2019-06-01

Deep Synthetic Noise Generation for RGB-D Data Augmentation

Patrick Douglas Hammond
Brigham Young University

Deep Synthetic Noise Generation for RGB-D Data Augmentation

Patrick Douglas Hammond

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Bryan Stuart Morse, Chair
David Wingate
Mark Joel Clement

Department of Computer Science

Brigham Young University

ABSTRACT

Deep Synthetic Noise Generation for RGB-D Data Augmentation

Patrick Douglas Hammond
Department of Computer Science, BYU
Master of Science

Considerable effort has been devoted to finding reliable methods of correcting noisy RGB-D images captured with unreliable depth-sensing technologies. Supervised neural networks have been shown to be capable of RGB-D image correction, but require copious amounts of carefully-corrected ground-truth data to train effectively. Data collection is laborious and time-intensive, especially for large datasets, and generation of ground-truth training data tends to be subject to human error. It might be possible to train an effective method on a relatively smaller dataset using synthetically damaged depth-data as input to the network, but this requires some understanding of the latent noise distribution of the respective camera. It is possible to augment datasets to a certain degree using naive noise generation, such as random dropout or Gaussian noise, but these tend to generalize poorly to real data. A superior method would imitate real camera noise to damage input depth images realistically so that the network is able to learn to correct the appropriate depth-noise distribution.

We propose a novel noise-generating CNN capable of producing realistic noise customized to a variety of different depth-noise distributions. In order to demonstrate the effects of synthetic augmentation, we also contribute a large novel RGB-D dataset captured with the Intel RealSense D415 and D435 depth cameras. This dataset pairs many examples of noisy depth images with automatically completed RGB-D images, which we use as proxy for ground-truth data. We further provide an automated depth-denoising pipeline which may be used to produce proxy ground-truth data for novel datasets. We train a modified sparse-to-dense depth-completion network on splits of varying size from our dataset to determine reasonable baselines for improvement. We determine through these tests that adding more noisy depth frames to each RGB-D image in the training set has a nearly identical impact on depth-completion training as gathering more ground-truth data. We leverage these findings to produce additional synthetic noisy depth images for each RGB-D image in our baseline training sets using our noise-generating CNN. Through use of our augmentation method, it is possible to achieve greater than 50% error reduction on supervised depth-completion training, even for small datasets.

Keywords: RGB-D images, depth completion, synthetic augmentation, deep-generative neural networks, variational autoencoders, conditional GANs

ACKNOWLEDGEMENTS

Thanks go to my mother and father for being willing to listen to me rant about mode-collapse and overfit without completely understanding what I was talking about. The venting got me through. Special thanks also goes to Dr. Morse for sticking with me this whole time even when research didn't run so smoothly. Y'all are great.

Table of Contents

List of Figures vi

List of Tables viii

1 Introduction 1

2 Related Work 6

2.1 Sparse and Missing Depth-Completion 6

2.2 Monocular Depth-Prediction 8

2.3 GAN Data Augmentation 9

3 GAN and VAE Background 11

3.1 Generative Adversarial Networks 11

3.2 Variational Autoencoders 13

3.3 Alternative Models 14

3.4 Mode Collapse 15

4 Synthetic Noise Generation for RGB-D Data Augmentation 17

4.1 Dataset 19

4.1.1 Noise Definition 20

4.1.2 Denoising Pipeline 20

4.1.3 Noise Representation 22

4.1.4 Dataset Organization 24

4.1.5 Discussion of Dataset Design 25

4.2 Model Architecture 26
4.2.1 Setup for VAE Training 28
4.2.2 Setup for Conditioning 28
4.3 Model Training 30
4.3.1 Data Pre-Processing 31
4.3.2 VAE Training 33
4.3.3 Post-Processing 37
4.4 Using our Method for Data Augmentation 40

5 Experiments **41**
5.1 Baseline Method 41
5.1.1 Discussion of Alternate Baseline 43
5.2 Baseline Experiments 43
5.3 Augmentation Experiments 50
5.3.1 Effects of More Synthetic Data 57
5.3.2 Efficiency Compared to Real Data 61

6 Conclusion **63**
6.1 Limitations and Future Work 64

References **67**

List of Figures

1.1 Denoised RGB-D image with real and synthetic depth maps 2
1.2 Different types of camera noise 3
1.3 Damaged RGB-D image from NYU Depth V2 4
1.4 The Intel RealSense D415 and D435 cameras 4

2.1 Sample of noisy RGB-D completion results 7
2.2 Sample RGB-to-depth prediction results 8

3.1 GAN diagram 12
3.2 VAE diagram 13
3.3 Reversible network diagram 14
3.4 Mode collapse 15

4.1 Synthetic augmentation example 18
4.2 RealSense D415 and D435 RGB-D images 19
4.3 Motion removal 20
4.4 Automated denoising pipeline 21
4.5 Explicit noise representation 23
4.6 Encoder Architecture 26
4.7 Decoder Architecture 27
4.8 U-Net comparison 29
4.9 Conditional VAE training setup 30
4.10 Resizing inputs for noise generation 31

4.11 Softening binary dropout masks 32
4.12 Real and synthetic dropout masks 34
4.13 Real and generated depth residuals 35
4.14 Fine-tuning with cGAN training 36
4.15 Post-processing pipeline for generated depth residuals 38
4.16 Post-processing pipeline for generated dropout 38
4.17 Pipeline for generating synthetic noisy depth images 39

5.1 Modified SparseNet architecture 42
5.2 Recalculated proxy ground-truth depth images 44
5.3 Baseline validation curves 47
5.4 Baseline completed RGB-D images 50
5.5 Augmented versus baseline validation curves 52
5.6 3D visualization of augmented error reduction 56
5.7 Synthetic versus real validation curves 59
5.8 Completed RGB-D results using more synthetic training data 60

List of Tables

5.1 Baseline validation mean squared error on the D415 And D435 datasets . . . 45

5.2 Results of baseline experiments 48

5.3 Augmented validation mean squared error for initial experiments 51

5.4 Results of data augmentation experiments 54

5.5 Percent improvement on augmented training sets 55

5.6 More synthetic frames compared with more real frames 58

Chapter 1

Introduction

Innovations in depth-imaging techniques in the past few years have driven the development of consumer-available depth cameras. Depth data, when aligned with RGB photography, forms RGB-D images, which may be applied to such technologies as self-driving cars, robotic navigation, facial recognition [20], object classification, and more. Unfortunately, depth-capture technologies tend to be unreliable, capturing depth maps containing large holes or significant depth noise. Machine-learning techniques have proven to be adept at repairing these images but at the cost of requiring large amounts of pre-completed ground-truth RGB-D images for training [21, 31]. This is problematic, since it is non-trivial to correct depth data, and most solutions either use other automated hole-filling techniques or painstakingly hand-correct every image [30]. Both forms of correction are subject to error and variance, and usually result in sub-par ground-truth targets. As such, most depth-correction datasets use small amounts of carefully-corrected ground-truth data and rely on synthetic data augmentation to boost the size of the training set.

Most depth-completion techniques use random dropout as a proxy for depth noise [21]. Such augmentation assumes accurate depth samples and structureless dropout throughout the image, which is not accurate regarding real data. Real depth noise is not simply random but is conditioned on the geometry and appearance of the color scene. This causes methods trained using random dropout to generalize poorly to real RGB-D images. It is possible instead to train networks to correct real depth noise, but this requires each ground-truth image to be paired with one or more real noisy RGB-D images, which is difficult with limited data. A

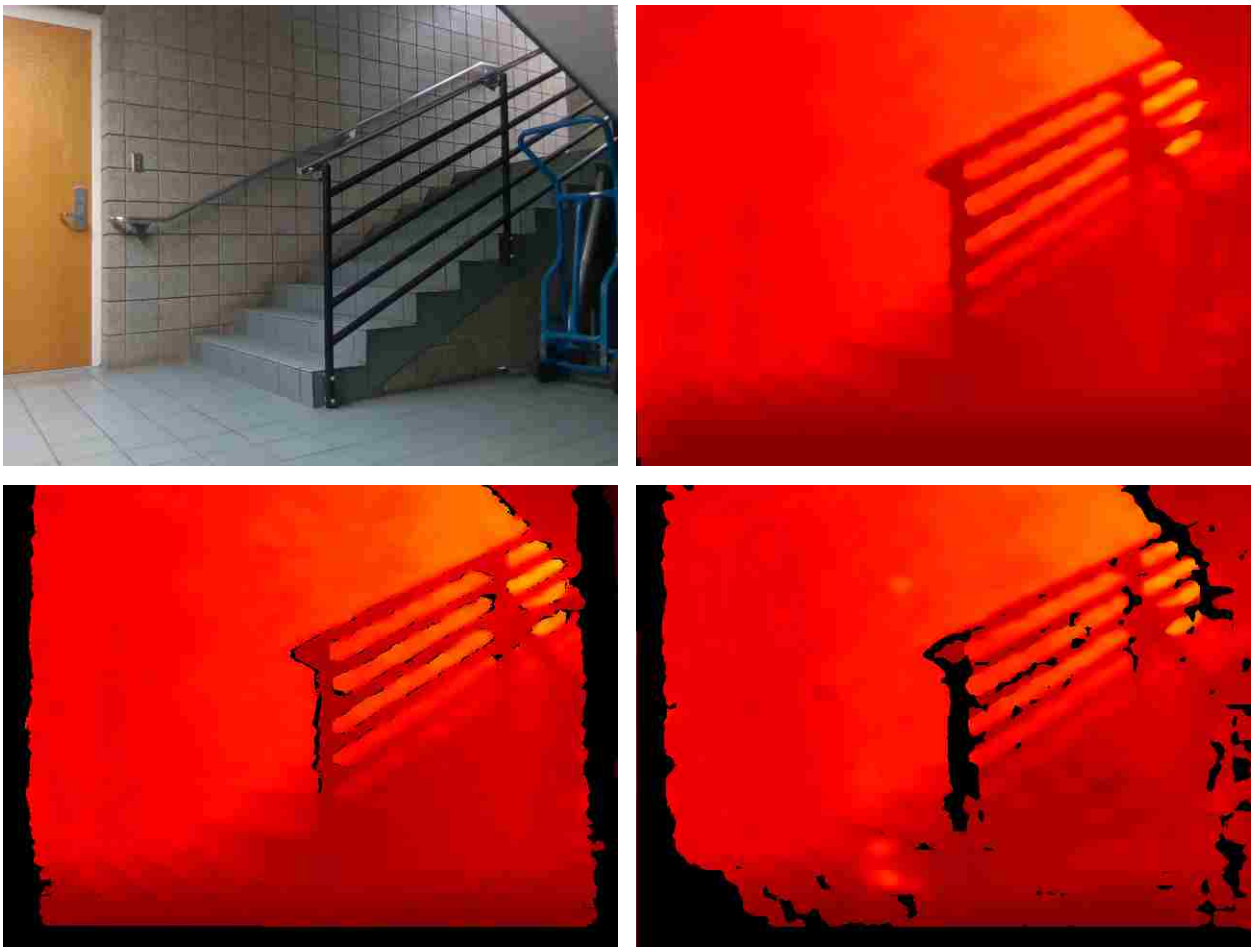


Figure 1.1: Denoised RGB-D image with real and synthetic depth maps. Top: the denoised RGB-D image. Bottom left: the real noisy depth map. Bottom right: the synthetically-generated noisy depth image.

Figure 1.2: Different types of depth-camera noise. Depth cameras exhibit noise differently depending on how they measure distance. In the above examples, each image was captured with an active-lighting/stereo camera. These types of camera tend to struggle with partial occlusions, reflective surfaces, dark colors, and textureless regions.

better form of augmentation would imitate the structure of real camera noise to provide associated cues to the depth-completing network. Depth noise is connected to the type of camera and technology used, however, so a unique depth-completion and augmentation solution would be needed for each individual depth-capture technology.

We present an adaptable solution to the augmentation problem that is motivated in part by recent successes augmenting image-classification datasets using Generative Adversarial Networks (GANs) [1, 5, 22]. In the same spirit, we train a Variational Autoencoder (VAE) to synthesize photorealistic depth noise that may be used as data augmentation in training supervised depth-completion methods. Our model borrows from the Conditional GAN (cGAN) paradigm to generate realistic depth noise and dropout reliant on the underlying geometry and appearance of the scene [11, 23]. We use this model to artificially boost the size of limited depth-completion datasets with realistic camera noise to create additional training pairs. This helps improve baseline performance using a modified version of the sparse-to-dense completion network proposed by Ma et al. [21]. Additionally, instead of

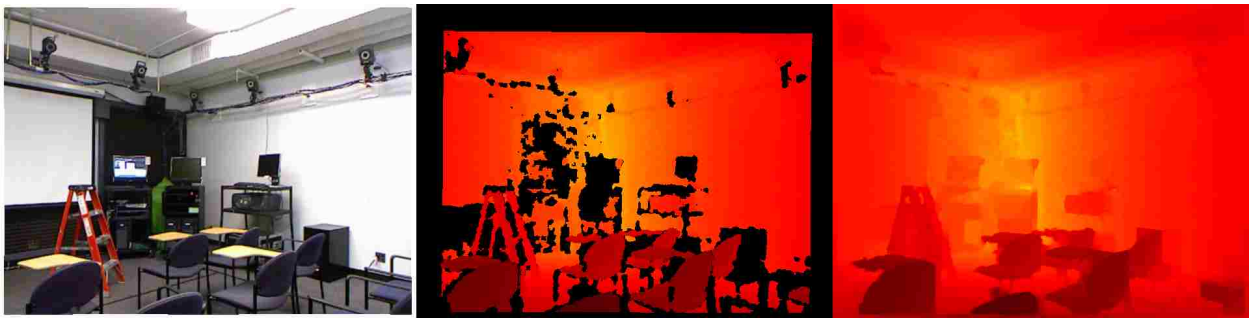


Figure 1.3: Damaged RGB-D image from the NYU Depth V2 dataset compared with its completed ground-truth [30]. There is only one noisy example per clean depth image in this dataset. Left: the original color image. Center: the raw damaged depth map. Right: the hand-corrected ground-truth depth map.

Figure 1.4: The Intel RealSense D415 and D435 cameras from left to right respectively [10].

requiring ground-truth depth data to train, we demonstrate that consolidation of multiple depth images of a static scene can serve as a suitable proxy. Our model may be adapted to a variety of noise distribution and yields improvements over baseline results even when training on small datasets.

Our primary contributions are therefore twofold. First, we have gathered a novel depth dataset consisting of nearly 5.8K still RGB-D images for the Intel RealSense D415 and D435 depth cameras. The dataset is split between the two cameras, and each still image corresponds with approximately 200 sequential depth frames modeling the noise distribution of the respective camera. Although our dataset is large, we demonstrate that only a small subset is required to train a reasonable noise-generating model. Second, we have developed a novel VAE capable of reproducing realistic depth noise, as well as a training routine that may be adapted to a variety of noise distributions. We demonstrate the generalizability

of this network on both halves of the dataset, since both cameras produce unique noise distributions. We further demonstrate the effectiveness of our VAE as an augmentation strategy by showing improvements over baseline results on a modified version of the sparse-to-dense depth-completion network proposed by Ma et al. [21]. By using our method for data augmentation, it is possible to train a reasonable depth-completion method on small or otherwise restricted sets of RGB-D training data.

Chapter 2

Related Work

Depth completion attempts to correct damaged RGB-D images by using RGB and unreliable depth data to make predictions. As depth becomes more unreliable, the solution must rely on monocular RGB-cues with fewer input depths. Taken to the extreme, where no depth data is available at all, depth completion becomes monocular depth-prediction, a related but fundamentally different problem. Monocular prediction works independently of depth-sensing hardware, requiring only an RGB input, and usually performs worse than depth-completion solutions due to inferior data. Although we are not specifically concerned with monocular depth prediction in this thesis, it is closely related to the depth-completion problem, and a passing understanding of the subject is useful to our own work. We discuss some of the more crucial papers to the construction of the depth completion and monocular-prediction problems below, as well as several contemporary methods of using GANs for data augmentation. None of these approaches directly solve the problem addressed by this thesis, but all help to motivate the use-case and construction of our own method.

2.1 Sparse and Missing Depth-Completion

Sparse depth-completion involves inferring depths from an RGB-D input where the depth samples are either randomly or evenly distributed with large gaps of no data inbetween. Because sparse depth completion is a better-constrained problem than monocular depth-prediction, many have proposed non-machine-learning algorithmic solutions. Some have used wavelet methods to produce dense disparity maps from sparse depth samples [9], while others

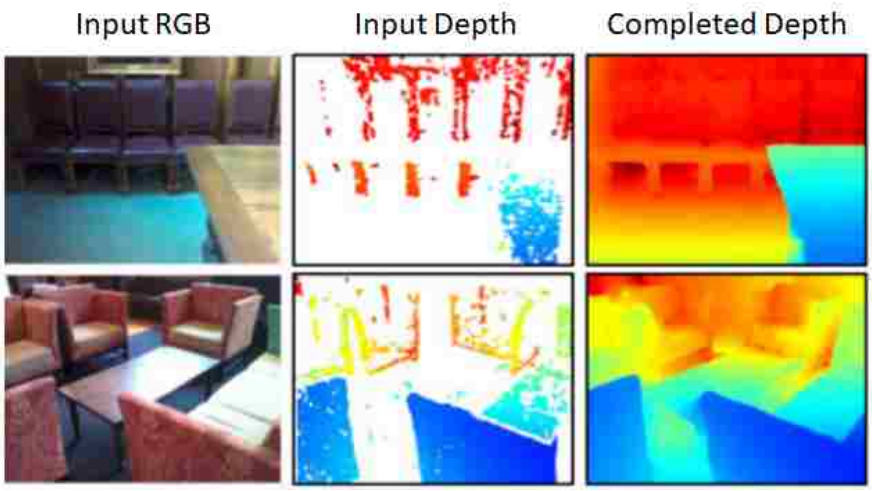


Figure 2.1: Sample of noisy RGB-D completion results. Figure from [31].

have used wavelet-contourlet dictionaries to directly reconstruct dense depth maps [18]. A recent supervised-learning approach demonstrated a 50% improvement over state-of-the-art approaches on the NYU Depth V2 dataset by using a deep CNN with 100 randomly-sampled depth values per input [21, 30]. It is noteworthy that these methods assume accurate depth samples and random dropout, which is not representative of real camera noise. Our noise-generator is designed to replicate realistic depth noise under the assumption that the magnitude and placement of that noise is informative about the true geometry of the scene.

