Semantics-guided Exploration of Latent Spaces for Shape Synthesis

by

Tansin Jahan

A thesis submitted to the
Faculty of Graduate and Postdoctoral Affairs
in partial fulfillment of the requirements for the degree of

Master of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
Ottawa, Ontario
November, 2019

The undersigned hereby recommends to the
Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Semantics-guided Exploration of Latent Spaces for Shape Synthesis

submitted by **Tansin Jahan**

in partial fulfillment of the requirements for the degree of

Master of Computer Science

Professor Oliver van Kaick, Thesis Supervisor

Professor David Mould, School of Computer Science

Professor Yongyi Mao,
School of Electrical Engineering and Computer Science

Professor Matthew Holden, Chair,
School of Computer Science

Ottawa-Carleton Institute for Computer Science
The School of Computer Science
Carleton University
November, 2019

Abstract

We introduce an approach to incorporate user guidance into shape synthesis approaches based on deep networks. Synthesis networks such as auto-encoders are trained to encode shapes into latent vectors, effectively learning a latent shape space that can be sampled for generating new shapes. Our main idea is to allow users to start an exploratory process of the shape space with the use of high-level semantic keywords. Specifically, the user inputs a set of keywords that describe the general attributes of the shape to be generated, e.g., "four legs" for a chair. Next, we map the keywords to a subspace of the latent space that captures the shapes possessing the specified attributes. The user then explores only the subspace to search for shapes that satisfy the design goal, in a process similar to using a parametric shape model. Our exploratory approach allows users to model shapes at a high-level without the need of advanced artistic skills, in contrast to existing methods that allow to guide the synthesis with sketching or partial modeling of a shape. Our technical contribution to enable this exploration-based approach is the introduction of a label regression neural network coupled with a shape synthesis neural network. The label regression network takes the user-provided keywords and maps them to the corresponding subspace of the latent space, where the subspace is modeled as a set of distributions for the dimensions of the latent space. We show that our method allows users to efficiently explore the shape space and generate a variety of shapes with selected high-level attributes.

Acknowledgments

At first, I would like to express my deep gratitude to Dr. Oliver van Kaick, my research supervisor, for giving me the opportunity and immense support to make this research a reality. His encouragement, guidance and patience truly helped me to shape this research into what it is today.

I would also like to thank my defense committee Dr. David Mould, Dr. Yongyi Mao and Dr. Matthew Holden for their valuable and useful critiques towards the thesis. Special thanks to my fellow lab member Yanran Guan, who helped me in labeling the training dataset for this research. My sincere thanks goes to all members of GIGL lab for their consistent motivation and support in the past two years.

I am also grateful to the School of Computer Science, Carleton University for accepting me as a research student and facilitating me with financial assistance to complete my degree.

At last, I would like to thank my parents, sister, family members and my beloved husband for their unconditional love and support that encouraged me to complete my research.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Creating 3D shapes is a necessity in any domain where a 3D virtual world needs to be modeled. From medical research to the video game industry, 3D models are widely used and serve a variety of purposes, e.g, modeling biological organs for medical analysis, characters and objects for games, and models of buildings for architectural visualizations [75]. Existing 3D modeling tools such such as Blender, 3Ds Max, and Maya allow users to model 3D shapes with great precision. However, creating digital models of 3D shapes is a challenging task for novice users, since traditional modeling tools involve the creation of shapes at a low-level, where all the fine details of the surfaces have to be explicitly modeled. This is a time-consuming process and requires advanced knowledge of the modeling tools. Thus, over the last twenty years, computer graphics research has also developed alternative approaches to facilitate modeling of shapes for non-expert users.

One line of research proposes the use of parametric models that allow to model a shape at a high level. With these models, the user can synthesize new shapes, such as human faces [11] or bodies [4], by manipulating a set of sliders that specify the parameters of a shape model. Thus, the shape does not have to be explicitly modeled by the user and can be synthesized at a high-level. However, conventional parametric shape models are only applicable to shapes that can be described by a template, since a one-to-one mapping of all the shapes in a collection to the template is required by these methods.

Recently, there has been great interest in using deep neural networks for synthesizing shapes, such as variational autoencoders (VAEs) [15] or generative adversarial

networks (GANs) [73], which learn statistical models from collections of shapes. In these approaches, the networks learn to encode high-dimensional shapes into low-dimensional latent vectors and then decode the latent vectors back into shapes, so that a shape can be synthesized by simply sampling a latent vector and providing it to the decoder. Thus, the space spanned by the latent vectors can be seen as a parametric shape space. These approaches have sparked great interest since the shape spaces can be learned with weaker constrains, requiring only a consistent alignment of all the shapes in the collection rather than a one-to-one mapping to a template.

Although the shape spaces learned by the networks are low-dimensional when compared to the shapes themselves, to allow the networks to learn effective shape models, the latent vectors still need to possess a large number of dimensions, e.g., 128 dimensions or more. Thus, manually exploring the shape space spanned by the latent vectors is impractical, since many dimensions have to be considered. As a result, in the literature, most of the synthesis methods have been evaluated by randomly sampling latent vectors or interpolating these random vectors [73]. One recent work proposes a "snapping" mechanism akin to sketching approaches, where the user creates a partial model by aggregating cubic blocks with an interface similar to Minecraft game, and the method then "snaps" the shape to the manifold defined by the latent space, updating the user-modeled shape to the closest matching shape in the latent space [47]. Other works introduce approaches that convert 2D sketches drawn by a user into 3D objects, either with the use of procedural modeling [33] or deformable models [61]. These are valuable tools for aiding modeling with neural networks, but they still require sufficient artistic skills from the user.

1.2 Objective of our work

In this thesis, we develop an approach to let a user generate a variety of shapes by exploring a low-dimensional shape space learned by a neural network. As discussed above, this low-dimensional space is still challenging to navigate as it consists of a significant amount of dimensions. Thus, we introduce a method where the user has to explore only a small number of meaningful dimensions rather than all dimensions. The user starts the exploration by selecting a subspace of the latent space with a set of keywords, and can then explore this subspace with a set of sliders. In this manner, artistic skills or knowledge of advanced modeling tools are not needed from the user.

Specifically, the user sets preferences for a shape using pre-defined keywords and then our system generates possible shape variations that maintain the user's preferences. Thus, our proposed method allows the user to explore a variety of 3D shapes generated according to their choices.

Designing 3D models interactively with the help of semantic keywords has been researched before for content creation in limited settings. The method of Chaudhuri et al. [19] reuses parts from a database of example shapes to create new models. Thus, the shapes produced have fixed geometric variations because of the limited number of example shapes. Yumer et al. [79] create 3D shapes by deforming a given shape according to a mapping from semantic keywords to geometry. The mapping is derived with a crowd-sourcing approach. However, the variation of the generated shapes is also limited since a shape is only deformed but not modified in any other manner. Although these works create visually appealing results, the variability that can be created is limited, and none of the methods explores deep networks. On the other hand, we take advantage of the computational power of deep networks for creating 3D shapes.

Previously, there have been investigations into the best shape representation to be used in conjunction with deep networks, such as voxel grids [15, 73], octrees [69], atlases of parameterizations [30], signed distance functions [51], point clouds [1], and graphs of parts [46, 49]. However, less effort has been spent on how to effectively control these generative models for enabling users to create a shape according to a pre-defined set of goals or an intended design.

The goal of our work is to allow users to control the latent spaces learned by deep neural models for generating meaningful shapes. From the users' perspective, we interpret the shape spaces using semantic keywords which is easier to understand compared to other 3D modeling methodologies where low level details need to be explicitly defined, as implemented by softwares such as Maya or Blender.

1.3 Overview of the proposed method

We introduce a method to facilitate the navigation of the shape spaces learned with deep networks by incorporating semantic information into the process. Our key idea is to let the user explore the shape space according to high-level semantic keywords,

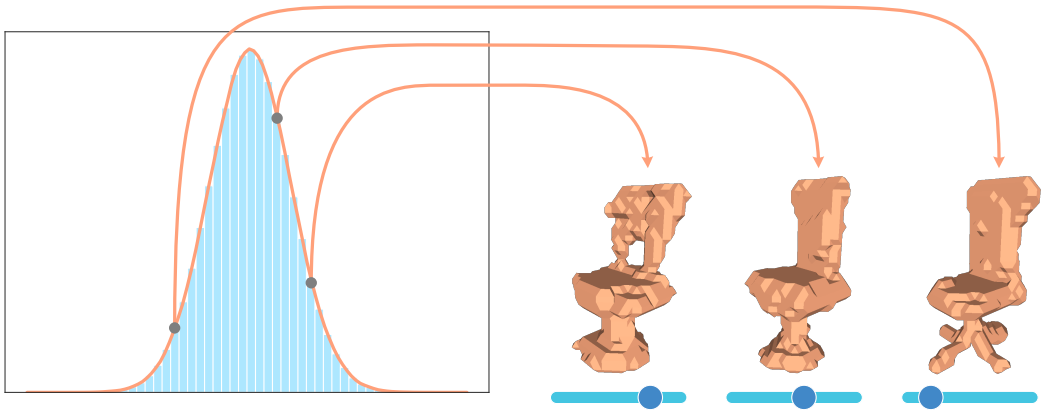


Figure 1.1: Our approach for user-guided shape synthesis: after specifying keywords that constrain the attributes of the synthesized shapes ("straight back" and "short legs" in this example), the user can manipulate a set of sliders to explore the subspace of shapes with these attributes. The subspace is modeled as a set of distributions, one for each dimension of a latent representation of the shapes. The example shows three shapes generated by navigating through one specific dimension of the shape space.

where the keywords characterize the attributes that the intended shape should possess, e.g., "four legs" and "two arms" for a chair. Specifically, given the user input, we map the attributes to a subspace of the latent space that captures the shapes possessing the attributes. The user then explores only the subspace to search for shapes that satisfy the intended design (Figure 1.1), similarly to using a parametric shape model. To enable an efficient exploration, we map the keywords to the distributions of the latent spaces, such that the user is only required to explore dimensions with significant variance (such as the distribution shown in Figure 1.1). Thus, our method enables an exploratory modeling approach, where the user can model shapes at a high-level and is not required to possess artistic skills as required by sketching or partial modeling approaches.

To enable this exploration-based approach, we introduce a learning-based method composed of a label regression neural network (LRN) coupled with a shape synthesis neural network (SSN). The LRN is a feed-forward neural network composed of three hidden layers, one input, and two output layers. The label regression network takes the user-provided keywords and maps them to the corresponding subspace of the latent space. The network is trained with the latent representation of a collection of shapes paired with semantic keywords. The SSN is an autoencoder which consists of

two symmetrical networks named encoder and decoder. Whereas the encoder encodes a representation of a shape into a latent vector, the decoder reconstructs a shape from a latent vector. For representing a shape in our method, we place a voxel grid over the triangle mesh representing the shape and discretize the mesh into a set of 32^3 or 64^3 voxels. We use a volumetric representation of shapes as it is easier for neural networks to perform convolutions over this type of representation. Although we test our approach with a voxel-based autoencoder network for synthesizing shapes, our method is general enough that it can be applied to other types of encoder-decoder networks, such as AE-GANs or VAE-GANs. The latent representations provided by the encoder for a set of shapes are used for training the LRN.

Since our method is based on learning, the main requirement for training our networks is a set of training shapes, where each shape is labeled with semantic keywords.

We apply our method to a collection of chairs and lamps and show with a qualitative evaluation that this solution enables users to explore the latent spaces for generating a variety of 3D shapes. Figure 1.2 shows some example shapes that are produced following our method. We also present an analysis of the method to demonstrate that the learned models are sound and that the method can be used in an interactive setting.

1.4 Contributions of this work

We summarize the contributions of this research as follows:

- **Conceptual:** we introduce a method to facilitate user-guided synthesis of shapes with deep networks, using a combination of semantic keyword selection and exploration. In this method, the user selects a set of keywords that constrains the exploration to a subspace of shapes with the required attributes.

- **Data:** for our experimental evaluation, we label the shapes of two datasets with semantic keywords. The keywords reflect a variety of features of the shapes. We use the keywords and dataset of shapes to train our method.

- **Experimental validation:** we perform experiments to demonstrate that we can combine existing network architectures, such as a shape synthesis model based on an autoencoder and a feed-forward network, into a framework that

Figure 1.2: Examples of chairs and lamps generated with our method.

allows users to create shapes according to their preferences. We also analyze other aspects of our method, such as execution time, to conclude that our method allows users to generate shapes interactively.

1.5 Organization of the thesis

The remainder of this thesis is organized as follows:

Chapter 2 provides background information about deep neural networks and how they relate to our work. We also discuss synthesis problems and recent developments addressing this problem.

Chapter 3 introduces our method, explaining how two networks are integrated together to synthesize shapes according to user input. We describe the input and output of the networks, loss functions, and optimizers.

Chapter 4 discusses our dataset and presents results obtained with our method. We visualize a variety of shapes produced by our method and also evaluate different aspects of our neural networks.

Chapter 5 concludes the thesis by discussing limitations of our method and directions for future work.

Chapter 2

Background and Related Work

Since our method use deep learning models for shape generation, in Section 2.1, we first provide background on deep neural networks, discussing their most important components and a variety of network architectures. Next, in Section 2.2, we discuss related work addressing the problem of shape synthesis, including classic approaches and methods based on deep learning.

2.1 Deep learning

As illustrated in Figure 2.1, deep learning (DL) is a subset of both artificial intelligence (AI) and machine learning (ML), where neural networks with multiple layers are trained to perform inference. Although the concept of deep learning was introduced a long time ago [23], the current abundance of data and computational capabilities of GPUs make deep learning models more robust and scalable than models based on traditional machine learning methods [3]. Especially, the more recent development of networks such as *AlexNet* [42], *ResNet* [32], and *VGGNet* [60] brought revolutionary advances to the field of machine learning. Deep learning has evolved to a level where applications like image classification [42] and AI for playing games produce results that are comparable to humans in terms of accuracy and speed [20]. Most of the machine learning problems can be divided into supervised learning, unsupervised learning, semi-supervised learning, or reinforcement learning. Depending on the nature of available data and problem to be solved, a particular learning approach is chosen.

A deep learning model commonly consists of a set of components such as a set of

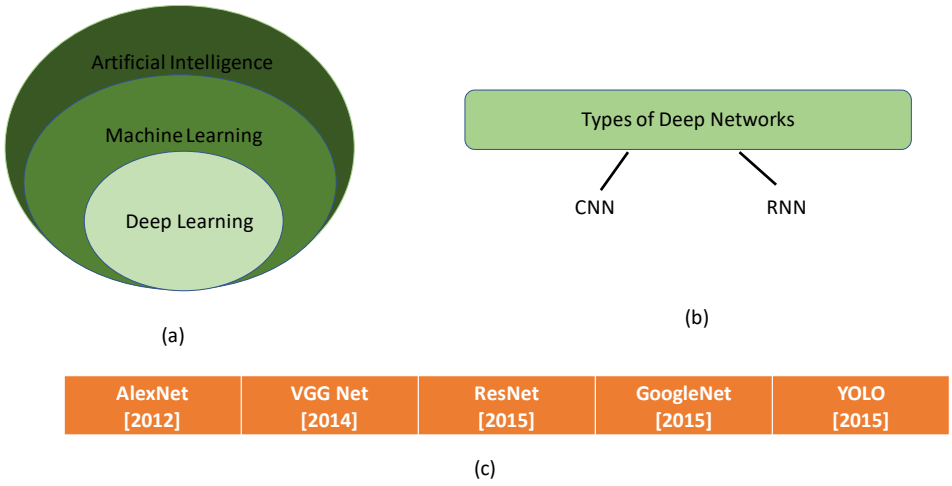


Figure 2.1: Overview of deep learning. (a) Taxonomy relating AI, ML, and DL. (b) Two types of deep networks. (c) Example models that are considered the state of the art of deep networks.

layers through which all the mathematical calculations are carried out from the input to the output, an optimizer and a loss function. The layers are typically divided into: input, output and hidden layers. The input layer receives and pre-processes an input and then forwards its result to multiple hidden layers that can be of a variety of types: convolution layers, pooling layers, fully-connected layers, batch normalization layers, etc. At the end of the network, an output layer produces the result of the network. To drive the output from a source layer to a target layer, often an activation function is applied to the output of the source layer. The organization of layers is typically called the *architecture* of the network. Deep networks are commonly classified into different types according to their architecture. Common classifications include: Convolutional Neural Networks (CNNs) and Recurrent Neural Network (RNNs) (Figure 2.1). Both CNNs and RNNs have a common neural network structure except for the connection between neurons inside of the hidden layers. In general, CNNs perform a convolution over the input to extract important features locally through filtering. In contrast, RNNs have a recurrent connection inside their hidden layers to consider a previous state of the network in the current computation. Hence, CNNs have less parameters in each layer compared to RNNs. Depending on the architecture, RNNs make use

of long-short-term memories (LSTM) or gated recurrent units (GRU). Both CNNs and RNNs can handle 1D data (signals or sequences), 2D data (images), and 3D data (video or volumetric images). The type of network is chosen depending on the problem to be solved.

In the next section, we briefly describe the basic architecture of a CNN and parameters needed to train a CNN model, since our work is employs a CNN model. Following this explanation, we discuss a few more complex deep network architectures specific to the application of synthesizing objects.

2.1.1 Convolutional Neural Networks (CNNs)

CNNs are a stack of layers that perform operations such as convolution and pooling. The layers can be seen as a set of filters, each of which is processing a specific region of the input independently of the other layers. As illustrated in Figure 2.2, the processing of an input image goes through different hidden layers and produces an output based on the activation of hyperparameters set for each specific layer. The hidden layers are often of one of three main types: *convolutional layer*, *pooling layer*, or *fully-connected layer*. In Figure 2.2, each of these three layers are stacked twice on top of each other to produce the output.

Although the concept of CNN was already introduced in 1988 [27], due to the limited computational power of hardware at that time, the CNN network architecture could not be easily implemented. In contrast, today, because of the advancement in parallel computation brought by GPUs, the multiple layers of a CNN can be executed in parallel. Hence, applications like object classification, detection, reconstruction, segmentation of scenes and instances can utilize CNN architectures to produce good results. For example, Krizhevsky et al. [42] use a CNN for the classification of 1.2 million images of 1,000 different classes in ImageNet. Their CNN architecture consists of five convolutional layers, each of these followed by max-pooling layers and three fully-connected layers with a softmax layer applied at the end. Most interestingly, this architecture consisting of 60 million parameters when ran on a GPU sped up the processing up to 10 percent compared to using a CPU [42]. Farabet et al. [26] introduce a multi-scale convolutional network for scene labeling and segmentation of an image, which contains multiple copies of a single network that encodes information from raw pixels separately and then combines the information to represent shape, texture, and contextual features of the image. For real time autonomous driving, CNNs

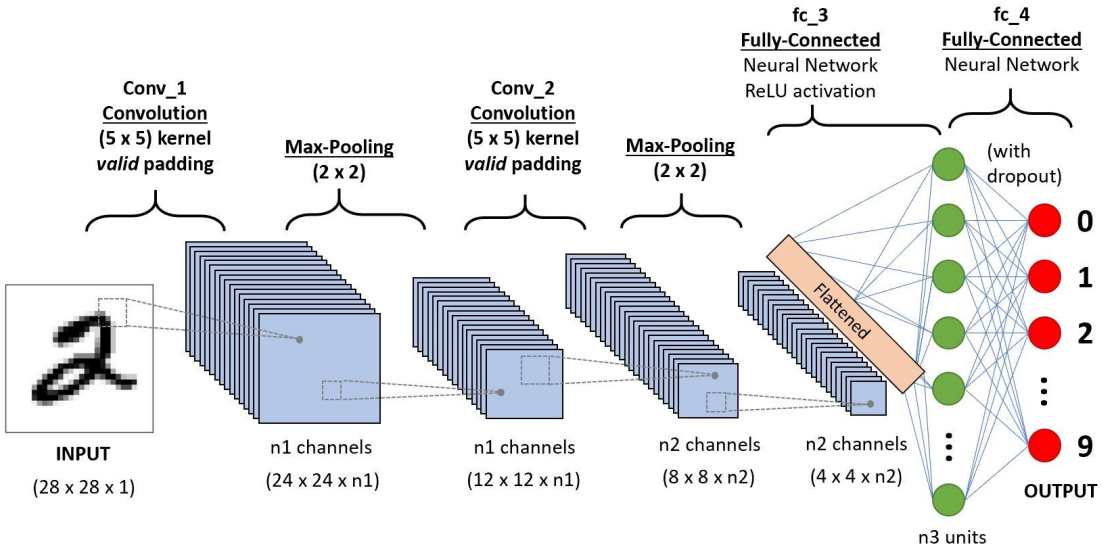


Figure 2.2: A convolutional neural network architecture for classifying images of 2D handwritten digits. Image courtesy of Sumit et al. [56].

have shown significant performance on analyzing images related to traffic signals and vehicle movement [24, 40].

For processing 3D models, CNNs also outperform other architectures in terms of accuracy and simplicity. 3DWINN [34], SAGNet [74] and the 3D shape descriptor network [76] are some recent developments on 3D object analysis using CNNs. We discuss 3D object synthesis with deep networks in more detail in Section 2.2.3.

Next, we briefly discuss the basic functionality of the different layers of a CNN.

