DEGREE PROJECT IN INFORMATION AND COMMUNICATION TECHNOLOGY,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2018

Generating 3D-objects using neural networks

PATRIK STIGEBORN

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

GENERATING 3D-OBJECTS USING NEURAL NETWORKS

By

Patrik Stigeborn

ROYAL INSTITUTE OF TECHNOLOGY

2018

Abstract

Enabling a 2D- to 3D-reconstruction is an interesting future service for Mutate AB, where this thesis is conducted. Convolutional neural networks (CNNs) is examined in different aspects, in order to give a realistic perception of what this technology is capable of. The task conducted, is the creation of a CNN that can be used to predict how an object from a 2D image would look in 3D. The main areas that this CNN is optimized for are Quality, Speed, and Simplicity. Where Quality is the output resolution of the 3D object, Speed is measured by the number of seconds it takes to complete a reconstruction, and Simplicity is achieved by using machine learning (ML). Enabling this could potentially ease the creation of 3D games and make the development faster. The chosen solution is to use two CNNs. The first CNN is using convolution to extract features from an input image. The second CNN is using transpose convolution to create a prediction of how the object would look in 3D, from the features extracted by the first neural network. This thesis is using an empirical development approach to reach an optimal solution for the CNN structure and its hyperparameters. The 3D-reconstruction is inspired by a sculpting process, meaning that the reconstruction starts with a low resolution and improves it iteratively. The result shows that the quality gained from each iteration grows exponentially whilst the increased time grows a lot less. Thereof, the conclusion is that the trade-off between speed and quality is in our favor. However, when looking at commercializing this technology or deploy it in a professional environment, it is still too slow to generate high resolution output. Also, in this case, the CNN is fragile when there are a lot of unrecognized shapes in the input image.

Acknowledgments

New technological advancements have always been a huge interest of mine. This is the main reason for why I chose this thesis. Therefore, I would like to give a special thanks to David Ekermann who took the time to discuss a variety of interesting projects and for becoming my supervisor at Mutate AB. I would also like to thank Daniel Råström at Mutate, for saving me a lot of time by creating a script in 3DsMax to generate images from 3D models (Used to create the second dataset). Furthermore, I would like to thank everyone Mutate for the energetic positive spirit each day and for treating me like one of their own, making this project an amazing experience.

I would also like to acknowledge my sincere appreciation towards my examiner, Magnus Boman, and supervisor, Henrik Boström, at KTH. For the fast responses and helpful suggestion towards improving the structure and readability of this thesis.

Contents

List of Abbreviations

AE	Auto Encoder
AI	Artificial Intelligence
CAE	Convolutional Autoencoder
CNN	Convolutional Neural Network
DL	Deep Learning
NN	Neural Network
FC	Fully Connected
GAN	Generative Adversarial Network
GB	GigaByte
GPU	Graphical Processor Unit
GT	Ground Truth
lReLU	Leaky Rectified Linear Unit
ML	Machine Learning
MSE	Mean Squared Error
NN	Neural Network
PoC	Proof of Concept
ReLU	Rectified Linear Unit
Voxel	Volumetric Pixel

Chapter 1

Introduction

This chapter aims toward giving an introduction to the problem area, the purpose of this thesis, and how the thesis is structured.

1.1 Background

A big difference between human beings and computers is that humans have developed some fundamental understanding of the world. Due to this understanding, it is easy to transfer a complex structure through a brief explanation. For example, telling a person to “think of a pink horse, with wings”, the person in question would have a pretty clear image of the creature you are talking about, without having seen it before. Generative artificial intelligence (AI) is a vastly growing area of research, and it is believed that Generative AI will enable computers to gain an understanding of the world [12]. This would enable computers to complete complex tasks, without someone controlling its every move. It would make artistic work much easier, a music producer could simply tell the computer to “increase the beat” or “add more drums”. A graphical designer could specify features (“cute dog”, “bigger head”, “more red color”) instead of spending a lot of man-hours on drawing. This could have a huge impact on the game creation industry. Machine learning (ML) is a subsection of artificial intelligence, it is used to enable systems to automatically learn from experience instead of being programmed exactly how to perform a task [8]. Meaning that it can be used to solve problems without the need to specify how to solve the problem, this is what makes ML incredibly powerful. This thesis will look into feature extraction from 2D images and reconstruction of 3D objects, using machine learning.

1.2 Problem

Creation of 3D objects can be a complex and time-consuming task for those who have little to none prior experience in 3D modeling. This can be a big problem for game engines since developers either have to learn 3D modeling or pay for models. If the models instead were automatically generated, the developing process would be easier and could result in more developers using the game engine.

To make the automatic 3D generation viable, three requirements are focused in order to consider the program *consumer friendly*, for this thesis:

1. Quality, output resolution.
2. Speed, generation time.
3. Simplicity, easy to use.

Both Wu and Häne's papers [23][10] shows that it is possible to generate a 3D object, using one input image and convolutional neural networks. This will be considered simple enough for the third *consumer friendly* requirement. What needs to be tested is the limitation of the output resolution and the generation time, which is not specified in any of the papers referenced above.

1.3 Purpose

The purpose of this thesis is to find out the limitations of using CNN to generate 3D voxelized objects, with regards to generation speed and output resolution. It will also focus on how big the trade-off is, between improving the resolution against the increase in time for generating an output. This is summarized in the research question below:

† How significant is the trade-off between generation speed and output resolution when generating 3D voxelized objects from a 2D RGB-image, using convolutional neural networks?

1.4 Objectives

The solution from this project focuses on three sub-areas (Quality, Speed, and Simplicity) to make the 3D generation program consumer friendly, as specified in Section 1.2. These will be explained more in detail in Section 3.1. In order to give an answer to the research question of this thesis, the following objectives will need to be fulfilled:

1. Decide upon how the machine learning model to extract features from a 2D image should be constructed.
2. Decide upon how the generative 3D machine learning model should be constructed.
3. Create and link the two machine learning models.
4. Tweak hyperparameters to give the best result.
5. Draw conclusion from the final model.

When this project is complete, the intention is to enable a 2D-to-3D reconstruction of an object, from a mobile camera photo to a 3D object. The object should be able to be imported into a game engine and it will be limited to only reconstruct chairs.

1.5 Methodology

The approach for this thesis project is to use an iterative empirical development method. Where a bigger problem, is split into smaller pieces using reductionism. As shown in figure 1.1, the generative model is first split into two different neural networks. One that is used to extract features from an input image, the other will use the output from the first NN to create a 3D object. These networks are trained and optimized individually using an exploratory, iterative, empirical approach. Meaning that small changes are made to hyper-parameters / dataset and the neural network (NN) is retrained, to see if the NN is reducing the mean squared error (MSE) further.

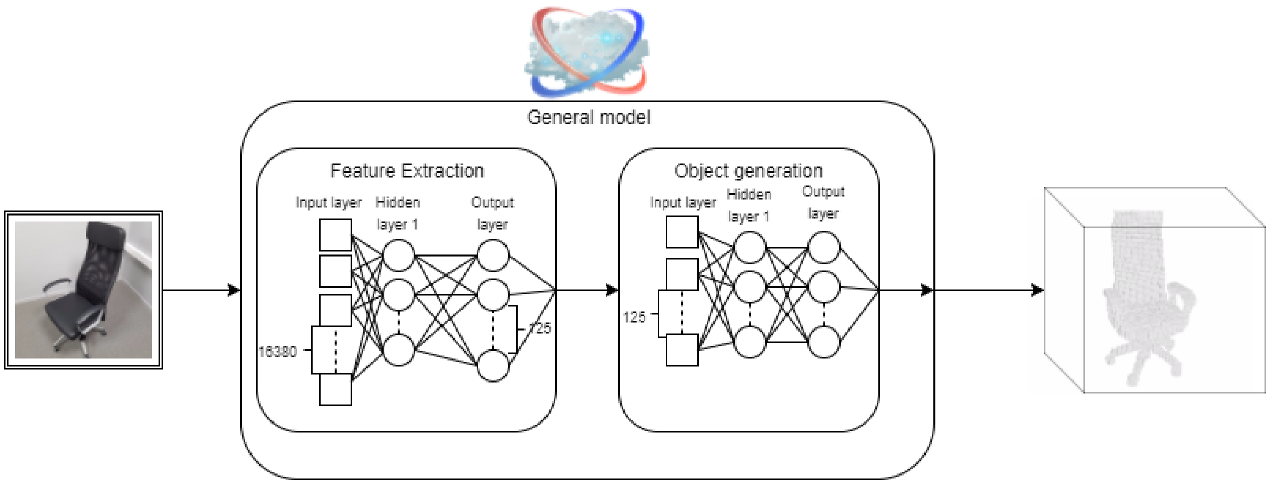


Figure 1.1: Generative architecture is split into two separate tasks, one for feature extraction and one for object generation.

This thesis is considering two, state of the art, methods for 3d object reconstruction. One Generative Adversarial Network (GAN) and one Autoencoder (AE) with an octree structured architecture.

The data used in this project originates from an IKEA dataset [14], it is analyzed manually and corrupt data is either removed or replaced.

1.6 Delimitations

The focus of this project is to look at one use case (reconstruction of chairs), thus a limitation is that this NN will not be generalized to reconstruct other objects than chairs.

1.7 Disposition

Chapter Two contain information gathered during the pre-study to give a good understanding of the foundation for this project. It will explain basics of neural network, give two different suggestion of generative models and inform on structural designs like an octree.

