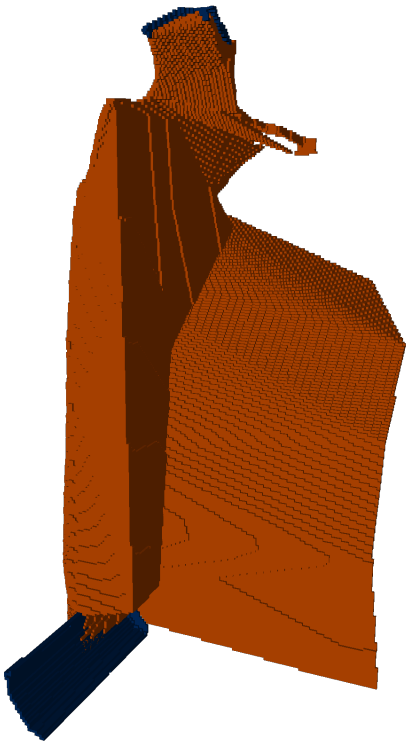


Learning Geometry Compatibility with 3D Convolutional Neural Networks

Master of Science Thesis in Computer Science and Engineering

Florian Stellbrink

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2019

Master's thesis 2019

Learning Geometry Compatibility with 3D Convolutional Neural Networks

Florian Stellbrink

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Learning Geometry Compatibility with Volumetric Convolutional Neural Networks
Florian Stellbrink

Supervisor: Erik Sintorn, Department of Computer Science and Engineering
Advisor: Johan Leijon, Ghost Games
Examiner: Ulf Assarsson, Department of Computer Science and Engineering

Master's Thesis 2019
Department of Computer Science and Engineering
Computer Graphics Research Group
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Voxelized representation of a geometry pair

Typeset in LaTeX
Gothenburg, Sweden 2019

Learning Geometry Compatibility with Volumetric Convolutional Neural Networks
Florian Stellbrink
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Modern video games offer substantial amounts of customization options. Manually testing the visual compatibility of all options is time-consuming and error-prone. Together with Ghost Games, we present a method of learning the visual compatibility between pairs of geometries. We introduce a transformation pipeline and model architecture, which we train on hand-labeled data. Furthermore, we explore a part of the hyperparameter space of our proposed architecture and extend it to accommodate confidence predictions. Finally, we run a quantitative study on the trained model and suggest improvements and extensions for future work.

Keywords: computer science, computer graphics, machine learning, geometry, voxel, thesis

Acknowledgements

I thank everyone at Ghost Games, for introducing me to their world, lending helping hands whenever I needed them, and always making me feel welcome.
I would especially like to thank my advisor Johan Leijon, for always being engaged and helpful and consistently caring. I appreciate the sincerity with which he considered my ambitions. His guidance made this thesis what it is.
I thank Nicolas Mercier for taking a personal interest in the project and helping out whenever he was needed.
I thank my supervisor Erik Sintorn, for supporting me through all twists and turns. I appreciate our conversations that consistently opened my mind to new possibilities, and made me see things from different perspectives.
I would also like to thank my examiner Ulf Assarsson, for always being responsive and helpful.
Finally, I must express my very profound gratitude to my parents and to my brother for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.
Thank you.

Florian Stellbrink, Gothenburg, June 2019

Contents

List of Figures

List of Tables

1

Introduction

In some video games, players get to customize various parts of the game, e.g., their own character, vehicles, and sometimes the game world itself. Such options can increase enjoyment by making players attached to their own creations [9].

In this thesis, we are working with Ghost Games, who are making the racing game series Need for Speed. Their games allow players to customize their cars. Most parts of a vehicle's body, such as hoods, fenders, and spoilers, can be replaced with other pieces. Their shape is mostly unrestricted, to maximize artistic freedom in creating these models. At the same time, it is possible to choose freely from all possible combinations of customizations. Having many different parts and the ability to combine them freely leads to an enormous amount of possible combinations and makes manual tasks labor intensive.

In this thesis, we try to determine which combinations of car parts visually fit together automatically.

1.1 Motivation

The process of creating new parts for cars is iterative. It usually starts with artists designing a new set of parts for the entire vehicle. This is called a *body kit* and begins with all contained parts fitting every neighboring part.

Now, since the player should be able to pick and choose parts across all body kits, the team needs to go over all parts and make sure that they visually fit together. This means testing all pairs for visible intersections, significant gaps, and other visual inconsistencies. During that review, a list of incompatible parts is made. Based on that list, artists create variations that overcome the visual problems. Once all problems were resolved, or new parts are added, this process starts over.

As more customization options are added, checking for inconsistencies gets more complicated, because more parts need to be considered. We also need to provide an ever-increasing amount of variations due to the exponentially growing number of part combinations. At some point, the effort of creating variations for new parts becomes too large, and we cannot add more customizations.

Automating this process could replace some of the manual quality assurance, remove human error from the process, and allow a more substantial amount of customization options. It could be integrated into the developing pipeline to give continuous information about variations that still need to be created to resolve conflicts.

1.2 Problem Statement

When we are talking about car parts, we mean one or more polygon meshes. A variation of such a part is another list of meshes that were created to overcome a restriction with another car part. We call visually incompatible variations restricted and store a list of restricted parts in each variation.
The purpose of this thesis is to decide whether two variations fit together. Using that information and existing algorithms, we can decide if a car can be used to resolve any player choice of parts. Operations on entire vehicles are a little more involved because restrictions form a graph that propagates the effects of earlier decisions. This is why we restrict our scope to considering only pairs of parts.
We are going to start with an existing set of parts, variations, and restrictions. These are handcrafted and form the basis of our dataset. To decide whether any pair of variations is restricted, we are going to use machine learning. We were able to choose that approach thanks to the existing dataset. We explain why we chose that approach later on.
We are going to develop a neural network architecture based on previous work. We discuss the architectural decisions and trade-offs. We are also going to create alternative architectures to test the effects of changing hyperparameters and adjust the network accordingly.
The ultimate goal is to have an algorithm that can imitate human decision making on geometric data with high accuracy. We can measure error precisely and optimize for correct predictions. The goal is to have a system that can make useful suggestions. To that effect, we are going to discuss ways in which inferences from our neural network can be integrated with existing workflows.

2
Previous Work

Guo et al. [7] applied deep convolutional neural networks to the problem of labeling meshes. This problem is somewhat related to the one we are facing, in that the input is a geometry represented as a mesh. What makes their problem different is that they do not compare different geometries, but instead classify each polygon within the mesh. To achieve that they extract geometric features, such as the curvature from faces.
Similarly, Engelcke et al. [4] applied a variation of that model structure called sparse convolutional neural networks to the task of object detection in point clouds. This circumvents the problem of traditional convolutional neural networks being represented as dense matrices, which would have been at odds with the sparse point cloud representation.
Huang et al. [8] solved the similar problem of labeling point cloud data by converting them into a voxel representation. They proceed to do their classification with dense convolutions on those voxels and translate the results back into point clouds.
We are going to follow a similar approach by converting our meshes into voxels. This is a popular pattern, as dense convolutional neural networks can be applied to them directly. This representation has been used, among others things, to classify objects [12], find generative representations of objects [15], and for medical volumetric image segmentation [13].
Research in convolutional networks has also been extended into non-image directions. Abdel et al. [1] have applied the similar concepts to speech recognition, while Gehring et al. [6] used it for machine translation. Both report promising results despite the unusual application.
Occasionally it is essential to measure a network's certainty of predictions. Krstajic et al. [10] introduce an additional category to binary classification, that handles uncertain cases. They manage to improve prediction accuracy by iteratively relabeling data as uncertain when it is close to the decision threshold.
Zhu et al. [17] and Gal et al. [5] separately explore the application of dropout at test time to determine uncertainty. They respectively consider a continuous mask and grid search approach and come to solutions that lessen the impact of expensive drop out testing.
Chopra et al. [3] introduce a method of learning similarity-metrics and present fundamental architecture choices. These are reiterated in the related survey about metric learning by Kulis et al. [11].
Finally, Zagoruyko et al. [16] present a method of learning distances between image patches. Their overall architecture decisions are very closely related to our problem, as they also deal with pairs of data on a regular grid.

3

Theory

3.1 Geometry Representation

Three-dimensional geometry representations are used in a variety of use cases, such as computer-aided design (CAD) for engineers, computer-generated imagery (CGI) for movies, and real-time rendering in video games. All of these applications have different requirements, and different geometry representations work best for each of them. This is somewhat similar to how there are a variety of representations of images for different purposes.

A designer might, for example, use vector graphics to define precise shapes and curves. Similar tools form the underlying representation of modern CAD software. Even though one is working in two dimensions and the other in three, they share the need for an easy to edit representation of complex shapes. It does not matter that these representations are somewhat expensive to compute, as long as they scale without losing quality and are expressive.

One step removed from these high-level representations we find polygon meshes, which are popular in CGI and real-time rendering. They consist of many vertices that are grouped into polygons. While this representation cannot express continuous curves, polygons are still infinitely scalable and preserve their look at any resolution. Computer graphics hardware is heavily optimized for this representation, allowing for high-performance real-time rendering.