Dense depth completion attempts to reconstruct true depths based on an input RGB-D image with real camera noise present in the depth channel. Previously, some have found success in framing monocular depth-prediction as a discrete-continuous optimization problem [19]. Zhang et al. take this approach one step further by using a deep CNN to predict surface normals from input color, and then globally optimizing for depth using input noisy depths and predicted normals as loose constraints [31]. They leverage point clouds from the Matterport3D dataset to construct a large training set of corrupted/completed RGB-D pairs [2]. This approach is the most similar to the particular technique we would like to augment, however, input depths are only used in the optimization step, meaning that the network does not learn to handle them directly. Our VAE therefore is unable to directly

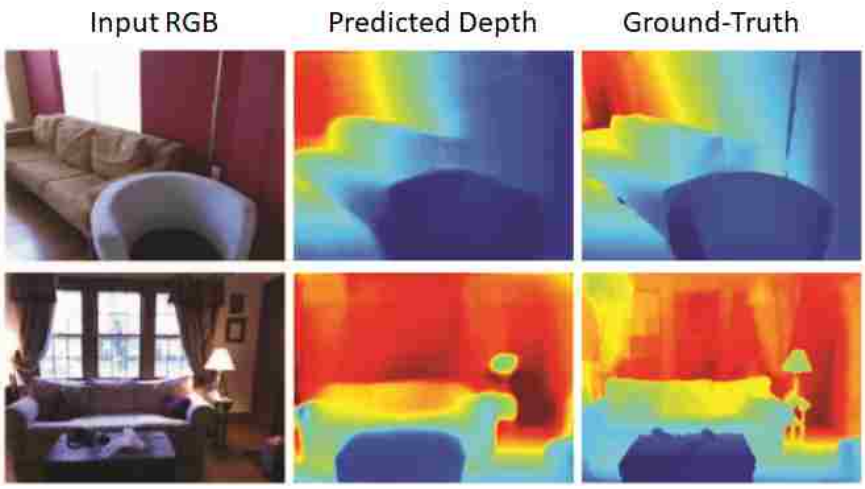


Figure 2.2: Sample RGB-to-depth prediction results. Figure from [14].

augment training for this specific method. Instead, we augment a modified version of the sparse-to-dense depth-completion network proposed by Ma et al. [21]. Our modifications allow the network to better handle dense input depths, and will be discussed in detail later.

2.2 Monocular Depth-Prediction

Monocular depth-prediction attempts to predict depth based on 2D color data alone. This problem is ill-posed, because a single RGB image could have infinitely many valid depth maps. Because most of these depth maps are impossible, however, the problem lends itself well to machine learning. Early approaches used hand-crafted features and Multi-Layer Perceptons (MLPs) to make crude depth approximations [28], but these have been supplanted in recent years by fully-convolutional approaches. Fully-convolutional methods learn their own features during training, and come in a few flavors. These include multi-tiered approaches, straightforward deep CNNs, and multiple-task networks, which combine depth-prediction with related tasks, such as surface-normal estimation and semantic labeling [3, 4, 17]. Recently, deep-convolutional residual networks have outperformed all other supervised monocular depth-prediction methods [14]. However, all of these methods are restricted by limited training data, and tend to produce blurry predictions as a result.

Unsupervised methods address the data problem by using large stereo datasets without ground-truth depth measurements. These methods train on stereo pairs, learning monocular prediction by using estimated depths to warp an input image to its counterpart, and vice versa [6]. Since the loss is based on image warping, no ground-truth depth data is needed for training. Others have found that applying an additional consistency loss or taking a semi-supervised approach with sparse depth inputs helps improve detail in predicted depths [7, 13]. Such depth inputs could possibly be included in a stereo dataset by using an array of aligned depth cameras for stereo capture. However, many types of depth sensors interfere with one another when operated in tandem, and this could invalidate the collected depth data. Recently, both supervised and unsupervised methods have used multi-view stereo with single depth cameras to complete damaged RGB-D images to synthetically boost the amount of ground-truth data available for training [16, 31, 32]. These stereo-completed datasets may also be used for depth-completion training, with the caveat of containing only one noisy depth image per ground-truth.

2.3 GAN Data Augmentation

Our data augmentation method is inspired by recent successes using GANs to generate synthetic training data for small or unbalanced image-classification datasets. These approaches build on cGANS, which produce outputs conditioned on user-specified inputs and are proficient in translating images from one domain to another [11, 23]. The Data Augmentation GAN (DAGAN) uses this idea to produce additional training instances for image-classification datasets conditioned on a user-specified class [1]. The Balancing GAN (BAGAN) takes this a step further by specifically producing additional training instances for underrepresented classes [22]. Recently, Frid-Adar et al. demonstrated the viability of GANs as data augmenters by significantly improving baseline results on liver lesion detection by generating novel lesion images using a GAN trained on a limited dataset [5]. Our VAE is an extension of these

approaches, and produces realistic depth noise conditioned on the geometry, appearance, and other factors of an input clean RGB-D image.

Chapter 3

GAN and VAE Background

Generative machine learning models come in a variety of flavors, including Variational Autoencoders, Generative Adversarial Networks, reversible generative networks, and others [8, 12, 29]. The basic idea of a generative network is to synthesize artificial data from a random input. The synthesized data could be pictures of faces, objects, outdoor scenes, or even pieces of music. In order to generate coherent data, the network learns a low-dimensional encoding of the data domain, as well as a decompression algorithm. Ideally, the network learns a good encoding of the concept that covers the entire vector space, and will translate any input vector into a coherent output. To use facial generation as an example, this means that we could sample a completely random input vector (often denoted as z) and use the generator to convert that vector into a unique, coherent human face. Since we are interested in generating realistic depth noise, we use our VAE to synthesize noise patterns from random input vectors. This allows us to generate as much unique noisy training data as we need.

3.1 Generative Adversarial Networks

Generative Adversarial Networks are an unsupervised machine learning variant that simulates an adversarial game between two agents: the generator and discriminator [8]. The generator attempts to fabricate realistic-looking data to fool the discriminator, while the discriminator tries to determine whether or not input data is real. The generator converts a low-dimensional encoding into a high-dimensional output, and the discriminator produces a binary classification for a high-dimensional input. The two networks are optimized using an adversarial loss, where

Figure 3.1: GAN diagram. The generator converts random noise into realistic-looking data, while the discriminator classifies that data as real or fake. In our case, the generator learns to produce realistic depth noise, while the discriminator determines if that noise looks real or not.

the discriminator tries to minimize its classification error between real and fake data, while the generator tries to maximize the discriminator's classification error on fake data. The generator is updated at each training step by backpropagating error through the discriminator, such that the discriminator actually teaches the generator to produce more realistic outputs. Ideally, training ends when the output of the generator is indistinguishable from real data.

Conditional GANs (cGANs) attempt to control the type of output the generator learns to produce, while still maintaining a degree of stochasticity in the outputs [11]. In standard GAN training, the generator learns to generate coherent data from completely random inputs. It is not trivial to understand how to manipulate the encoding learned by the generator, so it is difficult to control the type of data the generator produces. cGANs solve this problem by adding an additional input to both the generator and discriminator to restrict the type of output generated. For example, a cGAN tasked with learning to produce MNIST (handwritten numeric) digits could be conditioned using a one-hot vector to indicate the number that should be produced [15]. The discriminator learns to reject mismatched image/label pairs, and teaches the generator to create digits accordingly. Ideally, the generator still learns to use the random input vector to manipulate the handwritten style of the output digit. We construct our VAE following this conditional paradigm such that it is able to produce believable depth noise custom to a input ground-truth RGB-D image.

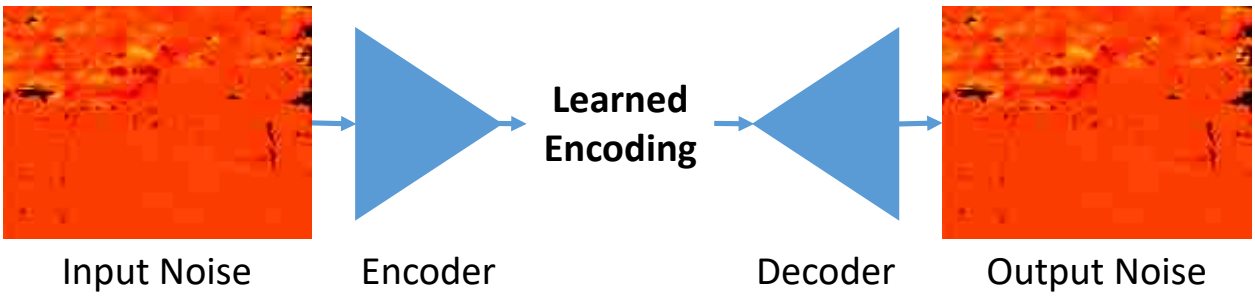


Figure 3.2: VAE diagram. Both Autoencoders and Variational Autoencoders learn a compressed representation of input data. in our case, we use our VAE to learn a compressed representation of depth noise and dropout.

3.2 Variational Autoencoders

Variational Autoencoders are something of a forerunner to the more modern GAN, and are often worse at synthesizing well-detailed outputs. This is because VAEs stem out of the idea of traditional Autoencoder networks, which learn aggressive down-sampling and reconstruction techniques for given data domains [26]. Autoencoders are constructed of consecutive encoder/decoder networks, which may be fully-connected, convolutional, or both. The encoder takes a high-dimensional input, potentially an image, and reduces it to an low-dimensional vector, which the decoder then up-samples to reconstruct the original input. Autoencoders are considered to be unsupervised networks, since they ideally produce the same outputs as inputs. Autoencoders are usually used for data compression rather than data synthesis because they tend to learn poor encodings that are not well-represented over the entire vector space. This means that feeding the decoder network a random input vector will likely only produce chaotic noise rather than a coherent output.

VAEs take Autoencoders to the next level by forcing the learned encoding to be similar to a normal distribution [12]. In this case, the encoder produces two N-dimensional vectors, representing the mean and standard-deviation of the input data respectively. The decoder uses an additional reparameterization trick to probabilistically convert these values into a coherent z-vector, which it then up-samples to produce a recognizable output. The algorithm for training a VAE is basically the same as for training a regular Autoencoder, since both are

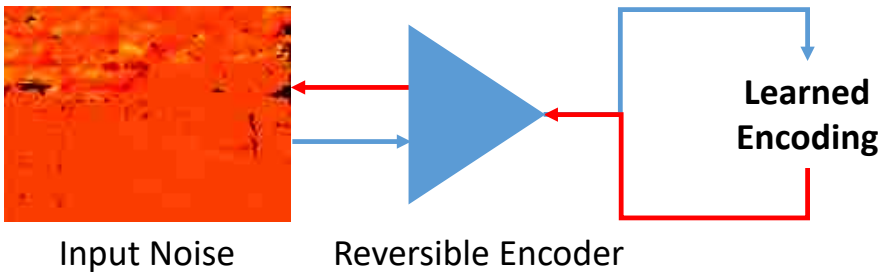


Figure 3.3: Reversible network diagram. RNs learn an invertible encoding for input data. In this example, the network learns a compression for input depth noise which may be passed backwards through the network to obtain the original image.

tasked with learning a low-dimensional embedding for the input and reconstructing it on the other side. Under the VAE paradigm, the learned encoding is well-represented across the entire normal distribution, meaning that any z-vector sampled from the same distribution will correspond to a coherent output. This property makes VAEs generative, since we can pass a completely random vector into the decoder to synthesize novel, believable data.

3.3 Alternative Models

Generative networks is an active area of machine-learning research, and new models are proposed from month to month. Recently, Reversible Networks (RNs) have begun to gain traction as a new generative model that may eventually be competitive with current GAN architectures [29]. RNs are similar in structure to VAEs, except each layer is designed to be invertible. This means that any input to the network may be recovered from its output, such that it is only necessary to learn a half of the encoder/decoder structure, since that half may be flipped to perform the opposite task. RN training therefore focuses on encouraging the learned encoding to resemble a normal distribution, since any encoded input is guaranteed to be recoverable. Once training is complete, the network may be used to generate novel data by passing in random vectors drawn from the appropriate distribution. While RNs may eventually become competetive with GANs, they currently lag behind both GANs and VAEs

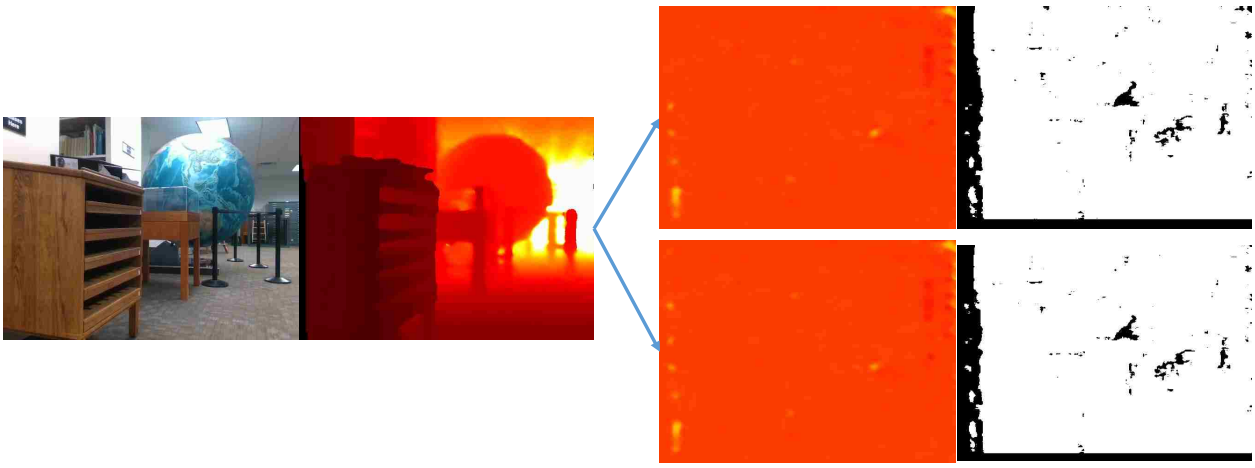


Figure 3.4: Mode collapse. For each RGB-D input scene, the generator produces outputs with almost no variation.

in terms of visual performance, producing blurry images lacking high-frequency detail. Due to this and other issues, we leave an RN solution to depth noise-generation to future work.

3.4 Mode Collapse

A known problem with both cGANs and traditional GANs is *mode collapse* [8, 11]. Mode collapse occurs when the generator learns to output a single value rather than a stochastic sample from the represented data distribution as shown in Fig. 3.4. This often occurs because it is simply easier for the generator to learn produce the same value every time than to learn to produce multiple different kinds of image, especially if one image is enough to fool the discriminator. Alternatively, mode collapse can also occur when the discriminator becomes so powerful that the generator is unable to fool it, causing learning to halt. Mode collapse occurs in Conditional GANs when the generator begins to ignore the random z-vector in favor of learning a one-to-one mapping between the input condition and a believable output. In many domain-transfer applications, such as satellite photos to maps, this kind of mode collapse is more acceptable, since a reasonable transfer function has still been learned. However, data augmentation requires our generated outputs to be highly stochastic, so the generator must be incentivized to avoid avoid mode collapse.

There are multiple ways to discourage mode collapse in both traditional and conditional GANs, but most are dependent on the situation. Isola et al. attempted to enforce stochasticity in generated outputs by applying random dropout to different layers within their generator to limited effect [11]. More recently, it has been suggested that forcing generated images to be reversible by applying an additional reconstruction loss to the generator produces more stochastic results [33]. Other methods include using a batch-wide loss to discourage generation of similar images, and periodically re-training the discriminator using a replay buffer of old generated images, though it is unclear how much the latter strategy helps [24, 27]. Each of these methods increases the complexity of the network architecture and training algorithm, and it is possible that none of them will entirely solve the problem. Because we instead use a VAE model rather than a GAN for noise generation, we do not explore these solutions further in this work.

Chapter 4

Synthetic Noise Generation for RGB-D Data Augmentation

As previously stated, the goal of this thesis is to present a method for generating photorealistic depth noise for use in augmenting limited and existing ground-truth RGB-D datasets. For this to work, we first need a sufficiently large dataset to train our model to produce noise for a variety of different scenes. To train a noise-generating method, we require ground-truth RGB-D images paired with many examples of noisy depth images. While some RGB-D datasets, including NYU Depth V2 and the modified Matterport 3D dataset, already pair ground-truth RGB-D images with noisy depth images, most of these only have one such pair per scene [30, 31]. Experiments demonstrate that this is sufficient to learn some noise augmentation if enough unique scenes are provided, but depth noise varies over multiple shots of a static scene, so it is best to have multiple examples per image. In order to be considered a good method of data augmentation, our model must produce photorealistic stochastic results for each input image, ideally free of synthetic artifacts that will allow the depth-completion method to cheat and recognize augmented outputs.

We therefore construct our own dataset designed to model the underlying depth-noise distribution the Intel RealSense D415 and D435 depth cameras. This dataset contains nearly 5.8K proxy ground-truth RGB-D images, obtained by consolidating approximately 200 noisy depth frames per image. It should be noted that such a large dataset is not necessary to train a depth generator. Our experiments demonstrate that it is possible to train a reasonable depth-completion method using our augmentation strategy on a dataset of only a few-hundred proxy ground-truth images with fewer than ten frames each. This is critical to our method,

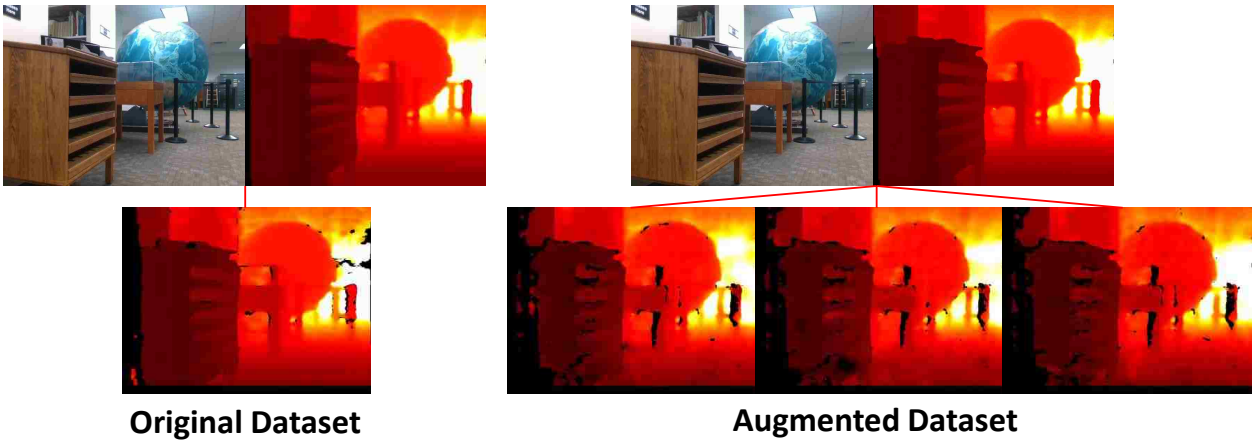
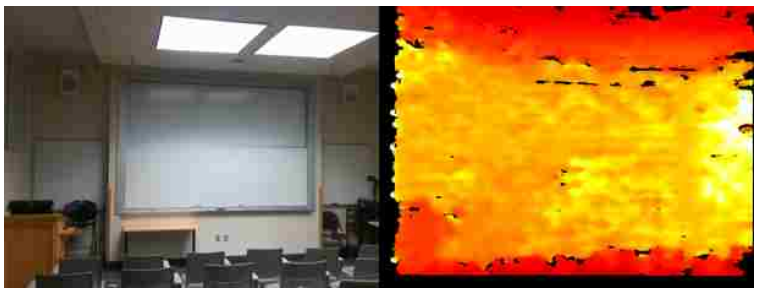


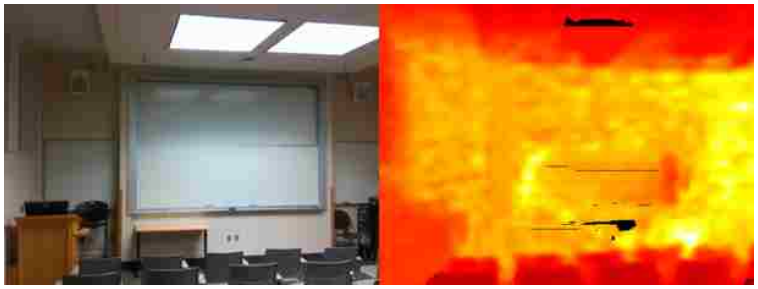
Figure 4.1: Synthetic augmentation example. Our method augments existing or limited datasets by adding realistic noise to denoised images. In this example, we augment an instance of our Intel RealSense D415 camera dataset using a noise generator trained on the same dataset.

since our dataset is not meant to be adapted to depth datasets captured with different cameras. To augment the NYU Depth V2 dataset, for example, one would need to either train a depth generator on the existing ground-truth depth data, or capture additional training data using a Kinect depth sensor. Because our augmentation strategy is effective even when trained on small datasets, the required data collection may be completed in just a few hours, while our dataset took the better part of a year to collect.