Chapter Three aims to give an understanding of the different iterations in this thesis. Method background that explains how this project is conducted. Result extraction that explains how different results are compared. Research strategy explains in-depth, how data was collected in different stages.

Chapter Four, focuses on recreate ability and architecture. It aims to give a detailed view of how each subpart (Feature extraction, object generation, interface, etc) is connected as a whole and which architectural structures are considered.

Chapter Five, focuses on validity. Results, both numerical and graphical to give a visual relation to what the numbers mean.

Chapter Six, discussion, and conclusion of how the projects went and the contribution that comes from the outcome of this thesis. This chapter is finishing up with suggestions for future work in three different areas.

Chapter 2

Theory

This chapter works as an informational foundation for this thesis. It explains data structures and some basics in machine learning. Two different suggestion of generative models are explained (Generative Adversarial Network & Autoencoder with an octree structure).

2.1 Structural foundation

This section explains structures that have been used to enable bigger resolution in the output.

2.1.1 Octree, Voxel, and Level of Detail

Octree is a tree structure where each node has exactly eight child nodes [7] (see middle section of Figure 2.1).

Voxel is an abbreviation for volume and element [16]. It has a cubic shape which makes it a good building block for representing a 3D object. To see a representation of a single voxel, see LoD 0 in Figure 2.1.

Level of Detail (LoD) originates from a rendering technique to save processing power when rendering landscapes in video games [21]. Objects that are far away would lose detail. In video games, the objects would be rendered with fewer polygons. In this thesis the LoD will be considered as the depth in the Octree, the further down in the Octree the better the quality will be (See Figure 2.1).

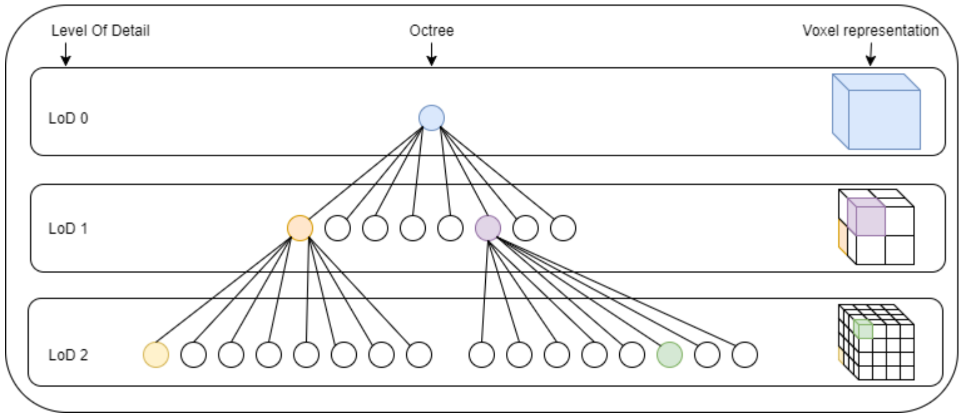


Figure 2.1: Visual description of the relation between Level of Detail, Octree, and Voxels. The Color coding is to show that each node corresponds to a specific voxel in the voxel space.

2.2 Machine Learning

Machine learning can use statistics and probability on a set of features to do classifications [18]. One way of comparing machine learning against traditional programming is that the traditional programming uses data together with a program to generate a specific output, see top image of Figure 2.2. However, by switching places of the output and program (see Figure 2.2) it would become a machine learning model, that produces a program instead of the same output as traditional programming. The input is 'data' and 'output' and it will create a program that can generate similar output from another set of data.

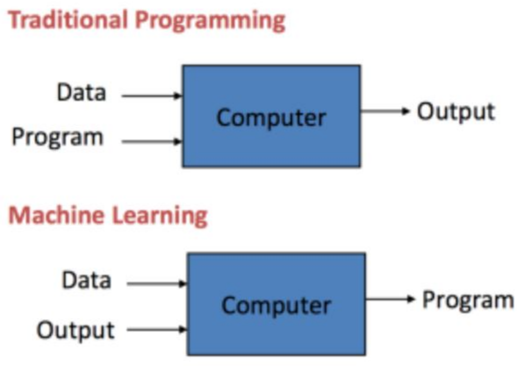


Figure 2.2: An example of how traditional programming and machine learning is constructed. The image is copied from Brownlee [2].

There are a few different choices of methods when working with machine learning: supervised, unsupervised, semi-supervised and reinforcement learning.

Supervised learning can be used when both input and output is accessible to the developer. It is used to find a mapping between the input and output, this mapping can then be used on unseen input, to map it against a similar output. An example is a classifier that can classify an e-mail as ‘spam’ or ‘not spam’. If the developer has a dataset, consisting of classified e-mails, the dataset can use the content of the e-mails as input and the classification as output. This could create a ‘program’ that can tell the difference between e-mails and spam.

Unsupervised learning can be used to cluster data. An example can be to find out how a language is constructed, with verbs/adjectives/nouns/etc. Unsupervised learning could be used by feeding a model a lot of texts, and specify that the model should group words that are used in a similar fashion.

When the input is specified and some output, but not all output. This is a bit more complex but an example is a Generative Adversarial Network (GAN) (see Subsection 2.3.2). To create a program that can create spam e-mail, two neural networks can be created to compete against each other. One tries to create spam mail and the other tries to tell if the mail is a spam generated from the other neural network, or if it’s a real e-mail from the correct dataset. While this would train, both network would improve, resulting in ‘better’ spam messages and a better spam detector. This would be a Semisupervised model since some data is existing from the start but new data is also created during the learning process.

When neither input nor output is given, but the available actions and a current state of an environment are available. In this case, reinforcement learning could start by randomly choosing different actions and by making assumptions on if it was a good action, depending on the environment changes for the better after the action was executed. An example could be to learn a helicopter to fly as high as possible, it would quickly learn that the rotor blade on the tail does not give as much height as if the big propeller blade on top of the helicopter is rotating.

A common way of utilizing machine learning is to use neural networks which are a biologically-inspired programming paradigm [15]. This technique gives computers the power to learn from observed data.

2.3 Convolutional Neural Network

Convolutional neural network (CNN) is a biologically-inspired network, inspiration from the visual cortex [4], adapted from the multilayer perceptrons (MLPs) [11]. CNN is a technique that can be used to represent data with fewer parameters, like classification. To explain the effectiveness of CNN, there is a competition called Large Scale Visual Recognition Challenge (LSVRC)[19]. Since mid-2010 there has been an annual competition in computer science. In this competition, people around the world try to make an artificial intelligence that could correctly classify the most images, in a dataset called ImageNet. In the year 2012, Alex Krizhevsky used a CNN to won the annual ImageNet competition and set a whole new standard for classifying images. They reached an error rate of 15.3%, comparing this against the second-best entry which reached an error rate of 26.2% [13] really shows its effectiveness. From this point, CNN became the new standard of image classification. Convolutional neural networks work by sliding a filter across the input data and performing a dot product calculation between the filter and the data, the result of this calculation will generate a new activation map. The activation map can either be processed to become an output or it can be part of the next hidden layer. A hidden layer would be processed in a similar way as the previous layer (see Figure 2.3).

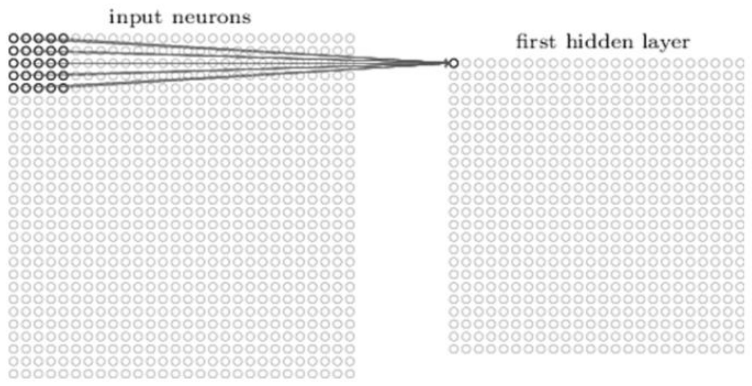


Figure 2.3: A visualization of 5 x 5 sized filter convolving around an input volume and producing an activation map, This image copied from the book Neural Networks and Deep Learning by Michael Nielsen [15].

2.3.1 Convolutional layer

A convolutional layer usually consists out of three parts: Convolution, activation function, and pooling. The Convolution is what happens in Figure 2.3. Dot product multiplication between filter/weights and input layer or hidden layer. Activation function will make the neural network nonlinear, to make it able to find more complex patterns, see Section 2.3.2. Pooling is used to reduce the data, hoping to remove unnecessary neurons and improving the performance, see Section 2.3.3.

2.3.2 Activation function

The purpose of an activation function is to make the neural network nonlinear. This will make it able to find more complex structure in data. The different activation functions that will be considered are Sigmoid, tanh, ReLU, leaky ReLu, (see Table 2.1).

Table 2.1
Activation Functions considered in this thesis.

Function 1 *Sigmoid*	**Function 2** *Softplus*	**Function 3** *Tanh*	**Function 4** *ReLU*	**Function 5** *lReLU*
$\frac{1}{e^-x}$	$log(e^x+1)$	$\frac{e^x-e^{-x}}{e^x+e^{-x}}$	$max(x,0)$	$x>0 \Rightarrow x$ $else \Rightarrow \alpha * x$

2.3.3 Pooling

Pooling is used to reduce the size of each activation map, to improve the speed of the neural network. A downside of pooling is that it will remove the ability of the neural networks to know the location of objects in an image since only the “best” value is saved. Two commonly used pooling functions are max-pooling and average-pooling. Normally a 2x2 sized window slides over the matrix of data and if max-pooling is used, only the highest value in the window is stored. Average pooling will store the average of all values in the window. By having a 2x2 sized window the size of the activation map can be reduced by 75%.

