DEEP 3D SEMANTIC SCENE EXTRAPOLATION

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

ALI ABBASI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

AUGUST 2018

Approval of the thesis:

DEEP 3D SEMANTIC SCENE EXTRAPOLATION

submitted by **ALI ABBASI** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Halit Oğuztüzün
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Yusuf Sahillioğlu
Supervisor, **Computer Engineering Department, METU**

Assoc. Prof. Dr. Sinan Kalkan
Co-supervisor, **Computer Engineering Department, METU**

Examining Committee Members:

Assoc. Prof. Dr. Yusuf Sahillioğlu
Computer Engineering Department, METU

Assist. Prof. Dr. Ramazan Gökberk Cinbiş
Computer Engineering Department, METU

Assist. Prof. Dr. Hacer Yalım Keleş
Computer Engineering Department, Ankara University

Date: 14.08.2018

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: ALI ABBASI

Signature :

ABSTRACT

DEEP 3D SEMANTIC SCENE EXTRAPOLATION

Abbasi, Ali
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Yusuf Sahillioğlu
Co-Supervisor : Assoc. Prof. Dr. Sinan Kalkan

August 2018, 54 pages

In this thesis, we study the problem of 3D scene extrapolation with deep models. Scene extrapolation is a challenging variant of the scene completion problem, which pertains to predicting the missing part(s) of a scene. While the 3D scene completion algorithms in the literature try to fill the occluded part of a scene such as a chair behind a table, we focus on extrapolating the available half scene information to a full one, a problem that, to our knowledge, has not been studied yet. Our approaches are based on deep generative adversarial (GAN) and convolutional neural networks (CNN). As input, we take the half of 3D voxelized scenes, then our models complete the other half of scenes as output. Our baseline CNN model consists of convolutional and ReLU layers with multiple residual connections and Softmax classifier with voxel-wise cross-entropy loss function at the end. We use the baseline CNN model as the generator network in the proposed GAN model. We regularize our GAN model with a discriminator network, consisting of two internal, local-global networks to have sharper and more realistic results. Local discriminator takes only the generated part of scenes, while global discriminator network takes not only the generated part but also the first real part to distinguish between real and fake scenes. Using the CNN model we also propose a hybrid model, which takes the top view projection of input scene in parallel with the 3D input. We train and evaluate our models on the synthetic 3D SUNCG dataset. We show that our trained networks can predict the other half of the scenes, and complete the objects correctly with suitable lengths. With a discussion on the challenges, we propose scene extrapolation as a challenging testbed for future research in deep learning.

Keywords: 3D scenes, Deep Learning, Scene Extrapolation, Convolutional Neural Network, Generative Adversarial Network.

ÖZ

DERİN 3B SEMANTİK SAHNE EKSTRAPOLASYONU

Abbasi, Ali
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Yusuf Sahillioğlu
Ortak Tez Yöneticisi : Doç. Dr. Sinan Kalkan

Ağustos 2018 , 54 sayfa

Bu tezde, derin modellerle 3 boyutlu sahne çıkarımı problemini inceliyoruz. Sahne çıkarımı, bir sahnenin eksik kısımlarını tahmin etmekle ilgili sahne tamamlama probleminin zorlu bir çeşididir. Literatürdeki 3B sahne tamamlama algoritmaları, bir masanın arkasındaki bir sandalye gibi bir sahnenin tıkanmış kısmını doldurmaya çalışsa da, mevcut yarım sahne bilgisini tam olarak hesaplamaya odaklanıyoruz. Yaklaşımlarımız, çekişmeli üretici ağlar (GAN) ve konvolüsyonel nöral ağlara (CNN) dayanmaktadır. Giridi olarak 3B vokselli sahnelerin yarısını alıyoruz, daha sonra modellerimiz sahnenin diğer yarısını çıktı olarak tamamlıyor. Temel CNN modelimiz, çoklu artık bağlantılara sahip olan konvolüsyon ve ReLU katmanlarından ve sonunda voxel-based cross-entropy kayıp fonksiyonuna sahip ve Softmax sınıflandırıcısından oluşmaktadır. Önerilen GAN modelinde temel CNN modelini jeneratör ağı olarak kullanıyoruz. GAN modelimizi, daha net ve daha gerçekçi sonuçlara sahip iki yerel, yerel-küresel ağdan oluşan bir ayrımcı (Discriminator) ağı ile düzenliyoruz. Yerel ayırımcı sadece sahnelerin oluşturulmuş kısmını alırken, küresel ayrımcı ağı sadece üretilen kısmı değil, aynı zamanda gerçek ve sahte sahneleri birbirinden ayıran ilk gerçek parçayı da alır. CNN modelini kullanarak, giriş sahnesinin üstten görünüş projeksiyonunu 3D girişe paralel olarak çeken bir karma model önermekteyiz. Modellerimizi sentetik 3D SUNCG veri kümesinde eğitiyor ve değerlendiriyoruz. Eğitimli ağlarımızın sahnelerin diğer yarısını tahmin edebildiğini ve objeleri uygun uzunluklarla doğru şekilde tamamlayabildiğini gösteriyoruz. Zorluklarla ilgili bir tartışma ile, derinlemesine öğrenmede gelecekteki araştırmalar için zorlu bir test yatağı olarak sahnenin ekstrapolasyonunu öneriyoruz.

Anahtar Kelimeler: 3D sahneler, Derin Öğrenme, Sahne Ekstrapolasyonu, Konvolüsyel Sinir Ağı, Çekişmeli Üretici Ağlar.

ACKNOWLEDGMENTS

Firstly, I would like to thank my thesis advisor Assoc. Prof. Dr. Yusuf Sahillioğlu and co-advisor Assoc. Prof. Dr. Sinan Kalkan for their enthusiasm, guidance and continuous support.

Besides, I would like to thank my dear friends for their support, help and inspired me during my thesis study.

Finally, I would like to express my deepest thanks and appreciations to my family.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

3D	Three Dimensional
CNN	Convolutional Neural Network
GAN	Generative Adversarial Network
SVM	Support Vector Machine
MLP	Multilayer Perceptron
GPU	Graphical Processing Unit
ANN	Artificial Neural Network
LMS	Least Mean Square
SGD	Stochastic Gradient Descent
CE	Cross Entropy

CHAPTER 1

INTRODUCTION

1.1 Problem Definition

Thanks to the availability of consumer-level 3D data acquisition sensors such as Microsoft Kinect or Asus Xtion Live, it is now quite easy to build digital copies of the environments or the objects. Regardless of its rapid development, some problems will always remain in the raw output of such sensing technologies. The notorious example is the missing data artifact which emerges due to self-occlusion, occlusion by other objects, or insufficient coverage of the viewing frustum of the device. While the occlusion-related problems have been addressed extensively in the literature under the *shape completion* or *scene completion* problems, the coverage-related version is, to our knowledge, not studied yet. In this study, we address this new problem that we cast as *scene extrapolation*. Specifically, given the first half of the scene, we extrapolate it to recover the second one. This is a valid scenario as the 3D sensor may not be able to capture the other half due to, for instance, physical limitations. This is also a more challenging problem than the completion task which is able to use the available surroundings of the occluded holes to be filled, an information that is unavailable to our extrapolation framework. Our problem is also fundamentally different from the family of *shape or scene synthesis* problems which aims to reconstruct the target based on textual or geometric descriptions such as captions or 3D point clouds (see Figure 1.1).

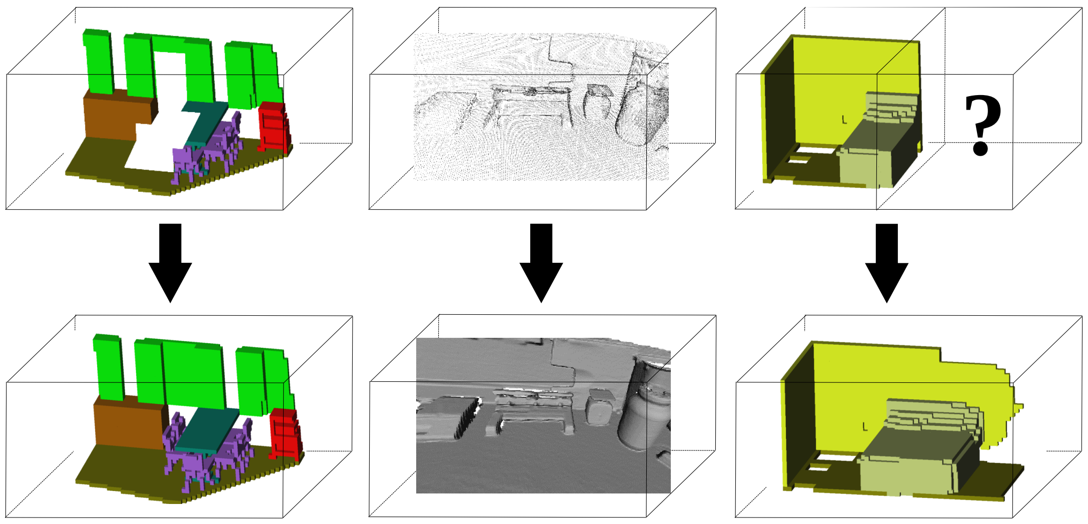


Figure 1.1: An illustration of the relevant problems. (Left) Scene completion aims at estimating the missing hole in the middle of scene. (Center) Scene synthesis constructs 3D scenes from point clouds. (Right) Scene extrapolation estimates the unseen part of the environment from the seen scene (the problem addressed in this study).

1.2 Scope of Thesis

In this thesis we study the problem of 3D scene extrapolation with deep generative adversarial and Convolutional Neural Networks (CNNs). The dataset we use [53] is voxelized 3D indoor scenes with object-labeling at voxel level. Our proposed models take the first half of each 3D scene as input, and complete (estimate) the other side of it. We design a fully CNN as our baseline model. Then, we solve the same problem with a Generative Adversarial Network (GAN) model by using the same architecture as the generator. In the proposed GAN model, the generator uses the first half of scene to generate second half, and like other GAN models, a discriminator is trained to distinguish fake (generated) scenes from real ones. We use voxel-wise Softmax cross-entropy loss function in the generator and GAN loss [18] to train our models.

1.3 Contribution

The main contributions of this thesis can be described as follows. First of all, we introduce a novel problem different than other completion tasks, namely the extrapolation of a half scene into a full one. Secondly, we propose a deep CNN model that estimates the unseen half of the scene given the visible part as its input. Thirdly, taking the CNN model as a baseline, we extend it into a GAN model with two local-global discriminator network to handle this problem. Also taking the 2D top view of input scene as parallel we build a hybrid model, which combines 2D and 3D generated scenes to predict the unseen half of input scene. A part of the thesis has been disseminated in the following paper: “Ali Abbasi, Sinan Kalkan, Yusuf Sahillioğlu, Deep 3D Semantic Scene Extrapolation, The Visual Computer Journal, 2018”.

1.4 Outline

This thesis document, which focuses on the 3D scene extrapolation task using deep models, is divided into 5 main chapters:

- Introduction
- Background
- 3D Scene Extrapolation with Deep Models
- Experiments
- Conclusion

The first chapter is designed to introduce the topic and purpose of the study. Chapter 2 is allocated to the literature survey in detailed way and also to the theoretical explanations and mathematical form of the deep learning. The third chapter explains the methods and models we use for scene extrapolation task. In chapter 4, experimental procedures, comparative results of proposed method, and detailed discussions about the study and results are presented. Conclusion and future work can be monitored in chapter 5.

CHAPTER 2

BACKGROUND

Missing data can be completed in various ways depending on the size of the misses. We broadly categorize these methods as rule based and learning based approaches.

2.1 Rule Based Missing Data Completion Approaches

A set of rule based approaches utilize an “as smooth as possible” filling strategy for the completion of sufficiently small missing parts, or holes [38, 70, 61]. When the holes enlarge to a certain extent, context-based copy-paste techniques yield more accurate results than smooth filling techniques do. In this line of research, convenient patches from other areas of the same object [19, 42, 51] or from a database of similar objects [16, 32, 36] are copied to the missing parts. Although these methods generally arise in the context of 3D shape completion, appropriate modifications enable image [2, 21, 33] and video [25, 58] completion as well. A common limitation to this set of works is the assumption on the existence of the missing part in the context. Besides, these algorithms may generate patches inconsistent with the original geometry as they may not be able to take full advantage of the global information. Although semi-automatic methods that operate under the guidance of a user [20, 65] may improve the completion results, a more principled solution is based on supervised machine learning techniques, as we discuss in the sequel.

A different rule-based pipeline is based on the family of iterative closest point algorithms [5, 6, 43, 64]. These algorithms are applicable when there are multiple sets of partial data available at arbitrary orientations, a case that typically emerges as a result of 3D scanning from different views [40]. By alternating between closest point correspondence and rigid transformation optimization, these methods unify all partial scans into one full scan, hence completing the geometry. Requiring access to other complementary partial data is a fundamental drawback on these works.

2.2 Learning Based Missing Data Completion Approaches

Scene completion, posed either on 2D image domain or 3D Euclidean domain, is more challenging than the shape completion case discussed thus far. The difficulty mainly stems from the fact that it is an inherently ill-posed problem as there are infinitely many geometric pieces that look accurate, e.g., a chair as well as a garbage bin is equally likely to be occluded by a table. A rule based approach such as [71] should, therefore, be replaced with a machine learning based scheme as the latter imitates the thinking mechanism of a human for whom the completion task is trivial, e.g., we immediately decide the highly occluded component based on our years of natural training. To this end, [53] recently proposed a 3D convolutional network that takes a single depth image as input and outputs volumetric occupancy and semantic labels for all voxels in the camera view frustum. Although this is the closest work to our paper, we also deal with the voxels that are not in the view frustum, hence the even more challenging task of scene extrapolation instead of completion, or interpolation. Similar to [53], unobserved geometry is predicted by [13] using structured random forest learning that is able to handle the large dimensionality of the output space.