3.1.1 Voxel Representation

Rasterized representations are popular for images and videos. Figure 3.1 shows how a triangle can be represented in a two-dimensional regular grid. We call elements in a picture grid pixels. This format is very popular because it maps well to the hardware used in digital cameras and computer screens. It is also easy to convert a vector representation of an image, like the triangle from our figure, into pixels through what we call rasterization.

Rasterization is the process of intersecting each pixel with shapes from an input and marking them accordingly. If for example, the color of a triangle was red, we could mark each intersected pixel with that color. In theory, grid representations can store any number of values in each cell; however, in practice, it usually boils down to three color channels and an opacity channel. It is important to note that rasterization is a lossy process. We lose information like surface normals and which object a particular pixel comes from unless we encode that information explicitly.

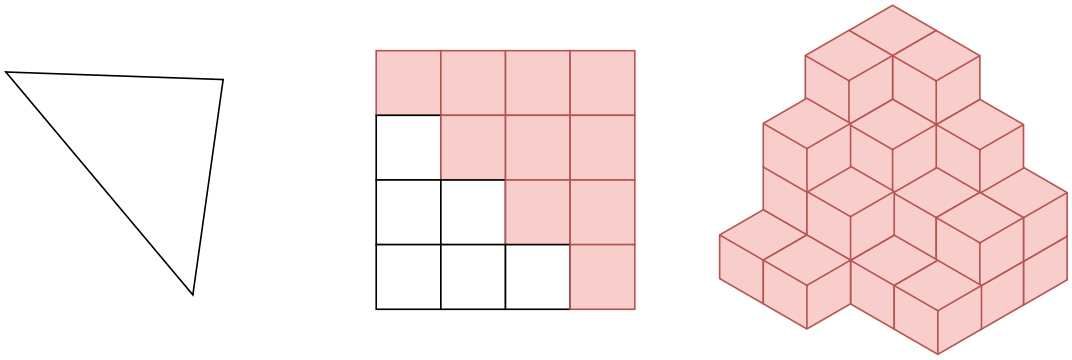
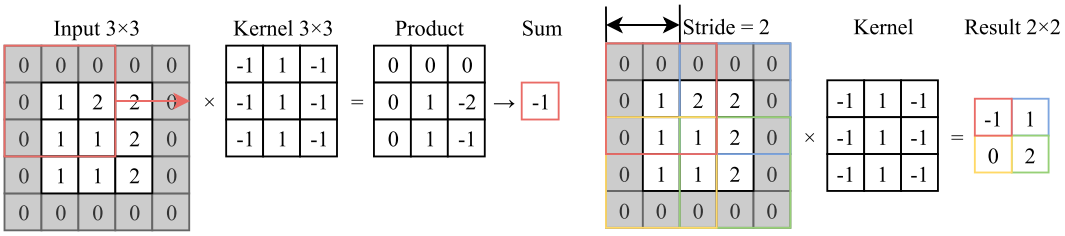


Figure 3.1: Illustration of a triangle being rasterized in a regular two-dimensional grid. Each cell of this picture grid is called a pixel. The same regular grid concept can be extended into three dimensions. This allows us to rasterize 3D geometry. We call each cell of the grid volume a voxel.

However, even then we cannot scale rasterized representations infinitely, as they have a limited resolution.
This same concept of having a regular grid of values can be extended into a third dimension. Instead of forming a rectangle, the grid is a cuboid, and we call cells within this volumetric grid voxels. We can still rasterize existing geometry into voxels, but it is also still a lossy process with the same tradeoffs described previously. Just as before we can store additional information in voxels. We could, for example, store a binary value that indicates whether a voxel is occupied, its color or material density [13].
The most significant advantage of rasterized representations in the context of this thesis is the implication for machine learning. Because all information is organized into a grid, we no longer have to encode positional information explicitly. Instead, all local information is encoded implicitly. This makes finding a cell's neighbours as easy as stepping along the grid. This property has been used extensively in convolutional neural networks, which we will look at in the next section. By using a voxel-based representation, we gain access to the same kind of tools for our geometry based problem.

3.2 Machine Learning

This thesis is partially set in the domain of analytical artificial intelligence. We want to imitate the human perception in the narrow problem of judging whether two pieces of geometry fit together.
In this thesis, we are working with data that has already been labeled. Using these data points as examples allows us to apply supervised machine learning. All methods in this field require inputs and outputs of the learning problem to be available. They can learn from the given examples and approximate a solution that ideally generalizes to unseen examples.
For supervised learning, we can choose from models such as support vector machines,



(a) Single Convolution **(b)** Convolution with Stride

Figure 3.2: 2D Convolution with a 3×3 kernel and stride of 2. The input in both figures is padded with zeros, which is indicated with a gray background.
Figure (a) shows how a 3×3 region of the input is multiplied element by element with the corresponding kernel entries. The resulting product is summed up to produce a single output value.
Figure (b) shows how stride is applied to skip over entries when moving the region along the input. Each of the highlighted regions produces a single value in the 2×2 output, by following the method indicated in figure (a).

Bayesian networks, and genetic algorithms. However, artificial neural networks have recently been the center of attention. They are especially popular when it comes to image and geometry processing [7, 4, 8].
We are assuming that the reader has some knowledge of neural networks. It is essential to understand the basic structure of layers, the function of losses, and the concept of training. We do, however, not assume more specialized knowledge about concepts such as convolutional layers. [2]

3.2.1 Convolutional Neural Networks

Convolutional neural networks originally became popular for image related tasks. They have recently been used for a wide variety of problems, such as volumetric classification [7, 4, 8], machine translation [6], and speech recognition [1]. Their design minimizes the memory footprint and allows for highly parallelizable computation, which makes them popular for many applications. Convolutional layers consist of kernels, which are arrays with the same number of dimensions as the input. However, each dimension usually has significantly fewer elements. Additionally, each kernel has the same number of channels as the input.
In Figure 3.2 (a), you can see a kernel being applied to a region of a larger input. All elements of the input are multiplied with the corresponding elements of the kernel. The elements of the resulting product are then summed up to create a single output value.
This process is repeated by applying the kernel to different regions of the input layer. In the example of Figure 3.2 (b), the input is two dimensional. The kernel is moved along both dimensions. While moving, the kernel's stride determines the step size. Using a larger stride means that the output will have fewer dimensions. By using a stride of two, the example scales the 3×3 input down to a 2×2 output.
The same concepts used in two-dimensional convolutions for images can be extended

into three-dimensional convolution for volumetric data. The kernels gain another dimension and are moved along all three dimensions. This allows us to exploit the same advantages convolutions bring to image processing.
The problem of deciding whether two geometries fit each other is mostly a local one. Kernels can determine intersections and gaps. Additionally, we consume much memory by extending into the third dimension. Using convolutional layers as the first steps of our model, allows us to reduce the size of the input quickly while using manageable amounts of memory.

3.2.2 Similarity Learning

Similarity learning is an area of machine learning that is about learning distances between inputs from examples [11]. In a supervised learning context, this is similar to a regression task. The two distinctions are: First that we are always dealing with pairs of input. And second, that the learned function follows distance metric axioms [14].

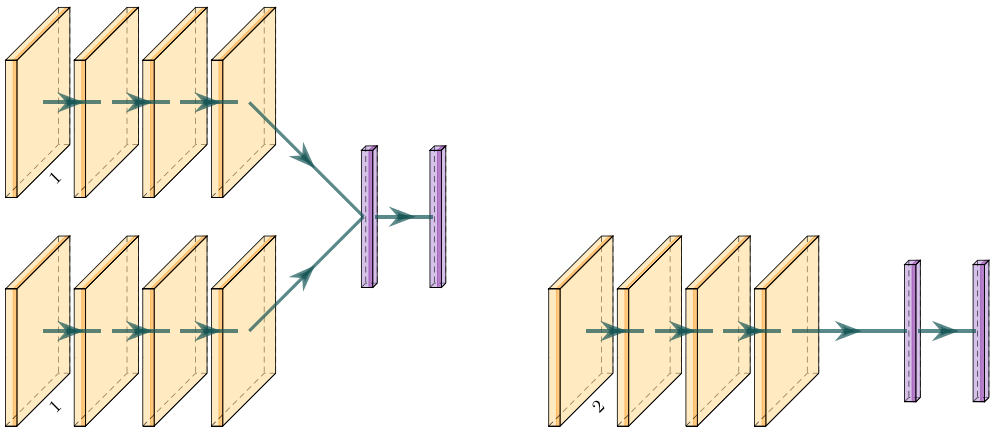
Metric

Metrics are functions that quantify the distance between a pair of elements. Examples for non-learned metrics include the Euclidean and Manhattan distance. We can use these distances to get a measure of similarity between items. For example, two points on a 2D plane can be said to be similar if their Euclidean distance is small. Similarly, a learned distance function can be used to estimate the similarity between any pair.
Distance functions are generally defined as: $d : X \times X \to [0, \infty)$. They need to satisfy the following axioms for $\forall x, y, z \in X$:

1. Non-negativity: $d(x, y) \geq 0$
2. Identity of indiscernibles: $d(x, y) = 0 \Leftrightarrow x = y$
3. Symmetry: $d(x, y) = d(y, x)$
4. Triangle inequality: $d(x, z) \leq d(x, y) + d(y, z)$

Architecture

On the practical side of things, dealing with input-pairs involves changes from common neural network architectures. This report focuses on supervised learning. However, similarity learning can also be done in an unsupervised fashion.
Any neural network has to fulfill two separate functions. The first is to find a suitable representation of the inputs and the second is to learn the metric from that representation. Neural network architectures are strongly influenced by how strictly these two functions are separated.
For example, latent space representations of inputs could be learned entirely separately from the metric. An autoencoder might be used to get to that representation. A second network could then use the latent space representation of input pairs to learn the metrics.
If we were to learn both a representation and the metric in the same network, we could set up two identical branches. Inputs would be fed into those branches, and



(a) Siamese network **(b)** Two channel network

Figure 3.3: Architectures for similarity learning.
The inputs can be embedded separately and then merged, as shown in figure (a). This is called Siamese or pseudo-Siamese depending on whether the weights are shared between the two branches.
Alternatively, the inputs can be merged at the onset, as depicted in figure (b). This requires that the network has a channel for each input.

they would, later on, be merged into a shared network to learn the metric. There is a variation to this approach called a Siamese network, where the two branches are identical copies of each other. This enables learning a generalized representation that is independent of the input order. If the two branches have the same structure but do not share the same weights, they are called pseudo-Siamese. This architecture is depicted in Figure 3.3 (a).
Finally, the inputs could be stacked on top of each other in the input layer. This enables the entire network to process both inputs simultaneously, which in turn leads to intermediate representations that can contain information from either input. At the same time, this approach loses the advantage of having a cleanly separated representation of either input. Figure 3.3 (b) visualizes this approach.
Both of these architectures are valid in their own regard. Their tradeoffs are complex and in practice are often chosen depending on what works for a particular problem. [11, 3]

3.2.3 Uncertainty

In the context of binary classification, it is sometimes essential to know whether a result is reliable. Let us consider a straightforward classifier that produces a single output in the range $[0, 1]$. A simple approach to converting this number into a prediction would be to define some threshold t and classify all outputs $\hat{y} \geq t$ as positive and all others as negative results.

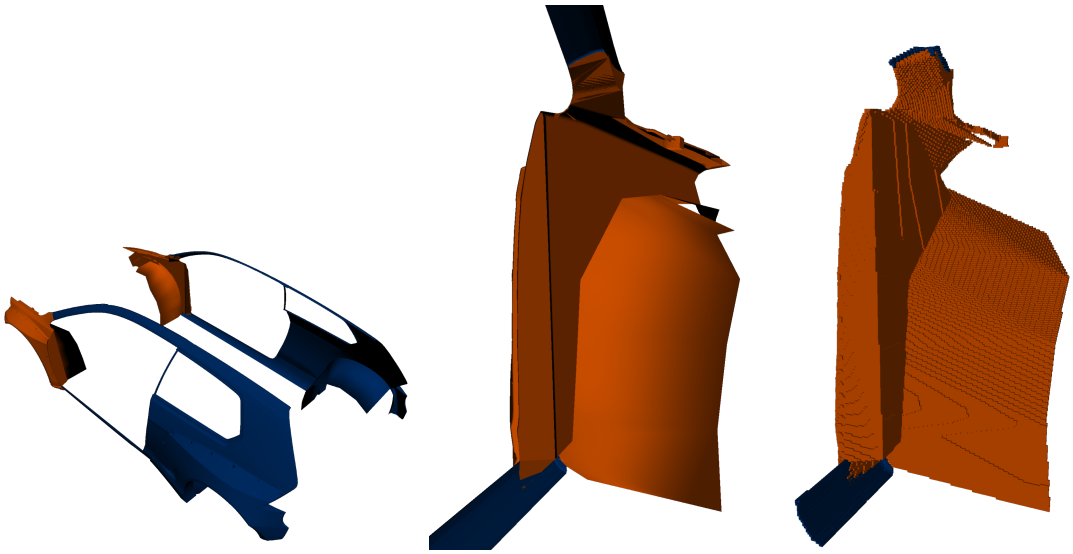
The problem we now run into is that the classifier is not perfect. It might be confronted with hard to decide or never before seen examples. We could define a range of values close to the t where we decide that the network is probably uncertain. However, in reality, the network has never been trained to handle these cases. Loss functions like binary cross entropy punish miss predictions. So a network can perform well, even when it occasionally makes polarized guesses for hard to decide cases.
Krstajic et al. [10] observed that some prediction values change depending on the random initialization of neural network weights. This means that even though networks were trained on the same data, and the labels do not change some predictions change randomly. They conclude that those predictions are unreliable and investigate the addition of an "Uncertain" category to binary classification. By iteratively placing unreliable predictions in this category, they can improve the precision of the binary classification. To make this work, they had to iteratively determine unreliable results and retrain the model with those new categories. Repeatedly training a model is not always feasible with long-running training tasks.
Others have made a similar observation of values changing depending on dropout [5, 17]. Dropout is usually applied during the training of neural networks. It randomly sets some values to 0, which reduces the network's reliance on those values and helps with overfitting. In this case, however, dropout is applied at test time to see how reliable the predictions are. Predictions are considered unreliable if they change when only a few inputs are missing. Despite offering elaborate tactics for choosing where to apply dropout, it still has to be applied iteratively to find weaknesses for every prediction. This slows inference down significantly.

4
Method

Our goal is to find a method that predicts the visual compatibility of vehicle parts. Depending on how we define visual compatibility, we can easily come up with algorithms to produce an answer. If the definition was, for example, two parts that do not intersect are compatible, the answer would be easy.
In reality, there is a particular intuitive understanding of what compatibility means. Humans looking at a car are able to pick up on visual discontinuities, significant intersections, and major gaps. They will, however not complain about overlaps hidden inside of parts or small, consistent gaps between them.
This intuitive definition is the one we want to follow. It has plenty of edge cases and details that humans can easily decide, but does not follow an obvious heuristic. Luckily we are in the fortunate position to have a large number of such decisions recorded and machine-readable. This allows us to apply machine learning methods to the problem and get close to human decision making.
In this chapter, we go through the steps of extracting data, transforming it into a suitable representation, and defining a model architecture to learn from the examples.

4.1 Geometry Representation

To achieve our goal of learning a function on geometry, we will need to feed geometry into a neural network. The commonly used mesh representation is well suited for real-time rendering but has some unfortunate properties as a neural network input. First of all, the number of vertices and polygons can vary between different models. This means that we would have to start padding the inputs to a fixed length or rely on model architectures such as recurrent neural networks (RNN) that can handle dynamically sized inputs.
The other unfortunate property of mesh representations is the impedance mismatch between an efficient surface representation and the learning task of deciding whether geometry fits together. One task our algorithm is going to have to work out is deciding whether geometry overlaps. That might be easy to decide after all triangles have been rendered, but if those polygons are fed into the network in an arbitrary order, finding two overlapping triangles will prove challenging. The network would have to come up with some internal representation that organizes data spatially.
We believe that this task represents a significant challenge for any learning algorithm, so we decided to use a different geometry representation as the learning task's input. Voxel-based representations organizes all information spatially, which suits or task



(a) Complete Parts (b) Intersection (c) Voxelized intersection

Figure 4.1: Example of two car parts before and after preprocessing. On the left, we see the original model files. The middle shows a cropped view of the shared regions of both models. Note that we exclude the mirrored half. The third picture shows a voxel representation of the models' shared bounding box.

well.
While using a voxel representation makes operations based on locality easier to process, it also comes with drawbacks. We lose some precision by rasterizing the geometry into a fixed size three-dimensional grid. This is a lot like rendering a vectorized image into a bitmap.
Another piece of information that is contained within the mesh are the surface and edge normals of polygons. These could have been used to judge how smoothly surfaces transition, which might be a deciding factor for telling whether two pieces of geometry fit together.
To compensate for the shortcomings, we can try to incorporate more data into our representation. One option is to increase the grid resolution, and the other is to add extra information such as the surface normals explicitly. Both of these options increase the size of our inputs, which uses some of the limited available memory. Because the other hyperparameter options also increased memory consumption, we decided not to explore this option. We always use the highest resolution that can fit in memory. This is arguably the most promising approach because the least amount of information is lost. Showing the effects of different voxelization resolutions remains future work.

