Object Discovery via Layer Disposal

by

Deniz Oktay

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author .....
Department of Electrical Engineering and Computer Science
August 18, 2017

Certified by .....
Antonio Torralba
Professor
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Object Discovery via Layer Disposal

by

Deniz Oktay

Submitted to the Department of Electrical Engineering and Computer Science
on August 18, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

A key limitation of semantic image segmentation approaches is that they require large amounts of densely labeled training data. In this thesis, we introduce a method to learn to segment images with unlabeled data. The intuition behind the approach is that removing objects from images will yield natural images, however removing random patches will yield unnatural images. We capitalize on this signal to develop an auto-encoder that decomposes an image into layers, and when all layers are combined, it reconstructs the input image. However, when a layer is removed, the model learns to produce a different image that still looks natural to an adversary, which is possible by removing objects. Experiments and visualizations suggest that this model automatically learns to segment objects in images better than baselines.

Some parts of this thesis represent joint work with Dr. Carl Vondrick and Professor Antonio Torralba.

Thesis Supervisor: Antonio Torralba
Title: Professor

Acknowledgments

I would first and foremost like to thank Antonio Torralba and Carl Vondrick for their help and guidance that have been invaluable to helping me grow as a researcher. The ideas in this work have resulted from the many discussions we have had.

I would like to thank Antonio for always providing inspiration for new ways to think about a problem and ideas to move forward at every point along the way, and to provide a great opportunity for starting my research career. His enthusiasm for the field is certainly contagious.

I would like to thank Carl for his endless patience and constant feedback over the past two years. The process of scientific research seems a lot more approachable thanks to him, and it has been a great pleasure. Continuing on in my career, I would be thrilled to have the positive impact as a mentor on someone that Carl has had on me.

I would also like to thank my family for their endless support, and my friends for the countless memories throughout my years at MIT. It has been a great honor working with many of you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Semantic image segmentation is a central problem in computer vision to categorize each pixel of an image into an object category, and has wide practical applications in scene understanding. However, current approaches rely on datasets containing a large number of images that have been annotated by a human pixel-by-pixel. A key to improving segmentation models is to collect more and more data, but annotation can be prohibitively expensive. For example, a human expert [38, 4] typically takes an hour to densely annotate all pixels in just one training image! With corresponding datasets for object recognition reaching millions of images [25, 37], how do we efficiently scale up semantic segmentation?

In this thesis, we work on a slightly weaker form of full image segmentation. Our goal is to, given an object category, identify all pixels in a set of images that belong to that category. The difference between our definition and full image segmentation is illustrated in Figure 1-1. We introduce and evaluate a principle for solving this problem with unlabeled images. The only prior knowledge about the images are that they are from a single scene category, such as bedrooms or kitchens. The main intuition behind our idea is that when an object is removed from an image, the resulting image should still look natural to a human eye. We propose and evaluate a method for using this idea as a signal for segmentation. For example, if you remove the bed from the scene in Figure 1-2, the image is still realistic. However, if you only partially remove the bed, the image is not realistic anymore. This principle provides

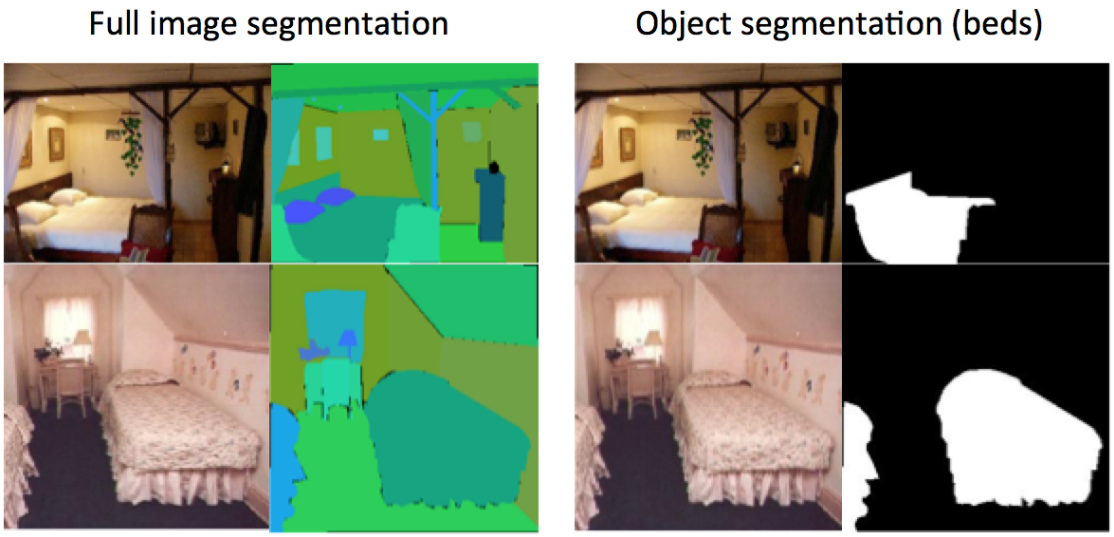


Figure 1-1: **Full-scene segmentation vs object segmentation:** Our goal is not to segment the entire image, but to label all the pixels from a certain object category. Note that we do not require training a separate model per object category. The model trained on unlabeled images discovers objects automatically. Examples are taken from the ADE20K dataset [38].

Figure 1-2: **Layer Disposal for Segmentation:** We make the simple observation that if you remove an object from an image, the image still looks natural (middle). However, if you remove a random patch, the image likely looks unnatural (right). In this paper, we use this intuition as a signal to learn to semantically segment images. Our model learns segmentations such that when a region is removed, the image looks perceptually real to an adversary, which it does by removing objects.

a high-level rule for grouping: when a pixel is removed, its neighboring pixels should also be removed if they belong to the same object. We operationalize this intuition to automatically segment images.

We develop a neural network model that decomposes images into layers. We train this model so that when all layers are combined together, it reconstructs the input image. However, we also train the model so that if we randomly remove a layer, the combination still appears perceptually real to an adversary trying to discriminate between real images and generated images. Consequently, the model learns a layered image decomposition that allows parts of the image to be removed. We show that the model automatically learns to remove objects in order to make the output image still appear realistic, a signal we capitalize on for learning to segment. Note that although the problem statement deals with labeling a given object category, we do not have to train a different model for each of these. We train a single model on images in a scene category, and it automatically discovers the various objects. Although our model could theoretically be used for full-image segmentation, we do not evaluate its capability of doing so. Such and application to full-image segmentation and evaluation could be part of future work.

We present three main experiments to analyze this approach, presented in Chapters 4 and 5. Firstly, experiments show that our model learns to automatically segment images into objects for some scene categories, even without any labeled training data, and our approach outperforms several basic baselines. Secondly, we show that we can combine a small amount of labeled data with our approach to improve performance. Finally, visualizations suggest that the model allows us to see behind objects automatically, enabling graphics applications such as removing windows from a picture or taking off the bed sheets.

1.1 Related Work

1.1.1 Image Segmentation

Pixel-wise segmentation is widely studied in computer vision. Edge and boundary detection seeks to recognize contours between objects [5, 20, 8, 15], but does not attach category labels to segments. In constrast, semantic segmentation seeks to both segment objects and assign labels, which is the task that we consider. [29, 33, 2, 3] learn to semantically segment objects in images, however they require large amounts of manual supervision. In this work, we do not require pixel-wise labeled data in order to learn to segment objects; we only require images that are known to be within a certain scene category. In related efforts, [19] investigate segmenting objects behind occlusions, but also require supervision. [10] explore how to remove occlusions from images, but require specifying the occlusions a priori. Our work is most related to Sudderth and Jordan [27], which also use layered models for unsupervised segmentation. However, our work differs because we learn a single model for semantic segmentation that can work across multiple images.

1.1.2 Layered Visual Models

Layered image models are widely used in computer vision [31, 34, 28, 32, 11, 30], however here we are leveraging them to segment images without pixel-level human supervision. We develop a model that learns to decompose an image into separate layers, which we use for segmentation. [34] is similar to our work in that they generate images by layers, however they do not show that randomly removing layers is a signal for semantic segmentations.

