
flower_pot	0.050	0.000	0.250	0.050	0.350	0.317	0.250	0.200
glass_box	0.980	0.930	0.970	0.970	0.940	0.850	0.970	0.760
guitar	1.000	1.000	1.000	0.990	0.990	0.997	0.980	0.980
keyboard	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.950
lamp	0.900	0.700	0.900	0.950	0.850	0.800	0.950	0.900
laptop	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
mantel	0.980	0.940	0.990	0.950	0.930	0.953	0.940	0.920
monitor	0.940	0.910	0.970	0.990	0.950	0.963	0.950	0.940
night_stand	0.698	0.558	0.767	0.721	0.651	0.698	0.709	0.674
person	1.000	0.950	0.950	1.000	0.900	0.800	0.800	0.950
piano	0.980	0.900	0.960	0.920	0.850	0.867	0.880	0.830
plant	0.890	0.870	0.870	0.890	0.690	0.730	0.740	0.750
radio	0.950	0.900	0.950	0.900	0.550	0.800	0.500	0.400
range_hood	0.930	0.820	0.900	0.930	0.880	0.867	0.880	0.900
sink	0.800	0.350	0.950	0.850	0.650	0.700	0.750	0.700
sofa	0.940	0.970	0.950	0.900	0.960	0.950	0.960	0.960
stairs	0.950	0.850	1.000	0.900	0.900	0.900	0.900	0.900
stool	0.800	0.800	0.850	0.850	0.750	0.650	0.750	0.700
table	0.760	0.770	0.770	0.760	0.800	0.783	0.710	0.710
tent	0.950	0.950	0.950	0.650	0.950	0.950	0.950	0.950
toilet	1.000	1.000	1.000	1.000	0.960	0.947	0.960	0.970
tv_stand	0.860	0.560	0.890	0.760	0.770	0.790	0.800	0.750
vase	0.830	0.880	0.780	0.770	0.750	0.810	0.750	0.700
wardrobe	0.850	0.850	0.800	0.550	0.700	0.533	0.650	0.700
xbox	1.000	0.950	0.850	0.700	0.700	0.700	0.750	0.650

Table A.2: Accuracy values for training and testing with original CAD and TSDF reconstructed data

	PointNet				PointNet++			
Training	**CAD**	**CAD**	**TSDF**	**TSDF**	**CAD**	**CAD**	**TSDF**	**TSDF**
Test	**CAD**	**TSDF**	**TSDF**	**CAD**	**CAD**	**TSDF**	**TSDF2**	**CAD**
Overall acc.	0.885	0.730	0.876	0.718	0.902	0.876	0.894	0.874
Mean class acc.	0.861	0.705	0.851	0.679	0.880	0.856	0.871	0.850
airplane	1.000	0.550	1.000	1.000	1.000	1.000	1.000	1.000
bathtub	0.820	0.700	0.820	0.620	0.940	0.940	0.900	0.880
bed	0.990	0.920	0.960	0.760	0.970	0.950	0.970	0.950
bench	0.650	0.600	0.750	0.750	0.800	0.750	0.700	0.700
bookshelf	0.890	0.820	0.920	0.890	0.930	0.930	0.930	0.920
bottle	0.940	0.880	0.940	0.800	0.960	0.930	0.950	0.940
bowl	0.950	0.900	0.900	1.000	0.950	0.800	0.950	1.000
car	0.980	0.880	0.970	1.000	0.980	0.980	0.990	0.990
chair	0.970	0.950	0.970	0.930	0.960	0.950	0.960	0.970
cone	1.000	0.750	1.000	0.700	1.000	1.000	0.900	1.000
cup	0.650	0.550	0.750	0.550	0.800	0.600	0.800	0.900
curtain	0.900	0.950	0.900	0.850	0.900	0.850	0.900	0.850
desk	0.826	0.512	0.779	0.395	0.930	0.826	0.884	0.895
door	0.850	0.900	0.850	0.850	0.900	0.850	0.850	0.900
dresser	0.651	0.535	0.698	0.430	0.698	0.430	0.733	0.477
flower_pot	0.250	0.250	0.250	0.100	0.350	0.150	0.300	0.350
glass_box	0.960	0.720	0.970	0.710	0.950	0.790	0.930	0.880
guitar	1.000	0.890	0.990	0.880	0.990	1.000	0.950	0.840
keyboard	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
lamp	0.950	0.900	0.850	0.550	0.800	0.850	0.850	0.800
laptop	1.000	0.300	1.000	0.000	1.000	1.000	1.000	1.000
mantel	0.960	0.920	0.940	0.630	0.970	0.960	0.970	0.960
monitor	0.950	0.940	0.930	0.890	0.990	0.980	0.970	0.980
night_stand	0.791	0.651	0.744	0.581	0.698	0.547	0.686	0.814
person	0.950	0.750	0.900	0.900	0.900	0.950	1.000	0.950
piano	0.870	0.640	0.890	0.800	0.940	0.940	0.950	0.950
plant	0.740	0.650	0.710	0.710	0.740	0.890	0.730	0.710
radio	0.800	0.450	0.800	0.400	0.750	0.800	0.750	0.700
range_hood	0.920	0.660	0.940	0.640	0.950	0.930	0.960	0.980
sink	0.750	0.400	0.750	0.700	0.850	0.900	0.850	0.850
sofa	0.980	0.940	0.940	0.970	0.970	0.960	0.960	0.960
stairs	0.900	0.750	0.800	0.600	0.950	0.950	1.000	0.950
stool	0.850	0.450	0.750	0.550	0.750	0.800	0.800	0.700
table	0.800	0.060	0.780	0.020	0.780	0.790	0.750	0.720
tent	0.950	1.000	0.950	0.550	0.950	0.950	0.950	0.950
toilet	0.970	0.840	0.970	0.880	0.990	0.990	1.000	0.990
tv_stand	0.800	0.720	0.830	0.720	0.880	0.800	0.870	0.810
vase	0.790	0.830	0.740	0.750	0.790	0.870	0.800	0.780
wardrobe	0.550	0.450	0.650	0.450	0.800	0.800	0.650	0.350
xbox	0.900	0.650	0.750	0.650	0.750	0.850	0.750	0.650

Table A.3: Accuracy values for training and testing with original CAD and TSDF reconstructed data

The figures A.1 to A.4 show confusion matrices for training and testing with CAD and TSDF reconstructed data in all combinations. Due to the space requirements and conciseness, the class names are not labeled on the axes. The 40 classes are shown in alphabetical order from top to bottom and left to right in the diagram. Figure A.17 can be seen as a reference with fully labeled axis.