Convolution layer

The convolution layer extracts features from an input with a convolution operation. It consists of a set of learnable filters that slide through the input image or volume. Specifically, the filters compute the dot product between a set of learnable weights and a connected region of the input, also adding a predefined bias value to the result. As

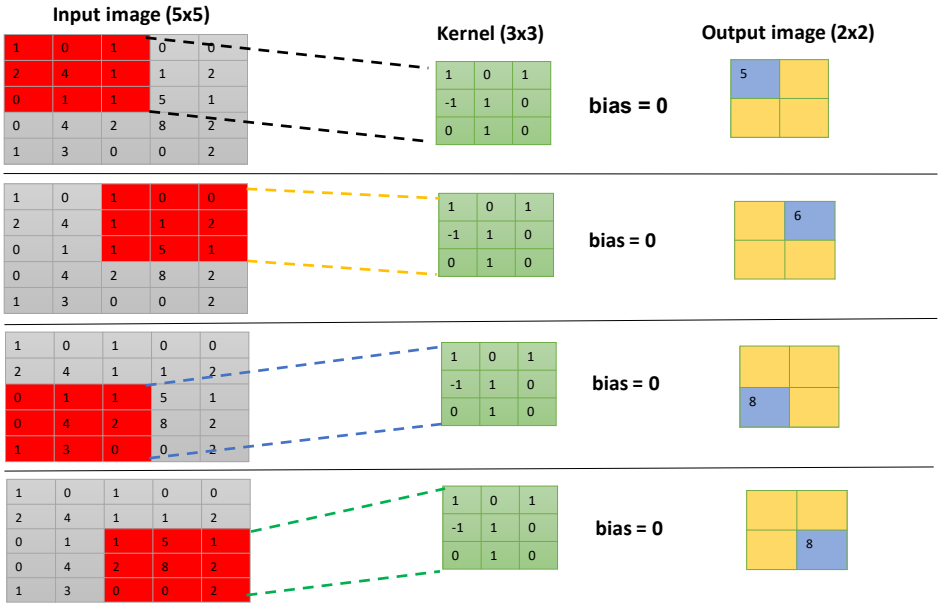


Figure 2.3: Example of the calculation inside of a typical convolutional layer. In this example, the input image is $[5 \times 5]$, the kernel size is $[3 \times 3]$, and the stride is 2, whereas the bias is 0.

the model is trained, the filters learn to extract features such as edges of the input, and different convolutional layers of the same network learn to consider different features of the input. Each convolution layer is defined by hyperparameters that compute the convolution. First of all, a *filter* of a given *size* is defined. The amount of sliding to perform when processing an input region is decided by the *stride* hyperparameter. Another important hyperparameter is *padding*, which defines the padding added to the input image or volume around the border. This hyperparameters define the output structure of convolutional layer which can be represented by the formula:

$$C_{out} = \frac{W - F + 2P}{S} + 1, \tag{2.1}$$

where W represents the size of the input, F represents the filter size, P represents the amount of padding, and S represents the stride [3]. After applying the filter to each pixel of a 2D image or voxel of a 3D volume, an activation map is produced that gives the responses of the filter at every spatial position.

A 2D image given as an input to a convolution layer typically consists of raw

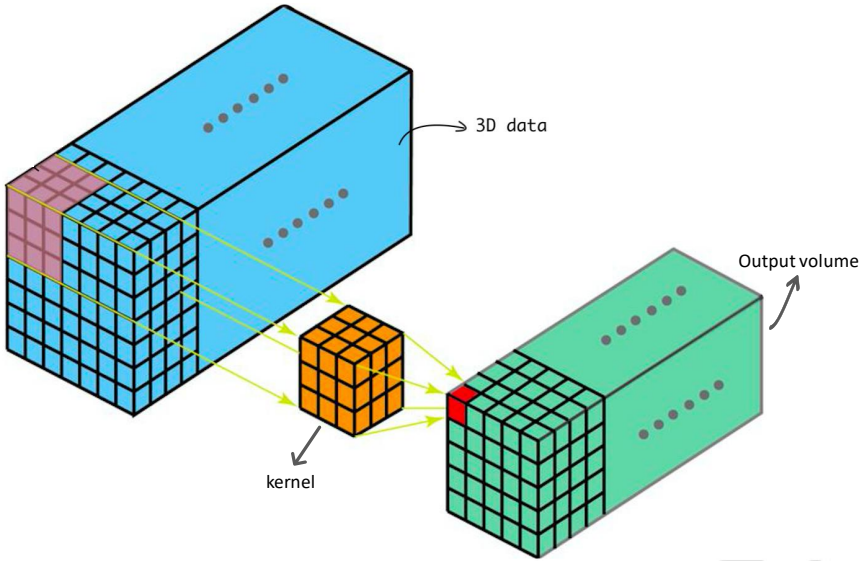


Figure 2.4: Example of convolution for a 3D volume, where the kernel in orange is applied to the region in light red, to produce the output value highlighted in dark red. Adapted from a figure of Shiva et al. [65].

pixel values which represent a color, as shown in Figure 2.2. In this image, the single channel encodes a grayscale pixel, whereas images with 3 channels encode RGB colors. In Figure 2.3, we see an example of a convolutional layer in operation for a 2D image. For simplicity, we consider only the height and width of the $[5 \times 5]$ image. The filter is of size $[3 \times 3]$, while padding and bias are set to 0. The stride is set to 2. According to Equation 2.1, the size of the output image produced with this arrangement is (5 - 3 + 2 * 0)/2 + 1 = 2. In the first row, a $[3 \times 3]$ window of a kernel slides through the input image and calculates the sum of the dot product between each local input region (highlighted in red) and weights (highlighted in green). The result of this calculation is shown in the last column (highlighted in blue). As the stride is 2, the kernel window slides two columns and two rows and performs the convolution. At the end, the output image size is reduced to $[2 \times 2]$. For 3D volumes, the convolution operation follows the same procedure but with an extra dimension. Figure 2.4 illustrates the convolution of a 3D volume.

Pooling layer

Similarly to a convolutional layer, a pooling layer also downsamples the input data, reducing the dimensionality of the image. However, as opposed to the convolutional

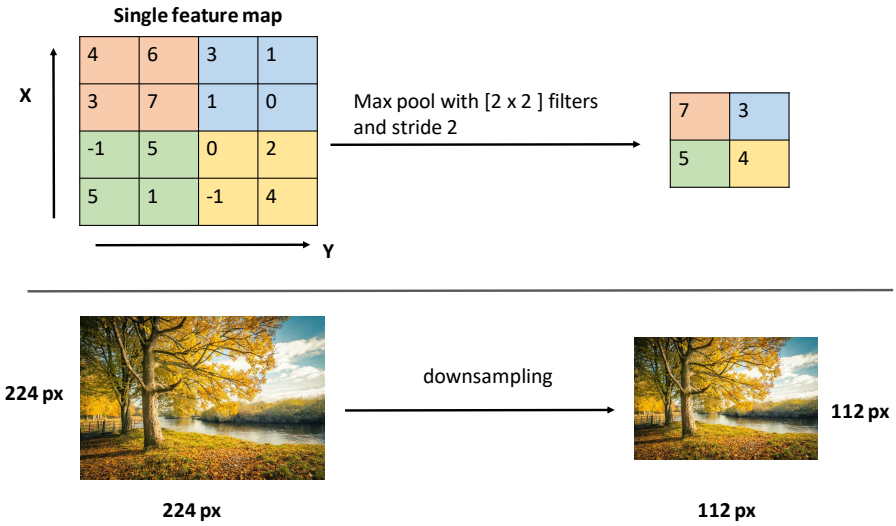


Figure 2.5: A pooling layer reduces the dimensionality of the input from each feature map. In this example, a max pooling with $[2 \times 2]$ filter is added for downsampling.

layer, it performs the downsampling with a fixed function rather than based on a filter and its learned parameters. Pooling layers are basically used to reduce the number of training parameters, also reducing the spatial dimension of the input.

There are mainly three different types of pooling: *max pooling*, *average pooling*, and *sum pooling*. Max pooling takes the maximum value for each patch of the feature map; average pooling takes the average value; and sum pooling takes the sum of the feature map. Figure 2.5 illustrates max pooling being applied to an input image. For each patch of the input feature map (highlighted with different colors), only the maximum value is chosen for downsampling the feature map.

Fully-connected layer

In a fully connected layer, all the neurons in the output layer are connected to each input neuron. Thus, a fully-connected layer considers information from each neuron of the previous layer, but does not have any self-connection. Whereas, convolutional and pooling layers extract features from the input, fully connected layers produce final classification or regression values by considering the extracted features. In general, fully-connected layers perform the same dot product operation between learned

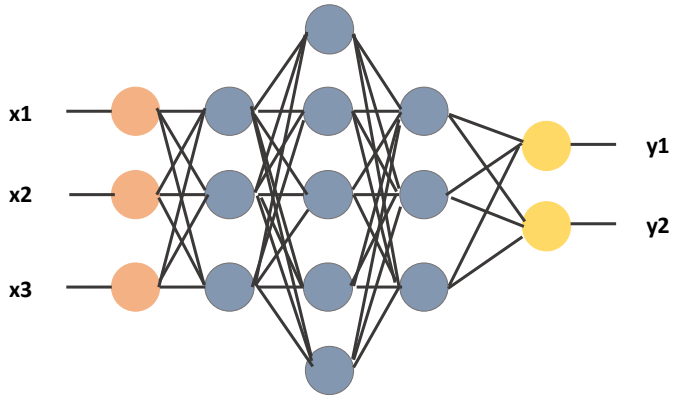


Figure 2.6: A fully connected network with three hidden layers.

weights and the input feature map. Figure 2.6 illustrates the connectivity of a fully-connected layer with three hidden layers, three input and two output neurons.

Activation function

After calculating a weighted sum of the input and adding a bias value, a neuron should output the result or not depending on an activation function. More precisely, an activation function performs a thresholding of the value calculated by the neuron. The value of the activation function might range from *-inf* to *+inf*, although the activation function can also be used to normalize the output value to a specific range. But for backpropagation, it is necessary for the activation functions to be finite, otherwise the gradient of the loss function might approach to zero and make the network very hard to train. Depending on the problem and model architecture, different activation functions can be used, such as *ReLu*, *Leaky ReLu*, *tanh*, and *sigmoid*. Figure 2.7 illustrates these four activation functions with their output ranges.

The *rectified linear unit (ReLU) activation function* is generally used for introducing non-linearity in the network, defined as $\text{ReLU}(x) = \max(0, x)$. It only retains the positive part of its input and turns negative inputs into zero. ReLU is computationally less expensive compared to other activation functions and its non-saturating characteristic is useful for avoiding vanishing gradients while backpropagating [77].

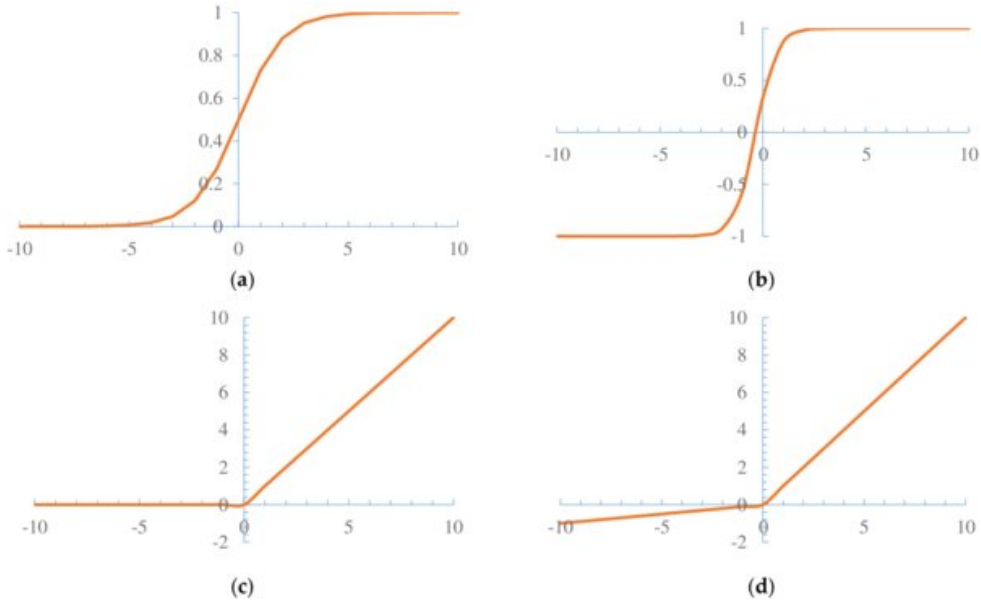


Figure 2.7: Different activation functions: (a) Sigmoid function, (b) Tanh function, (c) ReLu function, and (d) Leaky ReLu function. Image courtesy of Yang et al. [78].

Although ReLU avoids vanishing gradients, if the input becomes zero or negative, the gradient of the function also becomes zero. Thus, the neuron will not fire during backpropagation and will stop responding to error or input variations. This is known as the *dying ReLU* problem. To address this issue, variations of ReLU have been proposed, such as the *leaky ReLU* or *parametric ReLU* [77]. The *leaky ReLU* introduces a small negative slope with an alpha parameter to the *ReLu* function, to sustain and keep the weight updates alive during backpropagation. The *Parametric ReLU* is a type of *leaky ReLU* where the alpha parameter is gradually learned by the network while propagating the updates through the layers of the network [50].

The *sigmoid activation function* is also non-linear and its output ranges between [0, 1]. It is defined as:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}}. \tag{2.2}$$

This function is a little more computationally expensive than ReLU, as it is not sparse and might cause most of the neurons to fire. But it gives more accurate predictions because of this non-sparse behaviour of the function. Although sigmoid activation gives better results in accuracy, for very high or low values of the input, there is almost no change to the prediction. This causes a vanishing gradient, preventing the

network from learning further during backpropagation.

Tanh is another activation function, similar to sigmoid, but ranges between $[-1, 1]$. It is sometimes called *scaled sigmoid* and is defined as:

$$Tanh(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{2.3}$$

Tanh is also non-linear like ReLU and sigmoid and gives better performance in training for multi-layer neural network [37]. But the vanishing gradient problem also persists in this activation function if the input value is too far from the range $[-1, 1]$.

Loss function and optimizer

Supervised deep learning models are typically intended for predicting an output based on an input, and are learned from training data which is also known as ground-truth. To calculate the error between the prediction of the network and the ground-truth, the *loss function* is used. The loss function $L(\hat{y}, y)$ compares each predicted output $\hat{y}$ to the ground-truth y, calculating an error based on the differences.

Mean square error (MSE) is one of the most widely used loss functions which computes the square differences between prediction and ground truth and then averages them for the whole dataset:

$$\text{MSE}(\hat{y}, y) = \frac{1}{M} \sum_{i=1}^{M} ||\hat{y}_i - y_i|\,|^2, \tag{2.4}$$

where M denotes the total number of training samples.

For regression or classification problems, if the output layer is a neuron with a linear activation function, then the MSE can be applied.

Cross-entropy or log loss is another possible loss function, which measures the divergence between two probability distributions. It is defined as:

$$\text{H}(P, Q) = -\sum_{i=1}^{M} P(x) \log Q(x), \tag{2.5}$$

where P denotes the distribution of true labels and Q is the probability distribution of the predictions from the model. Usually, cross-entropy loss is applied to output neurons that use sigmoid or softmax non-linear activation functions.

To minimize the value of the loss function, the weights of the model are updated

during backpropagation by training the model with an *optimizer*. An optimizer starts with random model parameters. After calculating the error for one epoch, the optimizer updates the model parameters so that the loss is reduced. *Gradient descent, Adam, RMSprop,* and *stochastic gradient descent* are some of the most used optimizers for learning models.

Training, validation, and test data

The optimizer updates the weights and biases of a deep learning model by learning from example data which is called *training data*.

On the other hand, *validation data* refers to a subset of the training data which is not used for training but for testing the accuracy of the model and possibly adjusting the hyperparameters of the model. According to Ripley [54], validation sets are used to tune the parameters of a machine learning model. It is recommended to keep the validation and test set separated from the training set, though the purpose of both validation and test data is to evaluate the performance of a model [43, 55]. *K-fold cross validation* is one of the most popular methods for validation of models. In this method, the training data is split into k *folds* or subsets, and $k-1$ folds are used for training the model while the remaining fold is used for evaluation.

To verify the accuracy of prediction of a trained model, a set of unseen data is used which is known as *test data*. Test data is given to the model only once at the end of training and used to evaluate the performance of a model. Typically, validation data is generated by splitting training data but test data is recommended not be a part of training data. Hence, it might be considered as the gold standard used to evaluate the model.

Overfitting and underfitting

While learning the parameters of a model, overfitting and underfitting are two problems that may make a model not perform well on unseen data. Overfitting occurs when a model learns the detail and noise of the training data to such extent that it negatively affects the performance of the model on testing data. In contrast, underfitting happens when the model cannot learn from the training data and hence, cannot generalize to new data. To avoid both overfitting and underfitting, the number of training epochs, learning rate, and number of architecture layers must be chosen carefully. Moreover, specific layers such as *batch normalization layers* or *pooling layers*

can help to avoid these problems by introducing randomness while training. Another solution to address overfitting is to use cross validation as discussed in the previous subsection, which allows to evaluate the generalization capabilities of the model.

2.1.2 Generative Models

Generative models learn a distribution in an unsupervised manner, which has large applicability in synthesis applications. Recently, with the developments in deep learning networks, generative models have become popular for applications such as data synthesis (2D, 3D, audio, video, etc.), especially to create fake data. Although there are diverse types of generative models, in the following sections, we are going to focus on *generative adversarial networks (GANs)*, *variational autoencoders (VAEs)*, and *autoencoders (AEs)*.

Generative Adversarial Networks

Goodfellow proposed a model named *generative adversarial network (GAN)* in 2014 [29] which consists of two distinct networks named *generator* and *discriminator*. These two networks learn in an adversarial manner by competing with each other in the training phase. The generator G models the distribution of data and generates new data similar to the training data. The discriminator D tries to decide if the data given as input comes from the training set or the generator [29]. Specifically, the generator learns a distribution $\mathbb{P}_g$ for given data $\mathbf{x}$, which is used to map a prior noise vector $\mathbb{P}_z(\mathbf{z})$ into the learned space $G(\mathbf{z}, \theta_g)$, according to parameters θ_g. The discriminator D gives a single value as an output to represent whether the data is real (training data) or fake (data produced by the generator). For training, the convergence of the losses of both networks is important. Hence, the discriminator D is trained to maximize the probability of predicting correct labels for each sample. On the other hand, the generator G is trained to minimize the probability that the discriminator predicts the correct labels. As shown in Equation 2.6, the GAN follows a min-max strategy in a two-player game and its goal can be described with an objective function $O(D, G)$:

$$\min_G \max_D O(D, G) = E_{x \sim q_{data}(x)}[logD(x)] + E_{z \sim p(z)}[log(1 - D(G(z)))], \tag{2.6}$$

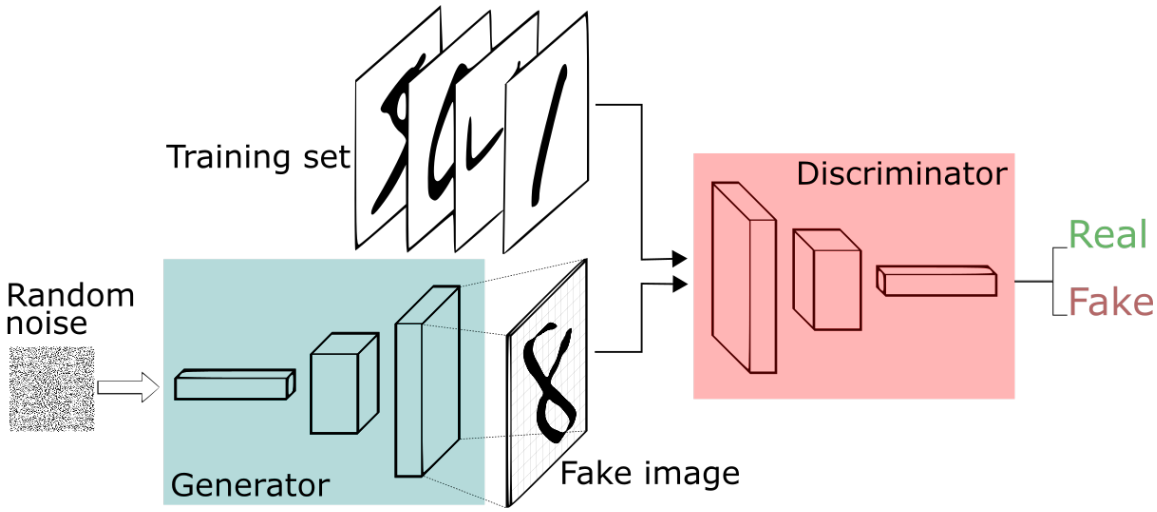


Figure 2.8: The architecture of a Generative Adversarial Network (GAN). Image courtesy of Silva et al. [59].

.

where $D(x)$ is the discriminator's estimate that the data instance x is real and not generated data, and $G(z)$ is the generator's output produced when given a noise vector z. Thus, $D(G(z))$ is the discriminator's estimate of the probability that a fake instance is real. The min and max on the left-hand side of the equation represent the game where the generator tries to minimize the above loss function while the discriminator tries to maximize it.

In this setup, both networks compete with each other, thus helping each other to perform better at their own task. In the beginning, because the learning of the generator G is slow, the discriminator's loss may converge quickly by easily rejecting samples from G. As the learning goes on, the generator G starts to produce more realistic samples, and the losses of both networks become more stable towards convergence.

Figure 2.8 shows a standard *GAN* architecture where the two networks are highlighted with colors. As seen in the diagram, the generator produces fake images that are fed to the discriminator for prediction as true or fake. The standard architecture of a GAN consists of a convolutional layer and dropout layer. The Generator uses *ReLU* and *sigmoid* activation functions, whereas the discriminator network uses *maxout* activation functions [29].

There are variations among the architecture of GANs for different applications.

Table 2.1: Some variations of GANs and their applications in different domains.

GAN variation	**Domain**	**Applications**
CycleGAN SRGAN DiscoGAN AlphaGAN BicycleGAN Pix2pix Age-cGAN StackGAN IcGan RenderGAN	Images	Image-to-image translation, object detection, image synthesis, image transformation (lower to higher resolution), image inpainting, face aging, image blending, photo editing, texture synthesis, text-to-image translation, image/scene segmentation
SeqGAN MidiNet MuseGAN MaskGAN WaveGAN DVD-GAN VoiceGAN	Sequential data	Music generation, text generation, video generation, audio generation, text-to-video generation, voice impersonation
3DGAN MP-GAN Ismo-GAN 3D-RecGAN	3D shapes/objects	3D object generation and reconstruction, shape interpolation, 3D scene reconstruction, 3D modeling

For synthesizing realistic images based on the ImageNet dataset, *BigGAN* is the state-of-the art for generating high-resolution images [14]. This work modifies the standard GAN architecture by using a hinge loss, class information with generator G, and projection for the discriminator network D. The initializer of the generator G also performs orthogonal regularization, which is commonly called the *truncation trick*. The result of the modified model is clearly visible in the paper, producing a variety of images in a particular domain [14]. Another interesting variation of GANs is *CycleGAN* for cross-domain transfer [80], which transform images from one domain to another. Applications like converting photos into paintings, changing the season of an image (converting summer pictures into winter), or replacing one object with another in an image are possible with *CycleGAN*. Other applications include creating super resolution images from lower resolution with *SRGAN* [45], transferring style and content between images with *StyleGAN* [38], and generating aging faces with *Age-cGAN* [5]. In general, GANs perform well and are considered the state-of-the-art for generative models. Table 2.1 lists some of the popular GAN architectures for different applications.