We use our dataset to train a generative VAE, leveraging the ideas presented by Isola et al. and Mariani et al. to generate synthetic depth noise conditioned on the latent geometry and visual appearance of the scene [11, 22]. This model is designed not only to generate noise in the depth channel, but also to assign dropout to regions where camera readings often fail. These two kinds of noise are dependent on the structure of the scene, and provide useful information to the depth-completion method that helps generalization to new data. We validate our generative model by using it for data augmentation on a modified version of the sparse-to-dense depth-completion network proposed by Ma et al., and compare to baseline results [21]. We consider our model a success in that it improves upon baseline results without the need for adding additional data, and does even better when more training



(a) D415



(b) D435

Figure 4.2: RealSense D415 and D435 RGB-D images. Note the differences in dropout and depth noise artifacts between the two cameras.

data is provided to the noise generator. We give details on each step of our process in the sections below.

4.1 Dataset

We chose to develop using the Intel RealSense D415 and D435 RGB-D cameras because they are available to the consumer public and are reasonably priced at less than $500 each. Both are designed to capture at a rate of about 30 frames per second and provide reliable depth readings up to about 10 meters. The cameras are technically able to provide much deeper readings but become extremely unreliable past the recommended 10 meters. The cameras measure depth using a combination of stereo imaging and infrared active-lighting that allows them to perform reasonably well even in low-lighting, but rather poorly in bright daylight and outdoor settings. In part because of this limitation, our dataset consists primarily of indoor images, similar to those presented by the NYU-Depth V2 and modified Matterport 3D datasets [30, 31].

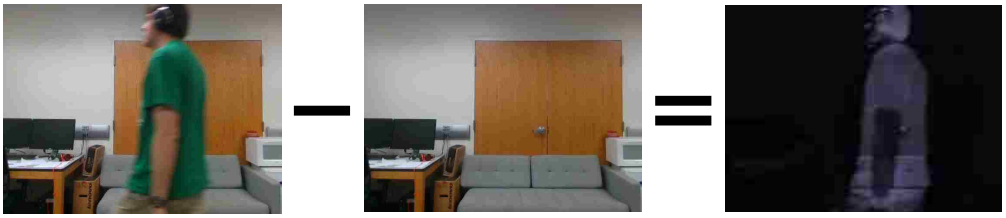


Figure 4.3: Motion removal. We remove frames with motion by taking the absolute difference of each frame with the median-stack image, and removing any frames that violate a threshold. ABS Difference colorized here for clarity.

4.1.1 Noise Definition

We begin by defining explicitly what we mean by depth noise. Noise is manifest in the depth channel in two ways: first by deviations of the depth reading away from the true value, and second by the complete loss of the depth signal at a given location. Both forms of noise are inconsistent across frames of a static scene, and fluctuate with some probability dependent upon the geometry and visual appearance of the scene. Throughout this work, we refer to the first type of noise simply as depth noise and the latter as dropout. Depth noise is visible within the depth channel in much the same way as regular noise is visible in an RGB image, while dropout is indicated by a value of 0 in camera readout. Though the two forms of noise often behave similarly, depth noise is the most severe at extremely near or far distances, while dropout tends to be the worst at object boundaries and on darkly-colored surfaces. The two behave differently enough that we represent them separately in our dataset, using residual-depth images to represent depth noise and binary masks to indicate points of dropout on each frame. See Fig. 4.5 for a visual example.

4.1.2 Denoising Pipeline

In order to train a generator to produce noise, we must first have examples of clean depth images. Rather than require extensive data collection and painstaking hand-corrections of depth maps, we consolidate multiple noisy depth frames from a static scene to achieve a

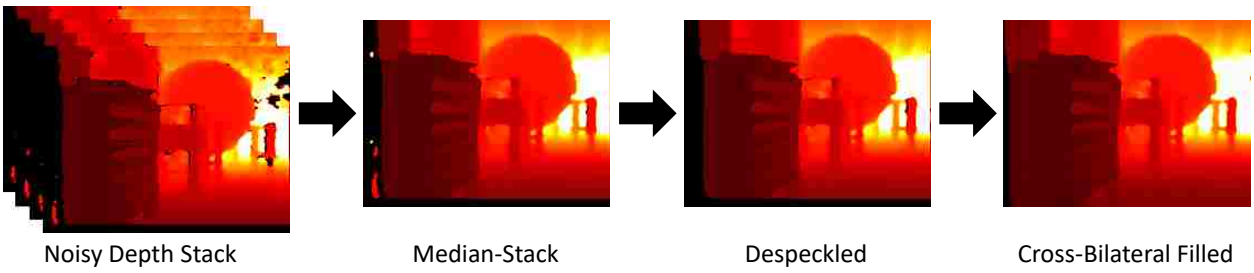


Figure 4.4: Automated denoising pipeline. After frames with motion are removed, we take the median-stack of the depth frames, remove speckles with a binary close operation, and finally apply cross-bilateral filtering to fill in remaining holes.

reasonable ground-truth proxy. This is important, because our method must be adapted specifically to each kind of depth camera, so it is not feasible to require pristine ground-truth data for training. Rather, a user may collect a limited number of static depth scenes and allow the camera to expose for multiple frames on each. By applying our automated depth-denoising pipeline, the user may convert this data into proxy ground-truth, which may then be used to train both the depth-noise generator and the depth-completion network. If the user is able to collect even more data, as we have done with our large dataset, results are even better. It should be noted, however, that gains due to synthetic noise generation taper off as additional real data is collected.

Before running our depth-denoising pipeline, we must first remove all frames from our dataset that contain motion or lighting changes. We have made the assumption that depth noise is stochastic over each frame of a static scene, and this assumption is broken if the geometry or other physical characteristics of the scene change during filming. By removing all motion from our training set, we help to ensure that the noise generator only learns noise patterns caused by the actual camera hardware, not changes in the environment. Fortunately, this step is easily automated. To remove frames with motion, we first take the median-stack over all the RGB frames in a scene. Next, we subtract this median-stack image from each individual RGB frame, and return the absolute difference between the two, as shown in Fig. 4.10. We remove any frames from the dataset with an absolute difference above a certain

threshold, and further remove any scene from the dataset with fewer than 50 valid frames. This results in some scene having fewer frames than others, though experiments demonstrate that even just a few frames are acceptable for generating proxy ground-truth.

After removing all frames with motion, we use our depth-denoising pipeline to consolidate the remaining depth frames into proxy ground-truth. To begin, we take the median over the stack of remaining depth frames. This yields a reasonably-well denoised depth image, often with most dropout regions automatically filled. Depending on the camera used, however, the resulting median-stack image often contains speckles and other outlier shapes and erroneous readings produced by variation in the camera. We remove these using an aggressive binary closing operation with a 15×15 circular kernel. This step may be adjusted or ommitted depending on the required dataset, but proved useful for denoising both of our camera distributions. Third, to fill any remaining holes, we apply the cross-bilateral filtering function used by the NYU Depth V2 data toolkit. Cross-bilateral filtering does not fill every hole in each image, but it provides reaonable approximations for regions where data is still missing. These steps are illustrated by Fig. 4.4.

4.1.3 Noise Representation

Our dataset explicitly represents both types of noise by capturing still RGB-D videos and taking the median over the stack of frames. This provides a moderately denoised and hole-filled RGB-D image that we use as pseudo-ground-truth for training the generator. To represent noise and dropout, we subtract the denoised images from each of the corresponding frames to obtain residual depth frames. We further separate the two kinds of noise from each of these frames by saving out individual dropout masks per frame, and filling the dropout regions in the residuals with a mean value. The frames are saved in tandem with the original depth readings so that they may be used to learn noise and dropout patterns over time for each associated denoised image. In the remainder of this work, we will refer to each denoised image with its collection of depth, residual, and dropout frames as a scene.

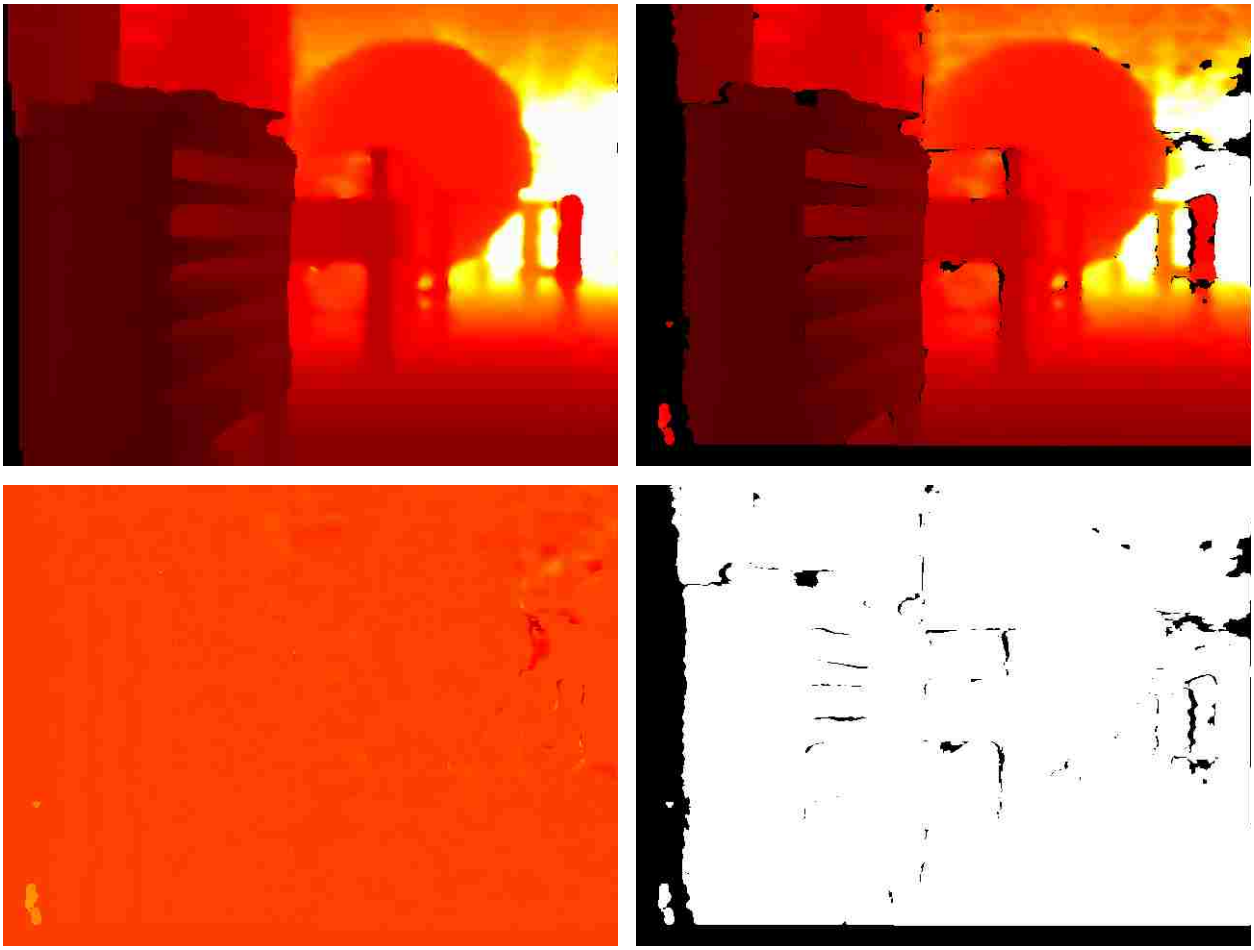


Figure 4.5: Explicit noise representation. We separate out depth noise and dropout by taking the difference of the raw depth image with the denoised depth image, and marking dropout regions with binary masks. Top left: the automatically-denoised depth image. Top right: a raw depth frame. Bottom left: the resulting residual-depth image. Bottom right: the corresponding dropout mask.

4.1.4 Dataset Organization

Our dataset consists of approximately 5.8K unique scenes (2.9K for each camera) where each denoised RGB-D image is paired with a set of 200 noisy depth frames. Since some of our scenes contain extremely deep depth values, we clip all of the depth and residual frames to 10 meters for the sake of consistency throughout the dataset, similar to the structure of NYU-Depth V2 [30]. Additionally, since we remove any frames containing motion, some scenes contain fewer than the standard 200 frames. In these cases, we completely remove any scene from the dataset with 50 or fewer valid frames to ensure that each scene has enough frames for proper denoising. Removing frames with motion from our dataset helps to ensure that each scene is as still as possible, and therefore only contains variance in the form of native camera noise. This helps to make the task for our noise generator as explicit and direct as possible.

We standardize our residual frames by scaling them into the range [0, 1], while dropout masks are represented using simple binary masks. Because dropout may be confused with standard depth noise, we are careful to mask out all dropout regions in each residual frame so that the two factors are not confounded. Because our dataset is large in terms of disk space, we save our noise and dropout masks as one-channel PNG images. This causes us to lose some precision to image quantization, but preserves the general shape and appearance of noise artifacts, which is sufficient for our purposes. It should be noted that all of our original depth images are saved as raw NumPy files, and preserve the same depth precision originally captured by the RealSense cameras. This allows us to recalculate depth residuals and dropout masks on the fly if necessary, as we do in many of our experiments. We also use these original NumPy files, clipped to 10 meters each, to train our modified version of the sparse-to-dense depth-completion method [21].

We reserve 500 scenes from each split of the dataset to serve as validation sets for depth-completion training. These validation scenes were captured in separate environments from the rest of the dataset to prevent overfit by environment memorization. All validation

scenes were captured indoors, with hallways, rooms, and furniture similar to the training scenes, but all captured in completely different buildings with different architectural styles, textures, and lighting conditions. To prevent accidental cross-contamination of the training and validation sets, we refrain from training or validating our noise generator on these scenes. In our experiments, the noise generator is only trained on provided subsets of the training scenes, while the depth-completion method is validated on the entire validation set. This helps provide a consistent error metric across each of our experiments, and ensures that the depth-completion method is not overfitting to the training data.

4.1.5 Discussion of Dataset Design

It is worth noting that our dataset is already in the format required to train an RGB-D completion method. Indeed, we do so in our experiments to demonstrate improvements over baseline capabilities. However, it should be noted that such a method is limited in performance by our substitute ground-truth images. Our proxy ground-truth RGB-D images still contain significant amounts of noise and dropout, and are poor substitutes for hand-tuned ground-truth depth images. Additionally, training a good method would require the bulk of our dataset, which took the better part of a year to collect. Since each depth-camera produces a different noise distribution, a method trained on our dataset would not generalize to data captured by a Kinect or a LiDAR sensor. Since it is impractical to gather datasets of this scale for each applicable kind of technollogy, we demonstrate that it is possible to achieve better-than-baseline results on limited-size datasets by using our augmentation method. Most of our experiments are therefore conducted only on small subsets of our data, and yet demonstrate impressive improvements over baseline tests.

Another potential concern is the necessity of separating out noise and dropout into explicit representations in the dataset. Doing so requires more disk space, since both types of noise are implicitly represented in the raw depth frames already. Our generator produces residual and dropout frames, which must be separately applied to ground-truth to produce

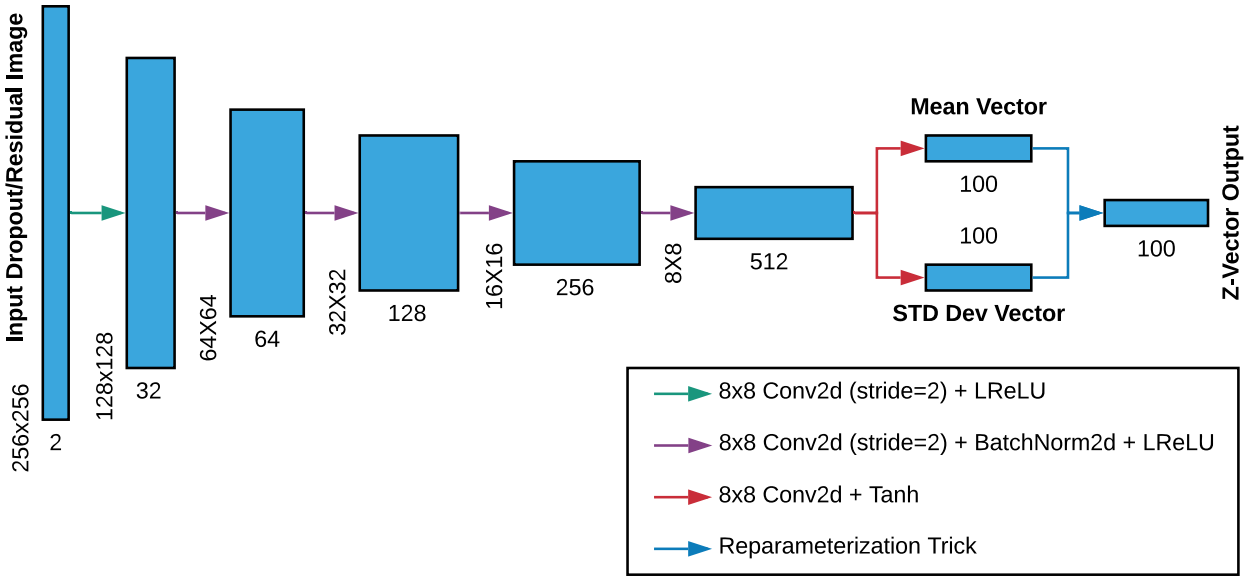


Figure 4.6: Encoder Architecture

noisy images, but it is possible to have the generator perform both steps. In our experience, however, the generator expends most of its learning simply trying to replicate the geometry contained in the input RGB-D image, and tends to severely blur the output. Rather than applying noise, the generator tends to produce smoothed depth images with blurry holes. Separating the types of noise helps disentangle the different tasks, and, in our experience, leads to visually superior results.