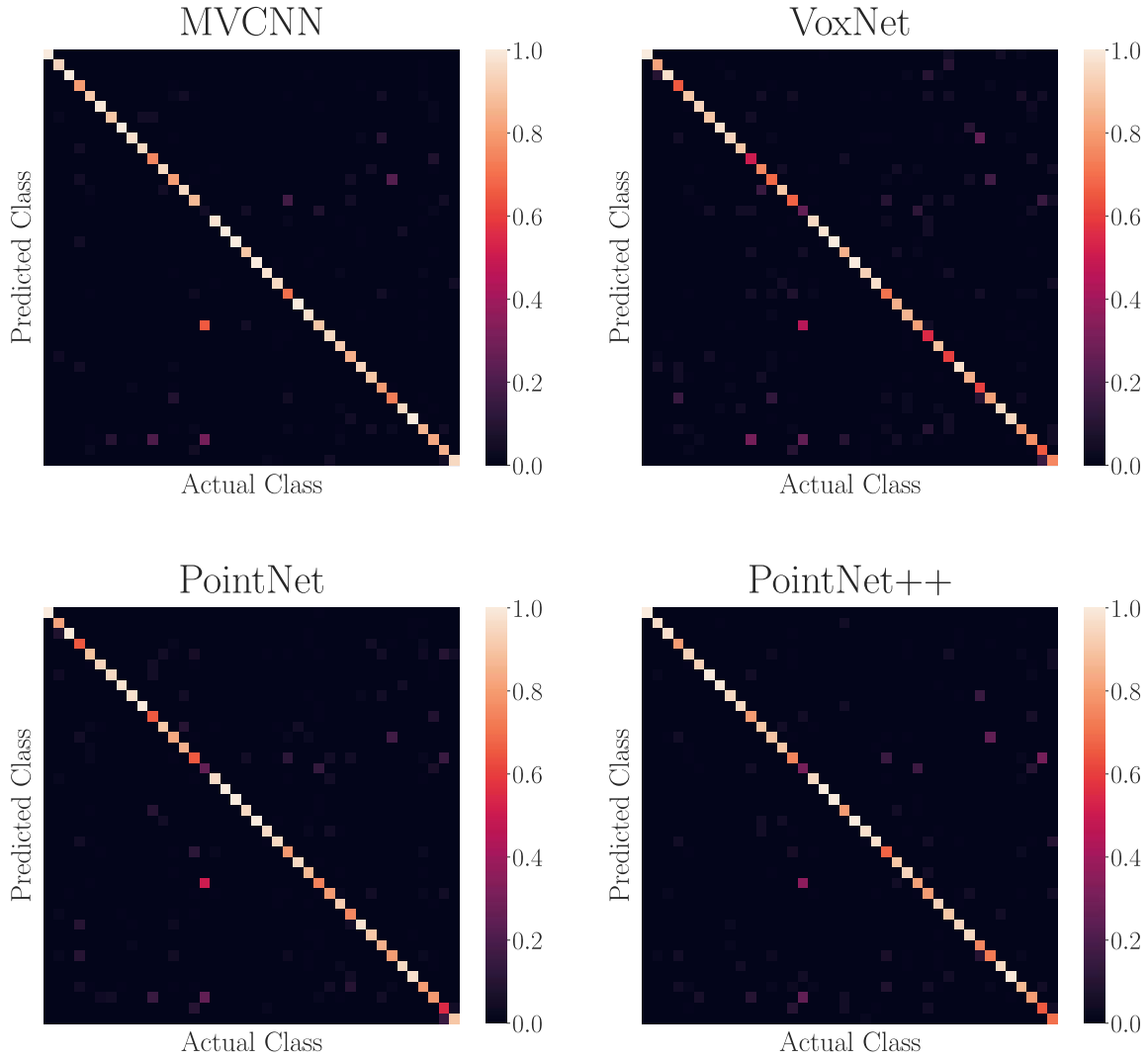


Figure A.1: Confusion matrices for CAD trained and CAD test data

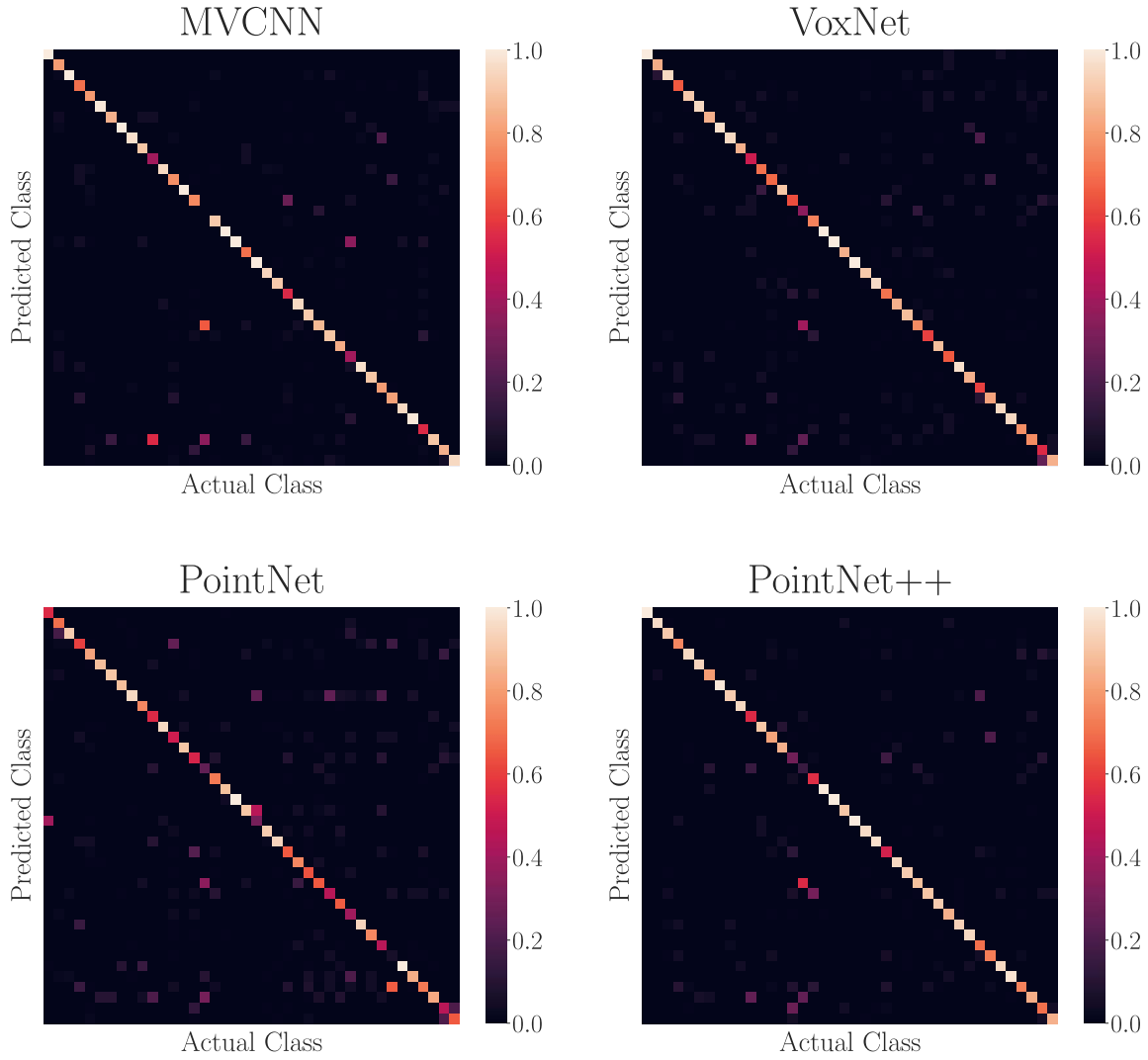


Figure A.2: Confusion matrices for CAD trained and TSDF test data

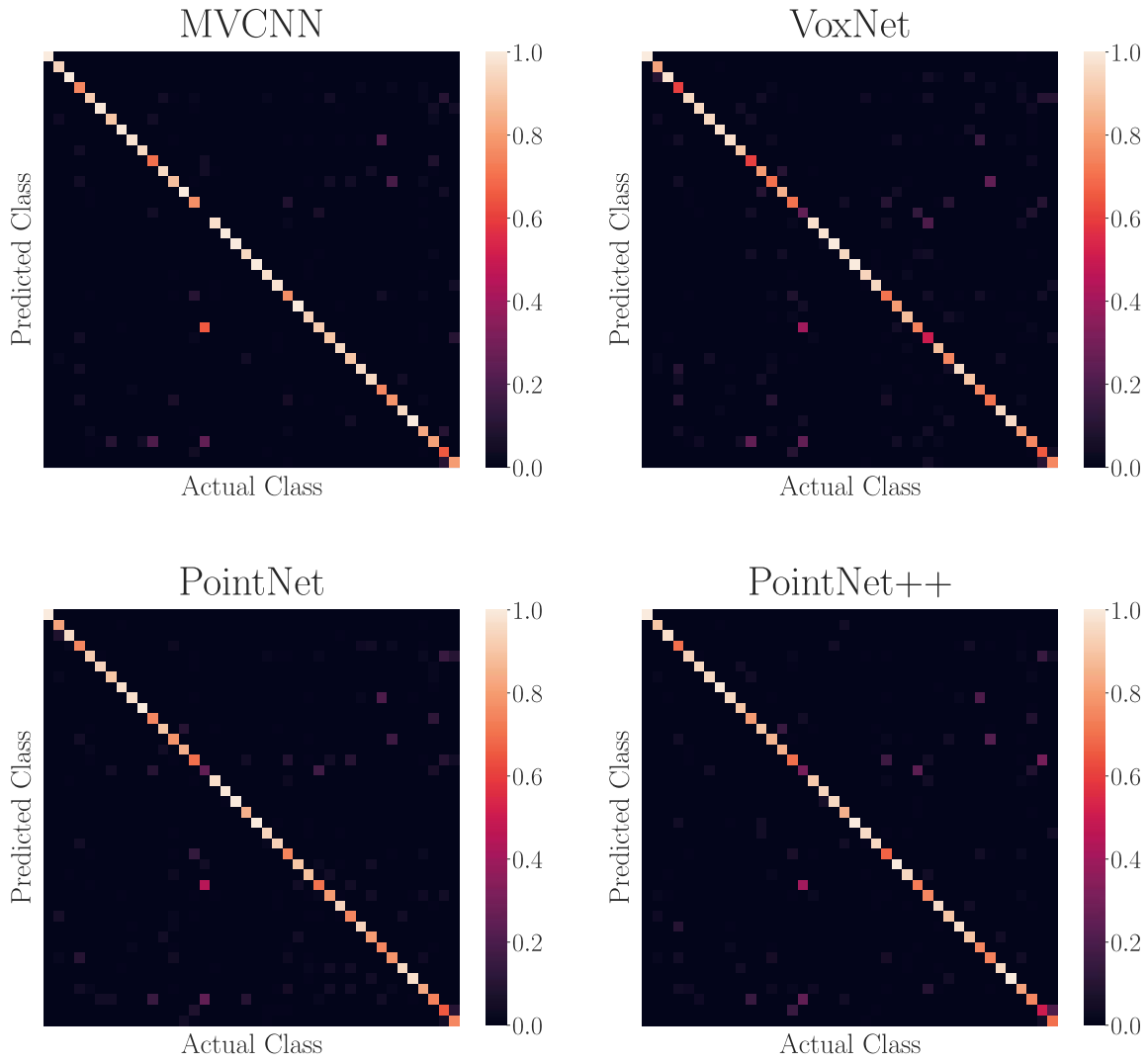


Figure A.3: Confusion matrices for TSDF trained and TSDF test data

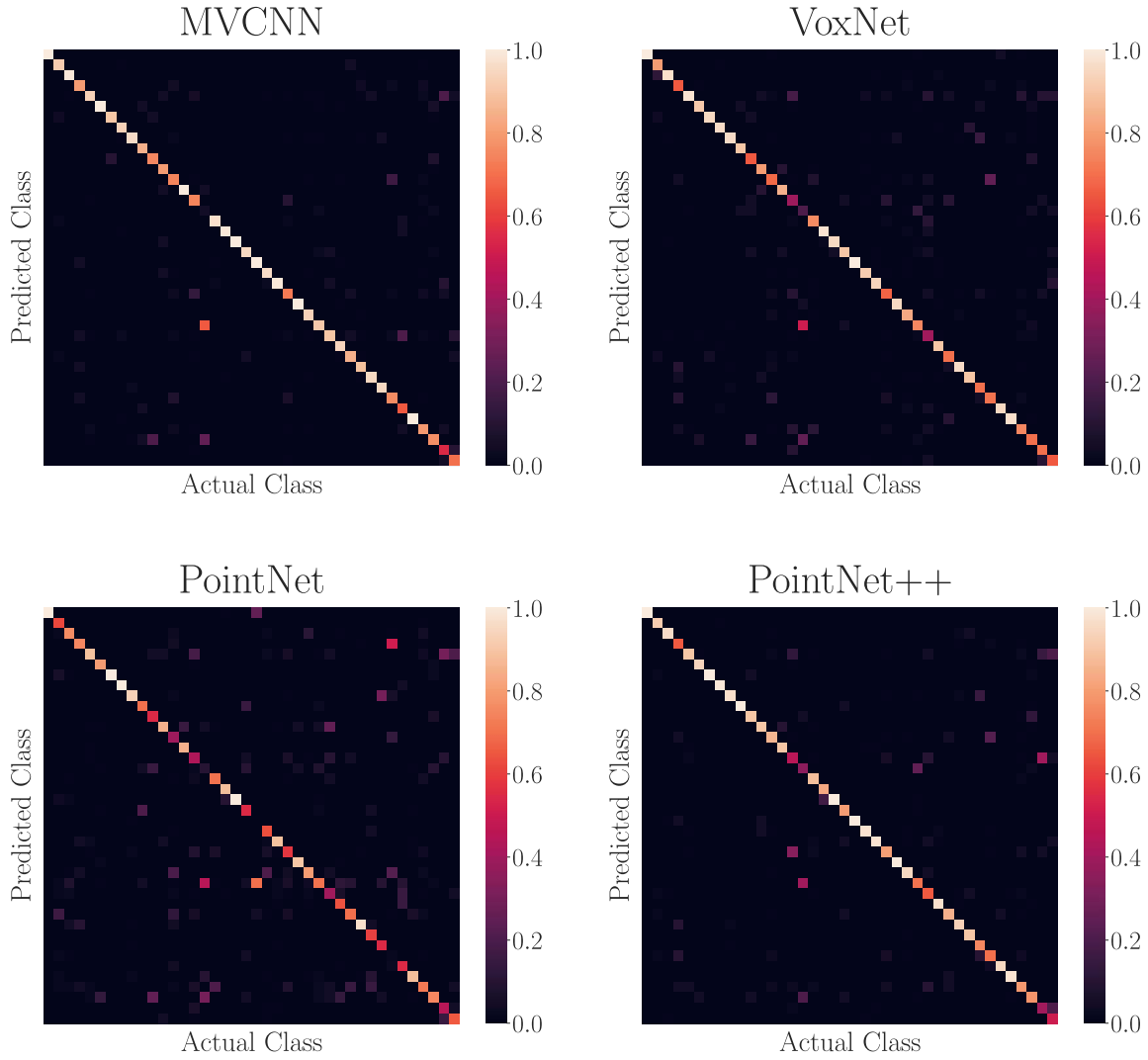


Figure A.4: Confusion matrices for TSDF trained and CAD test data

A.2 Results for random rotation

Refering to section 4.2.1, the results of the experiment with random rotation are presented in table A.4. In this experiment, all models are trained with reconstructed TSDF data and tested with the same, but with a random rotation around the up-axis. The corresponding confusion matrices are shown in figure A.5.

	MVCNN	VoxNet	PointNet	PointNet++
Overall acc.	0.915	0.859	0.880	0.891
Mean class acc.	0.888	0.825	0.850	0.866
airplane	1.000	1.000	1.000	1.000
bathtub	0.940	0.827	0.793	0.907
bed	1.000	0.980	0.970	0.970
bench	0.750	0.600	0.717	0.717
bookshelf	0.940	0.930	0.903	0.920
bottle	0.990	0.947	0.943	0.953
bowl	0.900	0.950	1.000	0.900
car	1.000	0.970	0.983	0.987
chair	0.980	0.967	0.983	0.960
cone	0.950	0.900	1.000	0.950
cup	0.700	0.600	0.733	0.817
curtain	0.950	0.833	0.900	0.900
desk	0.872	0.682	0.795	0.899
door	1.000	0.833	0.833	0.833
dresser	0.779	0.678	0.725	0.775
flower_pot	0.000	0.250	0.283	0.267
glass_box	0.970	0.967	0.970	0.900
guitar	0.990	0.960	0.990	0.950
keyboard	1.000	1.000	1.000	0.983
lamp	0.900	0.950	0.850	0.850
laptop	1.000	1.000	1.000	1.000
mantel	0.970	0.940	0.953	0.960
monitor	0.970	0.950	0.943	0.980
night_stand	0.756	0.659	0.752	0.663
person	1.000	0.850	0.833	1.000
piano	0.950	0.847	0.893	0.937
plant	0.920	0.727	0.723	0.750
radio	0.950	0.533	0.733	0.733
range_hood	0.960	0.887	0.927	0.963
sink	0.950	0.717	0.700	0.850
sofa	0.960	0.960	0.950	0.947
stairs	0.950	0.867	0.817	0.983
stool	0.750	0.717	0.733	0.750
table	0.760	0.737	0.783	0.760
tent	0.950	0.950	0.950	0.950
toilet	1.000	0.960	0.983	0.997
tv_stand	0.820	0.787	0.827	0.847
vase	0.810	0.760	0.747	0.793
wardrobe	0.700	0.583	0.617	0.617
xbox	0.800	0.733	0.767	0.717

Table A.4: Accuracy values for random rotation experiment

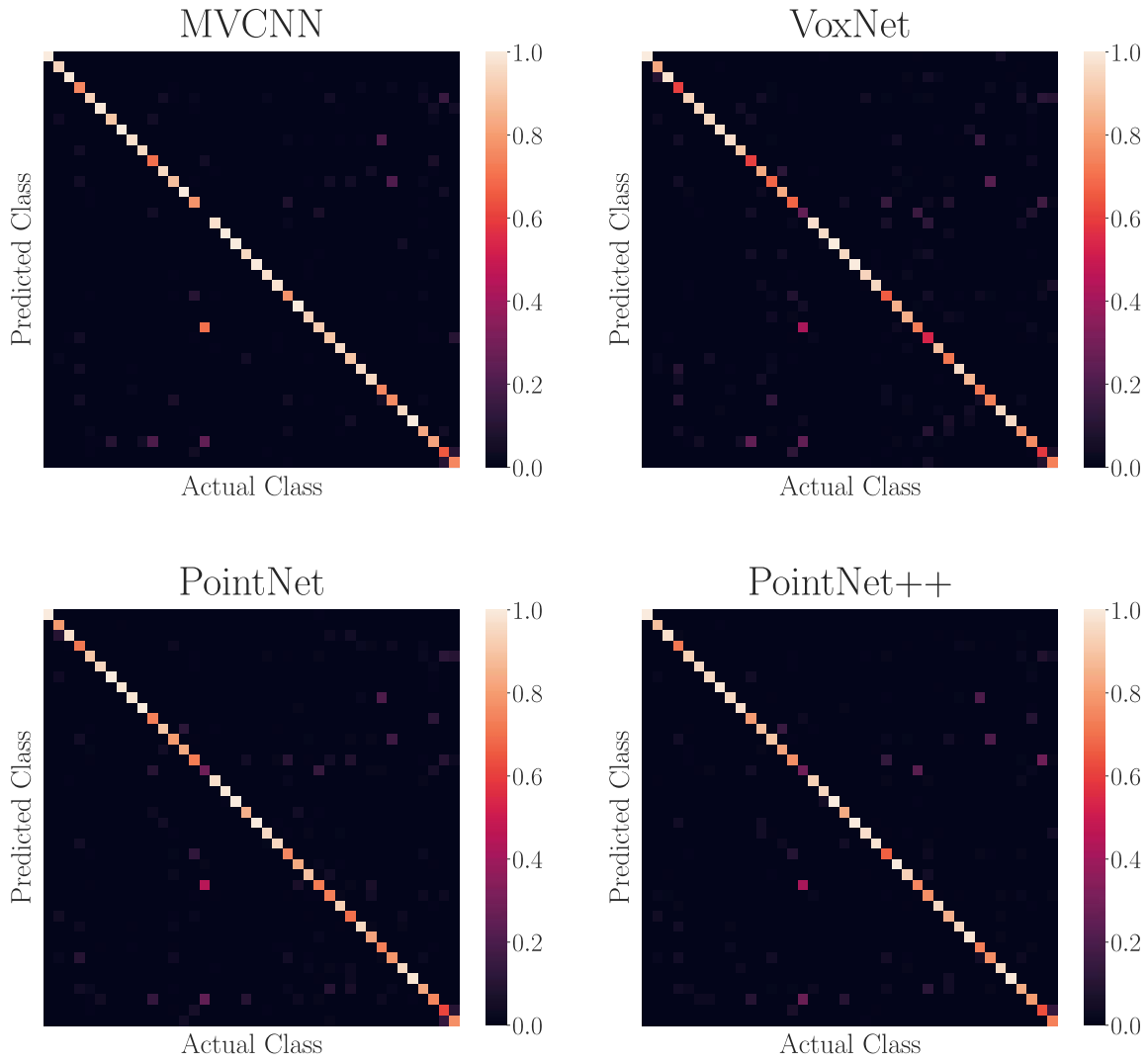


Figure A.5: Confusion matrices for random rotation experiment

A.3 Results for random holes

The tables A.5-A.8 show all the results for the random holes experiment described in section 4.3.1. The percentage indicates the dimensions of the hole in relation to the total surface of the object. The associated confusion matrices for 30% and 50% hole size is shown in figures A.6 and A.7.

	MVCNN							
	CAD				TSDF			
Hole Surface Area	**5%**	**10%**	**30%**	**50%**	**5%**	**10%**	**30%**	**50%**
Overall acc.	0.819	0.806	0.701	0.575	0.903	0.892	0.821	0.700
Mean class acc.	0.796	0.785	0.690	0.556	0.871	0.862	0.790	0.678
airplane	1.000	1.000	0.990	0.780	1.000	1.000	1.000	0.890
bathtub	0.780	0.720	0.420	0.200	0.940	0.860	0.800	0.500
bed	0.990	0.980	0.890	0.760	1.000	1.000	0.960	0.910
bench	0.700	0.750	0.750	0.500	0.750	0.800	0.800	0.750
bookshelf	0.750	0.760	0.560	0.380	0.950	0.930	0.890	0.790
bottle	0.980	0.950	0.920	0.730	0.990	0.970	0.950	0.770
bowl	0.850	0.800	0.900	0.750	0.900	0.950	0.900	0.900
car	1.000	1.000	0.990	0.960	0.990	0.990	0.980	0.920
chair	0.970	0.970	0.930	0.820	0.970	0.980	0.960	0.880
cone	0.900	0.800	0.850	0.700	0.950	0.900	0.900	0.800
cup	0.300	0.400	0.250	0.000	0.650	0.600	0.400	0.150
curtain	0.950	0.950	0.950	0.700	0.900	0.900	0.950	0.900
desk	0.709	0.779	0.605	0.477	0.872	0.884	0.837	0.721
door	0.950	0.850	0.800	0.650	1.000	0.950	0.900	0.850
dresser	0.581	0.512	0.291	0.081	0.744	0.663	0.419	0.233
flower_pot	0.050	0.050	0.100	0.050	0.000	0.000	0.000	0.050
glass_box	0.560	0.330	0.070	0.050	0.910	0.880	0.760	0.510
guitar	0.990	1.000	0.960	0.920	1.000	1.000	0.960	0.870
keyboard	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
lamp	0.750	0.750	0.650	0.550	0.900	1.000	0.750	0.600
laptop	1.000	1.000	1.000	0.800	1.000	1.000	1.000	0.800
mantel	0.910	0.910	0.770	0.590	0.980	0.980	0.950	0.810
monitor	0.900	0.860	0.780	0.670	0.990	0.980	0.980	0.930
night_stand	0.535	0.547	0.453	0.279	0.767	0.698	0.616	0.477
person	0.950	0.950	0.950	0.850	1.000	0.950	0.950	0.800
piano	0.910	0.910	0.820	0.660	0.930	0.930	0.890	0.740
plant	0.870	0.900	0.920	0.910	0.910	0.940	0.980	0.940
radio	0.800	0.700	0.650	0.250	0.800	0.750	0.750	0.700
range_hood	0.780	0.810	0.610	0.430	0.950	0.950	0.840	0.650
sink	0.300	0.350	0.200	0.050	0.850	0.850	0.700	0.600
sofa	0.940	0.970	0.910	0.910	0.950	0.940	0.910	0.860
stairs	0.900	0.850	0.850	0.750	0.950	0.950	0.950	0.900
stool	0.800	0.800	0.850	0.550	0.750	0.750	0.800	0.600
table	0.750	0.730	0.450	0.270	0.730	0.660	0.430	0.310
tent	0.900	0.900	0.750	0.800	0.900	0.900	0.900	0.750
toilet	1.000	0.990	0.950	0.880	1.000	1.000	0.970	0.820
tv_stand	0.340	0.260	0.080	0.020	0.770	0.700	0.430	0.230
vase	0.860	0.900	0.840	0.680	0.800	0.820	0.750	0.620
wardrobe	0.750	0.800	0.400	0.250	0.650	0.700	0.300	0.150
xbox	0.900	0.900	0.500	0.600	0.750	0.750	0.400	0.450