1.1.3 Noise in Learning

Dropout [26] is commonly used in neural networks to regularize training by randomly dropping hidden unit activations. [13] also randomly drops neural layers to facilitate training. Our work uses similar mechanism to randomly drop generated layers, but

we do it to encourage a semantic decomposition of images into layers of objects. Note that the layers we drop are image layers, not layers of a neural network.

1.1.4 Emergent Units

Our work is related to the emergent behavior of neural networks. For example, recent work shows that hidden units automatically emerge to detect some concepts in visual classification tasks [36] and natural language tasks [22]. In this work, we also rely on the emergent behavior of deep neural networks. However, we design the task so segmentation explicitly emerges.

1.1.5 Unsupervised Representation Learning

Methods to learn representations from unlabeled data are related but different to this work. For example, spatial context [7] and word context [21] can be used as supervisory signals for vision and language respectively. While our work is also using unlabeled data, we are not learning representations. Rather, we are directly learning to segment images.

1.2 Contributions and Thesis Overview

The main contribution of this thesis is to introduce a novel method for learning to segment images in a given scene category with unlabeled data by capitalizing on the observation that removing a region from an image will result in an unnatural image unless the region masks an object. The remainder of this thesis describes this contribution in detail. Chapter 2 gives a brief background into the methods used. Chapter 3 present our method to auto-encode images with a layered decomposition, and shows how removing image regions is a useful signal for segmentation. Chapter 4 shows several quantitative experiments for semantic segmentation. Chapter 5 shows some example qualitative results of images generated layer-by-layer and Chapter 6 offers concluding remarks.

Chapter 2

Background

2.1 Convolutional Neural Networks

Deep Learning [18] has seen a lot of success in recent years in both supervised and unsupervised tasks in computer vision, sparked by the performance of AlexNet [17] in the ImageNet Large Scale Visual Recognition Challenge [24], an annual image classification challenge. Convolutional Neural Networks (CNNs) such as AlexNet are the most commonly used in vision applications. The main components of CNNs are convolution operations that take into account spatial locality of pixels during learning and inference. CNNs are a parametric model where the parameters are learned to optimize a given objective function for a task on either labeled or unlabeled data. The exact architecture and objective function of a convolutional neural network is task dependent and in our case will be specified in later sections.

There is much current research on CNNs and their applications. Our contributions are not focused on creating new CNN architectures, but on techniques to apply commonly used architectures to unsupervised image segmentation. In many places in our model, the architectures can be substituted for others; hence improvements in CNN architectures could lead to a direct improvement of our qualitative and quantitative results.

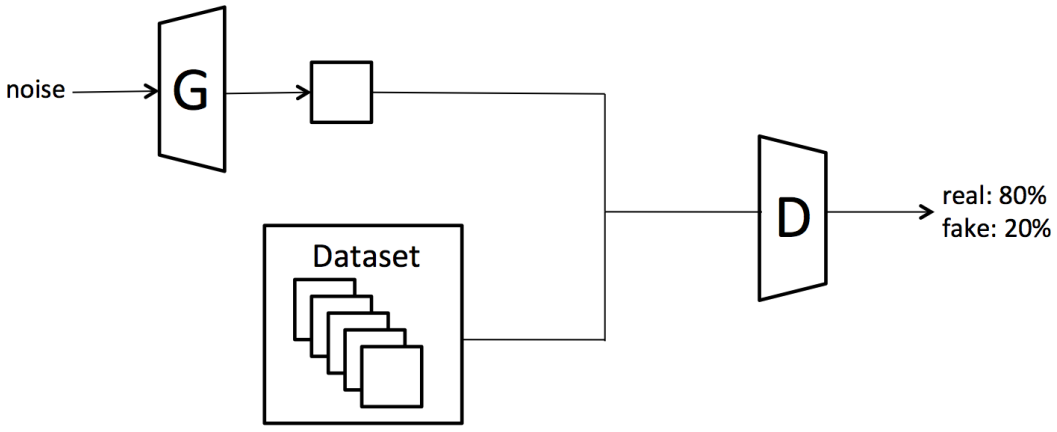


Figure 2-1: **GAN illustration:** The network G attempts to fool network D by generating realistic looking images. In the case of image generation, D and G are both convolutional neural networks.

2.2 Generative Adversarial Networks

A big focus in the deep learning research community has been generative modeling. Informally, given an un-annotated dataset, a generative model aims to create similar objects that are not in the dataset, but could belong in it. Essentially, the generative model attempts to learn and sample from the probability distribution of the dataset.

In this thesis we use generative image modeling as a tool for unsupervised image segmentation. In particular, we use the approach of Generative Adversarial Networks (GANs) [12, 23]. In GANs, two networks compete against each other. The generative network, G, maps noise vectors into images, while a discriminative network, D, takes as input either an image from a dataset or an output of G and attempts to discriminate between the two. The task of G is to maximize the error of D. The overall idea is illustrated in Figure 2-1.

(a) Generations of DCGAN bedrooms

(b) Real samples from LSUN bedrooms

(c) Generations of DCGAN kitchens

(d) Real samples from LSUN kitchens

(e) Generations of DCGAN towers

(f) Real samples from LSUN towers

Figure 2-2: **GAN Examples:** On the left are outputs of the trained generator. On the right are real samples from the dataset used to train the GAN. The architecture used during training is that of DCGAN: Deep Convolutional Generative Adversarial Networks. [23].

Formally, let $x_i \in \mathbb{R}^{W \times H}$ be an unlabeled image in our dataset, and $z \in \mathbb{R}^D$ be a noise vector sampled from $U \equiv \text{Unif}[-1, 1]$, the uniform distribution on $[-1, 1]$. For parametric functions $G_{\theta_G} : \mathbb{R}^D \to \mathbb{R}^{W \times H}$ and $D_{\theta_D} : \mathbb{R}^{W \times H} \to \mathbb{R}$, we wish to find θ_G, θ_D solving the objective function

$$\min_{\theta_G} \max_{\theta_D} \sum_{z \sim U} \log(D_{\theta_D}(G_{\theta_G}(z))) + \sum_i \log(1 - D_{\theta_D}(x_i))$$

The networks are trained using variants of Stochastic Gradient Descent, most commonly Adam [16].This objective function is intractable in practice, so GANs are trained using coordinate ascent; in each iteration of gradient descent the two networks are updated independently and one after the other.

GANs using convolutional neural networks are known to generate high quality images. An example set of generations of models trained on various LSUN [35] scene categories is shown in Figure 2-2, together with real samples from the dataset. One limitation of GANs is that they require a large amount of training data. Due to the large number of examples available in the LSUN dataset, we use this as the main training dataset for our generative model.

Chapter 3

Method

We present a method for learning a semantic segmentation model by taking advantage of a layered version of generative adversarial networks. Let $x_i \in \mathbb{R}^{W \times H}$ be an unlabeled image in our dataset. Note that for simplicity of notation, we assume gray-scale images, however our method easily extends to color images. We follow an encoder-decoder setup. We will encode an image into a latent code $z_i \in \mathbb{R}^D$, then decode the code into K image layers. The decoder mirrors G in the adversarial network setup in the previous chapter.