Autoencoder

While the noise vector input to a GAN is randomized, an *autoencoder* allows the possibility of using a noise vector methodically derived from a latent space. An autoencoder consists of two symmetric neural networks, named *encoder* and *decoder*. The primary aim of the encoder is to reduce the dimensionality of the input, generating a latent vector z, whereas the decoder aims to reconstruct the input from the latent vector. The latent vector is also called the bottleneck of the autoencoder. In this context, an autoencoder can be compared with *principal component analysis (PCA)*, if the encoder and decoder have only one layer extracting a linear structure from the data.

Figure 2.9 shows the symmetric structure of an autoencoder. At each step of the learning, the decoder generates the reconstructed output which is then compared with the initial data to calculate the *loss* as *mean square error* following Equation 2.4. This error then backpropagates throughout the network to update the weights. As the autoencoder maps any data into one single point in the latent space, two major problems of this model are:

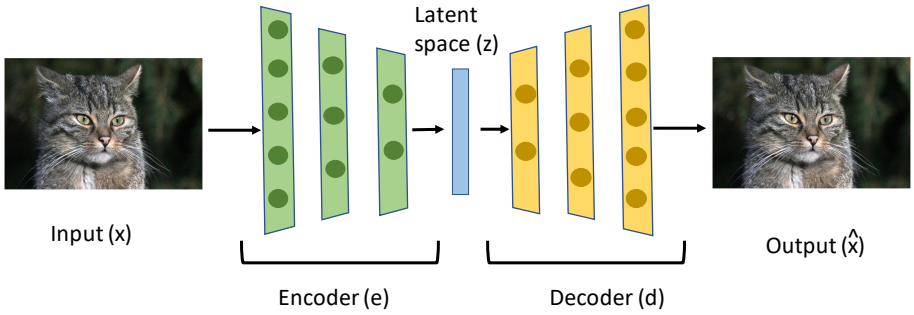


Figure 2.9: Architecture of an autoencoder network.

- The regularity of the latent space is very hard to control.
- It is difficult to choose the optimal structure of the latent space so that most of the important features are encoded in the latent space.

Because of these reasons, parameters like latent space size and depth of convolution and deconvolution inside symmetric networks must be chosen carefully to provide optimal results.

Intuitively, an autoencoder learns to reduce the dimensionality of the data by learning the important features of the data. Hence, several applications and variations of autoencoders can be found. Two of the most common variations of autoencoders are the *sparse autoencoder* and the *denoising autoencoder*. Sparse autoencoders are used to learn features for tasks such as classification. This type of autoencoder is regularized to find unique statistical features of the dataset. For example, a method for learning features from the surface of electromyography signals based on a sparse autoencoder is described by Ibrahim et al. [35]. In contrast, denoising autoencoders locate important features from the data by removing extra noise added to the initial data, hence, learning to generate a robust representation of the data [66].

For segmenting images of road scenes of twelve different classes, the VGG16 network for object classification has been proposed [7], based on an encoder-decoder architecture. Autoencoders have also been proposed for applications such as image

restoration (including denoising and super-resolution) [48], and style transfer between different image domains [71]. Zhu et al. [81] present the first hybrid method using an *autoencoder* for 3D shape retrieval. An autoencoder is used as a global descriptor of shape similarity and combined with BoF-SIFT as a local descriptor. Each 3D shape is projected onto several 2D images which are then given to the encoder in order to create low-dimensional vectors. Finally, the *Hausdorff Distance* is calculated between each pair of low-dimensional vectors extracted from the two shapes, which is aggregated to represents the global similarity of the 3D shapes. Recently, Zhang et al. proposed a network named IM-Net where the decoder network takes as input a 3D point (x, y, z) alongside a feature vector and returns a binary value representing whether the point is inside or outside of a shape [21]. This work shows improved visual results for generating shapes, single-view 3D shape reconstruction, and shape encoding.

Variational autoencoder

Before we discuss the architecture of a variational autoencoder, we would like to discuss what a latent space means for generative models. We consider a dataset $D = \{x_1, x_2, ..., x_n\}$ is generated through a random process, involving a continuous random variable z. Then the joint probability of an input and a random variable z for this dataset can be defined as:

$$p_\theta(x, z) = p_\theta(x|z)p_\theta(z), \tag{2.7}$$

where x is a discrete or continuous input, $z \in R^k$ is a latent variable and θ represents the parametric distribution for both $p_\theta(x|z)$ and $p_\theta(z)$ [39]. Here, x can be any data such as an image or volume and z represents a collection of factors that explain the features of the input. Within this setup, it is interesting to explore what the latent variables z are, given an observation x. Namely, we would like to know $p_\theta(z|x)$, which can be derived from the posterior probability and marginal probability. However, computing these two probabilities requires computing the integral over equation 2.7. Given the number of observations, D can be infinite. Thus, computing the integral is an intractable problem. The variational autoencoder architecture addresses this problem and provides solution for the following inferences:

- Given an input x, what can be the latent factors z or posterior inference over

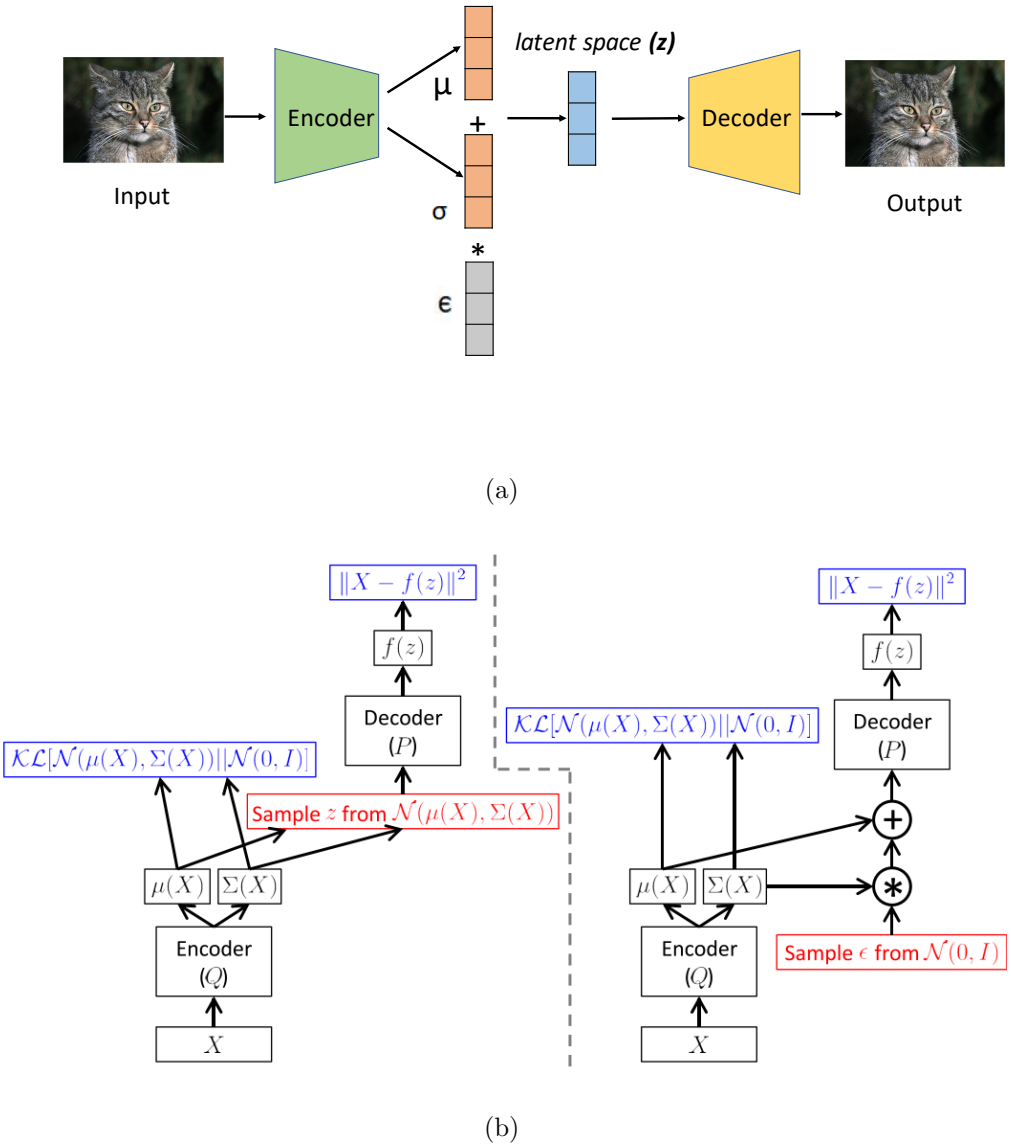
z.

- Given an under-constructed input x, how can we make a marginal inference over x.
- Learning parameters θ of p for equation 2.7.

A *variational autoencoder* provides an approach to efficiently solve these inferences following an optimized process of variational inference. Variational inference is a technique for solving an optimization problem over a class of distributions that has a closed-form expression. For example, using a distribution Q in order to find a $q \in Q$ that is most similar to p. Then, q can be used as an approximate solution of p. Both *autoencoders* and *variational autoencoders* aim to reconstruct a given input from a lower-dimensional latent space [39]. However, unlike an autoencoder, the encoder of a variational autoencoder outputs a distribution (Q as mentioned before) rather than a vector representation for the latent space.

It is difficult to regularize the latent space of an autoencoder as it neither depends on the distribution of the data nor the dimensionality of z. As a result, there is a high probability of *overfitting* the model, implying that some of the points on z would give accurate results, such as points taken from the training data, but other points would give meaningless outputs after reconstructed by the decoder. Thus, to avoid this problem and introduce regularity into the latent space, a variational autoencoder models the latent space as a distribution of input data, which helps to create new data without overfitting the model. In other words, the encoder network of a variational autoencoder infers the parameters of the posterior probability $q_\phi(z|x)$ so that the decoder can decode the prior $p_\theta(x|z)$ as well as new contents over input x.

Figure 2.10 shows the architecture of a variational autoencoder. Here, ϵ is a Gaussian vector and needed to determine the derivative of the stochastic variable z. The latent space (z) is a stochastic variable from which samples are randomly chosen to feed the decoder network. However, calculating derivatives of a random variable is not possible. In order to backpropagate through the network, ϵ is multiplied with σ and then added with μ, where $\epsilon \sim N(0, 1)$ [25]. In this way, both μ and σ become trainable and the only stochastic part of the network is ϵ, which does not need to be backpropagated as it is always coming from a standard normal Gaussian. This is called the *reparameterization trick* [39] of VAEs, that makes the network trainable.



(a)

(b)

Figure 2.10: Example architecture of a Variational Autoencoder. (a) The encoder network has two output layers, representing the μ and σ parameters of a distribution. The ϵ is a standard normal Gaussian vector which is multiplied with σ for backpropagating the error through the network. (b) Reparameterization of a VAE. Image courtesy of Doersch et al. [25].

The *loss function* of a VAE combines a negative log-likelihood with a regularizer, where the first one is the reconstruction loss and the second one calculates the

divergence of two distributions over the latent space:

$$l_i(\theta, \phi) = -\mathbb{E}_{z \sim q_\phi(z|x_i)}[\log p_\theta(x_i|z)] + \mathbb{KL}(q_\phi(z|x_i)||p(z)). \tag{2.8}$$

The expectation of the reconstruction loss is measured with respect to the encoder's distribution over the latent space. If the reconstruction given by the decoder is inaccurate, then the negative log-likelihood incurs a large cost on the overall function. The second term of the loss function is the Kullback-Leibler divergence between the encoder's distribution $q_\phi(z|x_i)$ and $p(z)$. This divergence estimates how much information is lost when using q to represent p. As it is specified that $p \sim N(0, 1)$, if the encoder's output representation z deviates too much from the standard normal distribution, then the KL loss will be higher. This regularizer is needed to keep z diverse enough in Euclidean space so that meaningful samples can be drawn for the reconstruction.

VAEs have a variety of applications for both 2D image and 3D shape processing. As a semi-supervised model, a VAE has advantages for synthesizing shapes from existing samples, and generating or exploring the shape space for new samples. Tan et al. [64] proposed a framework (mesh VAE) to interpolate shapes by deforming meshes with a conditioned latent space of a variational autoencoder. They modified the prior distribution with diagonal matrices which allows the user to select important features of models and explore the latent space for finding new models [64]. Another work that combines geometry and structural information into a single latent vector to synthesize new shapes use the GRU based encoder and decoder of VAEs [74]. For applications such as image inpainting or data augmentation, VAEs combined with GANs and a classification network (CVAE-GAN) produces remarkable results with inception score of 97% on various images of birds, faces and flowers [8].

Our work also involves synthesis of new shapes through the exploration of a latent space provided by an encoder-decoder architecture. In order to do this, we use an autoencoder (AE) model as a shape synthesis network. We also experimented with a variational autoencoder (VAE) and an AE-GAN. However, when comparing the AE to the VAE and AE-GAN, we found that the results given by the autoencoder are more visually appealing for our datasets (Section 4).

2.2 3D shape synthesis

In this section, we discuss a variety of approaches that can be used for synthesizing 3D shapes, including classic computer graphics approaches and methods based on deep learning.

2.2.1 Statistical models

Statistical shape models summarize an entire collection of shapes of the same class with a set of parameters [16]. To obtain a useful model, the number of parameters in the model should usually be much smaller than the dimensionality of the data, e.g., 20 parameters compared to thousands of vertices that compose the triangle meshes of the shapes. A popular parametric model in the literature is built by first representing all the shapes in a collection with a consistent triangle mesh, where all the vertices across the meshes are in correspondence. Then, by applying a statistical analysis method such as Principal Component Analysis (PCA) to the vertex positions, the main modes of variation of the shapes can be discovered. A user can then explore the shape space by tweaking a set of sliders that modify the parameters, and visualize the corresponding shapes [16]. This type of approach has been used most notably for creating shape models of human faces [11] and bodies [4]. Nevertheless, these methods can also be applied to more specialized types of shapes. For example, Wang et al. [67] create statistical models of botanical trees by defining an appropriate metric in the shape space and then analyzing the data with a geodesic form of PCA.

Recently, Streuber et al. [62] introduce a method to generate human bodies with user-defined ratings of textual shape attributes, e.g., the user quantifies how "muscular" or "curvy" the body should be. The main building block of the approach is the learning of a function that relates the textual attribute ratings to parameters of a human shape model. Our method shares similarities with this work, such as the use of keyword attributes to describe the characteristics of the shapes to be created. However, our method is designed specifically to be used with recent deep networks that learn shape spaces described by latent vectors, while our statistical regression provides an indication of the relevant parameters of the shape space that can be further explored by the users.

2.2.2 Shape synthesis by part composition

Generating shapes by recombining parts of existing shapes is an effective approach for shape generation. However, the geometry of the shape parts is fixed to that of the exising shapes. Jain et al. [36] proposed an algorithm which analyzes the hierarchical structure and contacts between parts of shapes in a database of 3D objects and then synthesizes new shapes in an interactive manner. The challenge in this kind of synthesis is to minimize the generation of shapes with undesirable parts. For that, a pairwise score between the matches of source and target parts are calculated. In a similar vein, Gonzalez et al. [28] propose a method for synthesizing shapes by recombining fine-grained parts of existing models. Kraevoy et al. [41] compute a compatible segmentation between two shapes, which provides a one-to-one correspondence between the parts of the two models. Their work seems faster and simpler for users to generate new shapes and the variation among synthesized shapes are also plausible. By following the blending approach and part composition between a source and target shape, Alhashim et al. [2] creates new 3D shape of similar class considering the variation both in topology and geometry of shapes. Other blending approaches create shapes using basic set theories - union, intersection, division which produces smooth transition between shapes [52]

2.2.3 Object synthesis with deep networks

In the last few years, deep neural networks have shown great success in performing generative tasks, being used for example to generate images [53], indoor scenes [68], and terrains [31]. Our focus in this thesis is on the generation of man-made objects, and a variety of deep learning approaches for creating this type of data representation have also been introduced. In this section, we review deep learning methods related to object synthesis.

Wu et al. [73] introduced a generative adversarial network (GAN) that learns a latent space of object shapes based on a voxel-grid representation of the shapes. The latent space can then be sampled to generate new objects. The dimension of their latent space is 200 and shapes are represented as volumes of $[64 \times 64 \times 64]$ voxels. In addition to a GAN, Wu et al. also implemented a 3D-VAE-GAN to synthesize 3D objects by mapping 2D images of the objects to the latent representation. This work demonstrates great potential for both object creation and reconstruction from

2D images with deep neural networks. Similarly, Brock et al. [15] use variational autoencoders and voxels grids of $[32 \times 32 \times 32]$ to learn a shape space that can be used for object generation, while Dai et al. [22] use an encoder-predictor network to perform shape completion, which can be seen as a constrained type of synthesis.

Moreover, since a voxelization is not the ideal parameterization for 2D surfaces embedded in 3D, other approaches have explored alternative shape representations in conjunction with neural networks. Achlioptas et al. [1] introduce a generative model for representing shapes as point clouds, where the topology of the shapes is not explicitly encoded. On the other hand, Groueix et al. [30] explicitly store the shape topology by encoding the shape as an atlas of local parameterizations. Each parameterization can then be sampled and transformed into a mesh patch with known topology, although artifacts may result at the seams of different patches. Park et al. [51] encode shapes as signed distance functions, which can be transformed into triangle meshes with existing reconstruction methods and provide much smoother objects. For shapes that can be represented as graphs of parts, Li et al. [46] introduce an approach that encodes shapes as a hierarchy of parts, while Nash and Williams [49] introduce a method that models both part geometry and connectivity. Both methods can be used for synthesis.

All of the methods discussed above can be used to synthesize shapes based on input latent vectors or example representations of shapes. The latter can be achieved in encoder-decoder networks by training the encoder with a different data representation than the decoder, e.g., encoding an image into a latent vector and then decoding it into a voxel grid. However, directly controlling the synthesis to achieve a desired design is challenging, as in the case when the user knows only the general attributes that the designed shape should possess. We see our work as a way of complementing the existing approaches by enabling the user to explore the latent spaces and more closely control the synthesis process.

2.2.4 Interactive modeling

The ability to synthesize or manipulate 2D images or 3D shapes interactively, traditional software provides high-level functionalities which is sometimes difficult for users to understand. On the other hand, several deep networks have achieved advancement in synthesizing new images and shapes without any high-level input from external object [73] [15]. Recently, using deep learning architecture, an application *GANpaint*

is introduced which allow user to synthesize 2D images interactively by selecting desired objects from pre-defined list [9]. But still, a few deep learning approaches have been proposed to control the synthesis of shapes more closely. In this section, we will discuss some traditional as well as deep learning approaches for modeling 3D shapes interactively.

Traditional 3D modeling

Botch et al. [13] presents a modeling framework where users' are allowed to define constraints on the model and deform a mesh using basis functions. Similary, Bokeloh et al. [12] proposes a variational deformation model where users' can provide constraint through some sliders that are applied to deform the mesh. Another approach uses the box-like templates to represent the shapes of a dataset and produce a 2D embedding of these templates which user can navigate to explore the collection or creating new shapes [6]. Although, similar to our approach, most of these works use semantic attributes as constraints or user input, it does not integrate with any deep neural architecture.

3D modeling with deep networks

Guérin et al. [31] introduce a terrain authoring system, where the user can draw a 2D sketch of the topographic features of a terrain, and a GAN then synthesizes a corresponding elevation model. For man-made shapes, Huang et al. [33] introduce an approach where a user can draw a 2D sketch of the desired shape, and the system then generates a corresponding 3D object. This is accomplished by learning a neural network that maps sketches to parameters of a procedural modeling program, which then synthesizes the shape based on an algorithm. Smirnov et al. [61] map sketches to deformable parametric templates. Liu et al. [47] allows to create partial shapes with a Minecraft-type interface, which are then projected onto the latent manifold learned by a GAN, providing a corresponding shape in the latent space. Sung et al. [63] introduce the *ComplementMe* system where a user can design a man-made shape in an interactive manner. Given a partially constructed shape, the system suggests additional parts that can be added to model, according to a graph-based shape model learned by neural networks.

In this thesis, we also introduce an approach that allows users to control the synthesis of shapes in a more interactive manner. In contrast to the works discussed

above, our method allows the synthesis to be performed as a mix of a guided and exploratory process. The user first provides keywords that describe the high-level characteristics of the shapes. The system then allows the user to explore the subspace of shapes that possess these characteristics. This is advantageous in that the user may have a general idea of the intended design, but may not have necessarily determined all the fine details that the final shape should possess, or may not have sufficient artistic skills to manually model or draw the fine details. We explain our method in more detail in the following chapter.

Chapter 3

Methodology

To address the problem of shape synthesis with user guidance, our method uses two different neural networks. Moreover, we represent the shapes processed by the networks as 3D volumes. In this section, we discuss the two networks first, and then explain the workflow of our method and dataset pre-processing.

3.1 Shape synthesis with neural networks

Our solution to user-guided synthesis consists in a label regression neural network coupled with a shape synthesis neural network. We describe these two networks in the following subsections.

3.1.1 Label regression network (LRN)

The label regression network (LRN) takes as input user-provided keywords that describe a target shape. The network then maps the keywords to the latent space learned by the shape synthesis network. Thus, the output encoding of the latent space used by the LRN is dependent on the representation learned by the synthesis network. Our synthesis network is an encoder-decoder type of network, where the encoder maps shapes to n-dimensional latent vectors, and the decoder maps latent vectors back to shapes. The synthesis network is described in the next section.