Table A.5: Accuracy values for random holes experiment for MVCNN

	VoxNet							
	CAD				TSDF			
Hole Surface Area	**5%**	**10%**	**30%**	**50%**	**5%**	**10%**	**30%**	**50%**
Overall acc.	0.855	0.841	0.761	0.600	0.861	0.861	0.760	0.583
Mean class acc.	0.823	0.813	0.723	0.569	0.830	0.821	0.722	0.553
airplane	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.997
bathtub	0.800	0.780	0.720	0.647	0.800	0.760	0.660	0.507
bed	0.930	0.940	0.900	0.700	0.970	0.960	0.890	0.657
bench	0.550	0.600	0.700	0.733	0.650	0.700	0.750	0.667
bookshelf	0.920	0.920	0.850	0.570	0.930	0.940	0.800	0.463
bottle	0.940	0.950	0.860	0.583	0.950	0.970	0.870	0.397
bowl	0.900	0.850	0.850	0.817	0.950	0.950	0.850	0.850
car	0.950	0.950	0.880	0.837	0.970	0.950	0.860	0.770
chair	0.980	0.970	0.980	0.943	0.970	0.960	0.940	0.927
cone	0.900	0.900	0.900	0.817	0.900	0.900	0.850	0.700
cup	0.750	0.700	0.700	0.400	0.600	0.600	0.500	0.250
curtain	0.750	0.800	0.700	0.717	0.750	0.800	0.700	0.717
desk	0.686	0.674	0.593	0.438	0.709	0.744	0.651	0.516
door	0.900	0.900	0.600	0.150	0.800	0.900	0.500	0.133
dresser	0.570	0.523	0.419	0.159	0.674	0.686	0.360	0.047
flower_pot	0.250	0.200	0.050	0.100	0.200	0.250	0.100	0.083
glass_box	0.810	0.680	0.380	0.110	0.970	0.940	0.510	0.180
guitar	1.000	1.000	0.980	0.957	0.970	0.980	0.960	0.987
keyboard	0.950	1.000	0.650	0.300	1.000	1.000	0.900	0.483
lamp	0.800	0.850	0.750	0.783	0.950	0.950	0.900	0.817
laptop	1.000	1.000	1.000	0.950	1.000	1.000	1.000	1.000
mantel	0.950	0.950	0.910	0.647	0.940	0.950	0.860	0.647
monitor	0.980	0.930	0.930	0.797	0.960	0.940	0.920	0.730
night_stand	0.663	0.651	0.523	0.244	0.733	0.733	0.605	0.391
person	0.800	0.800	0.650	0.383	0.850	0.850	0.750	0.350
piano	0.870	0.870	0.700	0.487	0.880	0.860	0.680	0.443
plant	0.780	0.790	0.830	0.767	0.730	0.720	0.800	0.760
radio	0.800	0.800	0.550	0.400	0.400	0.600	0.400	0.350
range_hood	0.860	0.860	0.830	0.653	0.890	0.850	0.820	0.697
sink	0.700	0.700	0.700	0.667	0.700	0.700	0.700	0.700
sofa	0.960	0.940	0.870	0.723	0.950	0.950	0.910	0.783
stairs	0.850	0.900	0.800	0.750	0.850	0.900	0.850	0.783
stool	0.750	0.700	0.600	0.333	0.700	0.750	0.700	0.233
table	0.790	0.800	0.730	0.617	0.740	0.740	0.670	0.487
tent	0.950	0.950	0.950	0.917	0.950	0.950	0.900	0.900
toilet	0.950	0.960	0.870	0.627	0.970	0.970	0.910	0.733
tv_stand	0.780	0.710	0.510	0.317	0.750	0.770	0.510	0.263
vase	0.810	0.780	0.710	0.570	0.750	0.760	0.740	0.567
wardrobe	0.550	0.450	0.250	0.000	0.700	0.600	0.050	0.000
xbox	0.800	0.800	0.550	0.133	0.700	0.650	0.550	0.150

Table A.6: Accuracy values for random holes experiment for VoxNet

	PointNet							
	CAD				TSDF			
Hole Surface Area	**5%**	**10%**	**30%**	**50%**	**5%**	**10%**	**30%**	**50%**
Overall acc.	0.723	0.688	0.504	0.270	0.876	0.868	0.718	0.446
Mean class acc.	0.702	0.670	0.517	0.293	0.846	0.838	0.703	0.463
airplane	0.560	0.540	1.000	0.100	1.000	1.000	0.930	0.380
bathtub	0.760	0.740	0.700	0.200	0.820	0.740	0.680	0.220
bed	0.880	0.890	0.900	0.400	0.960	0.970	0.810	0.290
bench	0.700	0.600	0.750	0.650	0.750	0.750	0.650	0.500
bookshelf	0.800	0.780	0.840	0.110	0.930	0.900	0.670	0.310
bottle	0.860	0.870	0.920	0.070	0.940	0.940	0.810	0.340
bowl	0.900	0.800	0.950	0.550	1.000	0.950	0.900	0.800
car	0.870	0.850	0.920	0.350	0.980	0.980	0.850	0.710
chair	0.960	0.940	0.970	0.890	0.980	0.970	0.930	0.810
cone	0.750	0.750	0.900	0.550	0.950	1.000	0.900	0.700
cup	0.400	0.400	0.600	0.100	0.700	0.750	0.600	0.350
curtain	0.950	0.950	0.900	0.550	0.900	0.950	0.850	0.750
desk	0.488	0.430	0.733	0.174	0.767	0.802	0.663	0.500
door	0.900	0.800	0.700	0.500	0.850	0.800	0.700	0.500
dresser	0.570	0.500	0.581	0.035	0.721	0.721	0.512	0.093
flower_pot	0.300	0.300	0.300	0.050	0.150	0.300	0.200	0.050
glass_box	0.650	0.400	0.760	0.000	0.970	0.920	0.300	0.030
guitar	0.880	0.870	0.990	0.740	0.980	1.000	1.000	0.990
keyboard	1.000	1.000	0.950	0.350	1.000	1.000	0.750	0.350
lamp	0.900	0.900	0.900	0.700	0.850	0.900	0.750	0.800
laptop	0.250	0.250	1.000	0.150	1.000	1.000	0.950	0.750
mantel	0.880	0.850	0.880	0.330	0.940	0.940	0.740	0.420
monitor	0.930	0.930	0.900	0.310	0.950	0.940	0.890	0.650
night_stand	0.674	0.616	0.628	0.070	0.779	0.674	0.465	0.151
person	0.750	0.750	0.750	0.150	0.900	0.850	0.750	0.350
piano	0.610	0.570	0.750	0.090	0.890	0.880	0.630	0.280
plant	0.670	0.620	0.730	0.590	0.710	0.760	0.710	0.680
radio	0.450	0.350	0.600	0.000	0.800	0.700	0.650	0.550
range_hood	0.680	0.580	0.790	0.100	0.930	0.920	0.740	0.410
sink	0.500	0.450	0.750	0.400	0.750	0.650	0.700	0.600
sofa	0.910	0.870	0.830	0.070	0.940	0.930	0.720	0.320
stairs	0.750	0.700	0.850	0.450	0.800	0.750	0.750	0.600
stool	0.500	0.500	0.600	0.150	0.750	0.700	0.550	0.300
table	0.050	0.060	0.720	0.030	0.770	0.740	0.740	0.510
tent	0.950	1.000	0.950	0.850	0.950	0.950	0.900	0.900
toilet	0.840	0.840	0.980	0.370	0.980	0.990	0.870	0.570
tv_stand	0.720	0.720	0.650	0.080	0.820	0.790	0.500	0.220
vase	0.820	0.790	0.700	0.460	0.730	0.750	0.680	0.500
wardrobe	0.450	0.400	0.350	0.000	0.500	0.500	0.200	0.050
xbox	0.600	0.650	0.750	0.000	0.750	0.750	0.550	0.250

Table A.7: Accuracy values for random holes experiment for PointNet

	PointNet++							
	CAD				TSDF			
Hole Surface Area	**5%**	**10%**	**30%**	**50%**	**5%**	**10%**	**30%**	**50%**
Overall acc.	0.852	0.827	0.718	0.529	0.882	0.881	0.799	0.659
Mean class acc.	0.827	0.810	0.707	0.531	0.858	0.853	0.772	0.636
airplane	1.000	1.000	1.000	0.930	1.000	1.000	0.990	0.800
bathtub	0.960	0.880	0.660	0.400	0.900	0.880	0.760	0.440
bed	0.920	0.840	0.760	0.560	0.950	0.960	0.900	0.800
bench	0.700	0.750	0.650	0.400	0.750	0.700	0.650	0.600
bookshelf	0.920	0.910	0.840	0.700	0.920	0.910	0.810	0.680
bottle	0.940	0.920	0.760	0.380	0.950	0.940	0.750	0.560
bowl	0.800	0.850	0.800	0.650	0.900	0.850	0.900	0.750
car	0.970	0.970	0.880	0.600	0.990	0.980	0.920	0.750
chair	0.950	0.970	0.930	0.730	0.970	0.960	0.960	0.770
cone	0.900	0.900	0.800	0.600	0.950	0.900	0.800	0.800
cup	0.600	0.500	0.550	0.550	0.750	0.800	0.750	0.600
curtain	0.900	0.800	0.750	0.500	0.900	0.950	0.850	0.700
desk	0.814	0.837	0.744	0.488	0.895	0.872	0.779	0.674
door	0.850	0.700	0.650	0.600	0.800	0.750	0.500	0.600
dresser	0.279	0.198	0.128	0.070	0.721	0.791	0.651	0.477
flower_pot	0.050	0.350	0.200	0.150	0.300	0.300	0.150	0.100
glass_box	0.710	0.600	0.370	0.270	0.890	0.900	0.770	0.520
guitar	0.990	0.980	0.900	0.770	0.940	0.930	0.870	0.740
keyboard	1.000	1.000	0.700	0.550	0.950	1.000	0.900	0.600
lamp	0.850	0.850	0.750	0.600	0.900	0.800	0.800	0.650
laptop	1.000	1.000	0.850	0.550	1.000	1.000	1.000	0.750
mantel	0.940	0.920	0.790	0.610	0.960	0.970	0.940	0.830
monitor	0.970	0.980	0.910	0.660	0.970	0.980	0.920	0.790
night_stand	0.500	0.430	0.291	0.081	0.605	0.651	0.488	0.384
person	0.900	0.950	0.800	0.700	0.950	0.900	0.800	0.700
piano	0.940	0.920	0.860	0.700	0.960	0.960	0.890	0.740
plant	0.880	0.870	0.940	0.920	0.780	0.810	0.860	0.910
radio	0.700	0.750	0.700	0.450	0.700	0.800	0.750	0.700
range_hood	0.900	0.900	0.660	0.450	0.960	0.950	0.870	0.740
sink	0.900	0.900	0.750	0.750	0.850	0.850	0.750	0.700
sofa	0.940	0.900	0.710	0.260	0.940	0.940	0.810	0.580
stairs	0.900	0.900	0.950	0.750	0.900	0.850	0.900	0.800
stool	0.850	0.750	0.650	0.500	0.850	0.800	0.650	0.350
table	0.740	0.720	0.630	0.580	0.710	0.640	0.610	0.580
tent	0.950	0.950	0.950	0.950	0.950	0.950	0.950	0.900
toilet	0.990	0.970	0.830	0.450	0.990	0.990	0.890	0.690
tv_stand	0.800	0.660	0.450	0.290	0.840	0.850	0.660	0.510
vase	0.840	0.830	0.650	0.480	0.760	0.750	0.650	0.560
wardrobe	0.550	0.600	0.450	0.300	0.550	0.600	0.600	0.200
xbox	0.800	0.700	0.650	0.300	0.750	0.700	0.450	0.400

Table A.8: Accuracy values for random holes experiment for PointNet++

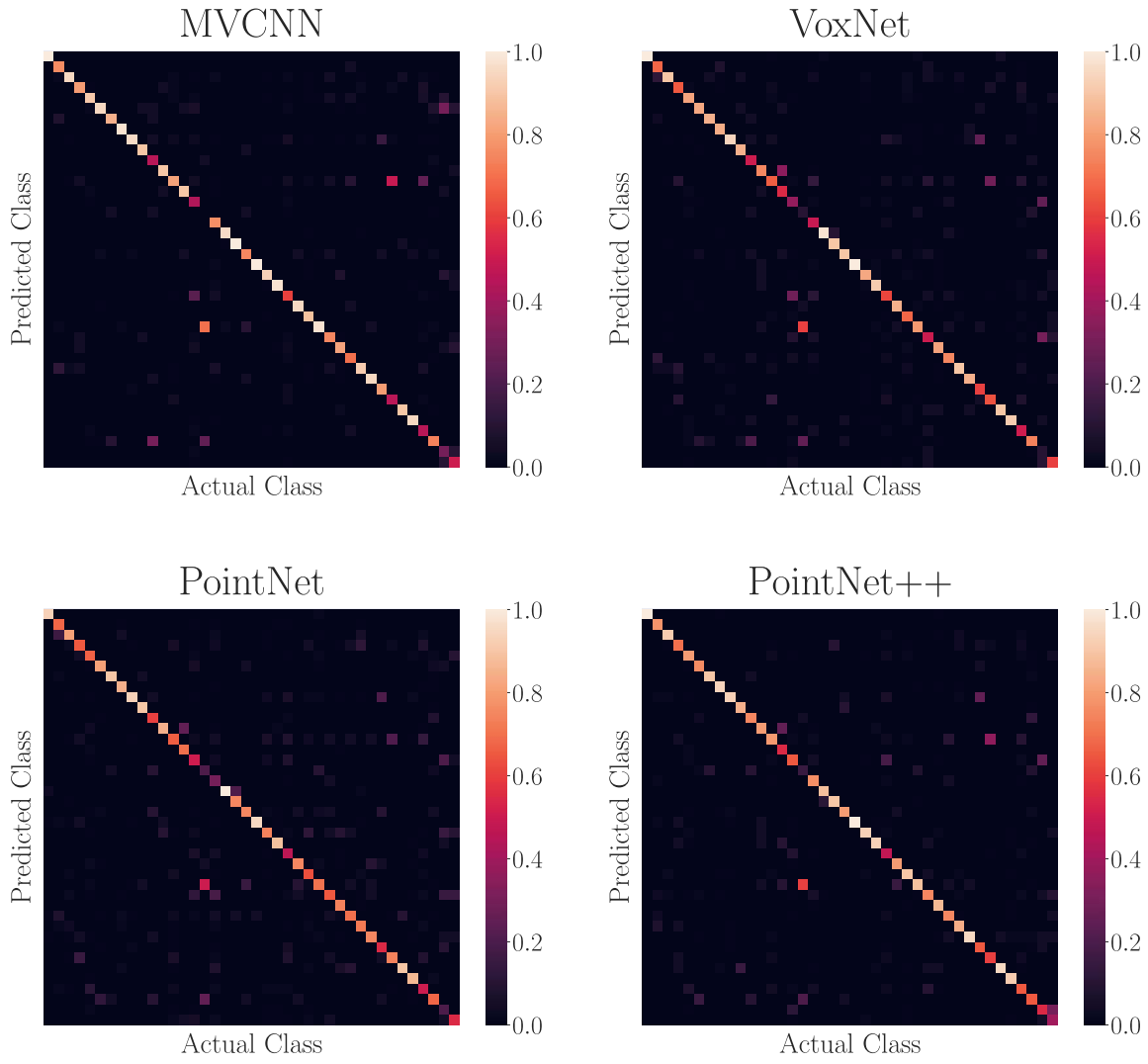


Figure A.6: Confusion matrices for random holes with 30% surface coverage

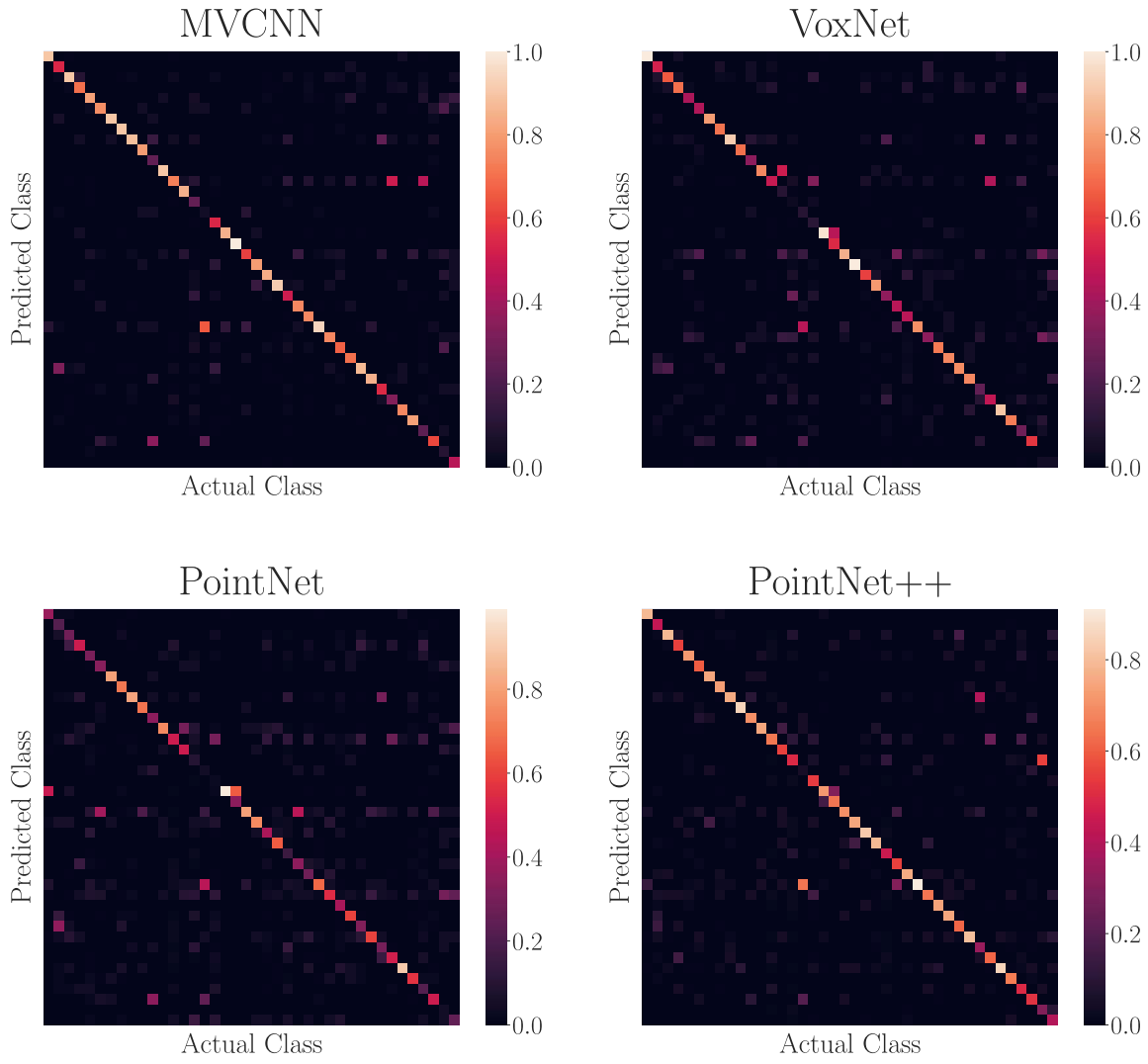


Figure A.7: Confusion matrices for random holes with 50% surface coverage

A.4 Results for circle camera

The tables A.9 and A.10 show the results of the circle camera experiment described in section 4.3.2. The viewing angles for the different coverages are shown in figure 4.7. The corresponding confusion matrices are shown in figures A.8 to A.10 for full, half and quarter circles.