2.3.4 Loss function

The loss functions compare a ground truth (GT) answer against a generated output. It is what decides how well the model is performing, during training the loss function is minimized each iteration to improve the model. A few commonly used loss functions that will be considered are the Cross-entropy loss, Mean Squared Error (MSE), L1, and L2.

Cross-entropy is calculated by adding up the log value of the correct predictions times -1 to get a negative (see Table 2.2). ex. A model that should decide the color of a blanket, if the output is 70% chance that is green and 30% that it is red, now if the ground truth would be green the cross-entropy would be $-log(0.7) \approx 0.15$ which is a lesser value than if the blanket would have been red ($-log(0.3) \approx 0.52$).

L1 is calculated by adding up the absolute difference between output and GT (see Table 2.2). By using the same example as above, with a small difference that the output is how much red is in the blanket according to the 'R' in 'RGB' which has an interval of $0 - 255$. If the model would suggest 245 for the Red value and the ground truth is 250, this would give an error of $(245 - 250) = 5$.

L2 is calculated by adding up the squared difference between the output and GT (see Table 2.3). By using the same example as above in L1, the calculated error would be $(245 - 250)^2 = 25$. As shown, L2 will give a higher penalty for bigger error. Mean Squared Error (MSE) is calculated similarly, the difference being that it will divide the answer to get an average value, see Table 2.3.

L1 & L2 is similar to each other, the big difference between them is seen in Table 2.2 below.

Table 2.2
The difference between L1 & L2, copied from [3].

L2 loss function	**L1 loss function**
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions

Table 2.3
Loss functions, 'x' is the GT and 'y' is predicted.

Function 1 Cross entropy	**Function 2** L1	**Function 3** L2	**Function 4** MSE
$\sum_{i=1}^{N} log(q(x_i))$	$\sum_{i=1}^{N} \|\|y_i - x_i\|\|$	$\sum_{i=1}^{N}(y_i - x_i)^2$	$\frac{1}{N}\sum_{i=1}^{N}(y_i - x_i)^2$

2.3.5 Optimizers

An optimization algorithm is used to minimize or maximize the loss function. This is done by calculating the gradient, which is used to decide if the weights and biases should be increased or decreased. There are three optimization algorithms that will be considered in this project, Gradient Descent, Adam, and Adagrad. They are chosen since they have been proved to provide a good result in other projects, and they are the most popular optimization methods [22]. Gradient Descent is chosen since it is seen as the foundation of optimization methods in this domain. Adaptive Moment Estimation, known as Adam is a popular method which calculates the learning rate for each weight/bias. It works well in practice due to it being fast to converge and has a good learning speed. Adagrad is an adaptive gradient method, each iteration it will decrease its learning rate which in theory will make it do smaller modifications the longer it trains, to easier find a minimum.

2.3.6 Upsampling

Upsampling is also called transpose convolution which works as an inverse of a convolution. It is how a smaller set of data is extrapolated with more data.

2.4 Generative Artificial Intelligence

Now that we got some explanation to what a convolutional neural network is, it is time to dive in deeper into how it can be used in generative tasks.

2.4.1 Autoencoder

Autoencoder is a popular use of CNN, it is usually known for its ability to learn using an easy to understand, unsupervised learning. It will use a convolutional network to reduce the data of an object (usually an image) into a latent space (or feature space). It will then use this latent space to reconstruct the original object (image). This shows that an autoencoder is able to extract features from an object in order to reconstruct it. To improve the autoencoder it will reduce the difference between the original and the reconstructed object, with backpropagation. Its main structure consists of an *encoder* and a *decoder*. The purpose of the encoder is to reduce the amount of data needed to describe an object (usually an image), then the decoder will try to reconstruct the object.

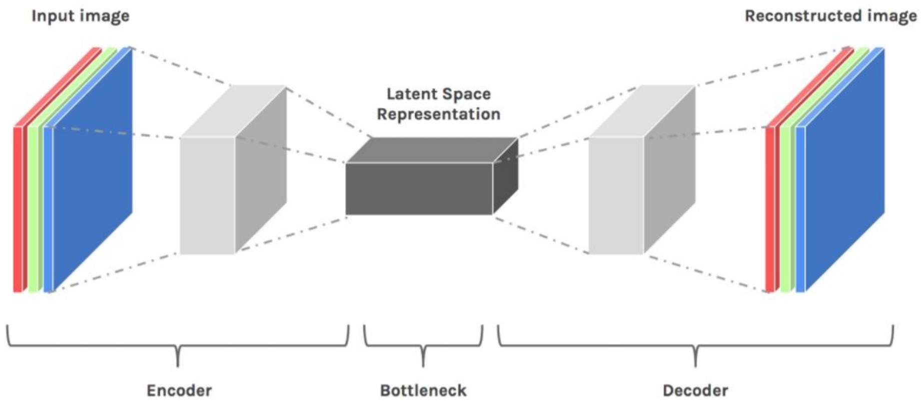


Figure 2.4: Visualization of the training structure of an autoencoder, when the input image and reconstructed image is the same, the latent space is usually good. Adapted from [5].

2.4.2 Generative Adversarial Network

Generative AI had a huge breakthrough in 2014, Goodfellow presented a new neural network model called Generative Adversarial Network (GAN) [9]. The main concept in GAN is, instead of training one neural network against predefined answers (supervised learning), this model consists of two neural networks, a Discriminator, and a Generator. The Discriminator-network's goal is to correctly classify whether an input data is part of a dataset or not. The Generator-network's goal is to fool the discriminator by creating data that it will classify as “part of a dataset”. By training both networks against each other, the Discriminator will become better at identifying objects from the dataset and the Generator will become better at creating data with attributes that match the real dataset.

Chapter 3

Methodology

3.1 Method Background

From the very beginning of this thesis, I could not find any similar published work. However, a few weeks into the project, generating 3D-voxelised objects, became very popular. A lot of papers was made available and I could start to feel the competition. Around three months into the project, Häne released a paper[10] doing a similar project, and giving a proof that this project was possible. Instead of continuing doing what now already was proven to work, I readjusted the focus of this project to find out if CNN could become a sustainable solution towards generating 3D-objects. This was done to give a new point of view, that focus more on what possibilities exists in the near future for 3D generation using CNNs, instead of what is possible today.

This thesis is using an experimental study approach which aims towards the understanding of what problem areas may arise when utilizing convolutional neural networks for professional and commercial use. One additional result that comes from this thesis is an implementation of a generative AI, that can be utilized inside the game engine called Cellbox. The thesis is be based on practical experiences from this ongoing project and on theoretical knowledge learned from courses at KTH and also from various articles found during the pre-study.

Three areas will be focused, in order to consider the program consumer friendly. *Quallity*, *speed*, and *simplicity* as described below.

Quality The resolution of the output. Measured by the number of voxels used to represent an object, this should be as high as possible.

Speed The amount of time that the user needs to wait before a reconstruction is complete. This should be as low as possible.

Simplicity The aim is to make the interaction as simple as possible by using convolutional neural networks.

An empirical research method will be used by constructing experiments and see how well they are performing. The performance is measured by comparing the output from the CNN against a preferred output.

3.2 Research Strategy

An empirical research method is a fundamental baseline though out this project.

3.2.1 Method for Literature Study

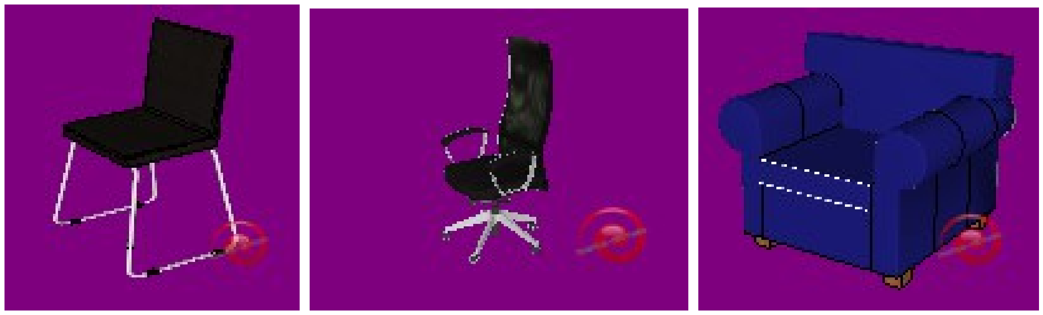
The project is initiated with a literature study to create a good theoretical foundation for this thesis. Keyword-based searches were made on Google to find helpful articles. search phrases including: “3D reconstruction”, “generative adversarial network”, “autoencoder”, “3D GAN”

3.2.2 Data Collection

Data is one of the most important aspects whilst dealing with artificial intelligence. To reduce the scope of this thesis, the dataset is only looking into chairs that can be found in the IKEA dataset [14]. The IKEA dataset is chosen due to its availability and that it is relatively easy to modify and enhance. To make sure that no incomplete data is corrupting the neural network, multiple datasets are considered, using both existing data and generating a new dataset. The IKEA dataset was collected from Joseph J. Lim et al paper from 2013 [14]. The three approaches described below is only used to modify the input, the 3d output is the same for all image dataset. Object files (3D) from the IKEA dataset is converted into a voxelized python format, using a web-browser converter by Arjan Westerdiep [6]. It uses a simple GUI that is easy to understand and the local computers GPU which makes the conversion quick and painless.