Recent deep generative models have shown promising success on completion of images at the expense of significant training times, e.g., in the order of months [26, 29, 37, 66]. There are also promising 3D shape completion results based on deep neural nets [8, 63], which again suffer from the tedious training stage.

In the shape synthesis problem, given a description of a shape or a scene to be synthesized, it is aimed to reconstruct the query in the form of an image [68, 69], video [55], or surface embedded in 3D [14, 24, 28, 50]. There are many sources for the descriptive information ranging from a verbal description of the scene to 3D point cloud of a shape. Synthesis can be performed by either generating the objects [59] or retrieving them from a database [15, 30].

In deep learning, there are many generative models that can scale well at large spaces such as natural images. The prominent models include Generative Adversarial Networks (GANs) [18], variational autoencoders [45], and pixel-level convolutional or recurrent networks [47]. In addition to modeling the extrapolation problem as a voxel-wise classification problem, we chose GANs among the generative models since (i) they are easier to construct and train, (ii) they are computationally cheaper during training and inference, and (iii) they yield better and sharper results.

As a summary, completion and synthesis of shape, scene, images or videos have been extensively studied in the literature. However, scene extrapolation is a new problem that we propose to solve in this thesis using deep networks.

2.3 Deep Learning

Artificial neural network and its structure has inspired researchers for developing the concept of 'Deep Learning'. Deep learning has appeared in the last decade as a research field in machine learning area [4, 23]. The name 'deep' came from its deep and hierarchical structure. Shallow architectures include support vector machines (SVM) recently used in machine learning, signal processing, and other learning-based areas. The data amount has been grown and become more complicated and demanding in recent yeras, therefore some real-world applications and problems can not be solved by shallow structures. Studies in deep learning has influenced areas in the literature such as data science, robotics, computer vision and graphics, speech recognition, natural language processing, audio processing and medical areas.

'Deep learning' is a branch of machine learning which has complex layers for analyzing and processing the raw data. It includes the classification, feature extraction (or selection), data generation and transformation in both supervised and unsupervised approaches. These deep structures even can fuse any sorts of detailed and complicated features.

The popularity of deep learning in recent years has rised due to some reasons which are explained as below. Firstly, graphical processing units (GPU) comes in this field and make the processing of large amount of data in a parallel way. The mathematical matrix operations are more convenient to do with GPUs, so the processing of large size of training data is no longer difficult. The other reason for the popularity of deep learning is the advantage of having the feature extraction without need of defining them manually [9, 52]. In the machine learning algorithms, there are some needs of defining kernels and cost functions by a good knowledge and powerful analysis whereas working with deep learning helps us to build a model which can select the useful features to learn from low to high level [3].

The power of the model is affected by the number of hidden layers and their neurons. By increasing the capacity (deeper and wider networks) of the model, the time and resource complexity will be rise. Having more capacity in models leads to have better learning process; however, by adding more hidden layers and neurons it will not always work efficiently.

2.3.1 Deep Feedforward Networks

Deep feedforward neural networks are the typical deep learning models. The goal of a feedforward network is to approximate some functions. A feedforward network defines a mapping and learns the value of the parameters that result in the best function approximation. These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used for defining f, and finally to the output y. These networks consist of some layers like chains, which are the most commonly used structures of neural networks. The length of this chain defines the depth of the model. The final layer of a feedforward network is called the output layer. During the training of neural networks, we drive the actual function $f(x)$ to match the approximation function. Each example x has a label of $y \approx f^{\star}(x)$. The training samples specify what the output layer must produce for each x, which should be a value that is close to y. The reaction of other layers except the output layer are not directly determined by the training samples. The learning algorithm must decide how to use those layers to produce the desired output, but the training data do not say that what each individual layer does. Instead, the learning algorithm must decide how to use these layers to implement an approximation of f^* in a best way, since the training samples do not show the desired output for each layer, these layers are called "hidden layers".

2.3.1.1 Gradient-Based Learning

Building and training a neural network is like to train other machine learning algorithms with gradient descent. The training in neural networks usually iterative gradient-based optimization. On the other hand, in training linear models, we use a convex optimization algorithm, which guarantees convergence. In optimization of a convex function, any initial parameters lead us to convergence. However, in a stochastic gradient descent algorithm for non-convex loss functions there is no guarantee convergence and it has strong dependency on the initialization of the parameters. It is necessary to initialize the parameters to small random value for feedforward networks. We must choose a cost function for optimizing the output of the model to train the neural network with gradient-based learning.

Cost Function The cost function is usually defined as a term that relates the network output and expected target values and it estimates a penalty for the network outputs. The cost function is an important point in designing and training a deep neural network.

Activation Function The activation function is usually a "non-linear transformation" which transforms the output of each neuron and makes it as an input to the next layer neurons. Without activation function, we will have the simple linear transformation in networks. There are some popular activation functions in neural networks such as Sigmoid, Tanh, ReLU, Leaky ReLU, and Softmax.

Backpropagation The most common training method for the neural network is the backpropagation algorithm. One of the essential problems in training neural network is setting the weight and biase parameters. Backpropagation method tries to find optimal weights based on training samples and desired target values. Difference between the desired and actual output values is used in the least mean square (LMS) error calculation which is the main input of the backpropagation algorithm. LMS training error on a training set is the summation of squared difference between the actual and the desired output unit [12].

LMS training error is formulated as:

$$J(w) = \frac{1}{2}\sum_{c}^{k=1}(t_k - z_k)^2, \tag{2.1}$$

where t is the desired output and z is actual output vectors of length c, while w is the weights of the network. These weights are randomly initialized values which are updated consecutively by the gradient descent algorithm to reduce the error. Learning rate defines the amount of the change in weights. The most common and useful neural network training optimization methods are batch and stochastic training. In stochastic training, the training patterns are selected from training samples randomly and with the help of gradient descent algorithm, weights are updated gradually. However, in batch training, all of the training samples are used in the learning process. Lastly, a stopping criterion should be defined in order to stop the training process automatically.

2.3.2 Overfitting and Regularization in Deep Learning

The main issue in the machine learning algorithm design is how to make it perform well not only on the training data but also on the unseen test data. Many methods are used in machine learning to reduce the test error. These methods are known as regularization. There are many forms of regularization available for deep learning. Furthermore, developing the effective regularization strategies has been one of the research areas in this field.

2.3.2.1 L_2 and L_1 Regularizations

One of the most commonly used regularization terms is L_2. This regularization forces the weights to get closer to zero by adding a regularization term $\delta(\theta) = \frac{1}{2}\|w\|^2$ to the objective function. The idea behind L_2 regularization is to use distributed weights vector rather than peaky vectors. This regularization encourages the network to use all of its inputs equally rather than using some of them more than other weights.

L_1 regularization forces the weights to become sparse in optimization, which causes them go very close to zero. In fact, with this regularization, the network uses a subset of their most critical inputs and their weights since most of the parameters are becoming very close to zero and having very small influence input. Practically, using the L_2 regularization gives better performance compare to L_1, except in the case of explicit feature selection.

2.3.2.2 Dataset Augmentation

The domain specific transformation for expanding the training dataset synthetically is called dataset augmentation [10]. In other words, it means increasing the number of data points. Theoretically, in machine learning algorithms, the more data for training phase will result in the better model. In some cases, the process of collecting the real data such as image, text, video and, etc., is very costly in terms of the time and other resources. So, we need to execute some methods to augment the existing data to increase the dataset size. There are some ways to do dataset augmentation for instance in image data, we can rotate, shift, make blur, add some noise, change lighting condition and having different subsamples of an image.

2.3.2.3 Early Stopping

Training a deep model needs to play with some hyperparameters and finds the most suited one for the network. One of these hyperparameters is the number of training epochs. One epoch means one pass of the full training set. In some cases, using more epochs for training causes in over-fitting and few epochs maybe results in under-fitting (the network does not learn very well). The early stopping is an approach to determine in which epoch we should stop the training process. In this approach, we start training the network on training data and after each epoch evaluate the network on validation data. We will stop where the network performance on the validation set gets worse.

2.3.2.4 Dropout

Deep neural networks are powerful models, however, over-fitting is a big problem in the training of a deep model. Mostly, networks with higher capacity have a large number of neurons, and this causes the difficulty of dealing with the over-fitting problem. Dropout comes to end this issue by randomly dropping the units with their connection during training from the network. This technique stops the units in the network from a mutual adaption. Dropout improves the performance of the network in comparison with other regularization methods [54].

2.3.3 Optimization for Training Deep Models

Deep learning and optimization are wraps in each other in many grounds. The most difficult optimization in deep learning is training the neural network. It may take months to find an optimized solution for a neural network. In the optimization problems, the aim is to find the parameters θ in the network which decreases the loss function, $J(\theta)$. The optimization process in deep model trainings is different than classical optimization methods. In the machine learning problems, we usually act incidentally. In the most cases with the machine and deep learning problems, we concern about performance measure P, which is defined with respect to the validation set and in the next step we try to optimize P. With decreasing the loss function $J(\theta)$ we expect to see an improvement in P. This is what has paradox with pure optimization, which minimizes the J in an indirect way.

Batch and Minibatch Trainings The algorithms for optimization which use all the training set are batch methods because they fit the whole training data in a batch. This definition may also be confusing because of the term “batch”, which also used for explaining the minibatch training process. It is common to use the term of “batch size” to show the size of minibatch.

2.3.3.1 Challenges in Deep Neural Networks Optimization

Typically, optimization is a very complicated problem. Deep learning tries to reduce this complexity by designing a cost or objective function with some limitations which force it to be a convex function.

Local Minima The biggest advantage of a convex optimization problem is that it can be converted to the problem of finding a local minimum. In a way, any local minimum also can be a global minimum.

2.3.3.2 Stochastic Gradient Descent

The most common algorithm in optimization is stochastic gradient descent (SGD). Algorithm 1 shows the SGD and how it calculates the downhill gradient.

Algorithm 1 Stochastic Gradient Descent (SGD) [17]

```
Require: Learning rate $\epsilon_k$.
Require: Initialized Parameter $\theta$.
  while not reach stopping point do
      Take a minibatch of $m$ from the training samples with their labels $y^{(i)}$.
      Calculate the gradient: $\hat{g} \leftarrow +\frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)};\theta), y^{(i)})$
      Update: $\theta \leftarrow \theta - \epsilon\hat{g}$
  end while
```

The critical parameter in SGD algorithm is the learning rate. Normally the SGD has a fixed learning rate ϵ, but in practice, it is important to have decreasing learning rate over training time. The ϵ_k shows learning rate at iteration k.

2.3.3.3 Parameter Initialization

Since the training deep neural networks is a complicated task, the most models and networks are influenced by their way of initialization parameters. The initial values for parameters can determine the convergence or failure of the model. Even with the converged learning process, the initialization determines whether it converges with low or high cost. The usual way to initialize all the weights in the model to values drawn randomly from a Gaussian distribution. The measure of distribution for initialization has the reasonable influence on the progress of optimization and the capability of the model to generalize.

2.3.3.4 Adaptive Learning Rates Algorithms

The researchers in deep learning and neural network area found that the learning rate is the most important hyperparameter to set because it has the significant influence on the training for convergence and network performance. Recently, some methods has been created which have adaptive learning rate parameter and they are called incremental methods. There are three methods that are explained as below:

AdaGrad This method is an adaptive learning rate approach, proposed by [11]. Consider dx as gradient and ssg as sum of squared gradient per parameter,

$$ssg = \sum_{i}^{w} dx^2. \tag{2.2}$$

The ssg will be used as normalized parameter in each update step. The AdaGrad approach increases the learning rate for parameters with less effects and decreases for parameters which receive high gradients. Assume w as vector parameters so we have element-wise update rule as below:

$$w = w - lr \times \frac{dx}{\sqrt{ssg} + e}, \tag{2.3}$$

where lr stands for learning rate and e shows the smoothing term performing division by zero avoids. One of the disadvantages in this method for deep models is stopping the learning too early because of monotonic learning rate.

RMSProp This adaptive learning rate algorithm solves the problem of AdaGrad method and monotonically decreases the learning rate [56]. In fact adding a decay rate dr to the ssg in AdaGrad, to make it moving, instead of keeping it uniform. So we have ssg as follow:

$$ssg = dr \times ssg + (1 - dr) \times dx^2, \tag{2.4}$$

and update the parameters like formula (2.3). In this method the ssg is leaky and dr is a hyperparameter with usual values of 0.9, 0.99 and 0.999. Unlike the AdaGrad, updated value with this algorithm does not get uniformly smaller. So RMSProp adjusts the learning rate for each parameter based on the value of its gradient.

Adam Proposed by [31], Adam acts like RMSProp optimizer with momentum. This optimizer acts as momentum and preserves an exponentially decaying average of past gradients m_t. while momentum is like a ball going down on a surface with slope, the Adam optimizer acts like a rough weighty ball. While m_t and v_t show the past and past squared gradients respectively, decaying average is calculated as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t, \tag{2.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2. \tag{2.6}$$

Now the m_t and v_t are the approximation of the first and second moments of gradients (mean and the variance respectively). Since the m_t and v_t are initialized to 0, usually in initial steps they are going towards zero. And this happens mainly when decay rates are small (β_1 and β_2). The Adam optimizer prevents this problem by calculating m_t and v_t as:

$$\hat{m} = \frac{m_t}{1 - \beta_1^t}, \tag{2.7}$$

$$\hat{v} = \frac{v_t}{1 - \beta_2^t}. \tag{2.8}$$

Finally they use the following equation to update parameters and suggest default values of 0.9, 0.999 and 10^{-8} for β_1, β_2 and ϵ respectively.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t. \tag{2.9}$$

2.3.4 Convolutional Neural Networks

Convolutional neural network (CNN) is an instance of deep learning which has the considerable enhancement in image, video, speech and audio processing. This kind of models are made up some sequential layers like convolution, pooling and non-linearity [7]. This idea of CNN was presented in the 1960s, however, its preparing times were too high to make this an appealing and helpful method. In recent times, with the presentation of GPU in video cards, training these models are now much easier and bring its worth in fields of computer vision. CNN models learn features from input data layer by layer, as going deeper in the layers, the features are captured from low-level to high level.