	MVCNN					VoxNet				
Viewing Angle	**Full**	**Half Front**	**Half Back**	**Quart. Front**	**Quart. Back**	**Full**	**Half Front**	**Half Back**	**Quart. Front**	**Quart. Back**
Overall acc.	0.814	0.775	0.709	0.737	0.646	0.741	0.693	0.615	0.625	0.551
Mean class acc.	0.781	0.734	0.676	0.691	0.619	0.714	0.673	0.620	0.607	0.559
airplane	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
bathtub	0.700	0.620	0.700	0.270	0.280	0.700	0.760	0.700	0.230	0.230
bed	0.910	0.930	0.880	0.860	0.725	0.580	0.530	0.560	0.410	0.430
bench	0.700	0.750	0.650	0.650	0.600	0.650	0.700	0.750	0.750	0.825
bookshelf	0.960	0.940	0.500	0.950	0.505	0.940	0.920	0.470	0.920	0.485
bottle	0.980	0.980	0.980	0.975	0.965	0.970	0.970	0.970	0.980	0.970
bowl	0.950	0.950	1.000	0.875	0.850	0.700	0.700	0.750	0.675	0.650
car	0.990	0.960	0.950	0.920	0.800	0.990	0.980	0.970	0.850	0.530
chair	0.980	0.960	0.960	0.945	0.945	0.960	0.970	0.950	0.970	0.955
cone	0.950	0.950	0.950	0.950	0.950	0.900	0.900	0.900	0.900	0.900
cup	0.500	0.300	0.350	0.200	0.275	0.500	0.500	0.450	0.250	0.350
curtain	0.950	0.900	0.900	0.900	0.850	0.700	0.800	0.850	0.850	0.875
desk	0.337	0.430	0.244	0.366	0.250	0.628	0.628	0.419	0.535	0.407
door	1.000	1.000	1.000	0.975	1.000	0.850	0.700	0.750	0.600	0.575
dresser	0.372	0.174	0.012	0.198	0.000	0.407	0.047	0.012	0.000	0.000
flower_pot	0.050	0.050	0.050	0.050	0.000	0.250	0.250	0.250	0.125	0.175
glass_box	0.970	0.910	0.940	0.830	0.870	0.400	0.340	0.250	0.095	0.065
guitar	1.000	0.990	1.000	0.960	0.945	0.980	1.000	0.980	1.000	0.990
keyboard	0.750	0.650	0.200	0.650	0.200	0.700	0.500	0.150	0.475	0.150
lamp	0.950	0.850	0.850	0.875	0.875	0.900	0.900	0.900	0.950	0.950
laptop	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.950	1.000	1.000
mantel	0.970	0.980	0.790	0.960	0.610	0.670	0.680	0.490	0.605	0.450
monitor	0.980	0.980	0.960	0.995	0.965	0.960	0.920	0.910	0.910	0.910
night_stand	0.477	0.465	0.326	0.366	0.151	0.221	0.186	0.151	0.163	0.128
person	0.950	1.000	1.000	1.000	0.975	0.800	0.900	0.900	0.900	0.850
piano	0.770	0.640	0.570	0.625	0.430	0.620	0.480	0.210	0.465	0.145
plant	0.960	0.960	0.970	0.970	0.965	0.710	0.720	0.740	0.725	0.720
radio	0.750	0.700	0.450	0.625	0.450	0.300	0.250	0.250	0.225	0.250
range_hood	0.940	0.950	0.840	0.930	0.765	0.880	0.760	0.630	0.665	0.555
sink	0.500	0.400	0.400	0.350	0.350	0.650	0.700	0.650	0.650	0.650
sofa	0.780	0.500	0.490	0.495	0.440	0.910	0.730	0.430	0.580	0.335
stairs	0.900	0.850	0.900	0.950	0.975	0.900	0.800	0.900	0.800	0.900
stool	0.650	0.600	0.600	0.600	0.650	0.700	0.650	0.700	0.700	0.700
table	0.700	0.640	0.600	0.640	0.635	0.770	0.760	0.720	0.755	0.745
tent	0.900	0.850	0.800	0.675	0.650	0.900	0.950	0.950	0.925	0.925
toilet	0.910	0.940	0.860	0.870	0.820	0.820	0.810	0.770	0.660	0.760
tv_stand	0.450	0.410	0.300	0.295	0.190	0.510	0.420	0.300	0.390	0.260
vase	0.820	0.770	0.780	0.745	0.735	0.740	0.670	0.670	0.565	0.575
wardrobe	0.300	0.300	0.100	0.100	0.050	0.250	0.050	0.050	0.000	0.000
xbox	0.550	0.150	0.200	0.050	0.050	0.550	0.400	0.400	0.025	0.000

Table A.9: Accuracy values for circle camera experiment.

	PointNet					PointNet++				
Viewing Angle	**Full**	**Half Front**	**Half Back**	**Quart. Front**	**Quart. Back**	**Full**	**Half Front**	**Half Back**	**Quart. Front**	**Quart. Back**
Overall acc.	0.774	0.679	0.625	0.531	0.481	0.719	0.660	0.540	0.585	0.467
Mean class acc.	0.753	0.675	0.641	0.516	0.491	0.706	0.642	0.568	0.577	0.494
airplane	1.000	1.000	1.000	1.000	0.995	1.000	1.000	1.000	0.940	0.990
bathtub	0.700	0.680	0.660	0.050	0.050	0.480	0.320	0.300	0.180	0.070
bed	0.640	0.680	0.490	0.230	0.135	0.630	0.590	0.450	0.545	0.450
bench	0.750	0.800	0.800	0.775	0.775	0.600	0.600	0.750	0.600	0.700
bookshelf	0.880	0.850	0.400	0.830	0.415	0.790	0.810	0.390	0.790	0.390
bottle	0.870	0.870	0.880	0.735	0.725	0.870	0.910	0.890	0.900	0.830
bowl	0.800	0.750	0.800	0.350	0.350	0.800	0.800	0.850	0.700	0.725
car	1.000	0.990	0.990	0.510	0.225	0.970	0.980	0.670	0.850	0.310
chair	0.960	0.970	0.970	0.975	0.955	0.930	0.930	0.950	0.930	0.905
cone	1.000	1.000	1.000	0.950	0.925	1.000	0.950	1.000	0.925	0.950
cup	0.750	0.800	0.750	0.000	0.000	0.900	0.600	0.650	0.525	0.350
curtain	0.850	0.950	0.900	0.950	0.950	0.850	0.900	0.900	0.900	0.900
desk	0.698	0.674	0.547	0.663	0.535	0.686	0.651	0.453	0.628	0.436
door	0.900	0.600	0.700	0.400	0.475	0.800	0.450	0.450	0.300	0.300
dresser	0.314	0.012	0.000	0.000	0.000	0.221	0.093	0.023	0.029	0.012
flower_pot	0.350	0.250	0.250	0.075	0.075	0.250	0.200	0.300	0.200	0.175
glass_box	0.820	0.650	0.700	0.005	0.000	0.980	0.670	0.150	0.125	0.005
guitar	0.980	0.980	0.990	0.995	0.990	0.860	0.740	0.800	0.460	0.700
keyboard	0.750	0.400	0.200	0.400	0.225	0.500	0.400	0.200	0.475	0.150
lamp	0.850	0.850	0.850	0.850	0.850	0.950	0.950	0.850	0.900	0.900
laptop	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
mantel	0.810	0.430	0.460	0.425	0.460	0.790	0.860	0.420	0.850	0.425
monitor	0.910	0.810	0.880	0.840	0.875	0.960	0.860	0.900	0.880	0.915
night_stand	0.186	0.105	0.093	0.105	0.116	0.256	0.186	0.140	0.134	0.105
person	0.850	0.850	0.850	0.900	0.900	1.000	0.950	0.950	0.900	0.950
piano	0.730	0.520	0.260	0.395	0.265	0.470	0.400	0.140	0.370	0.100
plant	0.660	0.690	0.700	0.685	0.725	0.720	0.820	0.790	0.835	0.825
radio	0.400	0.350	0.400	0.325	0.400	0.400	0.350	0.350	0.275	0.300
range_hood	0.930	0.680	0.460	0.210	0.200	0.820	0.660	0.710	0.660	0.525
sink	0.700	0.600	0.600	0.325	0.375	0.750	0.550	0.550	0.425	0.375
sofa	0.830	0.340	0.240	0.265	0.090	0.630	0.370	0.020	0.260	0.025
stairs	0.750	0.750	0.800	0.775	0.850	1.000	0.900	1.000	0.975	0.975
stool	0.650	0.700	0.700	0.700	0.725	0.700	0.700	0.650	0.650	0.600
table	0.710	0.660	0.650	0.690	0.680	0.660	0.650	0.670	0.625	0.585
tent	0.900	0.850	0.900	0.475	0.625	0.900	0.950	0.950	0.750	0.725
toilet	0.920	0.890	0.870	0.715	0.715	0.760	0.800	0.470	0.640	0.135
tv_stand	0.610	0.440	0.330	0.410	0.285	0.360	0.250	0.240	0.230	0.195
vase	0.850	0.860	0.820	0.640	0.645	0.790	0.740	0.690	0.680	0.675
wardrobe	0.150	0.050	0.000	0.025	0.000	0.200	0.150	0.000	0.050	0.025
xbox	0.700	0.650	0.750	0.000	0.075	0.000	0.000	0.050	0.000	0.050

Table A.10: Accuracy values for circle camera experiment (continued).

