Sketch to 3D Model using Generative Query Networks

Max Nihlén Ramström
920429-3055
maxra@kth.se

Abstract

For digital artists and animators, translating an idea from a rough sketch to a 3D model is a time consuming process requiring a plethora of different software. In this work, a Generative Model which can directly generate images of 3D models from arbitrary view points by observing sketched 2D images is presented. The model is based on Generative Query Networks and two different generative models were tested for generating new images, the first a Variational Auto Encoder and the other a Generative Adversarial Network. The model learns to produce new images from any queried view point allowing it to perform so called mental rotation of an object as if a 3D model had been generated. A paired dataset containing images of 3D models, the view point from where each image is captured and corresponding sketch versions was created in order to train the model. It was found that the Variational Auto Encoder could create plausible images from as little as a single sketch while the Generative Adversarial Network failed to correctly condition on the given sketches.

Sammanfattning

För digitala artister och animatörer är processen att gå ifrån en idé i form av en sketch till en färdig 3D-modell tidskrävande och sträcker sig över en mängd olika mjukvaror. Detta arbete presenterar en Generativ Modell som direkt kan generera bilder av en 3D-modell ifrån sketchade bilder i 2D. Modellen är baserad på Generative Query Networks och två olika Generativa Modeller testades för att generera nya bilder, den första en Variational Auto Encoder och den andra en Generative Adversarial Network. Modellen lär sig att skapa nya bilder ifrån godtyckliga synvinklar vilket tillåter den att utföra så kallad mental rotation av ett objekt på samma sätt som om en 3D-modell hade genererats. För att kunna träna modellen skapades ett dataset där bilder sparades både i ursprungs- samt i sketchform tillsammans med synvinklarna där bilderna tagits ifrån. Modellen som använde sig av en Variational Auto Encoder visade sig kunna generera trovärdiga bilder efter att endast ha observerat en sketch medan modellen som använde ett Generative Adversarial Network misslyckades med att betinga de genererade bilderna på de sketcher den observerat.

Glossary

GAN: Generative Adversarial Network, a type of generative model.
VAE: Variational Autoencoder, a type of generative model.
GQN: Generative Query Network, a type of generative model for generating images of 3D scenes and models.
Convolutional DRAW: a recurrent Variational Autoencoder.
VAEQN: Generative Query Network using the Convolution DRAW as generative model.
GAQN: Generative Query Network using a GAN as generative model.

Contents

1 Introduction

1.1 Background

Digital artists, animators and people working with designing and creating 3D models spend a lot of time going from an idea to a model ready to be evaluated. This is mainly due to the fact that many different software needs to be used during the process. The time consuming procedure of creating a single model hampers the artist's ability to quickly try out new ideas. With a more efficient way of generating 3D models, artists could spend more time being creative.

1.2 Thesis Objective

In this thesis an attempt is made to train a neural network to render images of 3D objects from a set of 2D sketches. If an assumption is made that the natural images that the neural network is trying to render are drawn from a probability distribution which fully describes the world and its properties, the network can try to learn this distribution to then be able to sample images from it. This is something generative models are especially designed to do.

The Google DeepMind group developed a generative model called Generative Query Networks (GQN) [13] which can take an image of an object and the location from where that image was captured as input and generate a new image at a previously unseen view point. In this way, a 3D model can be generated from the original image by simply generating new images from different view points. More precisely, the GQN first creates a so called neural representation of a 3D model by sending observed images of it and their view points through a convolutional neural network and then summing up the contribution from each view point. After that, the neural representation is used by a generative model to generate a new image as seen from a queried location.

This thesis investigate if the GQN can learn to produce new images of a 3D model if the images it observes are of that model in sketch form. If this is possible, an artist could quickly view a potential 3D model by simply sketching it and then letting the GQN generate images of what it might look like. While the GQN performed very well in DeepMind's experiments, the results where obtained on a simple dataset representing Shepard-Metzler objects [43]. To be able to generate useful models from sketches, it was thus

necessary to try out whether the model had the capacity to generalise to a more realistic and diverse dataset.

While there exists public datasets for both 3D models and sketched images, to the best of the author's knowledge, there exists no dataset where single models are captured from different view points and where the view points are saved together with a sketch version of the image. It was thus necessary to create a novel dataset containing all these features. Creating a large enough dataset with real sketches would take aeons and therefore, to test if the concept was feasible, it was decided that a sketch like dataset with synthetic sketches would be created programmatically.

The original GQN model uses a Variational Auto Encoder (VAE) for generating images but the authors point out that any kind of generative model should work. Another type of Generative Model is the Generative Adversarial Network (GAN) which currently is the state of the art when it comes to image generation. This work will also briefly investigate whether GANs can be used in GQNs instead of VAEs. For the sake of clarity, the GQN will be called Variational Autoencoder Query Network (VAEQN) when a VAE is used as the Generative Model and Generative Adversarial Query Network (GAQN) if a GAN is used.

1.3 Delimitations

This thesis does not aim to implement a fully functioning system where a user can input a sketch and retrieve a 3D rendering of it. Instead, the objective is to investigate whether the GQN framework is suitable for such a task and what its limitations might be. Constructing an end-to-end sketch-to-3D-model system would require the design of additional features such as a way to draw images from multiple view points, which would be both time consuming and not directly relevant to the field of machine learning.

2 Ethics & Societal Aspects

More and more parts of society are automatically controlled by computer systems and extensive care must be taken to ensure that these system are robust and truly perform their task in the way they are supposed to. 3D models are used in everything between computer games to modeling the heart of a patient scanned in a MRI, and the requirements of the models' fidelity varies accordingly. This thesis does not aim to produce final, ready to be used 3D models but rather to create a tool which would allow people working with these to quickly explore ideas by turning their sketches into seemingly 3D models. It is important to convey to other readers that this tool is not suitable for tasks where high fidelity is necessary. Doing this is a way to ensure that as few people as possible misuse the system for things it was not designed for.

For artists and animators working daily with 3D modeling, the system developed in this project could potentially alleviate the tedious process of prototyping 3D models. A common concern when it comes to automation and technological advancement overall is that it will make humans redundant. In this case, the aim of the technology is not to replace human workers but to substitute a time consuming process of their work with a simpler alternative. This would allow the user to spend their time on something more fruitful saving both money and resources. Additionally, considering that the system developed in this thesis does not produce any real 3D models, the worker would still be needed to produce the final 3D model.

Training Deep Learning models requires a lot of computational power which in turn requires a lot of electricity. Since Deep Learning is becoming more and more popular, the amount of computing power spent to train models increase day by day. As with any other part of society, practitioners within the field of machine learning must make sure that the field scales in a sustainable way. The past years have seen an increase in the use of cloud computing where the computation is done at large data centers rather than locally. Concentrating the computation at data centers allows for better optimization of resources since the data centers themselves can be placed in locations optimal for cooling and with access to clean energy. Future practitioners should therefore aim to utilize cloud computing as much as possible.

3 Related Work

While the automatic generation of 3D models has been investigated for many years, the use of deep learning to generate these models is a relatively new field. This chapter will cover different approaches to represent 3D data and how the generative models can be constructed in order to learn the different representations.

A recent generative model is the so called Generative Query Network (GQN), proposed by the Google DeepMind group [13]. In the paper [13], the GQN is capable of rendering images almost indistinguishable from the ground truths from arbitrary view points after only receiving a single observation. While some of the alternative approaches discussed in section 3.2 are also able to perform single image reconstruction even in the form of actual 3D models, the quality of the results are not as good compared to those of the GQN. For the sake of quickly prototyping ideas, it is not necessary to explicitly obtain the final model as a 3D model as long as you can observe the results. With this in mind, the pros of being able to generate high quality results with the GQN were deemed to outweigh the cons of not obtaining the results in 3D format. The model's inner workings and the Variational Autoencoder it uses as generative model are thoroughly explained in section 3.5.

The GQN architecture allows other kinds of generative models to be used for its encoding and decoding networks. In the paper they make use of a convolutional recurrent latent variable model introduced by the DeepMind team called Convolutional DRAW [12], further explained in section 3.3.1, for the decoding network and a convolutional network for the encoding network. This work will also investigate whether a GAN can be used as the GQN's generative model, different GAN versions will therefore be examined in this section as well.

3.1 Generative Models

Generative models attempts to learn the probability distribution of data x, $P(x)$. Learning this distribution makes it possible to sample new data which resembles the original. There are many different approaches to learning this distribution, such as the more traditional Hidden Markov Models [40] and Gaussian Mixture Models [41] among others. With the rise of deep learning, several new generative models have seen the light of day and among the most powerful are Variational Auto Encoders [9] and Generative Adversarial Networks [3] which will be discussed further in section 3.3 and 3.4.

3.2 3D representations

Training neural networks to generate 3D models has been an active area of research in the past years. Since there are various ways to represent 3D models, such as the common formats point clouds, voxels and polygon meshes, researchers have tried many different approaches.

H. Kato et al. [10] discuss the problem of reconstructing 3D models in the form of polygon meshes since the act of rasterization is discrete and prevents back-propagation. They tackle the problem by designing an approximate gradient for the network obtained by comparing silhouettes of their generated models with silhouettes of the ground truth models, they then proceed to create a network which can perform single-image 3D reconstruction. J. Delanoy et al. [22] on the other hand trains a convolutional network to reconstruct 3D models from simple drawings by predicting occupancy of voxels in a voxel grid based on an initial 2D sketch. Another approach is to learn a final 3D representation in the form of point clouds as is done by Z. Lun et al. [23] where a 3D point cloud model is created from an input of one or several sketches.

Recent work [24] makes use of a Signed Distance Function (SDF) parametrized by a neural network. The SDF takes as input a spatial point and outputs the point's distance to the nearest surface. The sign of the distance indicates whether the point is inside or outside the surface. The surface of the model is then given by all the points where the SDF equals zero and this implicit surface can e.g. be rendered through raycasting.

Another approach that differs from the previously mentioned ones is the so called multi-view representation which is the one performed by DeepMind's GQN [13]. The GQNs rendering network outputs none of the common 3D formats but instead directly renders a 2D image representing a 3D scene from a learnt representation of the environment.

Generating 3D models with generative models is hard and whatever representation you chose there are pros and cons. The models produced by the GQN are in fact not 3D models in the traditional sense, but for the task of quickly prototyping models by sketching that is in fact not a problem. The quality of the rendered models from the GQN paper is so good that the models are often indistinguishable from their ground truths, comparing this with e.g. the often chunky voxel models it was deemed a good fit.

3.3 Variational Autoencoders - VAE

Variational Autoencoders (VAE) were first introduced by D.P. Kinga and M. Welling [9] and are based on variational Bayesian learning where intractable posterior distributions are approximated in different ways. The model is called a Variational Autoencoder when a neural network is used for predicting the posterior distribution.

VAEs assumes that the data of interest, x, depends on a latent variable, z, which describes features of x. If our data consists of written digits, as in the MNIST dataset, then these features might for example describe the number and the stroke width of a specific sample. Given a vector z, it should thus be possible to infer a conditional distribution over what data can be generated by this latent vector. Defining this distribution as $P(x|z)$, the probability of x is given by:

$$P(x) = \int P(x|z)P(z)dz. \tag{1}$$

Here an assumption must be made regarding the type of distribution for the prior, $P(z)$, and the likelihood, $P(x|z)$. Gaussian distributions are often used for the sake of simplicity, but any kind of distribution can be used. To generate new data it is necessary to infer good values of latent variables from the observed data, this is done by the posterior distribution. Using Bayes' theorem, the posterior distribution over the latent variables can be written:

$$P(z|x) = \frac{P(x|z)P(z)}{P(x)}. \tag{2}$$

In most practical cases, the dimensionality of the latent vector z is high making the integral in equation 1 intractable since it needs to be evaluated over all configurations of latent variables. Instead, it is possible to use a variational approximation of the true posterior, $q_\phi(z|x)$, parametrised by ϕ. The Kullback-Leibler divergence can be used to measure how well the variational posterior approximates the true posterior. It is defined as:

$$\mathbb{KL}(q_\phi(z|x)||p(z|x)) = \boldsymbol{E}_{\boldsymbol{z\sim q}}\big[\log(q_\phi(z|x))\big] - \boldsymbol{E}_{\boldsymbol{z\sim q}}\big[\log P(x,z)\big] + \log P(x) \tag{3}$$

where $\boldsymbol{E}_{\boldsymbol{z\sim q}}$ denotes the expected value with respect to q. It measures how much information is lost when approximating the true posterior. To evaluate this it is necessary to bypass the intractable $\log P(x)$ term which can be done by defining a new function, the so called Evidence Lower Bound ($ELBO$):

$$ELBO(\phi) = \boldsymbol{E}_{\boldsymbol{z\sim q}}\big[\log P(x,z)\big] - \boldsymbol{E}_{\boldsymbol{z\sim q}}\big[\log(q_\phi(z|x))\big], \tag{4}$$

where ϕ contains the variational parameters of the distribution, e.g. the mean and the variance if a Gaussian distribution is used. Equation 3 can now be written as :

$$\log P(x) = ELBO(\phi) + \mathbb{KL}(q_\phi(z|x)||p(z|x)). \tag{5}$$

$P(x)$ should be as large as possible and since the Kullback-Leibler divergence should be as small as possible while always being greater than or equal to 0, the $ELBO$ also needs to be as large as possible. Thus, maximising the likelihood, $P(x)$, can be done by maximising the $ELBO$ which does not require evaluating the intractable $P(x)$. Finally, equation 4 can be rewritten as:

$$ELBO(\phi) = \boldsymbol{E}_{\boldsymbol{z}\sim\boldsymbol{q}}\big[\log P(x|z)\big] - \mathbb{KL}(q_\phi(z|x)||p(z)). \tag{6}$$

The variational posterior term, $q_\phi(z|x)$ can now be parametrised by a neural network which takes the data x as input, encodes it and outputs the parameters to the posterior distribution. The likelihood term, $\log P(x|z)$, is then also parametrised by a neural network which takes a latent variable as input and outputs parameters to the data generating distribution.

Given that neural networks are trained by minimising a loss function, the loss function in VAEs is set to $-ELBO$. The network parameters can then be optimized by minimising the loss function using stochastic gradient descent.

3.3.1 Convolutional DRAW

Convolutional DRAW is a recurrent variational autoencoder first introduced by K. Gregor et al. [12]. The model uses recurrency to iteratively update the generated image as well as the prior and the posterior distribution in an auto-regressive fashion. If the recurrent model is rolled out $l = \{1, .., L\}$ steps the prior distribution can be written as:

$$q_\theta(z) = \prod_{l=1}^{L} q_{\theta_l}(z_l|z_{<l}) \tag{7}$$

where z_l is the latent vector and θ_l refers to the subset of parameters θ used by the conditional density at step l.

The explanation below closely follows the one in the DRAW paper [12]. The model contains the variables: input data x, the reconstructed data r, the reconstruction error ϵ defined as the difference between the input data and the reconstructed data, the hidden state of the encoder recurrent network h^e,

the hidden state of the decoder recurrent network h^d and the latent variable z. The hidden states and the reconstructed data are recurrent, i.e. they are iteratively updated over several time steps in the algorithm. For each time step $t \in \{1, T\}$, the model computes the following:

$$\begin{aligned} \epsilon &= x - r \\ h^e &= LSTM(x, \epsilon, h^e, h^d) \\ z \sim q &= q(z|h^e) \\ p &= p(z|h^d) \\ h^d &= LSTM(z, h^d, r) \\ r &= r + Wh^d \\ \mathcal{L}_t^z &= KL(q|p) \end{aligned} \tag{8}$$

where W denotes a convolution and LSTM is a LSTM recurrent neural network [42] with convolutional operators. $\mathcal{L}_t^z$ is the loss term contribution from the Kullback-Leibler divergence. The approximate posterior, q, and the prior, p, are both Gaussian where the means and log variances are linear functions of h^e and h^d respectively. Figure 1 demonstrates the gradual drawing of the image, i.e. the iterative updates of r for each $t \in \{1, 8\}$.

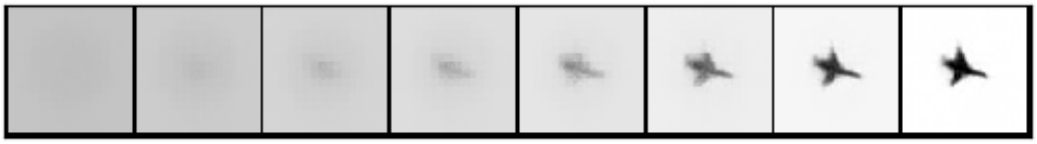

Figure 1: The generated image of an airplane in each time step of the Convolutional DRAW model

The parameters of the likelihood distribution are obtained from the final reconstruction, $r_{final} = r_T$. If a Gaussian distribution is used, the mean and the variance is obtained by splitting r_T giving us $p^x = \mathcal{N}(r_T^\mu, exp(r_T^\alpha))$. The likelihood loss term $\mathcal{L}^x$ can then be computed in the following way to obtain the total loss $\mathcal{L}$:

$$\begin{aligned} q^x &= U(x - s/2, x + s/2) \\ p^x &= \mathcal{N}(r_T^\mu, exp(r_T^\alpha)) \\ \mathcal{L}^x &= log(q^x/p^x) \\ \mathcal{L} &= \beta\mathcal{L}^x + \textstyle\sum_{t=1}^{T} \mathcal{L}_t^z \end{aligned} \tag{9}$$

where q^x is uniform noise with width s equal to the spacing between the image's discrete values which is added to the input in order to more easily

calculate the likelihood loss, further explained in the original DRAW paper [12].

3.4 Generative Adversarial Networks - GAN

An immensely popular generative model is the Generative Adversarial Network (GAN) invented by I. J. Goodfellow et al. [3]. GANs consists of two networks, a generative one which captures the data distribution and generates samples from it and a discriminative one which tries to estimate the probability that a given sample was drawn from the training data or generated by the other network. The generator tries to generate samples as similar to those from the dataset as possible in order to make it more difficult for the discriminator to determine whether a sample is from the real distribution or if it has been generated.

The training process can be seen as a two-player minmax game where the model has converged when the generator and discriminator reach a Nash equilibrium, the optimal point for the minmax game. GANs have proven to be a powerful model capable of producing impressive results but they are also notoriously difficult to train.