We describe the model in two phases. We first describe a generative model that extends the original DCGAN to produce realistic images in a layered representation. Similar to DCGAN, it will take as input randomly sampled noise and output an image. In our case, the Generator will be stochastic; there is a one-to-many mapping from noise vectors to output images, as opposed to the one-to-one mapping in the case of the DCGAN generator. One technicality in most current GAN architectures is that the mapping is not of noise vector to image, but of batch of noise vectors to batch of images. This is due to the batch normalization layer, and could be remedied with solutions such as layer normalization [1]. The second part of our model is the inference phase. We reinterpret the noise vector in the generator as a latent vector, and given an image from the dataset we wish to find a latent vector that when fed into the generator reconstructs the image. Assuming the generator reconstructs the image in layers, we will have essentially split the input image into layers containing different

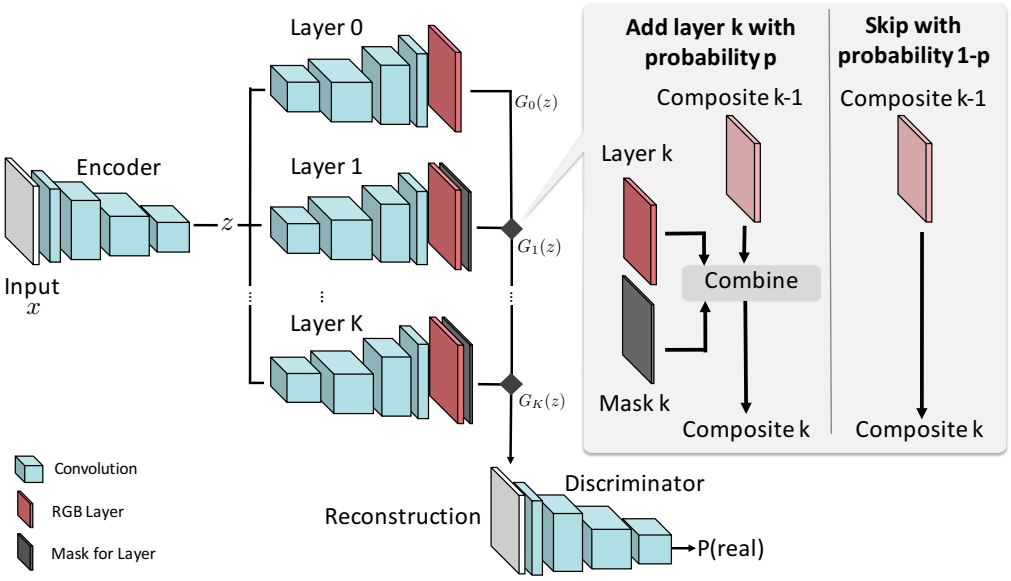


Figure 3-1: **Network Architecture:** We visualize our neural network architecture. Given an input image, we generate K layers and masks to combine them. However, each layer only has a certain probability of being combined with the current composite.

objects in each layer, which is our goal. Under a perfect generator that fully captures the distribution of the input dataset, there is always a latent vector that reconstructs the image perfectly. But in reality, the generator cannot be perfect. Therefore there is not necessarily a latent vector that perfectly reconstructs the image. This creates difficulties for inference, which we address in this chapter.

3.1 Generation Model

We use a simple layered model for image generation. Given a latent code or noise vector $z \in \mathbb{R}^D$, we stochastically and recursively decode it to produce an image:

$$G_0(z) = f_0(z) \tag{3.1}$$

$$G_k(z) = \begin{cases} f_k(z) \odot m_k(z) + G_{k-1}(z) \odot (1 - m_k(z)) & \text{with prob. } p_k \\ G_{k-1}(z) & \text{otherwise} \end{cases} \tag{3.2}$$

The kth layer is only added with probability p_k. Our intention is that the neural networks $m_k(z) \in \mathbb{R}^{W \times H}$ will generate a mask and $f_k(z) \in \mathbb{R}^{W \times H}$ will generate a foreground image to be combined with the previous layer. To ensure the mask and foreground are in a valid range, we use a sigmoid and tanh activation function respectively. $\odot$ denotes element-wise product. The base case of the recursion, $G_0(z)$, is the background layer and always present. To obtain the final image, we recurse a fixed number of times K to obtain the result $G_K(z)$. K is a hyper-parameter in our network, and any number of methods of assignments to p_k could work. In our work, we assume all p_k are the same.

3.1.1 Layer Disposal

The generation model $G_K(z)$ is stochastic because each layer is only added with a certain probability. We will train $G_K(z)$ to generate images that still look perceptually real to an adversary even when some layers are disposed. To be robust to this type of corruption, we hypothesize that the model will learn to place objects in each layer. Removing objects will fool the adversary, however removing an arbitrary patch will not fool the adversary because those images do not occur in nature.

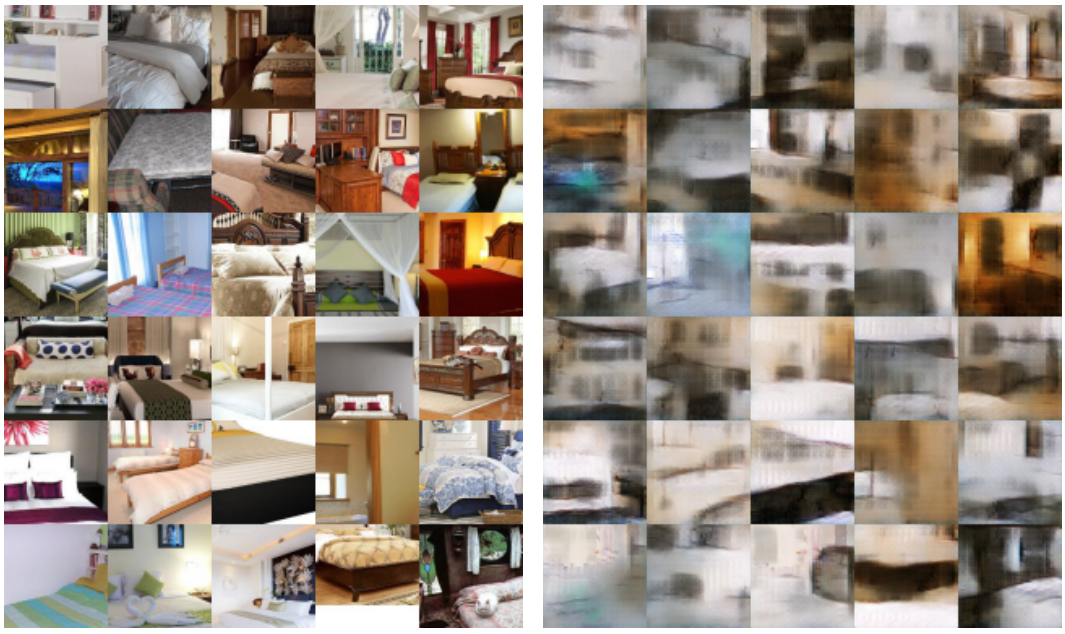
We train $G_K(z)$ to generate images that fool an adversary. To do this, we use $G_K(z)$ as the generator in a GAN. We use a convolutional network D as a discriminator and optimize our generator G to fool D while simultaneously training D to distinguish between generated images and images from the dataset. Figure 3-2 shows a few final generations of our model. Note that the final composite images look very similar in quality to the composite images of original DCGAN, shown in Chapter 2. Although the images are formed layer by layer, the training method allows the final image to look realistic. It is important to note that we use the exact same training objectives and discriminator as the original DCGAN, although our generator architecture is slightly different. As better and better models for GANs are available, we may use them in our model to improve our quality. Figure 3-6 shows a few qualitative examples of learned layer generations from this model. Notice the network can automatically learn a decomposition of objects and their boundaries.

(a) Regular Layered GAN Generations (b) Reduced bottleneck Generations

Figure 3-2: **Example Generations:** We visualize some example generations of our model. They are similar in quality to DCGAN, although they were produced in a layered representation. Note that there is less color variation in the reduced bottleneck images, possibly due to using a latent vector with spatial dimensions.

3.1.2 Reduced Bottleneck

One important difference in our generator as opposed to the original DCGAN generator is that our latent vector has spatial dimensions instead of being a flat vector. We are essentially reducing the bottleneck of the autoencoder. We choose $z \in \mathbb{R}^{D \times W_0 \times H_0}$, where D, W_0 and H_0 are hyper-parameters in our model. We assume that the generator is not able to capture the full variability of the probability distribution of the dataset. Since we want to use our generations for semantic segmentation, we care about the location of objects in the generations a lot more than overall color of the image. By adding spatial dimensions to the latent vector, we hypothesize that we are able to capture more variability in object locations instead of color differences. We see a qualitative example of this in Figure 3-2. It seems that the model trained with spatial dimensions has a lot less variability in color than the original model.



(a) Original Images (b) Reconstructions

Figure 3-3: **Pixel-wise Reconstructions of Real Images:** The original images are on the left and the reconstructions are on the right. Note that the textures of the reconstructions are similar to the original images, but there are no clear object boundaries, and the images do not look realistic.