2.3.4.1 Convolution Operation

The CNN models are different from the normal ANNs where the input image to the CNN model is divided into small equally sized tiles and features extracted from these tiles. These features are extracted by the filters which are learnable parameters. These filters are learned by doing the convolution operation between image and filter values. The convolution operation is sum of element-wise multiplication between two matrices (images and filter) in the form of an integer which is a single element of result matrix. This filter called convolution filter and has the parameter numbers as filter size. These filters are trying to learn distinct feature in images. In the first layers of the model, the convolution filters detect low-level features such as edges and corners, while going deeper the filters can fetch high-level characteristic with semantic meaning. Convolution filters are matrices of pixel values with defined size and depth. These values in the filters are initialized randomly, however, their values can be learned and updated automatically in some dimension. The value updating process is done by backpropagation algorithm [57].

2.3.4.2 Pooling Layer

The pooling layer is used for decreasing the size of the output matrix from the convolution layer. So we use an operation called max-pooling. The max-pooling finds the max value inside each grid tile of the input matrix. The pooling layer has two main effects in CNN models. 1) **Dimension Reduction.** Since the pooling layer reduces the size of the input image, it takes the data into low dimension space and preserve more global features. Also the pooling layer with reducing dimension avoids the curse of dimensionality. 2) **Position Invariance**. The pooling layer extracts the maximum value from a given region, irrespective of the feature value position. This maximum value would be from any positions inside the region and does not capture the position of the max value. Therefore it provides positional invariant feature extraction [34].

2.3.5 Generative Adversarial Networks

The main idea of GAN is first introduced in 2014 by Ian Goodfellow [18]. It has two models that one of them generates samples by taking noise as input which is called generator and the other one receives two samples from the generator and the training data. The second part is called discriminator which tries distinguish the fake data from the real one. Generator and discriminator models are trained together to learn to generate real samples and distinguish generated data from real data. The general structure of GAN is represented in 2.1.

The differences between generative and discriminative models are as below: The discriminative model is related to conditional distribution so that the model learns the function and map the real data to some desirable output class. On the other hand, the generative model is related to the joint probability which uses input data and labels. The generative models can be used for classification using Bayes rule and can be used for creating new samples.

2.3.5.1 Formal Representation

Assume G is the network which is used in generator part $G(z, \theta_1)$. The role of the generator is mapping noisy inputs z to the desired data space x. The other network, $D(x, \theta_2)$ shows the discriminator model and the outputs are the probability which belong to the real data; θ_i in both networks represents the weights of the neural networks. These weights in discriminator try to update themselves to maximize the probability of real data while minimizing the probability for fake data. The loss function maximizes $D(x)$ function and minimizes $D(G(z))$. Whereas the generator tries to update the weights to maximize the probability of the fake data which is classified as real

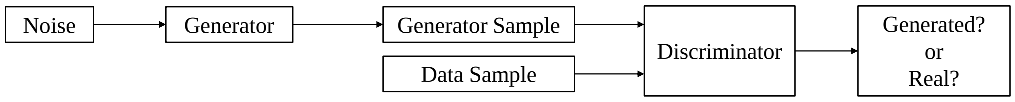


Figure 2.1: General structure of GAN models.

data and the loss function maximizes $D(G(z))$.

At the end of the training process, the generator and discriminator stops to improve themselves at a point. Then, the generator starts to generate realistic data and the discriminator can not distinguish between two types of input. The logarithm is used instead of probability in the loss functions because log loss penalizes classifiers in the case of incorrect classification. Cost function of the played game between generator and discriminator is as follow:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \quad (2.10)$$

CHAPTER 3

3D SCENE EXTRAPOLATION WITH DEEP MODELS

3.1 A Convolutional Neural Network for Scene Extrapolation (CNN-SE)

CNNs [35] are specialized neural networks with local connectivity and shared weights. With such constraints and inclusion of other operations such as pooling and non-linearity, CNNs essentially learn a hierarchical set of filters that extract the crucial information from the input for the problem at hand.

In our approach, we use a deep fully CNN model for the 3D semantic scene extrapolation task (Figure 3.2). Fully CNN models consist of only convolution and do not use pooling or fully-connected layers. Given the first half of 3D voxelized scene ($\mathbf{f}^{3D} \in \mathbb{V}^{w \times h \times d}$, where $\mathbb{V} = \{1, ..14\}$ is the set of object categories each voxel can take; w, h, d are respectively the width, the height and the depth of the scene), our task is to generate the other half of scene ($\mathbf{s}^{3D} \in \mathbb{V}^{w \times h \times d}$), which is semantically consistent with first half of scene ($\mathbf{f}^{3D}$) and conceptually realistic (e.g., objects have correct shapes, boundaries and sizes). Since our approaches work on voxelized data, we treat each 3D voxelized scene as '2D images' with multiple channels, where channels correspond to planes of the scene. We find that each voxel in a 3D scene is strongly dependent to adjacency voxels, or in the other words, each plane of voxels in a way similar to near planes. Therefore, as shown in Figure 3.1, we convert 3D voxelized scenes to 2D planes, like RGB images, instead we have 42 channels (42 layers of voxels - i.e., w is 42) for the input scene. Our models complete the missing channels from an input 3D scene.

We have experimented with many architectures and structures, including pooling lay-

Table 3.1: The architecture details of our CNN model. The layer type is shown in the 'Layer' column; 'Conv.' stands for convolution, and 'Res' for residual block, which is detailed in (b). 'R' shows the ReLU activation function in the second column. The last layer is a Softmax classifier.

(a) CNN model architecture.

Layer	Activation	Batch Norm.	Kernel	Stride	Dilation	Outputs
Conv.	-	x	7×7	1	1	64
Res1. $\times 4$	-	-	-	-	-	64
Res2. $\times 2$	-	-	-	-	-	64
Conv.	-	x	1×1	1	1	14×42

(b) Residual blocks internal architecture. Res2. uses dilation 2 and 4 respectively in two blocks.

	Layer	Activation	Batch Norm.	Kernel	Stride	Dilation	Outputs
Res1.	-	R	x	-	-	-	-
	Conv.	R	x	3×3	1	1	64
	Conv.	R	x	5×5	1	1	64
	Conv.	-	-	3×3	1	1	64
Res2.	-	R	x	-	-	-	-
	Conv.	R	x	3×3	1	1	64
	D.Conv.	R	x	1×1	1	2, 4	64
	Conv.	-	-	3×3	1	1	64

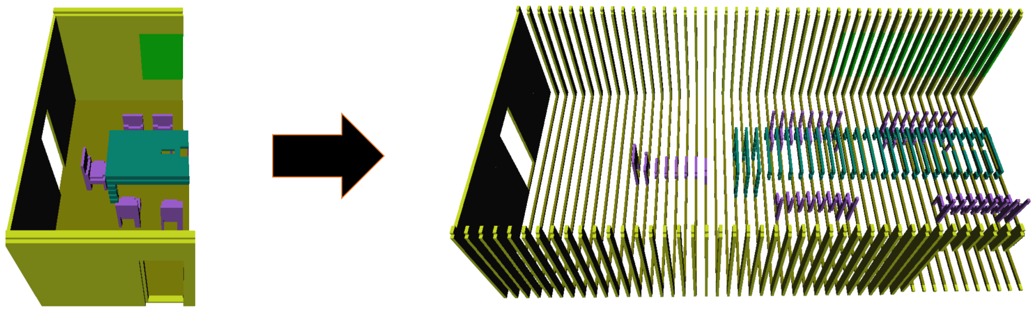


Figure 3.1: We convert 3D voxelized scenes to 2D planes, like RGB images, instead we have 42 channels for the input scene. Our models complete the missing channels from an input 3D scene.

ers, strided and deconvolution in bottleneck architectures, and dilated convolutions [67]. However, we found that a network with only convolution operations with stride 1 yielded better results. Therefore, we design our architecture as convolutional layers with stride 1 and same padding size, see Figure 3.2.

Table 3.1 lists the details of the CNN-SE model. The fully CNN model takes input scene and first applies a 7×7 convolutions, then the architecture continues with 6 residual blocks [22]. Each residual block contains 3 layers of ReLU non-linearity (defined as relu$(x) = \max(0, x)$; see [46]) followed by convolution after each. It finishes with a 1×1 convolutional layers, and Softmax classifier with voxel-wise cross-entropy loss, L_{CE}, between the generated and the ground truth voxels as follows:

$$L_{CE} = \frac{1}{m} \sum_{i} H(\mathbf{t}_i^{3D}, \mathbf{s}_i^{3D}), \tag{3.1}$$

where m is the batch size; $\mathbf{s}_i^{3D}$ is the extrapolated scene for the i^{th} input, and t_i^{3D} is the corresponding ground truth. Cross-entropy (H) between two vectors (or serialized matrices) is defined as follows:

$$H(\mathbf{p}, \mathbf{q}) = -\sum_{i} p_i \log(q_i). \tag{3.2}$$

We also use a smoothness penalty term, L_S, to make the network produce smoother results by enforcing consistency between each generated plane $\mathbf{s}_{ij}^p$ (i.e., the generated j^{th} voxel plane for the i^{th} input instance) and its next plane in the ground truth, $\mathbf{t}_{i(j+1)}^p$:

$$L_S = \frac{1}{m} \sum_{i,j} H\left(\mathbf{t}_{i(j+1)}^p, \mathbf{s}_{ij}^p\right), \tag{3.3}$$

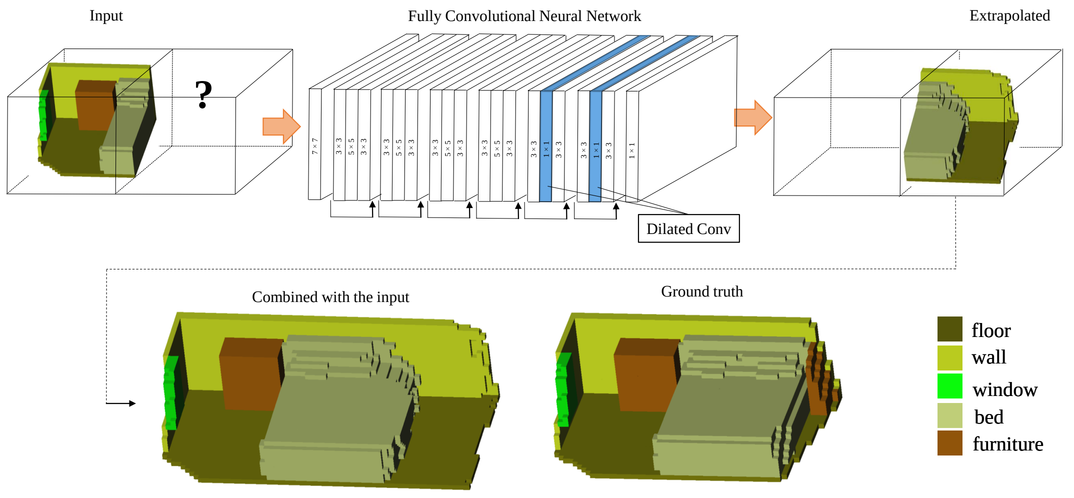


Figure 3.2: The scene extrapolation problem addressed in the study, and how we use deep learning to solve it. Our deep CNN model takes in a half of the 3D voxelized scene as input and after applying some convolution operations, generates the full scene as output. The numbers inside the layers indicate the kernel sizes. The architecture consists of several residual connections, as shown by the arrows.

where m is again the batch size.

Moreover, against overfitting, we add L_2 regularization on the weights:

$$L_2 = \frac{\lambda}{2} \sum_i w_i^2, \tag{3.4}$$

$$L_f = -\sum_i (1 - p_i)^{\gamma} p_i \log(q_i) \tag{3.5}$$

where λ is a constant, controlling the strength of regularization (λ is tuned to 0.001), and w_i is a weight.

In addition the focal loss L_f [39] is added to have faster convergence. Therefore, the total loss that we try to minimize with our CNN model is as follows:

$$L_{total} = L_{CE} + L_S + L_2 + L_f. \tag{3.6}$$

3.2 A Generative Adversarial Network for Scene Extrapolation (GAN-SE)

GANs [18] rely on interplay between two kinds of networks, a generator network and a discriminator network, which have compete against each other in a minimax game. In a GAN, the generator tries to generate novel data given its input, whereas the discriminator aims to distinguish the generated data from the real one. Generally speaking, the generator (G) maps a vector ($\mathbf{z}$) of random variables to the space of data to be generated. The discriminator, on the other hand, is trained to discriminate $\mathbf{x} \sim p_{data}(\mathbf{x})$ with label 'real' from $G(\mathbf{z})$ with label 'fake'.

We design our GAN model as a deep convolutional generative adversarial network (DCGAN) [48] enhanced with reconstruction loss for the generator to improve results – as inspired from [37, 26] – see Figure 3.3. The generator, a fully-convolutional neural network, is similar to the CNN-SE model presented in the previous section, whereas the discriminator is composed of two sub-networks, one for local and one for global processing.

Formally speaking, let $G(\mathbf{f})$ be the output of the generator network (note that our generator network does not use noise as input – we will discuss this in Section 3.2.3), where $\mathbf{f}$ is first part of scene, and $D(\mathbf{f}, \mathbf{u})$ denote the discriminator network where $\mathbf{u}$ is either a generated ($G(\mathbf{f})$) or a real scene ($\mathbf{x}$). In our case, we try to solve the following problem:

$$\min_{G} \max_{D} \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x}), \mathbf{f} \sim p_{data}(\mathbf{f}),}[\log D(\mathbf{f}, \mathbf{x})] + \mathbb{E}_{\mathbf{f} \sim p_{data}(\mathbf{f})}[\log(1 - D(\mathbf{f}, G(\mathbf{f})))], \quad (3.7)$$

where $\mathbf{x}$ is the ground truth. We have also trained the discriminator with $D(\mathbf{f}', \mathbf{x})$ (with label 'fake') and with $D(\mathbf{f}', G(\mathbf{f}))$ (with label 'fake'), where $\mathbf{f}'$ is the first part of another scene in the dataset. However, this has not improved the results.