4.2 Preprocessing

We start this project with many hand-labeled cars. Each of these cars contains a list of available parts. Additionally, each part has variations that were made to resolve

visual conflicts between parts. For each variation, we also get a list of restrictions. Every variation that is referenced in this list is incompatible with the other variation. For our learning tasks, we would like to break this down to the most straightforward possible representation. Specifically, we would like to have a list of voxel pairs for all pairs of car parts. Additionally, we need to know whether a given pair is restricted or free, i.e., whether it is visually compatible.
To get there, we start by iterating over all cars in the database. For each vehicle, we find all pairs of parts that have at least one restriction in any variation. The reason being that we are only interested in pairs that demonstrate the difference between being restricted and free. We argue that pairs that are never restricted have limited usefulness in providing that information. If all combinations of parts of a specific kind are compatible with all parts of another kind, the network could not learn what it means for them to be restricted. We will later test this assumption.
Having this list of what kinds of parts might be helpful, we continue to find all of their variations. After eliminating duplicate and symmetrically identical pairs, we have our fundamental dataset.
Next, we look at all the meshes that make up the parts referenced in our pair list. For each pair, we find the shared bounding box for both of its models. The reason is that we expect all interesting overlaps and interactions between parts to happen within their overlapping region.
We use this bounding box as the size for the voxelization region. We then proceed to voxelize both parts into that region with a fixed resolution. Meanwhile, we only consider one half of the model, because all cars in our database are almost symmetrical.
The reason behind finding the shared bounding box and eliminating the symmetric half is to maximize our usage of the fixed voxel resolution. If we were to voxelize parts of the model that are irrelevant for the decision making, they would reduce the resolution of the critical parts in our representation. Figure 4.1 visualizes the impact of this size reduction.
One side effect of maximizing the usage of our voxel representation is that we rescale the shared bounding box to occupy it completely. This means that we also rescale the geometry. Accordingly, parts with a large shared bounding box are scaled down, and parts with a small shared bounding box are scaled up.

4.3 Model Architecture

The fundamental goal of our architecture is similar to that of a similarity measure. We use two separate inputs with identical dimensions and output a single value. The difference is that our output value does not fulfill the requirements for a similarity measure, and is not going to be used as one. For example the identity of indiscernibles axiom says that $d(x, y) = 0 \Leftrightarrow x = y$. However, for our method $d(x, y) = 0$ simply means that x and y are not restricted, without any implications on their equality. This means that our function is not a metric, but we can still get some inspiration from the models used for similarity learning.
There are two general architecture approaches in similarity learning. They are depicted in Figure 3.3. One is the Siamese network approach where the two inputs

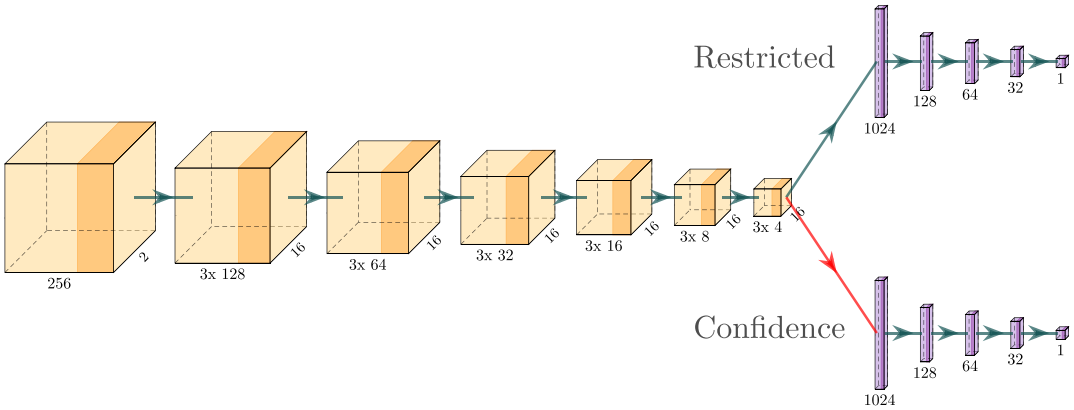


Figure 4.2: Our final model architecture starts with a series of convolutional layers. These reduce the dimensions of the input by using a stride of two. After those, we flatten the representation and feed the data through fully connected layers while reducing the vector size until we output a single value to predict restrictions.
The network branches into a separate path for the confidence prediction. Back-propagation for this branch is stopped from entering the convolutional layers, as indicated by the red arrow.

are fed into identical clones of the same network which are then merged later on to produce the similarity measure. The other approach is to stack both inputs on top of each other from the very beginning and only have one path through the network. While neither approach is generally better than the other, we argue that Siamese networks are less suitable for our problem. What we would end up doing is comparing embedded representations of both geometries. If however, we opted to stack the inputs on-top of each other from the beginning, the network could process inputs from both regions in the same layers, and even in the same kernels. We can see that this is indeed the case, by remembering that kernels cover all channels of a given input layer. If we arrange our inputs as additional channels, any kernel processes the same region from both items simultaneously. This makes it easier to detect overlap and gaps because each kernel has all the required information and processing capabilities individually. This allows us to skip intermediate representations for these elementary operations, and we should end up with a significant advantage.

4.3.1 Model Layers

Our model architecture is inspired by previously mentioned work on geometry [7, 8, 12, 13, 15], in that it mostly consists of convolutional layers. You can see a schematic depiction of the entire architecture in Figure 4.2. All layers except the output are activated with ReLU. The final output is activated using a sigmoid to normalize the prediction into the $[0, 1]$ range. As you can see, we decided to flatten out the last convolutional layer and usefully connected layers to produce the output. At this point in the model, the size of the layers becomes small enough to support fully connected layers. Their amount and size were subject to experimentation. We also consider the number of convolutional layers, as well as the number and size of kernels,

to be hyperparameters. We trained our model with different configurations starting from a baseline model with only eight kernels of size 3^3 and a single fully connected layer for the output. Then we created the following experiments to observe the effects of our parameters:

- Base line for comparison.
- Increase number of kernels to 16.
- Increase number of kernels by a factor of two after every convolution.
- Increase kernel size to 5^3.
- Add two additional fully connected layers.
- Add two additional convolutional layers for every convolutional layer in the default model.

In the following subsections, we are going to go through the reasons for each of these modifications and the effects we expect them to have.

4.3.1.1 More Channels

This modification increases the number of kernels in all convolutional layers from 8 to 16. Having more kernels means the number of channels for all convolutional outputs is increased to 16 as well. This means that we do not only gain more kernels but that each one of the kernels has to accommodate more channels in its input. We deal with an effective size increase of 4 for the memory consumption of all kernels.
We expect the increased number of kernels to be able to pick up on more patterns and to have an easier time encoding those in the output. This should make learning faster and enable a higher performance as more complexity can be learned. At the same time, we increase the potential for overfitting, which might affect the validation performance negatively.

4.3.1.2 Channel Doubling

By having a stride of 2 in all three dimensions of the convolutional layers, we rapidly decrease the layer size. But, by doubling the number of kernels between consecutive convolutional layers, we reduce the rate of shrinking. At the same time, we allow kernels in later convolutional layers to encode more features into more channels. The growing number of kernels increases the memory requirements significantly, but enables the extraction of more important features.
We expect the resulting model to fit the training data tightly and beat most other modifications. However, we also expect a high degree of overfitting.

4.3.1.3 Larger Kernels

Increasing the size of all kernels from 3^3 to 5^3 allows them to detect larger scale features. At the same time, they consume a larger amount of memory and are more expensive to evaluate.
This modification can only have a positive effect if there is information that can only be covered by a larger kernel. However, our architecture keeps compressing larger regions into smaller spaces. This means that eventually even a 3^3 kernel covers large

parts of the original input. Unless the larger kernels find new information in the first convolutional step, we do not expect them to increase performance significantly.

4.3.1.4 More Convolutions

By adding two additional convolutional layers for each existing convolutional layer, we enable more processing before scaling down. The problem we are trying to solve with this modification is that each convolutional layer compresses the output right away. Further steps could extract different features or modify the already extracted information in helpful ways. Adding extra convolutional layers should enable that kind of processing.
We expect these additional layers to improve the quality of the learned features [13]. Since we do not increase the size of the data representation at any point, there is little potential for overfitting. We can expect any improvement to translate well into the validation set.

4.3.1.5 Deeper classifier

The default network uses a single dense output layer after flattening the last convolutional layer. This means that the output is essentially a linear combination of features extracted up to that point. We might gain better behavior by gradually reducing the size of that flattened output over multiple fully connected layers. This has the potential to extract previously lost information.
We expect this to improve our accuracy slightly.

4.3.1.6 Final

After evaluating the performance of each modification in section 5.2, we came up with a combination of the above experiments. We carefully considered the memory impact and improvements to the learning process. Some combinations like extra convolutional layers combined with doubling kernel amounts were completely impractical due to memory limitations.
The final architecture is depicted in Figure 4.2. It adds two additional convolutional layers for every layer of the original architecture. Each convolutional layer has 16 kernels of size 5^3. After the last dense layer is flattened into 1024 dimensions, we add another three dense layers with sizes 128, 64, and 32, before feeding that into the output layer. All of the additional convolutional and dense layers are ReLu activated.