3.2 Inference Model

We have so far described the generation process given a latent code z. To segment an image x, we need to infer this code. We will train an encoder $E(x)$ to predict the code given an image x. We investigate two strategies to infer the latent code. The first method tries to match the image with its reconstruction, and the second method tries to match the latent codes directly.

As a note, in the samples shown in this section we do not use the reduced bottleneck. We use the original latent vector of DCGAN with no spatial dimensions, although reconstruction quality is similar in either case. In the final model in which we report results in the next chapter we use the reduced bottleneck.

(a) Original Images (b) Reconstructions

Figure 3-4: **Pixel-wise Reconstructions of Fake Images:** The original images are on the left and the reconstructions are on the right. The reconstructions are better, even though the encoder was not trained on images from the dataset.

3.2.1 Pixel-wise Image Reconstruction

One possible strategy is to train E to minimize the pixel-wise reconstruction error over our unlabeled dataset. The encoder would be given by

$$E = \arg\min_{E} \sum_{i} \|G_K(E(x_i)) - x_i\|_2^2$$

In this case we also have a choice of either training the generator G_K with this objective along with the standard GAN objective, or letting it stay fixed while training the encoder only. It is known that the GAN training objective is unstable and difficult to balance with other objectives, so we decided to keep the generator fixed while training the encoder with the above objective. The resulting reconstructions are shown in Figure 3-3. The reconstructions are low-quality, and not fit to be used for object segmentation. We hypothesize that this is the case because the generator cannot fully capture the probability distribution of the dataset. Therefore, when the

encoder is training to minimize pixel-wise distance to a real image, it cannot match exactly and training is unstable, since it is not guaranteed that there exists a latent vector that reconstructs the image.

Another strategy is to train the encoder to reconstruct generated images, i.e. outputs of $G_K(z)$ for randomly sampled z. Although this seems an indirect method for reconstruction since we care in the end about reconstructing real images from the dataset, the problem is guaranteed to have a solution, since by definition the output of the generator will be in its own range. After training, we give as input to the encoder real images and look at reconstruction quality. The result is pictured in Figure 3-4.

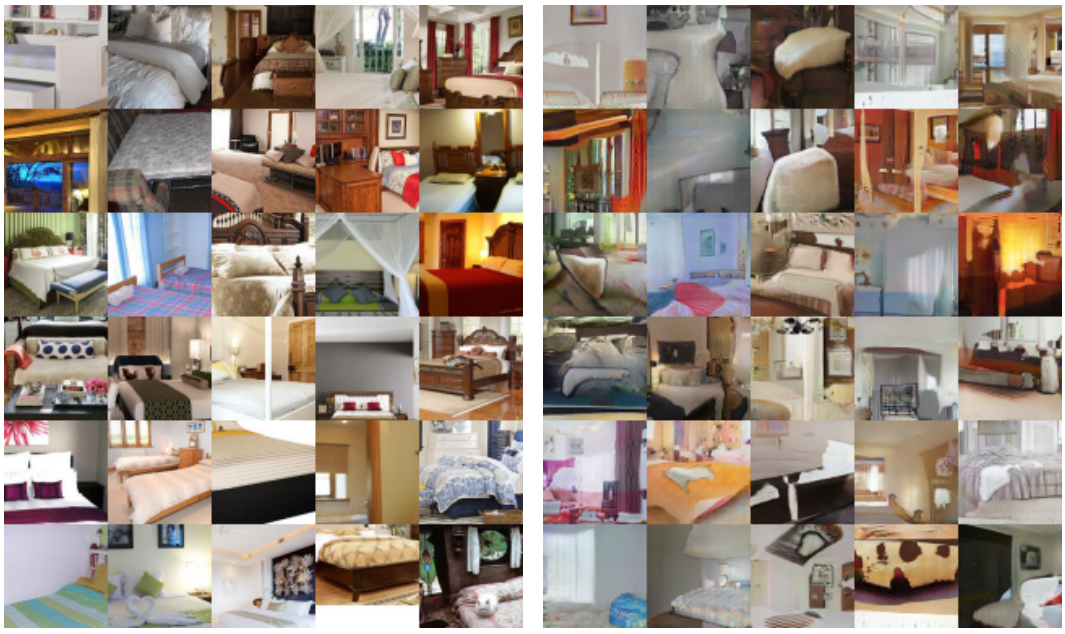
Note in the pixel-wise reconstruction models we have to account for the fact that the generator is stochastic; each generator output for a single latent vector might have different layers enabled. We account for this by training a binary classifier for each layer. The binary classifier is a standard CNN classifier. This classifier takes as input the input image and the output layers of the generator, and decides which layers should be included in the image for the reconstruction. It takes as input the layered decomposition of the reconstruction, not the original layered decomposition. This is not the architecture we end up using for the final model, as the reconstructions are still not strong and the overall architecture is unnecessarily complicated, so it is not included in our model architecture figure above.

3.2.2 Latent Vector Reconstruction

The reconstructions in both cases above were not strong. We hypothesize that this is because the mean-squared reconstruction in pixel-space is not semantically meaningful. We therefore use a different strategy. We will train E to reconstruct the latent codes from sampled scenes from the generator. The new objective function becomes

$$\min_{E} \sum_{z \sim \mathcal{N}(0,I)} \|E\left(G_K(z)\right) - z\|_2^2$$

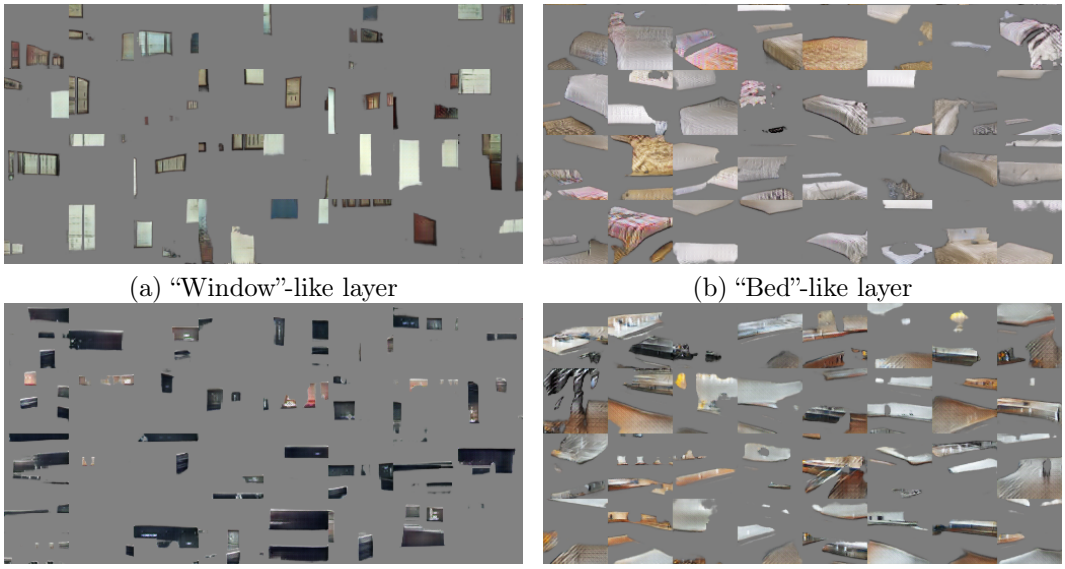
While this does not guarantee a strong reconstruction in pixel space, it may enable a more semantic reconstruction, which is our goal. We note this strategy is discussed



(a) Original Images (b) Reconstructions

Figure 3-5: **Latent Vector Reconstructions of Fake Images:** The original images are on the left and the reconstructions are on the right. The reconstructions are much higher quality, and have crisp object boundaries. In addition, the images are semantically very similar.

by [9], but they do not experimentally use it. Other inference strategies are also possible, however this is beyond the scope of this thesis. Note that once again, during training the encoder never sees images from the dataset; it only trains on outputs of the generator, $G_K(z)$. Only during testing does the encoder see real images from the dataset. We visualize the reconstructions for this new training method in Figure 3-5. The reconstructions are now much higher quality. We also note that the reconstructions are semantically similar instead of pixel-wise similar; they generally have similar types of objects in similar locations as the original image, although the exact details of the object might be different.