3.2.1 The Generator (G)

The generator (G) is a fully CNN, the architecture is similar to CNN model, with the difference that G has Leaky ReLU [41] as the activation function. The input to the generator, $\mathbf{f}$, is the first half of the 3D voxelized scene, and the output is the other half,

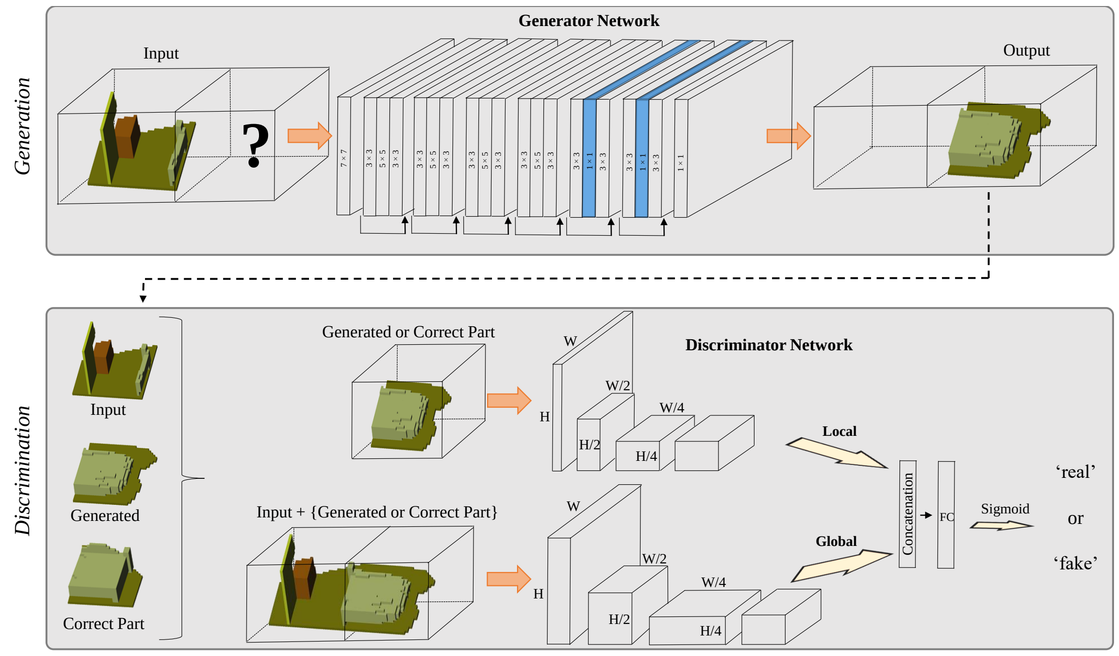


Figure 3.3: The GAN model used in our study. We feed the half of 3D voxelized scene as input to the generator network (same as the CNN model that we used in Section 3.1). The bottom network is the discriminator consisting of two local and global sub-networks, inspired from [26, 37]. The local network takes in only the generated part while the global network feeds in the generated part along with the input scene. The discriminator network concatenates the activations of these two sub-networks with a fully-connected layer to estimate a realness probability.

$G(\mathbf{f})$. In contrast to original GANs [18], we do not use noise ($\mathbf{z}$) during generation, similar to [37, 26] – see Section 3.2.3 for a discussion on this.

For training the generator, in addition to the GAN loss, we use the loss that we used for the CNN model in Equation (3.7). This strategy has already been employed in face completion [37] and image completion [26], both using GANs, where squared-error loss was used. They have shown that combination of the GAN loss and the reconstruction loss (CNN-SE loss in our case) leads to much better results. In our case, since we have a voxel-wise classification problem, instead of squared-error loss, we used voxel-wise cross-entropy loss.

As a precaution against mode collapse in GAN training, we use dropout in all layers, except for the first and the last, as suggested in the literature [27].

3.2.2 The Discriminator (D)

Similar to the generator, we use a CNN model to build our discriminator network, with which we squeeze the input to a more compact feature space first. The aim of discriminator network is to recognize if a complete scene is real or generated. As shown in Figure 3.3, D takes two different inputs; the right-hand side of the scene ($G(\mathbf{f})$ or $\mathbf{x} \sim p_{data}(\mathbf{x})$) for a *local* sub-network, and the left-part (**f**) concatenated with the right-hand side ($G(\mathbf{f})$ or $\mathbf{x} \sim p_{data}(\mathbf{x})$) for a *global* sub-network, inspired from [26] and [37]. The outputs of these local-global networks are concatenated together and mapped to a binary label (real or fake).

Table 3.2 describes the details of the discriminator architecture. The local discriminator sub-network takes second part as input as 42×44 grid size with 42 channels. The global discriminator follows the same architecture as local, but takes the whole scene (**f** concatenated with $G(\mathbf{f})$ or $\mathbf{x} \sim p_{data}(\mathbf{x})$). The motivation behind this global network is not only to distinguish real and fake scenes but also to force the discriminator network to learn (i) the features from the whole scene and (ii) the relation between the first and second halves of the scenes. We will show that this combination yields better results than either of them.

3.2.3 Stable Training and Loss Functions

As proposed in [18], because of the gradient vanishing problem in minimizing $\log(1-D)$ for the generator, we use as Equation (3.7) to maximize $\log(D)$ instead, which also allows to have stronger gradient. The GAN loss for the discriminator tries to maximize $\log(D(\mathbf{f}, \mathbf{x}))$ for input and its real other half, and to minimize $\log(D(\mathbf{f}, G(\mathbf{f})))$ for input and its generated other half. In fact, the GAN loss of the generator tries to maximize the probability of the generated data, while the discriminator wants to maximize the real data and minimize the generated data probability.

We have also tried adding noise (**z**) to the input of the generator, which however did not lead to any stable training. As also noted by others [60], creating 3D data from noise is rather complicated, since there is too much variability in the 3D space, requiring more data, and different architectures or strategies. Therefore, in our results, we

Table 3.2: The discriminator architecture of our GAN model. The last layer has sigmoid activation to squash the output between between 0 and 1. 'LR' shows Leaky ReLU activation function. Batcn normalization used in all layers except concat and FC layers.

(a) Details of the local sub-network.

Layer	Activation	Kernel	Stride	Dilation	Outputs
Conv.	LR	3×3	2	1	64
Conv.	LR	3×3	2	1	64
Conv.	LR	3×3	1	1	64
Conv.	LR	3×3	2	1	64

(b) Details of the global sub-network.

Layer	Activation	Kernel	Stride	Dilation	Outputs
Conv.	LR	3×3	2	1	64
Conv.	LR	3×3	2	1	64
Conv.	LR	3×3	1	1	64
Conv.	LR	3×3	2	1	64

(c) Details of the fully connected layer.

Layer	Activation	Kernel	Stride	Dilation	Outputs
Concat.	-	-	-	-	9768
FC	-	-	-	-	1

only use the visible part of scene to generate the other part, which made the training process more stable.

Furthermore, training the GAN models and keeping the generator and discriminator in a balance is tricky. Because the discriminator learns to detect fake scenes from real scenes long before the generator can start to generate meaningful data. Then, it is impossible for the generator to learn training data distribution form ahead discriminator. To overcome this problem we train the generator for 50,000 iterations, before starting to train the discriminator in parallel, which makes the training process more stable in our case. However, training more than 700,000 iterations result in somehow partial mode collapse, which causes the network to generate the same shape of ‘walls’ and ‘floors’ in most scenes. Therefore, we stop training somewhere between 700K and 1000K iterations.

3.3 3D Scene Extrapolation with Hybrid Model (Hybrid-SE)

This model, in addition to taking the first half in 3D ($\mathbf{f}^{3D}$) as input, processes the 2D top view projection in parallel. Figure 3.4 shows the overall view of our hybrid model. $\mathbf{f}^{3D}$ is fed to the CNN-SE model as explained in section 3.1, in parallel, $\mathbf{f}^{top}$, the 2D top view projection of $\mathbf{f}$ goes into an autoregressive generative model, namely PixelCNN [47], to generate the top view of the second part ($\mathbf{s}^{top}$). This top view should provide whereabouts and identities of the objects in the space to be extrapolated. Then a *2D to 3D* network takes this generated 2D top view as input and predicts the third dimension to make it 3D. The output from CNN-SE ($\mathbf{s}^{3D}$) and *2D to 3D* network ($\mathbf{s}^{top}$) are aggregated together and fed into a smoothing network consisting of 4 ResNeXt blocks [62]. In this model, we use the loss in Eq. 3.6 for CNN-SE network; for PixelCNN network, Eq. 3.1, 3.4; and for the focal loss for *2D to 3D* network, Eq. 3.1, 3.2 and 3.4; and for the smoother network, Eq. 3.1 and 3.2. All the weights in our hybrid model are trained from scratch, and the training process is end-to-end.

Our hybrid model predicts the top-view of the unseen part. However, this 2D extrapolation is itself as challenging as 3D extrapolation. In order to demonstrate the full potential of having a good top-view estimation of the unseen part, we also solved the

3D extrapolation by feeding the known 2D top-view of the unseen part.

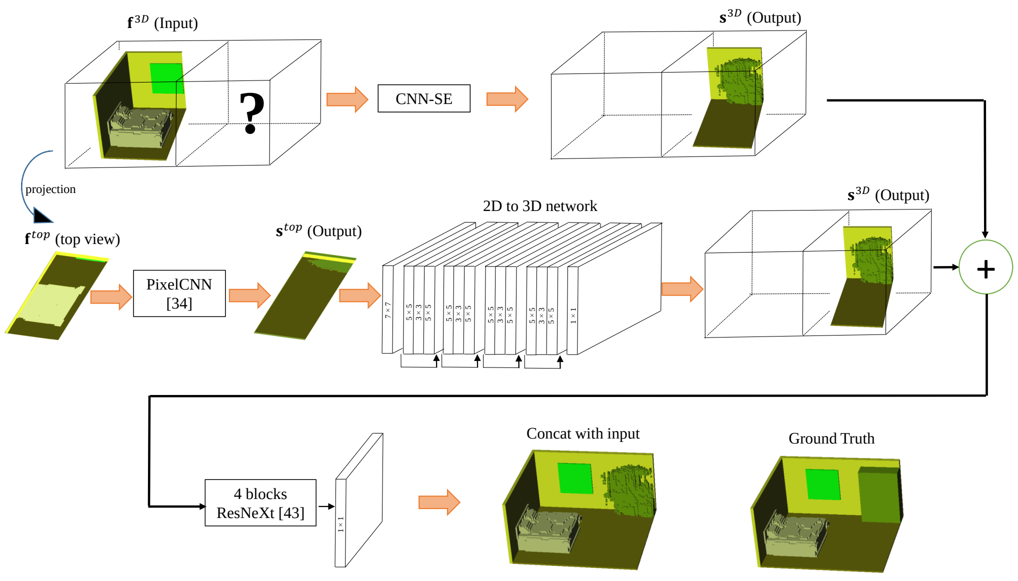

$\mathbf{f}^{3D}$ (Input)
?
CNN-SE
$\mathbf{s}^{3D}$ (Output)
projection
$\mathbf{f}^{top}$ (top view)
PixelCNN [34]
$\mathbf{s}^{top}$ (Output)
2D to 3D network
7×7
5×5
3×3
5×5
1×1
$\mathbf{s}^{3D}$ (Output)
+
4 blocks ResNeXt [43]
1×1
Concat with input
Ground Truth

Figure 3.4: The hybrid model used in this study. We feed the half of 3D voxelized scene as input to the CNN-SE network - see Section 3.1. In parallel, the top view projection of the first-half (i.e., $\mathbf{f}^{top}$) is fed into PixelCNN [47] to generate the other side top view ($\mathbf{s}^{top}$). The 2D to 3D network consists of multiple residual block map its 2D input to 3D. Then this output is aggregated with CNN-SE network output and goes into a smoother network consisting of 4 ResNeXt [62] blocks.

CHAPTER 4

EXPERIMENTS

In this section, we explain our experiments and results on 3D scene extrapolation problem. In order to have a comparison, we also implemented U-net [49] and SSC-Net [53] architectures for 3D scene extrapolation task. We list the quantitative and qualitative results in the section 4.3.

4.1 Dataset

We used SUNCG [53] synthetic 3D scene dataset, for training and inference. This dataset contains 45,622 different scenes, about 400k rooms and 2644 unique objects. We constructed scenes by their available information with provided as JSON file for each scene. We parsed each scene JSON file and build a separate scene for each room. In the room extraction process we ignored scene types such as outdoor scenes, swimming pool and etc. We used the binvox [44] voxelizer, which use space carving approach for both surface and interior voxels. To make the voxelizing process faster, we first voxelized each object individually then put them in their place in each room. During the object voxelization, we set the voxel size to 6cm, then voxelized the objects with respect to their dimensions. To find the resolution for each object, the longest dimension in the object was divided by 6 to give the grid size of voxelization for that object.