4.3.2 Uncertainty

When we talked about the final architecture in Figure 4.2 before, you may have noticed another branch for confidence. In this section, we discuss why some approaches to uncertainty prediction are impractical for our use case and what we do to predict confidence.
When we get a prediction from the network, it is difficult to tell whether the prediction is reliable or not. If we were to optimize for just false positives or negatives, we

could move our threshold accordingly. Unfortunately both our error cases should be avoided. False positives, i.e., predicting a restriction were both parts fit, are unnoticeable in the best case. This is only true if this pair is redundant and another pair can take its place. If it is not redundant, we might get too many restrictions that end up being unresolvable. This renders the entire car useless and would force us to find the culprit manually. False negatives, on the other hand, occur when we predict that a restricted pair is compatible. This means that when we later resolve a car, those parts might show up next to each other even though they are not visually compatible. This mistake would need to be noticed by a human, forcing us to double check the network's predictions.

Instead of moving the threshold, we would like to minimize the occurrences of either error. This is generally done by measuring uncertainty.

The first approach we introduced is applying dropout to the test results [5, 17]. Predictions that are close to the classification threshold or unstable for other reasons may switch classes when dropout is applied. This can be used to systematically search the space of inputs for synopses to set to zero. Unfortunately, this requires many iterations of inference, and our inference process is already taking too long.

Another alternative mentioned in the theory section is to bake uncertainty as an additional class into training [10]. While this also promises better results, we cannot afford multiple training passes at the moment.

Instead, we are adding a simple error estimation. For each processed item we determine the prediction correctness as $c = 1 - |(Y - \hat{Y})|$, where Y indicates whether the input was restricted, and $\hat{Y}$ what the network predicted. Now we add a new confidence output and use binary cross entropy between that output and c as the loss function. The idea is that a high output on this confidence corresponds to a correct prediction, and a low output indicates low confidence.

If, for example, an input was restricted, and we correctly predicted that it is restricted, we would reward a high confidence output. If we made an incorrect prediction instead, this loss function would prefer a low confidence output.

Having this additional value means we can inspect the predictions that were made by our model and have those with low confidence reviewed by humans.

We want to avoid having another training pass after the primary one, so we are going to add the error estimation as a new output to the existing network. Ideally, we would have a completely separate branch to learn this function. However, to save memory, we are going to reuse the convolutional part as indicated in Figure 4.2. However, we do introduce a gradient stop, to prevent backpropagation from the confidence output into the convolutional layers. This is intended to prevent the new loss function from negatively affecting the performance of the restriction prediction.

5

Results

The following chapter will go through the details of our data set, the setup of training and testing, and quantify the results. We will also go through the impact of tuning the hyperparameters that were laid out in subsection subsection 4.3.1.

5.1 Data Set

After extracting data pairs, as described in section 4.2, we obtained a total of $122,584$ from 84 vehicles. We based the extraction process on the assumption that car parts that never conflicted would not contribute significantly to the training process. The theoretical number of combinations is in the millions, but heavily biased towards unrestricted pairs.

We filtered out $34,764$ pairs, whose bounding box did not intersect. The idea was that these pairs are likely unrelated and therefore not restricted. This assumption turned out to be mostly correct. Only 0.29% of these pairs were marked as restricted.

To get a reliable evaluation, we set aside all samples from 15 vehicles. We never used these 9521 pairs during training or validation. They were not used for hyperparameter tuning either. Doing so should minimize the bias in this evaluation. Furthermore, doing testing on never before seen vehicles forces us to test the generalization capabilities of the trained network. If we had considered unseen parts from viewed cars, we might have encountered similar elements. This way, we are guaranteed only to examine different looking parts.

To test the previous assumption, that never restricted pairs do not contribute to training, we have included those in a second test set. This set is based on the same 15 vehicles as the first test set. It does also not contain pairs with an empty shared bounding box. By dropping the first assumption, this set grows to $22,495$ pairs. If training does not generalize to this set, we can assume that the hypothesis did not hold.

During training, we encountered a significant bias. There were 1.86 times more restricted than unrestricted pairs. We decided not to oversample the restricted class, because that would have affected the training duration. Our training already took over ten days to converge fully. Instead, we introduced class weights to reduce the impact of restricted samples to 0.54.

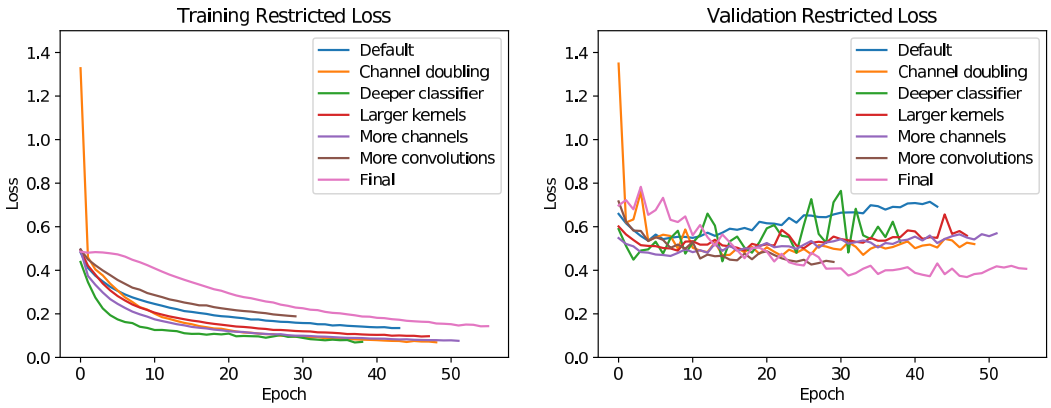


Figure 5.1: Hyperparameter tuning results for estimating restriction. The epoch counts the number of full iterations over the training set. The loss is binary cross-entropy.

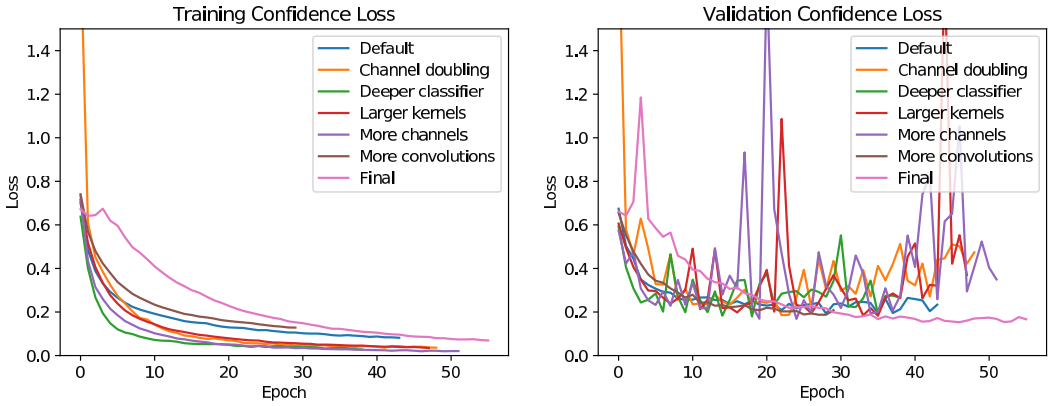


Figure 5.2: Hyperparameter tuning results for estimating confidence. The loss function is binary cross-entropy between the predicted confidence and the correctness of the restriction prediction.

5.2 Hyperparameter Tuning

We created neural networks for all modifications described in subsection 4.3.1. We trained each of them for a minimum of 30 epochs. Figure 5.1 depicts the training and validation losses for each modification as a function of epochs. During the same training runs, we also measured the loss functions for confidence predictions. These are shown in Figure 5.2. We are going to go over the effects each modification has in the following sections.

5.2.1 Default

For restriction loss, the default configuration converges relatively slowly and never fits the data as closely as most other configurations. In the validation set, the default configuration reaches its minimum before epoch ten and start to worsen afterward.

This indicates that what is learned in this configuration tends to be too specific to the training set and does not generalize well.
Surprisingly this changes in the confidence prediction. Even though its performance on the training set is very similar, it does not overfit as much, and it performs relatively well on the validation set.

5.2.2 More Channels

Adding more kernels to the convolutional layers increases the performance significantly. This configuration fits the training data quicker and tighter and generalizes better. It still tends to overfit slightly but to a much lesser degree, which can be seen when comparing its restriction validation loss to that of the default configuration.
In terms of confidence prediction, the new kernels seem to work wonders. They fit the data better than any other configuration. However, when it comes to the validation set, the behavior becomes completely erratic.

5.2.3 Channel Doubling

When doubling the number of channels after each convolutional layer, the results are remarkably similar to increasing them to a fixed number.
The only visible difference is the decreased amount of unpredictable behavior in the confidence validation loss.

5.2.4 Larger Kernels

Increasing the size of kernels also affects results similarly to changing their number. The performance is consistently better than the baseline, except for erratic behavior in the confidence validation loss.
This result suggests increasing both the size and number of kernels to combine their individually positive characteristics.

5.2.5 More Convolutions

Adding more convolutional layers that do not scale the data down seems to slow down training concerning both losses. It does, however, improve the characteristics in the restriction validation loss, indicating that the learned function generalizes better.