(a) "Window"-like layer

(b) "Bed"-like layer

(c) "Kitchen Appliance"-like layer

(d) "Kitchen Counter"-like layer

Figure 3-6: **Example Layers:** We visualize some generations from different layers. For example, some layers specialize to segmenting and generating windows, while others specialize to beds.

3.3 Learning

We learn the parameters of the neural networks D, E, and G jointly. We optimize:

$$\min_{D,E} \sum_{z \sim U} \left[\log D\left(G_K(z)\right) + \lambda \| E\left(\bar{G}_K(z)\right) - z \|_2^2 \right] + \sum_i \log\left(1 - D\left(x_i\right)\right) \tag{3.3}$$

$$\max_{G} \sum_{z \sim U} \log D\left(G_K(z)\right) \tag{3.4}$$

where U is the uniform distribution on the interval $[-1, 1]$ and $\bar{G}$ indicates that no layers are dropped. To optimize this min-max objective, we alternate between minimizing Equation 3.3 and maximizing Equation 3.4 using mini-batch stochastic gradient descent. Note that this objective is similar to a generative adversarial network [12], however there is also an encoder E. We use $\lambda = 1$. Importantly, to train our model, we only need a collection of unlabeled images. The model will learn to auto-encode images such that layers can be randomly removed and still produce a realistic image.

3.4 Semantic Segmentation

We take advantage of the emergent masks of the layers for semantic segmentation. After training, we will have K different masks $m_k(z)$. Since K is typically small (we use $K = 5$), we can manually inspect a few examples on the training set and attach a name to each one. We use these masks as the semantic segmentation prediction. Figure 3-6 shows a few examples of learned masks from this model. Note that not all masks are meaningful.

3.5 Network Architecture and Implementation Details

Our network architecture is similar to DCGAN [23] when possible. The encoder contains 3 layers of 4x4 convolutions with a stride of 2, followed by a single layer of 3x3 convolutions of stride 1, and then another single layer of 4x4 convolutions of stride 2. Since we use reconstructions for image segmentation, we care about encoding spatial location of the objects, so we use a latent vector of size 64 x 4 x 4. The decoder has identical architecture, but contains up-convolutions instead. Each layer is generated independently from the hidden state vector without tied weights.

We add batch normalization [14] between layers, leaky ReLU for the encoder and discriminator and ReLU for the generator. We train with Adam [16] with learning rate 0.0002 and beta 0.5 for the object discovery experiments and learning rate 0.00002 for finetuning. We train for 2 epochs over the dataset for both scene categories. In all examples we use 5 foreground layers and set the probability that a layer is included to 0.4. We plan on making all code and data publicly available.

Chapter 4

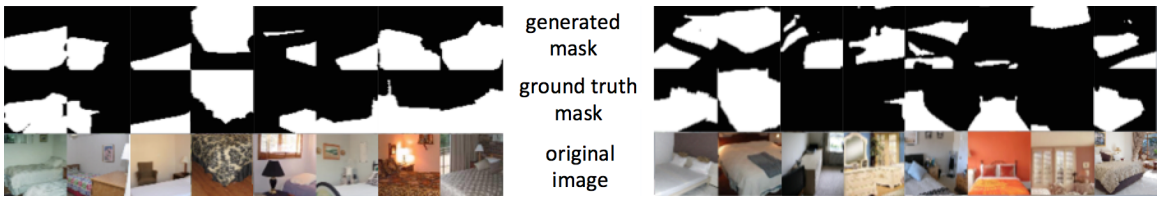
Quantitative Experiments

We present two quantitative experiments to evaluate our model. We evaluate how well layers automatically emerge to classify pixels to belong to a specific object category. We present both an unsupervised experiment and a semi-supervised experiment, where we augment our model with very few labeled examples to further improve our results.

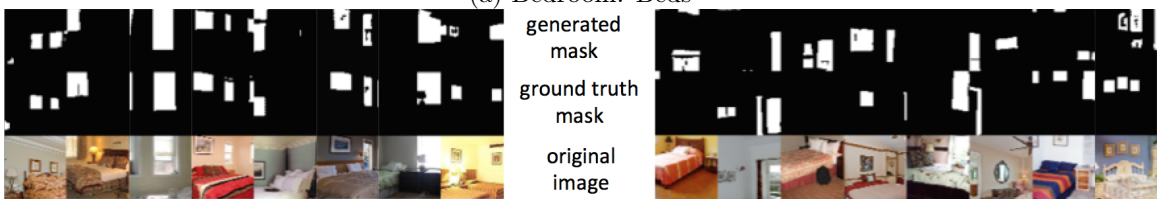
4.1 Experimental Setup

We experiment with our approach using images of certain scene categories from the LSUN dataset [35], specifically bedrooms and kitchens. For bedrooms, we focus on segmenting bed and window. For kitchens, we focus on segmenting appliances and countertop. The dataset contains a total of $3,033,042$ images of bedrooms and $2,212,277$ images of kitchens which we train separate models on. Note that apart from scene category, these images are otherwise unlabeled, and do not have any pixel level annotations. We random crop images to $3 \times 64 \times 64$ and scale to $[-1, 1]$.

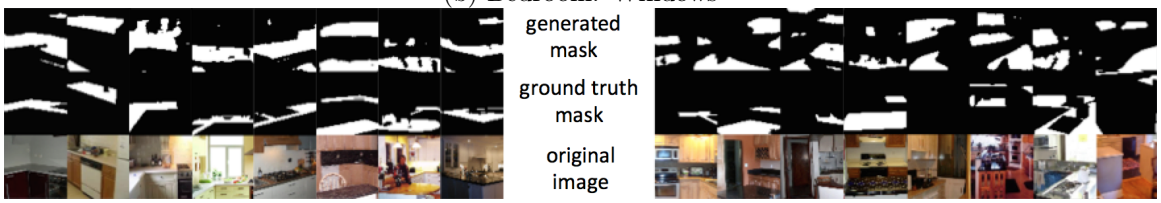
Precision-Recall and Average Precision: A key to our quantitative evaluation is using precision-recall curves and average precision. For a binary classifier, the *precision* is the fraction of the selected elements that are positives. The *recall*, on the other hand, are the fraction of the total number of positives selected. The precision-recall curve plots the recall as the independent variable and precision as the dependent variable. A perfect classifier would have perfect precision of 1.0 for all values of recall



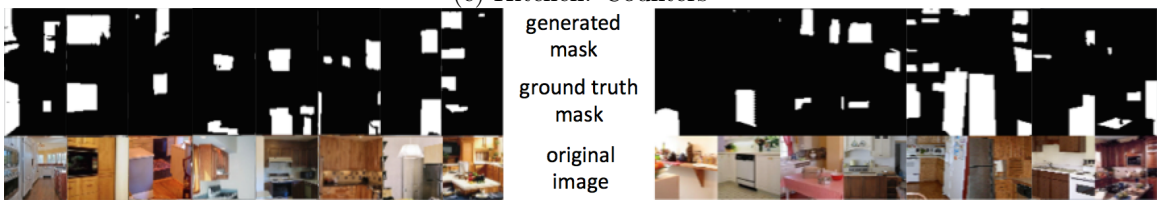
(a) Bedroom: Beds



(b) Bedroom: Windows



(c) Kitchen: Counters



(d) Kitchen: Appliances

Figure 4-1: **Example Results:** The left image contains some cases where we correctly segment the mask of objects, and the right image contains some failure cases. The first row is the generated mask, second row is the ground truth mask, and third row is the input image.