In our experiments, we removed categories that did not have sufficient number of instances, (or smaller than frequently occurring objects in scene like beds, sofas, cabinets, windows and etc), which reduced the number of object categories from 84 to

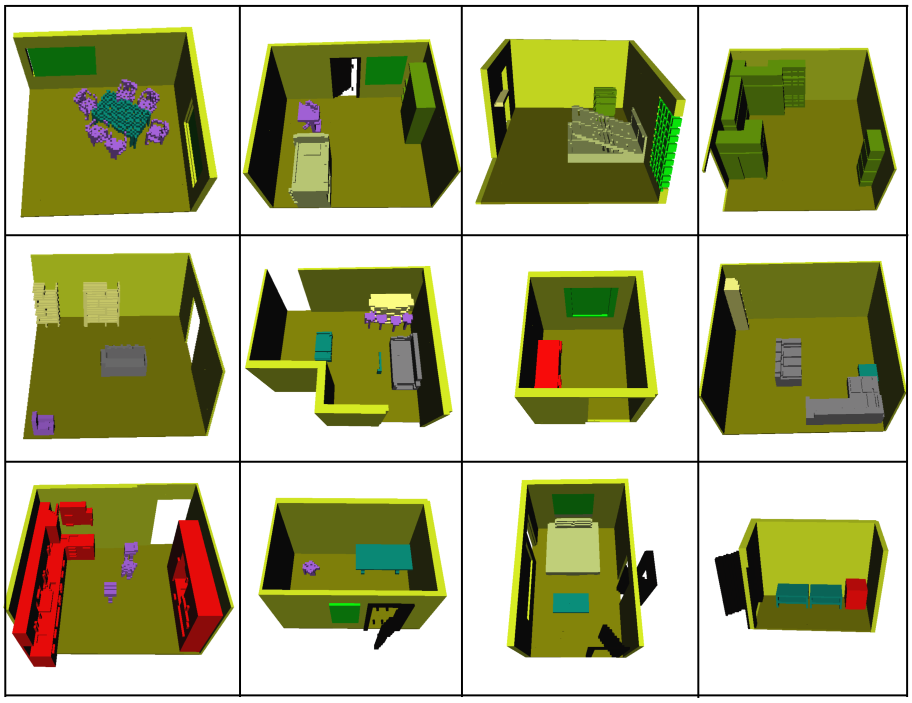


Figure 4.1: Sample scenes from the synthetic SUNCG dataset [53], a challenging dataset containing diverse kinds of chairs, beds, sofas, shelves, tables, cabinets and coffee tables in different shapes and sizes.

13 (plus one category, for emptiness). The removed objects include TVs, plants, toys, kitchen appliance, floor lamps, people, cats and etc. We set a fixed resolution in our models to $84 \times 44 \times 84$. We observed that this resolution was sufficient to represent the objects and the content of the scenes. We also removed scenes which has less than 12000 voxels in size, and leave about 211K scene, we used 200K as the training data and the rest as the test data. The data processing and building the 3D scenes roughly took a week. Figure 4.1 shows some sample scenes from this challenging dataset.

4.2 Training Details

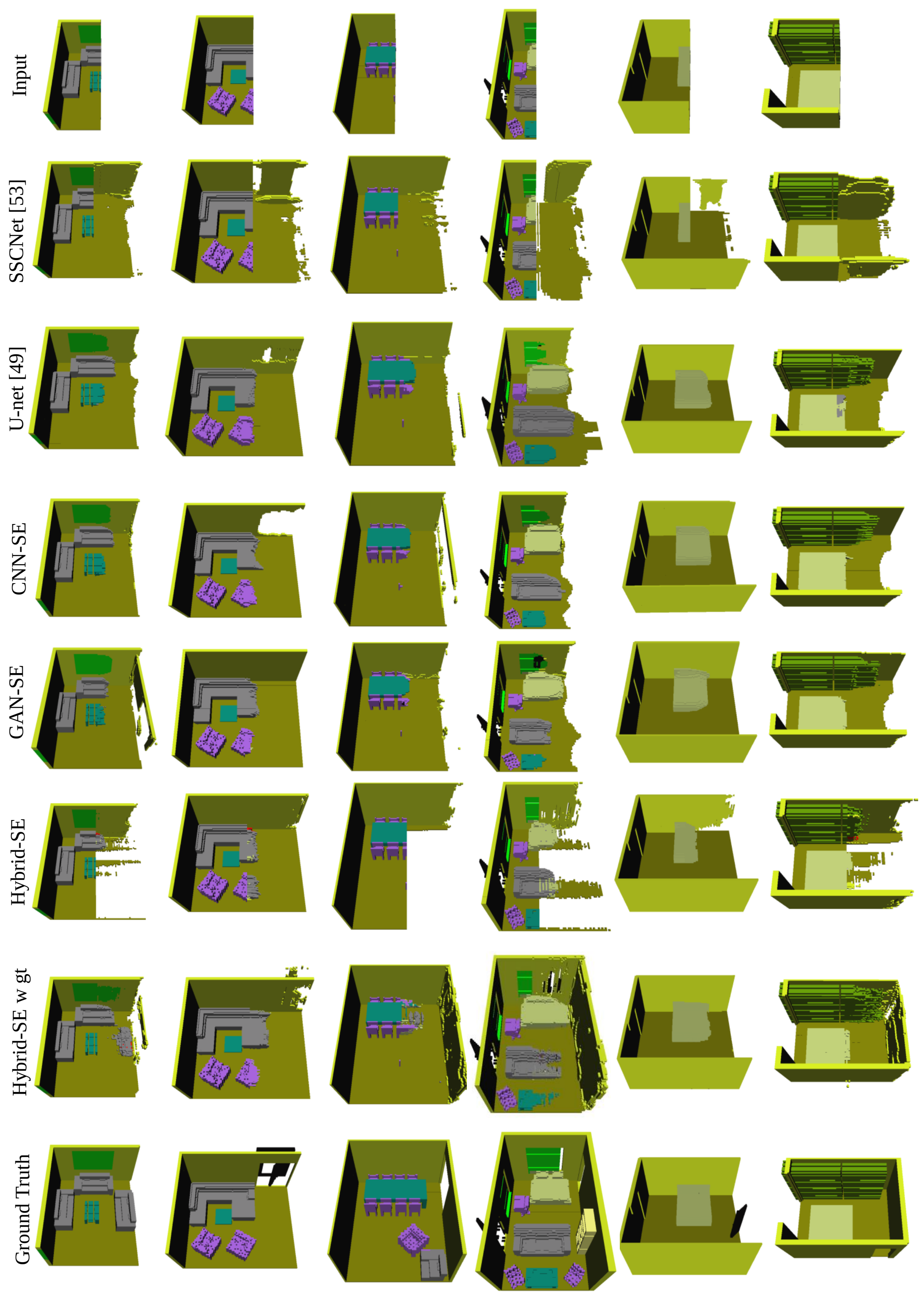
We used Tensorflow [1] to implement our models. In our models we train the weights from scratch. The training process in all our models is end-to-end. Table 4.1 summarizes the training details for each model individually. For all training processes, we used a single NVIDIA GeForce Titan X GPU.

Table 4.1: The training details for our models. The weights are initialize randomly with mean=0 and std=0.01. The training process for each model took about 8 days. 'w gt' indicates known 2D top view. In our models we train the weights from scratch. The training process in all our models is end-to-end.

model	lr	bs	iter.	optimizer
CNN-SE	5×10^{-5}	32	500,000	Adam
U-net [49]	5×10^{-5}	32	500,000	Adam
GAN-SE	5×10^{-5}	32	500,000	Adam
H-SE	1×10^{-5}	16	500,000	Adam
H-SE w gt	1×10^{-5}	16	500,000	Adam

4.3 Results

Figure 4.2 shows results on the SUNCG dataset [53] from our models. We use two metrics in order to evaluate the results. First, the accuracy metric, defined as the number of correctly generated voxels in the output divided by the total number of ground truth voxels. In the first metric, we also consider empty voxels if they are estimated at the right place. The second metric is completeness, which is the number of correctly generated non-empty voxels in the output divided by the total number of non-empty voxels. Table 4.2 shows the quantitative results of our models, as long as per-category F1 scores and Figure 4.4 shows the cost and accuracy plots for trained models. Also, SSCNet [53] and U-net [49] in Figure 4.2 and Table 4.2 are the baseline models we implement them for scene extrapolation task to have a comparison.



🟫floor 🟨wall 🟩window ⬛door 🟪chair 🟩bed ⬜sofa 🟩table 🟨shelve 🟩cabinet 🟫furniture

Figure 4.2: Results on the SUNCG dataset [53] from our models.'w gt' indicates known 2D top view.

Table 4.2: The accuracy, completeness, precision and recall measures of our models. The score under each object category shows F1 score. SSCNet [53] and U-net [49] are the baseline models we implement them for scene extrapolation task to have comparison. CNN-SE shows our implemented model consists of CNN layers explained in section 3.1. GAN-SE shows our generative model which explained in section 3.2. GAN-SE-G uses alone global discriminator sub-network. GAN-SE-L uses the alone local discriminator sub-network. H-SE stands for Hybrid-SE model and 'w gt' means with ground truth of 2D projection top view explained in section 3.3.

model	accuracy	completeness	precision	recall	empty voxels	floor	wall	window	door	chair	bed	sofa	table	cabinet	shelving	wardrobe	house stuff	average
SSCNet [53]	89.1	27.3	16.4	22.9	-	-	-	-	-	-	-	-	-	-	-	-	-	-
U-net [49]	94.2	46.7	32.2	75.6	97.3	94.5	88.6	77.3	28.2	44.4	82.2	74.5	47.5	68.5	**49.3**	**85.9**	1.0	65.8
Our CNN-SE	93.7	40.8	45.7	77.0	**98.3**	**98.1**	82.5	57.8	26.8	21.6	80.6	69.7	12.5	61.6	38.0	64.9	5.0	55.2
Our GAN-SE-G	94.4	48.3	63.1	67.5	97.3	94.8	88.4	75.2	32.5	50.1	**84.8**	77.2	35.9	72.6	41.2	73.8	14.3	64.5
Our GAN-SE-L	94.4	48.5	58.4	**78.9**	97.5	93.2	83.1	78.0	48.7	**60.9**	75.5	78.4	**63.8**	53.8	37.6	68.1	23.2	66.3
Our GAN-SE	94.6	50.8	**64.1**	76.3	97.5	93.4	86.4	79.5	**55.4**	52.1	82.3	**87.7**	54.7	71.0	38.1	65.4	**24.2**	68.3
Our H-SE	94.4	49.2	58.2	46.2	97.2	90.4	83.3	44.0	13.0	18.0	61.5	51.3	8.0	45.7	21.0	58.3	8.0	46.1
Our H-SE w gt	**95.1**	**66.2**	62.2	62.6	97.4	95.5	**87.4**	**82.5**	44.3	49.4	81.1	78.1	58.1	**84.6**	43.7	71.0	21.0	**68.6**

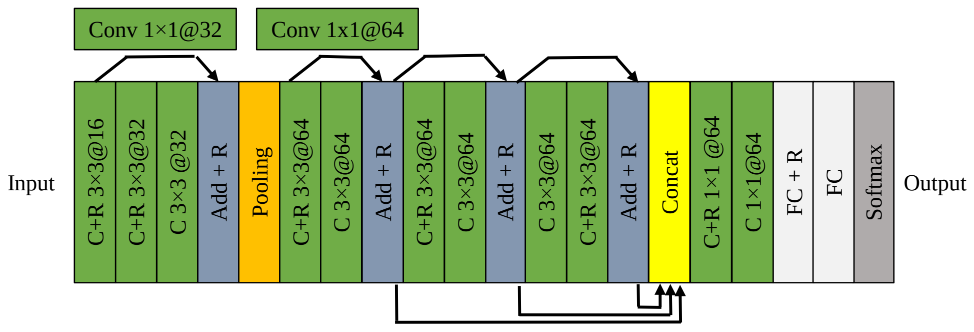


Figure 4.3: The deep convolution neural network architecture consist of multiple 3D convolution and one pooling layer. Abbreviation of 'C' stands for conv and 'R' stands for ReLU. The numbers after @ sign in each layer indicates the output filter number. Upper arrows show the merge function between start point and end point of arrow. They are shortcut connections to improve propagation gradient in network.

4.4 Failed Cases/Tested Models

In this section, we mention all the architectures and approaches we tested and got no desirable results on scene extrapolation task.

For obtaining the desirable method and results, we tested different models and approaches. Firstly, inspired by [53], we design our network architecture consist of 3D convolution, max pooling and fully connected layer. Figure 4.3 shows the deep 3D convolution neural network architecture of our model. The network takes input and applies multiple 3D conv layers with one pooling layer. Then the concatenation operation put together the output of the last 3 merge layers, and pass it to the conv and fully connected layers. The error of network is calculated by Softmax Cross-Entropy loss function between network score and target values.

Table 4.3 shows the summary of what we tried. The 'Classifier' column indicates loss function and classifier type. 'CE' means Cross-Entropy. With 'Weighted CE' in order to solve the problem of unbalance data between empties and non-empties, we assign a constant weight to non-empty voxels loss. '3D' type for input data means the training data was in 3D format, and '2D' means training data was 2D images with channels.

Before solving the problem of completing half of the scene, we simplified the problem

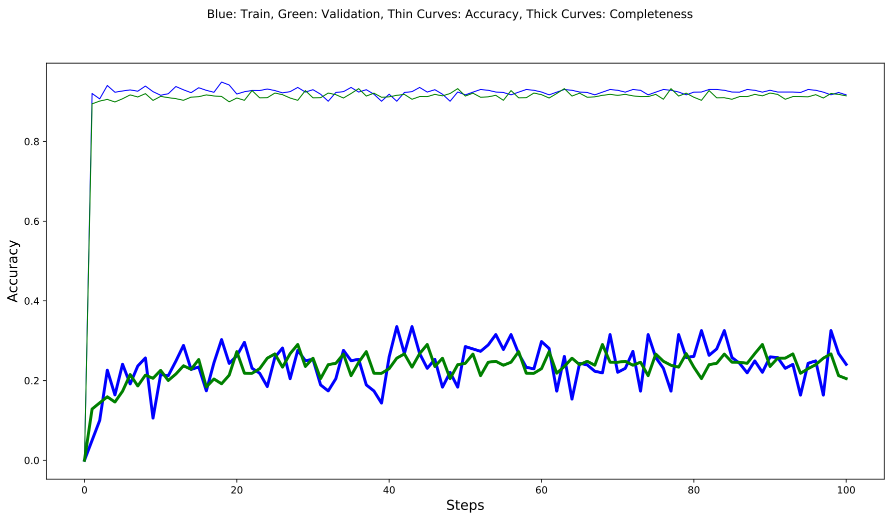


Figure 4.4: The accuracy plot of SSCNet [53] for scene extrapolation problem. The blue and green colored curves are related to train and validation measures respectively. The thin curves show accuracy and thick curves indicate completeness measure.