3.2.2.1 Image Dataset One

The IKEA dataset contained both 3D object files and 28 images from different angles on each object, See Figure 3.1 for example images.



(a) Image of Bernhard (b) Image of Markus (c) Image of Ektorp

Figure 3.1: Three example images from the first dataset, included in the IKEA dataset.

3.2.2.2 Image Dataset Two

The purpose of this approach is to gain a better quality of the image dataset, in order to make the NN more adjusted to higher resolution images. The 3d objects are imported into 3dsMax [1] and cameras are scripted to take images from different angles of each chair, this gave a dataset of roughly 1000 images. Example of these images can be seen in the Figure 3.2 below.

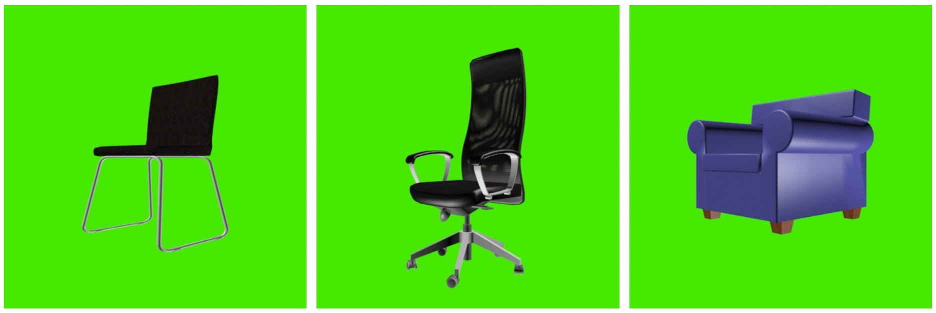

(a) Image of Bernhard (b) Image of Markus (c) Image of Ektorp

Figure 3.2: Three example images from the second dataset, generated with 3dsMax.

3.2.2.3 Image Dataset Three

To increase the dataset from approach two, an extra 500 photo images are extracted manually from Google with queries like "Bernhard ikea" to get images on the chair called "Bernhard" from the IKEA dataset. A test dataset is also added by taking photos on chairs around the office at Mutate AB. Example of these images can be seen below in Figure 3.3. The images are captured with the camera on a Samsung S7 mobile phone, with settings to make the photos squared.

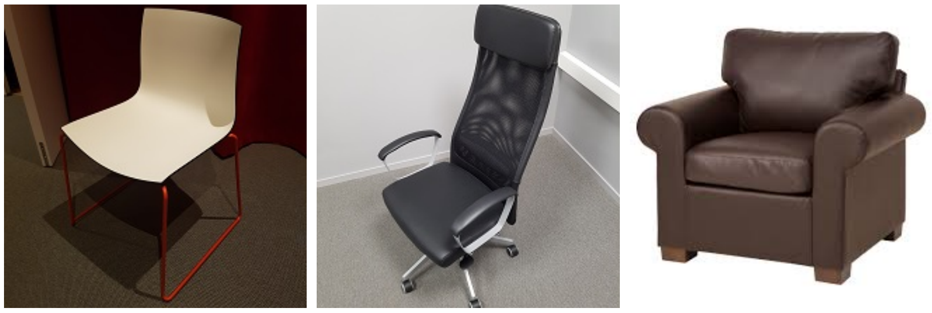

(a) Image of Bernhard (b) Image of Markus (c) Image of Ektorp

Figure 3.3: Three images from the testing dataset.

3.3 Result Extraction

The result of this project will use MSE to gain a quantitative data-driven comparison between ground truth (GT) object and the output. The NN is improved by reducing the MSE. Additionally, there will be an ocular graphical comparison, this will be considered a qualitative comparison which will give the reader a visual understanding of the result.

3.4 Experiments

The experiments conducted is done as an empirical binary search tree structure, for each test that is based on fine-tuning numerical hyperparameters. This structure gives a $O(log(n))$ amount of tests, which is the upper limit to the worst case scenario, and n is the total number of possible test. Where this type of testing is not applicable, every solution within the scope of the thesis is tested.

Empirical binary search tree structure, Step by step:

1. Find min & max value for interval.
2. Run test on min & max individually.
3. Choose new min or max.
   (a) If min is better than max $\Rightarrow$ new max is the median value.
   (b) If max is better than min $\Rightarrow$ new min is the median value.
4. Repeat until only one value remains.

3.4.1 First Neural Network

The first neural network is mainly tested on the different datasets, to see that improvements have been achieved when switching dataset. Since the first neural network looks the same as in similar projects [23], only the dropout hyperparameter is tuned, the rest is assumed to be optimized.

3.4.1.1 Dataset testing

Three different datasets (see Section 3.2.2) is tested, the goal of this testing is to see if it will be a big difference depending on the quality of the data. The different datasets are accessible in different directories, thus each will be trained individually by changing the input directory. The result that will be compared is the MSE (lower is better) and the difference in MSE between training data and test data, the lower the difference is the more generalized the neural network is.

3.4.1.2 Dropout

Only the third dataset is considered for this test since it is shown to work best out of the three datasets (See section 5.2.). The starting minimum dropout will be '0.1' (90%

chance to drop) and the maximum dropout will be '1.0' (0% chance to drop). Each training session will consist of 500 epoch and the lowest MSE during this training will be considered. The following dropout values will be tested: [0.1, 0.5, 0.60, 0.75, 1.0].

3.4.2 Second Neural Network

The experiments for the second neural network is testing Dropout rates, activation functions, loss functions, and optimizer functions.

3.4.2.1 Activation Function

The tested activation function is Sigmoid, Tanh, ReLu and leaky ReLu (lReLu). Since Sigmoid, Tanh, and Softplus may suffer from vanishing gradient, thus only ReLu and lRelu is considered for the hidden layer. Sigmoid, Tanh and Softplus will be used in the output layer. All architectures shown in Appendix B is trained and the MSE result is compared to decide which architecture to use.

The implementation of Sigmoid, Softplus, Tanh, and ReLU exists in the Tensorflow library, thus they are used. However, lReLU is not included, the code snippet below is used to calculate the lReLU, where the α-variable is fixed to 0.2.

Code snippet: $tf.maximum(\alpha * x, x)$

3.4.2.2 Droupout Rate

The empirical binary search tree structure described in Section 3.4.1.1 is used with the following start value: minimum = 0.1, maximum = 1.0. The tested dropouts ends up being: [0.1, 0.5, 0.75, 0.90, 0.95, 0.99, 1.0].

3.4.2.3 Loss Function

To test which loss function should be used, the brute force technique is used. Each loss function described in Subsection 2.3.4. is trained individually. Since the loss are calculated differently, an ocular inspection of the result will be the deciding factor of which is assumed to be best.

3.4.2.4 Optimizer

The optimizer testing uses the brute force technique where each optimizer is trained individually and the one that converges fastest with lowest MSE is chosen. The tested optimization methods are Gradient Descent, Adam, and Adagrad (see Subsection 2.3.5)

3.4.3 Benchmark On Both Neural Network Together

To test the total generation speed of the neural networks, a benchmark is done individually on LoD 0 to 6. This test is used find the difference in time when increasing the output resolution.

Chapter 4

Implementation

For the purpose of reproducibility, this chapter starts with the environmental setup, followed by the architecture of the CNN.

4.1 Environment

The software and hardware used in this project:

- † Python 3.5
- † Tensorflow 1.3
- † Windows 10
- † GPU: NVIDIA GeForce GTX 1070 8GB (1.7715GHz)
- † 16GB RAM
- † Dataset Size: input images $\sim$ 1500, output objects $\sim$ 30

4.2 Architecture

The basic architecture consists of two convolutional neural networks (CNN), one called 'Convolutional Autoencoder' and the second called 'Convolutional 3D Decoder' see architecture in Figure 4.1 below. The task of the first CNN is to extract features from an image, this feature is then fed as input to the second CNN. The second CNN task is to generate an octree structured voxel space from the latent space. This process is repeated to increase the resolution of the voxel-space, see the right iterative side of Figure 4.1. Due to the 3D object generator being harder to generalize, it will be the feature extractions job to make the latent space similar for the same object, independent of the angle of the object in the input image. To see how the class hierarchy looks, see Appendix C

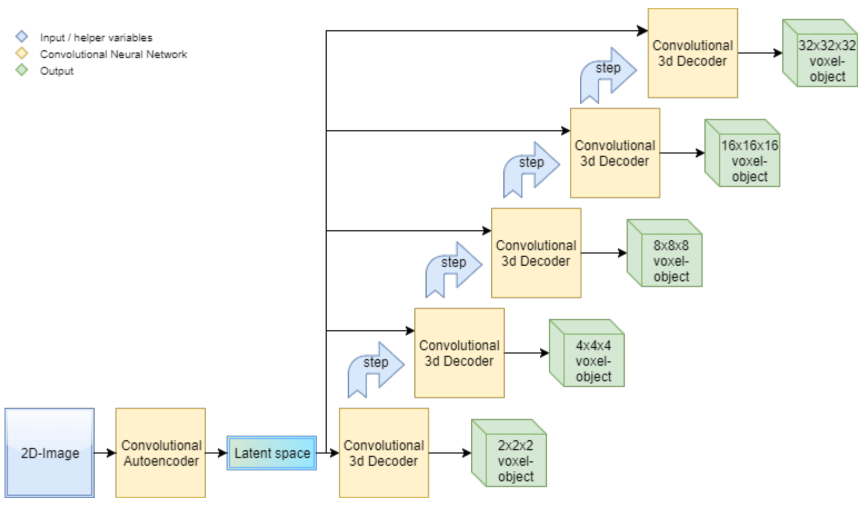


Figure 4.1: Architectural structure of neural networks used during generation.