4.2 Model Architecture

Our VAE architecture is a variant of the Deep Convolutional GAN (DCGAN) architecture proposed by Radford et al. [25]. The DCGAN architecture was originally used for simple tasks such as generating MNIST digits and CIFAR-10 images and is relatively shallow when compared with many modern deep neural networks. The formulation consists of two separate networks, namely the generator and discriminator, which use decoder and encoder architectures respectively. We concatenate these two networks to form the encoder/decoder architecture needed for VAE training, as well as deepen the network to handle much larger images. The encoder architecture is shown in Fig. 4.6 and consists of five 8×8 2-strided

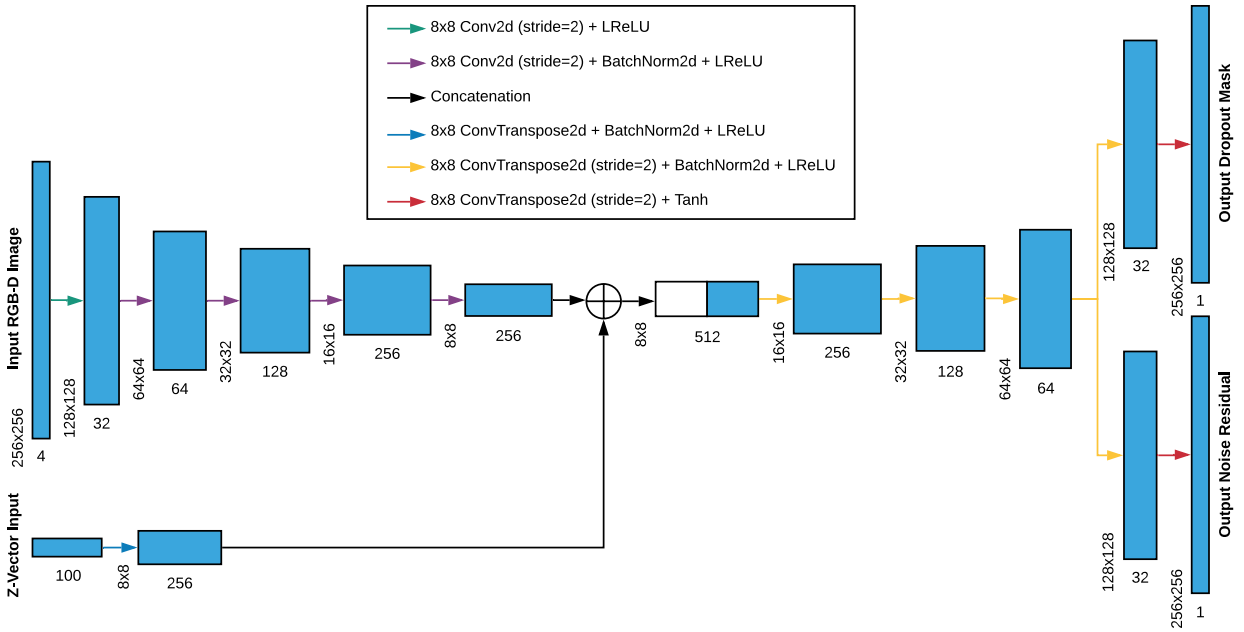


Figure 4.7: Decoder Architecture

convolutional layers with two unstrided 8×8 convolutional output layers that produce the mean and standard deviation vectors respectively. We follow the pattern of the DCGAN architecture in using batch-normalization on all intermediate layers, and Leaky-ReLU activation functions on all but the output layers, which use tanh activations instead. We likewise use strided-convolutions in place of pooling operations to allow the network to learn its own down-sampling function, the same as in the DCGAN formulation.

The decoder architecture is shown in Fig. 4.7 and is significantly different from the standard DCGAN generator design. To support conditional training, the decoder accepts two inputs: the encoded z-vector, and a denoised 4-channel RGB-D image. The z-vector is simply up-projected using an 8×8 unstrided convolution, while the RGB-D input is passed through a down-sampling network mirroring the first five layers of the encoder architecture. The results of the two input streams are concatenated together, and then passed through three 8×8 transpose convolutions with strides of 2 for up-sampling. To improve generated noise quality, the decoder splits into two output streams that produce depth-noise and dropout estimates respectively. Both output streams use two 8×8 transpose-convolution layers with

strides of 2, the same as in the previous up-sampling layers. Like the strided convolutions in the encoder, strided transpose-convolutions let the network learn a custom up-sampling function. All intermediate layers, with the addition of the z-vector up-projection layer, use batch-normalization. Each layer also uses a Leaky-ReLU activation except for the two output layers, which both use tanh activations.

4.2.1 Setup for VAE Training

The bottleneck between our encoder and decoder architectures is modified to support VAE training, which enforces the learned encoding to be similar to a normal distribution. In support of this pattern, the encoder architecture produces estimated mean and standard deviation vectors as separate outputs. During the forward pass, the reparameterization trick uses these two predicted vectors to estimate a concrete z-vector, which is then passed into the decoder portion of the network. Besides this step, training the VAE is identical to training a standard Autoencoder. This is because the method used for estimating the z-vector is differentiable, and does not interfere with backpropagation during training. Since the decoder architecture produces two separate outputs, we concatenate them into a single 2-channel dropout/residual image during the forward pass, and apply the network training loss individually to each channel. At runtime, we discard the encoder architecture and simply pass random z-vectors with conditioned RGB-D images into the generator, since this is all that is needed for noise generation.

4.2.2 Setup for Conditioning

It's worth discussing that VAEs are not usually conditional. Typically, a VAE accepts only one input and learns a compressed encoding that embodies all of the features of the target domain in a single vector. In our case, this is not practical, because we want every possible z-vector to be valid on all possible input scenes. For this to work, the z-vector needs to only encode general information about the placement and magnitude of the noise and depth

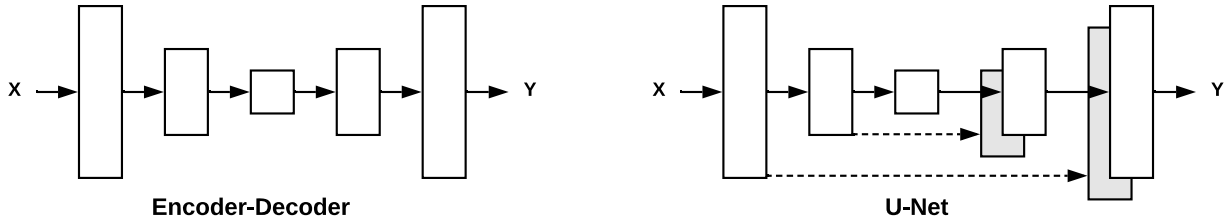


Figure 4.8: U-Net comparison. Compared with standard encoder-decoder architecture. Graphic from [11].

holes, not any geometric or material cues. Otherwise, most randomly-sampled z-vectors will not be appropriate for any given scene, since the embedded geometric information would be completely random. This is why we only add a conditional input stream to the decoder portion of the network, since we don't want the encoder inserting scene cues into the learned embedding. This disentangles the tasks between the two networks, and helps ensure that our model generalizes to new scenes as intended.

Since the basic DCGAN architecture does not support conditioning, we modify the structure as shown in Fig. 4.7 to accept denoised RGB-D images in addition to the standard z-vector as input. This same type of conditional input is used in the traditional cGAN formulation, though the authors of that method use a U-Net structure rather than the modified encoder-decoder architecture we use [11]. U-Nets are useful in that they tend to produce better-detailed outputs thanks to skip-connections between corresponding down and up-sampling layers, as shown in Fig. 4.8. These skip connections allow high-frequency data to skip past the network bottleneck, but may also circumvent real learning deep in the network if the target output is too similar to the network input. It is possible that our architecture could benefit from skip connections, since it is unlikely to be able to cheat by passing information through with no modification. However, early experiments demonstrate a sufficient level of detail in generated outputs that we find skip-connections to be an unnecessary complication to the network architecture.

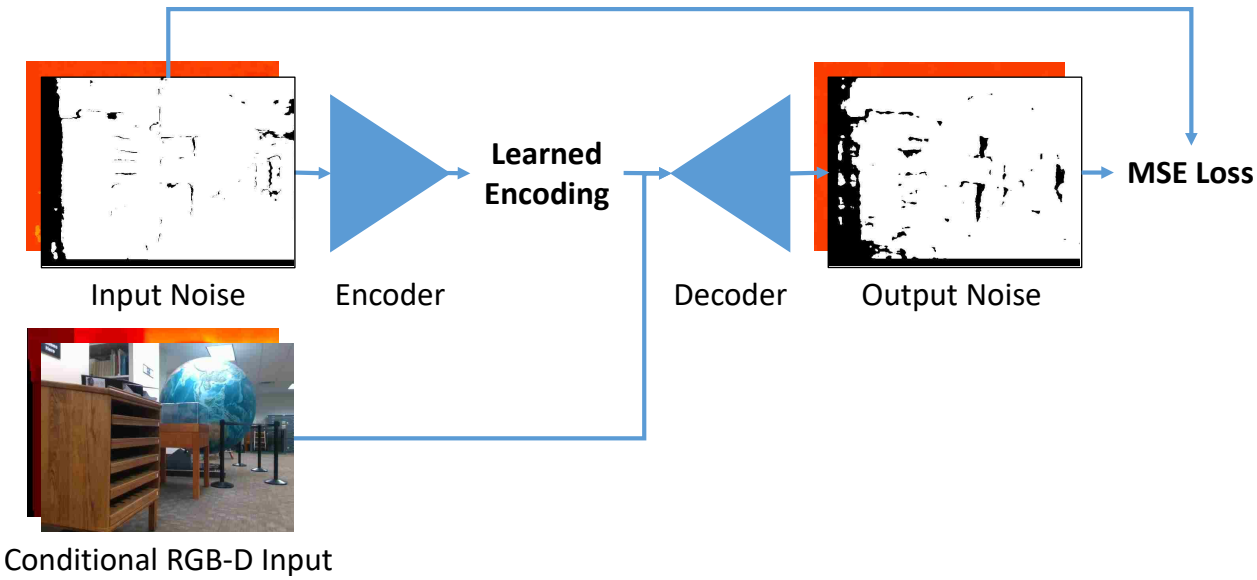
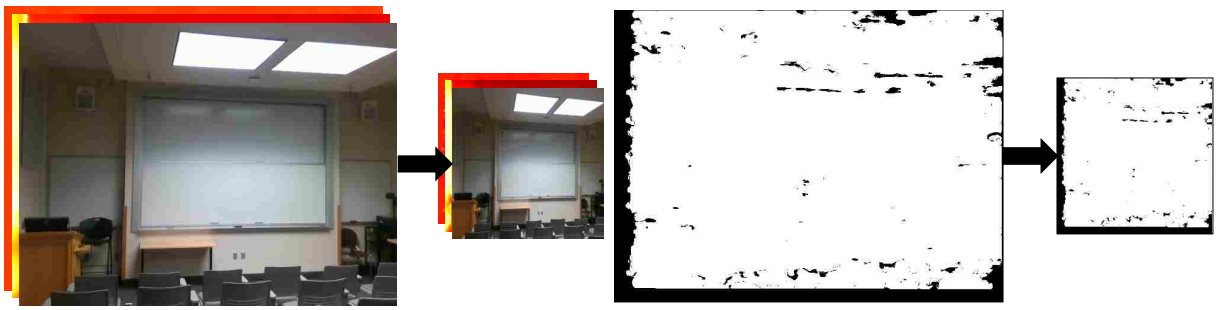


Figure 4.9: Conditional VAE training setup

4.3 Model Training

Because both the D415 and D435 cameras produce different kinds of depth noise, we train separate generators on each split of our dataset so that the model will not confuse the respective noise distributions. During training, we batch out data from the chosen training split in batch sizes of 20, sampling 20 random scenes and pairing each denoised RGB-D image with a randomly selected dropout/depth-noise pair from the respective scene. The denoised RGB-D images are used for all conditional inputs into the model, while the concatenated dropout and depth-noise images are fed directly into the VAE encoder. We optimize the model with regards to a scaled pixel-wise mean squared error (MSE) loss shown in Eq. 4.1 and cease training after no improvements have been made for 50 epochs. We consider an epoch to be one complete pass through all of the training scenes and each of their respective noise and dropout frames.



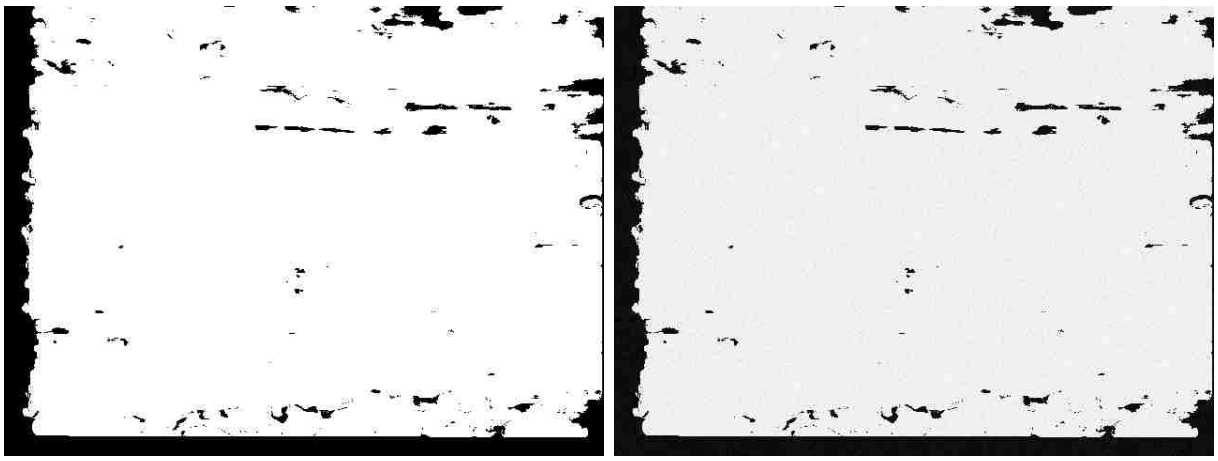
(a) Bilinear Downsampling (b) Nearest-Neighbor Downsampling

Figure 4.10: Resizing inputs for noise generation. The color, depth, and residual images are downsampled using bilinear interpolation. The dropout masks are downsampled using nearest-neighbor interpolation to preserve binary values.

4.3.1 Data Pre-Processing

Before each batch, we take a pre-processing step that transforms the data into the proper range and configuration. All of the images in our dataset are saved as 640×480 PNG files with pixel values in the range of [0, 255], while our network is designed to operate on images of size 256×256 with values scaled between -1 and 1. Therefore, at runtime, we not only scale all image values into the range [-1, 1], but we also resize the input images to be 256×256 squares using bilinear interpolation for all images except the dropout masks, for which we use nearest-neighbor interpolation. This has the problem of warping image geometries and textures, which could potentially interfere with network learning and generalization. However, the change in aspect ratio is not terribly severe (4:3 to 1:1), and since we are reducing the dimensions of the input images rather than expanding them, we manage to avoid introducing severe artifacts. Our experiments demonstrate reasonable ability to generalize even to unknown scenes, and empirical evidence demonstrates that generated noise appears photo-realistic despite the warping. This is true even after re-warping the generated noise image to match the original RGB-D scene.

Additional pre-processing is needed in order to train the network to regress binary dropout masks. While the depth-noise images may be regressed directly, binary classification is usually done using a two-layer output channel and a Cross-Entropy loss function. This is



(a) Binary Labels (b) Soft Labels

Figure 4.11: Softening binary dropout masks. Left: dropout mask before softening. Right: dropout mask after softening. Label perturbations scaled up for visibility.

because it it easier for a network to assign confidence to a predicted class rather than regress to an exact binary pixel value. Constructing the network to support both kinds of output is difficult, however, especially in designing a balanced loss function. To overcome this difficulty, we use instead use "soft" binary labels for dropout classification. To do so, we randomly perturb the binary dropout masks by small amounts in the range of [-.1, .1] and clip the results to the range [-1, 1]. This allows some degree of noise in the network's predictions, and helps to prevent layer weights from over-saturating. It further allows us to use the same MSE loss for both our depth-noise and dropout outputs. The generated masks are then binarized during the post-processing step by thresholding about 0 to produce binary masks.

For training the VAE, we forgo most forms of standard image augmentation in order to not accidentally violate the conditions of the noise distribution. The D415 camera, for example, produces noisy black bars on the left and bottom edges of its captured depth images due to a restricted depth diagonal field-of-view angle, as shown in Figs. 4.2 and 4.14. Since these specific artifacts remain in place regardless of the orientation of the camera, we do not wish to confuse the generator by training it on flipped, rotated, or randomly cropped versions of the noisy images. Despite this, our dataset has a kind of built-in data augmentation, since

each scene is associated with between 50 and 200 unique depth frames. This high degree of variability in the inputs helps prevent the model from overfitting, and allows it to generalize well to unknown scenes. It is likely that there are other methods of data-augmentation that would be useful in training better versions of our VAE model, but due to the already high-quality of our results, we leave this to future work.

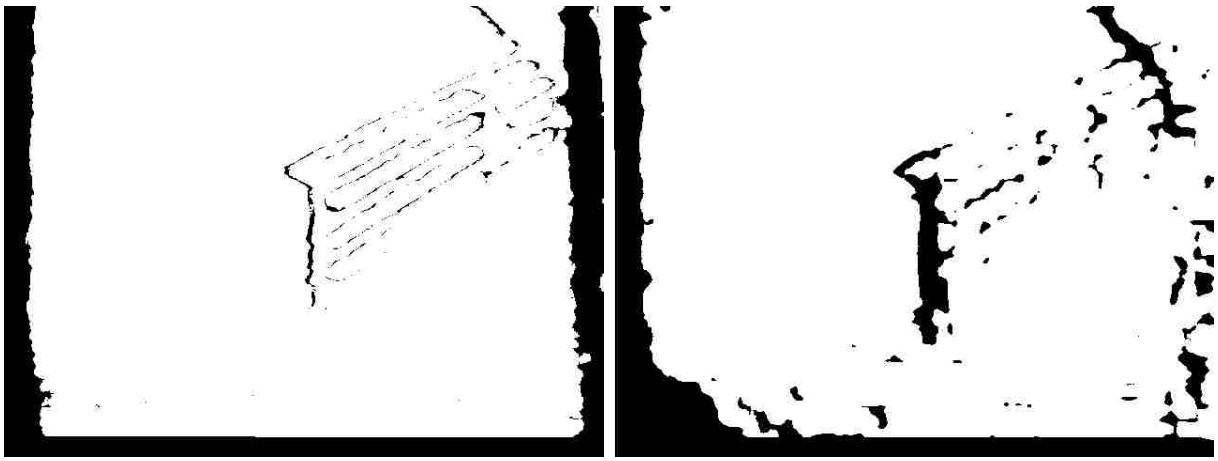
4.3.2 VAE Training

We train using a simple unsupervised Autoencoder training algorithm that passes the noise and dropout data into the encoder end of the network and then reconstructs them on the other side, as shown in Fig. 4.9. Our method takes an additional denoised RGB-D input into the conditional stream of the decoder network but otherwise functions the same as a traditional Autoencoder. We optimize our model using the loss function

$$\mathcal{L}_{vae} = \lambda \| r^{'} - r \|^2 + \| d^{'} - d \|^2 \tag{4.1}$$

where λ is a scalar weight, $r^{'}$ is the generated depth residual, $d^{'}$ is the generated dropout mask, and r and d are the input depth residual and dropout mask respectively. In our experiments, λ is set to 100. We the weight λ to account for the fact that error in the generated dropout masks is often several magnitudes larger than error in the generated depth residuals. This is because the residual-depth images tend to have small values clustered closely about 0 while the dropout masks use larger numbers for binary classification. Scaling the error of the generated residual images helps the errors of both kinds of generated noise to reduce at about the same rate.