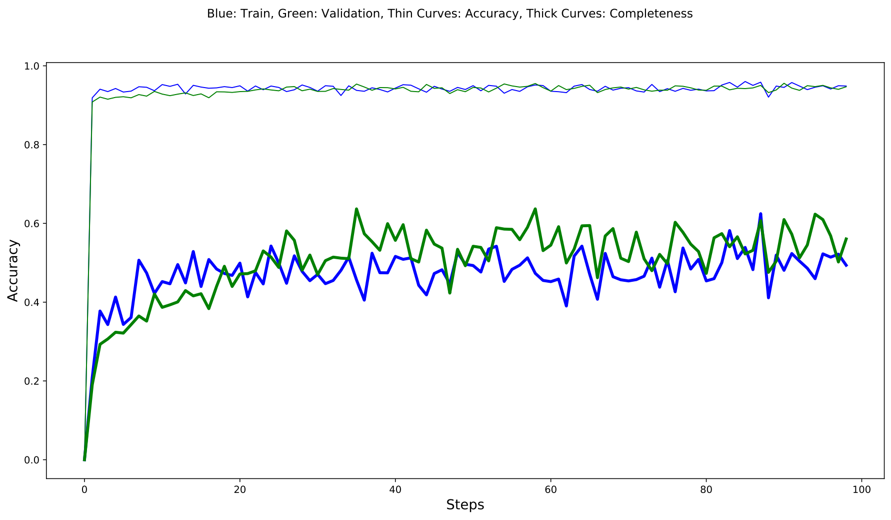


Figure 4.4 (Continued): The accuracy plot of our CNN-SE model. The blue colored curve shows the accuracy while the red one shows completeness measure.

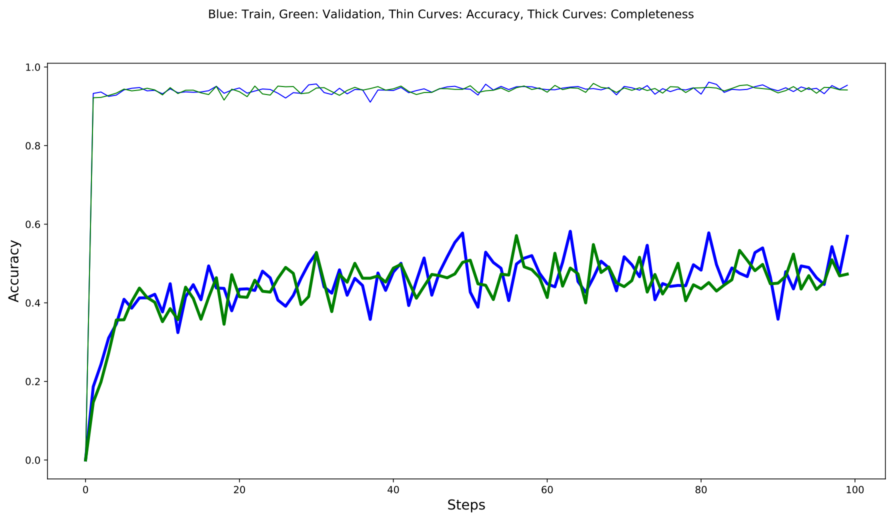


Figure 4.4 (Continued): The accuracy plot of U-net [49] architecture for scene extrapolation problem.

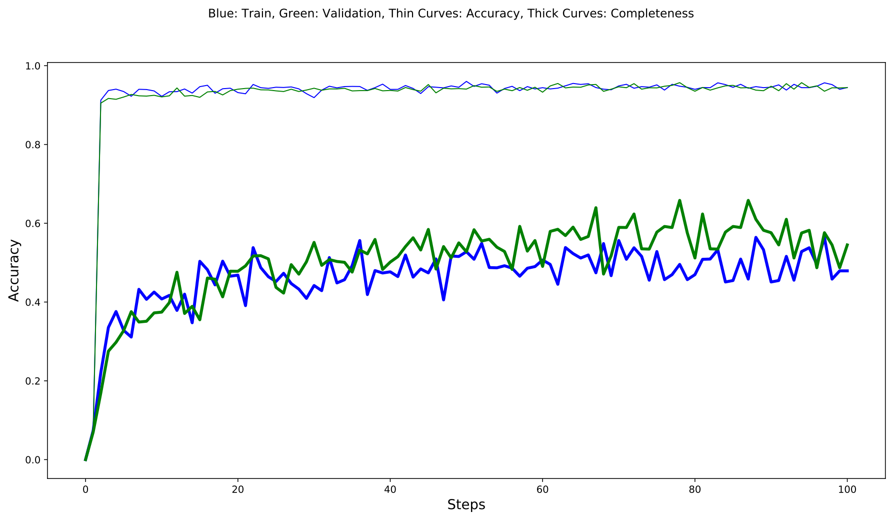


Figure 4.4 (Continued): The accuracy plot of our GAN-SE model.

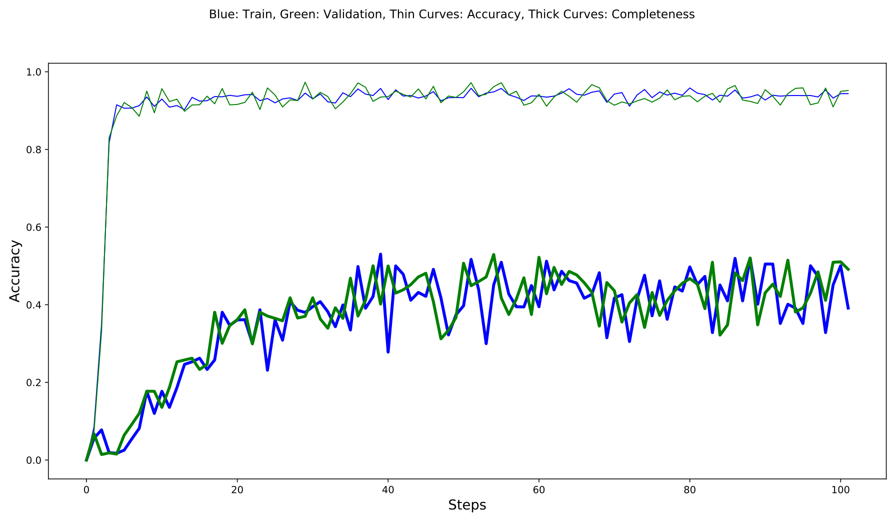


Figure 4.4 (Continued): The accuracy plot of our Hybrid-SE model.

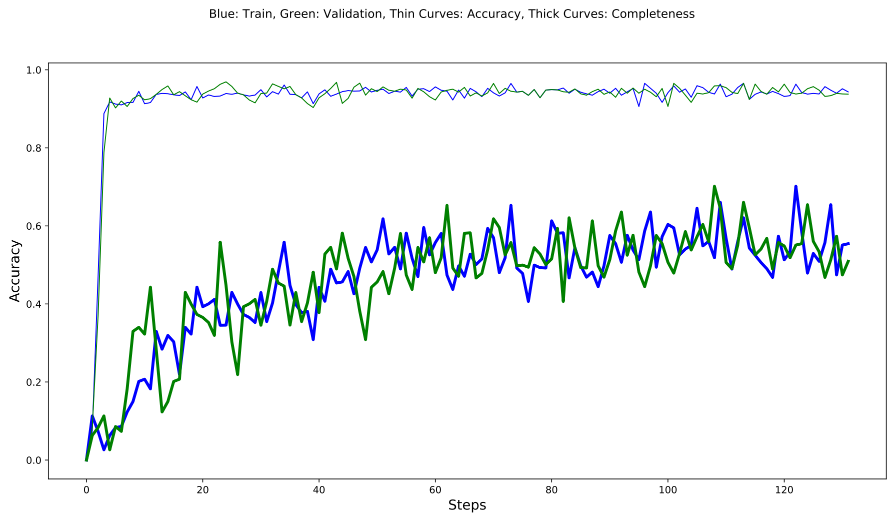


Figure 4.4 (Continued): The accuracy plot of our Hybrid-SE model with ground truth of 2D projection top view.

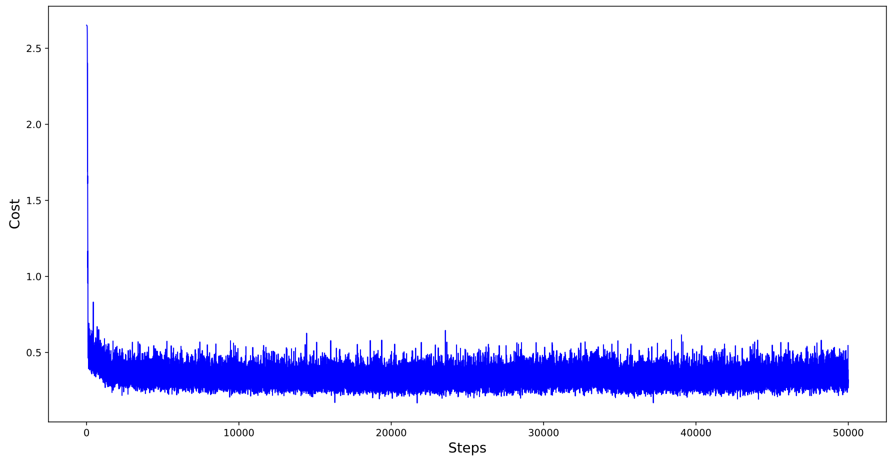


Figure 4.4 (Continued): The cost plot of SSCNet [53] model for scene extrapolation problem.

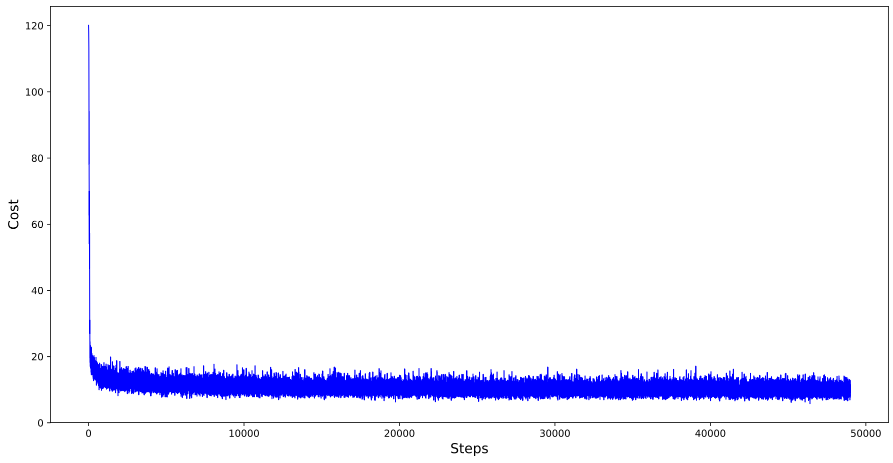


Figure 4.4 (Continued): The cost plot of our CNN-SE model.

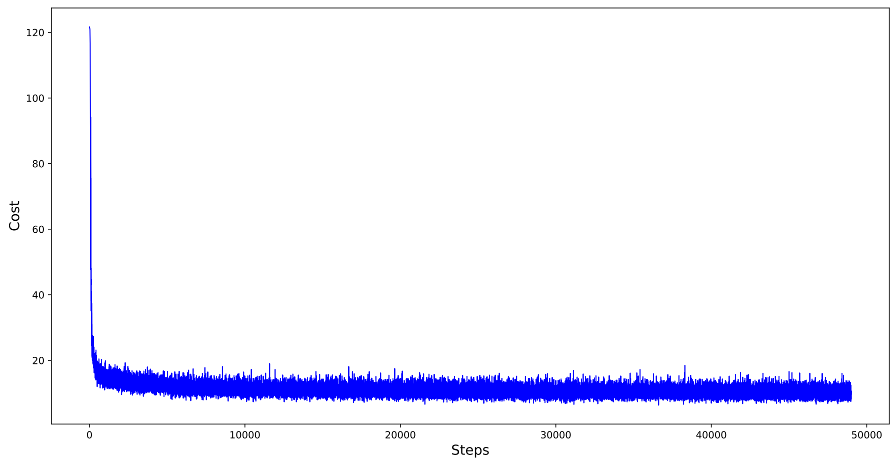


Figure 4.4 (Continued): The cost plot of U-net [49] architecture for scene extrapolation problem.

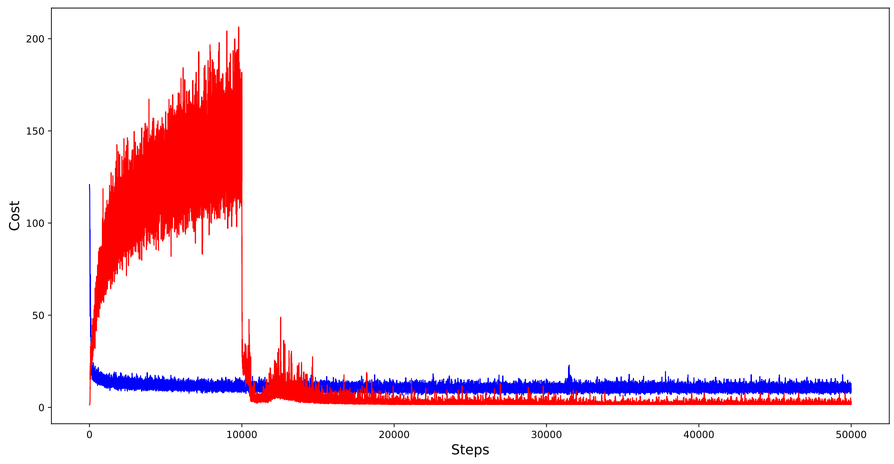


Figure 4.4 (Continued): The cost plot of our GAN-SE model. Blue curve shows the generator network cost while the red one shows discriminator network loss. The training of the discriminator network start at step 100k.