A first design for the LRN would be to simply train it to map keywords to latent vectors. That would allow to map a set of keywords to a single point in the shape space, but would not allow the users to explore the shape variability. Thus, our chosen design for the network is to map a set of keywords to a distribution of latent

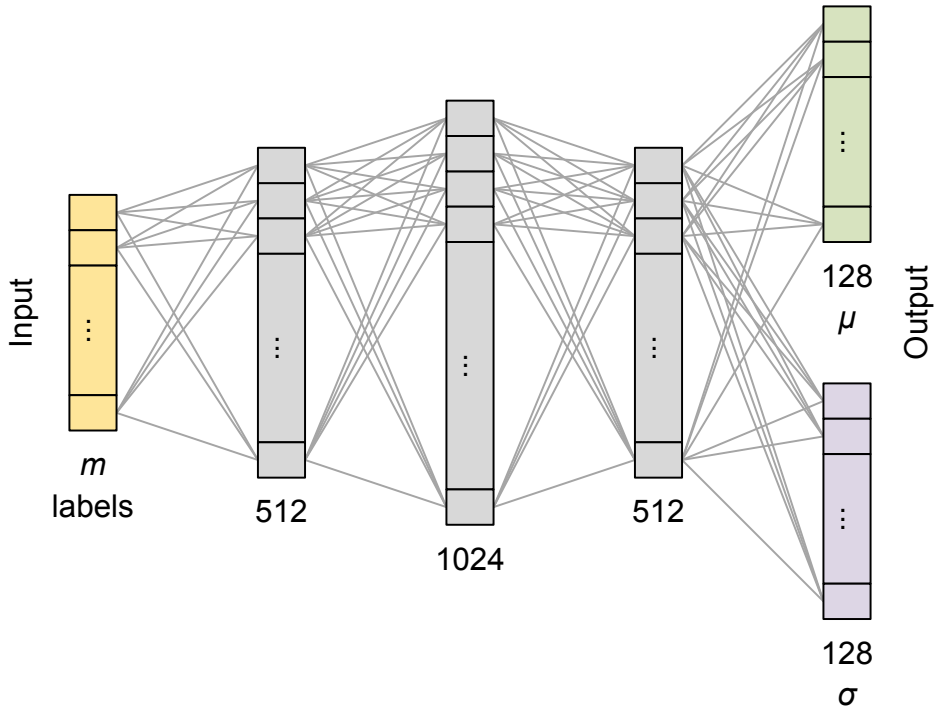


Figure 3.1: Architecture of our feed-forward deep network for label regression (LRN), mapping $m = 25$ input labels into $n = 128$ Gaussian distributions. The numbers denote the dimensions of the input/output and intermediate representations.

vectors, which can be seen as a subspace of the shape space. The user can then sample different vectors from the distribution to explore the subspace, or methodically change the entries of the latent vectors according to the distribution.

Input and output. The input to the LRN is a set of keywords represented as a binary vector $l \in \mathbb{B}^m$, where the entry l_i associated to the i-th keyword in the dictionary is set to 1 or 0 depending on whether the keyword is selected or not. The output of the network is a distribution of latent vectors conditioned on the input labels, represented as two vectors μ and σ. Specifically, for each dimension i of a hypothetical latent vector z, the network outputs a mean μ_i and standard deviation σ_i that describe a Gaussian distribution. In our work, we use $m = 25$ labels and latent vectors of size $n = 128$.

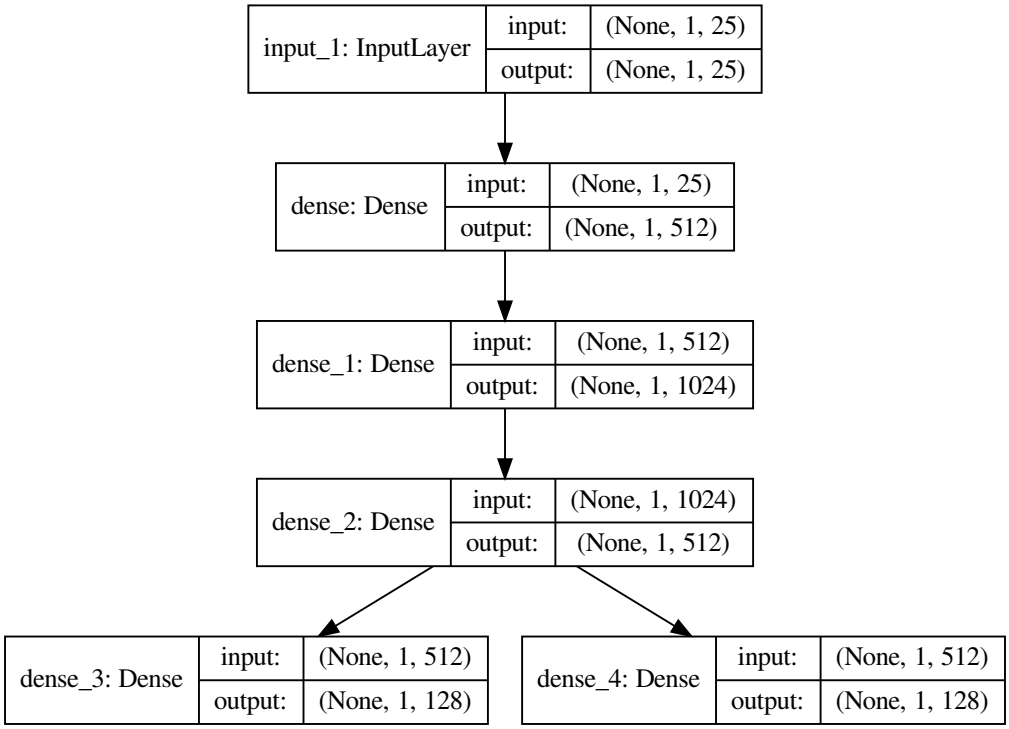


Figure 3.2: Description of each layer of the label regression network (LRN). The format of the input and output data is denoted in the form (batch size, height, and width). Note that the batch size is initially "None" because the network is configured for constant input and output dimensions.

Network architecture. The architecture of our LRN is shown with the diagram in Figure 3.1. The regression is performed with a feed-forward deep network composed of three hidden layers, one input layer with m nodes (with one node per label), and two output layers that provide the parameters of n Gaussian distributions. All the layers are fully-connected, since all of the output distributions may be dependent on all of the input labels. Since the standard deviations σ_i should always be positive, for the output layer producing σ, we use modified exponential linear units (ELUs) as the activation functions to ensure the non-negativity of the outputs. We define the

ELU for an input scalar x as:

$$\mathrm{ELU}(x) = \begin{cases} x + 1, & \text{if } x \geq 0, \\ \exp(x), & \text{otherwise.} \end{cases} \tag{3.1}$$

Figure 3.2 shows the flow of information between the layers of the LRN. Since the output of the LRN is a distribution (μ, σ), each of the output layers has dimension (1,128), which is the same as the dimension of the latent space given by the SSN.

Training and loss function. The LRN is trained with a set $\mathcal{T}_{LRN} = \{(l^j, z^j)\}$. Each sample (l^j, z^j) corresponds to one binary label vector l^j that describes the training shape and the latent vector $z^j \in \mathbb{R}^n$ produced for the shape by the synthesis network. We describe our dataset in Section 4. The training objective of the regression is to produce, for each entry i of each sample z^j, a Gaussian distribution from which the sampling probability of z_i^j is maximized. Thus, for each pair of predicted μ_i and σ_i that describe the Gaussian distribution, we need to maximize the probability density function (PDF) of the distribution for the target value z_i^j. Equivalently, the loss function of the network is defined as minimizing the negative logarithm of the PDF, which for one entry i of one sample j is denoted as:

$$\mathcal{L}_{\mathrm{LRN}}(z_i^j) = -\log\left(\frac{1}{\sigma_i\sqrt{2\pi}}\exp\left(-\frac{\left(z_i^j - \mu_i\right)^2}{2\sigma_i^2}\right)\right). \tag{3.2}$$

3.1.2 Shape synthesis network (SSN)

We use an autoencoder (AE) to synthesize shapes with our approach, where shapes are represented as volumes partitioned into voxels. Although there exist more advanced synthesis approaches using deep networks, such as the ones discussed in Section 2.1.2 that employ representations based on atlases or signed distance functions, we opt to use a baseline synthesis approach to demonstrate our exploration-based method. Nevertheless, our method is general enough and can be combined with other approaches.

On the other hand, as discussed in Section 3.1.1, the training data $\mathcal{T}_{LRN}$ for the label regression is composed of pairs (l^j, z^j) of label and latent vectors. Thus, to generate the training data, we require a synthesis network that can encode shapes into a latent representation. This requirement would rule out the use of a standard GAN, but any encoder-decoder network can be used, such as AEs, AE-GANs, variational

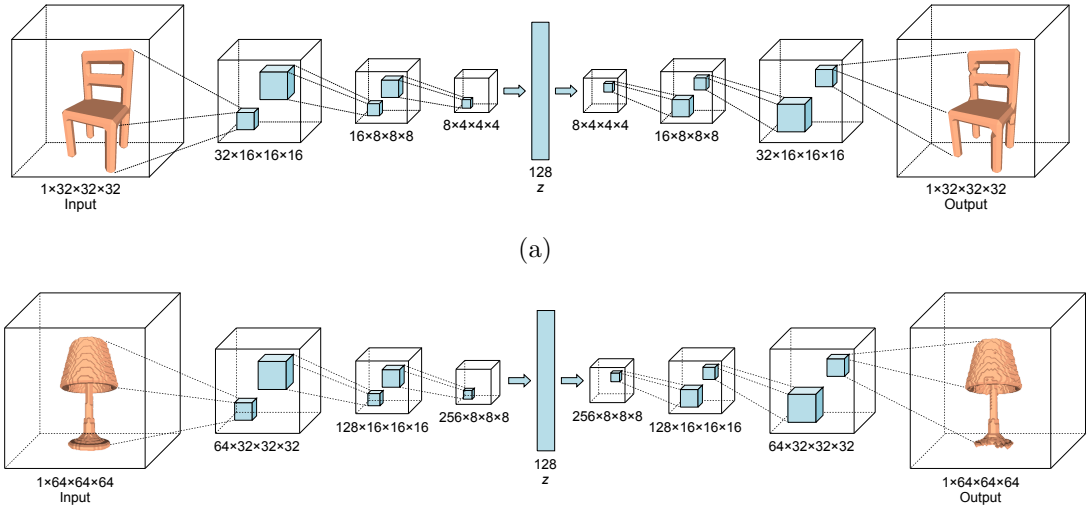


Figure 3.3: Architecture of the auto-encoder network that we use for shape synthesis (SSN). Diagram (a) shows the SSN used for input shapes represented as 32^3 volumes, and (b) shows the SSN for input shapes represented as 64^3 volumes. In each architecture, the encoder translates an input volume into a latent vector z by using three levels of convolution and downsampling, while the decoder translates a latent vector back into a volume shape. The numbers denote the dimensions of the intermediate representations and the dimensions of the input/output volumes and latent vector.

autoencoders (VAEs), and VAE-GANs [44].

In principle, using a VAE or VAE-GAN should provide a more meaningful latent space and thus better interpolation results. However, we experimented with AEs, VAEs, and AE-GANs, and found that the interpolation results are comparable to an AE architecture. This may be in part since generative models require large datasets for training, whereas in our work we use datasets in the order of hundreds of shapes. We selected the AE as our SSN at the end since its results are more stable and reproducible. Networks following the generative adversarial paradigm, such as AE-GANs, are difficult to train and often do not converge to a reasonable latent space.

Input and output. The SSN is composed of two networks: an encoder and a decoder. We propose two different architectures for the SSN: one architecture encodes the input shapes as volumes of $32 \times 32 \times 32$ voxels, while the other architecture encodes shapes as $64 \times 64 \times 64$ volumes. A voxel can be defined as either occupied (1) or empty

(0). We experiment with these two architectures to evaluate whether data loss in the voxelization process affects the learning of the model and the quality of the synthesized shapes. In each architecture, the encoder takes as input a shape represented as a set of 32^3 or 64^3 voxels and outputs a latent representation $z \in \mathbb{R}^n$ of the shape. The decoder then translates a latent vector back into a shape represented as a volume. Given that the last layer of the decoder uses a sigmoid activation function, each output voxel contains a real value in the range $[0, 1]$. Thus, we transform each voxel into a binary value according to a threshold of 0.5, that is, the voxel becomes 0 if the real value is below 0.5, or 1 otherwise.

Network architecture. The two proposed architectures of the SSN are shown in Figure 3.3. In each architecture, we use a symmetric AE where the encoder and decoder have the same number of layers. The encoder uses three layers for three levels of downsampling, while the decoder performs upsampling also at three different levels. In the first architecture, shown in Figure 3.3(a), the input is downsampled from $32 \times 32 \times 32$ to $4 \times 4 \times 4$ using three convolutional layers followed by a batch normalization layer that helps the network to learn features independently from the output of previous layers. The second architecture, shown in Figure 3.3(b), is composed similarly of convolutional layers and a batch normalization layer. The input is downsampled from $64 \times 64 \times 64$ to $8 \times 8 \times 8$. The number of filters, which represents the output dimension of each layer, is different in the two networks. However, in both architectures, we use kernels of size $4 \times 4 \times 4$ and stride 2 for all the convolutional layers. The output layer of the encoder, which generates the latent vector z, is a fully connected layer with a ReLU activation function. We experimented with different values for the size of the latent vectors and found that 128 dimensions provide the best encodings with our architecture. The decoder follows the inverse of this architecture for performing upsampling, with the exception that the output layer uses a sigmoid activation function. Figures 3.4 and 3.5 show the two architectures with more detail, including input and output for each layer. Note that, a flatten layer is used in the encoder network prior to providing input to the fully connected layer.

Training and loss function. We adopt the mean squared error as the loss function of the SSN. For each component $\hat{x}_i$ of the reconstructed shape $\hat{x}$ (with $1 \leq i \leq N$,

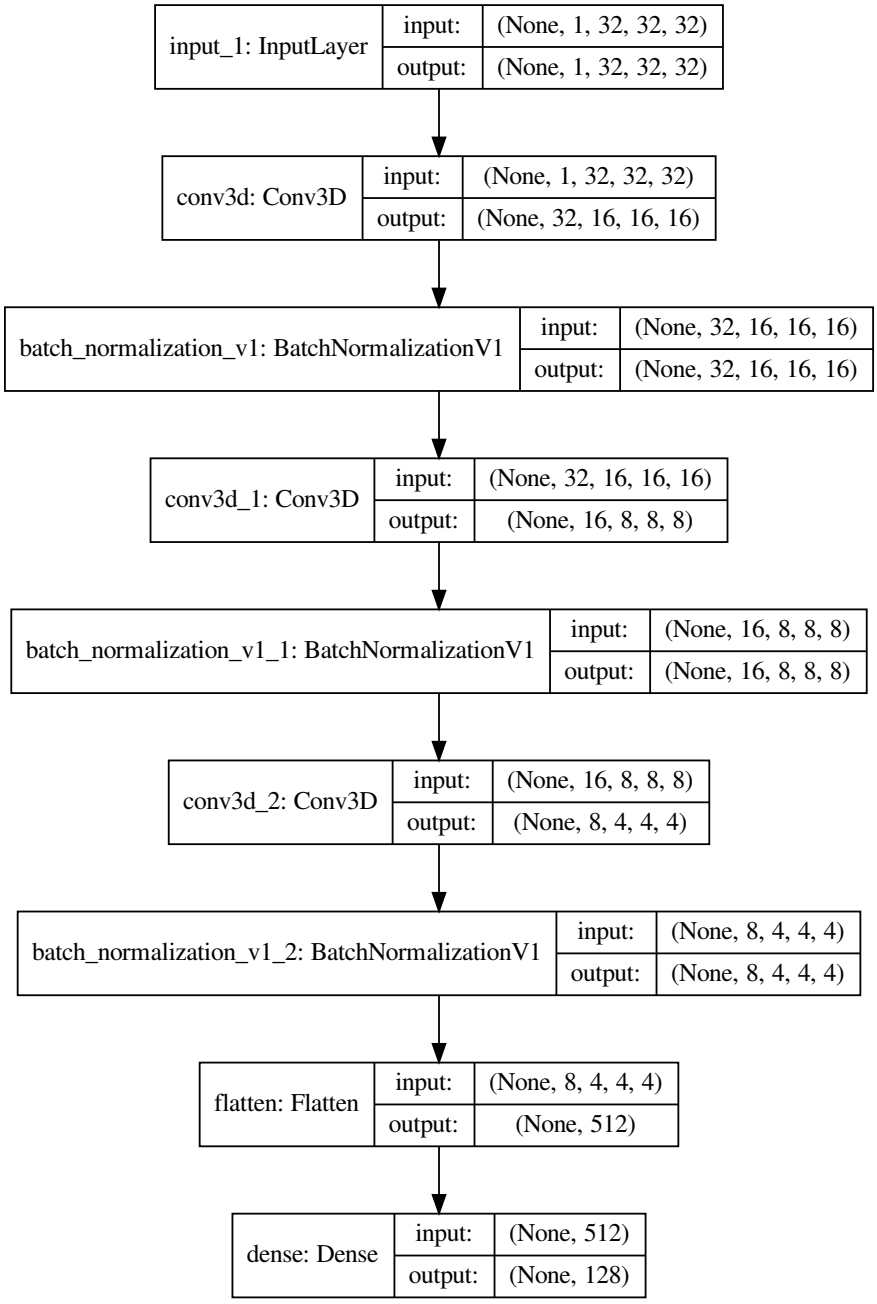


Figure 3.4: Detailed view of the layers and input/output parameters of the encoder network used in the SSN. The format of the input and output data is denoted in the form (batch size, height, and width). The batch size is initially "None".

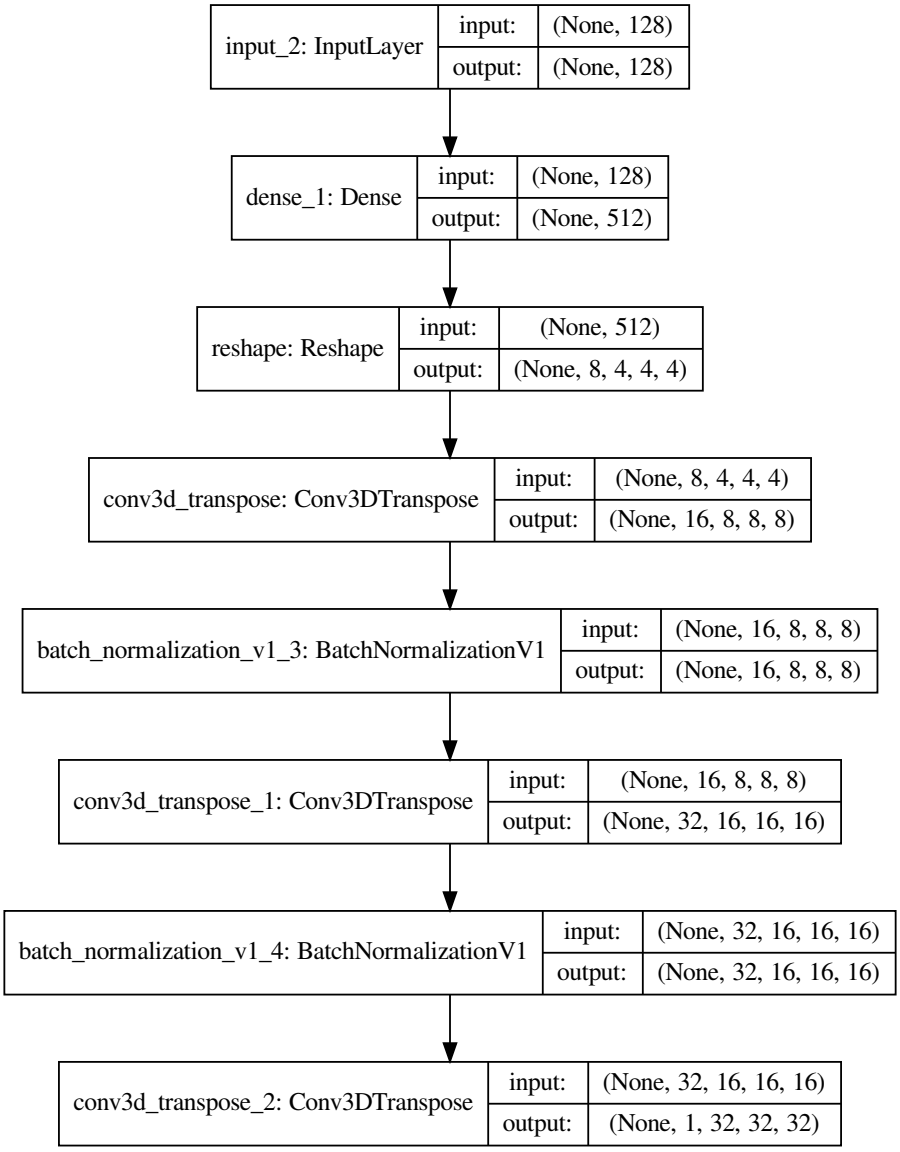


Figure 3.5: Detailed view of the layers and input/output parameters of the decoder used in the SSN. Note that the encoder and decoder networks are symmetric architectures. The format of the input and output data is denoted in the form (batch size, height, and width). The batch size is initially "None".

where $N = 32^3$ is the total number of samples), we minimize the average squared

difference between $\hat{x}_i$ and its corresponding component x_i in the input shape x:

$$\mathcal{L}_{\text{SSN}}(\hat{x}, x) = \frac{1}{N} \sum_{i=1}^{N} (\hat{x}_i - x_i)^2 . \tag{3.3}$$

The training set $\mathcal{T}_{SSN}$ is composed of triangle meshes voxelized as explained in Section 3.3.

3.1.3 Exploration of the shape space

Once the two networks are trained, as explained above, the user can employ them to perform an exploration of the latent space and synthesize new shapes. The user first inputs a set of labels l which are mapped with the LRN to two vectors (μ^p, σ^p) representing the predicted distributions. The distributions are then used to create a latent vector z according to further user input. Specifically, for each dimension $i \in \{1, \ldots, n\}$, our system exposes the dimension to the user only if σ_i^p is significant (larger than a threshold ϵ). If σ_i^p is not significant, we set $z_i = \mu_i^p$. We show the exposed dimensions to the user sorted by the magnitude of the standard deviations. Finally, the user can explore the exposed dimensions by varying a set of sliders that navigate around the space $(\mu_i^p - \sigma_i^p, \mu_i^p + \sigma_i^p)$ to define each z_i. For any z defined with this process, the synthesis network generates a corresponding shape. This results in an interactive experience where the user can analyze the shape generated by the new parameters selected. As we will discuss in Section 4, we observed experimentally that the number of dimensions with large σ is relatively small, and thus it is feasible for the user to explore the latent space, as only a small number of dimensions needs to be manipulated.

3.2 Summary of the workflow of the method

Using our method comprises two main stages: training and testing. Figure 3.6 provides an overview of the workflow of the method.