We train our model on batch sizes of 20 using a learning rate of 2e-4 until convergence. Similar to a supervised method, we test our model after each training epoch on a held-out validation set to check its generalization. Unlike training a GAN, there is no risk of mode-collapse while training our VAE, though we do still need to be careful of overfit. To help

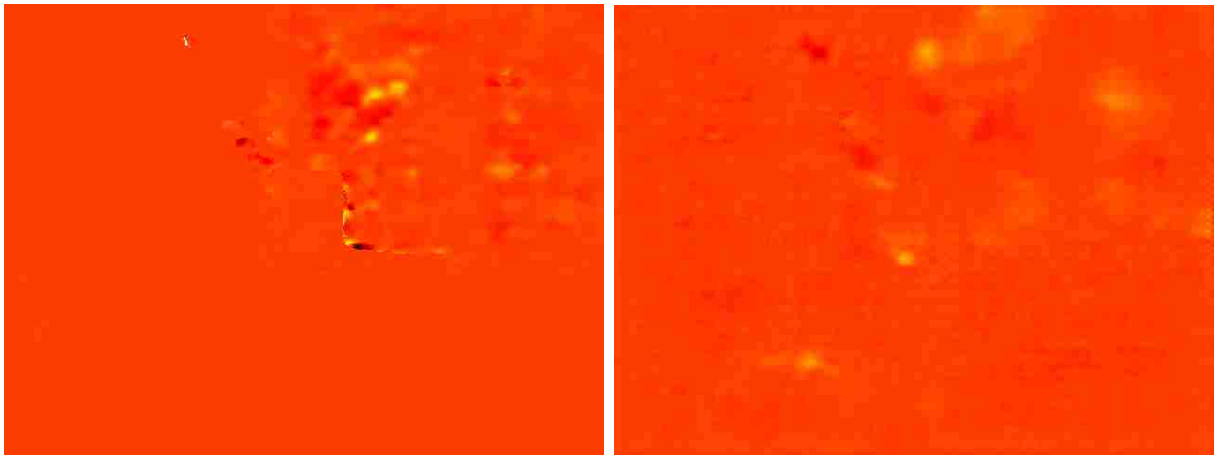


(a) Real (b) Synthetic

Figure 4.12: Real and synthetic dropout masks. Note that while the placement of the generated dropout is generally correct, the size and shape of the synthetic artifacts tend to be larger and rounder than the real dropout artifacts. Despite the differences, generated noise is similar enough to real noise to be used effectively as data augmentation.

avoid overfitting to the training data, we terminate training early if the network proceeds for 50 epochs without improving error on the validation set. The required length of training varies with the size of the dataset, with larger training sets requiring fewer epochs. Because of this, on large datasets it is possible to terminate after only a few epochs if no improvements are made, though, for the sake of consistency, we keep that number to 50 throughout our experiments. We also experimented with reducing the learning rate periodically during training, but noted no meaningful improvements in network loss and so leave the learning rate static throughout.

The only notable drawback to using our VAE formulation is that generated noise images tend to lack high-frequency details and may contain unrealistic blob artifacts in generated masks. Additionally, a distinct grid pattern is visible in many of the generated residual-depth images, which prevents outputs from appearing entirely photorealistic. Fortunately, the grid-like artifacts may be removed in post-processing and do not severely impede the models ability to act as a data-augmentation strategy. Our cGAN model tended to produce sharper noise images without the grid artifacts but failed to do so with a satisfactory degree of



(a) Real (b) Synthetic

Figure 4.13: Real and generated depth residuals. The synthetic residual has been resized, but has had no other post-processing steps applied. Note the grid-like artifacts contained in the generated image. These do not appear in every generated residual but may be removed by using a median-blurring filter.

variability. On the other hand, our VAE model is able to produce exceptionally diverse noise and dropout images appropriately conditioned on RGB-D inputs. This advantage, along with the simplicity of VAE training, makes our VAE formulation preferable to a cGAN model for our purposes. Even despite the the generated artifacts, the VAE model is able to produce sufficiently-realistic depth noise with enough variabilty to improve baseline results on the modified sparse-to-dense method.

Fine-Tuning with cGAN Training

Since the biggest limitation of our VAE is loss of high-frequency detail, and because the two architectures are so similar, we explored the option of fine-tuning our model using cGAN training after VAE training has converged. Indeed, this approach is used by other GAN data-augmentation strategies to reportedly good effect [22]. The idea is to use VAE training to force the network to learn a good embedding for the target data distribution, and then shift over to GAN training to learn to produce fine details and other elements that might be missed training with a per-pixel loss. At the time of GAN training, the encoder/decoder

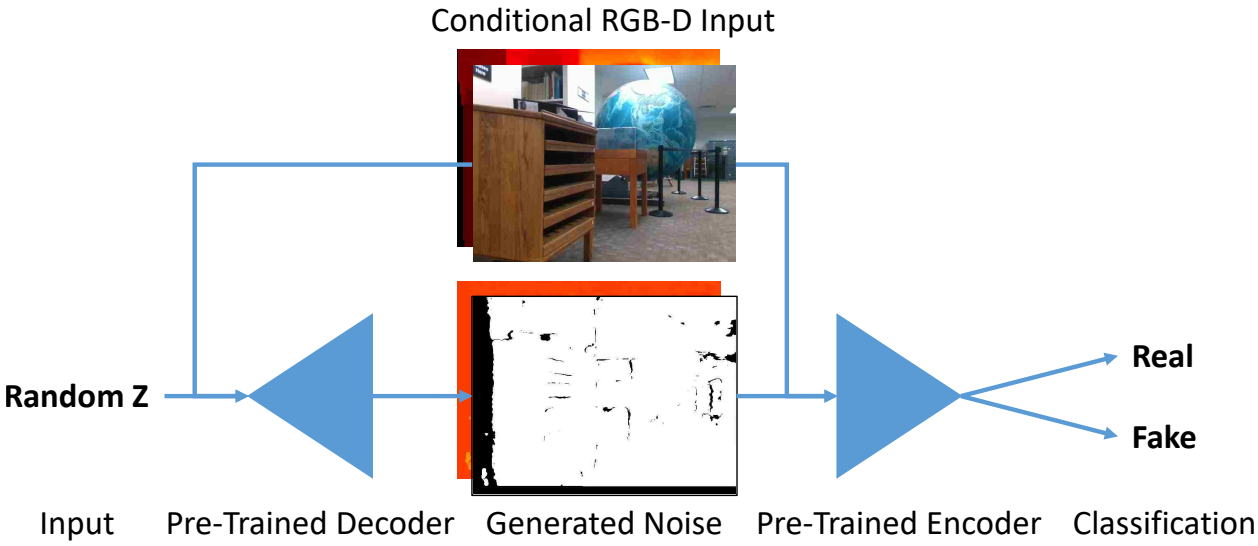


Figure 4.14: Fine-tuning with cGAN training. The pre-trained decoder and encoder structures are separated and used as the generator and discriminator models respectively. cGAN training requires both models to be conditional, which is problematic for our setup, since only the decoder takes a conditional input.

halves of the VAE model are simply separated and used respectively as the discriminator and generator for GAN training. This is especially useful for avoiding mode collapse, since in this instance, the generator has already learned a good embedding for the chosen image space, and will need only minor tweaks to improve the generated images further. Since variety is built in, the network naturally avoids learning to repeat the same generated outputs.

While this approach has worked for other augmentation strategies, we found that attempting to fine-tune our VAE with cGAN training tended to undo all the learning the network had previously done. This is due in part to the fact that our VAE formulation is not exactly the same as the cGAN formulation, not only because the outputs of the encoder network are different, but also because the encoder network is not conditional in our VAE formulation. As such, an entirely new network must be trained to act as the discriminator model which has not had the benefit of VAE training. This new discriminator model tends to cause the generator to obliterate its previously learned weights, even if it is given a substantial pre-training period before moving to real GAN training. The end result is that cGAN fine-tuning usually turns into cGAN training from the ground-up, with no time saved

and exactly the same problems as regular cGAN training. Finally, because our generated VAE results were of such high quality, we determined that it was not necessary to explore this route further.

4.3.3 Post-Processing

Noise images produced by our model are 256×256 square, while most depth-completion methods expect the dimensions of the input image to be approximately 240×320. As before, when we scaled down our input images for training the generative model, we use bilinear interpolation to resize the output images appropriately. This causes some blurring and distortion artifacts in our predicted masks, but since we are using these noise images as augmentation for existing proxy ground-truth, the overall effect of these artifacts is small. It is possible to adjust our model to produce images of the correct scale and aspect ratio, but doing so requires a significant overhaul of the network architecture. Restructuring our network in this way led to memory issues related to the introduction of additional layers, and required us to train with significantly-smaller batch sizes. Because of the already good performance of our smaller architecture, we leave exploration of deeper models to future work.

As previously described, generated residual-depth images tend to contain grid-like artifacts that reduce the plausibility of the fabricated noise. These artifacts must be removed before using our generated noise images for augmentation, because they are regular enough that the augmented method could potentially use them for image memorization. We eliminate the worst of these artifacts using a spatial median blur, which preserves the major shapes and characteristics of the generated depth residuals while suppressing high-frequency artifacts. For our purposes, we use a median filter with a 7×7 kernel, which is sufficient to remove the most noticeable of the grid-like artifacts. The median filter is computationally efficient, so this additional post-processing step does not significantly slow down noise generation. The post-processing pipeline for generated residuals is shown in Fig. 4.15.

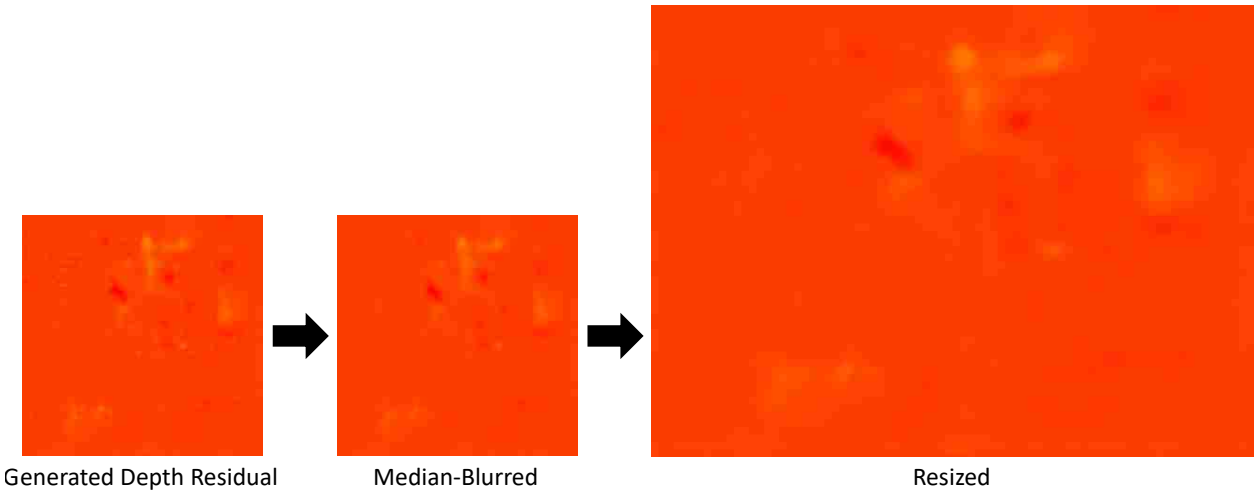


Figure 4.15: Post-processing pipeline for generated depth residuals. We apply median-filtering before resizing for efficiency, but this step may be conducted in any order.

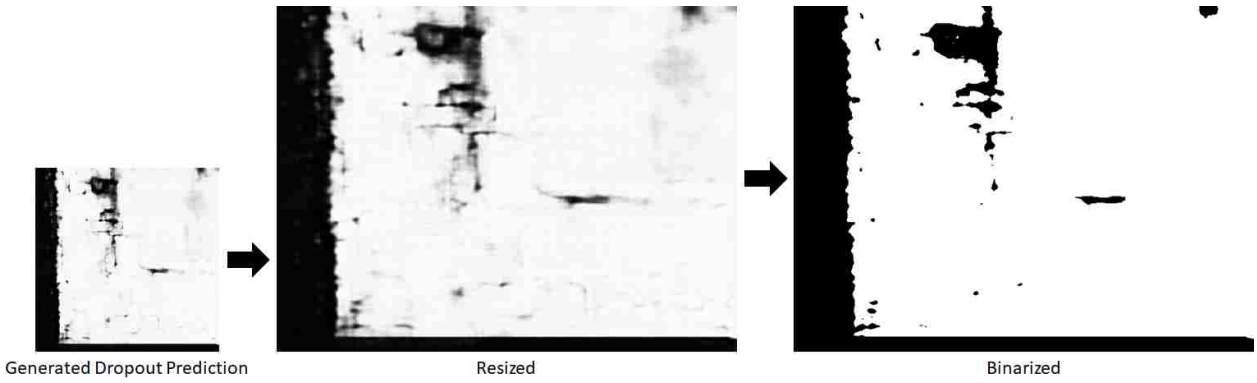


Figure 4.16: Post-processing pipeline for generated dropout. Note that we resize before binarization to avoid creating pixelated dropout boundaries.

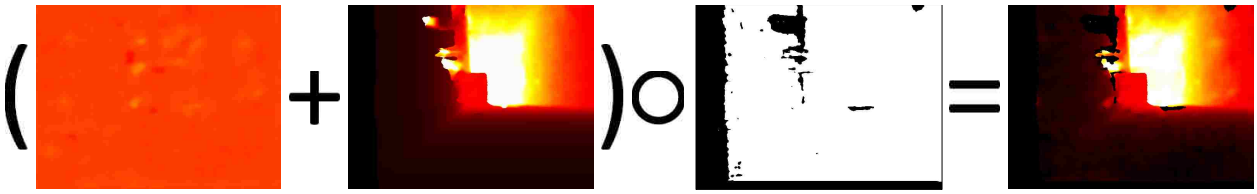


Figure 4.17: Pipeline for generating synthetic noisy depth images. ∘ indicates point-wise multiplication.

Because our network generates smooth dropout predictions rather than discretized binary masks, we are able to reshape these predictions using bilinear interpolation the same as the generated depth residuals. After reshaping, the dropout predictions are converted into binary masks by thresholding about 0. Reshaping and thresholding in this way tends to produce slightly-warped dropout regions with ragged edges, but performs considerably better than performing these operations in reverse order. Thresholding first and then resizing using nearest-neighbor interpolation produces pixelated dropout regions that are far less believable and much more likely to cause problems during depth-completion training. Once the dropout masks are reshaped to match the original RGB-D scene, we clip all values to the range [0, 1], where 0 indicates depth dropout. The post-processing pipeline for predicted dropout is shown in Fig. 4.17.

The entire pipeline for generating a novel noisy image is therefore as follows. First, we use a pre-processed ground-truth RGB-D image and a random z-vector drawn from a normal distribution to generate a depth-residual/predicted-dropout pair. Second, we reduce grid-like artifacts in the generated depth-residual image using a spatial median filter, and then resize using bilinear interpolation. Third, we resize the smooth dropout image using bilinear interpolation, and then binarize it by thresholding about 0. Fourth, we ensure both post-processed images are in the correct ranges, scaling the residual-depth values into the range [-1, 1] and the dropout mask values into the range [0, 1]. Afterwards, we add the generated residual-depths to the original proxy ground-truth depths to apply depth noise, and apply dropout by point-wise multiplying the result with the binarized dropout mask.

4.4 Using our Method for Data Augmentation

Our method may be used for either online or offline data augmentation. Online data augmentation means that training instances are augmented on-the-fly while the depth-completion method is training. Because of this, online augmentation requires access to the codebase for the method-of-choice, and should be avoided if such access is unavailable. In our case, the source code for the sparse-to-dense depth-completion method already contained a framework for adding additional data-augmentation strategies, so the required modifications to support our own method were minimal. Offline augmentation, on the other hand, involves pre-generating additional training pairs to be included with the core dataset. Because of this, offline augmentation adds no time to network training other than the initial cost of pre-generating the augmented data. This data may be re-used between training sessions, and is useful for performing consistent experiments, since the augmented data is identical for each run. We use both modes of augmentation in our experiments to explore the strengths and weaknesses of each.

Chapter 5

Experiments

5.1 Baseline Method

For our baseline experiments, we augment the Sparse-to-Dense depth-completion network proposed by Ma et al., which we call SparseNet for short [21]. SparseNet is the most similar to the type of depth-completion solution we would like to augment apart from the network proposed by Zhang et al., which we will discuss in detail below [31]. SparseNet assumes input RGB-D images with sparse (a few hundred) spatially-random depth samples, and is specifically designed to work with point clouds generated using SLAM methods. It uses an encoder/decoder architecture with a tight bottleneck to project the sparse input depth samples onto dense depth predictions. Because of the aggressive bottleneck, SparseNet produces blurry predictions and is not well-suited to correcting dense noisy RGB-D images. In order to use SparseNet for our experiments, we modify the architecture slightly as shown in Fig. 5.1 to support dense inputs and higher-fidelity outputs.

In the Sparse-to-Dense paper, the authors propose two architectures to operate separately on the NYU-Depth-V2 and KITTI-Cityscapes datasets. This is because the images in the KITTI dataset are so much larger than those in the NYU-Depth-V2 dataset that using the same architecture for both exceeded memory capabilities of the authors' hardware. We use the NYU-Depth-V2 variant of the SparseNet architecture because our RGB-D images are the same size and shape as those contained in the NYU-Depth-V2 dataset. This architecture uses the first four layers of the pre-trained ResNet-50 network for down-sampling, followed by a four-layer decoder that up-projects the image to about half the input size. Afterwards,

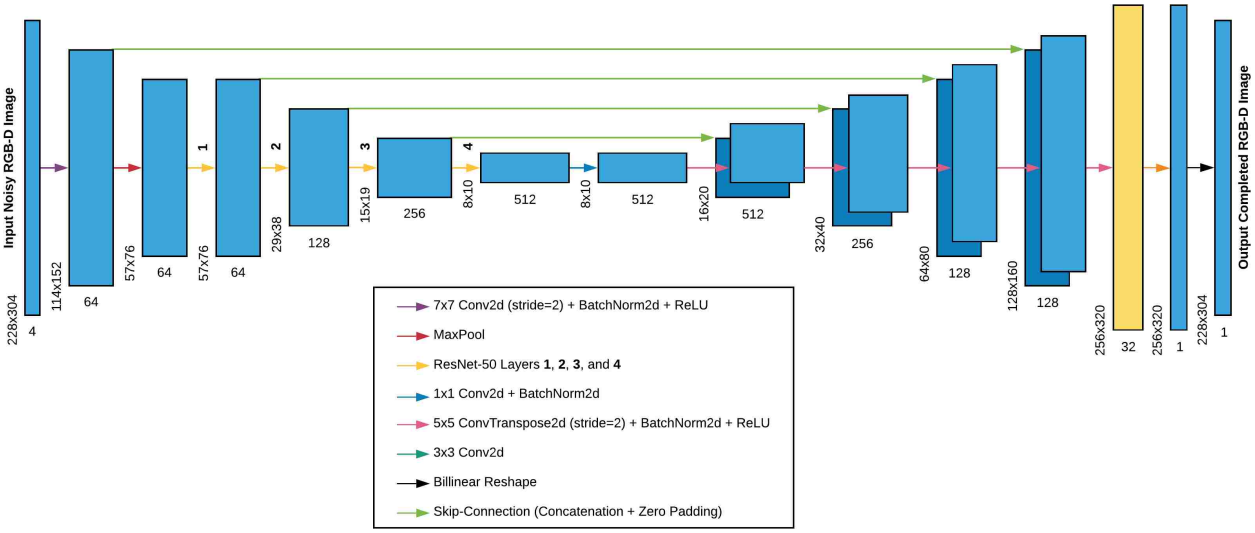


Figure 5.1: Modified SparseNet architecture. We add the layer marked in yellow, as well as the skip-connections between the ResNet-50 layers and the up-sampling layers. We also increase the kernel size of the up-sampling layers from 2×2 to 5×5 to reduce pixelated artifacts in output depth maps.

it uses a 3×3 convolution followed by a bilinear up-sampling layer to project the image to the original shape. The authors found that using the UpProj modules proposed by Laina et al. [14] worked best for the decoder, but we use transpose-convolution layers for simplicity and consistency with our noise-generating architecture.