3.4.1 WGAN & WGAN-GP

The difficulty of training GANs as well as their powerful generative properties have made them a popular area of research in the past years and there exists many alterations to the original design. One of these is the Wasserstein GAN [4] (WGAN) which tries to stabilize training by introducing a new loss function based on the Earth-Mover distance which can be seen as a measurement of how much "mass" needs to be transported in order to transform the generated distribution into the real data distribution. In the paper they make use of the Kantorovich-Rubinstein duality to simplify the calculation of the earth mover distance, but in order to do this the functions learnt by the neural networks needs to be K-Lipschitz, a kind of regulation on how fast the function can change. In the original WGAN they enforce this constraint by performing weight clipping after each parameter update which, as they state themselves, is a terrible way to do it since it hampers the networks learning. Due to this a new, improved WGAN version was invented called the Wasserstein GAN with gradient penalty [5] (WGAN-GP) where the Lipschitz constraint instead is enforced by regularizing the norm of the gradient to make sure that it tries to stay around 1. WGAN and WGAN-GP makes training more stable and its loss function allows the generator to learn

even if the discriminator network becomes more powerful.

3.4.2 Conditional GAN

There are also GANs which allows conditioning on the generator to make it produce samples from e.g. a specific class or of a specific color. M. Mirza and S. Osindero presented the Conditional GAN [6] which allows conditioning on class by sending in a one-hot encoded class dependent vector to both the generator and discriminator networks and then combines it in a hidden layer with the latent vector for the generator or the input data for the discriminator. Conditioning can also be done on more complex features such as on sketched boundaries and sparse color strokes as is done in the work by P. Sangkloy et al. [18] or on low resolution images to produce high resolution versions of the same images as done by C. Ledig et al. [7].

3.4.3 SAGAN

While convolutional networks are the de facto architecture to use when it comes to image generation, it has been noted that they struggle to learn about long-term dependencies within images; such as separated feet for a dog as opposed to short-term dependencies such as the pattern of the dog's fur. Self-Attention Generative Adversarial Networks (SAGAN) is a GAN proposed by H. Zhang et al. [33] trying to address this issue by introducing a novel self-attention mechanism which allows the networks to efficiently model relationships between different regions. This is done in a layer of the network by transforming the feature maps from the previous layer into two new feature spaces which are used to calculate how much attention is given to a specific location in the image, this attention layer is then combined with another feature space transformed from the same feature map and finally scaled and added to the input feature map.

SAGAN also makes use of Spectral normalization first proposed by T. Miyato et al. [32] which is a way to normalize the networks weights so that the function learnt becomes K-Lipschitz continuous believed to stabilize learning as is shown in the WGAN and WGAN-GP. Notably, in SAGAN they use Spectral Normalization in both the generator- and discriminator network while T. Miyato et al. use it only for the discriminator, they argue based on empirical evidence that the generator also gains benefits from it. The conditioning of the generator is done through conditional batch normalization and in the discriminator through projection. Finally, they use a different learning rate for the generator- and discriminator network, something they show speeds

up learning.

3.4.4 Recent Improvements of GANs

A recent publication by A. Brock et al. [31] investigates the importance of different features in the most popular GAN architectures at the moment in order to find out which ones really makes a difference. Their main findings is that GANs benefit dramatically from scaling, e.g. by enlarging the batch size or increasing the number of hidden units. They introduce something called the truncation trick which they find to be highly beneficial when scaling GANs. It is a way to control the sampling of latent variables used by the generator network during inference by only allowing samples between a specific range to be allowed, values outside this range will be re-sampled. Doing this makes it possible to control the variety and fidelity of the generated samples since more extreme, and thus less likely, samples can be allowed or discarded depending on preference.

3.5 Generative Query Networks - GQN

Generative Query Networks were first introduced by S. M. A. Eslami et al. [13] and presents some new interesting ideas. The first being its representation network which learns an internal representation of a scene. It does this by observing a few images of the scene from different view points together with the camera position from each perspective and then creates a representation from which the rendering network can generate a new image from a previously unseen perspective.

A view point, v, is defined as $v = [x, y, z, cos(yaw), sin(yaw), cos(pitch), sin(pitch)]$ where x, y, z are the Cartesian coordinates for the camera from which the image was captured and the yaw and pitch are the camera's corresponding yaw and pitch at that point, defined as shown in figure 2.

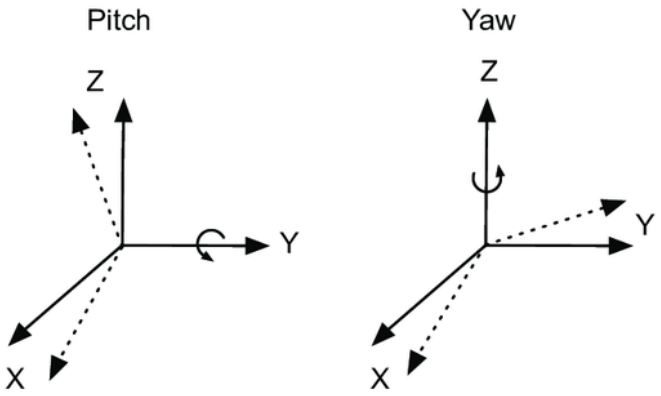

Figure 2: Camera's yaw and pitch angles[1]

The representation is produced by running each of the images from different perspectives as well as the camera positions through the representation network and then combining the results through element-wise summing into a final so called neural scene representation.

For a scene containing $[1, ..., M]$ images $x_{img}^{1,...,M}$ and $v^{1,...,M}$ viewpoints, the representation is calculated as follows:

$$v^k = (x^k, y^k, z^k, cos(yaw^k), sin(yaw^k), cos(pitch^k), sin(pitch^k)) \tag{10}$$

$$r^k = \psi(x_{img}^k, v^k) \tag{11}$$

$$r = \sum_{k=1}^{M} r^k, \tag{12}$$

where $\psi(x^k, v^k)$ is a convolutional network. The paper discusses three different choices of this network and what difference the network architecture makes for the learnt representation, the architectures can be seen in figure 3.

[1] `https://www.researchgate.net/figure/Roll-pitch-and-yaw-angles_fig6_319480685`

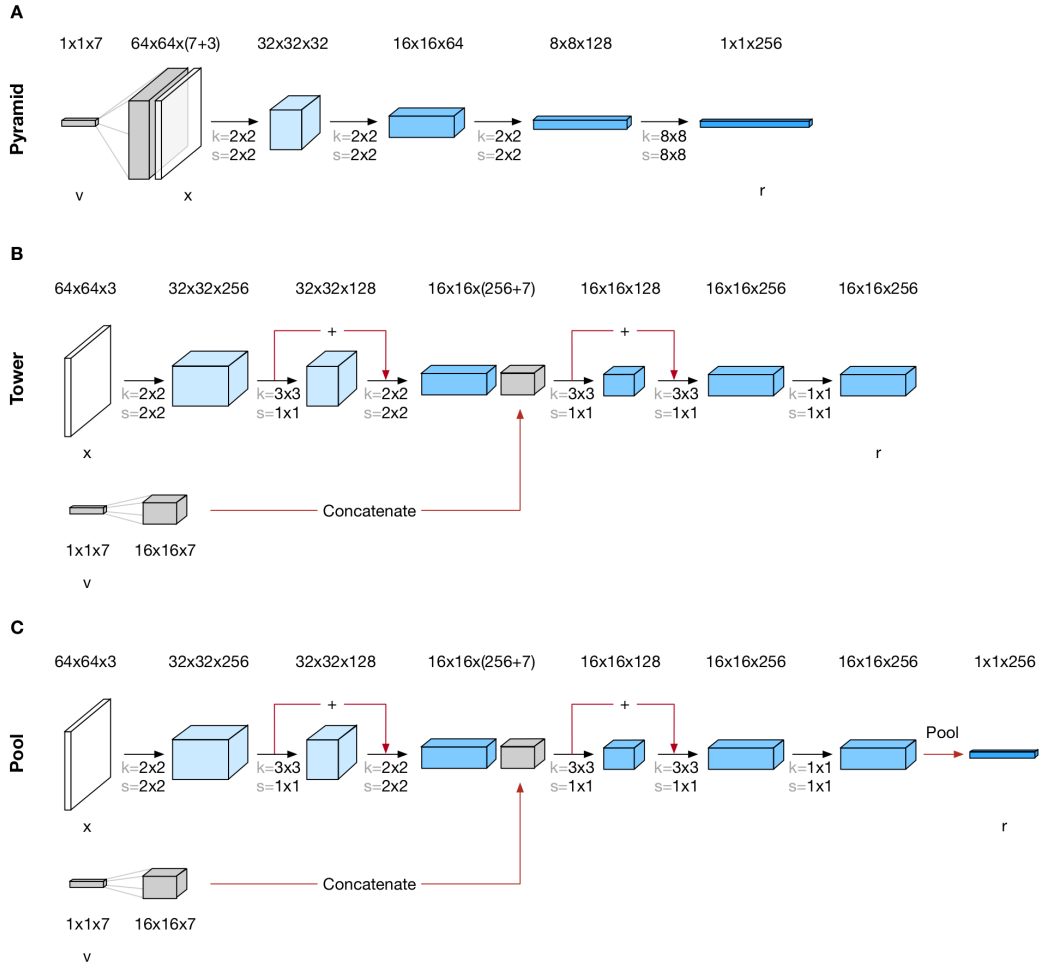

Figure 3: The pyramid, pool and tower representation networks as presented in the supplementary materials [14] to the original paper [13].

The pyramid and the pool architecture are shown to learn a factorised representation of the scene while the tower architecture is shown to learn a joint representation of object properties, meaning e.g. that changing the shape of an object in a scene also changes it's color. The networks architectures are shown in the figure where black arrows indicate convolutional layers with ReLU activations and with kernel size and stride determined by k and s. The convolutions with stride 1×1 preserves the size while the others are 'valid', meaning that if the filter with some multiple of the stride does not fit within the feature map, the remaining values after that stride will be discarded. Residual connections are shown as red arrows with a '+'. The viewpoints are concatenated by broadcasting in the spatial dimension to the right size.

3.5.1 Generation Architecture

In the original paper a conditional version of the Convolutional DRAW [12] is used as the Generative Model for generating new images from the queried view points. Since the Convolutional DRAW is a VAE, training the model and sampling new data is done in two separate steps. During training, a sample is generated from the approximate variational posterior using the query image, query view and the representation. The sample is then used to estimate the ELBO in order to optimize the network.

The procedure to estimate the ELBO is shown in algorithm 1 following the code provided in the GQN supplementary materials [14].

Algorithm 1 Estimate the ELBO

procedure ESTIMATEELBO($(x^k, v^k), (v^q, x^q), \sigma_t$)
  // Scene encoder - Neural representation
  $r \leftarrow 0$
  for $(k = 0, k < M, k++)$ **do**
    $v^k = (x^k, y^k, z^k, cos(yaw^k), sin(yaw^k), cos(pitch^k), sin(pitch^k))$
    $r^k = \psi(x^k_{img}, v^k)$
    $r \leftarrow r + r^k$
  end for
  // Generator initial state
  $(c^g_0, h^g_0, u_0) \leftarrow (0, 0, 0)$
  // Inference initial state
  $(c^e_0, h^e_0) \leftarrow (0, 0)$
  $ELBO \leftarrow 0$
  for $(l = 0, l < L, l++)$ **do**
    // Prior factor
    $\pi_{\theta_l}(\cdot|v^q, r, z_{<l}) \leftarrow \mathcal{N}(\cdot|\eta^\pi_\theta(h^g_l))$
    // Inference state update
    $(c^e_{l+1}, h^e_{l+1}) \leftarrow C^e_\phi(x^q, v^q, r, c^e_l, h^e_l, h^g_l, u_l)$
    // Posterior factor
    $q_{\phi_l}(\cdot|x^q v^q, r, z_{<l}) \leftarrow \mathcal{N}(\cdot|\eta^e_\theta(h^e_l))$
    // Posterior Sample
    $z_l \sim q_{\phi_l}(\cdot|x^q v^q, r, z_{<l})$
    // Generator state update
    $(c^g_{l+1}, h^g_{l+1}, u_{l+1}) \leftarrow C^g_\theta(v^q, r, c^g_l, h^g_l, u_l)$
    // ELBO KL contribution update
    $ELBO \leftarrow ELBO - KL[q_{\phi_l}(\cdot|x^q, v^q, r, z_{<l}))||\pi_{\theta_l}(\cdot|v^q, r, z_{<l})]$
  end for
  // ELBO likelihood contribution update
  $ELBO \leftarrow ELBO + \log \mathcal{N}(x^q|\mu = \eta^g_\theta(u_L), \sigma = \sigma_t)$
end procedure

Here C^e_ϕ and C^g_θ are the computational cores of the convolutional LSTMs and c^g_0, h^g_0, c^e_0 and h^e_0 are the output and cell state variables of the LSTM while u_0 is the state update, i.e. the image. The preceding variables and components can be seen in figure 4. The prior and posterior factor, $\eta^\pi_\theta(h^g_l)$ and $\eta^e_\theta(h^e_l)$ respectively are auxiliary convolutional networks which maps their inputs to the means and standard deviations of a Gaussian density.

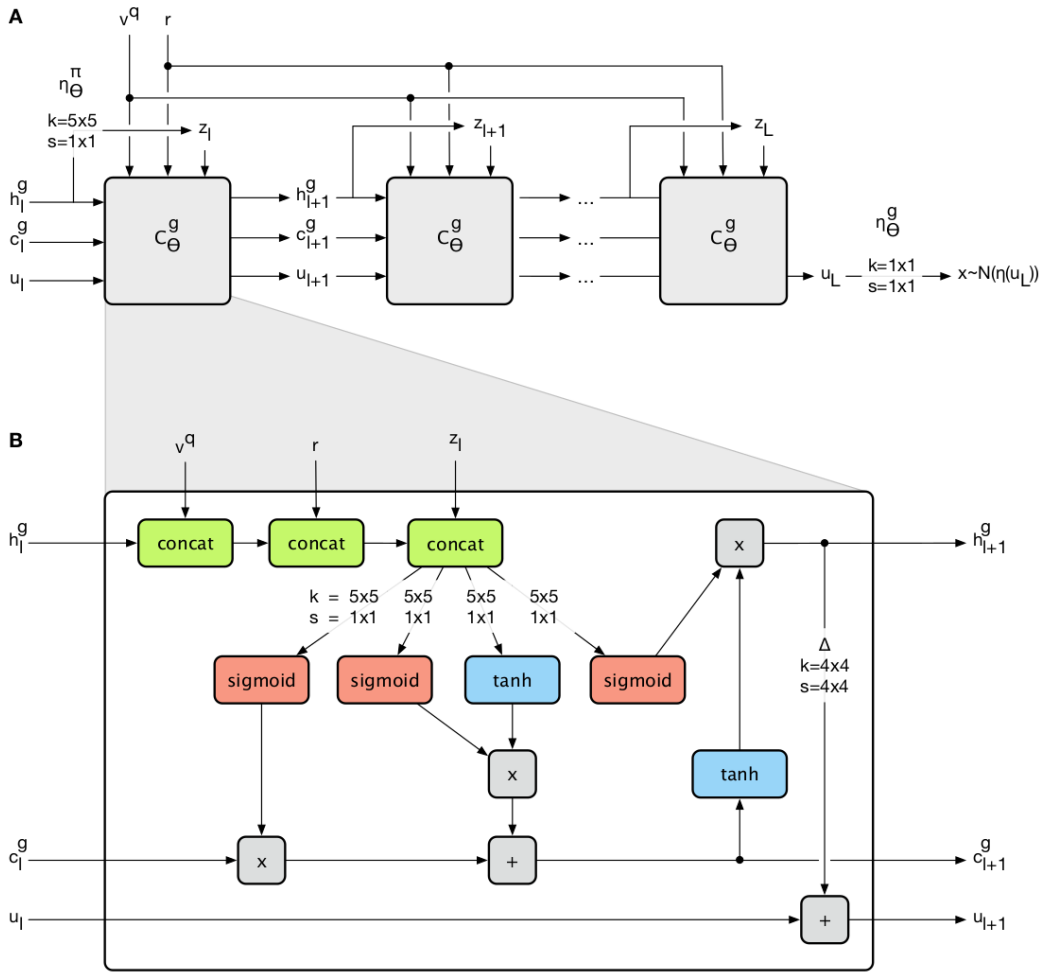

Figure 4: Convolutional DRAW computational cores from the GQN supplementary materials [14]. k and s are the convolutional kernel and stride size. (***A***) At each iteration l, the parameters are updated as shown in algorithm 1. (***B***) Each core is a skip-convolutional LSTM network [14].

When generating a prediction from the network, a latent sample is instead drawn from the prior distribution which is then used together with the query view and representation to infer the statistics to the data generating distribution from which the queried image can be sampled. The procedure is shown in algorithm 2 which also follows the code provided in the GQN supplementary materials [14].

Algorithm 2 Generate a prediction from GQN

procedure GENERATE($(x^k, v^k), (v^q)$)
  / Scene encoder - Neural representation
  $r \leftarrow 0$
  for $(k = 0, k < M, k++)$ **do**
    $v^k = (x^k, y^k, z^k, cos(yaw^k), sin(yaw^k), cos(pitch^k), sin(pitch^k))$
    $r^k = \psi(x^k_{img}, v^k)$
    $r \leftarrow r + r^k$
  end for
  / Initial state
  $(c^g_0, h^g_0, u_0) \leftarrow (0, 0, 0)$
  /Inference initial state
  $(c^e_0, h^e_0) \leftarrow (0, 0)$
  for $(l = 0, l < L, l++)$ **do**
    Prior factor
    $\pi_{\theta_l}(\cdot|v^q, r, z_{<l}) \leftarrow \mathcal{N}(\cdot|\eta^\pi_\theta(h^g_l))$
    / Prior Sample
    $z_l \sim \pi_{\theta_l}(\cdot|v^q, r, z_{<l})$
    / State update
    $(c^g_{l+1}, h^g_{l+1}, u_{l+1}) \leftarrow C^g_\theta(v^q, r, c^g_l, h^g_l, u_l, z_l)$
  end for
  / Image Sample
  $\hat{x} \sim \mathcal{N}(x^q|\mu = \eta^g_\theta(u_L), \sigma = \sigma_t)$
end procedure

3.5.2 Consistent Generative Query Networks

The GQN samples a new latent space vector for each new image it generates which can lead to temporal incoherence if several frames are to be sampled to form a sequence for a video. The DeepMind group tackles this problem in their follow-up paper [2] where they introduce the Consistent Generative Query Network which learns to generate a global latent representation out of an arbitrary set of frames within a video. Instead of creating a context, i.e. a scene representation, from a few observations, they now create the context from pairs of individual frames and the time stamps at which they occur. The query is changed to also contain the timestamps of the frames that are to be sampled rather than just the camera positions. Given the context the network now samples a stochastic latent scene representation that models the global stochasticity in the scene. When a query for a specific times stamp is sent in, the network uses this scene representation to render a corresponding

frame.