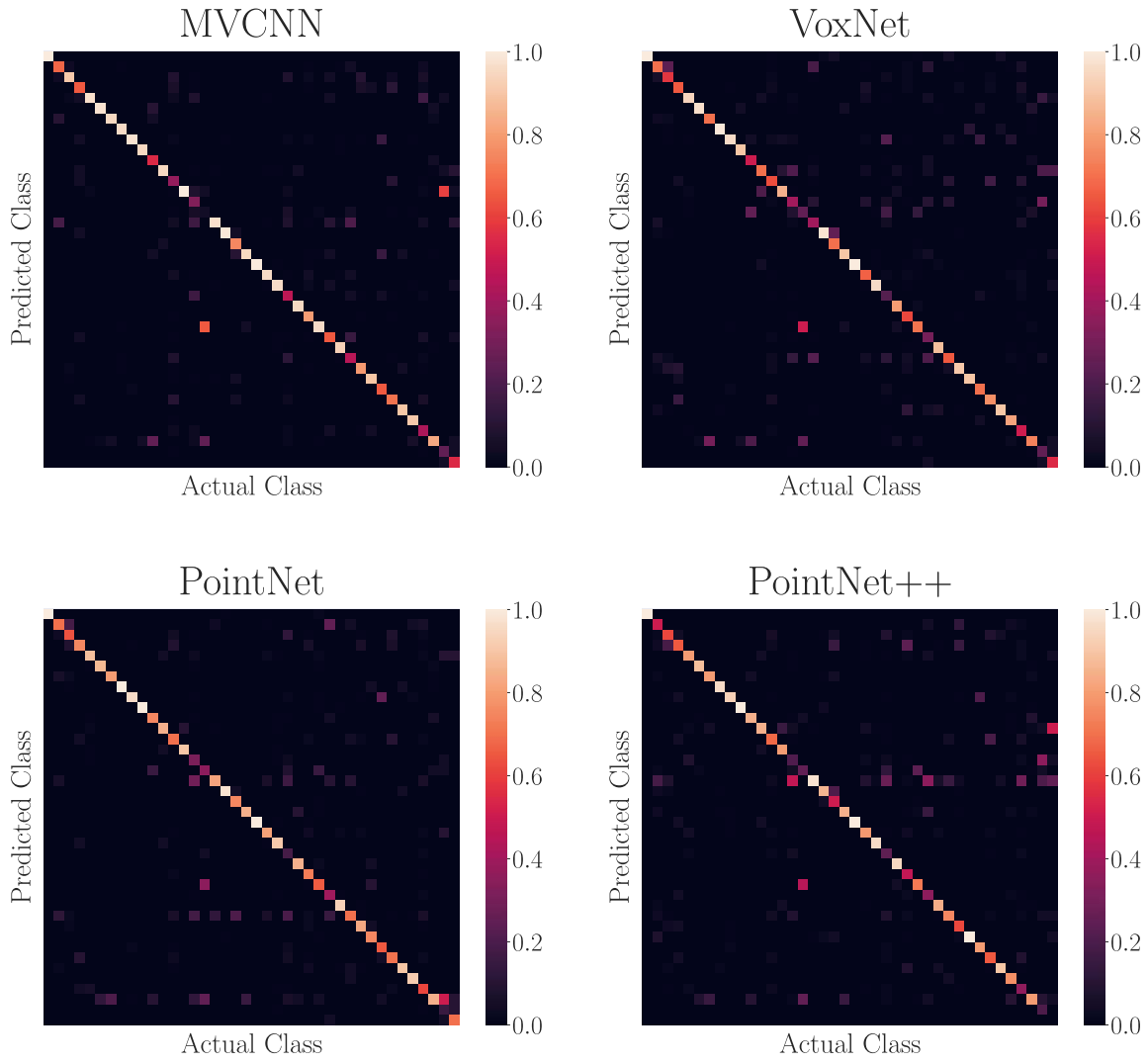


Figure A.8: Confusion matrices for full circle setup

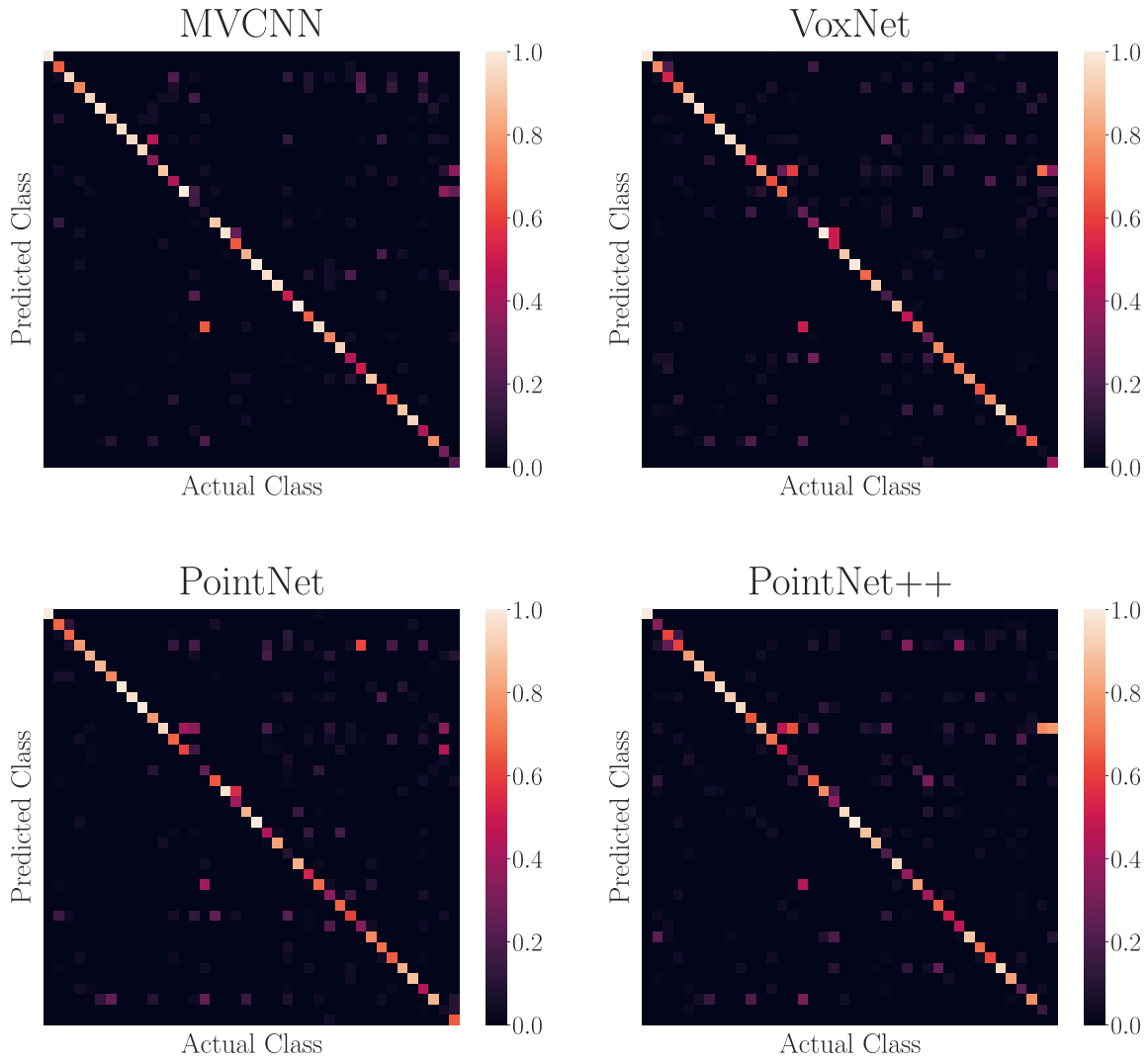


Figure A.9: Confusion matrices for half circle (front) setup

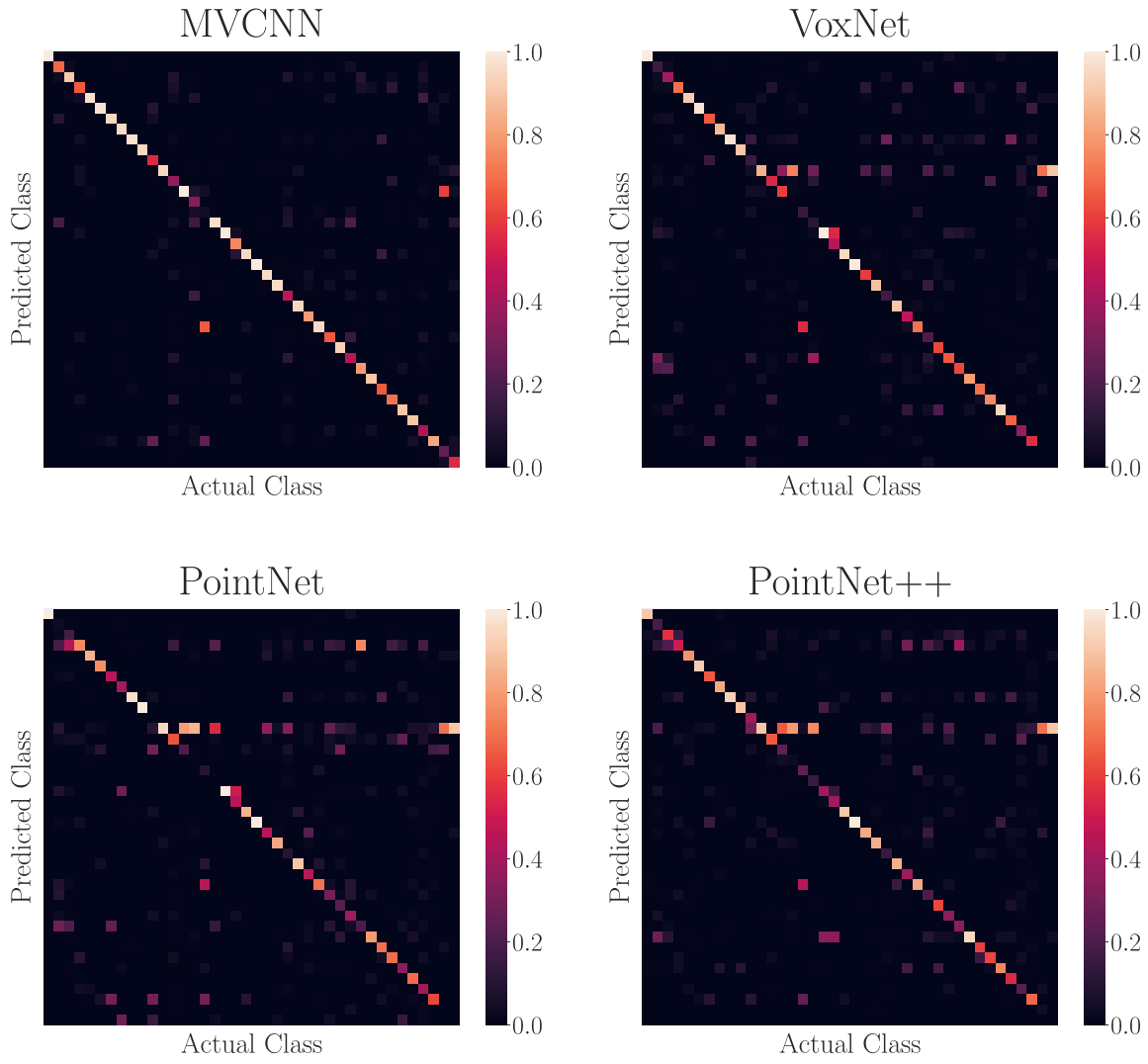


Figure A.10: Confusion matrices for quarter circle (front) setup

A.5 Results for impact of scale

The tables A.11 and A.12 show the results of the cut experiment corresponding to section 4.3.3 where the impact of scale is further investigated. For this experiment, the 3D models are cut and either rescaled or not to fit in a unit-sphere. Figures A.11 and A.12 show confusion matrices for 30% and 50% cut experiments with rescaling.

	MVCNN				VoxNet			
Cut amount	**30%**	**30% rescaled**	**50%**	**50% rescaled**	**30%**	**30% rescaled**	**50%**	**50% rescaled**
Overall acc.	0.838	0.894	0.710	0.765	0.780	0.767	0.773	0.515
Mean class acc.	0.831	0.861	0.710	0.727	0.735	0.730	0.730	0.452
airplane	1.000	1.000	0.990	1.000	1.000	0.996	1.000	0.860
bathtub	0.680	0.720	0.460	0.340	0.660	0.682	0.700	0.320
bed	0.890	0.990	0.740	0.870	0.750	0.788	0.700	0.410
bench	0.800	0.750	0.700	0.550	0.550	0.488	0.800	0.200
bookshelf	0.960	0.970	0.930	0.960	0.920	0.916	0.970	0.920
bottle	1.000	0.990	0.940	0.990	0.960	0.954	0.990	0.840
bowl	0.800	0.750	0.800	0.800	0.050	0.142	0.900	0.000
car	0.980	0.980	0.920	0.910	0.650	0.627	0.940	0.150
chair	0.970	0.960	0.790	0.850	0.970	0.963	0.990	0.680
cone	0.950	0.900	0.950	0.950	0.650	0.629	0.900	0.300
cup	0.500	0.500	0.550	0.300	0.650	0.713	0.500	0.100
curtain	0.950	0.950	0.950	0.900	0.750	0.771	0.850	0.650
desk	0.837	0.779	0.849	0.570	0.616	0.524	0.686	0.419
door	0.950	0.950	0.900	0.850	0.850	0.875	0.600	0.150
dresser	0.302	0.686	0.163	0.442	0.488	0.498	0.674	0.360
flower_pot	0.050	0.000	0.050	0.000	0.250	0.183	0.150	0.000
guitar	0.980	0.970	0.940	0.840	0.890	0.865	0.990	0.610
keyboard	1.000	1.000	1.000	0.950	0.950	0.929	0.000	0.550
lamp	0.900	0.900	0.900	0.800	0.850	0.804	0.900	0.750
laptop	1.000	1.000	0.600	0.700	1.000	0.938	0.950	0.350
mantel	0.940	0.960	0.760	0.840	0.880	0.867	0.970	0.700
monitor	0.980	0.970	0.940	0.870	0.940	0.933	0.990	0.670
night_stand	0.488	0.779	0.267	0.512	0.628	0.600	0.372	0.174
person	1.000	1.000	0.850	0.700	0.800	0.821	0.850	0.750
piano	0.810	0.920	0.610	0.730	0.870	0.803	0.870	0.520
plant	0.930	0.930	0.970	0.950	0.720	0.718	0.760	0.630
radio	0.900	0.900	0.650	0.850	0.550	0.571	0.550	0.250
range_hood	0.870	0.940	0.750	0.870	0.840	0.828	0.810	0.430
sink	0.850	0.900	0.650	0.500	0.550	0.583	0.700	0.200
sofa	0.890	0.920	0.730	0.730	0.900	0.887	0.870	0.480
stairs	0.950	0.950	0.900	0.950	0.800	0.813	0.900	0.550
stool	0.750	0.800	0.550	0.800	0.700	0.704	0.700	0.300
table	0.660	0.810	0.430	0.700	0.850	0.841	0.550	0.760
tent	0.950	0.850	0.800	0.700	0.950	0.954	0.950	0.650
toilet	0.990	0.990	0.770	0.800	0.970	0.930	0.930	0.810
tv_stand	0.520	0.840	0.200	0.510	0.700	0.728	0.610	0.520
vase	0.720	0.780	0.540	0.570	0.790	0.791	0.770	0.340
wardrobe	0.850	0.700	0.650	0.500	0.600	0.679	0.300	0.250
xbox	0.850	0.800	0.650	0.500	0.500	0.538	0.450	0.350

Table A.11: Accuracy values for the cut and rescaling experiment

	PointNet				PointNet++			
Cut amount	**30%**	**30% rescaled**	**50%**	**50% rescaled**	**30%**	**30% rescaled**	**50%**	**50% rescaled**
Overall acc.	0.820	0.784	0.248	0.422	0.866	0.861	0.762	0.747
Mean class acc.	0.797	0.764	0.260	0.404	0.843	0.833	0.711	0.688 h
airplane	1.000	0.970	0.180	0.320	1.000	1.000	0.990	0.990
bathtub	0.780	0.700	0.020	0.260	0.800	0.780	0.420	0.360
bed	0.810	0.790	0.020	0.270	0.860	0.840	0.740	0.700
bench	0.650	0.650	0.200	0.250	0.700	0.650	0.550	0.500
bookshelf	0.930	0.890	0.530	0.750	0.920	0.950	0.920	0.910
bottle	0.960	0.950	0.920	0.890	0.960	0.950	0.840	0.850
bowl	0.900	0.450	0.050	0.000	0.850	0.750	0.250	0.250
car	0.920	0.830	0.100	0.230	0.970	0.960	0.890	0.890
chair	0.970	0.950	0.230	0.440	0.950	0.950	0.810	0.770
cone	0.850	0.700	0.800	0.000	0.950	0.900	0.800	0.800
cup	0.750	0.750	0.000	0.100	0.750	0.750	0.300	0.150
curtain	0.900	0.900	0.550	0.800	0.900	0.900	0.900	0.750
desk	0.674	0.721	0.209	0.314	0.895	0.872	0.814	0.802
door	0.850	0.900	0.150	0.300	0.800	0.850	0.700	0.800
dresser	0.581	0.512	0.000	0.174	0.674	0.628	0.581	0.547
flower_pot	0.150	0.200	0.000	0.050	0.350	0.300	0.150	0.150
glass_box	0.610	0.390	0.050	0.120	0.850	0.860	0.800	0.860
guitar	0.990	0.880	0.960	0.610	0.920	0.950	0.760	0.690
keyboard	1.000	1.000	0.000	0.700	1.000	1.000	0.650	0.600
lamp	0.800	0.850	0.700	0.700	0.850	0.850	0.800	0.700
laptop	0.950	1.000	0.100	0.550	1.000	1.000	0.600	0.550
mantel	0.840	0.820	0.120	0.500	0.980	0.970	0.910	0.920
monitor	0.940	0.970	0.100	0.460	0.980	0.980	0.810	0.850
night_stand	0.628	0.570	0.000	0.140	0.640	0.547	0.488	0.500
person	0.800	0.800	0.850	0.750	0.900	0.950	0.850	0.850
piano	0.870	0.870	0.160	0.470	0.920	0.930	0.870	0.870
plant	0.700	0.760	0.430	0.470	0.730	0.740	0.870	0.830
radio	0.550	0.700	0.350	0.300	0.700	0.650	0.700	0.550
range_hood	0.920	0.830	0.100	0.480	0.950	0.980	0.890	0.890
sink	0.700	0.600	0.200	0.350	0.750	0.700	0.500	0.350
sofa	0.870	0.920	0.080	0.380	0.920	0.920	0.670	0.640
stairs	0.800	0.750	0.250	0.500	0.850	0.900	0.900	0.950
stool	0.700	0.700	0.300	0.450	0.800	0.750	0.850	0.800
table	0.580	0.520	0.130	0.560	0.700	0.690	0.610	0.680
tent	0.950	0.900	0.400	0.450	0.950	0.950	0.900	0.950
toilet	0.980	0.960	0.280	0.640	0.970	0.980	0.870	0.840
tv_stand	0.840	0.800	0.150	0.520	0.820	0.830	0.730	0.650
vase	0.750	0.700	0.340	0.260	0.770	0.780	0.710	0.680
wardrobe	0.550	0.650	0.000	0.150	0.700	0.700	0.600	0.600
xbox	0.900	0.750	0.400	0.500	0.750	0.700	0.450	0.500

Table A.12: Accuracy values for the cut and rescaling experiment (continued)

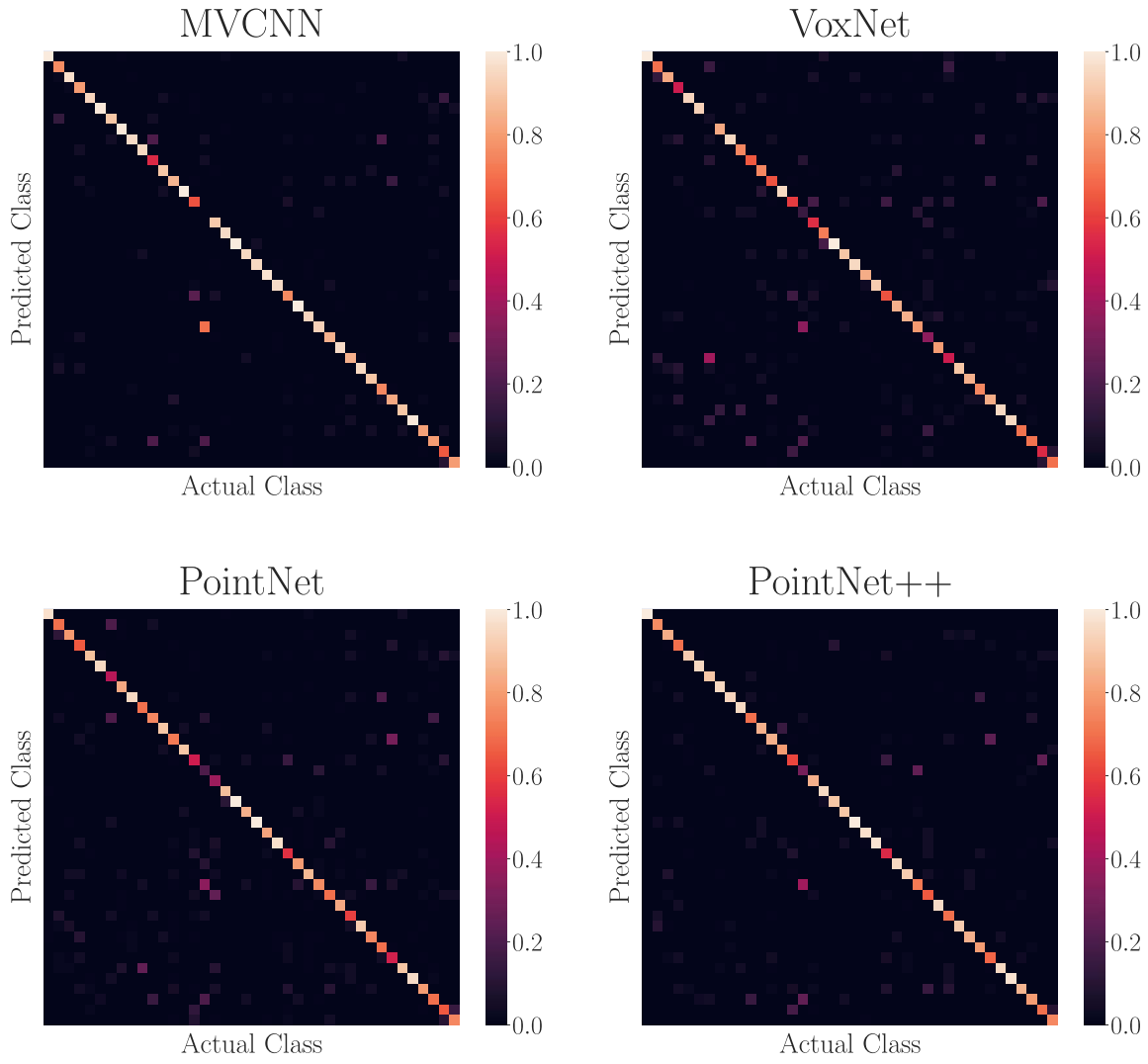


Figure A.11: Confusion matrices for cut experiment with 30% cut and rescaled models