4.2.1 Convolutional Feature Extraction Architecture

The feature extraction is heavily based on the same model as in Jiajun Wu paper 2016 [23]. One key difference is in the data. To make it more generalized in recognizing objects from different angles, all images of the same object is reconstructed to the same output. This is will also make it easier for the 3d generative NN since the angle of the input is irrelevant for a 3D object.

4.2.2 Convolutional 3D Generator

Generating 3d object is not a trivial task, thus two different approaches were taken into consideration. Conditional GAN was the first approach, due to it being a hyped area in content generation and is proved to give an OK result [23]. The second approach is a conditional autoencoder that uses an octree structure. Early in the development of the Conditional GAN, it was shown that it had an exponential growth in memory usage while increasing the resolution. This is not a stable solution, thus approach Two is focused.

4.2.2.1 Approach One: Conditional Generative Adversarial Network

The idea of the conditional GAN is to create two competing neural network. One of the networks tries to correctly classify which images come from the actual dataset, whilst the other neural network tries to fool the classifier that its generated content is from that same dataset. The purpose of doing this is to improve the quality of the generated objects without doing a direct comparison against an answer, making the NN more generalized. Some assumptions made to make the training fairer whilst building the networks were:

† Only training Generator while Discriminators accuracy is above 20%.

† Only train Discriminator while Discriminators accuracy is below 80%.

† The Discriminator will get a head start of 10 training sessions.

The first two bullets make the network improve alongside each other, instead of one being the superior which will make it harder for the other neural network to improve. The 3rd bullet is so that the Discriminator will have a better starting point, it will also improve the Generators start since it is better to compete against a slightly better component.

4.2.2.2 Approach Two: Conditional Autoencoder

The idea of this conditional autoencoder structure is to make the NN generate an object in steps. The foundation is inspired by a sculpturing process, which in this case starts of with a single voxel that is split into smaller pieces that are being validated to remove unnecessary voxel/space, this results in enhancing the resolution (See Figure 4.2. Each step is trained individually, which will enable the NN to be able to train on creating as high resolution as possible. The first step will generate 2x2x2 (8 voxels), for example, this step could find the proportions of the object. An object that is twice as high as width would only need to use 4 voxels (one side of the cube). To understand the power behind this, let us say that we would like to create an object with the quality of 64x64x64 (x,y,z) voxels. In this case, the first step would discard half of the voxels (64*64*64 /2 = 131072 voxels/permutations), which is a lot. Figure 4.3 shows how each iteration of the generation could improve the quality of a generated chair.

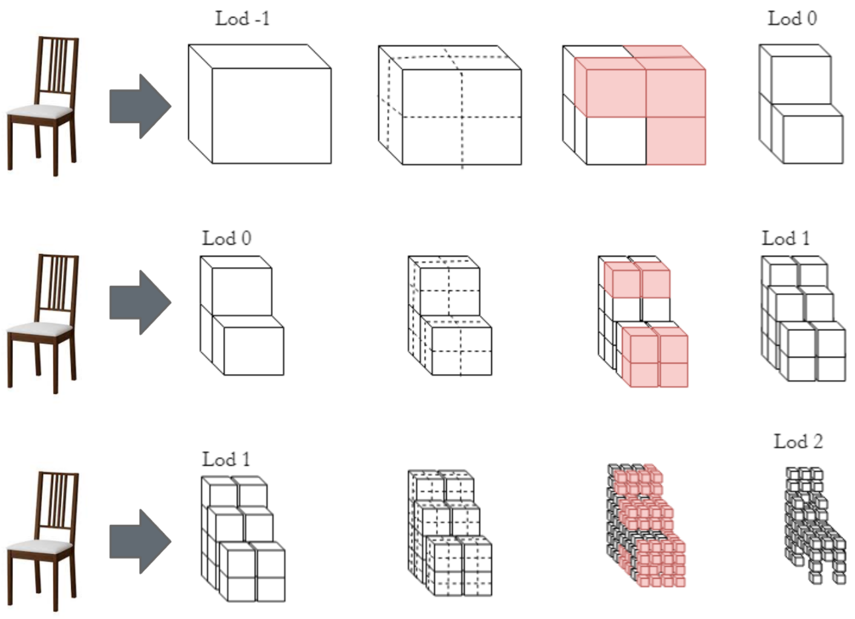


Figure 4.2: Visual representation of how each voxel is split and validated between each LoD, in the process of increasing the resolution.

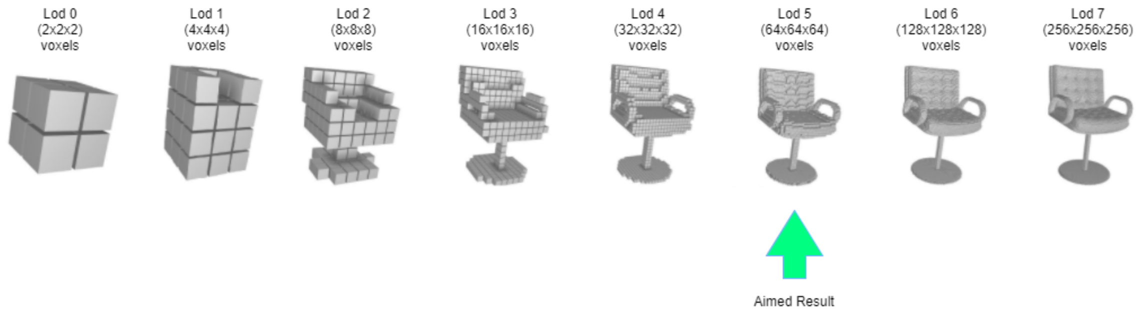


Figure 4.3: Shows an example of the iterative process of generating a chair. The green arrow is pointing at the quality that this thesis is aiming for.

The total amount of voxels required to represent a chair in the training data set, corresponding to a 64x64x64 quality, is between ~ 3000 and ~ 20000 voxels. Comparing the worst case (~ 20.000) against permutations for each voxel in the space, this would reduce the total number of predicted voxels to ~ 20000 from 262144 ($\sim 7.6\%$) and the best case is ~ 3000 predictions down from 262144 ($\sim 1.1\%$). The fact is that the percentage of the total voxel-space decreases by each LoD (More gain from increased quality), see the percentage of total voxel used in Table 4.1 for different LoD.

Table 4.1
A table showing the percentage of voxels used for each LoD.

Level of Detail	**Hight / Width / Depth**	**Percentage of used voxels**
LoD	Nr of voxels	%
0	2/2/2	37.5 – 100%
1	4/4/4	21.9 – 82.8%
2	8/8/8	10.4 – 48.8%
3	16/16/16	4.7 – 26.1%
4	32/32/32	2.7 – 13.5%
5	64/64/64	1.3 – 6.7%
6	128/128/128	0.7 – 3.5%
7	256/256/256	0.3 – 2.0%

Chapter 5

Result and Analysis

For the purpose of validity, this section shows both graphical and numerical result from the experiments defined in Chapter 4.

5.1 First Neural Network: Feature Extraction

This section shows result from different iterations in the process of optimizing the feature extraction.

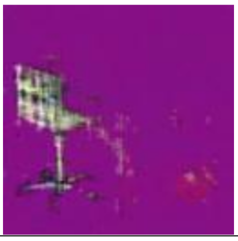
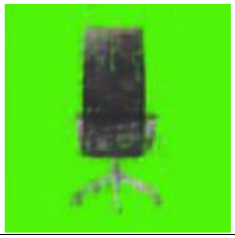
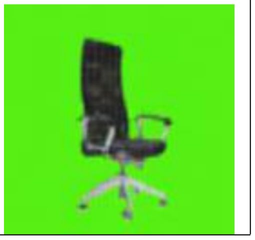
5.1.1 Dataset Modification Result

Due to the bad quality of dataset one (see Subsection 3.2.2.1), it was hard for the NN to find correct shapes. This made the NN good at predicting similar images, but not as good with realistic photos taken with a camera.
The second dataset (see Subsection 3.2.2.2) gave a big improvement on real photos, shapes were more precise which gave a good impact on the neural network.
The third dataset (see Subsection 3.2.2.3) further improved the NN. It added some diversity to the model, making the NN more generalized which can be seen by the lowered difference between training loss and testing loss in Table 5.1 below.

5.1.2 Dropout

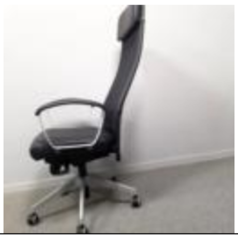
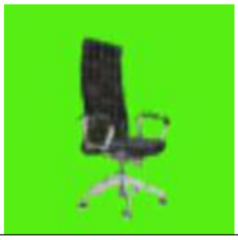
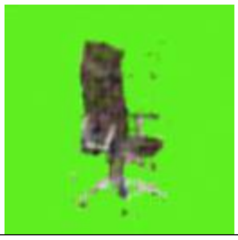
The best dropout rate was 0.5 which is shown in Table 5.2 below. It is easy to see the similarity, however, the NN that trained with 0.5 dropout rate is more generalized

Table 5.1
Result MSE from training on different datasets, The bottom row is a 2D-image reconstruction, the purpose of those is only to give the reader an understanding of the numbers. (Low loss is good).