We divide the training stage into two steps based on the data flow into the networks. In the first step, the shape synthesis network (SSN) is trained with the input volumes corresponding to all the shapes in the dataset. The result is a latent space

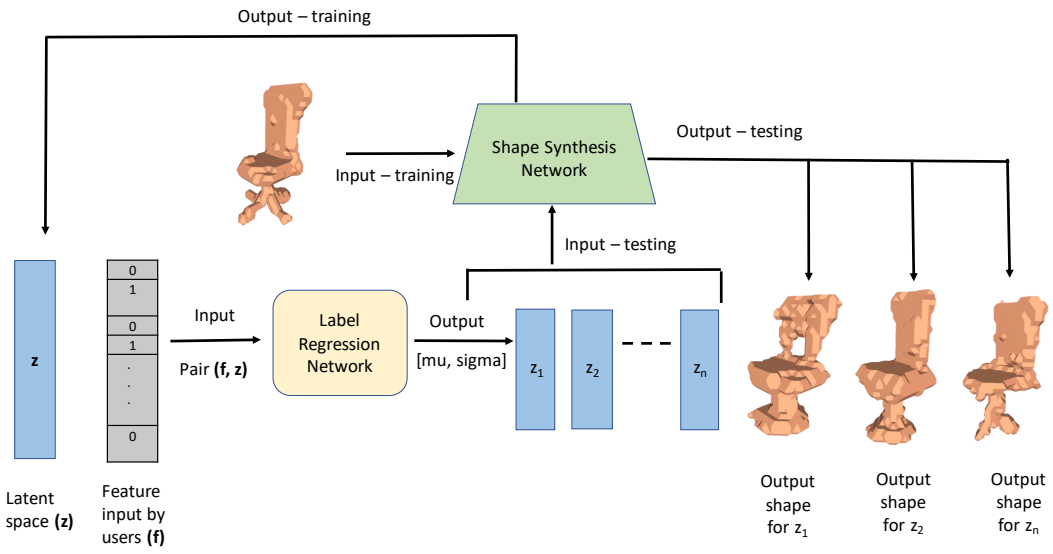


Figure 3.6: Summary of the workflow of the method. The shape synthesis network (SSN) and label regression network (LRN) are highlighted in green and yellow, respectively. The data flow among the networks is divided into two steps.

learned for the input shapes. To choose the dimension n of the latent space that contains enough information for the decoder to reconstruct the volumes with minimal loss, we experimented with multiple values for n. Figure 3.7 shows the visual differences in reconstructions of two training shapes based on different n. After analyzing the reconstructed volumes, we selected $n = 128$, as it produces shapes with reasonable detail. Once the SSN is trained, we feed each training shape into the SSN to obtain its latent representation. Finally, we train the LRN based on tuples of shape labels and their latent vectors.

During the test phase, the user explores the latent space with the aid of the two networks for generating shapes. Given a set of labels, the LRN produces a distribution (μ, σ). The user can then explore this distribution to produce a latent vector z, which is then given to the SSN to produce an output shape.

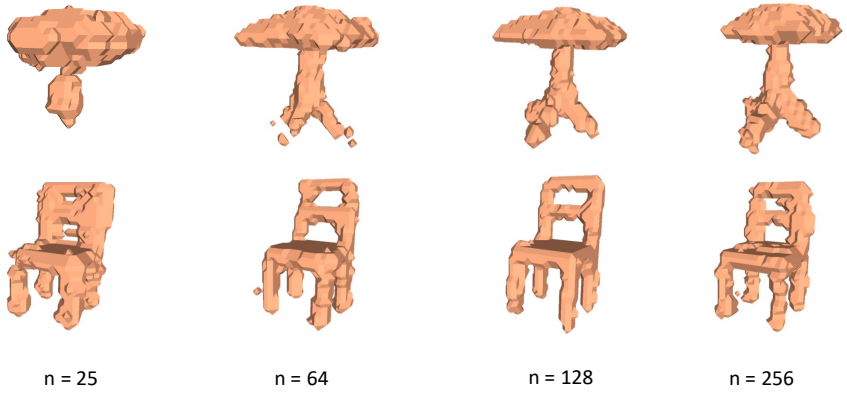


Figure 3.7: Reconstructions of training shapes obtained for different n, where n is the dimension of the latent vectors.

3.3 Voxelization of the datasets

Given that the shapes in our datasets are represented as triangle meshes, we voxelize the shapes into volumes, since it is easier to define convolutions over volumes and use them with a deep neural network. To voxelize a mesh, first we define a volumetric grid around the mesh. Next, we subdivide the triangles of the mesh until each edge is smaller than the sides of a voxel. Finally, we denote as occupied all the voxels that contain vertices of the subdivided mesh. Occupied voxels are assigned the value **1** while empty voxels are **0**.

Since it is unclear whether using a higher-resolution voxelization provides more accurate reconstructions, in the next section, we experiment with volumes of 64^3 voxels alongside 32^3. Note that the selection of keywords for each shape is done according to the voxelized result, to reflect the features that are present in the voxelized output rather than the original mesh. In our datasets, 18% of chairs and 7% of lamps looked different after voxelization which represents 72 out of 400 chairs and 7 out of 100 lamps. For example, some *back holes* of chairs and *top holes* of lamps disappeared after voxelization. However, for most of the shapes, the features are well preserved, and the learning is based on a consistent labeling of the voxelized shapes.

Chapter 4

Results

We first describe the datasets used in our work and setup of our experiments. Then, we present qualitative results, followed by an overall evaluation of our method.

4.1 Shape datasets

We present results for our method on a set of 400 chairs selected from the COSEG dataset [72] and 100 lamps selected from the auto-aligned version of the ModelNet40 dataset [58]. Both the COSEG and ModelNet40 datasets consist of several categories of objects. We selected two categories for our experiments, *chairs* and *lamps*, that have some of the shapes with the most structural variation in the datasets. All the shapes used in our work are pre-aligned, although a dataset including shapes with different orientations could also be considered, which would provide a form of data augmentation for increasing the size of the datasets. The Modelnet40 dataset contains a total of 144 lamp shapes, divided into 124 training shapes and 20 test shapes. We found some shapes in the datsets which are difficult to describe with our labeling scheme. Hence, we combined both train and test shapes and selected 100 lamps that can be described with our labels. We voxelize the shapes to train the autoencoder for shape synthesis. We experiment with both 32^3 and 64^3 voxelizations for both datasets. However, for simplicity, we will mainly show results for chairs with a 32^3 representation, and for lamps with a 64^3 representation.

When voxelizing both datasets into volumes of dimension 32^3 and 64^3, details on some of the shapes are lost. Figures 4.1 and 4.2 show a comparison between 32^3 and 64^3 voxelizations for one chair and one lamp, respectively. We see that a 64^3 volume is able to preserve more of the fine details of the shape compared to 32^3, although

the main structure of the shapes is preserved in both resolutions. Nevertheless, the learning of the LRN should not get affected by the quality of the volume, as long as we use a volume of at least 32^3 voxels and label the datasets based on the features visible on the voxelized shapes.

(a) (b) (c)

Figure 4.1: Comparison between the mesh of a chair and voxelizations of the mesh with different resolution: (a) Original mesh. (b) Voxelized shape with a 32^3 volume. (c) Voxelized shape with a 64^3 volume.

(a) (b) (c)

Figure 4.2: Comparison between the mesh of a lamp and voxelizations of the mesh with different resolution: (a) Original mesh. (b) Voxelized shape with a 32^3 volume. (c) Voxelized shape with a 64^3 volume.

4.2 Label datasets

For labeling the two datasets, we chose pragmatic keywords to describe features which are frequent and visible in most of the shapes. It is challenging to create an optimal set of keywords that can capture the diversity of features in the shapes, describe each shape precisely, and also be easily readable by users. Thus, we constrain our labels to simple keywords that describe a variety of visual attributes of the shapes, especially properties of their individual parts, rather than using keywords that describe artistic or abstract styles of the shapes.

To create the labels, we define a set of shape features and their admissible values, which are then transformed into binary labels. For chairs, we divided the shapes into three main regions: *back*, *seat*, and *leg*. Then, we defined labels to denote the attributes of each of these regions. For example, the property *leg length* admits the values *short* and *long*, which are then transformed into the binary labels *leg length short* and *leg length long*. In total, we have 25 labels for the set of chairs, which are summarized in Figure 4.3. For lamps, we divided the shapes into 3 regions: *shade*, *body*, and *base*, and defined attributes for each region. We have 24 labels in total to describe the set of lamps, as shown in Figure 4.4.

While selecting the labels for both datasets, we also performed a statistical analysis of the frequency of the features described by the keywords, to ensure that the labels appear with enough frequency and that the LRN has enough data to learn an accurate mapping for each keyword. For example, regarding the *back* regions of chairs, 16% are of type *half size*, 69% *full size*, and 13% *no back*. Regarding the *seat* regions, 61% are of type *square* and 39% *circular*. More significant differences in the distributions of labels are seen among the features in the *leg* regions of chairs, where 70% are of type *four legs*, 19% *one*, 7% *three*, 2.5% *two*, and only 0.75% are of type *five legs*. These differences in the frequencies of leg types are also visible in the generated results, where most of the variation is composed of chairs with *one, four,* and *three* legs. For lamps, significant differences among features can be seen in the *shade* and *body* regions. Whereas the *bell* type of shades appears in 42% of the whole dataset, 15% are of *cylinder* type, 32% *round*, 11% *rectangular*, and only 4% of shades are of the *funnel* type. To describe the *body* of lamps, four types of labels are used: *pipe, spiral, beam,* and *curved*. However, most of the lamps are of type *pipe* (79%) and *beam* (12%), with only 4% and 5% of type *spiral* and *curved*, respectively.

Although differences in the statistics of labels exist, both datasets hold enough

example shapes for most keywords. Figure 4.5 shows the frequency of the labels for the two datasets, to demonstrate the diversity of shape attributes in the datasets and unevenness of the label distribution. The majority of keywords have at least 5 example shapes in the dataset. For example, the number of chairs with the *box leg* keyword and lamps with the *curved body* keyword is 10 and 5, respectively. We were able to generate shapes with both of these features, which indicates that there is a sufficient number of example shapes in the datasets for learning to map these features. One exception are the keywords *five leg* for chairs and *funnel shade* for lamps, where the number of example shapes is only 3 and 4, respectively. For these keywords, we could not generate any good output shapes capturing the corresponding features.

We manually selected the labels according to the visual properties of the voxelized shapes to evaluate our method. As with any manual data labeling task, we observed that it is time consuming and sometimes difficult to label the shapes, as some voxelized shape might be deformed. To label all the 500 shapes, it took four weeks. However, to label larger datasets, crowdsourcing could be used as an effective means for performing the labeling.

4.3 Experimental setup

We trained both the label regression network (LRN) and the shape synthesis network (SSN) with the Adam optimizer and split the datasets into training batches of 10 shapes each. As the hidden layers of the LRN are fully-connected layers that require extensive training, we set the learning rate for the LRN to 0.0003 and train the network for 2,000 epochs (Figure 4.11(a)). For both versions of the SSN, we set the learning rate to 0.001 and train the networks for 200 epochs (Figure 4.11(b)).

4.4 Qualitative results

Figures 4.6 and 4.7 show example shapes synthesized with our approach for the chair and lamp datasets. The user first selects a set of labels, which are shown on the left column of the figures. The LRN then predicts distributions for all the entries of the

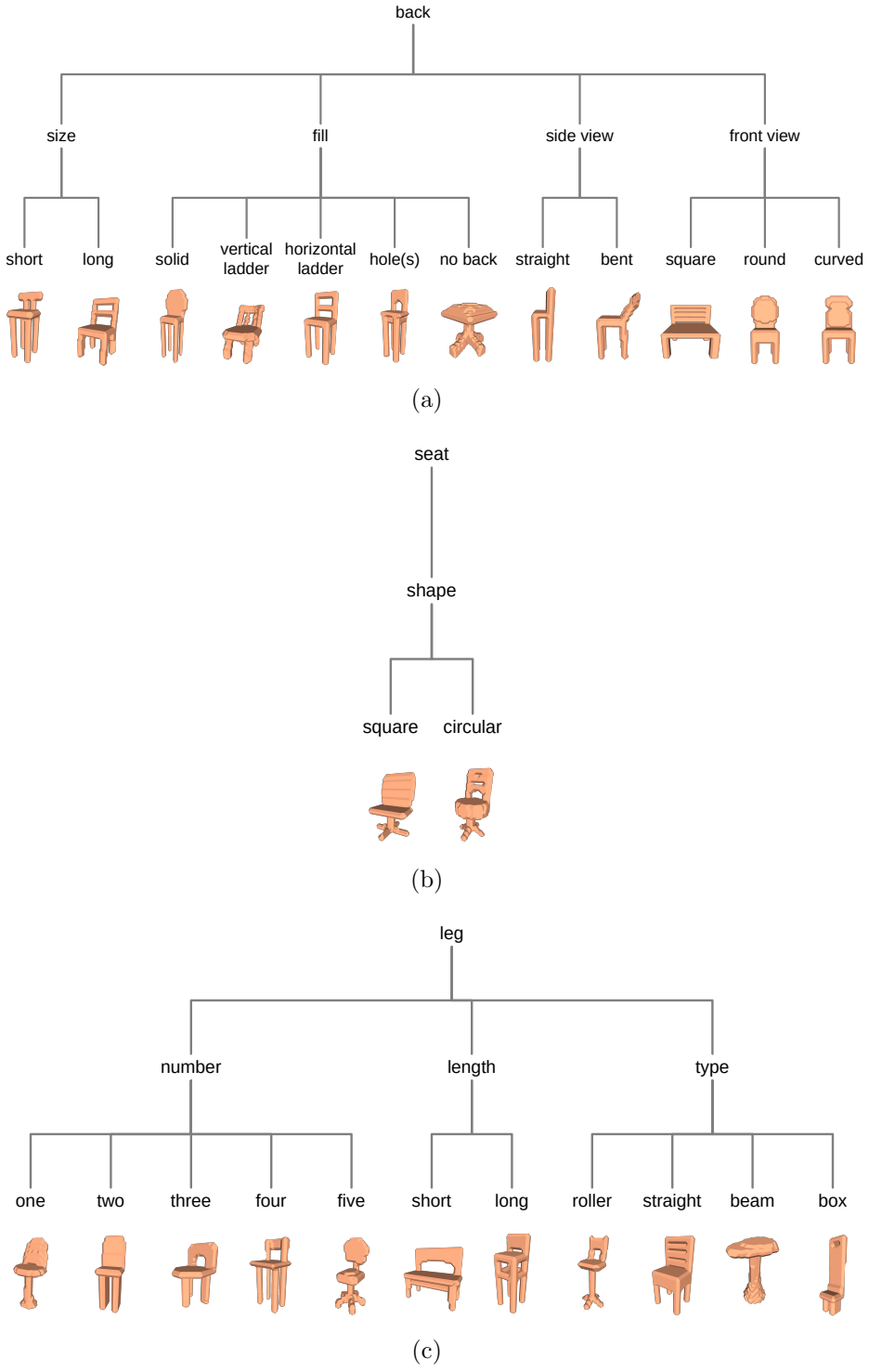
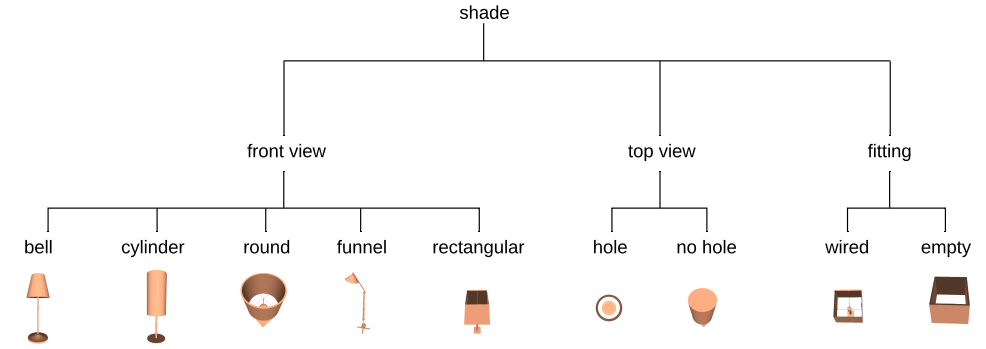
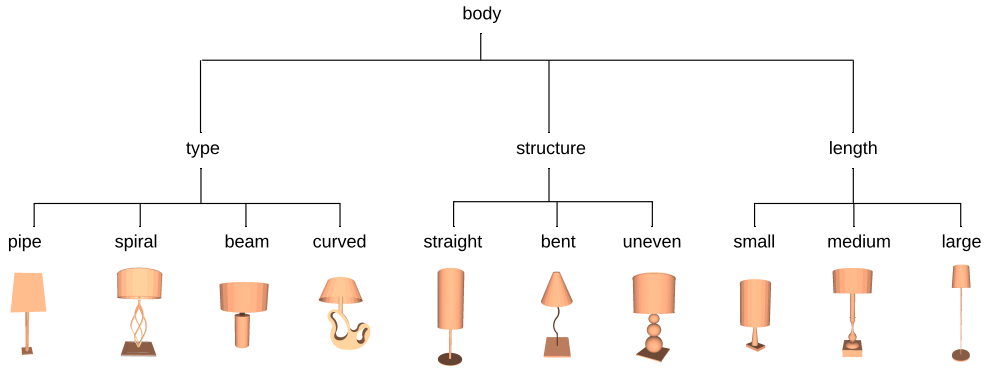


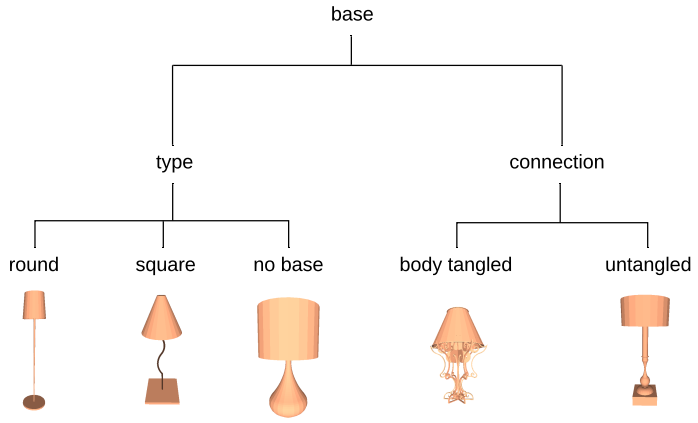
Figure 4.3: Labels that we use for describing the visual attributes of the set of chairs, along with an example shape for each attribute. (a) Attributes for the *back* region. (b) Attributes for the *seat* region. (c) Attributes for the *leg* region.



(a)

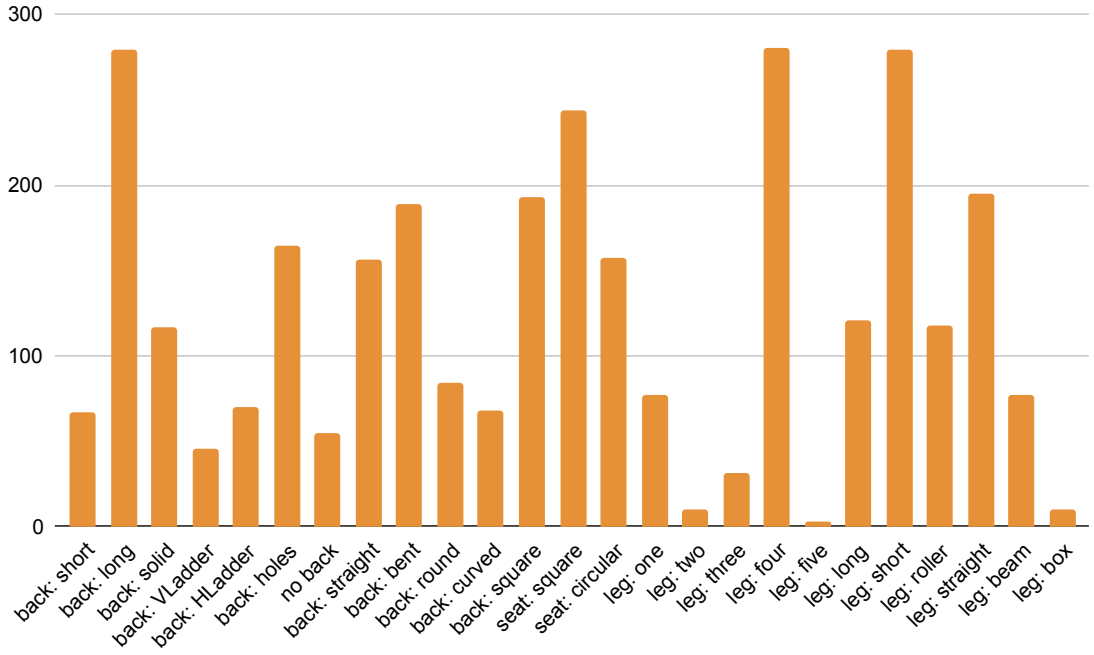


(b)

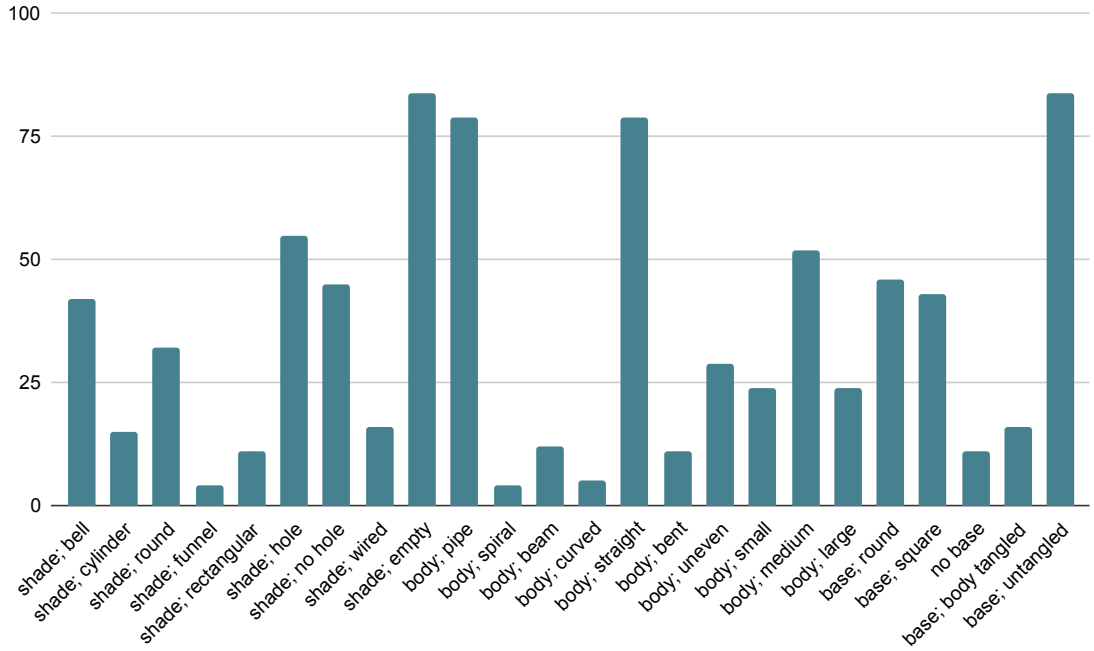


(c)

Figure 4.4: Labels that we use for describing the visual attributes of the set of lamps, along with an example shape for each attribute. (a) Attributes for the *shade* region. (b) Attributes for the *body* region. (c) Attributes for the *base* region.