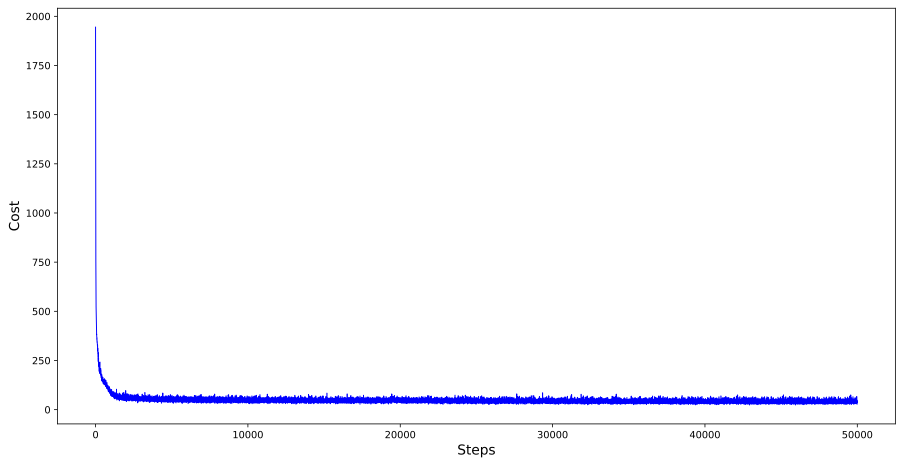


Figure 4.4 (Continued): The cost plot of our Hybrid-SE model.

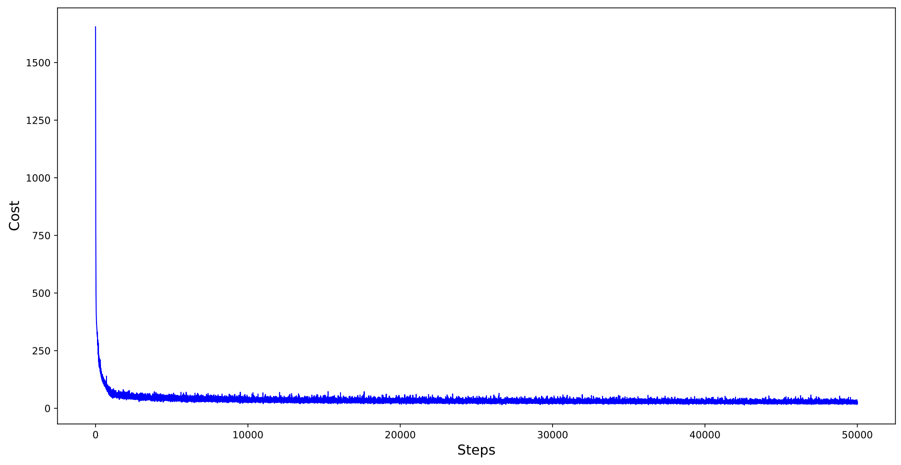


Figure 4.4 (Continued): The cost plot of our Hybrid-SE model with ground truth of 2D projection top view.

to complete only one channel from the second side of the scene. ‘Target’ column shows the measure of our completion tries ‘42 Planes’ means half of the scene. We use SGD, Decay learning rate for the first try, and Adam optimizer, fixed learning rate for the rest try.

‘Label Encode’ column shows the type of encoding we use for ground truth labels. ‘Normal’ indicates the one-hot vector encoding. ‘N-3N’ says that we convert the label values to one-hot format for all non-empties N, then do the same for 3N empty voxels. Since the ratio of empties to non-empties is 9 to 1. We apply this method to deal with the unbalancing problem between empty and non-empty voxels. To deal with unbalance data we use another label encoding format called ‘Threshold N’. Since in simplified problem we try to complete only one plane, we choose scenes which have at least N non-empty voxels in ground truth plane.

The ‘Labels’ column shows the number of objects category we consider in our experiments. With 84 categories, we have 36 object labels and the range of 36-84 considered as ‘others’ category. With 36 labels we have no ‘others’ category, and with 14 labels we have exactly 13 object category plus one for empties.

‘Margin’ column shows the margin distance in Hinge loss related to SVM classifier.

Table 4.3: The summary of all we try with convolution neural networks. ‘CE’ means Cross-Entropy loss.

Classifier	**Input Data**	**Target**	**Optimizer**	**lr**	**Label Encode**	**#Labels**	**Margin**
Softmax + CE	3D	42 Planes	SGD	Decay	Normal	84	.
Softmax + CE	3D	42 Planes	Adam	Fixed	Normal	84	.
Softmax + Weighted CE	3D	One Plane	Adam	Fixed	Normal	84	.
Softmax + CE	3D	One Plane	Adam	Fixed	N - 3N	84	.
Softmax + CE	3D	One Plane	Adam	Fixed	Threshold 100	36	.
Softmax + CE	3D	One Plane	Adam	Fixed	Threshold 100	36	.
Softmax + CE	3D	One Plane	Adam	Fixed	Threshold 100	36	.
Softmax + CE	3D	One Plane	Adam	Fixed	Threshold 1	13	.
Softmax + CE	3D	One Plane	Adam	Fixed	Threshold 1	13	.
Softmax + Weighted CE	3D	One Plane	Adam	Fixed	Threshold 10	13	.
SVM + Hinge	3D	One Plane	Adam	Fixed	Normal	84	1
SVM + Hinge	3D	One Plane	Adam	Fixed	Normal	84	2
SVM + Hinge	3D	One Plane	Adam	Fixed	Normal	84	10
Softmax + CE	2D	One Plane	Adam	Fixed	Normal	84	.
Softmax + Weighted CE	2D	One Plane	Adam	Fixed	Normal	84	.

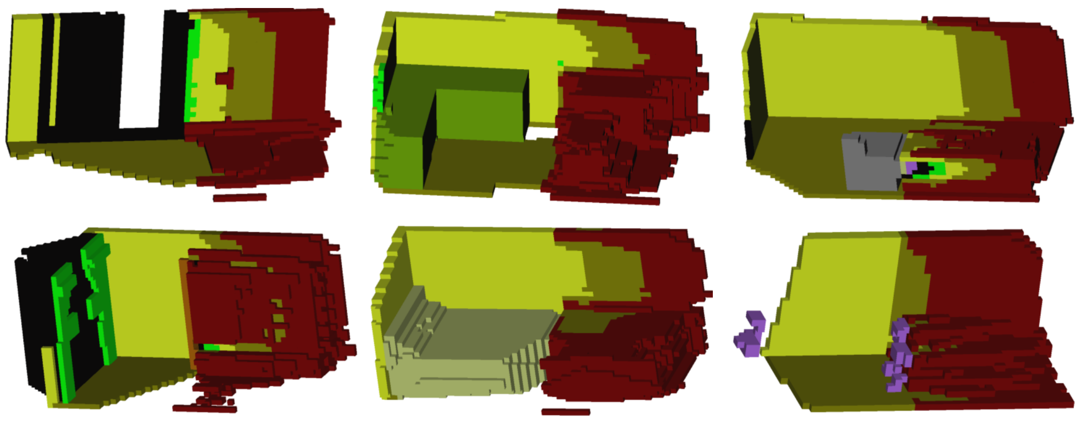


Figure 4.5: The tanh activation function int the last layer of generator network, GAN model. Left part of each scene is input and other side is generated part, the dark red color voxels have value 1 which shows ‘ceiling’ voxels, since its value is very close to 0 ‘empty’, 2 ‘floor’ and 3 ‘wall’ voxels.

Also Figure 4.5 shows the results of GAN-SE model with the tanh activation function for the last layer of generator network.

4.5 Discussion

GAN vs. CNN: As a general interpretation of the results from the CNN and GAN models, we see that the GAN model learns to generate more realistic objects, while the CNN model has a tendency to extend the parted objects without any correct knowledge of their lengths or shapes. In contrast, our results suggest that especially GAN has learned a prior on the length of objects and does not extend an object more than a canonical length (see Figure 4.2).

Local-Global discriminator: We find that the local-global discriminator forces the generator to build sharper and more realistic results, as shown in Figure 4.6. This is mainly due to the fact that, discriminator not only need to learn the local features of generated part, but also global features from whole scene and relation between input part and generated part to distinguish fake and real scenes. The abbreviation GAN-SE-L indicates the GAN model with only local discriminator, and GAN-SE-G shows the GAN model with only global discriminator. Table 4.2 lists the quantitative results from GAN-SE-L and GAN-SE-G models.

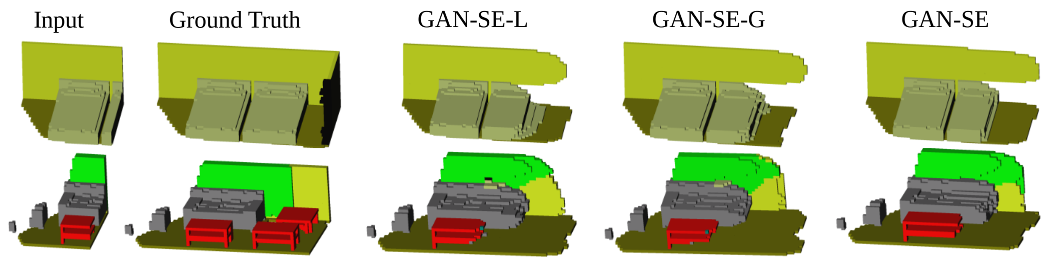


Figure 4.6: The effect of the local and global sub-networks. The GAN model with local-global sub-networks (last column) tries to generate more realistic and logical objects, while the local and global discriminators alone fail at produce correctly elongated objects.

The architecture: We have experimented with many architectures and structures for the generator, include pooling layers, bottleneck architectures with strided and deconvolution layers, yet, at the end, we concluded that a simple architecture of stride-1 convolution layers can generate better results. This suggests that it is better to keep the widths of the layers unreduced and allow the network to process information along the width of the scene at each convolution. This also allows the network to get a better estimation about the sizes of the objects since each convolution has access to the width of the scene. Moreover, worse performance on increased filter sizes suggests that highly-local processing is better and bigger filter sizes lead to averaging of information across voxels.

Challenges of the problem: The dataset we use for training has diverse kinds of objects in different shapes and sizes, which make the problem very challenging. Another challenge is the issue of partial objects in the scenes; i.e., due to the field of view of the cameras, some objects have been captured only partially, making the training process more difficult for the networks to generalize about the shapes and the sizes of the objects.

Due to these challenges, we observe that our models also make mistakes, as shown in Figure 4.7, and might extrapolate a geometrically similar object, rather than the correct one.

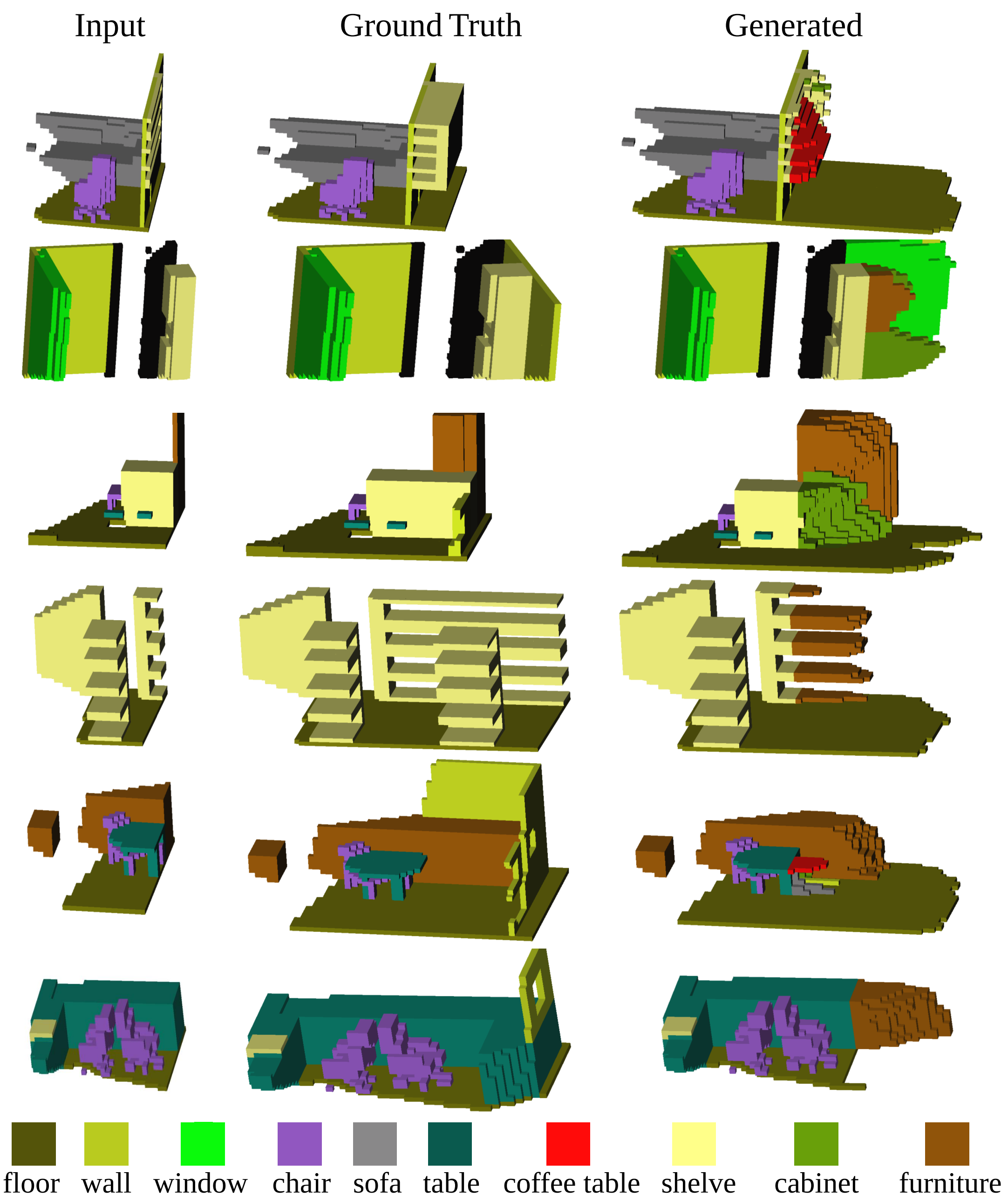


Figure 4.7: Failure cases. The columns from left to right respectively show input, ground truth and generated result by our GAN model . First four rows show the failing of generating ‘Shelve’, since geometrically it look likes furniture, ‘cabinet’, ‘sofa’ and also ‘table’ in first row. This similarity is also among other objects like ‘tables’ and ‘coffee tables’, ‘cabinets’ and ‘furnitures’ are shown in row five and six.