5.2.6 Deeper classifier

Adding additional fully connected layers to the classifier speeds up the learning concerning both losses. It also introduces some unpredictable behavior to both validation losses, while still slightly improving over the default configuration.

	Precision	Recall	F1-score	Support
Free	0.47	0.51	0.49	3431
Restricted	0.71	0.68	0.70	6089
Micro avg	0.62	0.62	0.62	9520
Macro avg	0.59	0.60	0.59	9520
Weighted avg	0.63	0.62	0.62	9520

Table 5.1: Restriction classification report on standard test set

5.2.7 Final

For the final model architecture, we attempted to combine the modifications with individually positive impacts on the learning behavior. Starting from the default model, we increase the number of kernels. This seemed to have a similar effect as doubling the channels after each convolutional layer while having a lower impact on the model's memory consumption. We further increase the kernel's sizes, add two new convolutional layers after each existing convolutional layer. Finally, additional dense layers are added before the outputs. To keep within memory limits, we reduce the size of the dense layers. All of these changes lead to the architecture we depicted earlier in Figure 4.2.
The combined model is larger than any other configuration. It is also the slowest in learning the training set. At the same time, its performance on the validation set is on par or better than all other architectures, while showing none of the unpredictable behavior others displayed.

5.3 Quantitative Study

Now that we have established the final model architecture, we are going to evaluate its performance. We created all evaluations in this section using both previously described test sets. Accordingly, the results can be used to assess whether our assumptions hold up.
Finally, we are going to do a similar evaluation of the confidence prediction. We are mainly interested in testing whether it helps identify wrong predictions.

5.3.1 Model Performance

In evaluating the model, we used it to infer predictions for both test sets. The first test set, depicted in Figure 5.3, filters out the same kind of pairs that were ignored in training, i.e., pairs of items that were never restricted. The curves in the figure are generated by moving the threshold between positive and negative classifications. As the limit gets closer to zero, we select more false positives, as it gets closer to one, we end up with more false negatives.
In a perfect world, the precision-recall (PR) curve would have a consistently high precision across all recall values. The precision value indicates the ratio of true positives to all selected items. The recall is the number of true positives among relevant items.

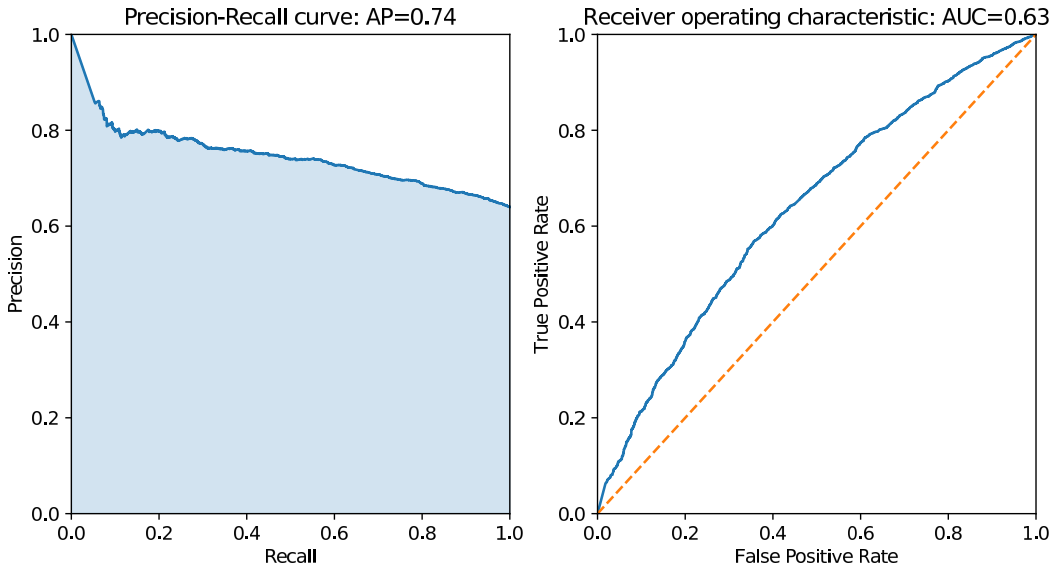


Figure 5.3: PR and ROC curves concerning restriction prediction on the standard test set. The first plot shows precision values for each recall, with the average precision being 0.74. The second plot shows false positive by true positive rates, where an ideal value would be in the top left corner. The area under the curve is indicated to be 0.63.

	Precision	Recall	F1-score	Support
Free	0.81	0.50	0.62	16404
Restricted	0.34	0.68	0.45	6088
Micro avg	0.55	0.55	0.55	22492
Macro avg	0.57	0.59	0.53	22492
Weighted avg	0.68	0.55	0.57	22492

Table 5.2: Restriction classification report on extended test set

What we see in the precision-recall curve is that our precision peaks at around 80% with an average precision of 74%. As we maximize the recall value by lowering the threshold, the precision drops close to 60%.

The receiver operating characteristic (ROC) in the same figure shows the ratio of true positives to false positives as the threshold is adjusted. Ideally, the curve would go through the very top left, which would indicate that there is a threshold producing both 100% true positives and absolutely no false negatives.

Our ROC curve does not seem to go anywhere near that top corner. We can explain that observation by looking at Table 5.1. The precision value for unrestricted pairs is listed as 0.81, whereas the precision of restricted pairs is 0.34. This difference seems to indicate a bias towards the more supported class. ROC curves are known not to reflect performance on unbalanced classes well, and our case appears to highlight that.

Let us now consider the test results for the extended test set. As we can see in

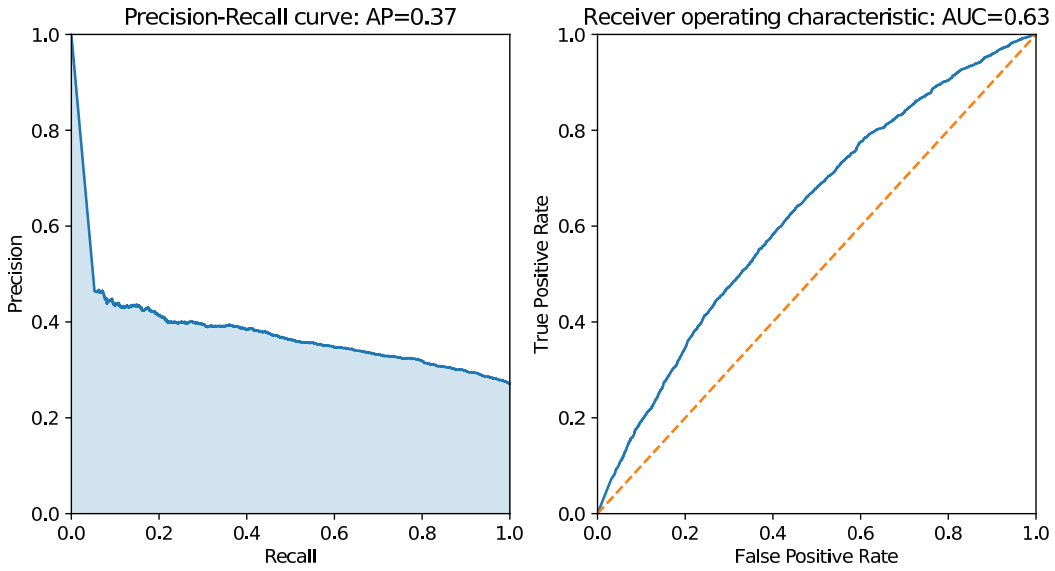


Figure 5.4: PR and ROC curves concerning restriction prediction on the extended test set. The first plot shows precision values for each recall, with the average precision being 0.37. The second plot shows false positive by true positive rates, where an ideal value would be in the top left corner. The area under the curve is indicated to be 0.63.

Figure 5.4, the precision takes a substantial hit. The average precision drops from 0.74 to 0.37. At the same time, the ROC curve does not reflect any significant changes, which again shows that it does not reflect the performance on imbalanced sets well. We are further going to consider the values reported for this test case in Table 5.2. In comparison to Table 5.1, the introduction of a large number of free samples has flipped the precision values for our classes. Due to the increased number of free pairs, their precision has naturally grown. However, the corresponding loss of precision in the restricted class seems to have lowered the overall model performance significantly.