(a)



(b)

Figure 4.5: Frequency of labels for (a) the dataset of chairs, and (b) the dataset of lamps.

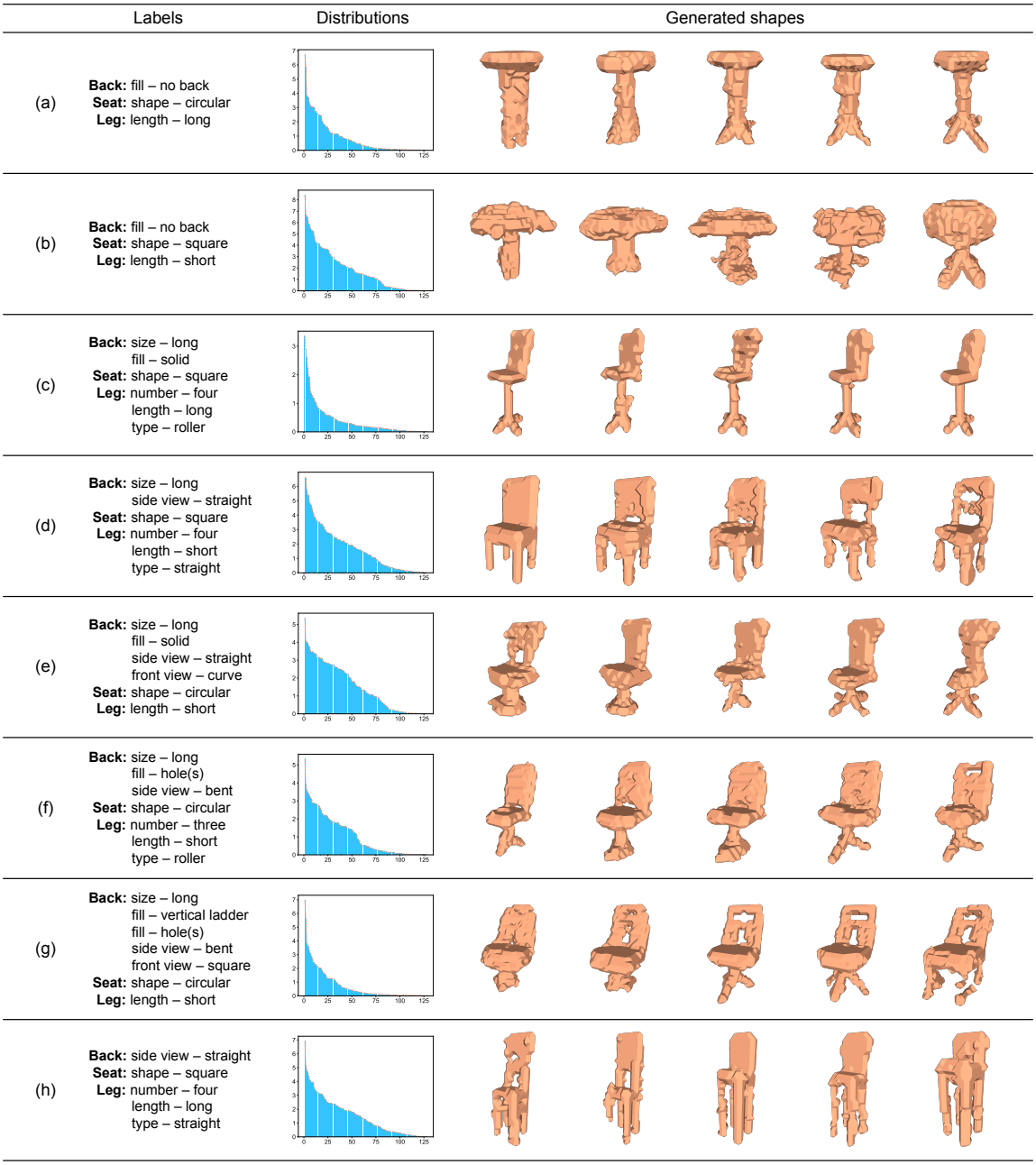


Figure 4.6: Examples of results obtained with our method for the set of chairs. Left: input labels selected by the user. Center: a plot of the sorted standard deviations (σ's) for the distributions of all the latent dimensions, summarizing the distributions derived by the label regression network from the selected labels. Right: shapes synthesized by creating different latent vectors according to the distributions. Note how the synthesized shapes possess the attributes described by the selected labels while revealing different variations in shape and structure.

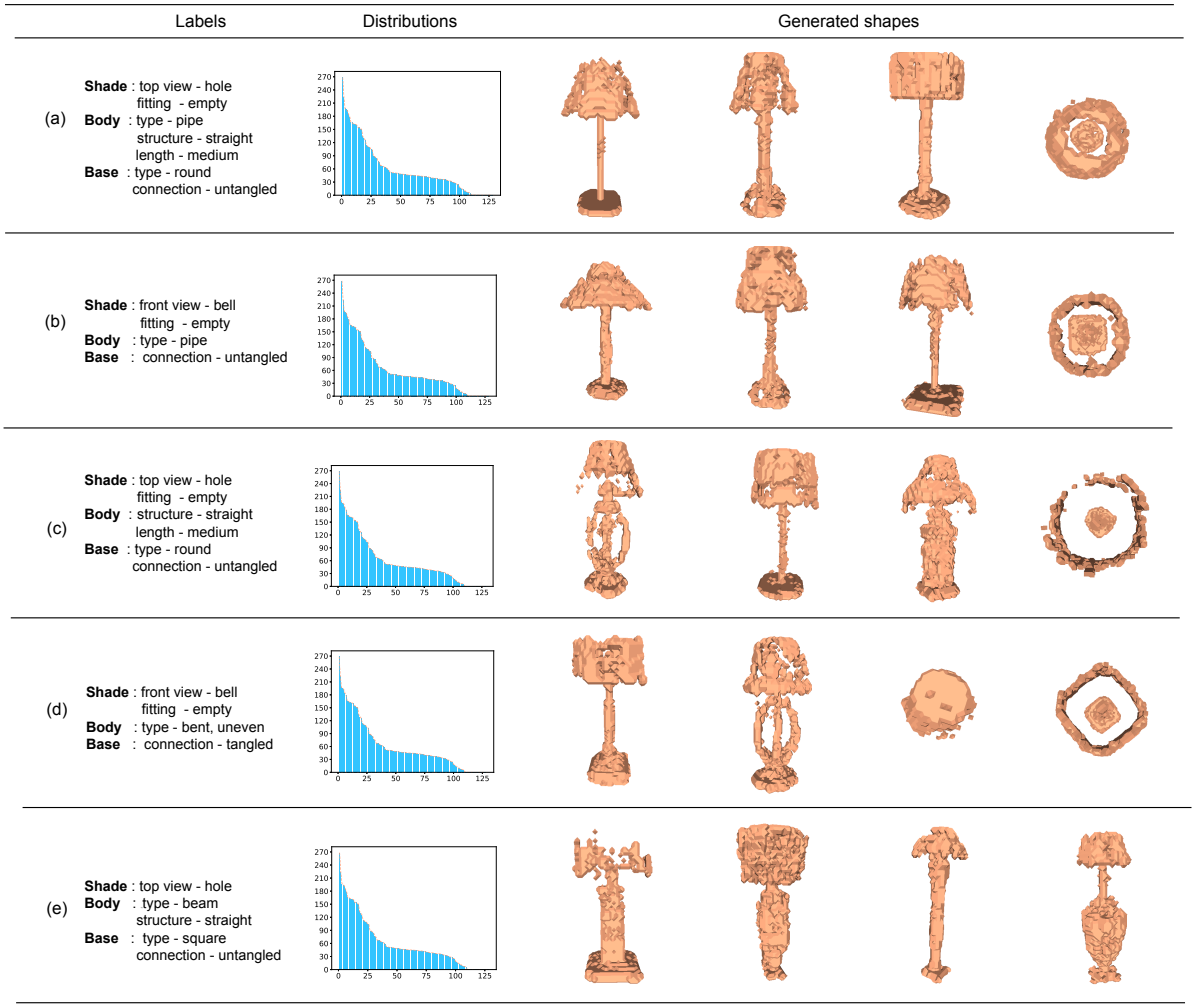


Figure 4.7: Examples of results obtained with our method for the set of lamps. Left: input labels selected by the user. Center: a plot of the sorted standard deviations (σ's) for the distributions of all the latent dimensions, summarizing the distributions derived by the label regression network from the selected labels. Right: shapes synthesized by creating different latent vectors according to the distributions. Note that the fourth example of a few rows is shown from a different viewpoint.

latent vectors. The center column shows a sorted plot of the standard deviations of the distributions, to demonstrate that not all the distributions need to be considered in the exploration. The right column shows examples of shapes that the user obtained by varying the entries of the latent vectors with significant standard deviation.

Although the synthesized shapes are noisy and not entirely smooth due to limitations of the synthesis network, we see how the generated shapes show variations in their structure and overall geometry. While the user was able to synthesize shapes with different structures, all of the shapes in a row still possess the attributes specified by the input labels. Moreover, we see that some of the selected labels provide richer subspaces with more variance (as seen by the shape of the σ plots), while other labels constrain more the variation that can be found. Note also that the selected labels include configurations of labels that do not exist in the training data.

We further examine the individual examples of synthesized chairs. In Figure 4.6(a), the user requested chairs with no back and a circular shape for the seat, but did not specify the shape of the legs besides their length. Thus, the exploration of the distributions provides shapes that mainly vary in the type of leg. The shapes in (b) are obtained from a similar set of labels but with short legs selected. In (c), the selected labels constrain the shapes much more. Thus, only the back varies from straight to bent and square to round, while other attributes remain fixed. In (d), the chairs are constrained by the labels to a fixed structure, except for the fill of the backs. Thus, the exploration uncovers shapes with solid or "ladder-type" backs. In example (h), the user selected both "holes" and "vertical ladder" for the attributes of backs, and the subspace allows to obtain shapes that have both a hole on the top and vertical ladder on the bottom of the back. Finally, we also see limitations of the approach in some of the examples. The chairs without backs in the dataset all have circular seats, and thus the examples in (b) have seats close to circular, although the user requested square seats.

For lamps, Figure 4.7(a) shows an example where the user requested a lamp with a body in the shape of a "pipe", straight, and of medium length, as well as a round base. However, the user did not specify any attributes for the shade except that it should have a hole on the top. Thus, the exploration provides lamps with different types of shades. A top view of a fourth example is also included at the end to show that the requested hole appears in the generated shapes. In (b), variations can be seen in the base of the generated lamps, which can be square or round. However, the

shade is constrained to a bell shape by the user. In (c), as there are no restrictions specified by the user on the body and shade type, bodies of spiral, beam, and pipe types are obtained with round and bell shades. In (e), the body is constrained to the beam type and to a straight structure, along with a square base. Based on these constraints, variations in the length of the body and type of shade can be seen among the generated lamps. In addition, (d) shows a limitation of the method where the exploration generates a lamp with a round shade and pipe body, although the user specified a shade of bell type and bent body. This likely occurs since lamps with the tangled base appear mostly in conjunction with round shades and pipe bodies. Nevertheless, variations such as a round or square base and a shade with or without a hole (as seen in the top view) are obtained.

Thus, as we see in these results, although the SSN can interpolate between similar styles, the overall variability of the results is dependent on the type of structures that appear in the training data.

4.5 Significance of standard deviation

One important question related to the exploration process is how to determine what standard deviation values are significant and will lead to variations in the synthesized shapes. To determine this threshold, we performed an experimental analysis of the sensitivity of the synthesized shapes to changes in the entries of the latent vectors. In this experiment, we start with a latent vector z derived from the mean of the distributions predicted for a set of labels, and synthesize an initial volume corresponding to z. Next, we add perturbation vectors with increasing magnitude to z, and measure the number of voxels that have changed when comparing the synthesized volume to the initial volume. We repeat this process for several perturbation vectors of the same magnitude and average out the observations.

The result of this experiment for two shapes is shown in Figure 4.8. We see that, for magnitudes below 1.5, the change in the volume is at most 800 voxels, which represents about 2% of voxels in a 32^3 volume. Figure 4.9 shows a few examples of the shapes synthesized from the perturbed vectors, where we see that a magnitude of less than 1.5 does not lead to noticeable changes in the results. Thus, we can select 1.5 as a threshold on σ for determining the dimensions to be explored by the user. In addition, we sort the dimensions by the magnitude of the σ values and thus the user

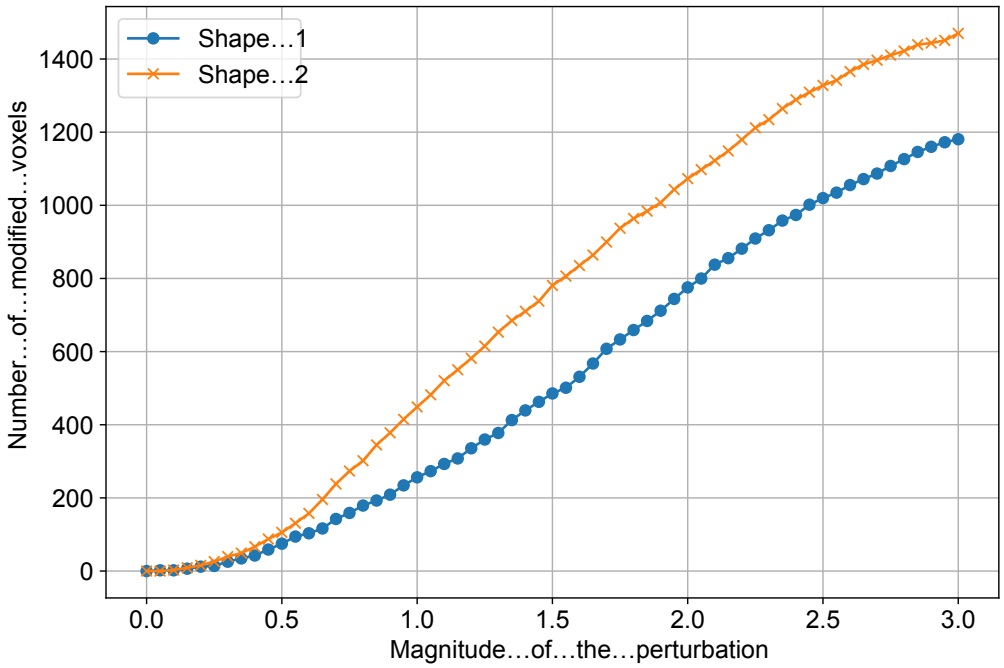


Figure 4.8: Sensitivity of synthesized shapes to changes in the entries of latent vectors. The x axis denotes the magnitude of the perturbation added to an initial latent vector, while the y axis reports the number of voxels modified in the synthesized volume, compared to the volume synthesized from the initial latent vector.

can give priority to dimensions with more variance. In the examples of Figure 4.6, we see that for many of the selected subspaces, the threshold requires the user to manipulate on average only 25 latent dimensions.

Finally, in Figure 4.10, we show an example of all the distributions predicted for a set of labels, to provide evidence that the idea of facilitating the exploration according to the standard deviation values is sound. We observe that dimensions with low standard deviations which are not degenerate (not zero) do naturally occur in the latent space. This implies that these dimensions still contribute to the encoding of the shape, but can be safely eliminated from the set of sliders that the user has to navigate by setting them to the mean value.

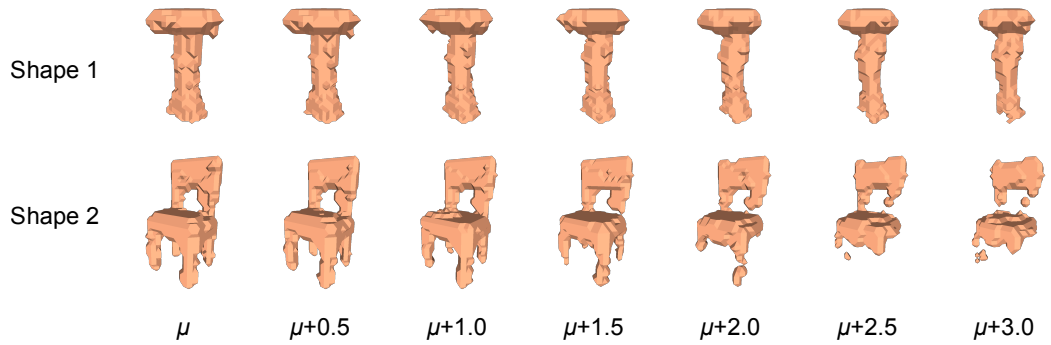


Figure 4.9: Visual examples of the changes that occur in a synthesized volume when adding perturbations to an initial latent vector.

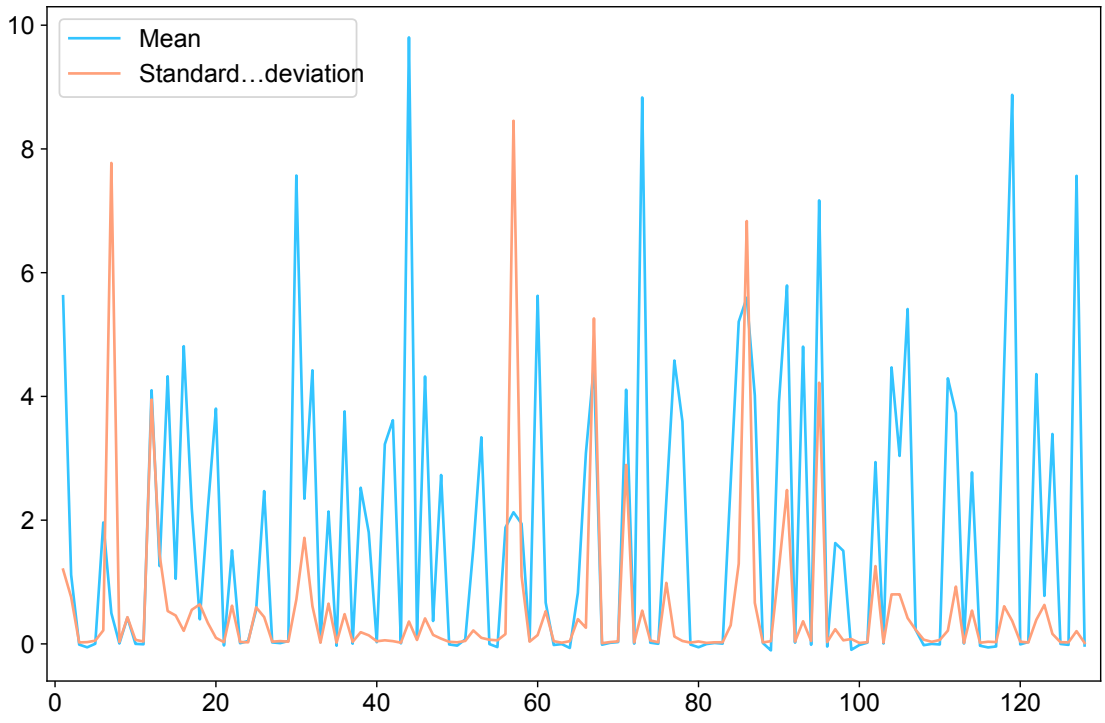
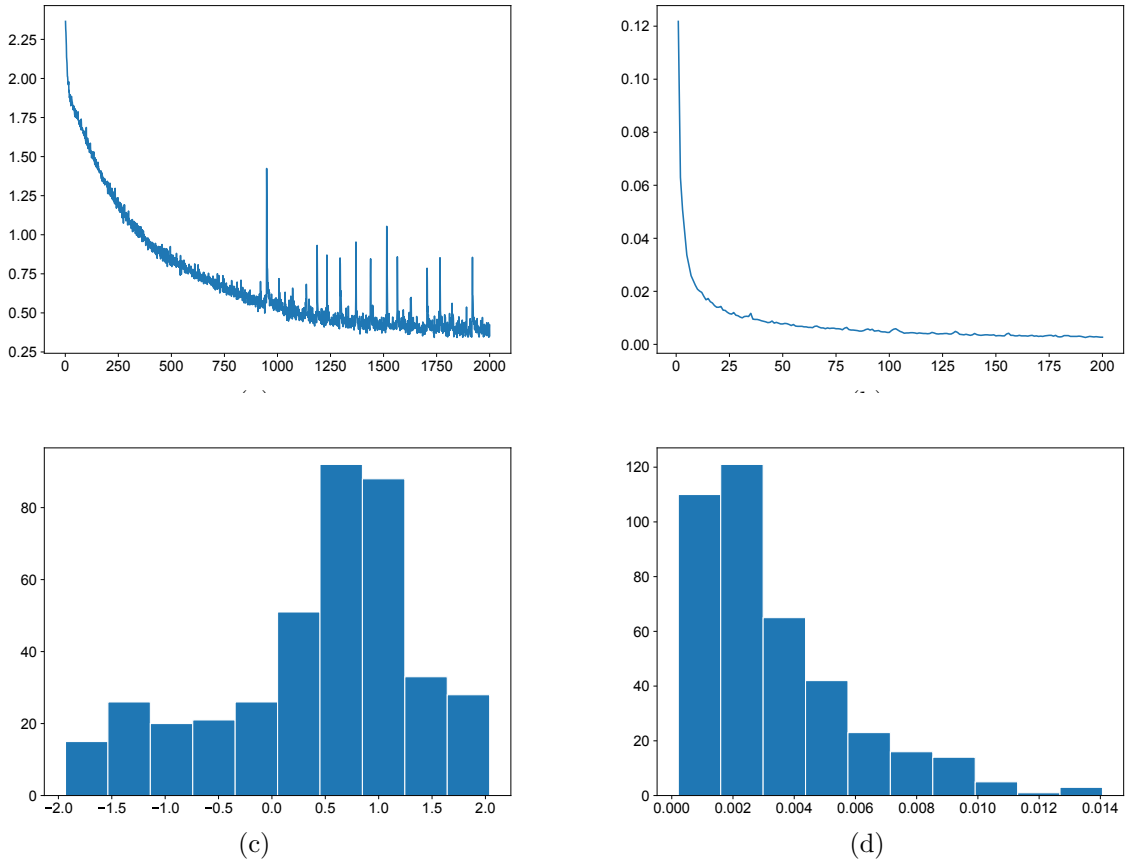


Figure 4.10: Distributions generated by the LRN for an input set of labels, represented with mean and standard deviation. Each entry of a latent vector is one point along the x axis. Note how several entries with non-zero mean can have near-zero variance.