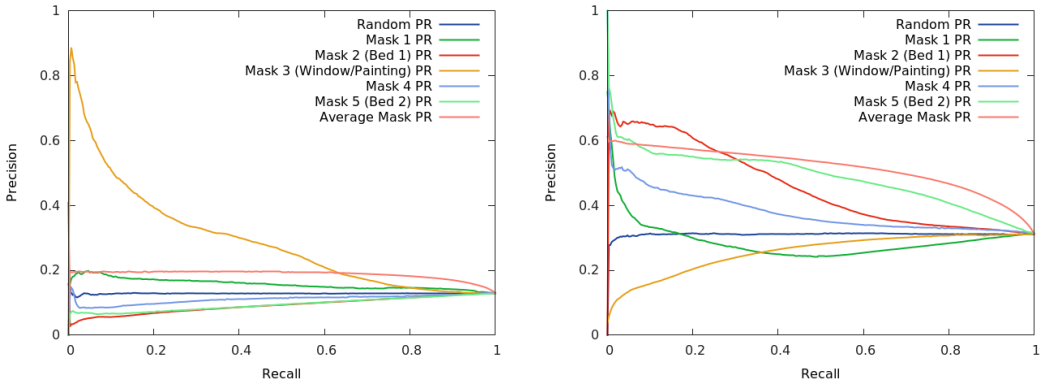
< 1.0. If the classifier assigns a score to each element, we assume we select the highest scores first. A perfect classifier would select all positives before selecting any negatives. The average precision is the area under the precision-recall curve, and is 1.0 for a perfect classifier.

		Bedrooms		Kitchens	
		Bed	Window	Appliance	Counter
without Labeled Images	Random	0.31	0.13	0.10	0.07
	Autoencoder	0.37	0.20	<0.10	**0.11**
	Kmeans 1×1	0.34	0.17	**0.15**	0.08
	Kmeans 7×7	0.34	0.15	**0.14**	0.07
	Disposal	**0.50**	**0.30**	**0.13**	**0.10**
with Labeled Images	Average Mask	0.52	0.19	0.12	0.10
	Random Init	0.58	0.32	0.17	0.11
	Disposal + Prior	0.59	0.33	0.13	0.11
	Disposal + Finetune	**0.69**	**0.50**	**0.19**	**0.15**

Table 4.1: **Semantic Segmentation Average Precision:** We report average precision (area under precision-recall curve) on pixel-wise classification for four object categories. Our approach can semantically segment images without supervision better than simple baselines. Moreover, the model can be fine-tuned with a little bit of labeled data to further improve results.

We do require some images with ground truth for evaluation. We use images and labels from the ADE20K dataset [38] for the kitchen and bedroom scene categories as the test set. For each scene category, we create a training dataset and validation dataset of randomly selected examples. For bedrooms, the training and validation each contain 640 examples. For kitchens, they each contain 320 examples. The sizes are limited by the number of annotations available in ADE20K for each scene category. We chose kitchens and bedrooms as they are the largest scene categories in the LSUN dataset and because we have a sufficient number of densely labeled examples in ADE20K.

For each identified object category in each scene, we create binary masks from the ADE20K dataset and pair them with their corresponding images. Due to the fact that ADE20K does not label behind occlusions, we combine labels to form the appropriate ground truth map. For example, pillows are often on the bed. We therefore define beds as the combination of beds, pillows, and comforters. For kitchen appliances, we define them as microwaves, ovens, dishwashers, stoves, and sinks. We evaluate the model versus baselines as pixel-wise binary classification. The mask represents the confidence of model that the pixel belongs to the specified object category. We run each experiment on a scene category and report the average precision as our metric.



(a) Windows (b) Beds

Figure 4-2: **Precision-Recall:** We plot precision-recall curves for each layer's mask. Our approach obtains good precision with low recall, suggesting that the model's most confident segmentations are fairly accurate. Notice how layers tend to specialize to certain object categories. The mask from layer 3 works well for segmenting windows, but the same layer does not work for beds. A similar trend exists for mask 5, but segments beds. This suggests that the model learns to group objects.

4.2 Object Discovery

We quantitatively evaluate how well our model is able to do semantic segmentation without labeled training images in Table 4.1. Our results suggest that a decomposition of objects is automatically emerging in our model, which can semantically segment objects better than unsupervised baselines.

For each scene category, we train on the LSUN dataset with 5 foreground layers. We then inspect the layers on the training set and identify which layer visually best represents each object category. We then extract the masks from the layers of each object that we chose. If multiple layers are chosen, we average the output masks together. We use the outputs of these masks as scores and calculate average precision. We also graph the precision-recall curves for the two objects for bedrooms in Figure 4-2. Important thing to note is that when evaluated on the bed objects, masks 2 and 5 indeed do the best, while mask 3 does worse than random. When evaluated on window objects, however, mask 3 does the best and masks 2, 5 do worse than random. This shows that each mask generally captures a single object, suggesting the masks are learning a semantic decomposition.

We compare to a few simple baselines. The random benchmark corresponds to a random guess for each pixel. The autoencoder benchmark corresponds to training

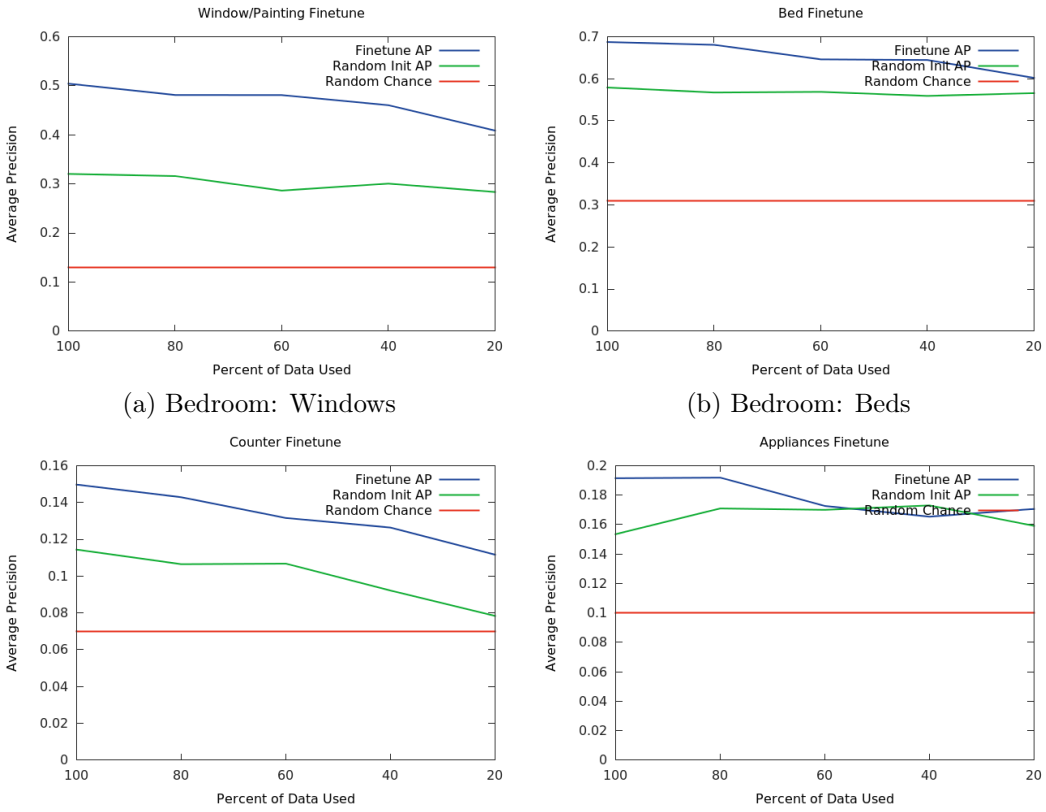
the model with the disposal probability set to 0. In almost every case, our model with disposal receives a higher average precision, suggesting that removing layers does help to obtain an object segmentation. The kmeans baseline corresponds to clustering RGB patches across the dataset, and using distance to cluster centers as a segmentation score. We try both 1×1 and 7×7 patches with the same number of clusters as our model ($K = 5$). For each object category, we find the best performing cluster center on the ADE20K training set and evaluate with this cluster center on the validation set. In almost every case, our model outperforms this baseline, suggesting that we do not just learn a color decomposition of the image.

Finally, we conduct an ablation on the model. In our experiments, each layer is initialized both randomly and independently. We also tried initializing each stream identically (but still randomly). We found that performance significantly dropped (beds dropped to 0.37 AP and windows dropped to 0.14 AP) and each stream produces similar outputs. This suggests that initializing layers independently helps each layer specialize to different objects.