5.3.2 Confidence

We decided earlier to measure model uncertainty by predicting its error. We can analyze the performance of that prediction, similar to the primary model evaluation. Figure 5.5 shows the performance of this prediction on our default test set. Both the shape of the curves and the critical values of average precision and area under the ROC curve match that of the original prediction in Figure 5.3. This means that the performance is generally very similar.
We originally intended this prediction to help us with identifying low confidence samples. To get a better idea of the relation between the confidence and restriction prediction, we have mapped all inferences from the test set into a histogram. This initial relation is depicted in the top left of Figure 5.6. The horizontal axis of this chart visualizes the prediction of whether a sample was considered restricted. Based

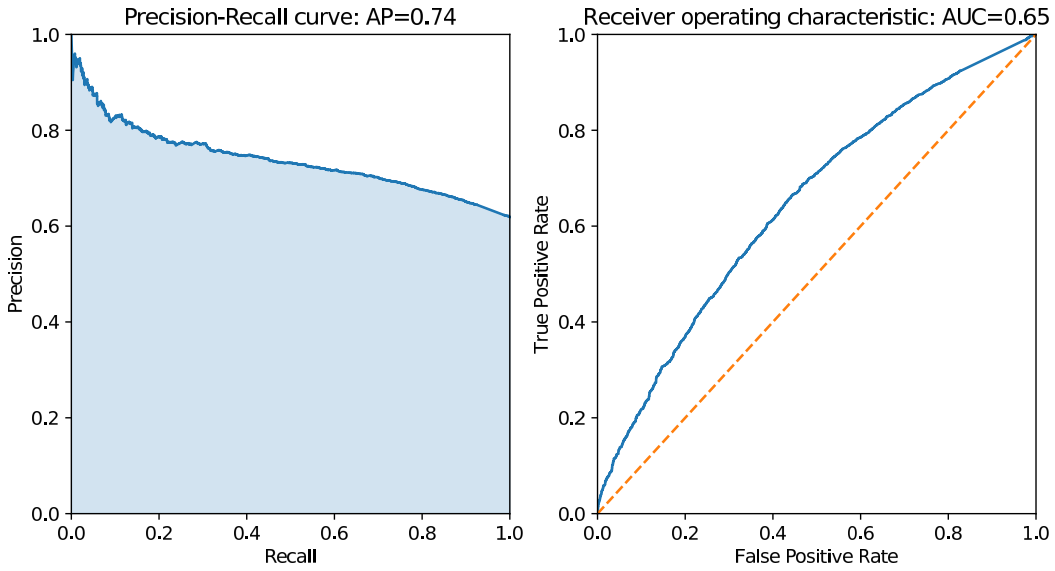


Figure 5.5: PR and ROC curves concerning confidence prediction on the standard test set. The first plot shows precision values for each recall, with the average precision being 0.74. The second plot shows false positive by true positive rates, where an ideal value would be in the top left corner. The area under the curve is indicated to be 0.65.

on the log scale color of cells, we can see that there are many values at the very extremes of this axis. The vertical axis indicates the confidence prediction for all values. When combined, the values arrange into a U shape. This general shape does not yet show whether confidence helps identify uncertain predictions.
To help visualize the impact of confidence prediction, we have separated the histogram into one for all correct predictions and one for all incorrect predictions. These histograms are depicted in the upper center and right of Figure 5.6. As you can see, both of them look similar to the initial distribution. This is because the overall distribution of samples dominates either distribution. To correct for that and get a better look at the difference between the distributions, we have added another row and normalized its values using the base distribution.
The lower left shows the base distribution normalized with itself, which leads to a constant value, as expected. The other two diagrams show that, relative to the base distribution, correct values tend to cluster in the top right, and incorrect values tend to be in the bottom left. Unfortunately, these tendencies are not nearly enough to cleanly separate correct from incorrect predictions. At the same time, we can see that the vertical axis does not seem to be much help with identifying errors either. This is backed by the remarkably similar performance characteristics in Figure 5.5 and Figure 5.3.

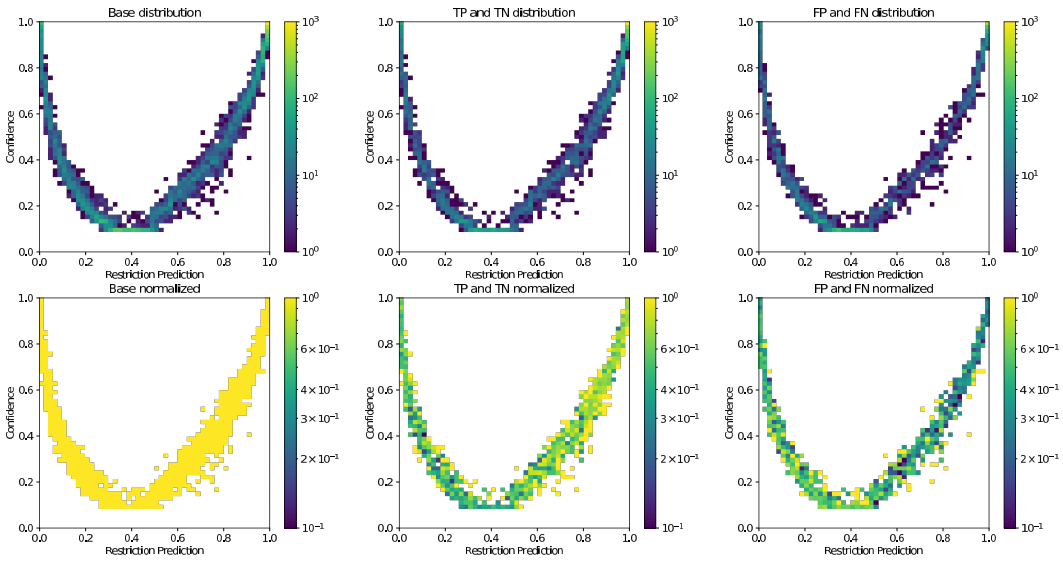


Figure 5.6: Histograms of predicted confidence by restriction distribution

6
Conclusion

We set out to automatically infer whether two pieces of geometry visually fit together. We do this to reduce the manual labor required to decide this for an exponentially growing number of combinations. Additionally, we look into ways of telling which predictions are reliable and which are not. This information can be used to limit further the number of pairs that need to be inspected by humans. If we would only predict compatibility without any information about the confidence of that information, we would have to manually double check every result to avoid mistakes.

We discuss a variety of geometry representations in the context of machine learning. We consider previous work to decide that the geometry in our problem can be represented in a regular volumetric grid. This representation further lends itself to processing with deep convolutional neural networks.

We learn about common model architectures in the related field of similarity learning. We choose to stack inputs at the very beginning of the model and decide against alternatives like siamese networks. We argue that this decision helps with key tasks such as detecting intersections and gaps in the geometry.

In predicting uncertainty, we discuss approaches ranging from test time dropout to additional categories and iterative reevaluation of labels. We conclude that most methods are infeasible with our setup. Instead, we settle on a simple error prediction extension to the existing network. Even though this is conceptually oversimplified, we intend to test its viability.

In the results, we test two assumptions concerning the data set. The first one being that pairs without shared bounding boxes are not restricted. The second assumption is that parts that are never restricted do not contribute to the training and can be correctly classified without including them in training.

The first assumption holds up, but the second one does not. Adding pairs that were never restricted to the dataset impacts its performance severely.

We proceed to test a variety of model variations, each of which was designed to highlight the effects of a different hyperparameter. We conclude that doubling the number of kernels is not significantly better than increasing them to a fixed amount. Adding more kernels and a deeper classifier both improves training performance. And finally, adding more convolutional layers slows training down, but makes it generalize better.

We construct our final model architecture by combining modifications with desirable performance characteristics.

In evaluating the model's performance, we notice the effects of class imbalance. We also find evidence contradicting the assumption that ignoring never restricted pairs

would have little impact.
We further inspect the performance of confidence prediction. In visualizing its relation with the first prediction, it becomes evident that it does not significantly help discriminate between reliable and unreliable inferences. Instead, the confidence output forms a parabolic shape when seen as a function of the restriction prediction.

6.1 Discussion

To meet our goal of making useful predictions for previously unseen pairs of vehicle parts, we have set up and trained a machine learning model. In this section, we are going to go through the major decisions we made along the way and discuss how they affected the results.
Initially, we meant to do a qualitative study. We intended to integrate predictions of our model into the game and get feedback from experts working on the same problem. Unfortunately, it turns out that some of our assumptions do not hold up, and the results on the full test set indicate insufficient performance. We are going to explain why we failed to produce good enough results and leave the qualitative evaluation to future work.

6.1.1 Results

The results we have on the limited test set look promising. The network has definitely picked up some skills in discriminating between restricted and free samples.
We also consider our hyperparameter tuning successful, even though not all consequences were predictable. For example, the extreme slow down in learning rate, could not be observed in any of the individual experiments, but only became apparent after combining them.
We cannot fully explain the erratic behavior of the learning rate in our confidence validation. We found that all experiments, which displayed this behavior, were trained using a relatively large batch size. However, not all runs with the larger batch size displayed this behavior. We have to assume that it may be a contributing factor. The architecture we used for the final training run has the largest memory footprint of any experiment. Accordingly, we had to lower its batch size to one. Even though this network incorporated changes from multiple variations with unpredictable behavior, it did not display any erratic behavior itself. This leads us to believe that a low enough batch size can compensate for the problem.
We might also want to consider the dynamic update of class weights in the confidence prediction. Since we did not set this up as a separate network, we had to rebalance the weights after every epoch. This may have disturbed the learning rates and lead to the observed effect. Additionally, the targets of confidence prediction were continually moving. A small change in the primary network could push some classifications over the threshold.
As the erratic behavior did not occur in our final training run we did not prioritize fixing it.