Input	**Dataset One**	**Dataset Two**	**Dataset Three**
Loss	0.31739	0.254156	0.165701
			

which result in more clear shapes.

Table 5.2
Graphical result to visualize the improvement from different dropout fractions.

Input	**Output dropout=0.5**	**Output dropout=1.0**
		

5.2 Second Neural Network: 3D Generation

This Section shows the result from different iterations while training the 3D generation CNN. The result is gathered from the process of finding the optimal dropout, activation function, and the loss function. This is followed up with a graphical visualization of both good and bad result. Ending this section with a visualization of each LoD, generated during a 3D reconstruction.

5.2.1 Dropout

The result was improved as the dropout was increased (see Table 5.3), due to the generalization part being in the feature extraction, this was an expected result.

Table 5.3
MSE from the empirical approach to optimize the dropout.

Dropout	0.5	0.75	0.90	0.95	0.99	1.0
Loss	0.27399	0.1710	0.06546	0.0405	0.02028	0.0094869

5.2.2 Activation Function

The testing of different setups for activation functions is done by training each setup on 50 epochs. The tests use the same activation function for input layer and all hidden layers, the output layer may differ. As can be seen in Table 5.4 below, leaky ReLU for hidden layers and Tanh for the output layer gave the best (lowest) result.

Table 5.4
MSE result after 50 training epochs (Low is good), using different activation functions (AF). Hidden AF is used for input & hidden layers, output AF is only used for the output layer.

Hidden AF	Output AF	Loss
ReLU	None	0.2659
ReLU	ReLU	0.3188
ReLU	Sigmoid	0.07869
ReLU	Tanh	0.17724
lReLU	None	0.1612
lReLU	lReLU	0.07082
lReLU	Sigmoid	0.0483
lReLU	Tanh	0.00879
lReLU	Softplus	0.02134

5.2.3 Loss Function

Testing of different loss function is done by training each loss function mentioned in Section 2.3.4, for 50 epochs on LoD 1 (see Table 5.5).

Table 5.5
Loss calculated by taking the mean value of the absolute difference between all predicted value and the correct output. Similar to L1-loss function.

Loss function	MSE	L1	Huber	Crossentropy
Loss	0.0087937755	0.002106585	0.0150602	0.671875

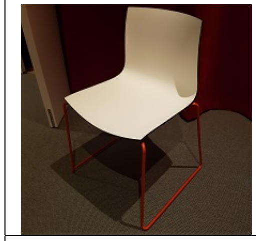
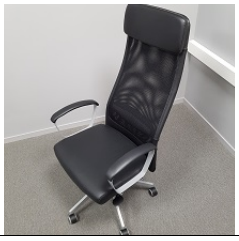
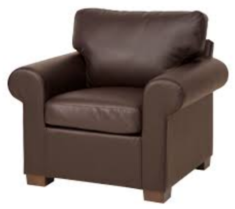
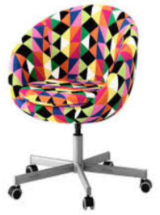
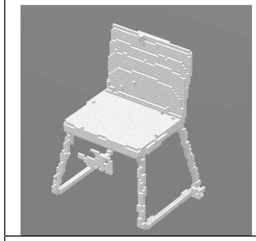
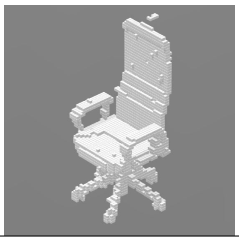
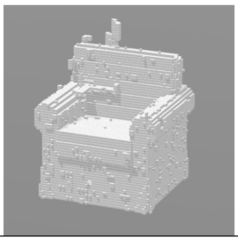
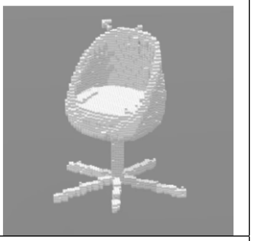
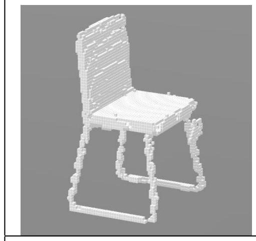
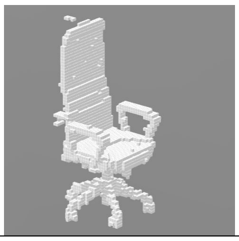
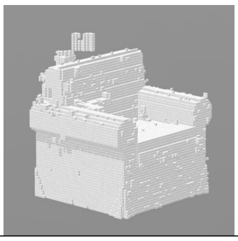
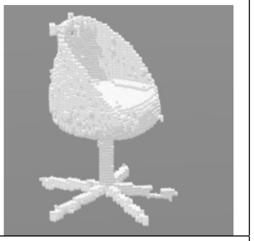
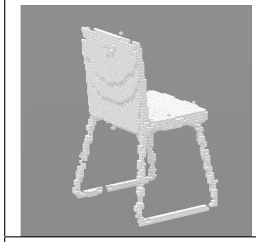
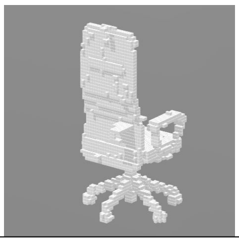
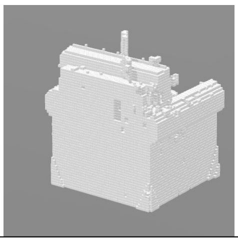
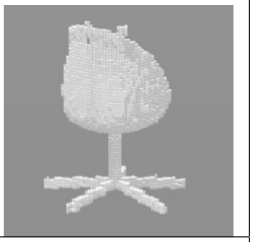
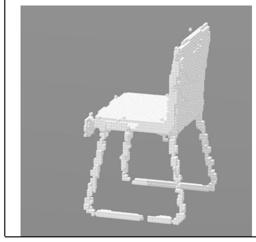
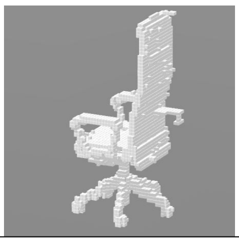
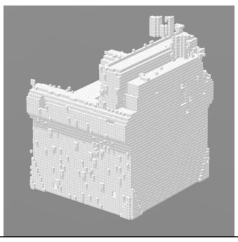
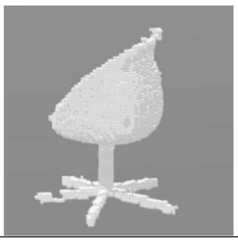
5.2.4 Graphical Test Cases

This subsection will display both good and bad result, generated from the trained neural network. It also shows partial result during generation in form of different LoD.

5.2.4.1 Successful Result

In Table 5.6 below, some of the better graphical results are shown. The NN has seen the object type that is in these images, but not the input images.

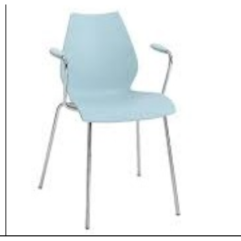
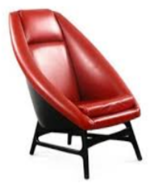
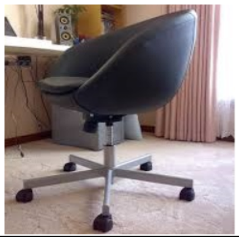
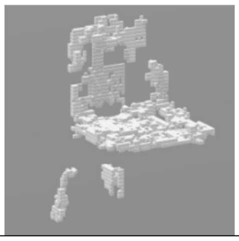
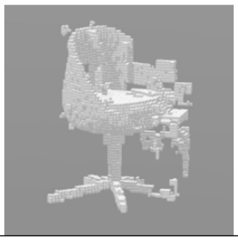
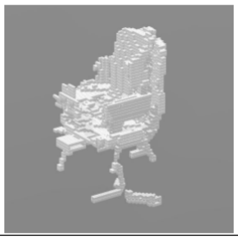
Table 5.6
Visual result of a generated object from 4 different angles.

Case One	Case Two	Case Three	Case Four
			
			
			
			
			

5.2.4.2 Less Good Result

This is the result of two images on models that it has never seen before, and one image of a model that it has seen (see Table 5.7). However, it is not able to recognize the object due to that the angle of the object is unknown.

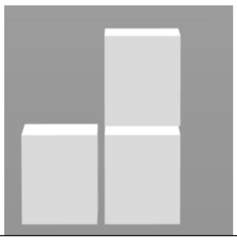
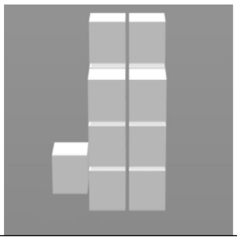
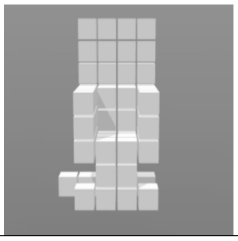
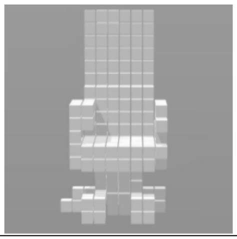
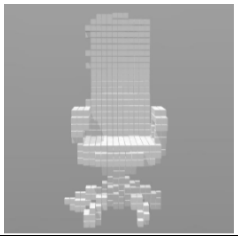
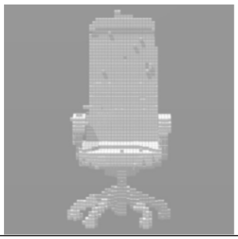
Table 5.7
Visual result of three cases that gives bad output.