4.3 Refining with Labels

We explore incorporating human labels into the model. As a baseline, we calculate the average segmentation over the ADE20K dataset for each object category. For each object category we average each mask from the labeled dataset and we evaluate with this single mask. Recall that our model did not have access to this prior because it never saw a densely labeled image!

Table 4.1 shows results for the Average Mask. For most object categories, the unlabeled model outperforms the average mask, which suggests that our model can outperform naive priors estimated with labeled data, even though it never saw labels. For the bed objects, the simple prior does better, possibly because beds are large and usually in a certain location (bottom). Windows, on the other hand, could be in many different locations and are usually smaller, hence the prior could not perform as well. Since we perform well in low-recall regions and lose to the prior in high-recall



(a) Bedroom: Windows

(b) Bedroom: Beds

(c) Kitchen: Counter

(d) Kitchen: Appliances

Figure 4-3: **Performance versus size of labeled data:** We plot segmentation performance versus the size of the labeled dataset during fine-tuning. In general, more labeled data improves performance.

regions, we obtain further improvement by averaging the output of the model and the prior. Results are shown under Disposal + Prior. As expected, the bed category improves, while the window category does not obtain as much improvement.

We experiment further with semi-supervised learning by fine-tuning the best performing mask for each object class. After training the model without labels as before, we choose the mask best suited for each object category and train the network with labeled examples from the ADE20K dataset for the object category. In other words, we use the unsupervised learning algorithm as an initialization for the supervised version of the problem, where images from ADE20K are the inputs and the masks from ADE20K are the labels. We also experiment with varying the size of the training set. We report average precision, and as a baseline we use the same model but initialized randomly from scratch. We run each model for 150 epochs over the training set.

Plots are shown in Figure 4-3 and average precision is reported in Table 4.1. In each case fine-tuning beats random initialization, and is the best performing model overall in all object classes. The plots also show that in many cases the fine-tuning model outperforms scratch initialization even with 20% of the training data. This shows that in the semi-supervised setting the model can be trained with much fewer examples, which is favorable in areas such as image segmentation where labeled data is expensive. Another interesting note is that for the window object class our unsupervised model actually comes close to the supervised random initialization model, which suggests that the unsupervised model in this case is strong enough to outperform a non-trivial supervised model.

Chapter 5

Qualitative Results

We qualitatively give several examples of images that are built up layer by layer in Figure 5-1. For each example we give the original image that was used as input, partial reconstructions for each layer that is added, the layer that is added, and the mask that the layer uses. These results suggest that as the generative model improves we will be able to remove layers from images to see behind objects. As an example, in the bottom right we can see that when the bed layer (layer 5) is removed we are able to uncover the headboard behind it.

In addition, we show further examples in Figure 5-2 where layers are always kept during the generation process; p_k for each layer is 0. We see that layers are a generally less meaningful decomposition, even though the final reconstruction is of similar quality. This qualitatively shows that the layer disposal actually provides a signal to the generator to create meaningful layer decompositions.

5.1 Cityscapes and LSUN Towers

We also show visualizations from a model trained on the Cityscapes dataset [6] and the LSUN Towers dataset in Figure 5-3. The Cityscapes dataset did not learn very well, as can be seen by the reconstruction quality and the layer decompositions. We hypothesize that this is because of the size of the dataset. With only 30,000 images, it does not have enough to train the GAN and Encoder to convergence. The LSUN

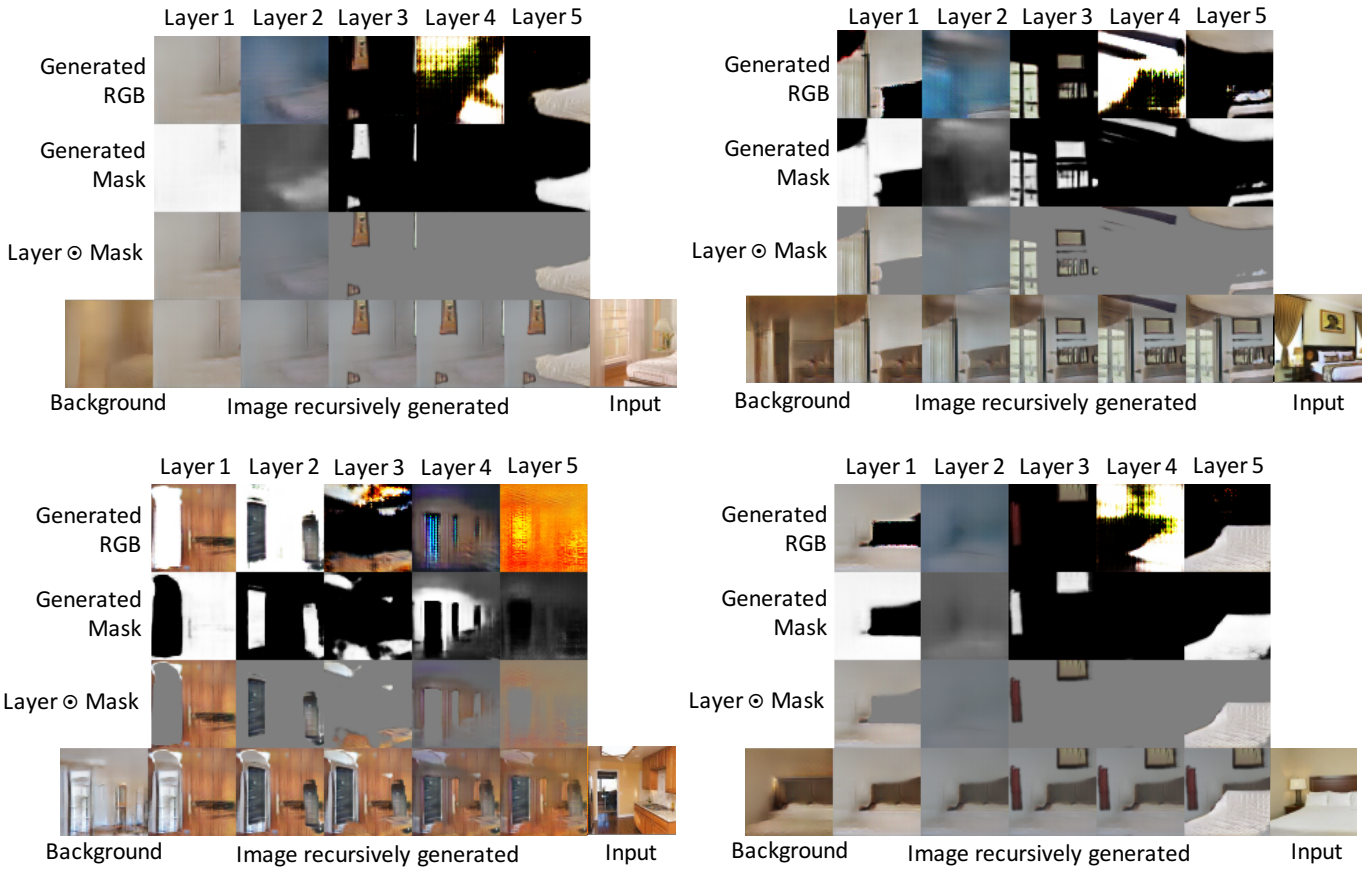


Figure 5-1: **Visualizing Layer Decomposition:** We visualize the outputs of different layers from our model given an input image. Different layers automatically emerge to both reconstruct objects and their masks, which we use for semantic segmentation. Moreover, this enables potential graphics applications, such as de-occluding objects in an image.

Towers dataset has 708,264 images. The reconstructions look generally good, but the layer decomposition is not perfect. Many layers have seemed to learn the complete image. This could be due to the images only having a single object. Explorations to improve these models could be the basis of future work.