3.6 Dataset

3.6.1 Suitable datasets

The aim of this master thesis is to train a generative model to render 2D images representing 3D scenes from a set of given 2D observations. No dataset containing images of the same object from different perspectives and with their respective camera position recorded could be found besides the Shepard-Metzler dataset which is not diverse enough for this project. It was thus necessary to create a custom dataset to solve the task at hand. A simple way to create 2D images of a 3D model is to take screenshots of the 3D model from different perspectives and then save the spatial position from where the screenshot was taken.

There are several 3D datasets available online, such as ShapeNet [11] and ModelNet [21]. Shapenet is made up of around 51,300 unique 3D models from 55 different categories while ModelNet contains 127,915 CAD models from 662 categories.

3.6.2 Data Preparation

The end goal is to render 3D models out of simple sketches, the dataset thus needs to contain sketches. There are several approaches to converting images to sketches, such as the method used by Y. Gucluturk et al. [19] where line sketches where produced from natural images of faces by first converting the image to greyscale and then inverting the image to obtain a negative image. A Gaussian blur is then applied followed by a color dodge to blend the blurred image with the original image. Another approach with much better results is the one done by E. Simo-Serra et al. [20] which aims to clean up a hand drawn sketch by running it through a neural network. Although not an actual proposed feature of the model it was found that this network could produce sketches out of regular images and good results were obtained if the images where first turned into edge maps.

In order for a neural network to learn something useful it is necessary to ensure that the data fed to the network is suitable to learn from. This can be done by performing various kinds of pre-processing of the data, such as normalizing, discretizing or extracting helpful features. In Deep Learning the feature extraction task is often given to the network itself and is done

by e.g. convolutional layers which can extract and combine edges found in images. Normalizing the data ensures that is has a more similar distribution which makes learning algorithms converge faster. Normalizing is often done in three different ways [30]: *min-max normalization* which fits data into a range $[min, max]$, *Z score normalization* which uses the mean and standard deviation from the dataset and finally *Decimal scaling* which scales the data to fit into the range of [-1, 1] if negative values are present and [0, 1] otherwise.

Since the dataset is to be produced from an existing, carefully crafted public dataset, a lot of pre-processing has already been done. For example, all models in ShapeNet are positioned in a consistent canonical orientation and there are never more than a single object included in each model.

Further preparations of the dataset will be discussed in the method section.

3.6.3 Shepard-Metzler dataset

In order to evaluate the training of the GAQN, an attempt to reproduce DeepMind's results with the Convolutional DRAW [13] was first done on DeepMind's Shepard-Metzler dataset as seen in the GQN paper [13]. The Shepard-Metzler dataset was constructed in a procedural fashion and the dataset exists in two versions where both contains 2 million scenes of Shepard-Metzler models [43]. Each model is made up of either 5 or 7 randomly coloured cubes that are positioned at different locations by a self-avoiding random walk in a 3D grid. Each scene contains 15 images and the view points from where they where captured. More information can be found in the paper [13].

An example of a Shepard-Metzler model taken from the dataset with 7 cubes can be seen in figure 5. The version with 5 cubes is considered easier to learn than the one with 7 cubes, the 5 cube version was thus used in the Shepard-Metzler experiments in section 4.4.2, 5.8 and 5.9.

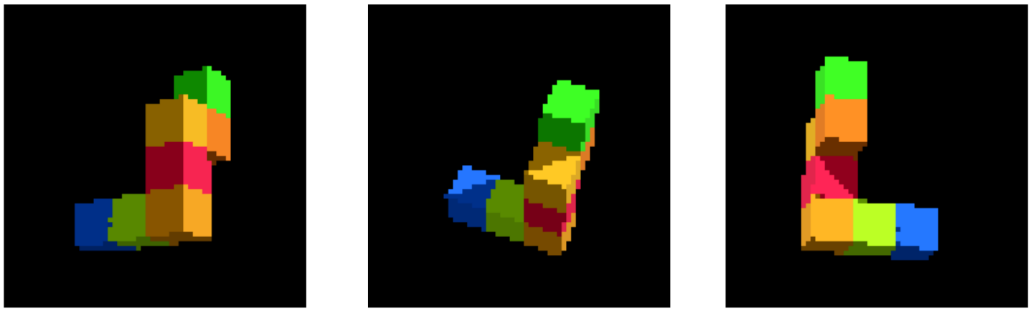

Figure 5: Example of a Shepard-Metzler model from the GQN supplementary materials [14]

3.7 Evaluation Metrics

While evaluating generated 3D models remains an unsolved problem as discussed in the paper by J. Wu et al. [25], the neural rendering network produces 2D images which can easily be compared with the ground truth image. The networks performance can thus be evaluated by constructing a test set and running the model on this unseen data. A quantitative evaluation of the model can therefore be done by for example measuring the mean absolute error between generated images and the ground truth as is done by P. Welander et al. [26] when comparing generated MR images with their corresponding ground truth, or by measuring the Mutual Information presented by J. P. W. Pluim et al. [38] and used by e.g. Q. Yang et al. [39] to evaluate so called NeuroImage-to-NeuroImage.

A common way to evaluate the quality of generated images by generative models is the Inception Score. The Inception Score was introduced in the paper by T. Salimans et al. [35] and measures the exponential of the Kullback-Leibler distance between the conditional label distribution $p(y|x)$ and the marginal distribution over classes $p(y)$ marginalized over the latent variables z. The conditional label distribution is obtained by applying the Inception model [36] to every generated image. Supposedly, if the model generates images with meaningful objects the conditional label distribution have low entropy while a high entropy of the marginal distribution indicate their diversity. As discussed in the Sagan paper [33], the inception score primarily indicates if the model generates samples that can be confidently recognized as belonging to a specific class as well as if the model generates samples from many classes. Measuring this is especially important when determining the performance of GANs which often suffers from mode collapse. For VAEs, mode collapse is less of a problem.

The Fréchet Inception distance (FID) is a more recent construct by M. Heusel et al. [37] in which both the real images and the activations from one of the layers of the Inception network are assumed to originate from a multidimensional Gaussian distribution. The difference between two Gaussians is measured by the Freć het distance, also known as Wasserstein-2 distance and the Fréchet Inception Distance is thus defined as the Fréchet distance between the two above mentioned Gaussian distributions.

Another popular measure is the so called perceptual loss which is calculated in a similar way to the FID but now instead the feature maps from a pre-trained VGG19 model [27] is used. The perceptual loss is supposed to be similar to human measurement of image quality in terms of what looks "real" and is not as sensitive to e.g. rotations and translations as the L2 norm loss is. It was introduced by C. Ledig et al. [7] and is defined as:

$$l_{VGG/i,j} = \frac{1}{W_{i,j}H_{i,j}} \sum_{x=1}^{W_{i,j}} \sum_{y=1}^{H_{i,j}} (\phi_{i,j}(x_{real})_{x.y} - \phi_{i,j}(G(z, r, v_q))^2 \qquad (13)$$

where $\phi_{i,j}$ indicates the j-th convolution after activation before the i-th max-pooling layer within the VGG19 network while $W_{i,j}$ and $H_{i,j}$ describes the dimensions of corresponding feature map. x_{real} is the ground truth image while $G(z, r, v_q)$ is the generated image for that view point and representation. As shown in J. Johnson et al. [28] and A. Mahendran et al. [29], comparing features in the different layers of the network gives information about different properties of the image, e.g. whether the image content and overall spatial structure is similar.

The perceptual loss was found to be easy to calculate and measured all the sought after features, it was thus chosen for measuring the quality of the outputs from the trained models.

4 Methods

The following section explains how to create the dataset as well as how to train and evaluate the generative models. To avoid excessively long names two abbreviations are hereby introduced, the first being VAEQN meaning a GQN where the generative model is the convolutional DRAW and the second being GAQN indicating a GQN where the generative model is a GAN.

4.1 Dataset

A new dataset had to be created to fit the needs of the project and two datasets under consideration were ModelNet[21] and ShapeNet[11]. While ModelNet contains more models, the models' quality is in general worse than ShapeNet's and the larger number of categories also means less examples per category which in the end led to ShapeNet being used. ShapeNet provides an API for browsing through its models but the task of loading the model and capturing images from it from different angles proved to be more easily done by using Blender [46].

The models are saved in *.obj* format which can be loaded and rendered directly in Blender. Each model was loaded one at a time into Blender, all centered at origo in 3D space. 15 triplets where then sampled from the surface of a sphere with radius 2 centered at the origin in the form (x, y, z) where x, y and z correspond to the values of the x-, y- and z-axis for that specific point. The view camera was then positioned at these points and made to look at the origin. The camera's yaw and pitch angles were then obtained directly from the Blender camera object. An image was then captured at each point and saved with size 512x512 pixels in jpg format. The triplet, the yaw and pitch of the camera and the model's class id was saved in a text document. The light in each scene was set at a fixed point which remained the same for all scenes.

The saved images where processed by first detecting edges using the Python Package PIL's edge finding algorithm [47]. This produced a black background with white edges. In order to make the images look more like real life sketches their colors where then inverted. Subsequently, to clean up and refine the edges the images were run through a pre-trained Sketch Cleanup Convolutional Neural Network made by E. Simo-Serra et al. [20] earlier discussed in the related work section. The intermediary steps and the result can be seen in figure 6 and 7 respectively.

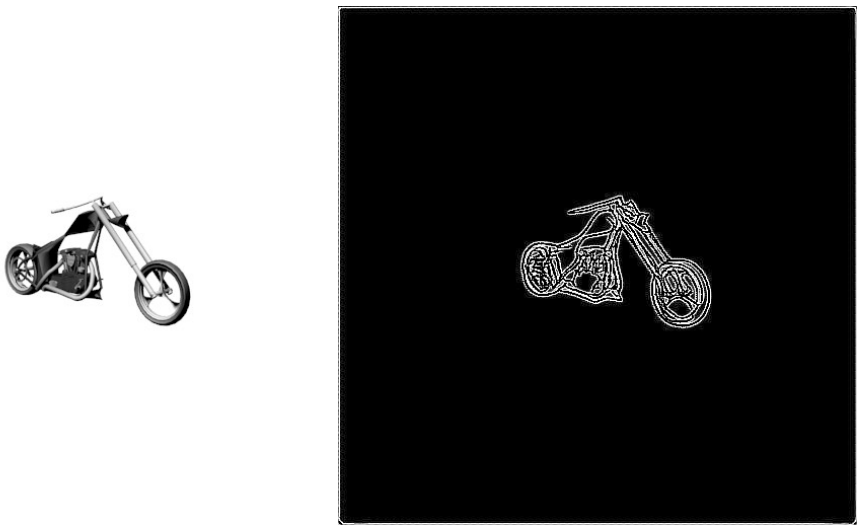

Figure 6: Left: Original Image. Right: Edge Detection

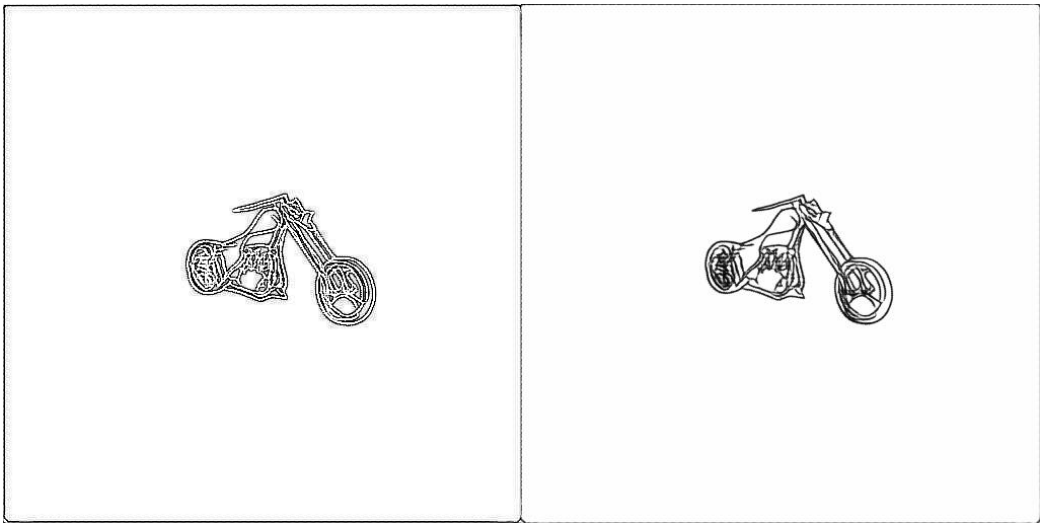

Figure 7: Left: Inverted Image. Right: Sketch produced by Neural Network

The images and their corresponding labels where then converted into PyTorch tensors [48] and the labels where transformed to the form shown in the GQN paper [13] with the addition of the class id:

$$[x, y, z, cos(yaw), sin(yaw), cos(pitch), sin(pitch), class\_id]. \quad (14)$$

The original, unprocessed images where then saved paired together with the processed images. Finally, the dataset was split into a training set and a test set where roughly 90% of the data went into the training set and 10% into the test set. More sketch samples can be found in appendix B.

4.2 Network Architecture

4.2.1 VAEQN

The same network structure was used for the Convolutional DRAW as in the original work by S. M. A. Eslami et al. [13], thoroughly explained in the related work section.

4.2.2 GAQN

Long-term dependencies within images are important when generating 3D models since the overall structure of the model needs to remain intact. A version of SAGAN was therefore used where the conditioning on scene representation and view points where done in the traditional fashion of concatenation rather than with projection and conditional batch normalization as is done in vanilla SAGAN. Those methods are used in order to improve the generated results but does not stabilize training more than e.g. using regular batch normalization does.

Two separate representation networks were used, one for the generator and the other for the discriminator. The motivation behind this was that the discriminator's task here is not only to determine whether an image is real or generated but also if it is the correct image for this particular scene from this particular view point. The generator also needs to obtain the same information about scene and view point, but if the same representation network is used for both networks it is unclear how to train it. The pool representation explained in the related work section was found to perform best and was thus used for both models.

The Adam optimizer [1] was used for training together with an exponentially decaying learning rate. The Generator network setup can be seen in figure 8:

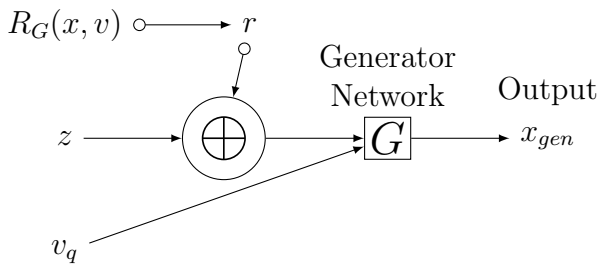

Figure 8: Generator setup

while the Discriminator network setup can be seen in figure 9:

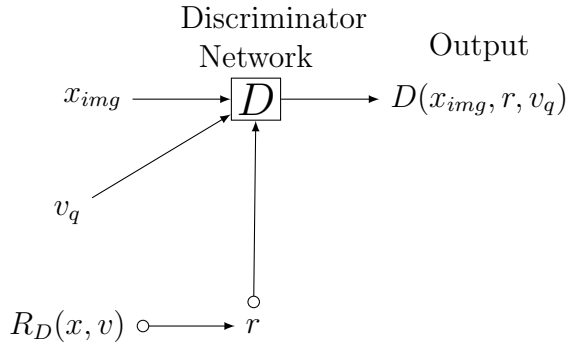

Figure 9: Discriminator setup

where x_{img} is either real data or generated and $\oplus$ indicates the concatenation of the incoming variables. v_q is the query view and R_G and R_D are the representation networks for the generator and discriminator.

The generator is based on the generator architecture used in DCGAN [34] with the minor changes needed in order to concatenate the conditional variables, r and v_q as well as the self-attention mechanism introduced in SAGAN. Batch normalization is used in all layers and spectral normalization is used in all convolutional layers as is done in the original SAGAN model [33]. ReLU is used as activation function for all intermediate layers. Several different architectures where tested but all performed similarly or worse compared to the baseline and only added to model complexity and training time. The investigated models are described in appendix A.2.

The discriminator network is also based on the one used in DCGAN but here also with the difference that spectral normalization is used in each layer as well as self-attention in the layer before the final layer. Like in the DCGAN, LeakyReLU is used as activation function. More details of the generator and discriminator architectures can be found in appendix A.1.

4.3 Hyperparameter Search

The Convolutional DRAW worked well with the minor parameter changes earlier described and due to the high computational cost no further attempts to tweak the values were done.

The SAGAN did however display typical issues for GANs during training

such as mode collapse, vanishing gradients, exploding gradients, the generator overpowering the discriminator and vice versa. An attempt to find better hyperparameter values was done by performing a grid search over the learning rates, γ, for the generator and discriminator while the learning rate for the representation networks was fixed to the same as in the Convolutional DRAW model. The other parameters tested were the batch size, m, and the number of discriminator updated per generator update, ψ. Evaluation of each parameter setting where done by comparing the perception loss calculated by a pre-trained VGG19 network between the ground truth image and the generated one after 10 000 gradient updates.

A coarse search was first done over the following values:

$$\begin{aligned} &\gamma : [5 \cdot 10^{-2}, 1 \cdot 10^{-3}, 5 \cdot 10^{-3}, 1 \cdot 10^{-4}, 5 \cdot 10^{-4}, 1 \cdot 10^{-6}, 1 \cdot 10^{-6}] \\ &m : [1, 2, 4, 8, 12] \\ &\psi : [1, 2, 3, 4, 5] \end{aligned} \tag{15}$$

A fine search was done as well but it also found the best values from the coarse search to be optimal. The parameters of the best performing model where found to be $\gamma = 0.001$ for both the generator and discriminator, 2 discriminator updates per generator update and a batch size of 4.

4.4 Training Procedure

The following section explains the training procedure for the VAEQN and GAQN in detail.

4.4.1 VAEQN

Updating the parameters of the network is done in the same way as in the original GQN, explained in algorithm 1 in the related work section, with some minor adjustments.

The training is done by first sampling a batch of scenes from the training data containing sketches of the scenes together with the ground truth images and the view points of each sketch and image. The sketches and the images are then scaled into the range $[0, 1]$. A random non-sketch image and it's view point is then selected from each scene to become the ground truth image and the query view point. An integer k between 1 to 15 is then sampled from a uniform distribution and k sketch images and their view points except for the one corresponding to the selected ground truth are then fed into the

representation network which creates a representation for each scene. The representation is thereafter sent into the Convolutional DRAW together with the ground truth image and the query view point in order for the network to estimate the ELBO which is then used to calculate the gradients to update the network parameters.

When generating data, an integer between 1 to 15 is again sampled and that number of sketches are fed into the representation network. This representation is then inserted into the network together with a query view point but this time without the ground truth image. The network then samples from it's prior and together with the representation and query view infers the mean to the data generating distribution from which a final image can be sampled.