6.1.2 Assumption

We assumed that all parts that are never restricted are easy to learn and classify. Accordingly, we excluded them from training and focused on the more promising pairs that had restrictions. Our results clearly show that this assumption does not hold up. A model trained with this limited set has trouble performing on a full set. There clearly is information in these pairs, and they should have been included during training.

6.1.3 Accuracy

In our limited test set, that did not suffer from wrong assumptions, we got an average precision of 0.74. We believe that values in that range are a great starting point to automate parts of the process. Naturally, the human labeled data set is not perfect. Some cases are hard to decide even for experts but have to be classified as restricted or free. Accordingly, there are always going to be some inconsistencies in the data set and predictions will never be 100% accurate. It is more important that the decisions made by our network are consistent and explainable. Unfortunately, we can not evaluate this property at the time of writing this, as the performance on the complete data set is lacking. We would produce too many errors to gain any valuable insights.
Still, we expect that expanding the training set will improve our overall performance, overcome the initial wrong assumptions, and lead to beneficial results.

6.1.4 Confidence

Our data set does not contain any indication of uncertainty. However, there exist cases that are hard to decide. This leads to inconsistencies which we try to detect. Further, having an estimate of confidence in predictions would help in deciding whether results can be relied upon or need to be reviewed by experts.
We explored a variety of uncertainty estimates but deemed most of them impractical due to the long training duration and a large number of inferences. Instead, we settled on a simple error prediction, which we added on to the primary prediction model.
The results show that this prediction does not help in discriminating between correct and wrong conclusions. The confidence prediction performs very similarly to the restriction prediction and does not seem to offer any advantages over it.
We believe that this is due to several shortcomings of our method. First, we never explicitly encode uncertain predictions. This means that we only try to learn the current mispredictions, even when many more predictions are uncertain and flip flop around the threshold. Secondly, we keep moving the goal post for confidence predictions. By merging both the primary classification and the confidence prediction into the same model, we force it to react to sudden changes. Samples that were classified as restricted in one epoch might be free in the next, and the confidence prediction gets mixed messages as to what it is to learn.
We believe that we can overcome these shortcomings by putting humans in the loop. We describe this approach in section 6.3.

6.2 Summary

Our goal is to replicate human decision making in the narrow scope of deciding whether two pieces of geometry visually fit together.

We develop a method for representing geometry in a suitable format for machine learning and set up a data extraction pipeline.

We iterate over multiple architecture variations and combine them into our final model. This model is then trained with the previously extracted data. Testing the trained model with an independent test set shows that the learned knowledge largely generalizes to previously unseen vehicles.

We additionally try to reduce the training data size. The corresponding test results reveal that all data is necessary for generalization capabilities.

Finally, we introduce an error estimator that we intend to use for uncertainty measurements. We design this estimator to avoid the performance penalty associated with established uncertainty estimates. Testing tells us that the estimator learns a skill similar to the initial prediction, but does not significantly help with identifying uncertain cases. We conclude that this might be due to its overly simplistic design, and the decision to merge it with the original architecture.

Even though not all parts of the system work perfectly yet, we produce promising results and believe that future work can overcome the encountered problems.

6.3 Future Work

There are a number of shortcomings in the method we presented. The following sections will briefly present approaches to solving those. Further sections will introduce interesting extensions that can be based on this thesis.

6.3.1 Interactive loop

There are some cases in our dataset which are hard to decide. If we present the possibly inconsistent decision a human as made as a fact, the network will have a hard time generalizing and learning all intricacies.

Instead, one could explore an iterative approach that has the network train on the same data to start with but then adds a human in the training loop. Once every couple of epochs, we would find misclassified samples and show them to an expert. Now, the human gets to decide whether the original classification was correct or not. Additionally, they get to label samples as uncertain if they could be classified either way. Results that were confirmed this way are stored in a database and are used with an increased weight during succeeding training.

This approach is similar to existing semi-supervised methods. It should be useful in getting a high accuracy, but it takes more time and importantly, human labor to train.

6.3.2 Iteration Speed

In the process of improving our method, we had to iterate over many modifications. In each iteration, we had to train a new model from scratch. This took up to 10 days for each iteration. Additionally, generating the data set via voxelization took hours depending on the used resolution. This is a task that would even need to be done at inference time. Finally, inference itself also took many hours for a large test set.
In a production setting, we would have to generate voxel intersections quickly and run inference for all of them. Fortunately, this is entirely independent for all pairs and can, therefore, be massively parallelized.
To speed up training, we should consider the following optimizations:

1. Implement voxelization on GPU
2. Speed up data loading
3. Parallelize training

Voxelization is a process that maps well to GPU paradigms. Using the available computation power should speed the process up significantly.
Data loading is currently a limiting factor in GPU utilization during training. Even though our pipeline already prepares data ahead of time, each batch still needs to be transferred to the GPU before any processing can happen. Right now, we move the unpacked voxel representation onto the GPU. Instead, we could pack eight booleans into single bytes, and unpack them on the GPU cutting bandwidth requirements by a factor of eight. If we wanted to take this one step further, we could apply a simple compression scheme to the data and only uncompress it once data has reached the GPU.

6.3.3 Comlexity Estimation

One application of our method is to judge how many vehicle parts do not yet have compatible variations for a specific neighbor. However, we do not yet know how much work will be involved in creating those variations. If we could predict both the amount of work required and which pieces need to be created, we could come up with an estimate of the total amount of work.
One problem with such an estimation is that we can not rely on any existing data. Instead, we would have to introduce heuristics that estimate the amount of work. These could, for example, be based on the difference between existing models. An objective evaluation of such heuristics would still be difficult.

6.3.4 Geometry Adjustment

With our current method, we can find incompatible vehicle parts. A useful extension would be to predict what changes are necessary to make an existing part fit. To that end, we could start with one of the incompatible variations and introduce changes that aim to satisfy the classifier introduced in this thesis.
Assuring that geometry generated in such a way is of sufficient quality would likely involve some sort of policy network to judge the visual appearance. A generative adversarial network might be useful in this context.

6.4 Ethical Consideration

In this section, we go through the ethical implications of this work.
As with any data science study, we need to make sure that we handle personal information with care. Fortunately, our dataset exclusively consists of vehicle models, so we do not run the risk of exposing personal data.
The method presented here has the potential to automate two tasks that were previously performed by humans. One consists of artists finding incompatibilities during the creation of parts. People dealing with this task described it as a tedious part of their job. As previously mentioned in chapter 1, the number of customizations is partly limited by this task. Automating it would free them up to work on more creative pursuits.
There also is quality insurance, which makes sure that no problematic combinations make it into the final product. Based on our results, we believe that we cannot automate quality assurance with the required confidence. Even if we could, it would only affect a small portion of that job.

Bibliography

[1] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10):1533–1545, 2014.

[2] H. Abdi. A neural network primer. *Journal of Biological Systems*, 2(03):247–281, 1994.

[3] S. Chopra, R. Hadsell, Y. LeCun, et al. Learning a similarity metric discriminatively, with application to face verification. In *CVPR (1)*, pages 539–546, 2005.

[4] M. Engelcke, D. Rao, D. Z. Wang, C. H. Tong, and I. Posner. Vote3deep: Fast object detection in 3d point clouds using efficient convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1355–1361. IEEE, 2017.

[5] Y. Gal, J. Hron, and A. Kendall. Concrete dropout. In *Advances in Neural Information Processing Systems*, pages 3581–3590, 2017.

[6] J. Gehring, M. Auli, D. Grangier, and Y. N. Dauphin. A convolutional encoder model for neural machine translation. *arXiv preprint arXiv:1611.02344*, 2016.

[7] K. Guo, D. Zou, and X. Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 35(1):3, 2015.

[8] J. Huang and S. You. Point cloud labeling using 3d convolutional neural network. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 2670–2675. IEEE, 2016.

[9] K. Kim, M. G. Schmierbach, M.-Y. Chung, J. D. Fraustino, F. Dardis, L. Ahern, et al. Is it a sense of autonomy, control, or attachment? exploring the effects of in-game customization on game enjoyment. *Computers in Human Behavior*, 48:695–705, 2015.

[10] D. Krstajic, L. Buturovic, S. Thomas, and D. E. Leahy. Binary classification models with" uncertain" predictions. *arXiv preprint arXiv:1711.09677*, 2017.

[11] B. Kulis et al. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[12] 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)*, pages 922–928. IEEE, 2015.

[13] F. Milletari, N. Navab, and S.-A. Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 565–571. IEEE, 2016.

[14] P. M. Vitányi. Information distance in multiples. *IEEE Transactions on Information Theory*, 57(4):2451–2456, 2011.

[15] 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*, pages 1912–1920, 2015.

[16] S. Zagoruyko and N. Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2015.

[17] L. Zhu and N. Laptev. Deep and confident prediction for time series at uber. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 103–110. IEEE, 2017.