(c) (d)

Figure 4.11: The training loss of (a) the LRN and (b) the SSN, where the x-axis is the epoch number. Histograms of per shape errors for (c) the LRN and (d) the SSN, at the end of training.

4.6 Evaluation of the learning

We analyze our results in more detail to demonstrate that the LRN is learning a meaningful mapping from labels to the latent space, and that the SSN is synthesizing the corresponding shapes.

In classification problems, to prevent deep networks from overfitting, it is important to train the networks with datasets that are large enough and with adequate parameter settings for the number of epochs and learning rate. On the other hand, our goal is mainly to learn a latent representation of a fixed training set. Specifically for the SSN, the main requirement in terms of generalization is that the network should be able to perform a meaningful interpolation between the latent vectors of

two training shapes. Thus, although our dataset is small, we did not observe significant overfitting with the chosen parameters, since our SSN allows us to interpolate between latent vectors of different shapes rather than just memorizing the individual latent vectors. However, we are not using our networks to predict the encoding for unknown test shapes, which could reveal some form of overfitting.

Given that our goal is to learn a latent representation of a fixed training set, the most important objective is to minimize the learning errors related to the training shapes. Figure 4.11(a) and (b) show the training loss for the LRN and SSN. We see that, at the end of the training, the loss is minimized significantly and becomes stable for the two networks. Moreover, Figure 4.11(c) and (d) show histograms of the losses for all the shapes obtained in the last iteration of training. We see that the error values for the majority of the shapes is low. The maximum absolute loss for the LRN is around 2 with an average of 0.42, while the maximum loss for the SSN is 0.015 with an average of 0.003.

We illustrate the relationship between these errors and the generated shapes by comparing a few reconstructions in Figures 4.12 and 4.13. We show example training shapes, their voxelization used for training the networks, and the voxelized model reconstructed from the latent encoding of the shape. In Figure 4.12 Shape 1, the *back* of the shape is described with attributes such as *vertical ladder* and *bent*. The corresponding voxelized shape preserves both of these attributes, although the thickness of the shape increases. However, the reconstruction is missing a few of the fine details on the ladders. For Shape 2, the feature *back* of the shape is described with the attributes *horizontal ladder* and *straight*. In contrast to the first example, the attributes are not preserved so well in the voxelized shape, although the reconstructed shape is close the the voxelization. Moreover, Figure 4.13 shows examples of voxelized shapes from the dataset of lamps. For this dataset, we show the shapes voxelized into 64^3 volumes and the front view is chosen as it shows the maximum amount of detail of the shape without obstruction. We see that the higher resolution better captures the details of the shapes, and the reconstuctions follow closely the training volumes.

Moreover, Figure 4.15, 4.14, 4.16 shows a visual representation of shape similarities derived from the trained networks and input labels, to demonstrate that the minimization of the losses leads to networks that learn meaningful statistical models.

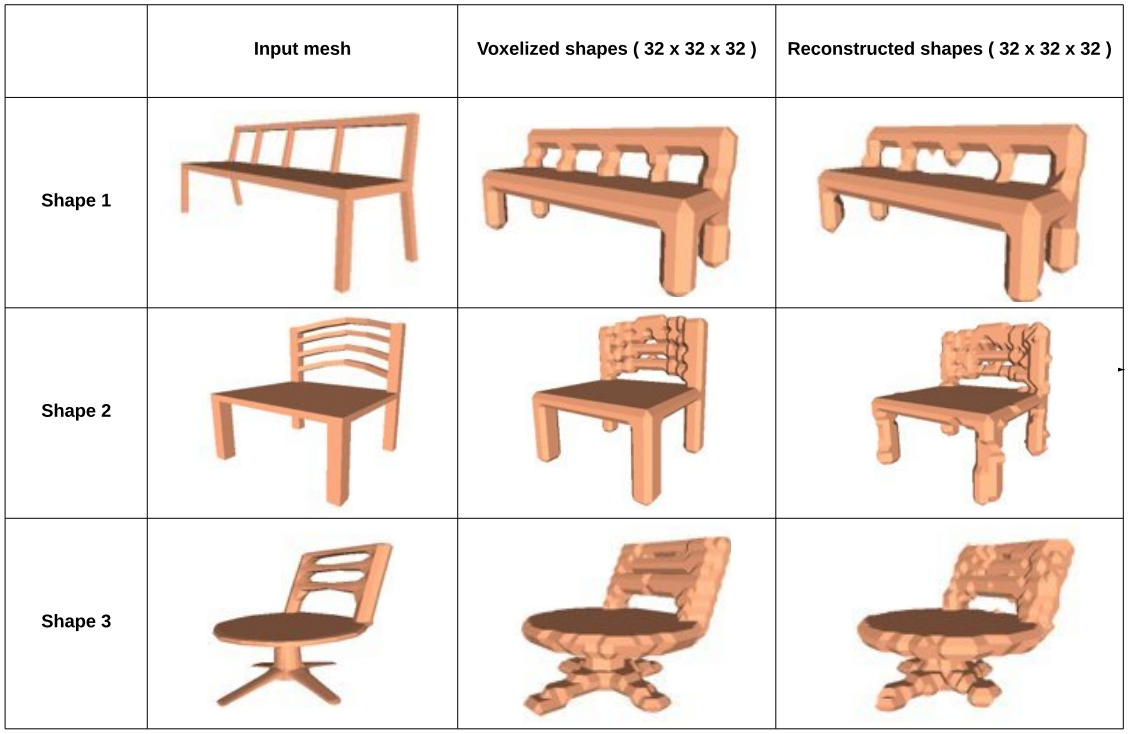


Figure 4.12: Visual comparison between original mesh, mesh voxelized into a 32^3 volume, and mesh reconstructed based on its latent vector, for examples from the chair dataset. Although fine details of some shapes are lost, the general structure and form of the shapes is preserved in the voxelization. Fine details of the voxelization can be lost when reconstructing the shape.

We show diagrams obtained by applying multi-dimensional scaling (MDS) to matrices of pairwise similarity among the labels of the training shapes, their latent vectors encoded by the SSN, and the means of the distributions predicted by the LRN for the labels of the training shapes. We show the same shapes as examples of the data points in the three plots. We observe that, the similarities are consistent across the plots, where the shapes with similar labels are also kept in close proximity in the latent spaces obtained by the SSN and in the distributions predicted by the LRN, implying that similar labels are mapped to similar distributions in the latent space. Thus, we can conclude that the objective of learning meaningful statistical models for the training shapes is being achieved for these two datasets.

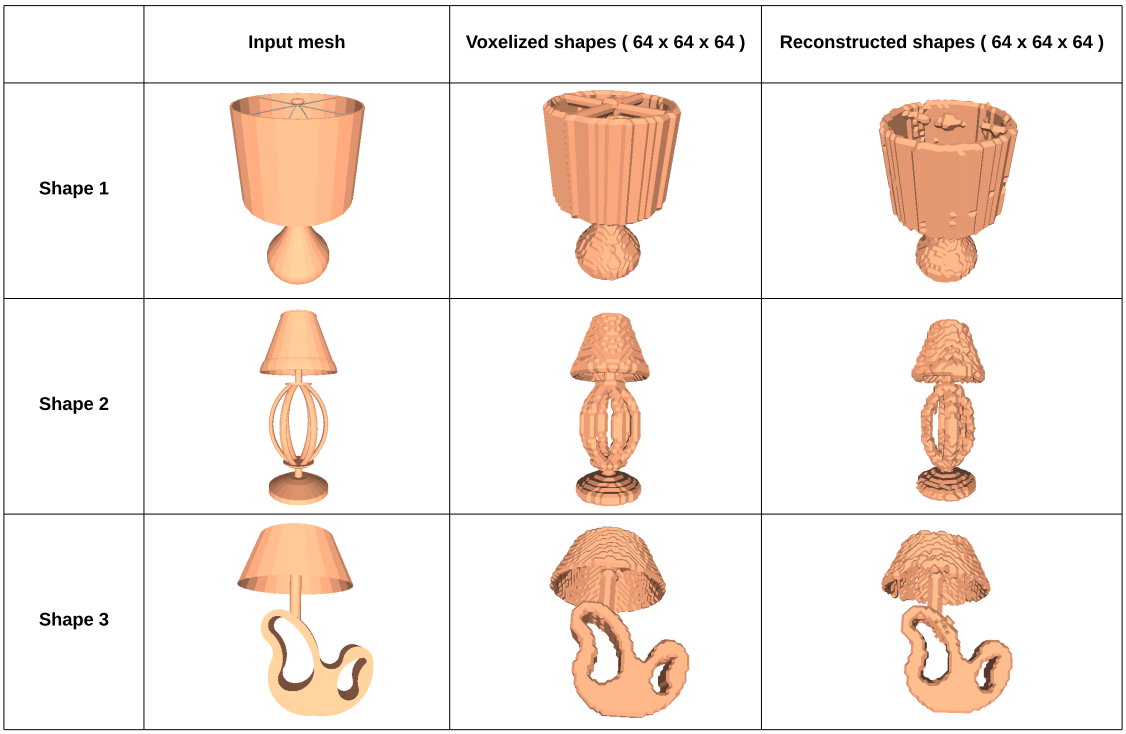


Figure 4.13: Visual comparison between original mesh, mesh voxelized into a 64^3 volume, and mesh reconstructed based on its latent vector, for examples from the lamp dataset.

4.7 Experiment with pre-trained model

To investigate whether the training set size influences the fitting of the SSN in our method, we performed an experiment with chairs where we pre-trained the SSN with the ShapeNet dataset [18] and then fine-tuned the network with the COSEG dataset. We use in total $1,288$ chairs for training, where 888 are from ShapeNet and 400 are from the COSEG dataset. We train the SSN for 150 epochs. In this experiment, we did no observe a significant difference in the generated shapes, as seen in Figure 4.17. This suggests that our method is applicable even if the size of dataset is not so large, given that we mainly rely on the intepolation capabilities of the SSN.

This experiment also relates to the scalability of our method. To augment a dataset with more shapes, it is possible to train the network starting from the current set of neuron weights to reduce training time.

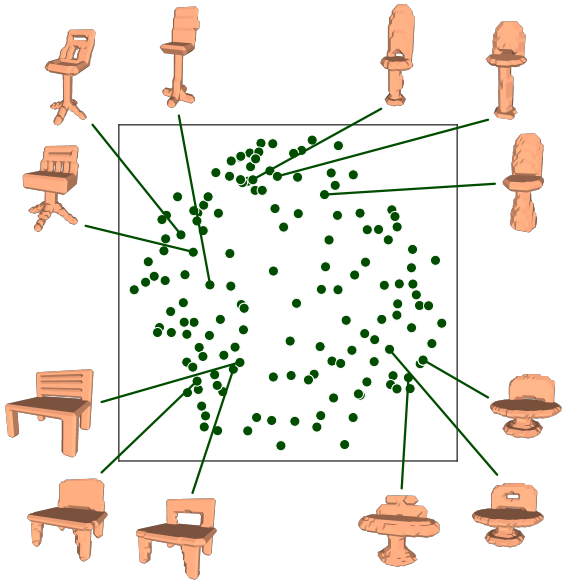


Figure 4.14: Multi-dimensional scaling diagrams reflecting similarity among mean of distributions predicted for the labels of the shapes (μ values). Only a few shapes are shown for clarity.

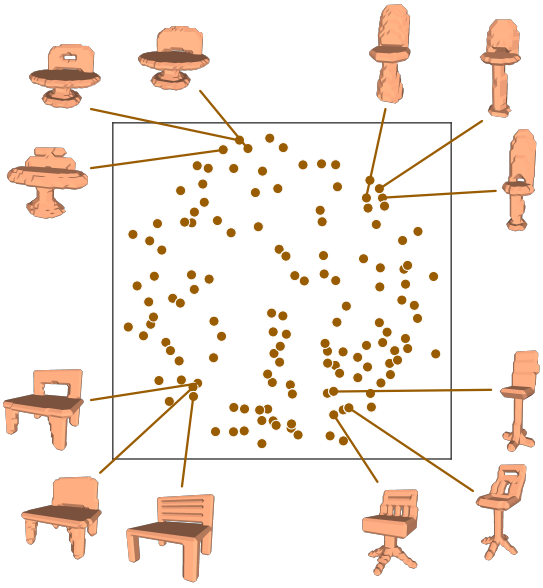


Figure 4.15: Multi-dimensional scaling diagrams reflecting label similarity for training shapes. Only a few shapes are shown for clarity.

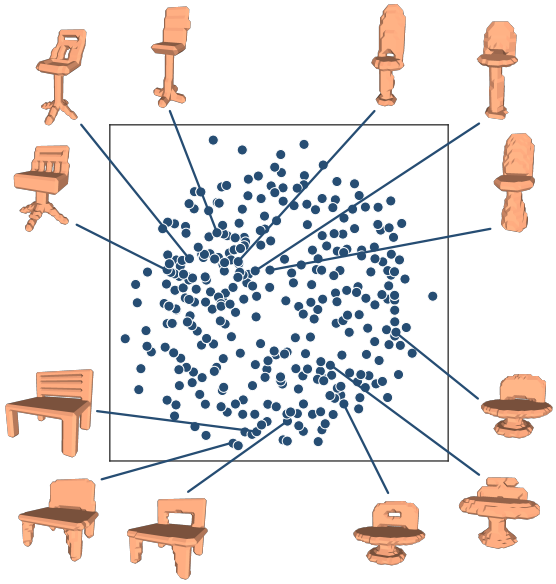


Figure 4.16: Multi-dimensional scaling diagrams reflecting similarity between latent vectors of training shapes (z vectors) Only a few shapes are shown for clarity.

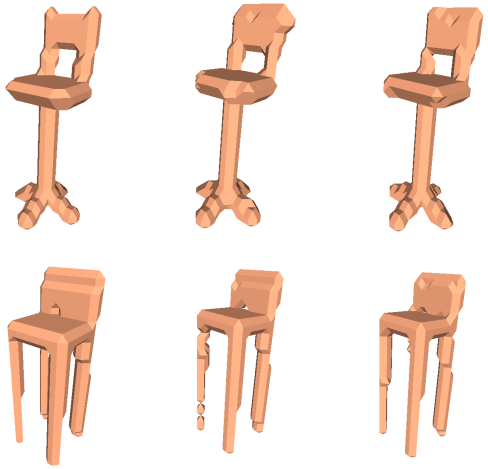


Figure 4.17: Comparison of results with and without pre-training the model. Left column: input shapes (voxelized). Middle: shapes obtained with the model trained only on the COSEG dataset. Right: shapes obtained with the model pre-trained on ShapeNet and then further trained on the COSEG dataset.

4.8 Shape synthesis network architecture

To select the best architecture for our SSN, we experimented with a few state-of-the-art architectures and compared their output. As mentioned before, we cannot use

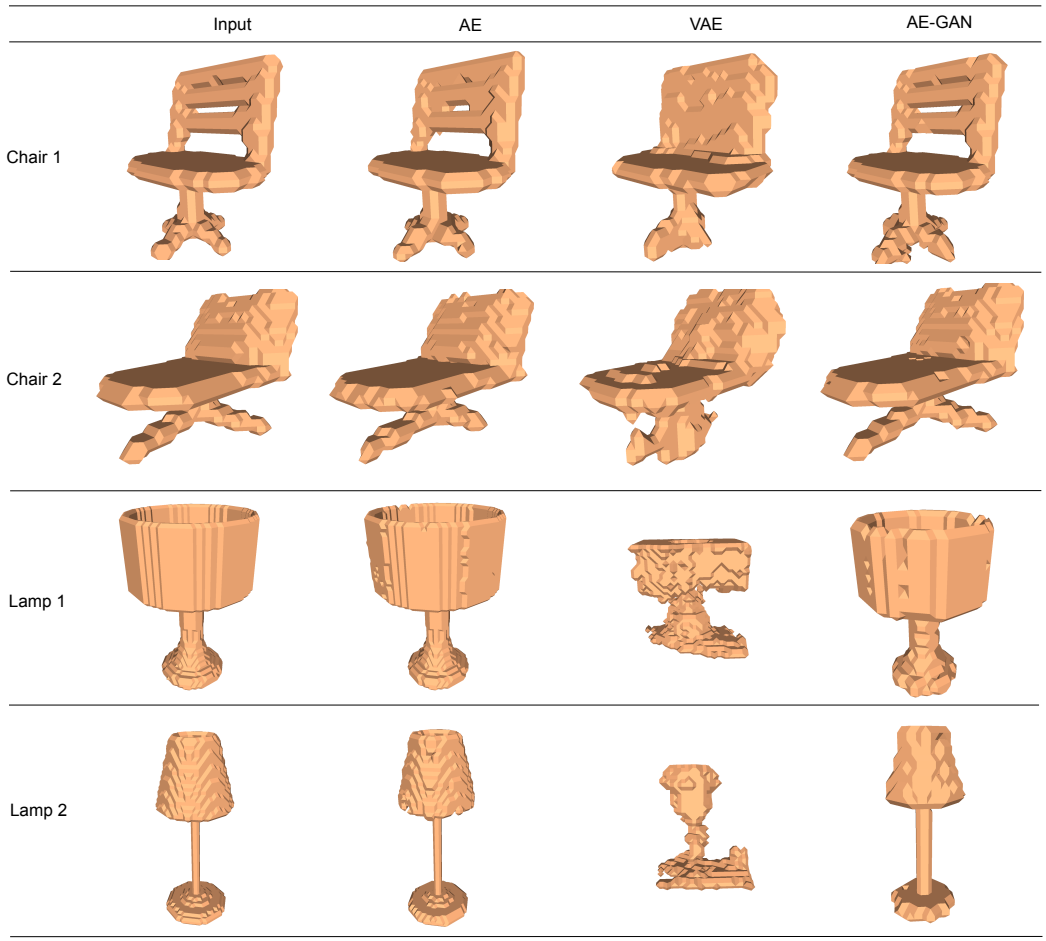
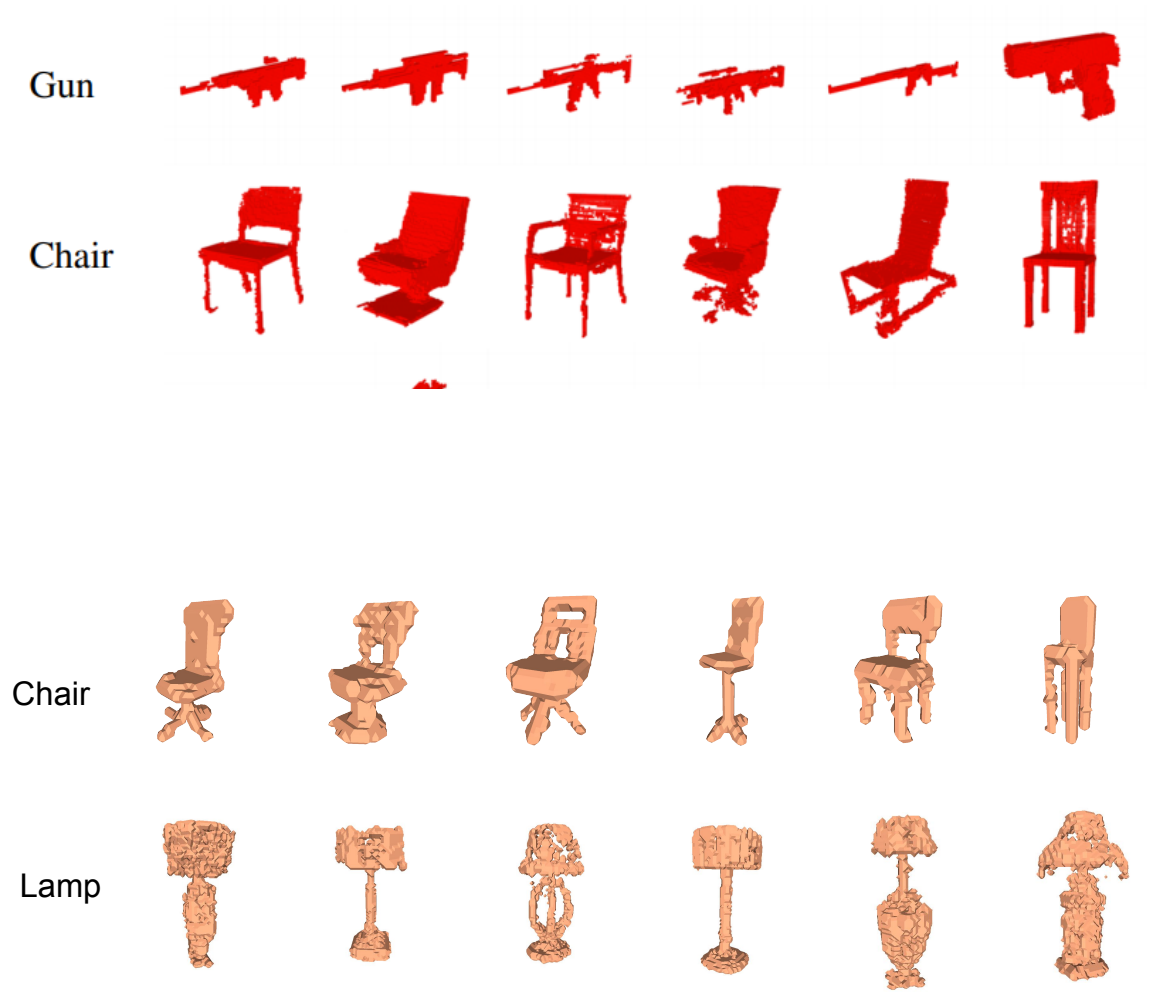


Figure 4.18: Examples of shapes reconstructed with three different architectures for the shape synthesis network. The first column shows the voxelized input, the second column shows the reconstruction obtained with an AE, and the third and fourth colums show reconstructions obtained with a VAE and AE-GAN.

a generative adversarial network with our method, as we need the ability to encode shapes into a latent space, rather than just generating shapes based on a random noise vector. Hence, we experimented with an AE-GAN and VAE, along with a more standard AE, since these networks provide latent encodings for shapes.