To evaluate the training process, the ELBO and the separated KL loss are computed on data from the test set to make sure that the model is not overfitting on the training data. The plots can be seen in section 5.1

Training the model was computationally expensive making it difficult to search for optimal hyperparameter values. The same values as in the original work [13] were thus used to the extent possible. It was however found through informal search that training was more stable if the initial as well as the final learning rate was lowered by a magnitude of 10 from $5 \cdot 10^{-4}$ and $5 \cdot 10^{-5}$ to $5 \cdot 10^{-5}$ and $5 \cdot 10^{-6}$. Additionally, the original paper argues that using separate LSTM cores for each roll out step also improves the performance. It was found empirically that using separate cores made it difficult for the model to learn, possibly because the batch size had to be decreased from 36 to 8 due to gpu memory limitations, the final model was thus trained with shared cores.

The Adam optimizer [1] was used for training together with an annealing of the learning rate as explained in equation 16:

$$\mu_s = max(\mu_f + (\mu_i - \mu_f)(1 - \frac{s}{n}), \mu_f) \tag{16}$$

where s is the current training step, n was set to $1.6 \cdot 10^6$ and the initial learning rate was set to the above mentioned $5 \cdot 10^{-5}$ and the final value set to $5 \cdot 10^{-6}$. The indices f and i indicates the final respectively the initial values for these parameters. Finally, the pixel standard-deviation used to calculate the likelihood of the data in the ELBO was annealed from a starting value of 2.0 to an end value of 0.7 as shown in equation 17

$$\sigma_s = max(\sigma_f + (\sigma_i - \sigma_f)(1 - \frac{s}{n}), \sigma_f) \tag{17}$$

where s again is the current gradient step while n now is set to $2 \cdot 10^5$.

It was first believed that overcoming the problem with inconsistency in GQNs, investigated by the DeepMind group and presented in section 3.5.2, would become one of the challenges in the project. This did however turn out to be simple to fix by setting the variance in the prior distribution to zero during the generation of new data. This was only done when generating images after the training was done, not when calculating the perceptual loss or the Bayesian surprise.

4.4.2 GAQN

To the best of the author's knowledge, no one has succeeded in using another generative model besides the Convolutional DRAW in the GQN framework. At first, an attempt was therefore made to reproduce the Convolutional DRAW model's result on the Shepard-Metzler dataset as presented in the GQN paper [13]. Note that this means that no sketches are involved at all during training.

The modified SAGAN model presented in 4.2.2 was used and trained in the following way. A batch of scenes from the training data containing the images and the view points of each image is first sampled. A random image and it's view point is then selected from each scene to become the ground truth image and the query view point. An integer k between 1 to 15 is then sampled from a uniform distribution and k images and their view points except for the one corresponding to the selected ground truth are then fed into the representation network which creates a representation for each scene.

Once the representation is obtained, a latent vector z is sampled from a Gaussian distribution with zero mean and standard deviation equal to one. The latent vector is then sent into the generator network together with the representation and the query view point which the network uses to generate a new image.

The discriminator generates a representation in the same way as the generator but using its own representation network. The discriminator is then fed the representation and the query view point together with either a generated or a real image. The usual gradients received to train the discriminator

and the generator network are also used to train their respective representation networks.

When generating new images after the models were trained, the truncation trick explained in section 3.4.4 was used to exclude highly unlikely samples from being generated.

While the GAN is conditioned on both the representation and query view points, it is known from the conditional GAN paper [6] that class labels also helps a GAN to learn. The sketch dataset made from ShapeNet contains such labels and an attempt to train the GAN model on this dataset was also done. The only difference in the training set up was that the representation network was fed images in sketch form instead of real images and that the view points now contained the additional class id.

For the sake of comparison, the same GAN but without any representation or view points is trained on the Shepard-Metzler dataset. The results from this model could give an indication whether the GQN uses the representation and view points or not. If not, the results from the two models should be similar since they both generate data solely based on the latent vector.

4.5 Quantitative Evaluation

4.5.1 Perceptual Loss

The number of observed view point's effect on the perceptual loss described earlier will be investigated by feeding various amounts of images from different viewpoints and then calculating the perceptual loss between the real and the generated image. This is done as discussed in section 3.7 and seen in equation 13 by feeding sketches $x_1, ..., x_M$ together with view points $v_1, ..., v_M$ and query view point v_q to the representation network to generate x_g. All data is from the test set. Then x_g is fed into the VGG19 network and the feature maps is returned. The same thing is then done with the ground truth image and the difference between these feature maps is calculated using the L1 norm. The results can be seen in figure 14 and 16 in section 5.3.

4.5.2 Bayesian Surprise

During training, the Convolutional DRAW model incorporates the information it has learnt about the data into it's prior distribution. When doing inference with a trained model, information about that particular data is

encoded into the model's posterior distribution. It is possible to measure the model's surprise over a new observation by quantifying the distance between the prior and the posterior distribution, this can be done using the Kullback-Leibler divergence.

In the Convolutional DRAW the prior is defined as $\pi_\theta(z|y)$ and the variational posterior as $q_\phi(z|x,y)$ obtained through the prior factor, $\eta^\pi_\theta(h^g_l)$, and the posterior factor, $\eta_\phi(h^e_l)$, as explained in the related work section. The y contains the previously available information such as representation and view points. The Bayesian surprise, here denoted $BS(x,y)$, when providing a new observation x to the model with previous information y is then approximated by summing up the KL contributions at each roll out step from 1 to L:

$$BS(x,y) \approx KL\left[q_\phi(z|x,y)||\pi_\theta(z|y)\right] \tag{18}$$

$$\approx \sum_{l=1}^{L} KL\left[\mathcal{N}(z|\eta^q_\phi(h^e_l))||\mathcal{N}(z|\eta^\pi_\theta(h^g_l))\right]. \tag{19}$$

To evaluate how the model's surprise is affected by the number of observations, the $BS(x,y)$ is evaluated by feeding between 1 to 5 views to the representation network and then sending in an image from an unseen view point to produce the posterior distribution. The unseen image remains fixed for each number of views and the final $BS(x,y)$ is obtained as the average over 1000 samples which are also used to estimate the standard error bars.

The surprise is also evaluated as an average over 50 different classes with 1000 samples for each calculation of the $BS(x,y)$ to see that observing more view points generally decreases the Bayesian surprise. The data in all experiments is from the test set and the results can be seen in figure 17, 18 and 19 in section 5.4.

4.6 Qualitative Evaluation

4.6.1 Mental Rotation

To test the GQN's ability to generate images at arbitrary viewpoints given a scenes representation in sketch form, it is possible to rotate the camera around the object by using spherical coordinates and simply change the polar and the azimuthal angle. The polar angle, θ, can take values in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$ while the azimuthal angle, ϕ, can take values in the range $[0, 2\pi]$. Ten values for θ are generated starting at $-\frac{\pi}{2}$ and ending at $\frac{\pi}{2}$ with a step size of $\frac{\pi}{10}$ and 10 values for ϕ starting at 0 and ending at 2π with a step size of $\frac{2\pi}{10}$.

For each combination of these angles the rest of the view point coordinates are calculated as follows:

$$\begin{aligned} r &= 2 \\ x &= rsin(\theta)cos(\phi) \\ y &= rsin(\theta)sin(\phi) \\ z &= rcos(\theta) \\ cos(yaw) &= cos(\phi) \\ sin(yaw) &= sin(\phi) \\ cos(pitch) &= cos(\theta) \\ sin(pitch) &= sin(\theta) \end{aligned} \tag{20}$$

where r is the radial distance from the origo. The class id is extracted from the sketch view points sent in to the representation network. All data used is from the test set and the results can be seen in figure 20 and 21 in section 5.5.

4.7 Training Setup

Training of the VAEQN was done on a Nvidia GTX 1080 Ti for 770 000 gradient steps which took 16 days and 23 hours. The SAGAN model was trained on a Tesla K80 on a Google Cloud Virtual Machine for 65 000 gradient steps which took 8 hours and 30 minutes. The SAGAN was trained until collapse and the most recently saved state before that was used in the experiments.

5 Results

5.1 Training Progression for the VAEQN

Figure 10 and 11 display the gradual decrease of the VAEQN model's loss over $7.7 \cdot 10^5$ gradient steps. Initially, the optimizer focuses on minimizing the likelihood term of the ELBO, but as the annealing of the pixel-standard deviation, σ, reaches the final value, the KL divergence begins to decrease.

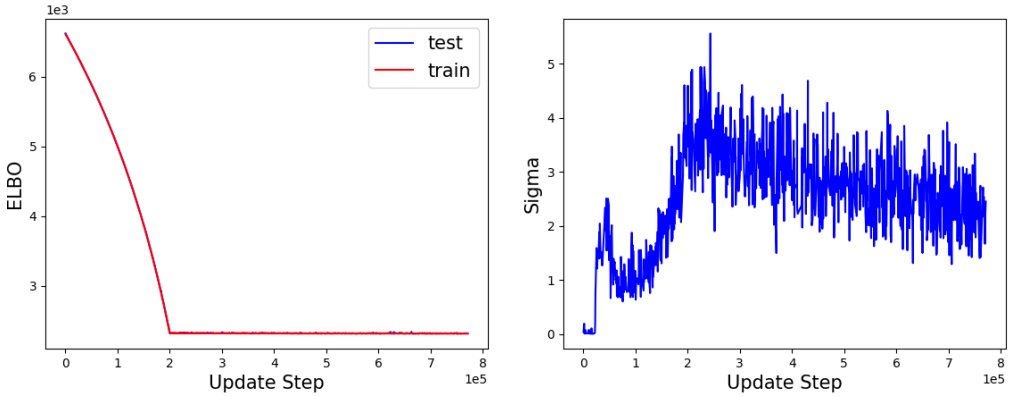

Figure 10: Left: ELBO loss. Right: KL term in ELBO

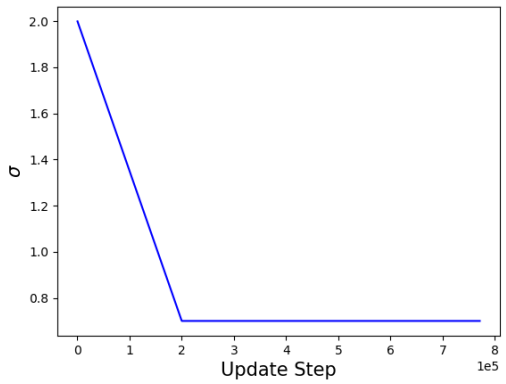

Figure 11: Left: Sigma Decay

The train and test losses match closely throughout training ruling out the possibility of the model having overfitted to the training data.

5.2 Generated Images and Ground Truths

The following images have been generated by the VAEQN after observing 14 sketches from different view points for random classes in the test data set.

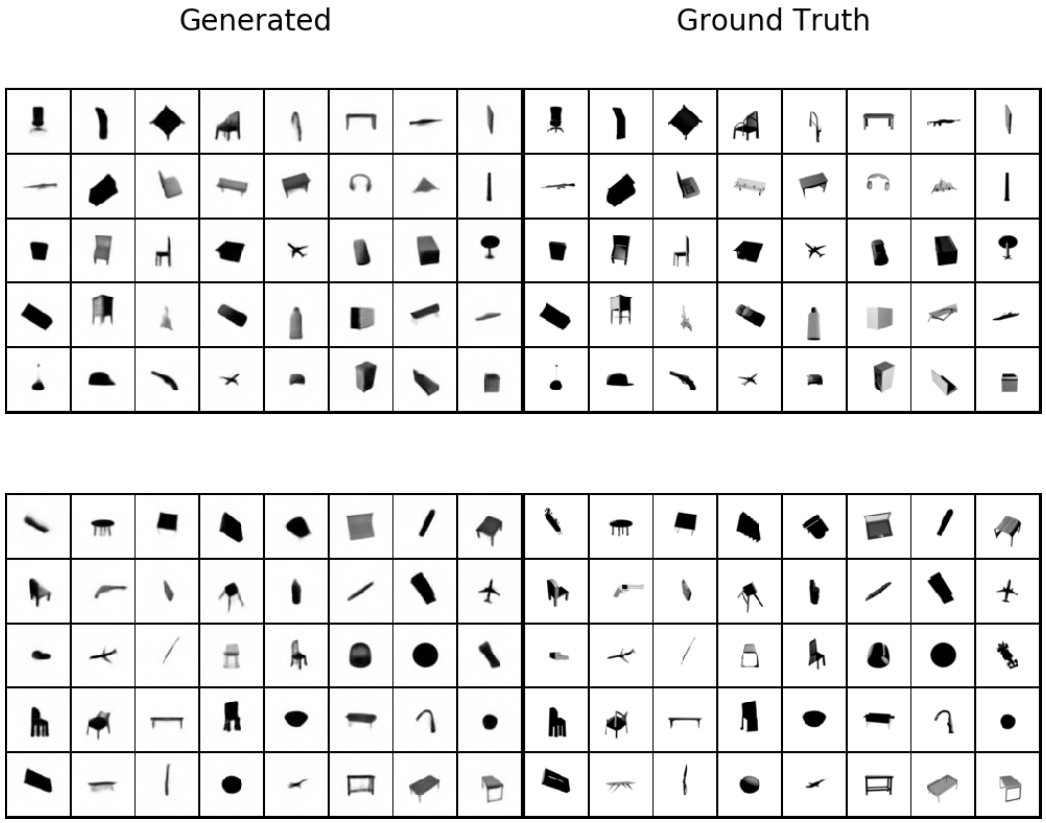

Figure 13: Left: generated images. Right: ground truths

The model captures the general structure of the object quite well while details on the models' surfaces are typically lost. It is better at generating some classes than others. More samples can be found in appendix C.

5.3 Perceptual Loss

Perceptual loss between a generated image and the ground truth image as a function of the number of observations can be seen in figure 14. Each value was calculated as the average between 1000 samples. The x-axis indicates how many sketches have been fed into the representation network this far. For example, for $x = 3$, the third sketch and the two proceeding ones have all been observed by the network. The error bars are smaller than the symbol

sizes for all measurements and the variance originates from the sampling of each image from the prior distribution.

5.3.1 Single Model Loss

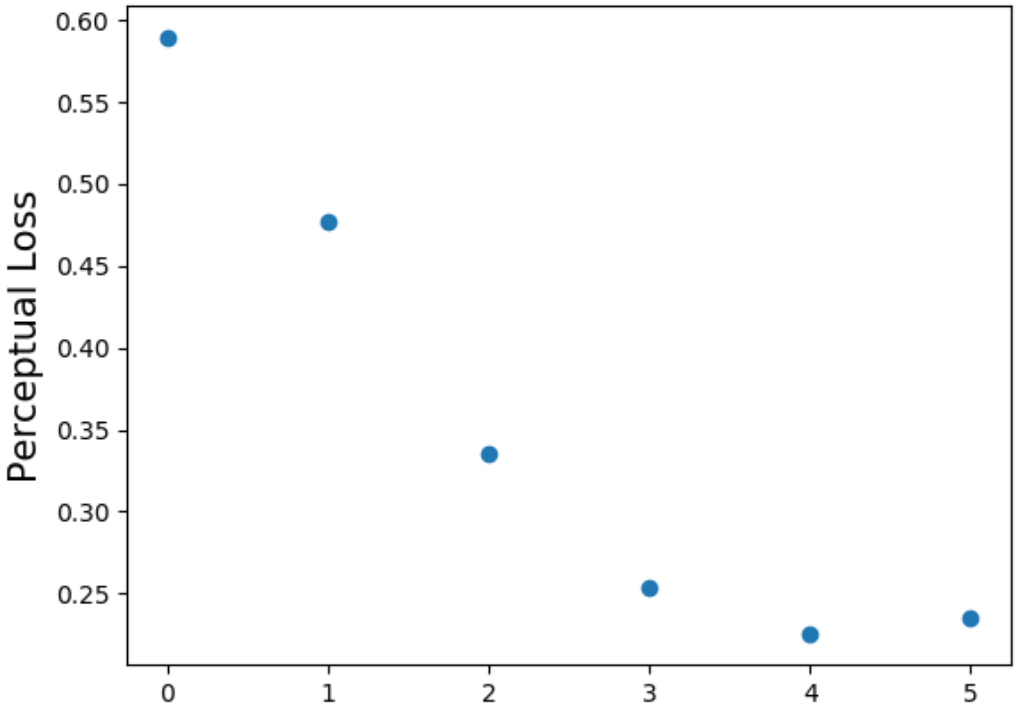

Figure 14: Average perceptual loss for a chair model after observing 0 to 5 sketches. The viewed sketches can be seen in figure 15.

In figure 15 the ground truth image used to calculate the perceptual loss can be seen together with the model's output after observing first zero to a final number of five sketches. The forth sketch is the only one which correctly captures the ribbed backrest of the chair, the fifth sketch then provides inaccurate information which leads to the perceptual loss increasing slightly.

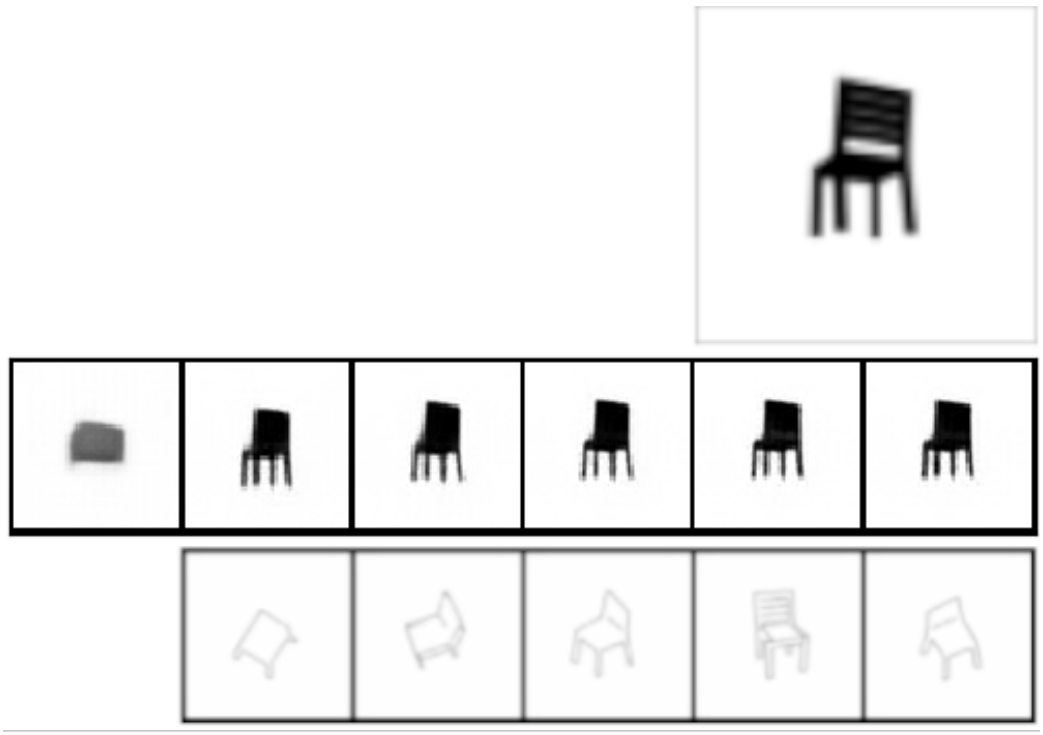

Figure 15: Top: ground truth image. Middle: images generated by observing an increasing number of sketches, from 0 to 5. Bottom: observed sketches

5.3.2 Averaged over classes

Average perceptual loss for 50 random models from the test data set. The error bars are calculated by sampling 1000 times per number of view point and model but they are smaller than the symbol sizes for most measurements. Figure 16 shows a general trend for more similarity between generated images and their corresponding ground truths as the number of viewed sketches increases.