To overcome the tight bottleneck, we add skip-connections between the four ResNet down-sampling layers and the original four up-sampling layers. This allows high-frequency details to skip the down-sampling layers without enlarging the bottleneck, which is useful for hole-filling and denoising operations. We zero-pad each of the ResNet feature maps to match the somewhat larger up-sampling feature maps before concatenating the two at each skip-connection. The size-mismatch between concatenated layers results in a border-like artifact that usually disappears by the time training concludes. Besides the skip-connections, we add an additional 2-strided 5×5 transpose-convolution layer to the decoder to improve image fidelity. This reduces blurring as a result of up-sampling with the bilinear layer, which is now used to down-sample the image to the correct shape. We likewise increase the kernel

sizes of each of the original decoder layers from 2×2 to 5×5 to reduce pixelated artifacts in output depths.

5.1.1 Discussion of Alternate Baseline

It should be noted that the Deep Depth Completion method proposed by Zhang et al. is both state-of-the-art and better fits the assumptions of our depth-augmentation strategy [31]. However, as previously discussed, this method is not end-to-end, and uses a deep network only to predict surface normals from color data, not complete damaged depths. The authors observe that including depth information in the surface-normal predictions actually hurts network performance, likely because the network learns to rely too heavily on input depth information and is unable to learn to predict surface normals for regions where depth is missing. Beyond surface-normal prediction, the actual depth-completion step is posed as a global optimization problem, which has a known algorithmic solution and does not benefit from additional training data. These factors make it seem unlikely that our method could effectively augment this particular approach, so we leave further investigation to future work.

5.2 Baseline Experiments

We perform separate baseline experiments on both the D415 and D435 splits of our custom dataset. The purpose of these experiments is to compare the effect of adding more data in the form of RGB-D scenes versus additional depth frames, as well as to test how well SparseNet is able to perform using our proxy ground-truth depth images. These experiments further demonstrate how much data is needed to train an effective depth-correction algorithm, and the rate at which adding additional real data decreases validation error. They provide the foundation for our augmentation experiments, since we expect augmented datasets to perform better in every case. For each of these experiments, we train the modified SparseNet model for 100 epochs on a randomly-selected subset of the chosen training split, optimizing

(a) 1 Depth Frame (b) 16 Depth Frames (c) 200 Depth Frames

Figure 5.2: Recalculated proxy ground-truth depth images. We recalculate proxy ground-truth depths for each training subset to ensure that each model only trains on the available data without receiving cues from the larger dataset. On subsets with only one depth frame per scene, we skip the median-stack filtering step and simply apply cross-bilateral filtering to produce proxy ground-truth depths.

with respect to the pixel-wise mean squared error and reporting the average validation error between two identical runs of the experiment.

To prevent accidental inclusion of extra information, we recompute our proxy ground-truth RGB-D images for each randomly-selected training subset as shown in Fig. 5.2. We do this to ensure that each depth-correction model is trained with respect to only the available training data, not the entire dataset at large. We use our previously-described depth-denoising pipeline in each case, with a slight modification for datasets where we only sample one depth frame per scene. We typically apply a median-stack filter to multiple depth frames to denoise our proxy ground-truth RGB-D images, but this is not possible in the case of scenes with only one frame. In these cases, we do not apply any special kind of denoising but merely use cross-bilateral filtering to fill in the majority of the holes. This produces noticeably sub-par proxy ground-truth data, but the reduction in quality is to be expected with only a single frame per scene.

We do not recompute proxy ground-truth images for our validation sets, but build these images using all available depth frames. We further validate each depth-completion model against the entire validation set of the corresponding split, using only one frame per scene for sake of efficiency. This provides consistency between experiments and allows

	Training Iterations				
	10000	20000	40000	80000	160000
More Depth Frames	1.230	**1.210**	1.133	0.829	0.706
More Unique Scenes	1.230	1.264	**1.029**	**0.813**	**0.670**
More Epochs	1.230	1.244	1.231	1.235	1.236

D415 Baseline Experiments

	Training Iterations				
	10000	20000	40000	80000	160000
More Depth Frames	1.104	1.060	0.924	**0.510**	**0.447**
More Unique Scenes	1.104	**1.059**	**0.804**	0.533	0.456
More Epochs	1.104	1.090	1.092	1.100	1.082

D435 Baseline Experiments

Table 5.1: Baseline validation mean squared error on the D415 and D435 datasets. Best validation errors are shown in bold. Each model is trained for 100 epochs except where indicated otherwise, and each is directly comparable to similar models trained for the same total number of training iterations. The total number of training iterations for each experiment is considered to be the number of unique training pairs multiplied by the number of training epochs.

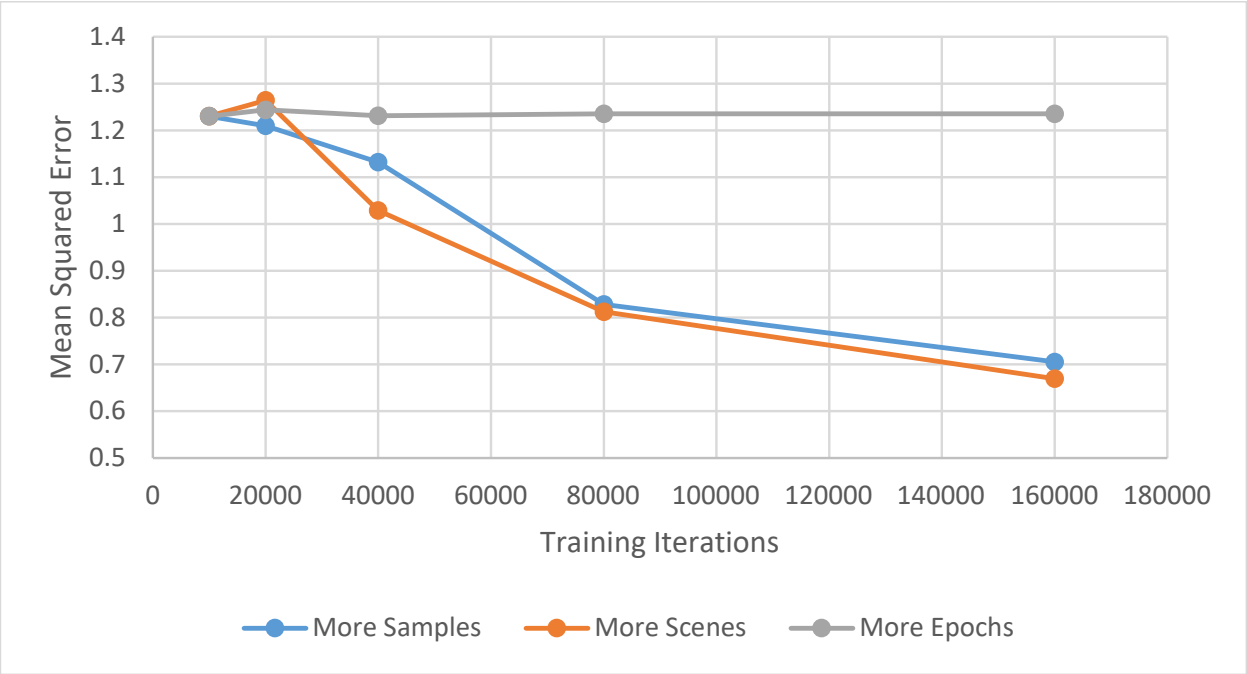
us to judge how well each model learns to generalize and denoise RGB-D data. It further provides us with an objective measure to compare how well depth-completion models perform when trained on different kinds of data. This includes different sizes of datasets, different ratios of depth frames to unique scenes, and different qualities of proxy ground-truth data. Interestingly, our experiments demonstrate that even depth-completion models trained on sub-optimal proxy ground-truth data tend to generalize well to the respective validation set if given enough training data.

For our initial experiments, we compare the isolated effects of increasing the number of training scenes versus the number of depth frames per scene. We begin by training a model on a randomly-selected subset of 100 scenes with one depth frame per scene. Next, we double the number scenes for each experiment, training models on 200, 400, 800, and 1600 scenes, each with only one depth frame per scene. We mirror this by training models on the original 100 scenes with 2, 4, 8, and 16 depth frames per scene. Note that we balance the

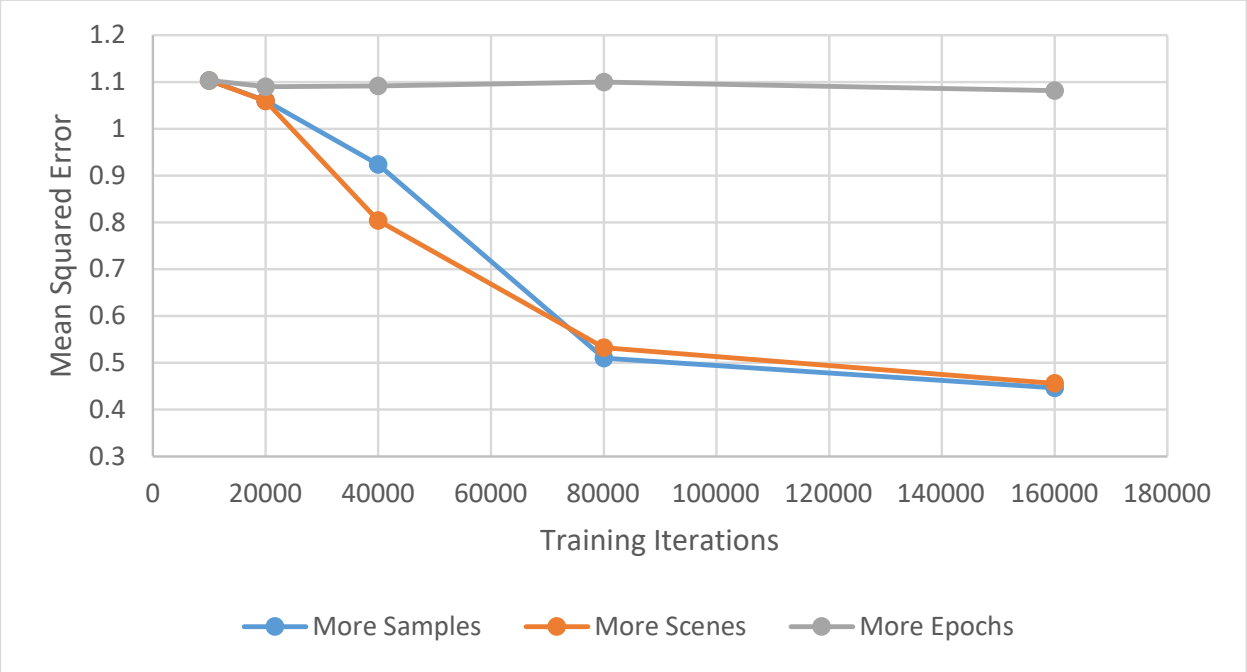
number of scenes and depth frames in each training subset so that each model is directly comparable to a model trained predominantly on the opposite kind of data. To control for the effect of simply training the models for longer, we train models on the original subset of 100 scenes with one depth frame per scene for 200, 400, 800, and 1600 epochs. We report the average pixel-wise mean squared error of these experiments for both the D415 and D435 splits in Table. 5.1 and compare the validation curves for each split in Fig. 5.3.

As expected, the error curve for increasing the number of epochs is nearly flat, since simply exposing the network to the same data more often tends to cause overfit. More interestingly, the error curves for the number of scenes and the number of depth frames are quite similar, indicating that the two improve the model at almost the same rate. Adding more frames seems to be better under most circumstances, though the error rates for each are similar enough that this may be within standard error. This indicates that both forms of data have a similar impact on model generalization, and supports our claim that additional synthetic noisy depth frames will improve depth-completion methods over baseline results. Further, the impact of additional data seems to be largest when training subsets are small, indicating that data augmentation will likely be the most useful for small datasets. This is as expected, since our goal is to enable the development of good depth-completion methods without the need for extensive ground-truth data collection.

Following this, we examine the combined effect of increasing both the number of scenes and depth-frames per scene, the results of which are shown in Table 5.2. As expected, error reduces monotonically as one increases the number of scenes going from left to right across each row. This follows the previously-observed behavior of our initial experiments, and matches the pattern one would expect to see by adding additional unique ground-truth RGB-D images to a training set. More significantly, increasing the number of depth frames per scene similarly reduces the resulting error as shown by the monotonically decreasing error as one goes down each column. These results demonstrate that the link between adding more unique scenes and more depth frames per scene holds for many different sizes of datasets. In



(a) D415



(b) D435

Figure 5.3: Baseline validation curves showing the effect of more scenes vs. more samples on the D415 and D435 datasets. More scenes (more ground-truth) seems to be generally more effective, but more depth frames per scene seems to have a similar effect on training. This suggests that augmentation by adding synthetically-damaged depth frames should improve over these results.

Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.230	1.264	1.029	0.813	**0.670**
2	1.210	0.992	0.800	**0.666**	0.589
4	1.133	0.888	**0.686**	0.606	0.553
8	0.829	**0.696**	0.601	0.542	0.495
16	**0.706**	0.610	0.560	0.502	0.459

D415 Baseline Experiments

Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.104	1.059	0.804	0.533	**0.456**
2	1.060	0.754	0.488	**0.432**	0.375
4	0.924	0.582	**0.445**	0.381	0.322
8	0.510	**0.461**	0.389	0.334	0.289
16	**0.447**	0.405	0.323	0.293	0.254

D435 Baseline Experiments

Table 5.2: Results of baseline experiments. Pixel-wise mean squared error is shown for each experiment. Note that error reduces monotonically as the number of scenes increases across each row, and also reduces monotonically as the number of frames increase down each column. We bold the center antidiagonal down each matrix (from upper right to lower left) demonstrating the similar validation errors between experiments with the same total number of training instances.

every case, adding more depth frames to each scene reduces pixel-wise error at about the same rate as introducing additional unique scenes. Put differently, adding more frames to each scene is about equivalent to adding more ground-truth data to the training set.

It further appears that the ratio of frames per scene to the number of unique scenes in the training set does not effect pixel-wise loss. The errors along the antidiagonals of each matrix of Table 5.2 use the same number of training pairs and produce validation errors within a tight error margin of one another, with few exceptions. This seems to indicate that it does not matter whether one adds more unique ground-truth images or more noisy depth frames to the training set, since both have a nearly identical impact on training. This bears implications for the development of future RGB-D datasets, since it is generally easier to collect multiple frames for a few scenes than many scenes which must be carefully hand-corrected. Assuming that other depth-completion models behave similarly to SparseNet, it might be possible to gather larger ground-truth datasets with less effort by following our own pattern of collecting multiple frames per scene.

Because we measure validation error quantitatively using pixel-wise mean squared error, it is possible that training on more depth frames leads to qualitatively different kinds of artifacts in corrected RGB-D images than training on more diverse scenes. Examples could include increased blurring on object boundaries and poor generalization to novel geometries when trained on fewer unique scenes than depth frames. Fig. 5.8 demonstrates that these artifacts are not readily apparent in corrected RGB-D images, so further inspection is required to discern any qualitative differences. A potential solution to determining qualitative error in reconstructed depth maps would be to use a pre-trained convolutional network to form a perceptual loss between output and target depth images. Since we use SparseNet for our depth-completion model, however, we follow the pattern set by Ma et al. and focus our efforts on improving quantitative baseline error [21]. Further study using a qualitative loss is beyond the scope of this thesis and is left to future work.

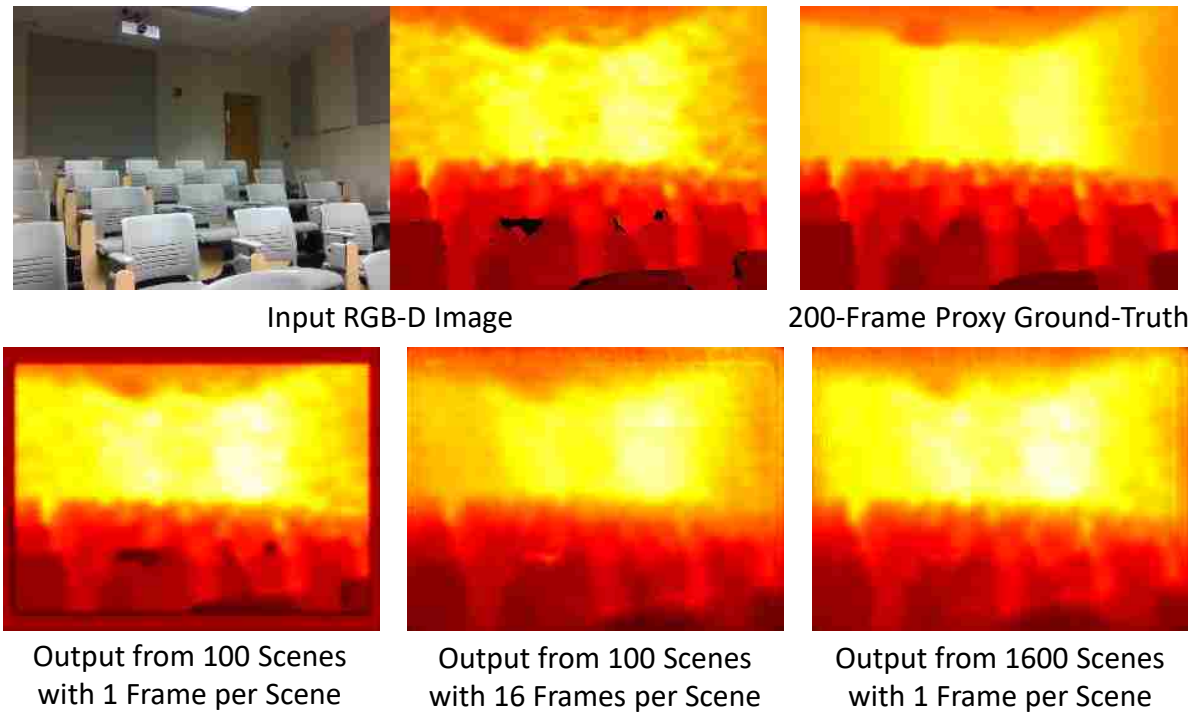


Figure 5.4: Baseline completed RGB-D images. Top row: Input RGB-D image and target output. Bottom row: SparseNet outputs trained on three different training sets.