We use a VAE with a weighted binary loss [15], while the AE-GAN consists of three networks, an autoencoder, a generator, and a discriminator, trained with the KL divergence and binary cross entropy loss. We show a comparison of a few selected examples for the three networks in Figure 4.18, where we reconstructed selected training shapes based on their latent vectors. We see that the reconstructions for chairs



(b)

Figure 4.19: Visual comparison between our results and state-of-the-art results: (a) Shapes generated by Wu et al. [73], and (b) Shapes generated by our method. The images in (a) are courtesy of Wu et al. [73]

.

given by the AE and AE-GAN are similar. In contrast, the AE provides better reconstructions for lamps when compared to the AE-GAN. For the VAE, some of the features of the reconstructed chairs are missing, and sometimes the shapes deform significantly (such as the *leg* of Chair 2). For lamps, the AE and AE-GAN also outperform the VAE. Although the shapes generated by the AE-GAN and AE are comparable in quality, the stability of the AE-GAN varies, meaning that the quality of the results is not consistent when the network is trained again. Thus, we chose the autoencoder (AE) architecture as the final shape synthesis network for our method,

Table 4.1: Average execution time for synthesizing one shape.

Steps (Testing)	Time (in seconds)
Prediction (LRN)	0.0011s
Generation (SSN)	0.0017s
Total	0.0028s

Table 4.2: Time calculated for voxelizing one shape.

Steps (pre-processing)	Time (in seconds)
Voxelization	53.0204s

given its stability and the quality of the generated shapes. Moreover, in Figure 4.19, we compare our generated shapes with the results of Wu et al. [73]. We see that our results using an autoencoder architecture are similar in quality to the results produced by the VAE-GAN architecture.

4.9 Timing and machine configuration

Our deep networks were trained with an NVIDIA GeForce GTX 1070 Ti GPU with 8GB of memory and CUDA version 10.0. Table 4.1 shows the execution time required to synthesize one shape, where *prediction* denotes the prediction of a distribution by the LRN, *generation* denotes the generation of a shape with the SSN. Table 4.2 shows the *voxelization* time taken to voxelize one mesh. Table 4.3 shows the time needed for training the networks. In summary, we see that, once the networks are trained, the method needs only 2.8 milliseconds to generate one shape. Therefore, it is clear enough that our method would allow for real-time exploration of the shape subspaces.

Table 4.3: Time taken for training the datasets.

Steps (Training)	Training time for chairs	Training time for lamps
LRN	721.27s	178.69s
SSN	1125.96s	584.26s

Chapter 5

Conclusion, limitations, and future work

5.1 Conclusion

We introduced a method to facilitate the exploration of latent spaces for the synthesis of 3D shapes with deep neural networks. We demonstrated that the combination of semantic labels and parameter selection enables users to explore subspaces of shapes constrained by the labels, effectively allowing the user to create a variety of shapes with the specified attributes. This is made possible by the combination of a label regression network that learns distributions of latent vectors conditioned on the labels, and a synthesis network which translates sampled latent vectors into 3D shapes. We showed examples of shapes generated with this method for two sample datasets. In summary, out method allows novice users to create shapes in an interactive manner without having to model the fine details of the shapes. In contrast to retrieving and reusing shapes or parts from existing databases, our approach can synthesize, or at least interpolate, new shapes by leveraging deep neural network models.

In terms of deep learning models, we introduced a label regression network (LRN) and showed that it is able to map semantic keywords into distributions of latent vectors. Our shape synthesis network follows the architecture of an autoencoder (AE). Additionally, we also compared the shapes synthesized by the AE to shapes synthesized by state-of-the art networks (VAE, AE-GAN). We found that these networks are comparable in performance, although these experiments may not be fully conclusive given the size, variation and categories of our datasets. Moreover, we introduced a labeling for the shapes of two datasets. The labeling scheme is simple yet diverse enough to describe the structure of the different shapes.

To conclude, we introduced a method that can inspire modeling, especially for

individuals who have little knowledge of 3D modeling tools or aristic skills. Although our generated shapes are not refined enough for use in high-end applications such as video games or movies, the models are reasonable enough to inspire the modeling of new shapes. In addition, the method enables a user to explore a shape space and easily generate a large variety of shapes that can be modified with other tools.

5.2 Limitations and future work

Although we demonstrated that the mappings learned by the two networks are sound and lead to a meaningful user experience and results, our work has certain limitations. Addressing these limitations could lead to improvements of the method.

First, we evaluated our approach on two sets of shapes (chairs and lamps) which are small in size (400 chairs and 100 lamps), hence variation is limited. Especially, for the 100 lamps, the variation among the dataset is not so significant, and thus the exploration of the latent space is limited. Although chairs and lamps are some of the categories of man-made shapes with the most structural and geometric variability, evaluating the method on additional categories and larger shape datasets could provide more insight on the general applicability of the method. A second limitation of our method is that it involves manual selection of keywords to describe shape features. Selecting keywords for large datasets may be unfeasible. Thus, crowdsourcing approaches might be useful for labeling additional datasets.

Moreover, the visual quality of the results, although in line with the results of previous volumetric synthesis networks [73], including GANs [70], are still not satisfactory enough as final products and yield only prototypes that provide inspiration to artist modelers. Given that the visual quality of the results depends mostly on the resolution of the volume representation of the input meshes, it would be interesting to combine our approach with more sophisticated networks, such as those that learn implicit representations of shapes [51] or part-based representations [46], to synthesize more aesthetically-pleasing shapes. In addition, when synthesizing a shape from a given latent vector, we could perform a "snapping" of the vector onto the latent space [46], to obtain a shape that is part of the learned manifold, possibly improving the visual quality of the synthesized shape. Another possible improvement is to post-process the results to improve the quality of the models, such as removing disconnected voxels or smoothing the surface of the generated shapes.

Regarding the technical components of the method, currently the LRN maps a set of labels to a single Gaussian distribution. In the future, we could instead learn mappings from labels to a mixture of distributions, which can be enabled by methods such as mixture density networks [10]. This would enable to capture multiple modes in the latent space that relate to common labels, although then the user would need to navigate through these modes.

A qualitative evaluation of the generated shapes is important for showing the applicability and robustness of our method. In past approaches, viewpoint-dependent quality assessment methods and viewpoint-independent quality assessment methods were introduced for 3D meshes [17]. In the future, to evaluate the quality of our generated shapes in a quantitative manner, an evaluation metric such as *inception scores* or *Frechet inception distances* could be calculated for the generated shapes. Saliman et al. [57] introduced the concept of *inception score*, where a classification network is used to evaluate the quality of 2D images synthesized by generative adversarial networks. Later on, for 3D volumetric modeling and shape synthesis using deep neural network architectures, inception scores were used to evaluate the diversity and quality of generated 3D objects [34, 76].

Moreover, further experiments can be performed by varying the size of the set of semantic labels, to evaluate the scaling of the label regression network and observe whether there are any significant differences in the predicted latent vectors. In addition, constraints can be applied to ensure that the latent space is sparse when mapping the keywords to a distribution, so that the exploration requires manipulating less dimensions of the latent vectors.

List of References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. J. Guibas, "Learning representations and generative models for 3D point clouds," in *Int. Conf. on Machine Learning*, 2018.

[2] I. Alhashim, H. Li, K. Xu, J. Cao, R. Ma, and H. Zhang, "Topology-varying 3d shape creation via structural blending," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 158, 2014.

[3] M. Z. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. S. Nasrin, M. Hasan, B. C. Van Essen, A. A. Awwal, and V. K. Asari, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, 2019.

[4] D. Anguelov, P. Srinivasan, D. Koller, S. Thrun, J. Rodgers, and J. Davis, "SCAPE: shape completion and animation of people," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 24, no. 3, pp. 408–416, 2005.

[5] G. Antipov, M. Baccouche, and J.-L. Dugelay, "Face aging with conditional generative adversarial networks," in *2017 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2017, pp. 2089–2093.

[6] M. Averkiou, V. G. Kim, Y. Zheng, and N. J. Mitra, "Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis," in *Computer Graphics Forum*, vol. 33, no. 2. Wiley Online Library, 2014, pp. 125–134.

[7] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 12, pp. 2481–2495, 2017.

[8] J. Bao, D. Chen, F. Wen, H. Li, and G. Hua, "Cvae-gan: fine-grained image generation through asymmetric training," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2745–2754.

[9] D. Bau, J.-Y. Zhu, H. Strobelt, B. Zhou, J. B. Tenenbaum, W. T. Freeman, and A. Torralba, "Gan dissection: Visualizing and understanding generative adversarial networks," *arXiv preprint arXiv:1811.10597*, 2018.

[10] C. M. Bishop, "Mixture density networks," Aston University, Tech. Rep. NCRG/94/004, 1994.

[11] V. Blanz and T. Vetter, "A morphable model for the synthesis of 3D faces," in *Proc. of SIGGRAPH*, 1999, pp. 187–194.

[12] M. Bokeloh, M. Wand, V. Koltun, and H.-P. Seidel, "Pattern-aware shape deformation using sliding dockers," in *ACM Transactions on Graphics (TOG)*, vol. 30, no. 6. ACM, 2011, p. 123.

[13] M. Botsch and L. Kobbelt, "An intuitive framework for real-time freeform modeling," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 630–634, 2004.

[14] A. Brock, J. Donahue, and K. Simonyan, "Large scale gan training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[15] A. Brock, T. Lim, J. Ritchie, and N. Weston, "Generative and discriminative voxel modeling with convolutional neural networks," *arXiv:1608.04236v2*, 2016.

[16] A. Brunton, A. Salazar, T. Bolkart, and S. Wuhrer, "Statistical shape spaces for 3D data: A review," in *Handbook of Pattern Recognition and Computer Vision*. World Scientific, 2016, pp. 217–238.

[17] A. Bulbul, T. Capin, G. Lavoué, and M. Preda, "Assessing visual quality of 3-d polygonal models," *IEEE Signal Processing Magazine*, vol. 28, no. 6, pp. 80–90, 2011.

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

[19] S. Chaudhuri, E. Kalogerakis, S. Giguere, and T. Funkhouser, "Attribit: content creation with semantic attributes," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 193–202.

[20] J. X. Chen, "The evolution of computing: Alphago," *Computing in Science & Engineering*, vol. 18, no. 4, p. 4, 2016.

[21] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5939–5948.

[22] A. Dai, C. R. Qi, and M. Nießner, "Shape completion using 3D-encoder-predictor CNNs and shape synthesis," in *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, 2017.

[23] R. Dechter, *Learning while searching in constraint-satisfaction problems*. University of California, Computer Science Department, Cognitive Systems . . . , 1986.

[24] T.-D. Do, M.-T. Duong, Q.-V. Dang, and M.-H. Le, "Real-time self-driving car navigation using deep neural network," in *2018 4th International Conference on Green Technology and Sustainable Development (GTSD)*. IEEE, 2018, pp. 7–12.

[25] C. Doersch, "Tutorial on variational autoencoders," *arXiv preprint arXiv:1606.05908*, 2016.

[26] C. Farabet, C. Couprie, L. Najman, and Y. LeCun, "Learning hierarchical features for scene labeling," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1915–1929, 2012.

[27] K. Fukushima, "Neocognitron: A hierarchical neural network capable of visual pattern recognition," *Neural networks*, vol. 1, no. 2, pp. 119–130, 1988.

[28] D. Gonzalez and O. van Kaick, "3d synthesis of man-made objects based on fine-grained parts," *Computers & Graphics*, vol. 74, pp. 150–160, 2018.

[29] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[30] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry, "AtlasNet: a papier-mâché approach to learning 3D surface generation," in *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, 2018.

[31] E. Guérin, J. Digne, E. Galin, A. Peytavie, C. Wolf, B. Benes, and B. Martinez, "Interactive example-based terrain authoring with conditional generative adversarial networks," *ACM Trans. on Graphics*, vol. 36, no. 6, pp. 228:1–13, 2017.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[33] H. Huang, E. Kalogerakis, E. Yumer, and R. Mech, "Shape synthesis from sketches via procedural models and convolutional networks," *IEEE Trans. Visualization & Computer Graphics*, vol. 23, no. 8, pp. 2003–2013, 2017.

[34] W. Huang, B. Lai, W. Xu, and Z. Tu, "3d volumetric modeling with introspective neural networks," in *AAAI Conference on Artificial Intelligence*, 2019.

[35] M. F. I. Ibrahim and A. A. Al-Jumaily, "Auto-encoder based deep learning for surface electromyography signal processing," *Adv. Sci. Technol. Eng. Syst*, vol. 3, pp. 94–102, 2018.

[36] A. Jain, T. Thormählen, T. Ritschel, and H.-P. Seidel, "Exploring shape variations by 3d-model decomposition and part-based recombination," in *Computer Graphics Forum*, vol. 31, no. 2pt3. Wiley Online Library, 2012, pp. 631–640.

[37] B. Karlik and A. V. Olgac, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence and Expert Systems*, vol. 1, no. 4, pp. 111–122, 2011.

[38] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[39] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[40] J. Kocić, N. Jovičić, and V. Drndarević, "An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms," in *Sensors*, 2019.

[41] V. Kreavoy, D. Julius, and A. Sheffer, "Model composition from interchangeable components," in *15th Pacific Conference on Computer Graphics and Applications (PG'07)*. IEEE, 2007, pp. 129–138.

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

[43] M. Kuhn and K. Johnson, *Applied predictive modeling*. Springer, 2013, vol. 26.

[44] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther, "Autoencoding beyond pixels using a learned similarity metric," in *Proc. Conf. on Machine Learning*, 2016, pp. 1558–1566.

[45] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[46] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas, "GRASS: generative recursive autoencoders for shape structures," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 36, no. 4, pp. 52:1–14, 2017.

[47] J. Liu, F. Yu, and T. Funkhouser, "Interactive 3D modeling with a generative adversarial network," in *Proc. Int. Conf. on 3D Vision (3DV)*, 2017, pp. 126–134.

[48] X. Mao, C. Shen, and Y.-B. Yang, "Image restoration using very deep convolutional encoder-decoder networks with symmetric skip connections," in *Advances in neural information processing systems*, 2016, pp. 2802–2810.

[49] C. Nash and C. K. I. Williams, "The shape variational autoencoder: A deep generative model of part-segmented 3D objects," *Computer Graphics Forum (Proc. SGP)*, vol. 36, no. 5, pp. 1–12, 2017.

[50] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *arXiv preprint arXiv:1811.03378*, 2018.

[51] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "DeepSDF: learning continuous signed distance functions for shape representation," in *Proc. IEEE Conf. on Computer Vision & Pattern Recognition*, 2019.

[52] G. I. Pasko, A. A. Pasko, and T. L. Kunii, "Bounded blending for function-based shape modeling," *IEEE Computer Graphics and Applications*, vol. 25, no. 2, pp. 36–45, 2005.

[53] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *Proc. Conf. on Machine Learning*, 2016, pp. 1060–1069.

[54] B. D. Ripley and N. Hjort, *Pattern recognition and neural networks*. Cambridge university press, 1996.

[55] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[56] S. Saha. (2018) A comprehensive guide to convolutional neural networks — the eli5 way. [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[57] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in neural information processing systems*, 2016, pp. 2234–2242.

[58] N. Sedaghat and T. Brox, "Unsupervised generation of a viewpoint annotated car dataset from videos," in *IEEE International Conference on Computer Vision (ICCV)*, 2015. [Online]. Available: http://lmb.informatik.uni-freiburg.de/Publications/2015/SB15

[59] T. Silva. (2018) An intuitive introduction to generative adversarial networks (gans). [Online]. Available: https://www.freecodecamp.org/news/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394/

[60] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[61] D. Smirnov, M. Bessmeltsev, and J. Solomon, "Deep sketch-based modeling of man-made shapes," *ArXiv:1906.12337*, 2019.

[62] S. Streuber, M. A. Quiros-Ramirez, M. Q. Hill, C. A. Hahn, S. Zuffi, A. O'Toole, and M. J. Black, "Body Talk: crowdshaping realistic 3D avatars with words," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 35, no. 4, pp. 54:1–14, 2016.

[63] M. Sung, H. Su, V. G. Kim, S. Chaudhuri, and L. Guibas, "ComplementMe: weakly-supervised component suggestions for 3D modeling," *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)*, vol. 36, no. 6, pp. 226:1–12, 2017.

[64] Q. Tan, L. Gao, Y.-K. Lai, and S. Xia, "Variational autoencoders for deforming 3d mesh models," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[65] S. Verma. (2019) Understanding 1d and 3d convolution neural network — keras. [Online]. Available: https://towardsdatascience.com/understanding-1d-and-3d-convolution-neural-network-keras-9d8f76e29610

[66] P. Vincent, "A connection between score matching and denoising autoencoders," *Neural computation*, vol. 23, no. 7, pp. 1661–1674, 2011.

[67] G. Wang, H. Laga, J. Jia, N. Xie, and H. Tabia, "Statistical Modeling of the 3D Geometry and Topology of Botanical Trees," *Computer Graphics Forum (Proc. SGP)*, vol. 37, no. 5, pp. 185–198, 2018.

[68] K. Wang, M. Savva, A. X. Chang, and D. Ritchie, "Deep convolutional priors for indoor scene synthesis," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 37, no. 4, pp. 70:1–14, 2018.

[69] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-CNN: octree-based convolutional neural networks for 3D shape analysis," *ACM Trans. on Graphics (Proc. SIGGRAPH)*, vol. 36, no. 4, pp. 72:1–11, 2017.

[70] W. Wang, Q. Huang, S. You, C. Yang, and U. Neumann, "Shape inpainting using 3d generative adversarial network and recurrent convolutional networks," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2317–2325, 2017.

[71] Y. Wang, J. van de Weijer, and L. Herranz, "Mix and match networks: encoder-decoder alignment for zero-pair image translation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5467–5476.

[72] Y. Wang, S. Asafi, O. van Kaick, H. Zhang, D. Cohen-Or, and B. Chen, "Active co-analysis of a set of shapes," *ACM Trans. on Graphics (Proc. SIGGRAPH Asia)*, vol. 31, no. 6, pp. 165:1–10, 2012.

[73] J. Wu, C. Zhang, T. Xue, B. Freeman, and J. Tenenbaum, "Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling," in *Advances in neural information processing systems (NIPS)*, vol. 29, 2016.

[74] Z. Wu, X. Wang, D. Lin, D. Lischinski, D. Cohen-Or, and H. Huang, "Sagnet: structure-aware generative network for 3d-shape modeling," *ACM Transactions on Graphics (TOG)*, vol. 38, no. 4, p. 91, 2019.

[75] L. Y. Xi-Dao LUAN, Yu-Xiang XIE and L.-D. WU, "Research and development of 3d modeling," *IJCSNS International Journal of Computer Science and Network Security*, vol. 8, no. 1, 2008.

[76] J. Xie, Z. Zheng, R. Gao, W. Wang, S.-C. Zhu, and Y. Nian Wu, "Learning descriptor networks for 3d shape synthesis and analysis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8629–8638.

[77] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[78] J. Yang and G. Yang, "Modified convolutional neural network based on dropout and the stochastic gradient descent optimizer," *Algorithms*, vol. 11, no. 3, p. 28, 2018.

[79] M. E. Yumer, S. Chaudhuri, J. K. Hodgins, and L. B. Kara, "Semantic shape editing using deformation handles," *ACM Transactions on Graphics (TOG)*, vol. 34, no. 4, p. 86, 2015.

[80] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[81] Z. Zhu, X. Wang, S. Bai, C. Yao, and X. Bai, "Deep learning representation using autoencoder for 3d shape retrieval," *Neurocomputing*, vol. 204, pp. 41–50, 2016.

Appendix A

Dataset labeling

Figure A.1: Some example chair shapes as mesh

We show some example mesh representation of our dataset in this section. We also present a segment of our excel sheet where we show the input labels for some

Figure A.2: Some example lamp shapes as mesh

chair and lamp shapes presented in figure A.1 and A.2

Figure A.1 and A.2 shows some example mesh of chairs and lamps. And figure A.3, A.4 denotes the labeling for four shapes presented in the first row of A.1 and A.2. Though we follow the voxelized input for labeling each shape but the chosen examples here in figure A.1, A.2 does not deform after the voxelization into respective dimension. Thus, the labeling (A.3, A.4) corresponds to both voxelized input and mesh representation of our dataset.

A	B	C	D	E	F	G
			shape_1	shape_2	shape_3	shape_4
Back	size	half	0	0	1	0
		full	0	1	0	1
	fill	solid	0	0	0	0
		vertical ladder	0	0	1	1
		horizontal ladder	0	1	0	0
		holes	0	0	0	1
		no	1	0	0	0
	side_view	straight	0	0	0	0
		bent	0	1	1	1
	front_view	round/less curved	0	0	0	0
		curved	0	0	0	0
		square	0	1	1	1
Seat	shape	square	1	1	1	1
		circular	0	0	0	0
Leg	number	one	0	0	0	0
		two	0	0	0	0
		three	0	0	0	0
		four	1	1	1	1
		five	0	0	0	0
	length	long	0	0	0	1
		short	1	1	1	0
	type	roller	1	0	0	1
		straight	0	1	1	0
		beam	0	0	0	0
		box	0	0	0	0

Figure A.3: Example labels of chairs presented in the first row of A.1

A	B	C	D	E	F	G
			shape_1	shape_2	shape_3	shape_4
Shade	front view	bell	1	0	0	0
		drum/cylinder	0	0	0	1
		round	0	1	0	0
		funnel	0	0	1	0
		rectangular	0	0	0	0
	top view	hole	1	1	0	0
		no hole	0	0	1	1
	fitting	wired	0	0	0	0
		empty	1	1	1	1
Body	type	pipe	1	1	1	1
		spiral	0	0	0	0
		beam	0	0	0	0
		curved	0	0	0	0
	structure	straight	1	0	0	1
		bent	0	1	1	0
		uneven	0	0	0	0
	length	small	0	0	1	1
		medium	1	0	0	0
		large	0	1	0	0
Base	type	round/hexagon	1	1	1	1
		square/rect	0	0	0	0
		no base	0	0	0	0
	connection	body tangled	0	0	0	0
		untangled	1	1	1	1

Figure A.4: Example labels of lamps presented in the first row of A.2