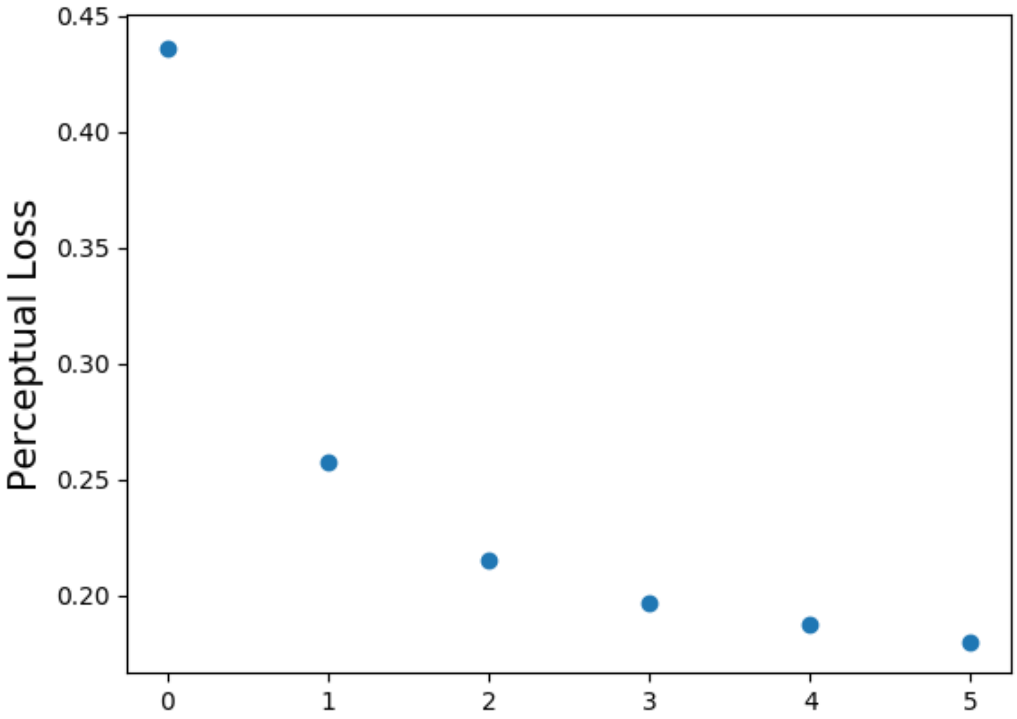

Figure 16: Average perceptual loss for 50 different models

5.4 Bayesian Surprise

The VAEQN model's Bayesian surprise after an increasing number of viewed sketches. Each value was calculated as the average between 1000 samples. The x-axis displays what sketches have been fed into the representation network this far. For example, for $x = 3$, the third sketch and the two proceeding ones have all been observed by the network. The error bars are smaller than the symbol sizes for most measurements.

5.4.1 Single Model

Bayesian surprise for a gun model from the test data set can be seen in figure 17.

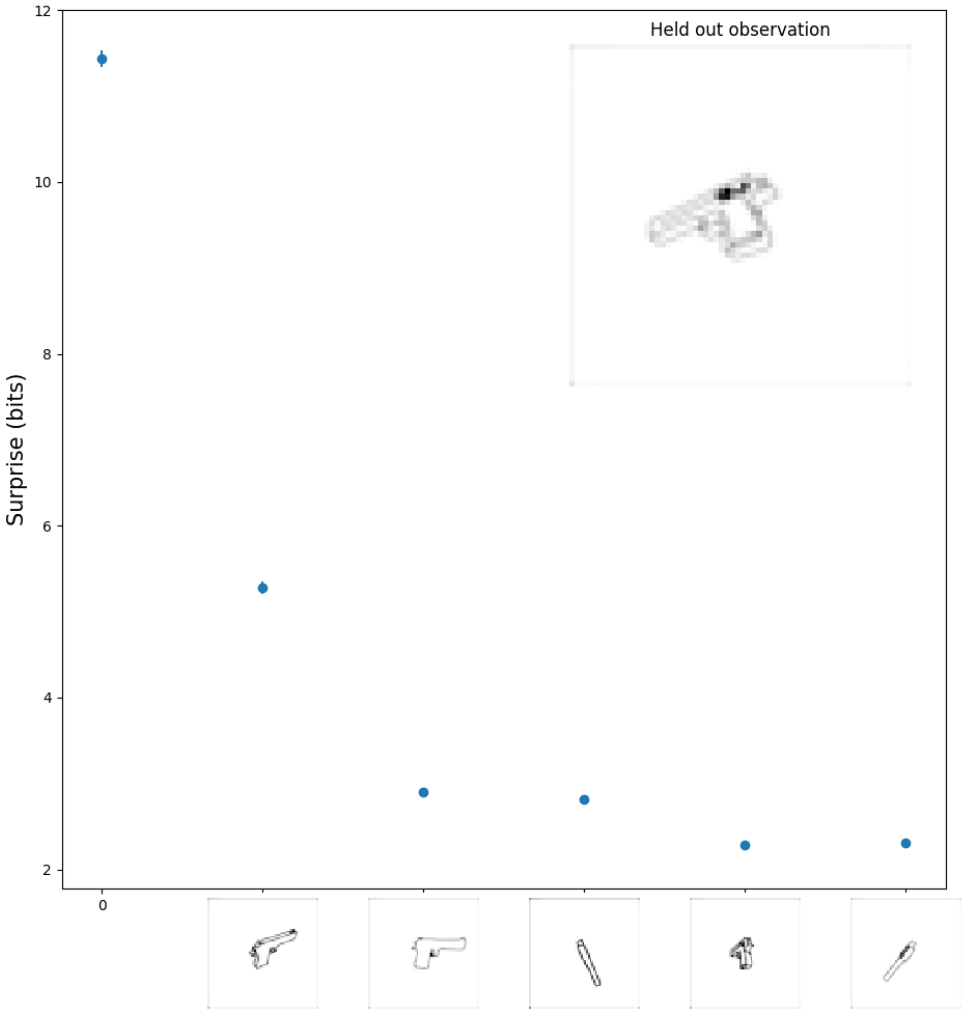

Figure 17: Bayesian surprise for a gun model

Bayesian surprise for a chair model from the test data set can be seen in figure 18.

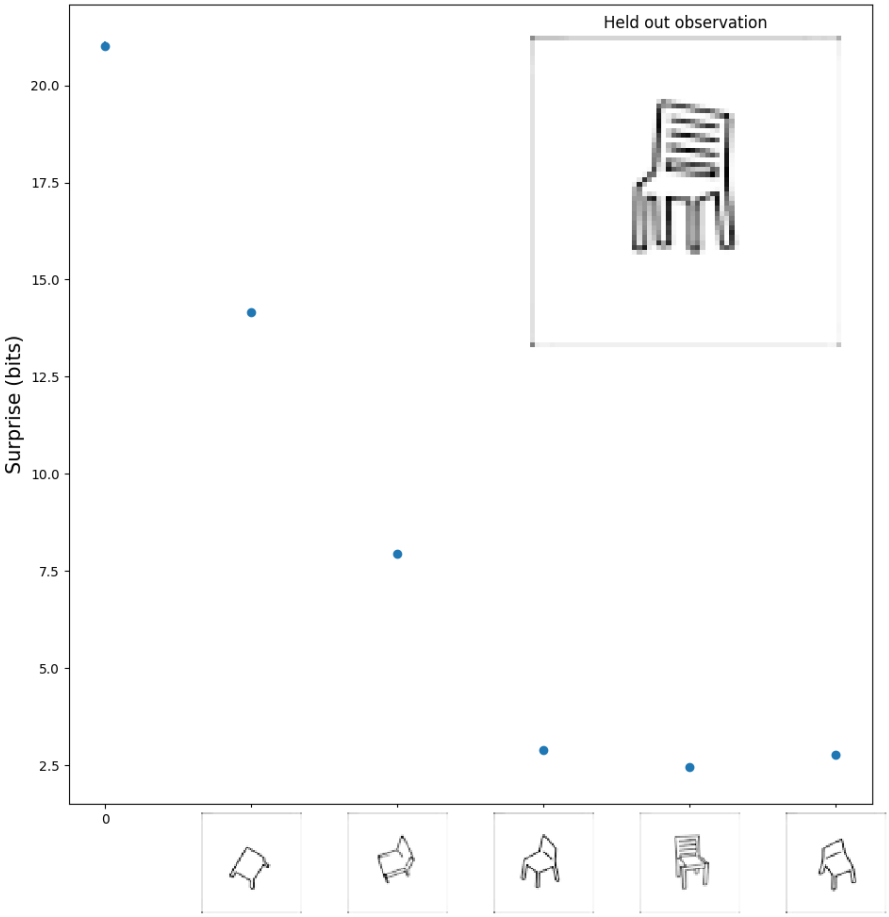

Figure 18: Bayesian surprise for a chair model

5.4.2 Averaged over classes

Average Bayesian surprise for 50 random models from the test data set. The error bars are calculated by sampling from the prior 1000 times per number of viewed sketch and model but they are smaller than the symbol sizes for most measurements. Figure 19 shows a general trend for a reduction of the model's uncertainty as the number of viewed sketches increases.

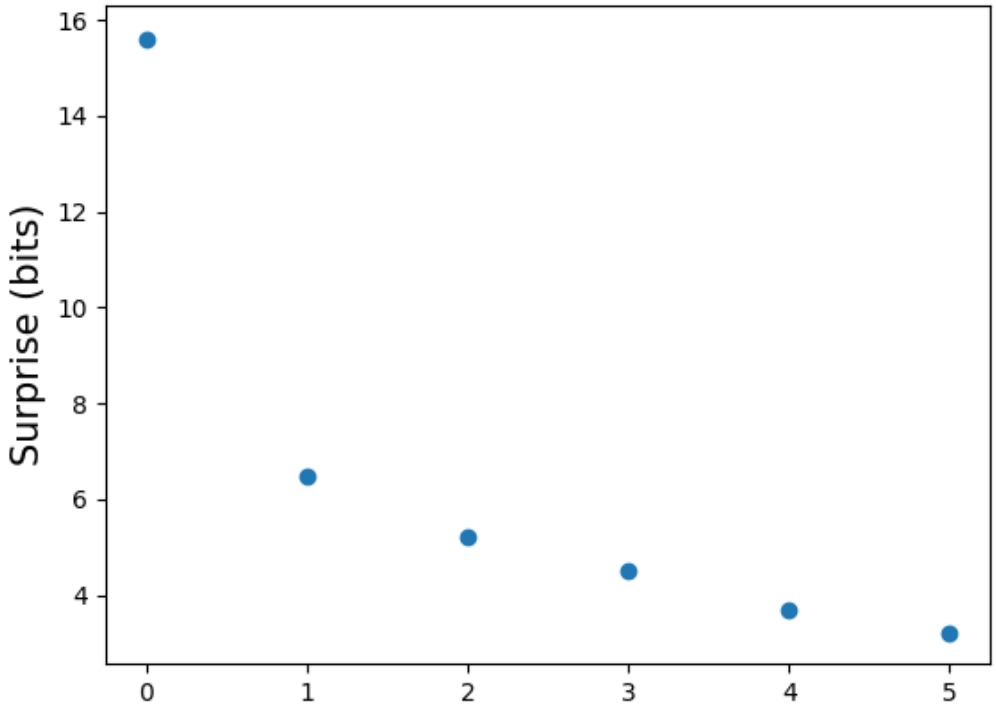

Figure 19: Average Bayesian surprise over 50 different models

5.5 Mental Rotation

Figure 20 displays the mental rotation of an airplane model from the test dataset. 15 sketches where sent in to the representation network.

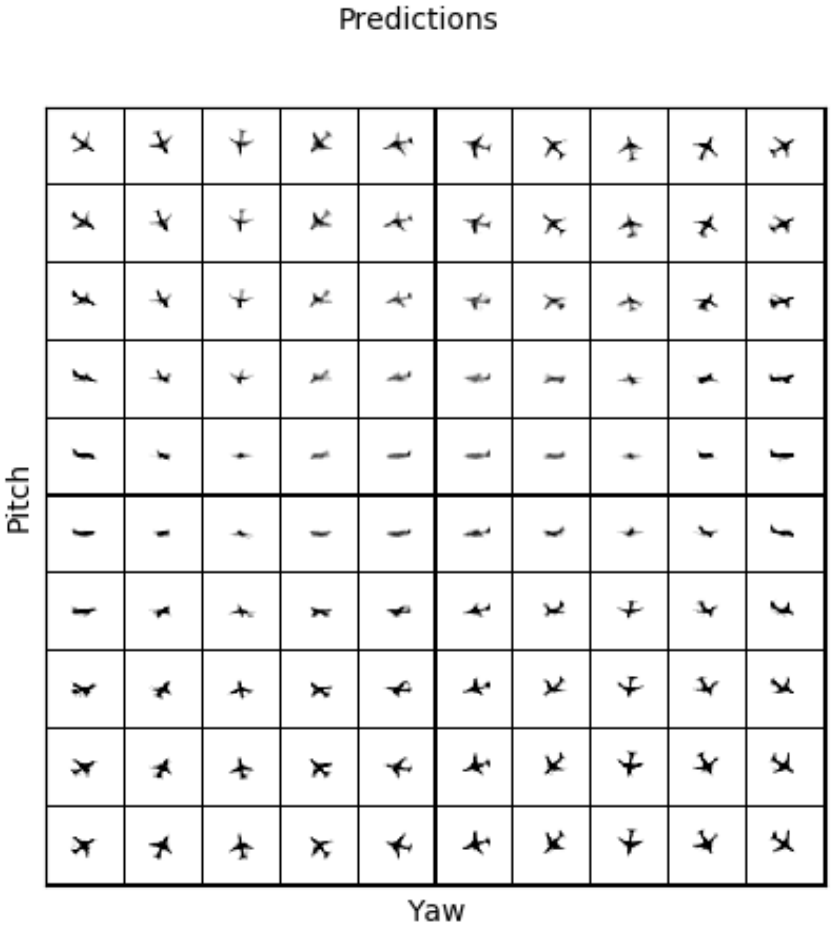

Figure 20: Mental rotation of an airplane model

Figure 21 displays the mental rotation of various models from the test dataset. 15 sketches where sent in to the representation network for each plot.

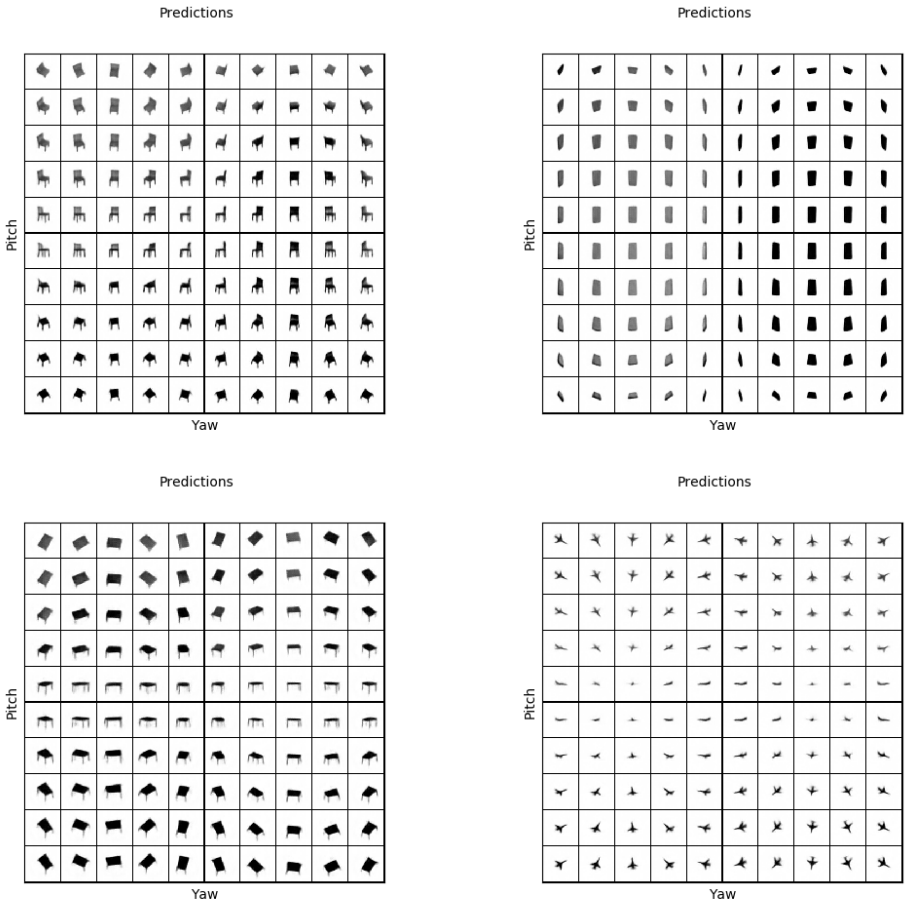

Figure 21: Upper left: Chair. Upper right: Iphone. Lower left: Table. Lower right: Fighter Jet

The VAEQN's ability to generalise to previously unseen view points gives another indication that it has not overfitted to the training data.

5.6 Single Sketch Image Generation

The following images in figure 22 and 23 have been generated by the VAEQN after observing a single sketch and corresponding view point for models from the test data set.