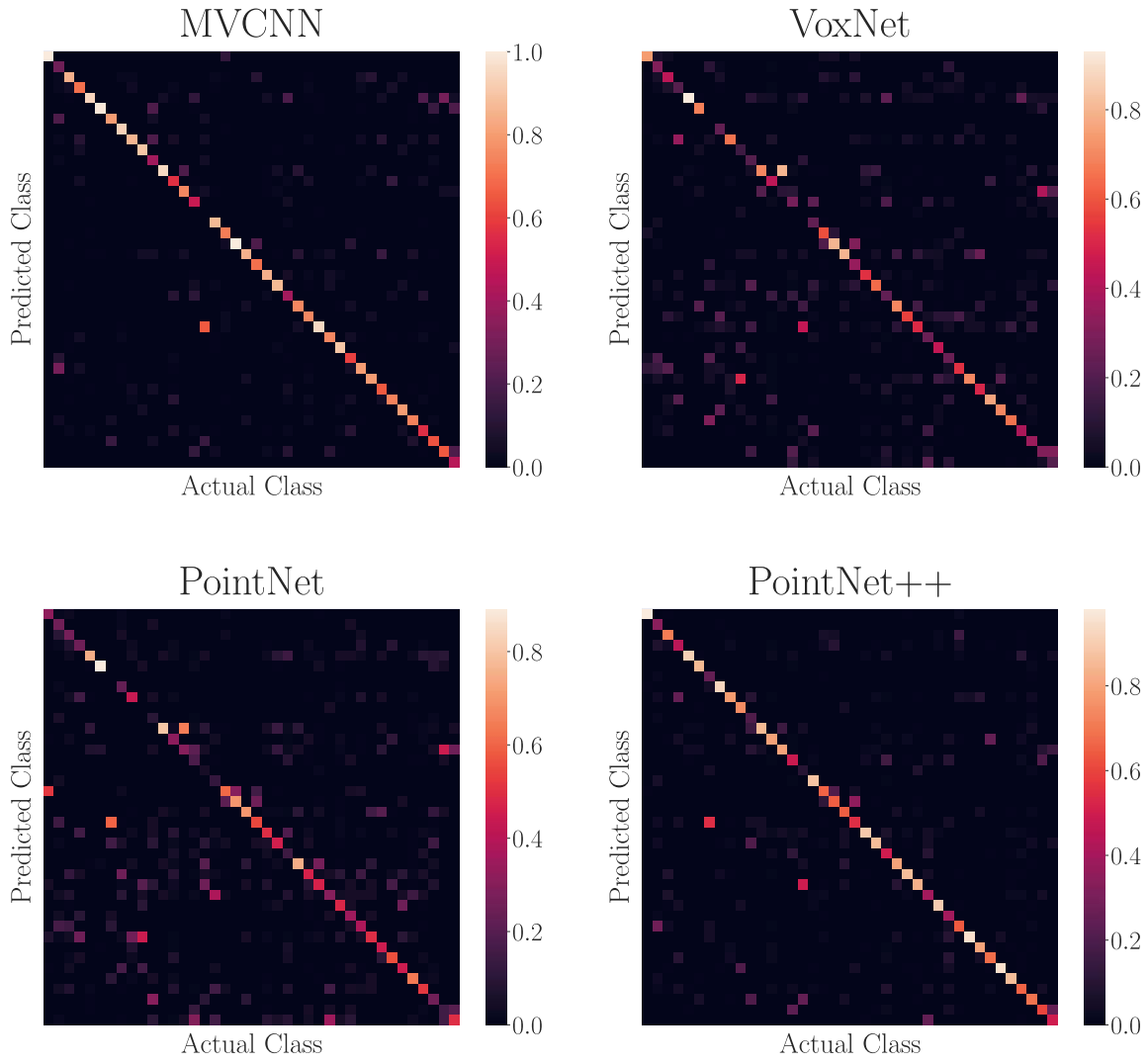


Figure A.12: Confusion matrices for cut experiment with 50% cut and rescaled models

A.6 Result for floor segmentation error

The results for the segmentation error experiment in section 4.3.4 are shown in table A.13. In this experiment, a slightly bigger plane is attached underneath each 3D model to simulate non-ideal segmentation. The corresponding confusion matrices are shown in figure A.13.

	MVCNN	VoxNet	PointNet	PointNet++
Overall acc.	0.777	0.645	0.242	0.253
Mean class acc.	0.783	0.665	0.232	0.253
airplane	0.080	0.020	0.000	0.000
bathtub	0.920	0.700	0.220	0.220
bed	0.980	0.910	0.000	0.120
bench	0.700	0.400	0.050	0.150
bookshelf	0.940	0.940	0.870	0.870
bottle	0.980	0.920	0.020	0.030
bowl	0.400	0.650	0.050	0.200
car	0.960	0.700	0.000	0.000
chair	0.940	0.980	0.000	0.050
cone	0.850	0.950	0.000	0.200
cup	0.800	0.750	0.050	0.600
curtain	0.800	0.600	0.800	0.650
desk	0.256	0.116	0.116	0.047
door	1.000	0.950	0.850	0.800
dresser	0.674	0.640	0.326	0.360
flower_pot	0.050	0.200	0.000	0.100
glass_box	0.960	0.290	0.520	0.540
guitar	0.140	0.070	0.000	0.050
keyboard	1.000	1.000	0.400	0.550
lamp	0.950	0.850	0.200	0.050
laptop	1.000	1.000	0.000	0.100
mantel	0.870	0.670	0.130	0.290
monitor	0.960	0.930	0.010	0.100
night_stand	0.593	0.477	0.500	0.407
person	0.850	0.800	0.000	0.050
piano	0.830	0.690	0.170	0.100
plant	0.880	0.440	0.350	0.510
radio	0.900	0.500	0.000	0.000
range_hood	0.950	0.780	0.000	0.000
sink	0.800	0.550	0.300	0.000
sofa	0.950	0.920	0.040	0.010
stairs	1.000	0.750	0.050	0.400
stool	0.850	0.600	0.300	0.300
table	0.360	0.530	0.650	0.240
tent	0.950	1.000	0.000	0.000
toilet	0.990	0.780	0.020	0.210
tv_stand	0.930	0.650	0.690	0.710
vase	0.840	0.760	0.830	0.700
wardrobe	0.700	0.500	0.400	0.300
xbox	0.750	0.650	0.350	0.100

Table A.13: Accuracy values for floor segmentation experiment

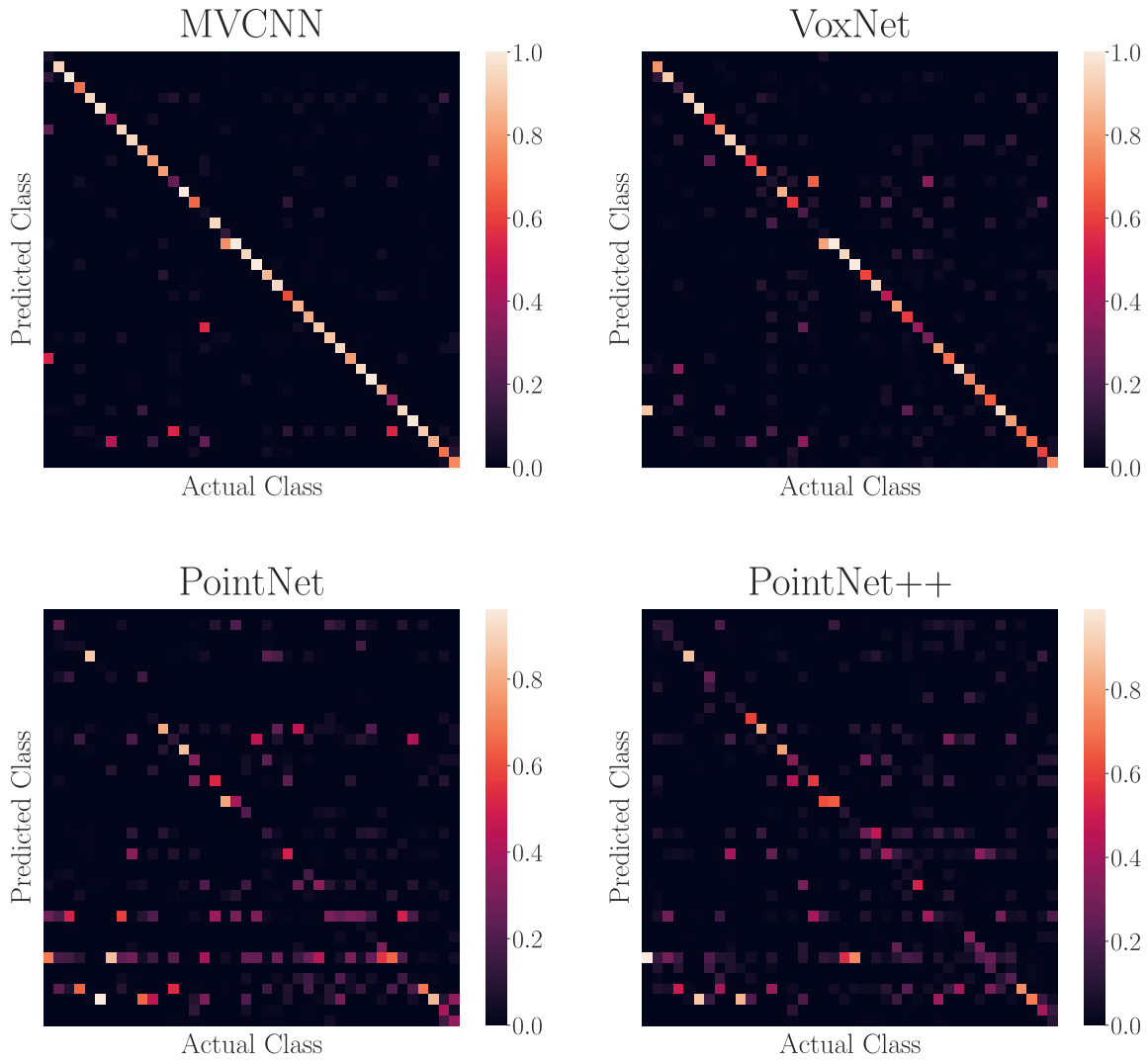


Figure A.13: Confusion matrices floor segmentation experiment

A.7 Result for ScanObjectNN

Tables A.14 to A.17 show the results of the experiments with ScanObject data. In order to investigate the Sim2Real Gap two new datasets are created. They are named ModelNet11 (MN11) and ScanObjectNN11 (SO11) and are listed in table 4.1. Note that both datasets have a significantly different ratio for objects per class. ScanObjectNN11 is available with two variants, where for one, the objects are cut out manually and for the other, the objects are cut out very roughly. This means that it contains a lot of the background or other objects.

The models for this much smaller datasets are trained with the unmodified CAD data and with the TSDF recreated data.

Confusion matrices are shown in figures A.14 to A.16 for models trained with the TSDF reconstructed data.

	MVCNN							
Training	**MN11 CAD**				**MN11 TSDF**			
Evaluation	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**
Overall acc.	0.951	0.909	0.526	0.497	0.868	0.932	0.672	0.598
Mean class acc.	0.949	0.898	0.468	0.458	0.875	0.935	0.627	0.586
bed	1.000	0.990	0.818	0.909	0.890	1.000	0.818	0.909
cabinet	0.943	0.934	0.133	0.133	0.858	0.962	0.427	0.240
chair	0.950	0.943	1.000	0.936	0.914	0.900	0.987	0.897
desk	0.837	0.767	0.100	0.000	0.605	0.872	0.167	0.033
display	1.000	0.990	0.762	0.690	0.940	1.000	0.643	0.595
door	0.950	0.900	0.286	0.619	0.950	0.950	0.786	0.929
shelf	0.980	0.890	0.612	0.531	0.910	0.960	0.959	0.918
sink	0.950	0.800	0.000	0.042	0.950	0.950	0.125	0.375
sofa	0.970	0.970	0.714	0.667	0.960	0.980	0.881	0.762
table	0.860	0.720	0.667	0.389	0.690	0.710	0.574	0.315
toilet	1.000	0.970	0.059	0.118	0.960	1.000	0.529	0.471

Table A.14: Accuracy values for ScanObjectNN11 (SO11) dataset without and with Background data (BG) trained with unmodified CAD and TSDF reconstructed data from the ModelNet11 (MN11) dataset

	VoxNet							
Training	**MN11 CAD**				**MN11 TSDF**			
Evaluation	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**
Overall acc.	0.908	0.904	0.560	0.566	0.891	0.914	0.545	0.552
Mean class acc.	0.890	0.882	0.517	0.548	0.879	0.901	0.498	0.544
bed	0.980	0.970	0.318	0.682	0.990	0.990	0.273	0.636
cabinet	0.887	0.953	0.093	0.093	0.708	0.925	0.067	0.080
chair	0.921	0.929	0.974	0.949	0.929	0.943	0.962	0.923
desk	0.709	0.709	0.167	0.367	0.698	0.721	0.233	0.267
display	0.980	0.960	0.810	0.929	0.970	0.970	0.738	0.833
door	0.950	0.950	1.000	0.976	0.950	0.950	1.000	0.929
shelf	0.980	0.920	0.510	0.612	0.970	0.930	0.510	0.571
sink	0.650	0.600	0.458	0.375	0.700	0.750	0.500	0.458
sofa	0.950	0.960	0.286	0.381	0.980	0.980	0.333	0.595
table	0.810	0.780	0.778	0.426	0.800	0.790	0.741	0.333
toilet	0.970	0.970	0.294	0.235	0.970	0.960	0.118	0.353

Table A.15: Accuracy values for ScanObjectNN11 (SO11) dataset without and with Background data (BG) trained with unmodified CAD and TSDF reconstructed data from the ModelNet11 (MN11) dataset (continued)

	PointNet							
Training	**CAD trained**				**TSDF trained**			
Evaluation	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**
Overall acc.	0.931	0.808	0.343	0.320	0.767	0.919	0.540	0.540
Mean class acc.	0.918	0.786	0.297	0.283	0.764	0.903	0.542	0.557
bed	0.950	0.860	0.045	0.000	0.880	1.000	0.273	0.545
cabinet	0.925	0.915	0.000	0.000	0.651	0.925	0.013	0.147
chair	0.943	0.964	0.795	0.744	0.943	0.936	0.872	0.923
desk	0.884	0.512	0.033	0.033	0.605	0.814	0.333	0.300
display	0.970	0.790	0.190	0.146	0.910	0.930	0.786	0.786
door	1.000	0.900	1.000	0.881	0.950	0.950	0.976	0.976
shelf	0.990	0.890	0.104	0.163	0.920	0.920	0.438	0.653
sink	0.700	0.550	0.417	0.500	0.650	0.700	0.583	0.375
sofa	0.950	0.520	0.073	0.100	0.940	0.930	0.439	0.571
table	0.810	0.800	0.547	0.426	0.000	0.830	0.604	0.389
toilet	0.980	0.940	0.059	0.118	0.960	1.000	0.647	0.471