5.3 Augmentation Experiments

To test our augmentation method, we repeat our baseline experiments using synthetically doubled training sets. We use offline augmentation for each experiment to control the amount of synthetic data each training subset receives. We train a new noise-generating VAE for each experiment using only the randomly selected training subset, holding out a random 20% for validation. We avoid validation on the actual held-out validation sets to prevent cross-contamination of the training and validation sets through data augmentation. Further, we use the same proxy ground-truth RGB-D images calculated for each training subset to train our noise generator as well as our depth completer. This ensures that both models train on exactly the same data, and that the noise generator does not have any outside information the depth corrector lacks. To further control for the effect of longer training due to larger datasets, we halve the number of training epochs for each experiment from 100 to 50. As before, we run each experiment twice and report the average error between the two runs.

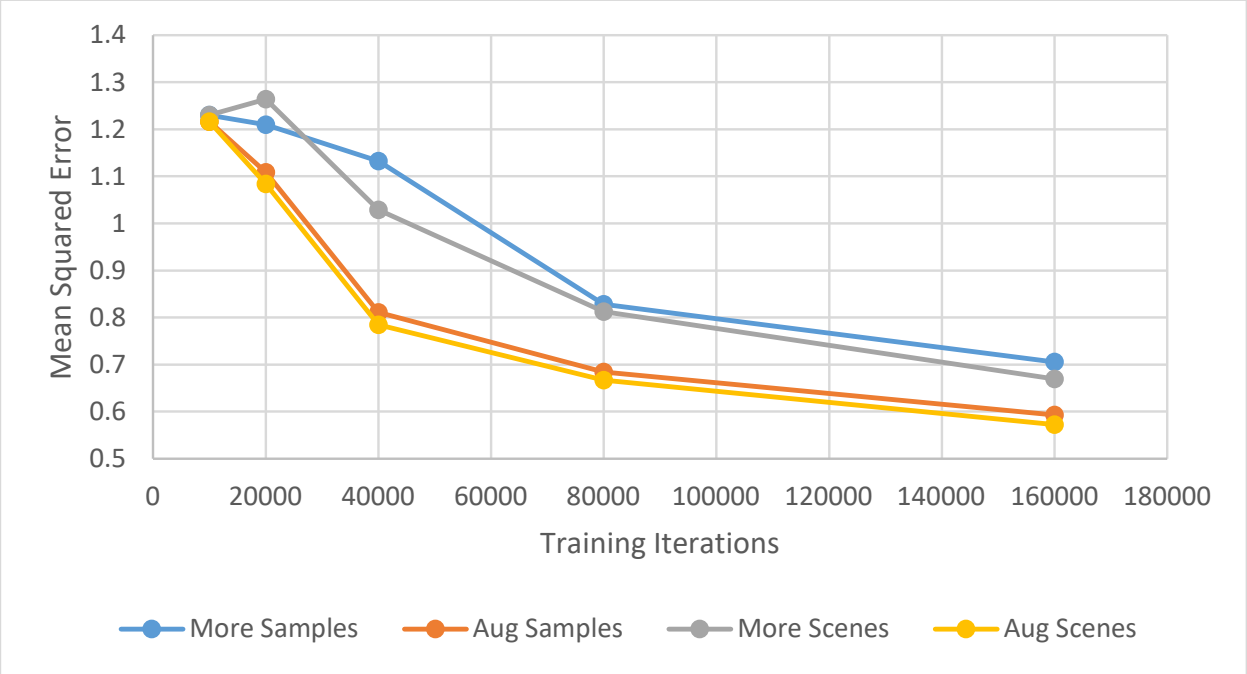
	Training Iterations				
	10000	20000	40000	80000	160000
More Depth Frames	1.230	1.210	1.133	0.829	0.706
More Depth Frames + Aug	**1.217**	1.109	0.811	0.685	0.593
More Unique Scenes	1.230	1.264	1.029	0.813	0.670
More Unique Scenes + Aug	**1.217**	**1.084**	**0.785**	**0.667**	**0.573**

D415 Augmentation Experiments

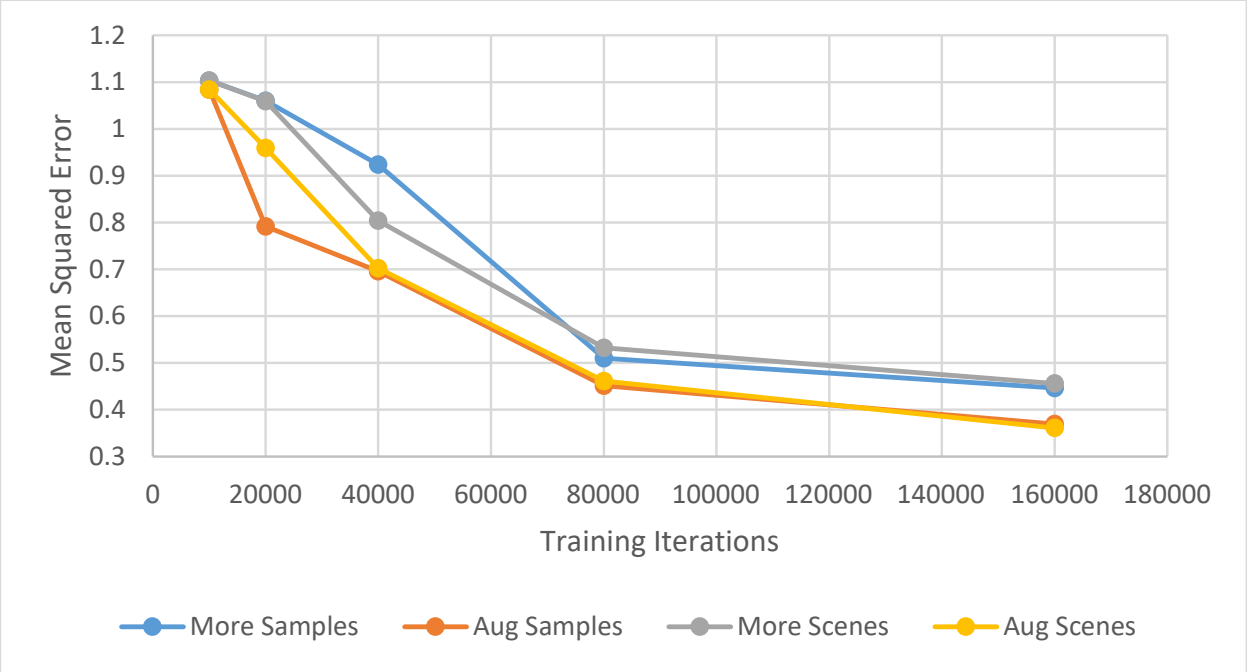
	Training Iterations				
	10000	20000	40000	80000	160000
More Depth Frames	1.104	1.060	0.924	0.510	0.447
More Depth Frames + Aug	**1.085**	**0.792**	**0.696**	**0.452**	0.370
More Unique Scenes	1.104	1.059	0.804	0.533	0.456
More Unique Scenes + Aug	**1.085**	0.959	0.703	0.461	**0.361**

D435 Augmentation Experiments

Table 5.3: Augmented validation mean squared error for initial experiments compared to baseline experiments. We mirror our initial baseline experiments using synthetically doubled training subsets and half as many training epochs. As before, best validation errors are marked in bold.



(a) D415



(b) D435

Figure 5.5: Augmented versus baseline validation curves. Note that augmentation appears to improve error in nearly every case, but helps the most on medium-sized datasets. Synthetically doubling the amount of training data on the smallest datasets does not visibly improve error, while doing the same on the largest datsets only reduces error slightly below the baseline.

We first mimic the experiments described by Fig. 5.3 in the previous section. We test a single augmented baseline using 100 scenes with one frame per scene, and then check the effect of additional scenes and frames as previously described. The results of this experiment are reported in Table 5.3. As demonstrated by both Table 5.3 and the validation curves given by Fig. 5.6, the augmented experiments outperform the baselines in every instance except on the set of 100 scenes with one frame per scene. In this case, it is noteworthy that the augmented method does not perform worse than the baseline, just not noticeably better, meaning that augmentation did not hurt training even on such a small dataset. In later experiments, we find that we are able to improve baseline results even on the smallest baseline training sets by simply generating much larger amounts of synthetic data. This suggests that, as expected, synthetic data is somewhat less useful for network training than real data, though the contribution is still significant.

The validation curves in Fig. 5.6 demonstrate that the effect of synthetically doubling training data tends to be more exaggerated on smaller datasets, with the exception of the smallest training subset. As real data increases, however, improvements due to additional synthetic data tend to taper off. To test how this applies to different sizes of training sets with different ratios of scenes to depth frames, we perform augmented versions of the experiments described by Table 5.2. In each experiment, we synthetically double the number of depth frames per scene in each training subset and reduce the number of training epochs from 100 to 50 as described previously. We report our results in Table 5.4. As previously observed, the percent improvement for augmented experiments reduces as the total number of training pairs increases, as shown by Table 5.5. Even so, our experiments demonstrate an average improvement of approximately 13.5% on the D415 dataset, and approximately 16.2% on the D435 dataset. It is possible that better results could be obtained within limits by adding even more synthetic data to the augmented training sets.

Observing Table 5.5 we note that our augmentation method yields limited improvements on both extremely small and extremely large datasets. It seems that the sweet spot

Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.217	1.084	0.785	0.667	**0.573**
2	1.109	0.820	0.679	**0.553**	0.515
4	0.811	0.732	**0.596**	0.542	0.490
8	0.685	**0.590**	0.542	0.493	0.458
16	**0.593**	0.538	0.485	0.462	0.429

D415 Augmented Experiments

Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.085	0.959	0.703	0.461	**0.361**
2	0.792	0.559	0.426	**0.383**	0.320
4	0.696	0.442	**0.397**	0.317	0.269
8	0.452	**0.377**	0.314	0.281	0.243
16	**0.370**	0.324	0.273	0.243	0.223

D435 Augmented Experiments

Table 5.4: Results of data augmentation experiments. We report average mean-squared error between two runs of each experiment. We synthetically double the number of frames per scene for each training subset, and halve the number of training epochs such that each experiment is comparable to the baselines shown by Table 5.2. In every case, the augmented methods produce lower error than their baseline counterparts as shown by Table 5.5. As before, we bold the center antidiagonal of each matrix to demonstrate similar errors between experiments with the same total number of training instances.

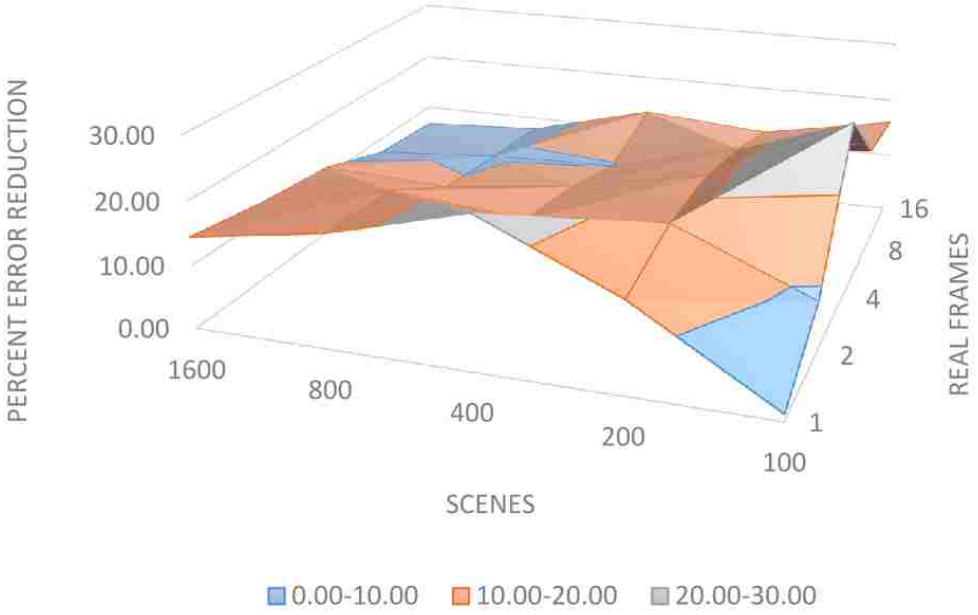
Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.10	**14.24**	**23.71**	**17.91**	**14.49**
2	**8.35**	**17.30**	**15.13**	**16.98**	12.56
4	**28.39**	**17.57**	**13.12**	10.49	11.39
8	**17.38**	**15.23**	9.82	9.13	7.47
16	**15.95**	11.81	13.39	7.88	6.64

D415 Augmentation Improvements

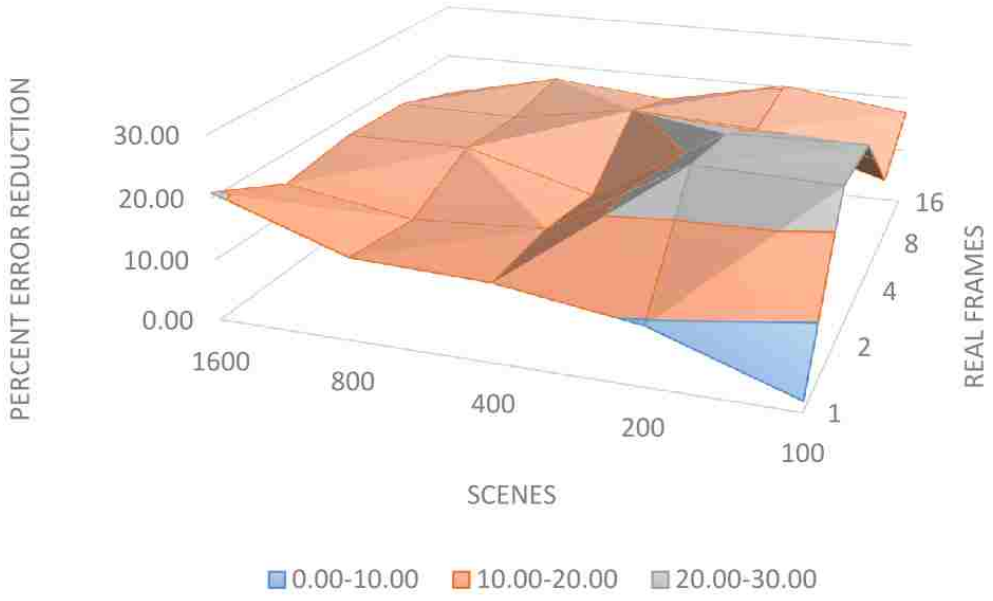
Frames Per Scene	Total Scenes				
	100	200	400	800	1600
1	1.72	**9.44**	**12.62**	**13.43**	**20.83**
2	**25.28**	**25.93**	**12.81**	**11.46**	14.80
4	**24.63**	**23.99**	**10.79**	16.69	16.49
8	**11.47**	**18.22**	19.41	15.74	16.09
16	**17.25**	20.12	15.35	17.24	12.23

D435 Augmentation Improvements

Table 5.5: Percent improvement on augmented training sets. Note that improvements are generally the greatest on medium-sized datasets with between 200 and 1600 total training pairs, though this range seems slightly wider on the D435 dataset. We bold the percent improvement for all experiments with between 200 and 1600 training pairs in both matrixes for easy readability.



(a) D415



(b) D435

Figure 5.6: 3D visualization of augmented error reduction. As demonstrated by Table 5.5, our augmentation method is the most effective on medium-sized datasets. In both the surfaces shown above, percent improvement is the lowest on the largest and smallest datasets, but reaches its peak on around 400 total training pairs. Blue regions indicate the smallest error reductions, while gray areas indicate the largest.

for improvement is somewhere between 200 and 1600 training pairs, with results tapering off before and after that. This is likely because our augmenter is unable to learn to produce realistic synthetic data from small datasets, and the effects of augmentation diminish as expected when more real data is available. Medium-sized datasets contain sufficient real data for the augmenter to learn to produce realistic outputs while still being small enough that augmentation contributes to training. Nonetheless, Table 5.5 demonstrates that augmentation improves depth-completion error over baseline results in every instance. This means that our VAE is able to learn useful characteristics of the latent noise distribution even on extremely small training sets. In the next section, we demonstrate the ability of our method to make large improvements even on small datasets by generating disproportionate amounts of synthetic data.

5.3.1 Effects of More Synthetic Data

We check the limits of our augmentation method by synthetically expanding randomly selected 100-scene training subsets from each camera split with only one scene per frame. We train a unique generator on each subset, holding out a random 20% of the data for validation, and use these pre-trained models to augment every experiment for the respective camera split. For control, we first train the depth-completion models on the real training subsets and then train the same models from scratch on augmented datasets with 2, 4, 8, 16, and 32 times the original number of frames. In each experiment, only the original 100 single-frame scenes are real, while the rest of the frames for each scene are synthetically generated. We check the effect of synthetic data versus additional real data by conducting the same experiments using 2, 4, 8, 16, and 32 times the original number of real frames with no augmented data. As additional control, we adjust the number of training epochs for each experiment so that each model trains for the same number of iterations.