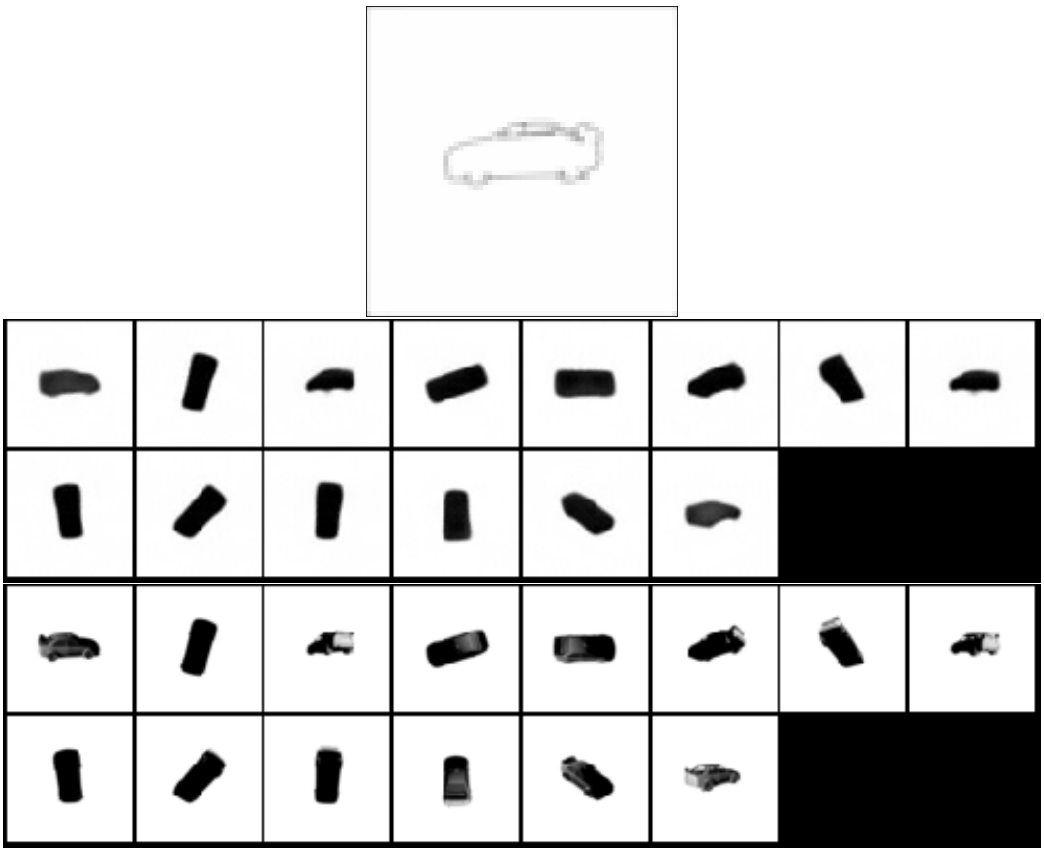

Figure 22: Top: observed sketch. Middle: generated images. Bottom: ground truths

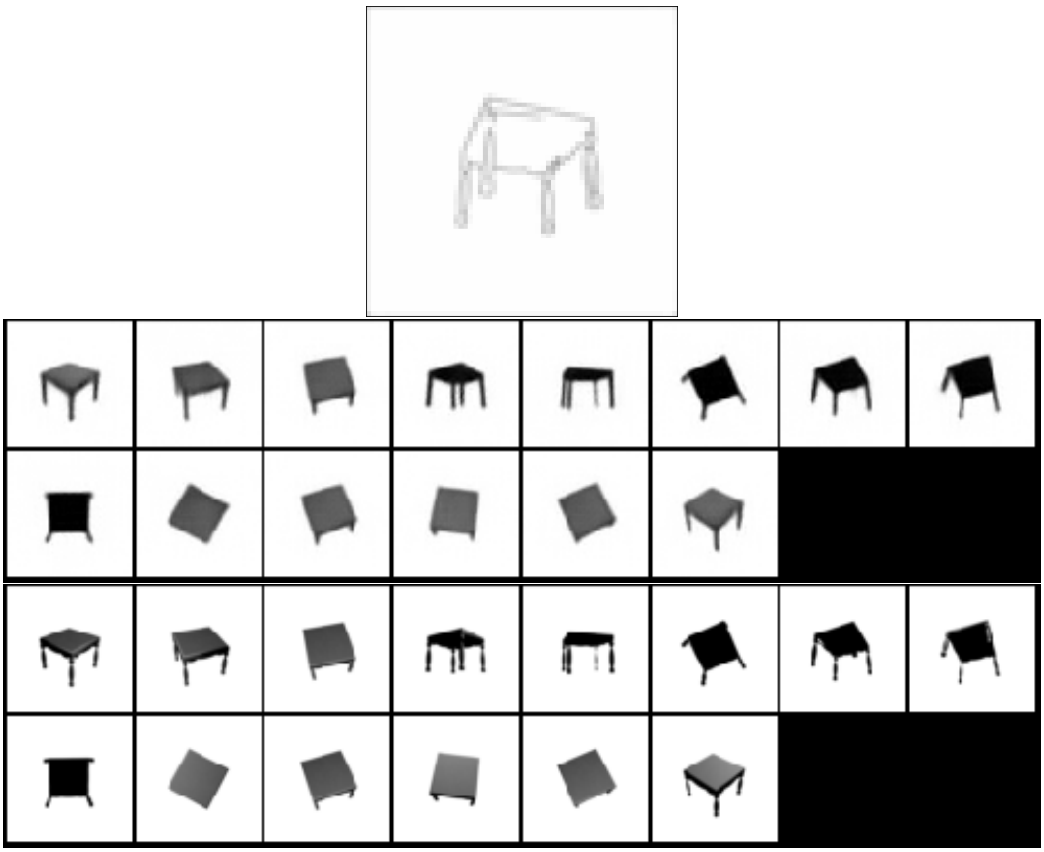

Figure 23: Top: observed sketch. Middle: generated images. Bottom: ground truths

Single image generation is important if a sketch is to be turned into 3D form. This is because sketching a model from different angles and providing the view point of each sketch is a non-trivial task for a user.

5.7 Real Sketch Image Generation

Here, the generalisation ability of the VAEQN is tested by feeding in human sketched samples of two different classes, a car and a chair. To test whether the VAEQN can generate images from arbitrary angles, the mental rotation approach is used in the same way as in section 5.5. Similarly to what is done in section 5.6, a single sketch is used to render all images. The sketches were drawn in GIMP [49] and the view point was determined by finding an image from the test dataset taken from a similar perspective and then copying that image's view point. Figure 24 shows a car with the original sketch on top and the produced images below.

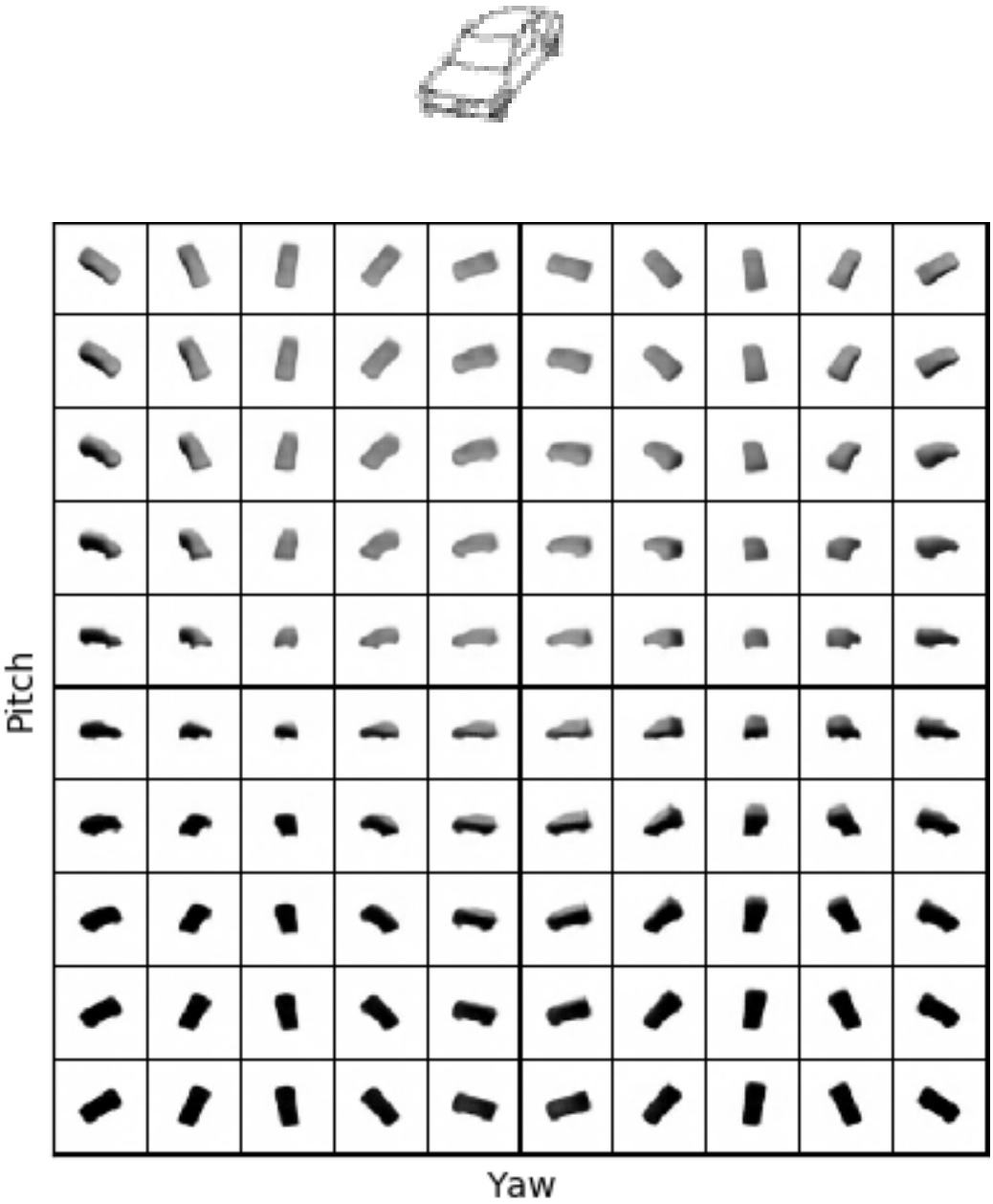

Figure 24: Top: observed sketch. Bottom: generated images

Figure 25 shows a chair with the original sketch on top and the produced images below.

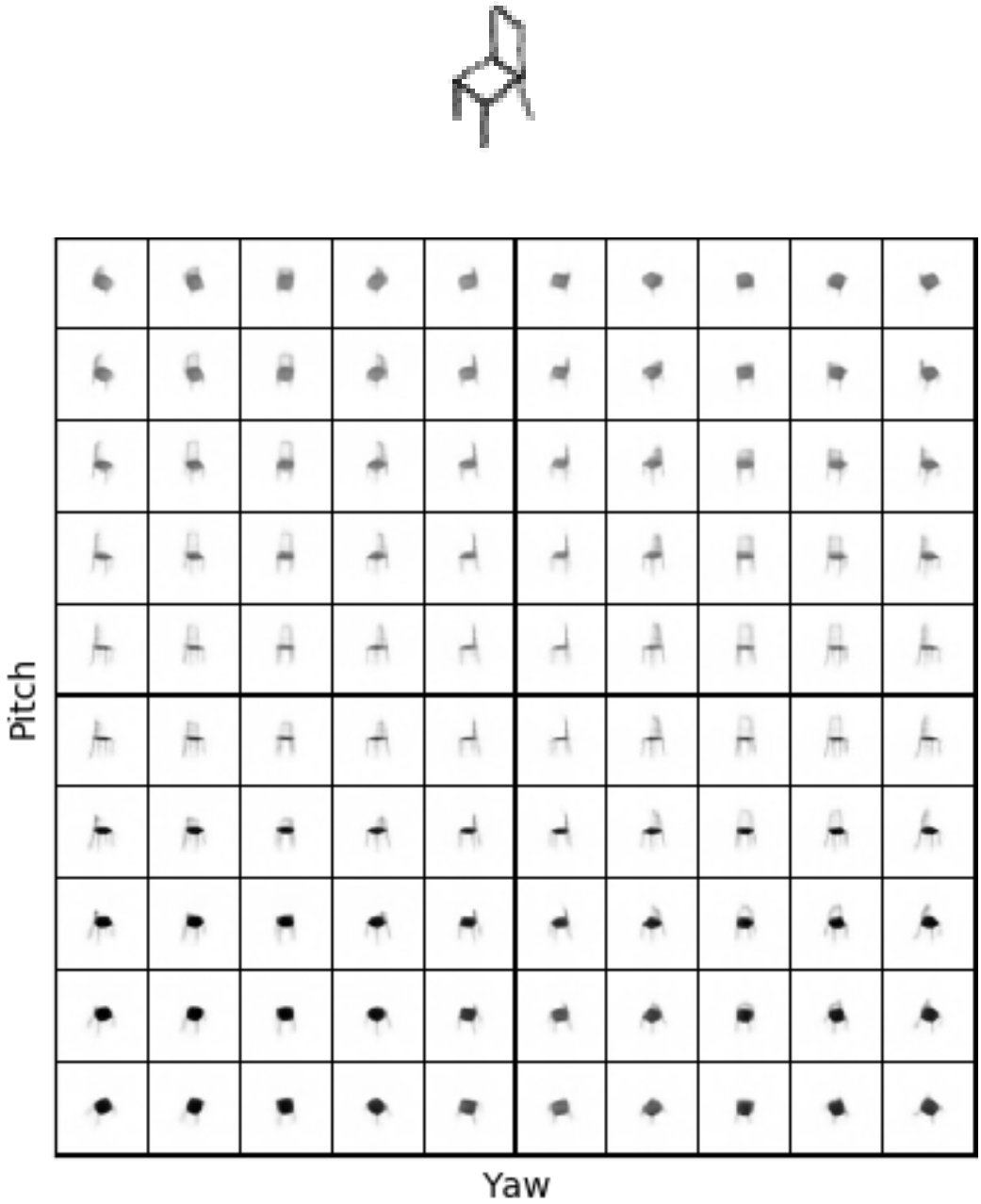

Figure 25: Top: observed sketch. Bottom: generated images

5.8 GAN trained on Shepard-Metzler

40 models generated by the GAN model by sampling from the latent distribution without any representation or view points.

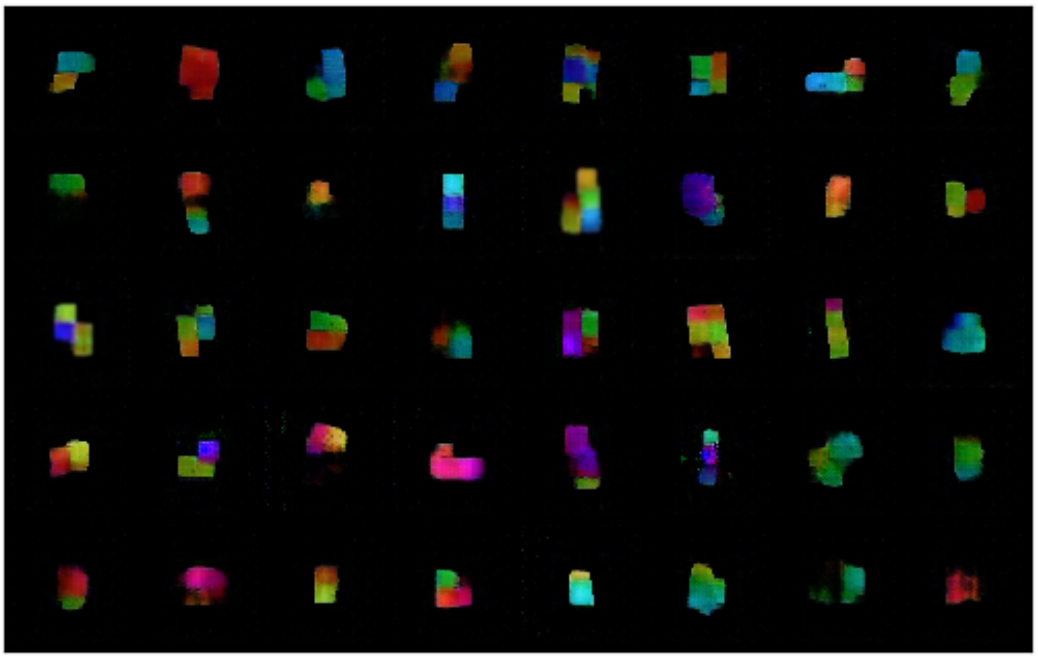

Figure 26: 40 models generated by a GAN on the Shepard-Metzler dataset

5.9 GAQN on Shepard-Metzler

The GQN is now trained with the described GAN as generative model. In figure 27 and 28, the top images are the ground truths and the lower images are generated. For each of the generated images, all the other ground truth images except for the one corresponding to that specific view point have been used to calculate the representation.

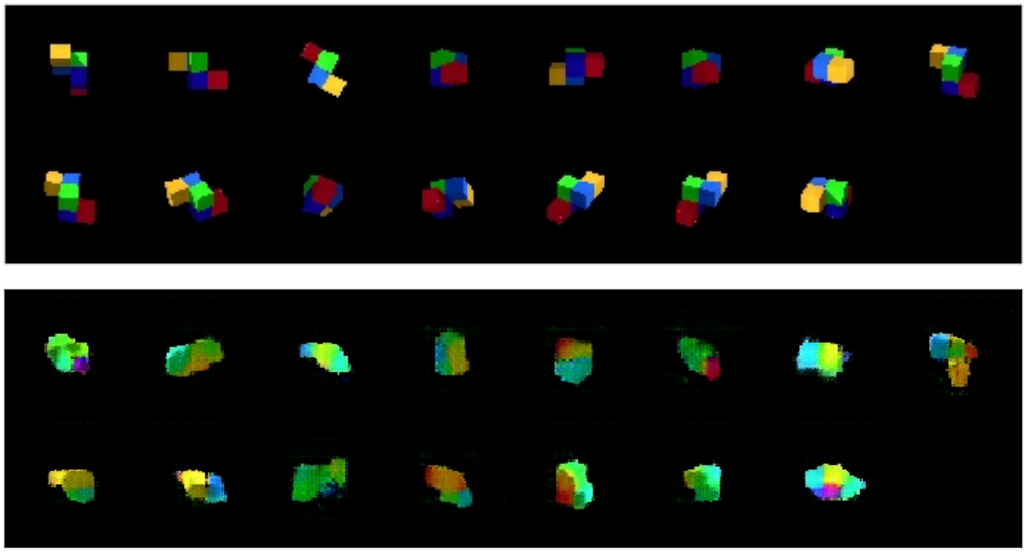

Figure 27: 15 models generated by the GQN with a GAN as generative model. Top: ground truth. Bottom: generated.