Table A.16: Accuracy values for ScanObjectNN (SO11) dataset without and with Background data (BG) trained with unmodified CAD and TSDF reconstructed data from the ModelNet11 (MN11) dataset (continued)

	PointNet++							
Training	**MN11 CAD**				**MN11 TSDF**			
Evaluation	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**	**MN11 CAD**	**MN11 TSDF**	**SO11**	**SO11+BG**
Overall acc.	0.937	0.923	0.543	0.484	0.893	0.936	0.558	0.573
Mean class acc.	0.930	0.917	0.516	0.454	0.892	0.933	0.525	0.558
bed	0.970	0.950	0.500	0.545	0.930	0.940	0.591	0.636
cabinet	0.953	0.925	0.120	0.133	0.717	0.943	0.160	0.080
chair	0.943	0.957	0.974	0.897	0.943	0.957	0.962	0.846
desk	0.826	0.814	0.567	0.367	0.802	0.884	0.400	0.467
display	0.980	0.980	0.500	0.595	0.990	0.990	0.548	0.707
door	0.950	0.950	1.000	0.881	0.950	0.950	1.000	0.952
shelf	0.960	0.920	0.306	0.653	0.980	0.960	0.388	0.531
sink	0.850	0.850	0.417	0.375	0.850	0.900	0.250	0.500
sofa	0.950	0.920	0.500	0.286	0.930	0.960	0.667	0.450
table	0.850	0.860	0.611	0.204	0.730	0.780	0.574	0.370
toilet	1.000	0.960	0.176	0.059	0.990	1.000	0.235	0.588

Table A.17: Accuracy values for ScanObjectNN (SO11) dataset without and with Background data (BG) trained with unmodified CAD and TSDF reconstructed data from the ModelNet11 (MN11) dataset (continued)

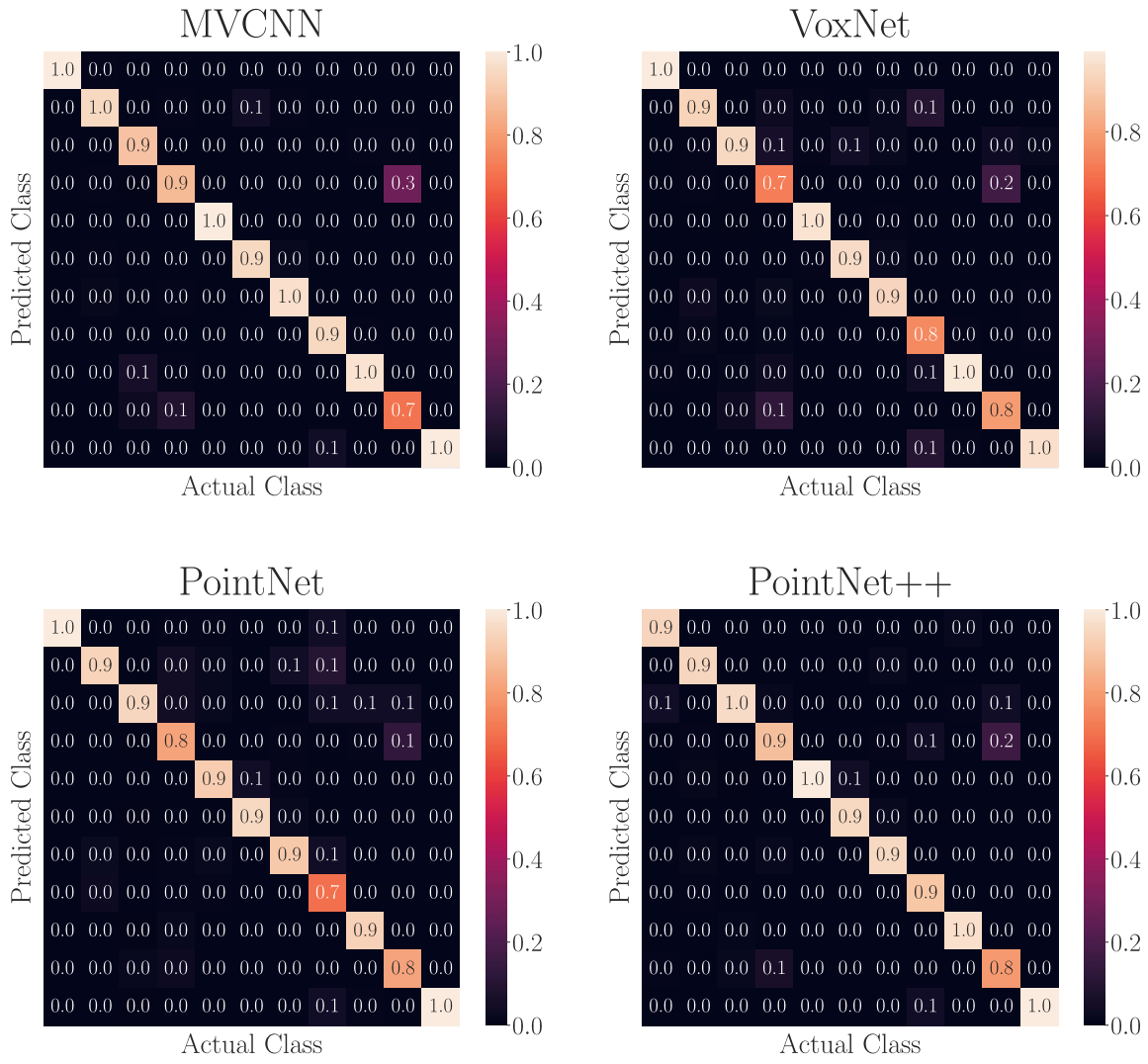


Figure A.14: Confusion matrices for MN11 dataset trained with TSDF reconstructed data

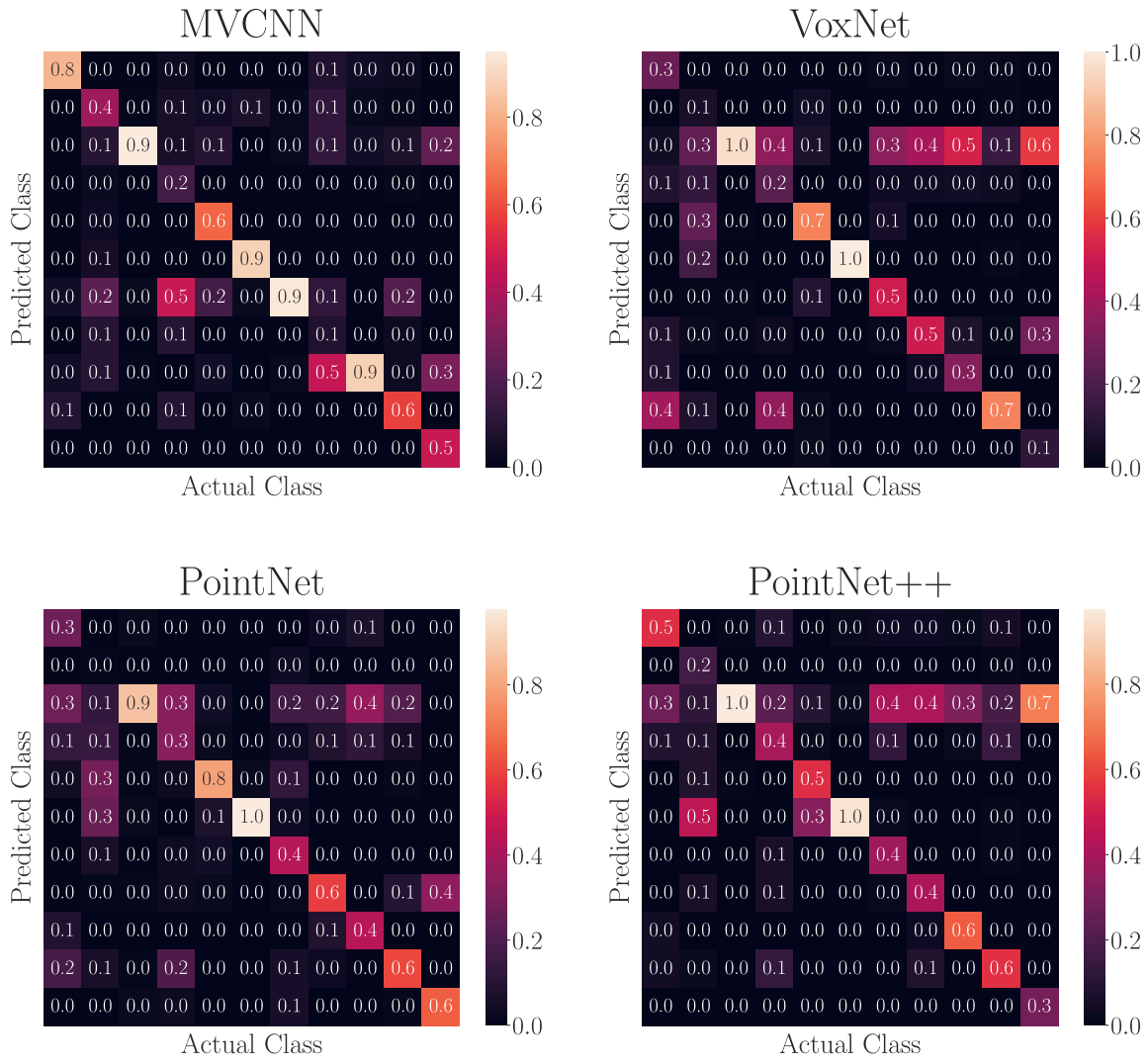


Figure A.15: Confusion matrices for ScanObjectNN11 dataset without background trained with TSDF reconstructed data

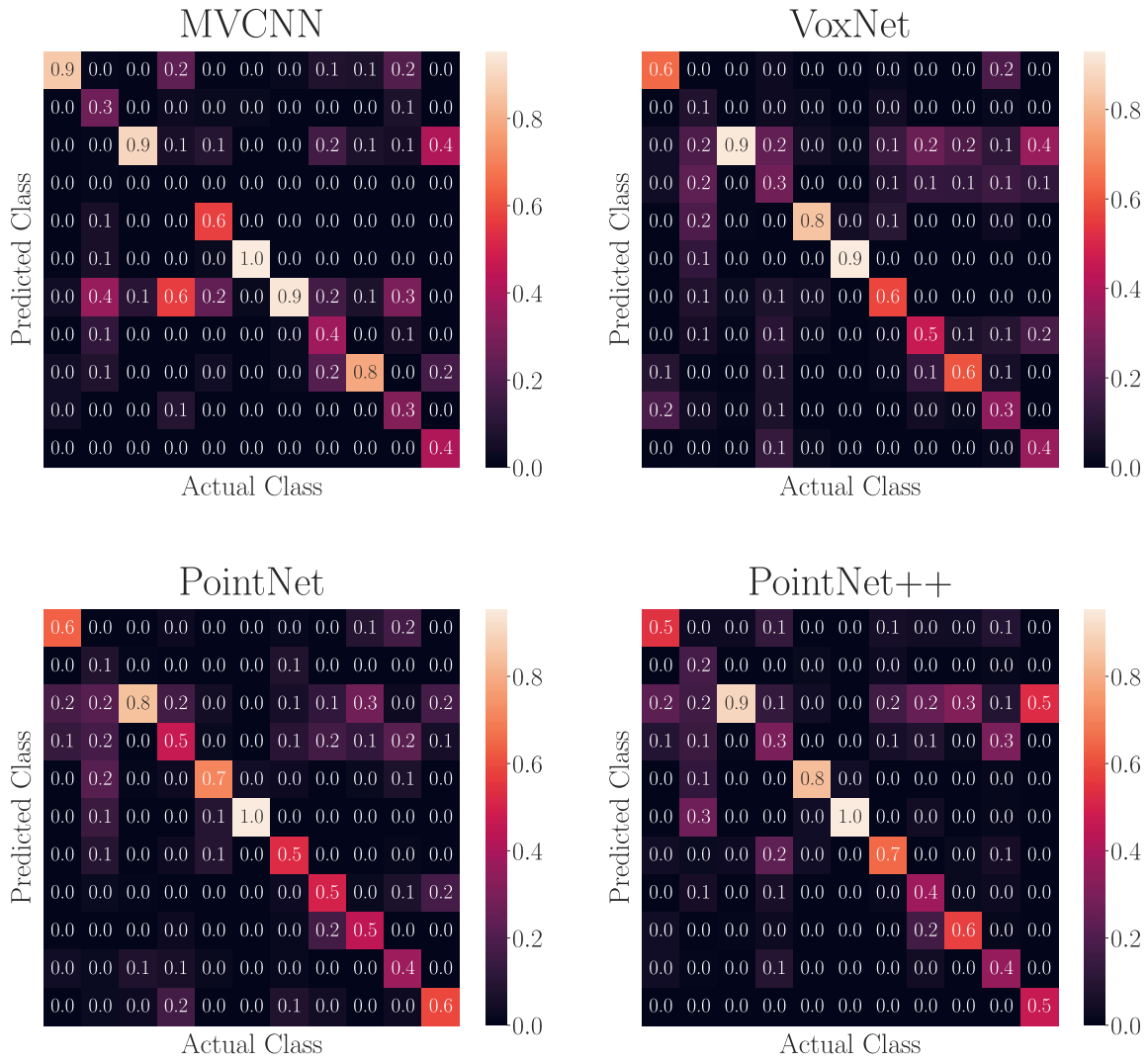


Figure A.16: Confusion matrices for ScanObjectNN11 with background trained with TSDF reconstructed data

A.8 Results for Test data degradation

The results for the test data degradation experiment in section 4.5 are shown in table A.18 and in the confusion matrices in figure A.17.

# Black Images	1	2	3	6	8	9	10	11	12
Overall acc.	0.909	0.912	0.911	0.912	0.908	0.903	0.885	0.835	0.041
Mean class acc.	0.882	0.883	0.882	0.879	0.878	0.872	0.867	0.817	0.025
airplane	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.990	0.000
bathtub	0.920	0.920	0.920	0.940	0.900	0.920	0.880	0.760	0.000
bed	0.990	1.000	1.000	1.000	1.000	0.990	0.970	0.950	0.000
bench	0.750	0.750	0.750	0.750	0.800	0.800	0.800	0.750	0.000
bookshelf	0.920	0.920	0.920	0.920	0.930	0.910	0.870	0.860	0.000
bottle	0.990	0.990	0.990	0.990	0.990	0.990	0.990	0.990	1.000
bowl	0.900	0.900	0.900	0.900	0.900	0.900	0.900	0.900	0.000
car	1.000	1.000	0.990	0.980	0.990	0.980	0.970	0.900	0.000
chair	0.980	0.970	0.980	0.980	0.970	0.980	0.960	0.910	0.000
cone	0.950	0.950	0.950	0.950	0.950	0.950	0.950	0.950	0.000
cup	0.700	0.700	0.700	0.700	0.700	0.700	0.700	0.750	0.000
curtain	0.950	0.950	0.950	0.950	0.950	0.950	0.900	0.850	0.000
desk	0.860	0.895	0.872	0.849	0.849	0.802	0.756	0.605	0.000
door	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000
dresser	0.767	0.756	0.756	0.791	0.779	0.767	0.721	0.547	0.000
flower_pot	0.000	0.000	0.000	0.000	0.050	0.050	0.050	0.050	0.000
glass_box	0.970	0.980	0.970	0.970	0.970	0.970	0.970	0.980	0.000
guitar	0.990	1.000	1.000	1.000	0.990	0.990	0.990	0.940	0.000
keyboard	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000
lamp	0.900	0.900	0.900	0.900	0.900	0.850	0.900	0.850	0.000
laptop	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.000
mantel	0.970	0.980	0.970	0.970	0.970	0.970	0.950	0.920	0.000
monitor	0.980	0.980	0.970	0.980	0.950	0.940	0.940	0.840	0.000
night_stand	0.767	0.767	0.791	0.802	0.779	0.791	0.756	0.628	0.000
person	1.000	1.000	0.950	0.950	1.000	0.950	0.950	0.900	0.000
piano	0.930	0.920	0.930	0.930	0.910	0.890	0.850	0.790	0.000
plant	0.920	0.920	0.920	0.910	0.930	0.920	0.910	0.900	0.000
radio	0.950	0.950	0.950	0.900	0.900	0.850	1.000	0.900	0.000
range_hood	0.940	0.960	0.950	0.960	0.940	0.940	0.930	0.900	0.000
sink	0.900	0.900	0.900	0.900	0.850	0.800	0.850	0.800	0.000
sofa	0.960	0.960	0.960	0.960	0.960	0.950	0.830	0.760	0.000
stairs	0.950	0.900	0.950	0.800	0.750	0.800	0.850	0.600	0.000
stool	0.750	0.750	0.750	0.750	0.800	0.800	0.800	0.750	0.000
table	0.760	0.730	0.760	0.790	0.780	0.800	0.770	0.690	0.000
tent	0.950	0.950	0.950	0.900	0.850	0.850	0.950	0.850	0.000
toilet	0.990	1.000	1.000	0.990	0.990	0.990	0.980	0.960	0.000
tv_stand	0.810	0.850	0.820	0.820	0.810	0.810	0.780	0.780	0.000
vase	0.810	0.820	0.820	0.820	0.820	0.820	0.810	0.790	0.000
wardrobe	0.650	0.650	0.700	0.700	0.750	0.750	0.750	0.700	0.000
xbox	0.750	0.750	0.700	0.750	0.750	0.750	0.750	0.700	0.000