We run each experiment twice and present the average mean squared error of our results in Table 5.6. Interestingly, while doubling the number of frames in such a small

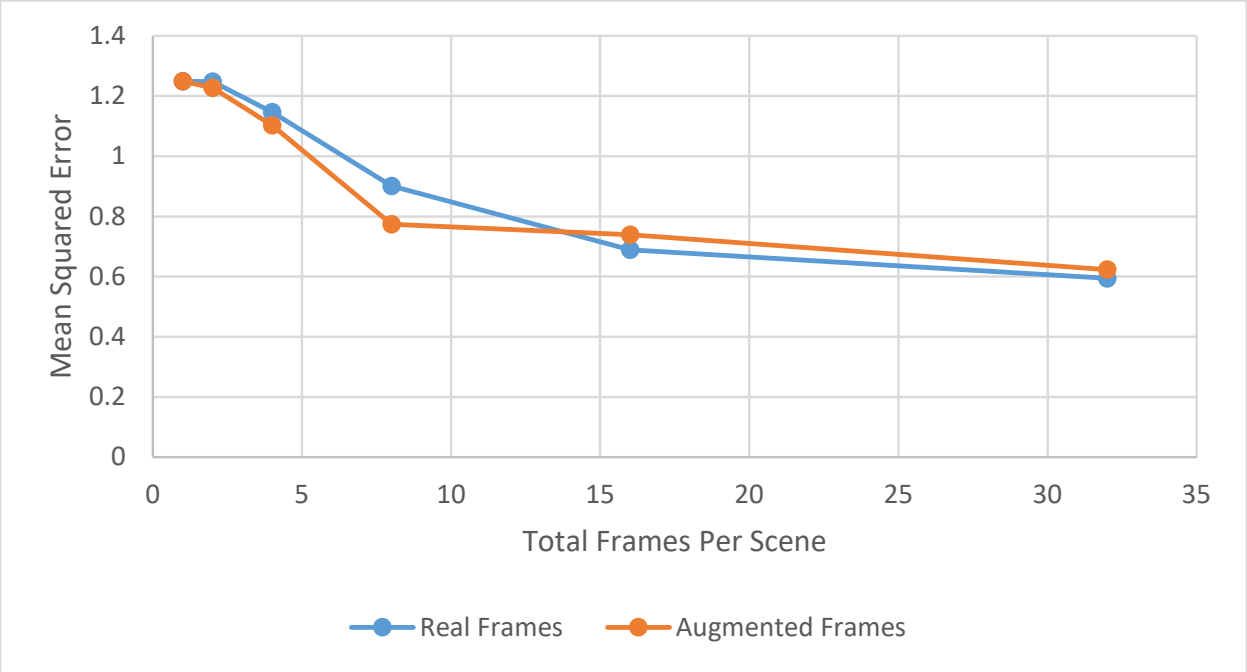
Training Epochs	Frames Per Scene	Real Frames	Synthetic Frames
1600	1	1.249	1.249
800	2	1.248	**1.227**
400	4	1.147	**1.103**
200	8	0.901	**0.774**
100	16	**0.690**	0.740
50	32	**0.595**	0.623

Increased Synthetic Data on D415 Dataset

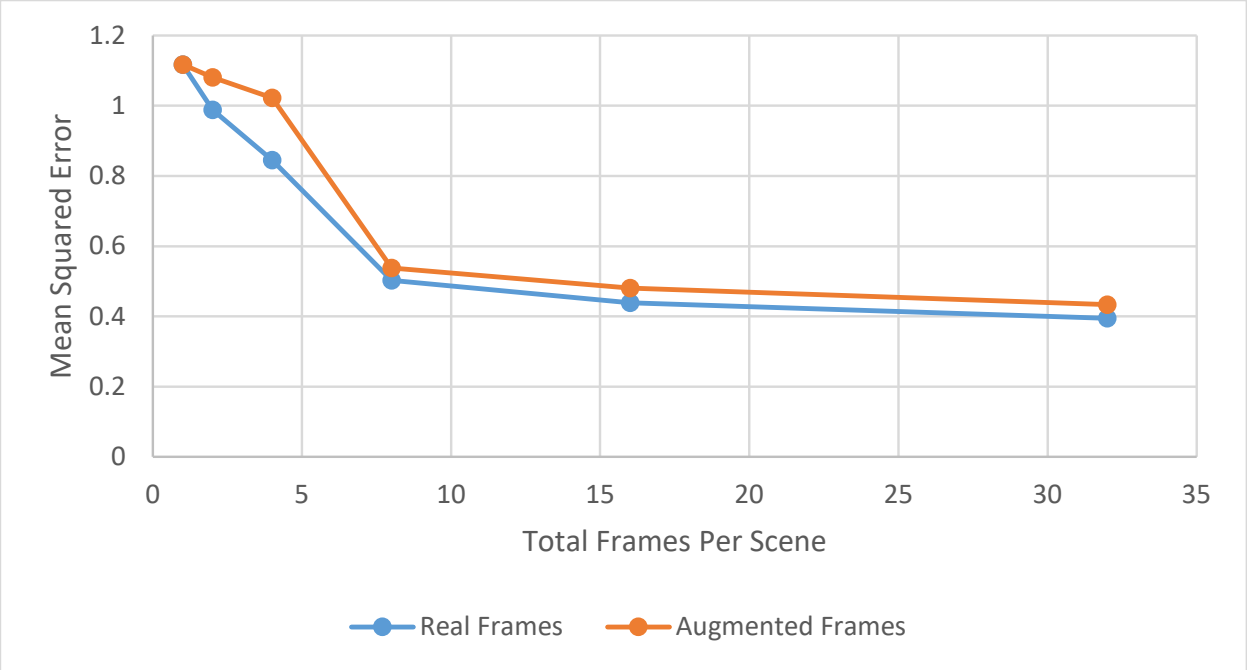
Training Epochs	Frames Per Scene	Real Frames	Synthetic Frames
1600	1	1.118	1.118
800	2	**0.988**	1.081
400	4	**0.845**	1.022
200	8	**0.503**	0.538
100	16	**0.439**	0.480
50	32	**0.395**	0.434

Increased Synthetic Data on D435 Dataset

Table 5.6: More synthetic frames compared with more real frames. Each training set consists of the same 100 randomly-selected scenes, and we report mean squared error on the appropriate validation set averaged between two runs of each experiment. Note that we balance the number of epochs for each experiment so that each model trains for the same amount of time. Best errors are marked in bold, and skew towards training on more real data. While real data performs better than synthetic data in most cases, both errors reduce at a similar rate.



(a) D415



(b) D435

Figure 5.7: Synthetic versus real validation curves. Each training set consists of the same 100 randomly-selected scenes, and we report mean squared error on the appropriate validation set averaged between two runs of each experiment. We use the same generator for each experiment, and train it only on the initial 100 scenes using one frame per scene. Note that increasing the number of synthetic frames per scene has a similar effect on training to increasing the number of real frames per scene.

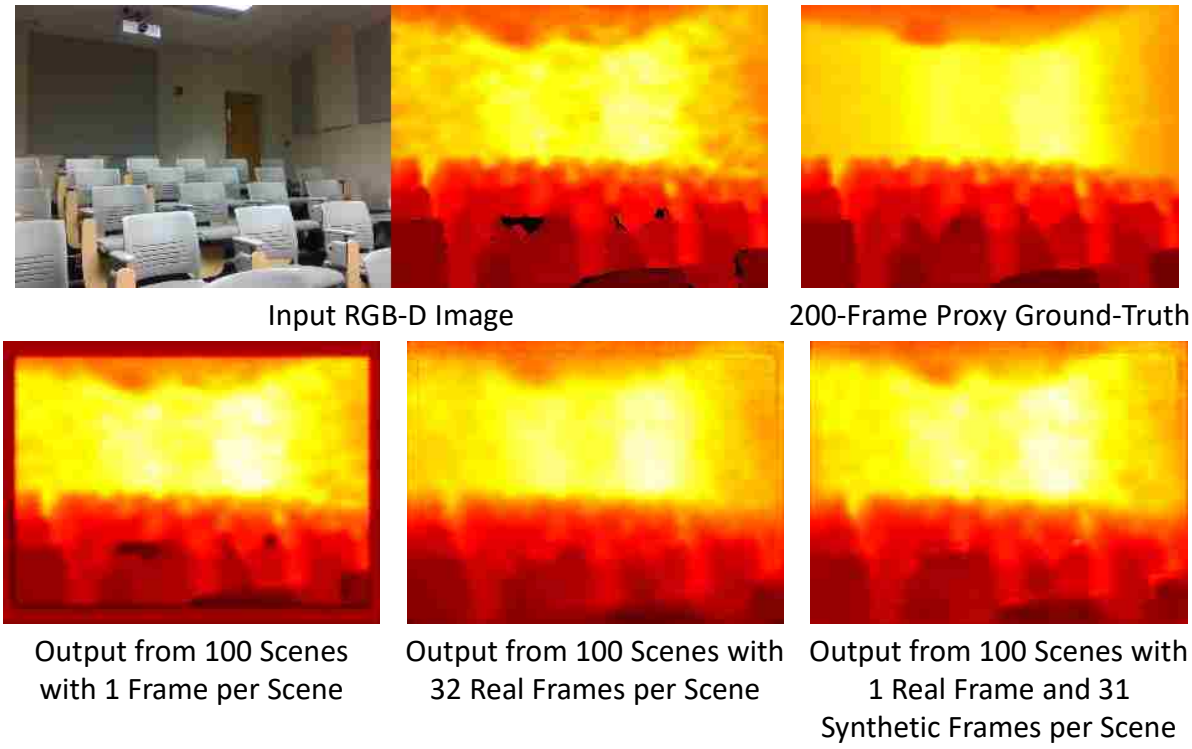


Figure 5.8: Completed RGB-D results using more synthetic training data. Top row: Input RGB-D image and target output. Bottom row: SparseNet outputs trained on three different training sets, including a dataset consisting of disproportionately-synthetic data.

training set is ineffective, we are able to achieve nontrivial improvements by adding much more synthetic data. By synthetically boosting the size of a 100-scene training set from one frame per scene to 32 frames per scene, we are able to achieve an average 50.1% improvement in pixel-wise mean squared error on the D415 validation set, and a 61.2% improvement on the D435 validation set. This is just slightly behind the average 52.4% and 64.7% improvements we see on the D415 and D435 datasets respectively using 32 real frames per scene. By using our augmentation method, we are able to train a depth-completion method on a limited amount of real data to be competitive with a real dataset 32 times larger, and possibly even better with more synthetic data. Fig. 5.7 illustrates how closely adding synthetic frames mirrors adding more real frames on pixel-wise error. The rate of improvement for both real and synthetic data tapers off at a similar rate, with methods trained on synthetic data tending to lag slightly behind those trained on real data as would be expected.

As we have demonstrated, it is possible to use our augmentation method to significantly reduce the amount of data required to train a reasonable supervised depth-completion model. It is further possible to do so using sub-optimal proxy ground-truth data obtained using only one or more sampled depth frames per scene. This is important because a depth-completion model trained on our Intel RealSense D415 or D435 datasets would likely generalize poorly to RGB-D images captured using different depth-sensing technologies. Users may customize our noise-generator to their specific camera noise distribution by collecting their own data or by using an existing dataset. Since our noise-generator is able to augment depth-completion models trained on small datasets to be competitive with those trained on much larger datasets, users' custom datasets may be as large or as small as necessary. As previously discussed, Table 5.5 demonstrates that the sweet spot for our augmentation method resides in medium-sized datasets and improvements diminish with extremely small or large amounts of training data. Despite this, we can still achieve meaningful improvements on small datasets by generating disproportionate amounts of synthetic data.

5.3.2 Efficiency Compared to Real Data

A valid concern is whether running the depth-denoising pipeline and training our noise-generating model offsets the cost of gathering more real training data. In our experience, we were generally able to capture a 200-frame scene in about one minute. Capturing multiple frames per scene generally takes far less time than capturing more diverse scenes, and also yields better proxy ground-truth RGB-D images. Following our pattern for depth-completion training, proxy ground-truth RGB-D images are required whether one wishes to augment using synthetic data or not. Our depth-denoising pipeline only needs to be run once to pre-generate proxy ground-truths, and may be set to run during off-hours with limited human supervision. Depending on the size of the dataset, running the pipeline and training the noise-generator may take anywhere from a few minutes to several hours. Once trained, the

generator is able to produce as much synthetic data as needed, making synthetic augmentation generally more efficient than gathering more real data.

Chapter 6

Conclusion

In the introduction to this work, we state that our aim is to demonstrate that it is possible to achieve better-than-baseline error on supervised depth-completion methods by using synthetic depth-map augmentation. In support of this claim, we have gathered large RGB-D datasets for the Intel RealSense D415 and D435 cameras. These datasets are designed to support both depth-completion training as well as noise-generation training by pairing multiple examples of noisy depth maps with automatically denoised and hole-filled RGB-D images. We demonstrate that these automatically completed RGB-D images may be used as reasonable proxies for hand-tuned ground-truth data in our baseline experiments by showing reasonable generalization to held-out datasets. We benchmark baseline depth-completion results on our datasets using a modified version of the Sparse-to-Dense depth-completion network proposed by Ma et al. [21] on different-sized subsets of our training data. Through our baseline tests, we demonstrate that adding additional noisy depth frames per scene has a similar effect on training as adding more unique scenes. This supports our claim that adding synthetic noisy depth frames to the training set will reduce error below baseline results.

We test whether this theory holds by training our own noise-generating model to produce synthetic depth frames from each training subset. We use this model to synthetically double the number of depth frames for each training subset, and retrain the depth-completion method on each augmented training set for half the original number of epochs. We demonstrate an average 13.5% error reduction on the D415 dataset, and an average 16.2% error reduction on the D435 dataset using synthetically doubled datasets. Our experiments show that our

method tends to work best on training sets with between 200 and 1600 training pairs, while providing less contribution on training sets larger or smaller than those bounds. Our method reduces depth-completion error by less than 3% when doubling datasets with only 100 training pairs, but yields as much as a 61.2% improvement when synthetically increasing those same datasets to 32 times their original size. These results demonstrate that depth-completion methods trained on small to medium-sized training sets may compete with methods trained on much larger amounts of real data by using our augmentation method.

6.1 Limitations and Future Work

Our augmentation method is not meant to be construed as a replacement for real training data. While our method enables small datasets to compete with larger datasets by synthetically boosting the amount of training data, methods trained on larger real datasets will of course perform better on average. Fig. 5.7 demonstrates similar mean-squared-error curves for training on increased amounts of real and synthetic data, but it should be noted that the curve for real training data tends to converge to a lower error. Our noise generator is able to produce high-quality synthetic data that is not always photorealistic, especially when it is trained on smaller datasets. Because of this, it might be possible to add too much synthetic data to a training set, causing the depth-completion model to overfit to synthetic noise and generalize poorly to real data. While we did not observe this behavior in our experiments, this is likely because we stopped adding synthetic data when improvements began to taper off.

While our initial results using our augmentation generator are promising, our present method leaves room for improvement. As discussed in the opening chapters, GAN models hold the current state-of-the-art in generative machine learning, whereas we have used a VAE model for the sake of simplicity and stable training. In the future, a cGAN structure would likely prove more effective and could possibly learn to generate more convincing synthetic noisy depth images. GAN models tend to be better at modelling high-frequency details than

VAE models and could potentially reduce artifacts caused by the down-sampling bottleneck of the VAE network. Additionally, GAN models learn adaptive loss functions through the discriminator that could potentially pick up on important attributes of the noise distribution even from small datasets. This could help improve results augmenting small datasets without the need for generating disproportionate amounts of synthetic data. Mode collapse remains an issue but may potentially be alleviated by using a BicycleGAN or a similar model that improves generated diversity by enforcing connections between latent encodings and generated images [33].

It would further be useful to inspect the qualitative differences between depth-completion methods trained on datasets containing disproportionate numbers of depth frames compared to scenes, and vice-versa. Our experiments demonstrate that both larger numbers of scenes and larger numbers of depth-frames per scene have a similar effect on pixel-wise error for the validation set. However, pixel-wise error may not be the best metric for measuring perceptual differences between images. It is possible, even likely, that a given depth-completion method will produce different artifacts when trained on a dataset containing more scenes than images. Determining the exact differences could be achieved using a perceptual loss function comparing layer activations on a pre-trained network between target and generated images. A perceptual loss might also be useful for training the noise-generating model as a means of discouraging unrealistic generated artifacts not penalized by a pixel-wise loss. However, the transfer of perceptual loss from networks trained on RGB images to RGB-D images remains an open question.

Finally, it would be useful to compare our method to other forms of depth-completion augmentation. A neural network is a potentially heavy solution to what could possibly be solved using a simpler algorithm. Given sufficient familiarity with a particular RGB-D camera and its noise distribution, it might be possible to write an effective noise-generating algorithm to augment that specific training set. The benefit of our method is that it is adaptable to a wide range of different camera noise distributions, and is relatively easy to set up and

train. Non-machine-learning solutions would need to be custom-crafted for each new type of camera and depth-sensing technology, and would require a certain level of expertise to arrive at a useful construction. Our augmentation method is able to train on the same data used to train a supervised depth-completion method and replaces human experts by learning key characteristics of the noise distribution automatically. However, it would be useful to compare our method to other approaches to gauge the utility of each.

The contributions of this work hold promise for the rapid development of depth-correcting algorithms for novel depth cameras and other depth-sensing technologies. Our noise generator is able to adapt easily to new noise distributions and produce additional training data where it is impractical or inconvenient to gather more by hand. By using our automated depth-denoising pipeline, it is further possible to produce passable proxy ground-truth RGB-D images without the need for extensive hand-tuning. We also demonstrate through our experiments that it is possible to improve beyond baseline depth-completion results without the need for collecting additional training data, even on small datasets. By using our augmentation method and depth-denoising pipeline, it may be possible to greatly relieve the burden of data collection for supervised depth-completion methods.

References

[1] Antreas Antoniou, Amos J. Storkey, and Harrison A Edwards. Data augmentation generative adversarial networks. *Computing Research Repository (CoRR)*, abs/1711.04340, 2018.

[2] Angel Xuan Chang, Angela Dai, Thomas A. Funkhouser, Maciej Halber, Matthias Nießner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from RGB-D data in indoor environments. *2017 International Conference on 3D Vision (3DV)*, pages 667–676, 2017.

[3] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2650–2658, 2015.

[4] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Advances in Neural Information Processing Systems 27*, pages 2366–2374. 2014.

[5] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using GAN for improved liver lesion classification. *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*, pages 289–293, 2018.

[6] Ravi Garg, Vijay Kumar B.G., Gustavo Carneiro, and Ian Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision (ECCV) 2016*, pages 740–756, 2016.

[7] Clement Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6602–6611, 2017.

[8] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *2014 Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[9] Simon Hawe, Martin Kleinsteuber, and Klaus Diepold. Dense disparity maps from sparse disparity measurements. In *2011 International Conference on Computer Vision (ICCV)*, pages 2126–2133, 2011.

[10] Intel. Overview of the Intel RealSense depth camera. URL `https://software.intel.com/en-us/realsense/d400`. Accessed 2019.

[11] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[12] Diederik P. Kingma and Max Welling. Auto-encoding variational bayes. In *2014 International Conference on Learning Representations (ICLR)*, 2014.

[13] Yevhen Kuznietsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2215–2223, 2017.

[14] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2016 Fourth International Conference on 3D Vision (3DV)*, pages 239–248, 2016.

[15] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010. URL `http://yann.lecun.com/exdb/mnist/`. Accessed 2019.

[16] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2041–2050, 2018.

[17] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning depth from single monocular images using deep convolutional neural fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 38(10):2024–2039, 2016.

[18] Lee-Kang Liu, Stanley H. Chan, and Truong Q. Nguyen. Depth reconstruction from sparse samples: Representation, algorithm, and sampling. *IEEE Transactions on Image Processing*, 24(6):1983–1996, 2015.

[19] Miaomiao Liu, Mathieu Salzmann, and Xuming He. Discrete-continuous depth estimation from a single image. In *2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 716–723, 2014.

[20] Yaojie Liu, Amin Jourabloo, and Xiaoming Liu. Learning deep models for face anti-spoofing: Binary or auxiliary supervision. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 389–398, 2018.

[21] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–8, 2018.

[22] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Costas Bekas, and Cristiano Malossi. BAGAN: Data augmentation with balancing GAN. *Computing Research Repository (CoRR)*, abs/1803.09655, 2018.

[23] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *Computing Research Repository (CoRR)*, abs/1411.1784, 2014.

[24] David Pfau and Oriol Vinyals. Connecting generative adversarial networks and actor-critic methods. In *2016 Conference on Neural Information Processing Systems (NIPS) Workshop on Adversarial Training*, 2016.

[25] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *2016 International Conference on Learning Representations (ICLR)*, 2016.

[26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. In David E. Rumelhart and James L. Mcclelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Volume 1: Foundations*, pages 318–362. 1986.

[27] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *2016 Advances in Neural Information Processing Systems (NIPS)*, pages 2234–2242. 2016.

[28] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. Learning depth from single monocular images. In *2006 Advances in Neural Information Processing Systems (NIPS)*, pages 1161–1168. 2006.

[29] Robin Tibor Schirrmeister, Patryk Chrabszcz, Frank Hutter, and Tonio Ball. Training Generative Reversible Networks. In *International Conference on Machine Learning (ICML) 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

[30] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *2012 European Conference on Computer Vision (ECCV)*, 2012.

[31] Yinda Zhang and Thomas Funkhouser. Deep depth completion of a single RGB-D image. In *2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 175–185, 2018.

[32] Tinghui Zhou, Matthew Brown, Noah Snavely, and David Lowe. Unsupervised learning of depth and ego-motion from video. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6612–6619, 2017.

[33] Jun-Yan Zhu, Richard Zhang, Deepak Pathak, Trevor Darrell, Alexei A. Efros, Oliver Wang, and Eli Shechtman. Toward multimodal image-to-image translation. In *2017 Advances in Neural Information Processing Systems (NIPS)*, pages 465–476. 2017.