Unknow model	Unknow model	Unknow angle of model
		
		

5.2.4.3 Level of Detail Sub-result

Training on different LoD levels was a success, Table 5.8 below shows the result of each iteration progress when the neural network is generating a chair. The input is the same as $CaseTwo$ in Table 5.6.

Table 5.8
Visual result for each iteration in the generation process.

LoD 0	**LoD 1**	**LoD 2**
		
LoD 3	**LoD 4**	**LoD 5**
		

5.3 Benchmark

The time to generate a chair in different LoD is shown in Table 5.9. A formula to calculate the generation time is: $[0.2 * 8^{LoD-1} * Voxels\%]$. This conclusion is drawn by comparing the result in Table 5.9 with Table 4.1.

Table 5.9
A benchmark to show the generation time for different level of details (Average of 10 reconstructions).

	LoD 0	**LoD 1**	**LoD 2**	**LoD 3**	**LoD 4**	**LoD 5**	**LoD 6**
Time (sec)	0.2	1.3	5.1	18	73	260	1500
#Voxels	8	8^2	8^3	8^4	8^5	8^6	8^7
% used voxels	70%	52%	15%	8%	4%	2%	1%

Chapter 6

Conclusion and Discussion

This chapter compares the result from Chapter 5 with the purpose in Chapter 1 to create a summary of the project. This is followed by a reflection of the work and proposal for future work.

6.1 Conclusion and Contribution

By looking at the result, it is possible to see a future where a similar system could be utilized for 3D generation. The key contribution of this project the gained knowledge about different problem areas for generating 3D voxel-based objects from 2D images. However, with the current implementation, the output is too unreliable and slow for high resolution.

6.1.1 Comparison against purpose

† How significant is the trade-off between generation speed and output resolution when generating 3D voxelized objects from a 2D RGB-image, using convolutional neural networks?

The trade-off is in our favor. As shown in Section 5.3 the gained resolution grows exponentially by each LoD, whilst the generation time is not exponential. Since the speed has a correlation to the amount of permutation needed to go from one LoD to another, and the permutation needed is getting a lower percentage value by each increasing LoD. In conclusion, increasing the resolution is also increasing the effectiveness. However, the total number of permutations is still growing rapidly, which means that even though a higher quality is technically more effective due to the trade-off, it is still not viable to generate high resolution objects, since the total generation time will get to high (30 minutes on LoD 6).

6.1.2 Ethics and Sustainability

An ethical aspect that is considered for this project is to be protective of the training data. It should not be possible to easily recreate the original training dataset if it consists of sensitive data.

According to the United Nations Sustainable Development Goal (SDG) 8 (Economic growth), it is important to encourage and help with the formalization of small-sized enterprises. This project could give smaller businesses a better chance in the gaming industry since they might not have the starting capital to hire a graphical team from the very beginning. With an artificial intelligence that can help with creating 3D environments, it would be possible to create games more quickly and give smaller businesses a better initial position.

6.2 Reflections on the Solution

One decision that I think has had a negative impact on this project is that the bad result from the first NN, due to dataset One, forced an adjustment on the training. This adjustment was that the output of all images on the same object should be similar. I think this may have been a bad decision since it creates a bottleneck for the second NN, this is why the second NN gain better result without any dropout. Which can be seen as overfitting it in order to work well with the first NN. Due to the time constraint of this project, there was not enough time to remake the first NN when better input data was available.

Pros of the solution created in this project is that it does not have any restrictions on how good resolution it can generate. The output is completely generated by the AI which is good since there is no need for an asset storage for every 3d object that is possible to create. Also, A model like this could be utilized in a Minecraft environment since it does not require a high-quality output.

Cons of the solution created in this project is that the generation time is slow for high-resolution output ($64^3 \leq Voxels$). Also that the NN is fragile to unrecognized angles of objects and that this NN is only trained to be able to generate objects that it has previous knowledge about.

6.3 The Project

The goals have been adapted throughout the entire project since generative AI has been a popular subject, which resulted in many other papers being released with a similar focus area. This lead to a very agile way of work. The first similar paper that I found was Jiajun Wu [23], this gave a good starting point for the project. However, I quickly found out that the GAN structure would not work for generating bigger models since the memory usage grows exponentially when increasing the size of the output. Around the time I figured this out I talked to my supervisor at Mutate and found out that they were using an octree structure in their game engine. This was when I realized that I could use a similar structure to generate 3d objects. By starting to generate the 'lowest' quality (LoD 0), which is 8 for an octree (ignoring the one initial node, since it is just a single voxel). After I had been working on this for a two week period, there was a new paper released by Christian Häne [10], this paper was also using an octree structure. It was not made open source but contained some values for their hyperparameters, this was helpful since I received a good starting point for tuning my own hyperparameters. A third paper that was released during this project was the 'Dynamic Routing Between Capsules' by Geoffrey E. Hinton [20]. The method described in this paper could be used to improve the generalization of the first NN: "feature extraction", however, this was released late in the development process which is why it will be put as a proposal for future work, to do more testing with Capsule Networks (CapsNet).

6.4 Future Work

This project has improvements that could be done in a few different areas. The first area I will call near-future work, which is to make it more generalized. This could be achieved by modifying the first neural network to compare the reconstruction to the input instead of the fixed image, thereafter increase the dropout in the second neural network.

The second area I will call the experimental-future work, which is another approach to gain a more generalized AI. As mentioned in Section 6.3, Hinton has released a paper on CapsNet [20]. I think it would be an interesting task to replace the first NN with a capsule network since it is shown to be less fragile to unseen variation in the data, in some cases.

The third area, technical-future work. This is a problem area that exists but will need some research to possibly find a solution. Since the current model will only activate the weights that are needed to generate each LoD and the needed weights is decided from the previous LoD, there would be a huge improvement to find a faster way to load weights into memory.

References

[1] Autodesk [Accessed 2017-01-17], *3ds Max [Software]*, Autodesk.

URL: *https://www.autodesk.se/products/3ds-max/overview*

[2] Brownlee, J. [2015 (Accessed 2018-01-05)], *Basic Concepts in Machine Learning [bloggpost]*, Machinelearningmastery.

URL: *https://machinelearningmastery.com/basic-concepts-in-machine-learning/*

[3] Chioka [2013 (Accessed 2017-11-28)], *Differences between L1 and L2 as Loss Function and Regularization [blogpost]*, Chioka.

URL: *http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/*

[4] Deshpande, A. [2016 (Accessed 2017-11-01)], *A Beginner's Guide To Understanding Convolutional Neural Networks [blogpost]*, Github, **URL**: https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner

[5] Despois, J. [2017 (Accessed 2017-10-18)], *Autoencoders Deep Learning bits [blog-post]*, Hackernoon.
URL: *https://hackernoon.com/autoencoders-deep-learning-bits-1-11731e200694*

[6] Drububu [Accessed 2017-11-06], *voxelizer, convert your 3D model to voxels online [software]*, Drububu.
URL: *http://drububu.com/miscellaneous/voxelizer/*

[7] Encyclopedia.com [Accessed 2018-01-22], *"octree." A Dictionary of Computing.*, Encyclopedia.
URL: *http://www.encyclopedia.com/computing/dictionaries-thesauruses-pictures-and-press-releases/octree*

[8] expertsystem.com [Accessed 2018-01-20], *What is Machine Learning? A definition*, Expertsystems.
URL: *http://www.expertsystem.com/machine-learning-definition/*

[9] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y. [2014], *Generative Adversarial Networks*, arXiv.
URL: *http://adsabs.harvard.edu/abs/2014arXiv1406.2661G*

[10] Häne, C., Tulsiani, S. and Malik, J. [2017], *Hierarchical Surface Prediction for 3D Object Reconstruction*, arXiv.
URL: *http://arxiv.org/abs/1704.00710*

[11] Kafunah, J. [2017 (Accessed 2017-11-01)], *Backpropagation in Convolutional Neural Networks [blogpost]*, Jefkine, **URL**: http://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/.

[12] Karpathy, A. [2016 (Accessed 2017-10-18)], *Generative Models [blogpost]*, Openai.
URL: *https://blog.openai.com/generative-models/*

[13] Krizhevsky, A., Ilya, S. and Hinton, G. E. [2012], 'Imagenet classification with deep convolutional neural networks', *Advances in Neural Information Processing Systems 25* pp. 1097–1105.

[14] Lim, J. J., Pirsiavash, H. and Torralba, A. [2013], 'Parsing ikea objects: Fine pose estimation', *2013 IEEE International Conference on Computer Vision* pp. 2992–2999.

[15] Nielsen, M. [2015 (Accessed 2017-10-18)], *Neural networks and deep learning*, Determination Press.
URL: *http://neuralnetworksanddeeplearning.com/*

[16] oxforddictionaries.com [Accessed 2018-01-22], *Voxel*, Oxforddictionaries.
URL: *https://en.oxforddictionaries.com/definition/voxel*

[17] Report of the Secretary-General [2017-05-11], *Progress towards the Sustainable Development Goals*, UN, **URL**: http://www.un.org/ga/search/view_doc.asp?symbol=E/2017/66&Lang=E.