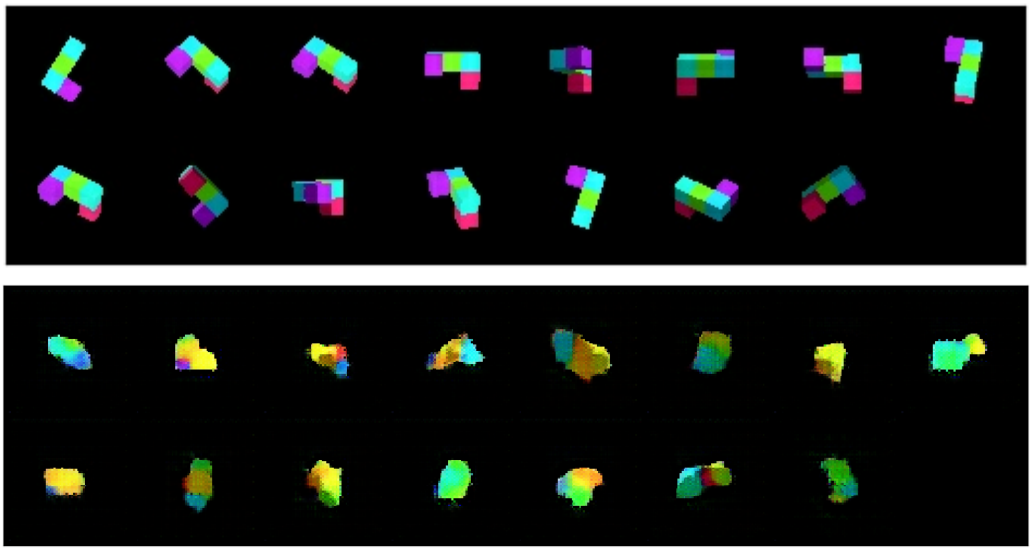

Figure 28: 15 models generated by the GQN with a GAN as generative model. Top: ground truth. Bottom: generated.

While color and shape is poorly captured, it can be seen that some of the conditional information is used by the network. For example from the view points since the generated models are bright in the correct places indicating that the network has learnt about the positioning of the light source.

5.10 GAQN on ShapeNet

The GAQN was trained on the sketch dataset presented earlier to investigate whether the extra conditioning on class id would help the model learn. The ground truth images can be seen in figure 29 and the images in figure 30 where generated in the same way as in section 5.9. It appears as if the network largely ignores the class id.

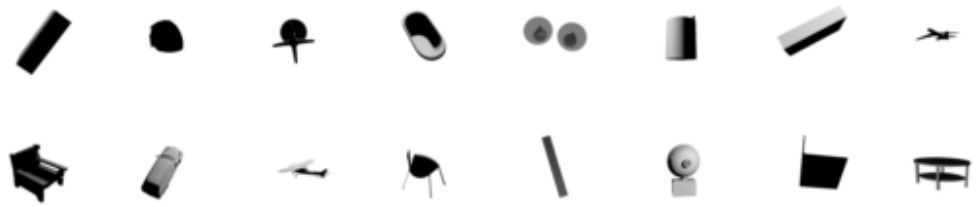

Figure 29: Ground truth images

Figure 30: Generated images

5.11 GAQN's usage of the representation and query view

Here the GAQN's usage of the representation and view points is tested by simply setting them to zero one at a time. All figures use the same scene shown in figure 31 and are generated in the same way as in section 5.9. Figure 32 shows the regular output without any tampering with the representation or view points.

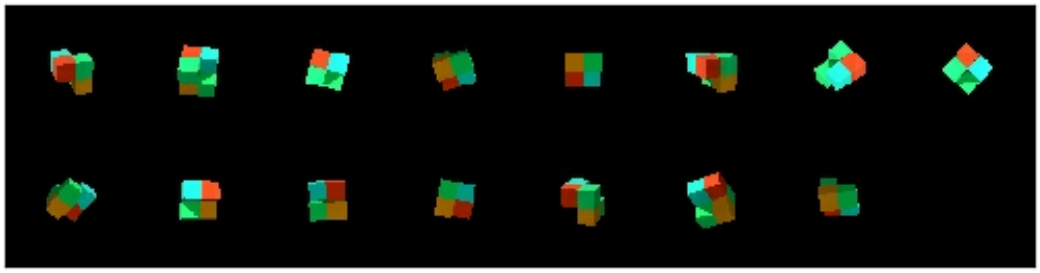

Figure 31: 15 ground truth models

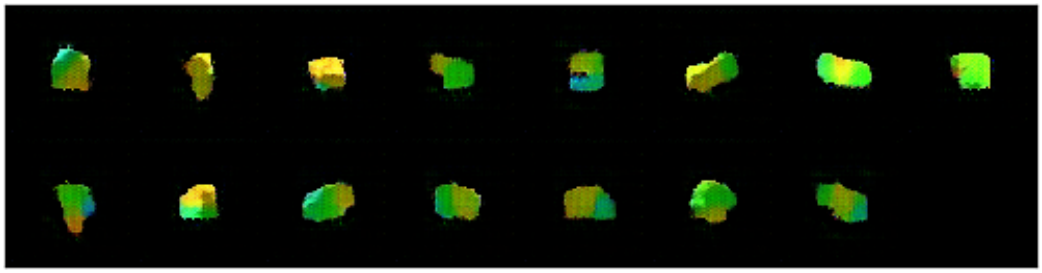

Figure 32: 15 generated models with regular representation and view points

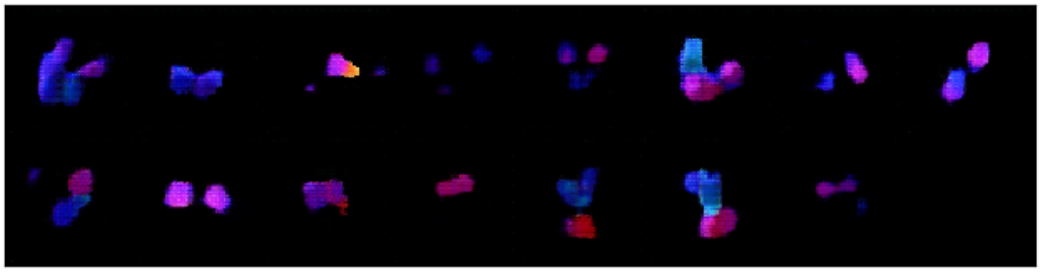

Figure 33: 15 generated models with regular view points but representation set to zero

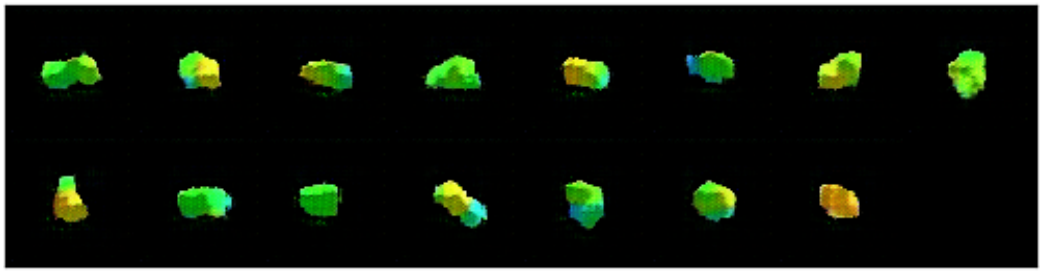

Figure 34: 15 generated models with regular representation but view points set to zero

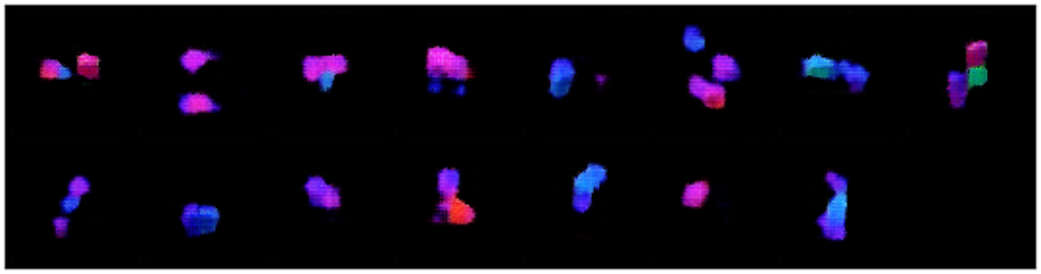

Figure 35: 15 generated models with both representation and view points set to zero

It can be seen in figure 33 that alternating the representation indeed changes the produced output. In figure, 34 it is evident that by setting the view point to zero, the lighting is no longer correctly captured and instead all models are brighter. In figure 35 the result of setting both the representation and view points to zero can be observed.

5.12 Summary of Results

A summary of the results obtained from the VAEQN and GAQN including the number of parameters in each model and their total training time.

Model	Parameters	Perceptual Loss	Bayesian Surprise	Training time
VAEQN	49426837	0.18±0.003	3.31±0.3	16d 23h
GAQN	21688900	N/A	N/A	8h 30m

Figure 36: Summary of results and information about the VAEQN and GAQN models

6 Conclusions

6.1 Discussion

The sketch-to-3D-model system using the Convolutional DRAW proved to be able to produce quite convincing images of 3D models from only observing sketches of the models. The system was able to produce decent samples from only observing a single sketch as can be seen in section 5.6, which is desirable if such a system were to be used by actual artists since drawing a model from several angles and providing the exact position from where it was drawn is a non-trivial task which would likely require additional software.

While the results from the perceptual loss in section 5.3 might not say too much when the values cannot be directly compared with other generative models, the decrease of its value as well as of the Bayesian surprise in section 5.4 as more view points are observed indicates that the network really uses the neural representation and that by receiving more information, at least quantitatively better images are produced. The results measured on the Bayesian surprise as well as the networks performance on the mental rotation task shows that the GQN framework indeed is capable of generalizing to more diverse data than what it was tested on in the original work [13].

The ShapeNet dataset is highly unbalanced, some classes such as chairs are represented by over 4000 models while there are only around 80 models of pillows. This is evident in the final results as seen in section 5.2 where the system is better at generating some classes than others. The Shepard-Metzler dataset contains 2 million different models of very similar objects while the ShapeNet dataset only contains 51 300 models from 55 diverse categories. Despite this difference, the VAEQN network produces images fairly close to the ground truths and from the correct perspectives after only a forth of the amount of gradient updates done by the DeepMind group [13] when they trained their model on the Shepard-Metzler dataset. It is fair to assume that better results can be obtained if a larger dataset is used and if the model is trained longer.

6.2 Concerning GAQN

There is much to be said about the difficulty of replacing the Convolutional DRAW with a GAN in the GQN. If the results from the GAN model are compared with those from the GAQN as seen in section 5.8 and 5.9, it seems as if the GAQN fails to properly condition on the representation and view

points and mainly generates data based on the latent vector. Part of the problem might derive from the ambiguity in how to train the representation networks. If e.g. a single representation network is used, the generator and discriminator might want to update the representation network in very different ways. The generator network might, for example, try to update the representation network so that its representation confuses the discriminator rather than learning an informative scene representation.

6.3 Future Work

Turning the models into sketch form resulted in most of the details being lost since only the general outline of the models remained. During training the network did however still observe the original image with full detail when calculating the likelihood term of the ELBO as well as when inferring the variational posterior. From this, the network seemed to learn about the position of the light source and also how to generate the rest of the image to make it look more like the ground truth. Additionally, the images in section 3.6 are shown in resolution 512×512 pixels when they, in fact, during training are down-sampled to 64×64 pixels. This does also result in some high frequency details being lost. Future work could investigate whether it is possible to generate a greater amount of details in the rendered images by e.g. following a similar scheme to what is done by A. Heljakka et al. in *Pioneer Networks: Progressively Growing Generative Autoencoder* [45] where the image size is progressively increased.

In appendix A.2, several architectural changes can be seen which where constructed and tested in an attempt to encourage the network to use the conditional variables. For further research it would be interesting to investigate whether there are other representation network architectures or set-ups that are better suited for GANs than the three presented in the GQN paper [13].

Part of the Convolutional DRAW's strength is that it recurrently encodes the conditional variables into its prior distribution from where the final latent sample is produced, it can thus make sure to draw this sample from the region which corresponds to that particular model from that particular view and then simply decode this into an image. The GAN, on the other hand, samples a latent vector before receiving any information about the model and then has to modify this to fit to the representation and view point. In theory this should not be a problem but it would be interesting to test out whether networks such as BiGAN [16] or VAE-GANs [17] that mixes concepts from GANs and VAEs are better suited as generative models in GQNs than GANs.

In addition, to further develop the VAEQN to produce better images of 3D models from actual sketches it would be possible to first train it on the synthetic sketch dataset developed in this thesis and then do transfer learning on a much smaller dataset containing real sketches. To generate higher quality images, a larger dataset would most likely have to be created where the different classes have about the same number of models present.

6.4 Summary

In this thesis, it is shown that Generative Query Networks are able to both generalise to a more diverse dataset as well as generating plausible images of 3D models after only receiving information in the form of sketches and view points.

The thesis also investigated whether GANs could be used as the generative model in the GQN framework. While there are no theoretical reasons why GANs could not work, it has been evident that training the GQN with them is far from trivial. This comes as no surprise as GANs in general exhibit this behaviour.

From the results it is possible to conclude that Generative Query Networks is a viable option for creating a sketch-to-3D-model application as long as the final 3D model is not explicitly needed. The network produces a 3D model in separate 2D images, it is thus not possible to e.g. import it directly into a game engine which would be possible if the final 3D model is produced in the form of a polygon mesh. However, for quick prototyping where you only want to get a feeling for what the 3D model would look like before actually creating it, the approach is promising due to the relatively good quality of the network's outputs.

References

[1] D. P. Kingma and J. L. Ba (2015). *Adam: a method for stochastic optimization*. 3rd International Conference on Learning Representations (ICLR), San Diego, CA.

[2] A. Kumar, S.M. Ali Eslami, D.J Rezende, Murray Shanahan (2018). *Consistent Generative Query Networks*. DeepMind, London N1C4AG.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio (2014). *Generative Adversarial Nets*. Departement dinformatique et de recherche operationnelle Universite de Montreal Montreal, QC H3.C 3J7.

[4] M. Arjovsky, S. Chintala, and L. Bottou (2017). *Wasserstein GAN*. Courant Institute of Mathematical Sciences, Facebook AI Research.

[5] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin and A. Courville (2017). *Improved Training of Wasserstein GANs*. Montreal Institute for Learning Algorithms, Courant Institute of Mathematical Sciences.

[6] M. Mirza and S. Osindero (2014). *Conditional Generative Adversarial Nets*. Departement dinformatique et de recherche operationnelle Universite de Montreal Montreal, QC H3C 3J7. Flickr / Yahoo Inc. San Francisco, CA 94103.

[7] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi (2017). *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*. Twitter.

[8] T. Karras, T. Aila, S. Laine and J. Lehtinen (2017). *Progressive Growing of GANs for Improved Quality, Stability, and Variation*. NVIDIA and Aalto University.

[9] Diederik P Kingma and Max Welling (2013). *Auto-Encoding Variational Bayes*. Machine Learning Group, Universiteit van Amsterdam.

[10] H. Kato, Y. Ushiku and T. Harada (2017). *Neural 3D Mesh Renderer*. 1,21 The University of Tokyo,2 RIKEN.

[11] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi and F. Yu (2015). *ShapeNet: An Information-Rich 3D Model Repository*. Stanford University, Princeton University, Toyota Technological Institute at Chicago.

[12] K. Gregor, F. Besse, D. J. Rezende, I. Danihelka and D. Wierstra (2016). *Towards Conceptual Compression.* Google DeepMind, London, United Kingdom.

[13] S. M. Ali Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu and D. Hassabis (2018). *Neural scene representation and rendering.* DeepMind, 5 New Street Square, London EC4A 3TW, UK.

[14] S. M. Ali Eslami, D. J. Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, D. P. Reichert, L. Buesing, T. Weber, O. Vinyals, D. Rosenbaum, N. Rabinowitz, H. King, C. Hillier, M. Botvinick, D. Wierstra, K. Kavukcuoglu and D. Hassabis (2018). *Supplementary Materials for Neural scene representation and rendering.* DeepMind, 5 New Street Square, London EC4A 3TW, UK.

[15] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende and D. Wierstra (2015). *DRAW: A Recurrent Neural Network For Image Generation.* DeepMind, 5 New Street Square, London EC4A 3TW, UK.

[16] J. Donahue, P. Krähenbühl and T. Darrel (2017). *Adversarial Feature Learning.* arXiv:1605.09782v7 [cs.LG].

[17] A. B. Lindbo Larsen, S. K. Sonderby, H. Larochelle and O. Winther (2016). *Autoencoding beyond pixels using a learned similarity metric* arXiv:1512.09300v2 [cs.LG].

[18] P. Sangkloy, J. Lu, C. Fang, F. Yu and J. Hays (2016). *Scribbler: Controlling Deep Image Synthesis with Sketch and Color.* Georgia Institute of Technology. Adobe Research. Princeton University.

[19] Y. Gucluturk, U. Guclu, R. van Lier, and M. A. J. van Gerven (2016). *Convolutional Sketch Inversion.* Radboud University, Donders Institute for Brain, Cognition and Behaviour, Nijmegen, the Netherlands.

[20] E. Simo-Serra, S. Iizuka, K. Sasaki and H. Ishikawa (2016). *Learning to Simplify: Fully Convolutional Networks for Rough Sketch Cleanup.* ACM Transactions on Graphics (SIGGRAPH)', 2016, 35, 4.

[21] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao (2015). *3D ShapeNets: A Deep Representation for Volumetric Shapes.* Princeton

University. Chinese University of Hong Kong. Massachusetts Institute of Technology.

[22] J. Delanoy, M. Aubry, P. Isola, A. A. Efros and A. Bousseau (2017). *3D Sketching using Multi-View Deep Volumetric Prediction*. arXiv:1707.08390v4 [cs.GR].

[23] Z. Lun, M. Gadelha, E. Kalogerakis, S. Maji and R. Wang (2017). *3D Shape Reconstruction from Sketches via Multi-view Convolutional Networks*. University of Massachusetts Amherst.

[24] J.J. Park, P. Florence, J. Straub, R. Newcombe and S. Lovegrove (2019). *DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation*. University of Washington, Massachusetts Institute of Technology, Facebook Reality Labs.

[25] J. Wu, Y. Wang, T. Xue, X. Sun, W. T. Freeman, J. B. Tenenbaum (2017). *MarrNet: 3D Shape Reconstruction via 2.5D Sketches*. MIT CSAIL. Google Research. Shanghai Jiao Tong University. Shanghai Tech University.