CHAPTER 5

CONCLUSION

In this thesis, we have proposed using deep generative-adversarial and convolutional networks for the scene extrapolation problem, which has not been addressed in the literature before. We showed that the proposed models are able to extrapolate the scene towards the unseen part and extend successfully the ground, the walls and the partially-visible objects.

However, we realized that the networks were unable to hallucinate objects in the extrapolated part of the scene if they did not have any part visible in the input scene. This is likely to be due to the highly challenging nature of the extrapolation problem, due to the huge inter-variability and intra-variability between the objects. To able to extrapolate at a level where new objects can be generated would likely require (i) a much larger dataset than we used and available in the literature, and (ii) extraction and usage of higher-level semantic information from the input scene, as a modulator for the deep extrapolating networks.

Scene extrapolation is a challenging and highly-relevant problem for the computer graphics and deep learning societies, and our study offers a new research problem and direction with which new and better learning and estimation methods can be developed.

Future Work One can formulate the scene extrapolation task as a sequence modeling problem, such that the input sequence (the visible scene) is encoded by a sequence modeling framework, and the rest of the scene is decoded by another. Moreover, a context network can be trained in parallel together with the generation network such that the context network captures a high-level scene information and layout of the scene and modulates the generation process.

REFERENCES

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

[2] Averbuch-Elor, H., Kopf, J., Hazan, T., and Cohen-Or, D. Co-segmentation for space-time co-located collections. *The Visual Computer* (2017), 1–12.

[3] Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence 35*, 8 (2013), 1798–1828.

[4] Bengio, Y., et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning 2*, 1 (2009), 1–127.

[5] Besl, P. J., McKay, N. D., et al. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence 14*, 2 (1992), 239–256.

[6] Bouaziz, S., Tagliasacchi, A., and Pauly, M. Sparse iterative closest point. In *Computer graphics forum* (2013), vol. 32, Wiley Online Library, pp. 113–123.

[7] Chatfield, K., Simonyan, K., Vedaldi, A., and Zisserman, A. Return of the devil in the details: Delving deep into convolutional nets. *arXiv preprint arXiv:1405.3531* (2014).

[8] Dai, A., Qi, C. R., and Niessner, M. Shape completion using 3d-encoder-predictor cnns and shape synthesis. *arXiv preprint arXiv:1612.00101* (2016).

[9] Deng, L., Yu, D., et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing 7*, 3–4 (2014), 197–387.

[10] DeVries, T., and Taylor, G. W. Dataset augmentation in feature space. *arXiv preprint arXiv:1702.05538* (2017).

[11] Duchi, J., Hazan, E., and Singer, Y. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research 12*, Jul (2011), 2121–2159.

[12] Duda, R. O., Hart, P. E., and Stork, D. G. *Pattern classification.* John Wiley & Sons, 2012.

[13] Firman, M., Mac Aodha, O., Julier, S., and Brostow, G. J. Structured prediction of unobserved voxels from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 5431–5440.

[14] Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., and Hanrahan, P. Example-based synthesis of 3d object arrangements. *ACM Transactions on Graphics (TOG) 31*, 6 (2012), 135.

[15] Fisher, M., Savva, M., Li, Y., Hanrahan, P., and Niessner, M. Activity-centric scene synthesis for functional 3d scene modeling. *ACM Transactions on Graphics (TOG) 34*, 6 (2015), 179.

[16] Gal, R., Shamir, A., Hassner, T., Pauly, M., and Cohen-Or, D. Surface reconstruction using local shape priors. In *Symposium on Geometry Processing* (2007), no. EPFL-CONF-149318, pp. 253–262.

[17] Goodfellow, I., Bengio, Y., and Courville, A. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

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

[19] Harary, G., Tal, A., and Grinspun, E. Context-based coherent surface completion. *ACM Transactions on Graphics (TOG) 33*, 1 (2014), 5.

[20] Harary, G., Tal, A., and Grinspun, E. Feature-preserving surface completion using four points. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 45–54.

[21] Hays, J., and Efros, A. A. Scene completion using millions of photographs. In *ACM Transactions on Graphics (TOG)* (2007), vol. 26, ACM, p. 4.

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

[23] HINTON, G. E., OSINDERO, S., AND TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation 18*, 7 (2006), 1527–1554.

[24] HU, S.-M., CHEN, T., XU, K., CHENG, M.-M., AND MARTIN, R. R. Internet visual media processing: a survey with graphics and vision applications. *The Visual Computer 29*, 5 (2013), 393–405.

[25] HUANG, J.-B., KANG, S. B., AHUJA, N., AND KOPF, J. Temporally coherent completion of dynamic video. *ACM Transactions on Graphics (TOG) 35*, 6 (2016), 196.

[26] IIZUKA, S., SIMO-SERRA, E., AND ISHIKAWA, H. Globally and locally consistent image completion. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 107.

[27] ISOLA, P., ZHU, J.-Y., ZHOU, T., AND EFROS, A. A. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004* (2016).

[28] KAZHDAN, M., AND HOPPE, H. Screened poisson surface reconstruction. *ACM Transactions on Graphics (TOG) 32*, 3 (2013), 29.

[29] KELLY, B., MATTHEWS, T. P., AND ANASTASIO, M. A. Deep learning-guided image reconstruction from incomplete data. *arXiv preprint arXiv:1709.00584* (2017).

[30] KERMANI, Z. S., LIAO, Z., TAN, P., AND ZHANG, H. Learning 3d scene synthesis from annotated rgb-d images. In *Computer Graphics Forum* (2016), vol. 35, Wiley Online Library, pp. 197–206.

[31] KINGMA, D., AND BA, J. Adam: A method for stochastic optimization. *International Conference on Learning Representations* (2015).

[32] KRAEVOY, V., AND SHEFFER, A. Template-based mesh completion. In *Symposium on Geometry Processing* (2005), vol. 385, pp. 13–22.

[33] KWATRA, V., ESSA, I., BOBICK, A., AND KWATRA, N. Texture optimization for example-based synthesis. *ACM Transactions on Graphics (ToG) 24*, 3 (2005), 795–802.

[34] LECUN, Y., BENGIO, Y., AND HINTON, G. Deep learning. *nature 521*, 7553 (2015), 436.

[35] LECUN, Y., BOTTOU, L., BENGIO, Y., AND HAFFNER, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (1998), 2278–2324.

[36] LI, D., SHAO, T., WU, H., AND ZHOU, K. Shape completion from a single rgbd image. *IEEE transactions on visualization and computer graphics 23*, 7 (2017), 1809–1822.

[37] LI, Y., LIU, S., YANG, J., AND YANG, M.-H. Generative face completion. *arXiv preprint arXiv:1704.05838* (2017).

[38] LIEPA, P. Filling holes in meshes. In *Eurographics/ACM SIGGRAPH Symposium on Geometry Processing* (2003), Eurographics Association, pp. 200–205.

[39] LIN, T.-Y., GOYAL, P., GIRSHICK, R., HE, K., AND DOLLÁR, P. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002* (2017).

[40] LIU, H., ZHANG, L., AND HUANG, H. Web-image driven best views of 3d shapes. *The Visual Computer 28*, 3 (2012), 279–287.

[41] MAAS, A. L., HANNUN, A. Y., AND NG, A. Y. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML* (2013), vol. 30.

[42] MAVRIDIS, P., SIPIRAN, I., ANDREADIS, A., AND PAPAIOANNOU, G. Object completion using k-sparse optimization. In *Computer Graphics Forum* (2015), vol. 34, Wiley Online Library, pp. 13–21.

[43] MELLADO, N., AIGER, D., AND MITRA, N. J. Super 4pcs fast global point-cloud registration via smart indexing. In *Computer Graphics Forum* (2014), vol. 33, Wiley Online Library, pp. 205–215.

[44] MIN, P. Binvox. `https://www.patrickmin.com/binvox/`, accessed June 21, 2018.

[45] MNIH, A., AND GREGOR, K. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030* (2014).

[46] NAIR, V., AND HINTON, G. E. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)* (2010), pp. 807–814.

[47] OORD, A. V. D., KALCHBRENNER, N., AND KAVUKCUOGLU, K. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759* (2016).

[48] RADFORD, A., METZ, L., AND CHINTALA, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434* (2015).

[49] RONNEBERGER, O., FISCHER, P., AND BROX, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention* (2015), Springer, pp. 234–241.

[50] SAHILLIOĞLU, Y., AND YEMEZ, Y. Coarse-to-fine surface reconstruction from silhouettes and range data using mesh deformation. *Computer Vision and Image Understanding 114*, 3 (2010), 334–348.

[51] SHARF, A., ALEXA, M., AND COHEN-OR, D. Context-based surface completion. *ACM Transactions on Graphics (TOG) 23*, 3 (2004), 878–887.

[52] SONG, H. A., AND LEE, S.-Y. Hierarchical representation using nmf. In *International Conference on Neural Information Processing* (2013), Springer, pp. 466–473.

[53] SONG, S., YU, F., ZENG, A., CHANG, A. X., SAVVA, M., AND FUNKHOUSER, T. Semantic scene completion from a single depth image. *Conference on Computer Vision and Pattern Recognition* (2017).

[54] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research 15*, 1 (2014), 1929–1958.

[55] SUWAJANAKORN, S., SEITZ, S. M., AND KEMELMACHER-SHLIZERMAN, I. Synthesizing obama: learning lip sync from audio. *ACM Transactions on Graphics (TOG) 36*, 4 (2017), 95.

[56] TIELEMAN, T., AND HINTON, G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning 4*, 2 (2012), 26–31.

[57] WAN, J., WANG, D., HOI, S. C. H., WU, P., ZHU, J., ZHANG, Y., AND LI, J. Deep learning for content-based image retrieval: A comprehensive study. In *Proceedings of the 22nd ACM international conference on Multimedia* (2014), ACM, pp. 157–166.

[58] WEXLER, Y., SHECHTMAN, E., AND IRANI, M. Space-time completion of video. *IEEE Transactions on pattern analysis and machine intelligence 29*, 3 (2007).

[59] WU, J., ZHANG, C., XUE, T., FREEMAN, B., AND TENENBAUM, J. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (2016), pp. 82–90.

[60] WU, J., ZHANG, C., XUE, T., FREEMAN, B., AND TENENBAUM, J. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *Advances in Neural Information Processing Systems* (2016), pp. 82–90.

[61] XIA, C., AND ZHANG, H. A fast and automatic hole-filling method based on feature line recovery. *Computer-Aided Design and Applications* (2017), 1–9.

[62] XIE, S., GIRSHICK, R., DOLLÁR, P., TU, Z., AND HE, K. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on* (2017), IEEE, pp. 5987–5995.

[63] YANG, B., WEN, H., WANG, S., CLARK, R., MARKHAM, A., AND TRIGONI, N. 3d object reconstruction from a single depth view with adversarial learning. *arXiv preprint arXiv:1708.07969* (2017).

[64] YANG, J., LI, H., CAMPBELL, D., AND JIA, Y. Go-icp: a globally optimal solution to 3d icp point-set registration. *IEEE transactions on pattern analysis and machine intelligence 38*, 11 (2016), 2241–2254.

[65] YANG, L., YAN, Q., AND XIAO, C. Shape-controllable geometry completion for point cloud models. *The Visual Computer 33*, 3 (2017), 385–398.

[66] YEH, R. A., CHEN, C., LIM, T. Y., SCHWING, A. G., HASEGAWA-JOHNSON, M., AND DO, M. N. Semantic image inpainting with deep generative models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2017), pp. 5485–5493.

[67] YU, F., AND KOLTUN, V. Multi-scale context aggregation by dilated convolutions. *International Conference on Learning Representations* (2016).

[68] ZHANG, H., XU, M., ZHUO, L., AND HAVYARIMANA, V. A novel optimization framework for salient object detection. *The Visual Computer 32*, 1 (2016), 31–41.

[69] ZHANG, H., XU, T., LI, H., ZHANG, S., HUANG, X., WANG, X., AND METAXAS, D. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242* (2016).

[70] ZHAO, W., GAO, S., AND LIN, H. A robust hole-filling algorithm for triangular mesh. *The Visual Computer 23*, 12 (2007), 987–997.

[71] ZHENG, B., ZHAO, Y., YU, J. C., IKEUCHI, K., AND ZHU, S.-C. Beyond point clouds: Scene understanding by reasoning geometry and physics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 3127–3134.