Table A.18: Accuracy values for MVCNN only experiment where a certain number of views was replaced by entire black images

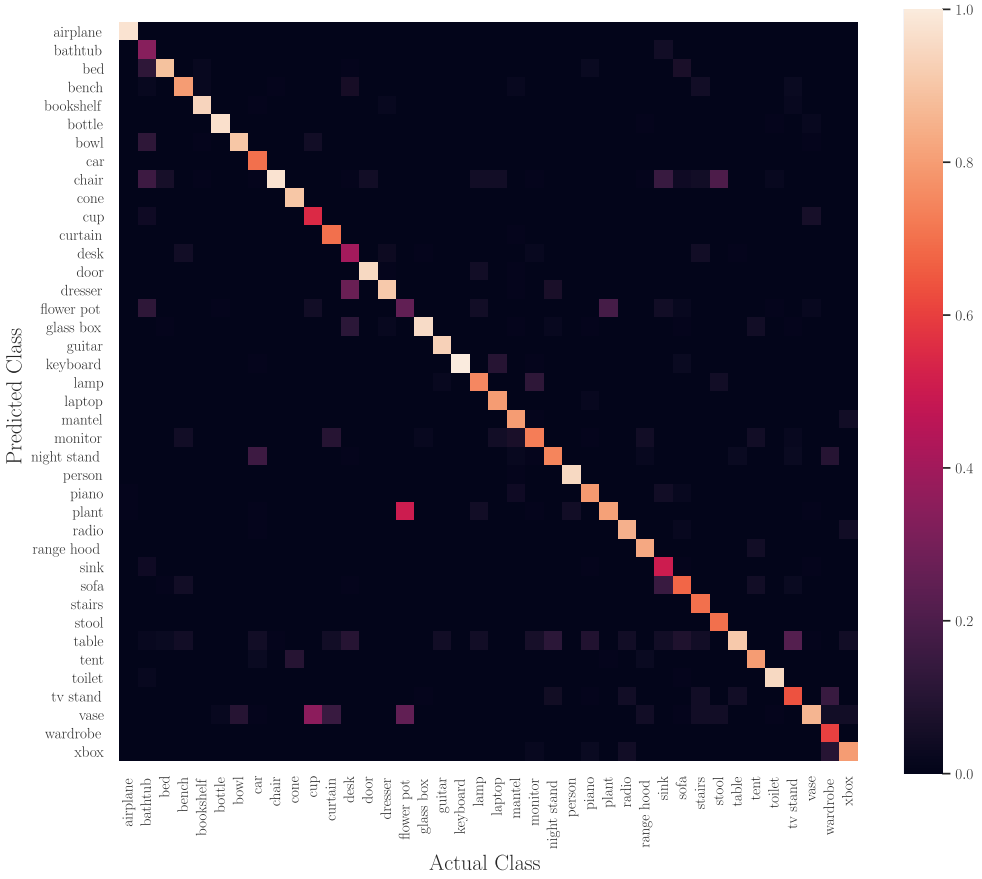


Figure A.17: Confusion matrix for MVCNN only experiment where 11 views of 12 views where replaced by black images

A.9 Repositories

A list of the most important code repositories used in this thesis is shown in table A.19.

Title	URL	Description
MVCNN_pytorch	https://github.com/jongchyisu/mvcnn_pytorch	Implementation of MVCNN based on PyTorch [64] by the original author
VoxNet.pytorch	https://github.com/MonteYang/VoxNet.pytorch	Implementation of VoxNet based on PyTorch
VoxNet	https://github.com/dimatura/voxnet	Original implementation of VoxNet based on Theano and Lasagne
PointNet	https://github.com/charlesq34/pointnet	Original implementation of PointNet based on TensorFlow[65]
PointNet++	https://github.com/charlesq34/pointnet2	Original implementation of PointNet++ based on TensorFlow[65]
ScanObjectNN	https://github.com/hkust-vgd/scanobjectnn	Description of ScanObjectNN dataset and model
Open3D	https://github.com/isl-org/Open3D	Library used for various 3D data manipulation including the TSDF reconstruction pipeline
Trimesh	https://github.com/mikedh/trimesh	Library used for various 3D data manipulation

Table A.19: A list of the most significant repositories.

Bibliography

[1] J. Straub, T. Whelan, L. Ma, Y. Chen, E. Wijmans, S. Green, J. J. Engel, R. Mur-Artal, C. Ren, S. Verma, A. Clarkson, M. Yan, B. Budge, Y. Yan, X. Pan, J. Yon, Y. Zou, K. Leon, N. Carter, J. Briales, T. Gillingham, E. Mueggler, L. Pesqueira, M. Savva, D. Batra, H. M. Strasdat, R. De Nardi, M. Goesele, S. Lovegrove, and R. Newcombe, *The Replica Dataset: A Digital Replica of Indoor Spaces*, 2019. [Online]. Available: `https://arxiv.org/abs/1906.05797`.

[2] W. Wohlkinger and M. Vincze, „Ensemble of shape functions for 3D object classification," Dec. 2011.

[3] X.-F. Han, S.-J. Sun, X.-Y. Song, and G.-Q. Xiao, *3D Point Cloud Descriptors in Hand-crafted and Deep Learning Age: State-of-the-Art*, 2020. arXiv: `1802.02297 [cs.CV]`.

[4] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, „PointNet++ deep hierarchical feature learning on point sets in a metric space," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 5105–5114.

[5] J.-C. Su, M. Gadelha, R. Wang, and S. Maji, „A deeper look at 3d shape classifiers," in *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*, 2018, pp. 0–0.

[6] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, „Dynamic graph cnn for learning on point clouds," *Acm Transactions On Graphics (tog)*, vol. 38, no. 5, pp. 1–12, 2019.

[7] Y. Li, R. Bu, M. Sun, W. Wu, X. Di, and B. Chen, „PointCNN: Convolution on χ-transformed points," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, 2018, pp. 828–838.

[8] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, „KPConv: Flexible and Deformable Convolution for Point Clouds," *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[9] „Advances in Computer Vision," *Advances in Intelligent Systems and Computing*, 2020, ISSN: 2194-5365. [Online]. Available: `http://dx.doi.org/10.1007/978-3-030-17795-9`.

[10] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, „Pointnet: Deep learning on point sets for 3d classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 652–660.

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

[12] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, „Multi-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 945–953.

[13] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, „3D ShapeNets: A Deep Representation for Volumetric Shapes," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1912–1920.

[14] M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen, and S.-K. Yeung, „Revisiting Point Cloud Classification: A New Benchmark Dataset and Classification Model on Real-World Data," in *International Conference on Computer Vision (ICCV)*, 2019.

[15] R. Rohrböck, J.-B. Weibel, and M. Vincze, „Analysis of 3D shape representations in presence of corrupted data," in *ARW*, Vortrag: Austrian Robotics Workshop 2021, Wien; 2021-06-10 – 2021-06-11, 2021, ISBN: 978-3-200-07799-7.

[16] J.-B. Weibel, R. Rohrböck, and M. Vincze, „Measuring the Sim2Real Gap in 3D Object Classification for Different 3D Data Representation," in *Computer Vision Systems. ICVS 2021. Lecture Notes in Computer Science*, Vortrag: 13th International Conference on Computer Vision Systems, Wien; 2021-09-22 – 2021-09-24, 2021.

[17] R. B. Rusu and S. Cousins, „3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China: IEEE, 2011.

[18] F. Bernardini, J. Mittleman, H. Rushmeier, C. Silva, and G. Taubin, „The ball-pivoting algorithm for surface reconstruction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 5, no. 4, pp. 349–359, 1999.

[19] R. Hanocka, A. Hertz, N. Fish, R. Giryes, S. Fleishman, and D. Cohen-Or, „MeshCNN," *ACM Transactions on Graphics*, vol. 38, no. 4, pp. 1–12, 2019. [Online]. Available: `https://doi.org/10.1145%2F3306346.3322959`.

[20] B. T. Phong, „Illumination for Computer Generated Pictures," *Commun. ACM*, vol. 18, no. 6, pp. 311–317, 1975, ISSN: 0001-0782. [Online]. Available: `https://doi.org/10.1145/360825.360839`.

[21] B. Smith, *Blinn Vectors*, `https://commons.wikimedia.org/wiki/File:Blinn_Vectors.svg`, Accessed: 2022-09-01.

[22] M. Kraus, *Phong components*, `https://commons.wikimedia.org/wiki/File:Phong_components_version_4.png`, Accessed: 2022-09-01.

[23] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, ser. Adaptive Computation and Machine Learning series. MIT Press, 2016, ISBN: 9780262035613. [Online]. Available: `https://books.google.at/books?id=Np9SDQAAQBAJ`.

[24] *Papers with code - machine learning datasets*. [Online]. Available: https://paperswithcode.com/datasets?mod=3d&page=1.

[25] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu, „ShapeNet: An Information-Rich 3D Model Repository," Stanford University — Princeton University — Toyota Technological Institute at Chicago, Tech. Rep. arXiv:1512.03012 [cs.GR], 2015.

[26] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, „ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes," in *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[27] A. Chang, A. Dai, T. Funkhouser, M. Halber, M. NieÃŸner, M. Savva, S. Song, A. Zeng, and Y. Zhang, *Matterport3D: Learning from RGB-D Data in Indoor Environments*, 2017. [Online]. Available: https://arxiv.org/abs/1709.06158.

[28] I. Armeni, A. Sax, A. R. Zamir, and S. Savarese, „Joint 2D-3D-Semantic Data for Indoor Scene Understanding," *ArXiv e-prints*, Feb. 2017. arXiv: 1702.01105 [cs.CV].

[29] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, and C. Theobalt, *BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration*, 2016. [Online]. Available: https://arxiv.org/abs/1604.01093.

[30] *Machine Learning - Training, Validation Test Data Set*, https://vitalflux.com/machine-learning-training-validation-test-data-set, Accessed: 2022-06-06.

[31] *Splitting a Dataset into Train and Test Sets*, https://www.baeldung.com/cs/train-test-datasets-ratio, Accessed: 2022-06-06.

[32] J. Miller, *Statistics for Data Science: Leverage the power of statistics for Data Analysis, Classification, Regression, Machine Learning, and Neural Networks*. Packt Publishing, 2017, ISBN: 9781788295345. [Online]. Available: https://books.google.at/books?id=gkFPDwAAQBAJ.

[33] V. Nair and G. Hinton, „Rectified Linear Units Improve Restricted Boltzmann Machines Vinod Nair," vol. 27, Jun. 2010, pp. 807–814.

[34] A. Krizhevsky, I. Sutskever, and G. E. Hinton, „Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[35] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, „Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[36] X. Han, Y. Zhong, C. Liqin, and L. Zhang, „Pre-Trained AlexNet Architecture with Pyramid Pooling and Supervision for High Spatial Resolution Remote Sensing Image Scene Classification," *Remote Sensing*, vol. 9, p. 848, Aug. 2017.

[37] D. P. Kingma and J. Ba, *Adam: A Method for Stochastic Optimization*, 2014. [Online]. Available: https://arxiv.org/abs/1412.6980.

[38] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, „Return of the Devil in the Details: Delving Deep into Convolutional Nets," *CoRR*, vol. abs/1405.3531, 2014. arXiv: 1405.3531. [Online]. Available: http://arxiv.org/abs/1405.3531.

[39] K. Simonyan and A. Zisserman, „Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv 1409.1556*, Sep. 2014.

[40] Y. Feng, Z. Zhang, X. Zhao, R. Ji, and Y. Gao, „Gvcnn: Group-view convolutional neural networks for 3d shape recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 264–272.

[41] Z. Han, H. Lu, Z. Liu, C.-M. Vong, Y.-S. Liu, M. Zwicker, J. Han, and C. P. Chen, „3D2SeqViews: Aggregating sequential views for 3D global feature learning by CNN with hierarchical attention aggregation," *IEEE Transactions on Image Processing*, vol. 28, no. 8, pp. 3986–3999, 2019.

[42] K. Sfikas, I. Pratikakis, and T. Theoharis, „Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval," *Computers & Graphics*, vol. 71, pp. 208–218, 2018.

[43] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, „3d shapenets: A deep representation for volumetric shapes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1912–1920.

[44] G. Riegler, A. Osman Ulusoy, and A. Geiger, „Octnet: Learning deep 3d representations at high resolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3577–3586.

[45] C. Ma, W. An, Y. Lei, and Y. Guo, „BV-CNNs: Binary Volumetric Convolutional Networks for 3D Object Recognition.," in *BMVC*, vol. 1, 2017, p. 4.

[46] C. Choy, J. Gwak, and S. Savarese, „4d spatio-temporal convnets: Minkowski convolutional neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3075–3084.

[47] W. Song, L. Zhang, T. Yifei, S. Fong, J. Liu, and A. Gozho, „CNN-based 3D object classification using Hough space of LiDAR point clouds," *Human-centric Computing and Information Sciences*, vol. 10, Dec. 2020.

[48] P. K. Nathan Silberman Derek Hoiem and R. Fergus, „Indoor Segmentation and Support Inference from RGBD Images," in *ECCV*, 2012.

[49] Y. Xu, T. Fan, M. Xu, L. Zeng, and Y. Qiao, „Spidercnn: Deep learning on point sets with parameterized convolutional filters," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 87–102.

[50] A. Bewley, J. Rigley, Y. Liu, J. Hawke, R. Shen, V.-D. Lam, and A. Kendall, *Learning to Drive from Simulation without Real World Labels*, 2018. [Online]. Available: https://arxiv.org/abs/1812.03823.

[51] T. Jaunet, G. Bono, R. Vuillemot, and C. Wolf, *SIM2REALVIZ: Visualizing the Sim2Real Gap in Robot Ego-Pose Estimation*, 2021. [Online]. Available: https://arxiv.org/abs/2109.11801.

[52] J.-B. Weibel, T. Patten, and M. Vincze, *Sim2Real 3D Object Classification using Spherical Kernel Point Convolution and a Deep Center Voting Scheme*, 2021. [Online]. Available: https://arxiv.org/abs/2103.06134.

[53] B. Curless and M. Levoy, „A Volumetric Method for Building Complex Models from Range Images," in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '96, New York, NY, USA: Association for Computing Machinery, 1996, pp. 303–312, ISBN: 0897917464. [Online]. Available: https://doi.org/10.1145/237170.237269.

[54] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, „KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, 2011, pp. 127–136.

[55] Q.-Y. Zhou, J. Park, and V. Koltun, „Open3D: A Modern Library for 3D Data Processing," *arXiv:1801.09847*, 2018.

[56] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, „Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions," Jun. 2013.

[57] W. E. Lorensen and H. E. Cline, „Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, no. 4, pp. 163–169, Aug. 1987, ISSN: 0097-8930. [Online]. Available: https://doi.org/10.1145/37402.37422.

[58] J.-M. Favreau, *Marching Cubes*, https://commons.wikimedia.org/wiki/File:MarchingCubes.svg, Accessed: 2022-09-01.

[59] C. V. Nguyen, S. Izadi, and D. Lovell, „Modeling kinect sensor noise for improved 3d reconstruction and tracking," in *2012 second international conference on 3D imaging, modeling, processing, visualization & transmission*, IEEE, 2012, pp. 524–530.

[60] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, „PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[61] J.-C. Su, *PyTorch code for MVCNN*, https://github.com/jongchyisu/mvcnn_pytorch, 2018.

[62] P. Min, *binvox*, http://www.patrickmin.com/binvox.

[63] B.-S. Hua, Q.-H. Pham, D. T. Nguyen, M.-K. Tran, L.-F. Yu, and S.-K. Yeung, „Scenenn: A scene meshes dataset with annotations," in *3D Vision (3DV), 2016 Fourth International Conference on*, IEEE, 2016, pp. 92–101.

[64] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, „PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[65] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, R. Jozefowicz, Y. Jia, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, M. Schuster, R. Monga, S. Moore, D. Murray, C. Olah, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow, Large-scale machine learning on heterogeneous systems*, Nov. 2015.