[26] P. Welander, S. Karlsson and A. Eklund (2018). *Generative Adversarial Networks for Image-to-Image Translation on Multi-Contrast MR Images A Comparison of CycleGAN and UNIT*. Division of Medical Informatics, Department of Biomedical Engineering, Linköping University. Division of Statistics and Machine Learning, Department of Computer and Information Science, Linköping University. Center for Medical Image Science and Visualization (CMIV),Linköping University, Linköping, Sweden.

[27] K. Simonyan and A. Zisserman (2015). *Very deep convolutional networks for large-scale image recognition*. International Conference onLearning Representations (ICLR), 2015.

[28] J. Johnson, A. Alahi and F. Li (2016). *Perceptual Losses for Real-Time Style Transfer and Super-Resolution*. Department of Computer Science, Stanford University.

[29] A. Mahendran and A. Vedaldi (2015). *Understanding deep image representations by inverting them*. Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR). (2015).

[30] I. Bichindaritz (2016). *MOOC course in Big Data, Genes, and Medicine*. `https://www.coursera.org/lecture/data-genes-medicine/data-normalization-jGN7k`. The State University of New York.

[31] A. Brock, J. Donahue and K. Simonyan (2018). *Large Scale GAN Training For High Fidelity Natural Image Synthesis*. DeepMind, arXiv:1809.11096 [cs.LG].

[32] T. Miyato, T. Kataoka, M. Koyama and Y. Yoshida (2018). *Spectral Normalization for Generative Adversarial Networks*. Preferred Networks, Inc. Ritsumeikan University. National Institute of Informatics.

[33] H. Zhang, I. Goodfellow, D. Metaxas and A. Odena (2018). *Self-Attention Generative Adversarial Networks*. Rutgers University. Google Brain.

[34] A. Radford, L. Metz and S. Chintala (2015). *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. indico Research Boston, MA. Facebook AI Research New York, NY.

[35] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford and X. Chen (2016). *Improved Techniques for Training GANs*. arXiv:1606.03498v1 [cs.LG] 10 Jun 2016.

[36] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2015). *Rethinking the inception architecture for computer vision*. arXiv preprint arXiv:1512.00567, 2015.

[37] M. Heusel, H. Ramsauer, T. Unterthiner and B. Nessler (2018). *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. arXiv:1706.08500v6 [cs.LG] 12 Jan 2018.

[38] J. P. W. Pluim, J. B. A. Maintz and M. A. Viergever (2003). *Mutual information based registration of medical images: a survey*. IEEE Transactions on Medical Imaging, vol. xx, no. y, month 2003.

[39] Q. Yang, N. Li, Z. Zhao, X. Fan, E. I-Chao Chang and Y. Xu (2018). *MRI Cross-Modality NeuroImage-to-NeuroImage Translation*. arXiv:1801.06940 [cs.CV].

[40] L. E. Baum and T. Petrie (1966). *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*. The Annals of Mathematical Statistics. 37 (6): 15541563. doi:10.1214/aoms/1177699147, (1966).

[41] Douglas Reynolds (2005). *Gaussian Mixture Models*. MIT Lincoln Laboratory, 244 Wood St., Lexington, MA 02140, USA.

[42] S. Hochreiter and J. Schmidhuber (1997). *Long short-term memory*. Neural computation, 9(8):17351780, 1997.

[43] R. N. Shepard and J. Metzler (1971). *Mental rotation of three-dimensional objects*. Science 171, 701703 (1971). doi:10.1126/science.171.3972.701 Medline.

[44] T. Karras, S. Laine and T. Aila (2018). *A Style-Based Generator Architecture for Generative Adversarial Networks*. Nvidia.

[45] A. Heljakka, A. Solin and J. Kannala (2018). *Pioneer Networks: Progressively Growing Generative Autoencoder*. Aalto University. GenMind Ltd.

[46] Blender Online Community. *Blender - a 3D modeling and rendering package*. Blender Foundation, Blender Institute, Amsterdam `http://www.blender.org`.

[47] *PIL - Python Imaging Library*. `https://pillow.readthedocs.io/en/4.0.x/reference/ImageFilter.html`

[48] *PyTorch*. `https://pytorch.org/docs/stable/tensors.html`

[49] *GIMP - GNU Image Manipulation Program*. `https://www.gimp.org/`

Appendices

A GAN Architectures

A.1 Baseline

The network architectures in figure 37 were the ones used for the GAN in this master thesis. In the figure, h is the activation from the previous layer and the numbers correspond to the following variables $(channels) \times (image\ width) \times (image\ height)$, k is the kernel size, s is the stride and p is the padding where the same value is used for the height and width dimensions. Channels and image sizes are written as the sizes obtained after that particular layer.

$z \in \mathbb{R}^{100} \sim \mathcal{N}(0, I)$
Linear, $1024 \times 4 \times 4$
Concat(h, r), $(1024 + 16) \times 4 \times 4$
ConvTranspose2d, $512 \times 8 \times 8, k : 4, s : 1, p : 1$
Concat(h, v_q), $(512 + 7) \times 8 \times 8$
ConvTranspose2d, $512 \times 16 \times 16, k : 4, s : 1, p : 1$
ConvTranspose2d, $256 \times 32 \times 32, k : 4, s : 1, p : 1$
ConvTranspose2d, $64 \times 64 \times 64, k : 4, s : 1, p : 1$
Self_Attention, $64 \times 64 \times 64$
ConvTranspose2d, $3 \times 64 \times 64, k : 3, s : 1, p : 1$
Tanh

(a) Generator Network

x_{gen} *or* x_{real}
Conv2d, $64 \times 32 \times 32, k : 4, s : 2, p : 1$
Conv2d, $64 \times 16 \times 16, k : 4, s : 2, p : 1$
Concat(h, r), $(64 + 7 + 1) \times 16 \times 16$
Conv2d, $72 \times 6 \times 6, k : 5, s : 2, p : 0$
Conv2d, $512 \times 3 \times 3, k : 2, s : 2, p : 0$
Self_Attention, $64 \times 64 \times 64$
Conv2d, $512 \times 3 \times 3, k : 1, s : 1, p : 0$
Linear, 1

(b) Discriminator Network

Figure 37: Network Architectures for a) generator and b) discriminator.

A.2 Tested GAN architectures

Abbreviating "Same as in figure 37" as "*", the following are other architectures that were tested:

* with 2×width of all layers as in BigGAN [31].
* with conditional variables concatenated at several different layers.
* with Self_Attention at several different layers.
* with instance noise in the discriminator.
* with 3 additional layers in both networks.
* with gradient penalty.
* with shared representation network trained only by discriminator.
* with shared representation network trained only by generator.
* with shared representation network trained by both networks.
* with tower/pyramid representation structure.
* with an attention mechanism on the representation networks before summing.
* with noise added to generator layers as done by Karras et al. [44].
* letting the discriminator predict the view point instead of concatenating it.
* adding a perceptual loss to the generator as done by C. Ledig et al. [7].
G and D changed to WGAN-GP with the same architectures as in the original paper [4].
G and D changed to WGAN-GP with shared and separate representation networks.

Table 1: Tested network architectures

B Additional Sketches from the dataset

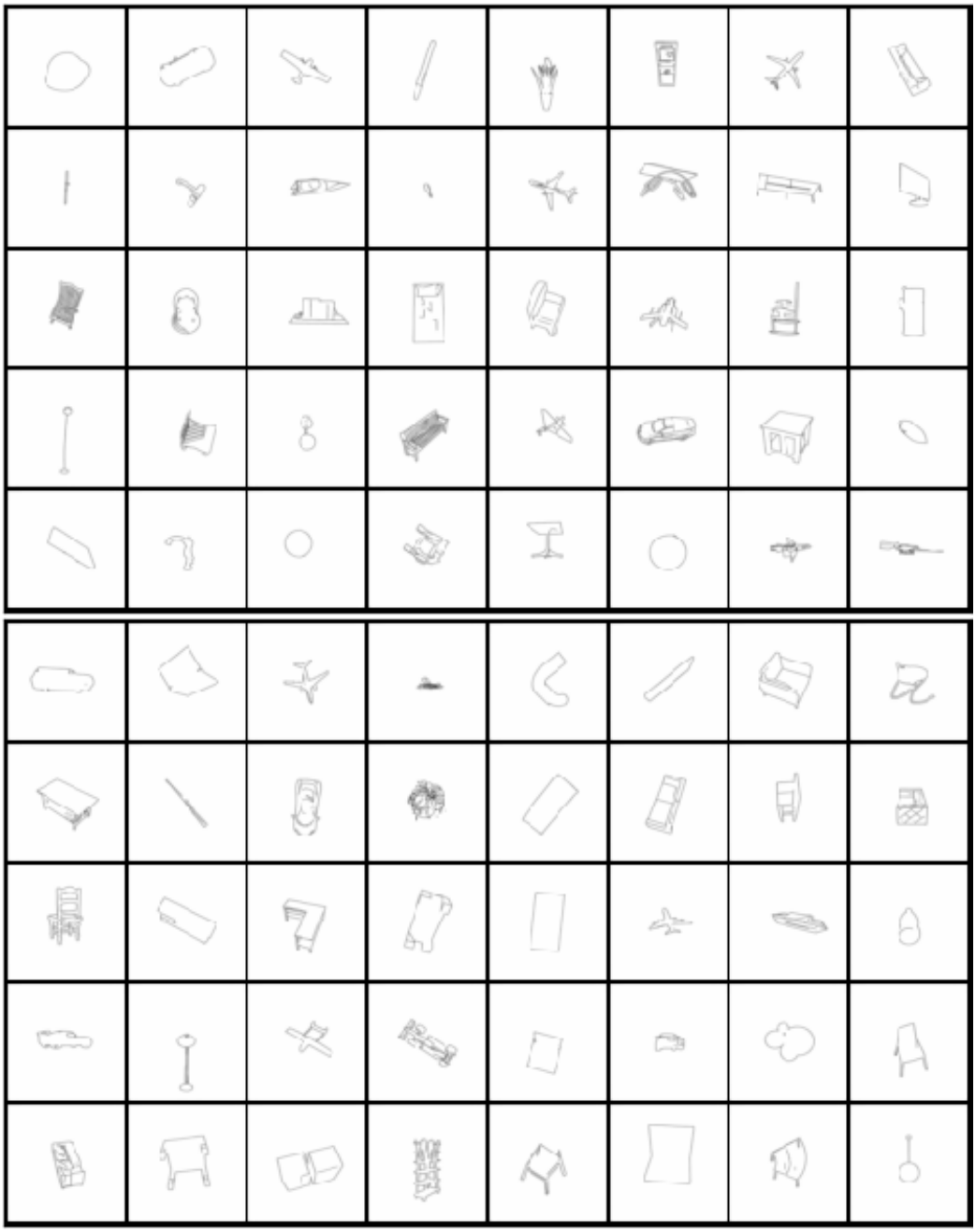

Figure 38: Sketch images from the test dataset

C Additional images generated by the VAEQN

The following images have been generated by the VAEQN after observing 14 sketches from different view points for random classes in the test data set.

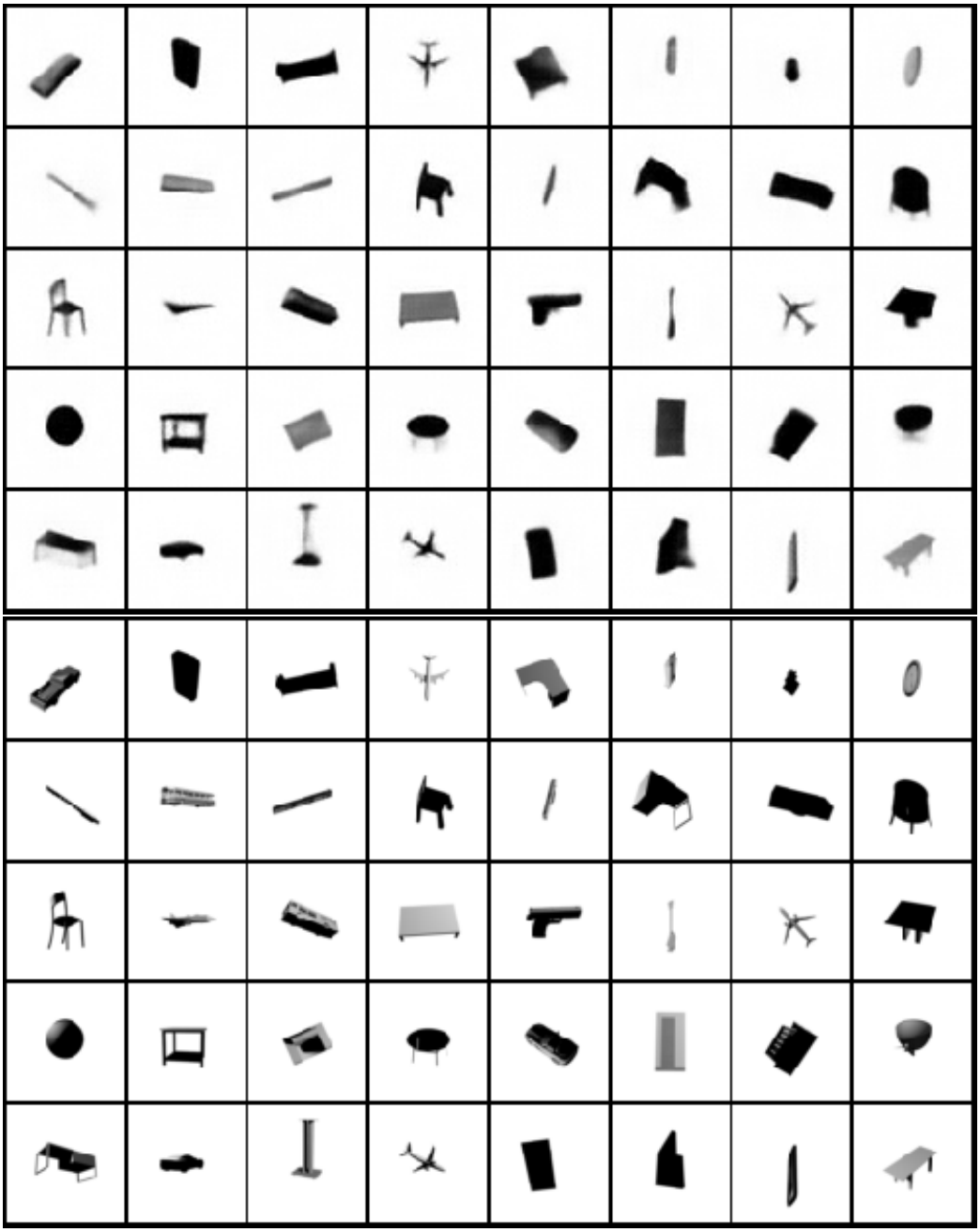

Figure 39: Top: Generated. Bottom: Ground truth

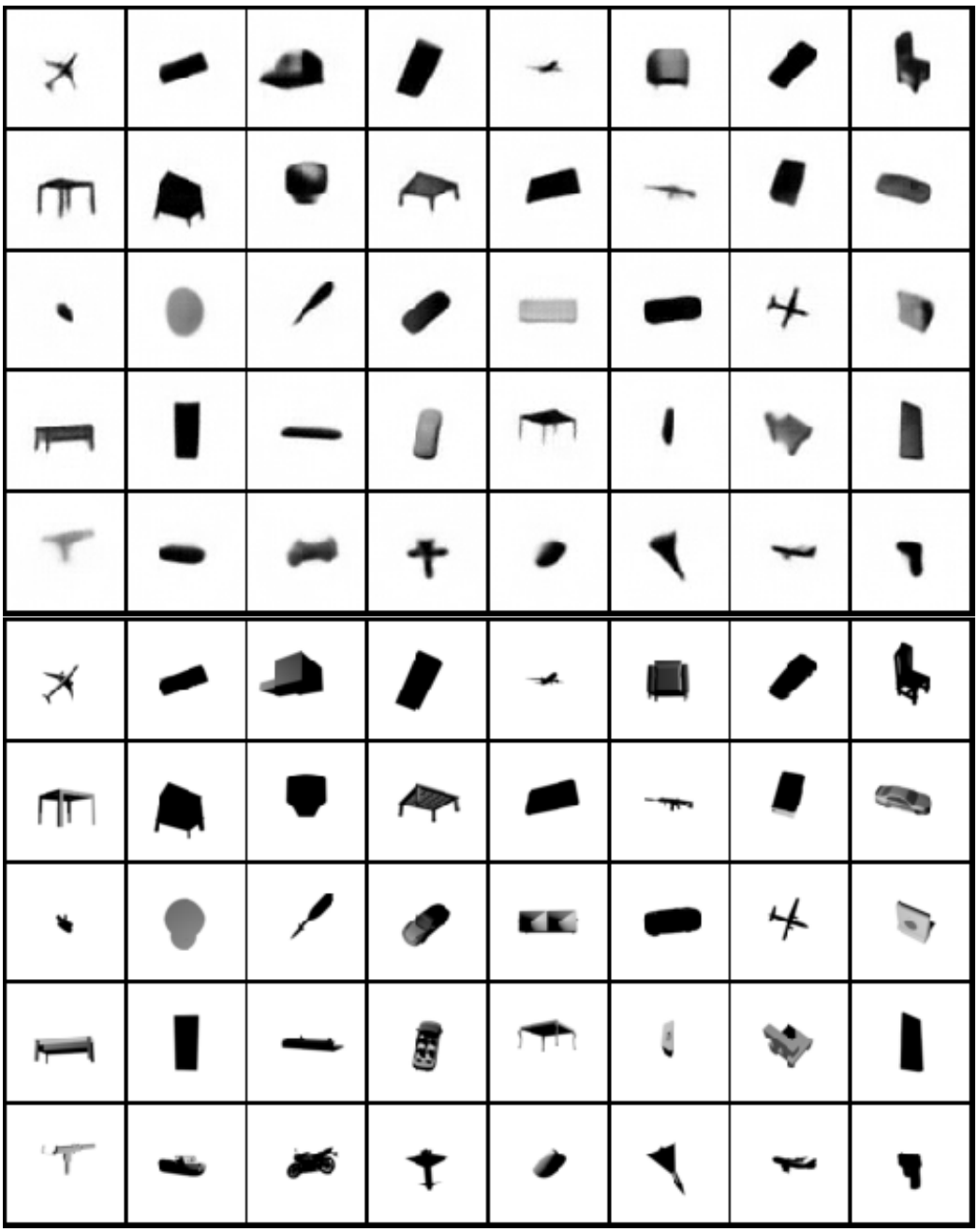

Figure 40: Top: Generated. Bottom: Ground truth