[18] Rouse, M. [2017 (Accessed 2017-10-18)], *machine learning*, Whatis.
URL: *http://whatis.techtarget.com/definition/machine-learning*

[19] Russakovsky, O. [2015], *ImageNet Large Scale Visual Recognition Challenge*, International Journal of Computer Vision (IJCV).

[20] Sabour, S., Frosst, N. and Hinton, G. E. [2017], *Dynamic Routing Between Capsules*, arXiv.
URL: *http://arxiv.org/abs/1710.09829*

[21] techopedia.com [Accessed 2018-01-22], 'Level of detail', *Techopedia* .
URL: *https://www.techopedia.com/definition/11791/level-of-detail-lod*

[22] Walia, A. S. [2017 (Accessed 2017-12-21)], *Types of Optimization Algorithms used in Neural Networks and Ways*, Towardsdatascience, **URL**: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f.

[23] Wu, J., Zhang, C., Xue, T., Freeman, W. T. and Tenenbaum, J. B. [2016], *Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling*, arXiv.
URL: *http://arxiv.org/abs/1610.07584*

Appendix A

Files

data.py

Helper class, used to parse input data into a usable format

voxels.py

Helper class, used to interpret voxel object.

cube_helper.py

Helper class, used to interpret cube (ex. converting octree-index into binary-index).

AIFeatureExtraction.py

Convolutional Neural Network, used to train and test AI for extracting features from 2d RGB-images.

AIVoxelReconstruction.py

Convolutional Neural Network, used to train and test AI for reconstructing 3d voxel object.

Appendix B

Activiation Function Structure

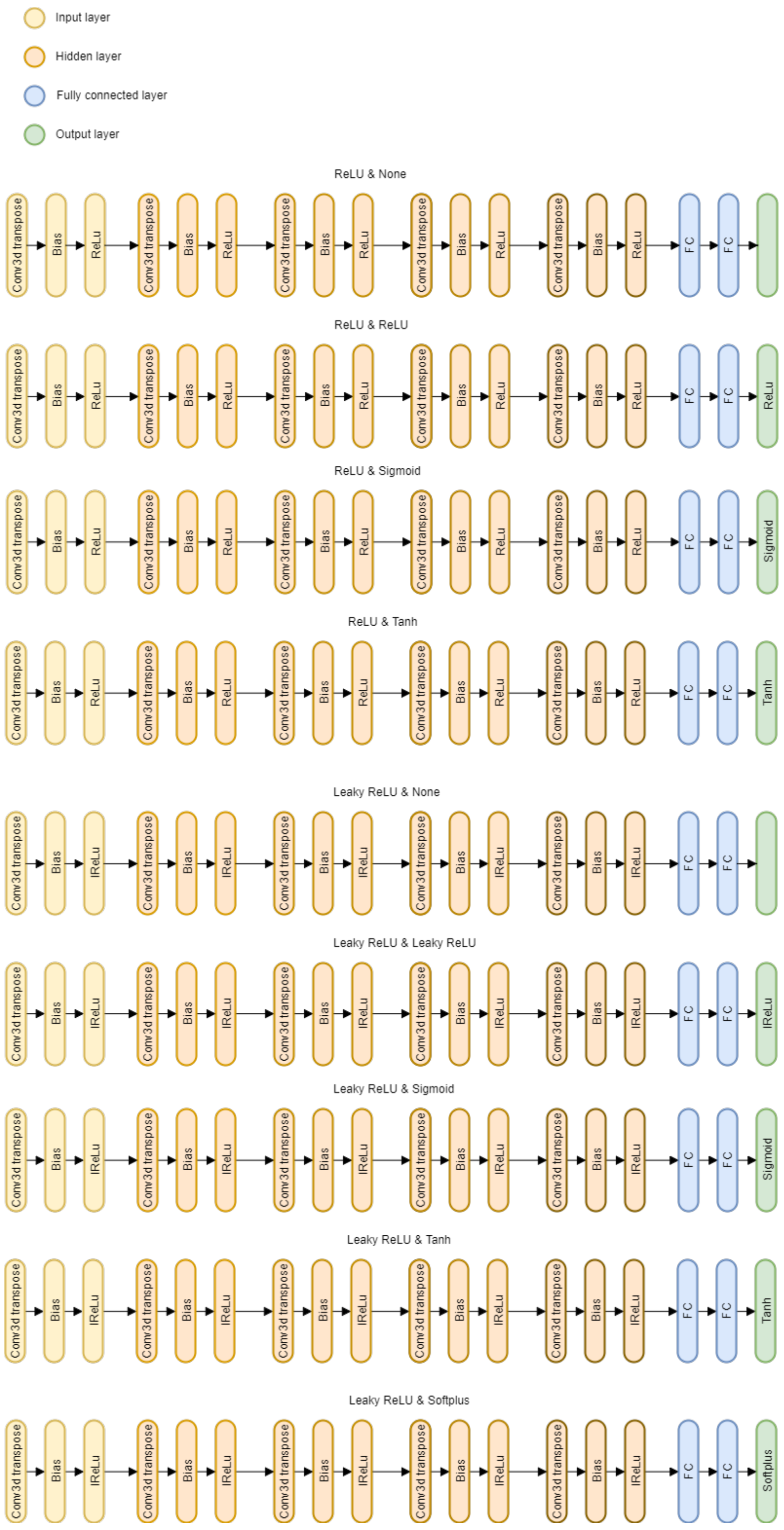


Figure B.1: Different architecture tested to find optimal structure of activation functions.

Appendix C

Class Hierarchy

Class hierarchy is shown. AIVoxelReconstruction contains the main functionality for training and generation. The generation function is testAI which takes 3 input parameters, the first is a location of where the model is stored; second is which chair to convert into 3d; third is which Level of Detail.

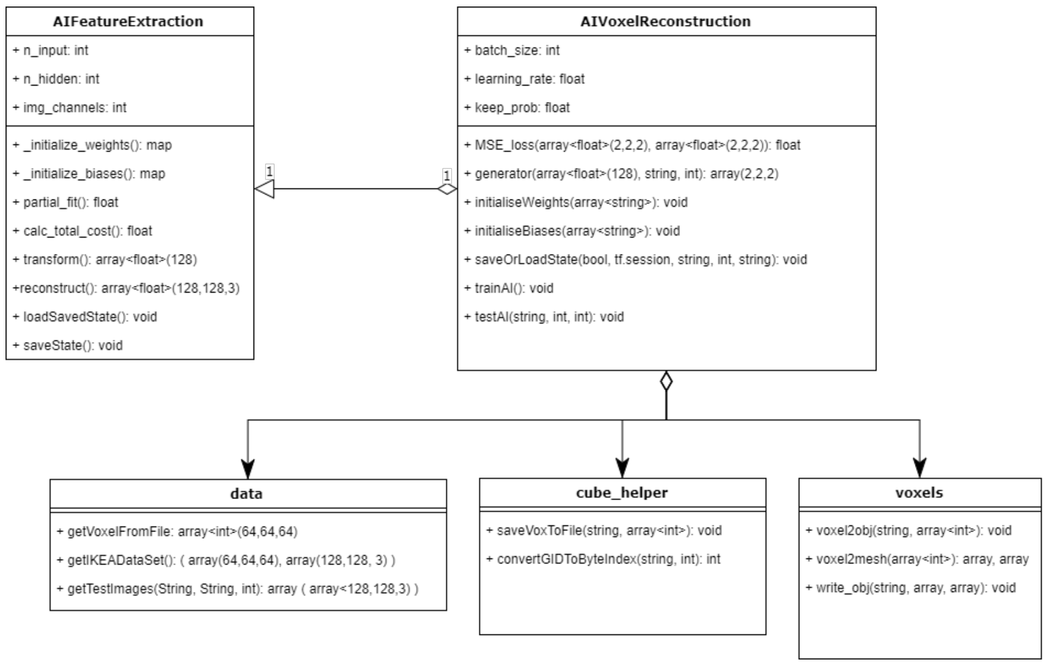

AIFeatureExtraction
+ n_input: int
+ n_hidden: int
+ img_channels: int
+ _initialize_weights(): map
+ _initialize_biases(): map
+ partial_fit(): float
+ calc_total_cost(): float
+ transform(): array<float>(128)
+reconstruct(): array<float>(128,128,3)
+ loadSavedState(): void
+ saveState(): void
1
1
AIVoxelReconstruction
+ batch_size: int
+ learning_rate: float
+ keep_prob: float
+ MSE_loss(array<float>(2,2,2), array<float>(2,2,2)): float
+ generator(array<float>(128), string, int): array(2,2,2)
+ initialiseWeights(array<string>): void
+ initialiseBiases(array<string>): void
+ saveOrLoadState(bool, tf.session, string, int, string): void
+ trainAI(): void
+ testAI(string, int, int): void
data
+ getVoxelFromFile: array<int>(64,64,64)
+ getIKEADataSet(): (array(64,64,64), array(128,128, 3))
+ getTestImages(String, String, int): array (array<128,128,3))
cube_helper
+ saveVoxToFile(string, array<int>): void
+ convertGIDToByteIndex(string, int): int
voxels
+ voxel2obj(string, array<int>): void
+ voxel2mesh(array<int>): array, array
+ write_obj(string, array, array): void

Figure C.1: Class diagram, visualizing relation between classes.

TRITA TRITA-EECS-EX-2018:92