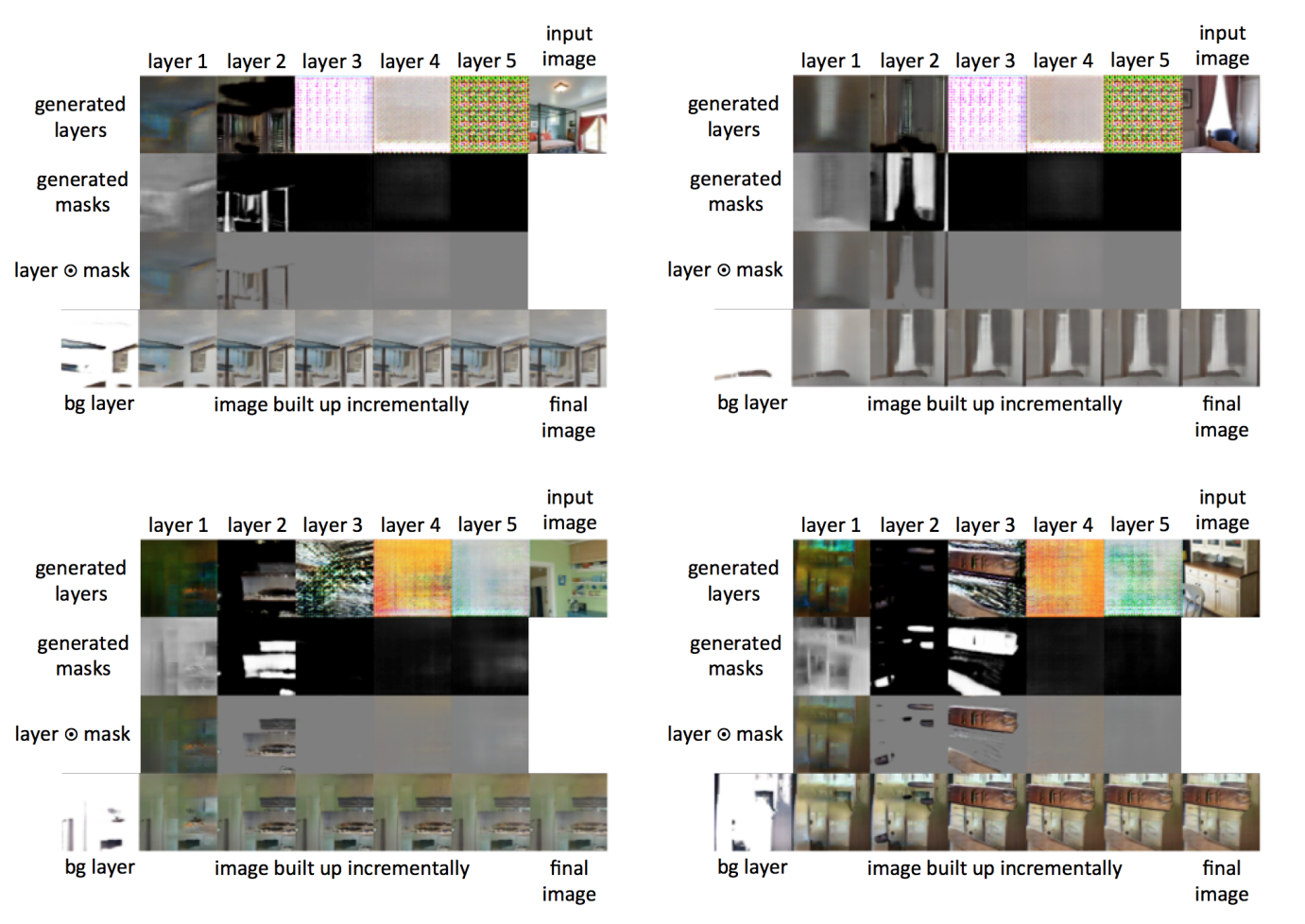


Figure 5-2: **Layer decomposition without disposal:** When disposal probabilities are set to 0, we observe that the layers do not learn the decomposition as well. It seems that compared to the previous model a lot of the layers do not learn anything.

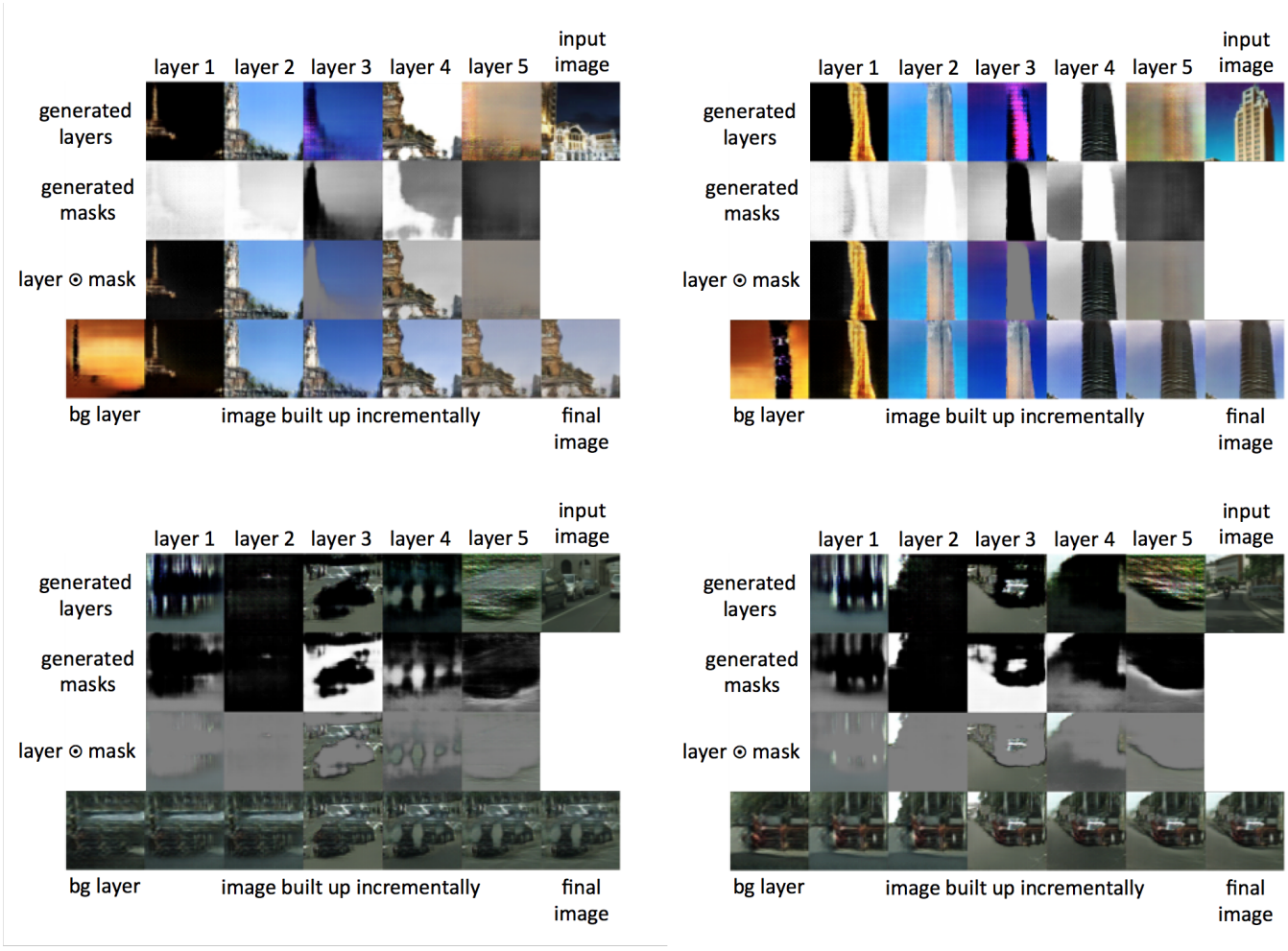


Figure 5-3: **Cityscapes and LSUN Towers:** In Cityscapes the reconstructions not look good, as well as the layer decompositions. We hypothesize this is because of the lack of training examples in the dataset. The LSUN Towers dataset has generally good reconstructions, but the layered decomposition is faulty.

Chapter 6

Conclusion and Future Work

We propose a simple principle for learning to do semantic segmentation with unlabeled data. We capitalize on the observation that removing a region from an image will result in an unnatural image unless the region masks an object. Since annotating large-scale and pixel dense training data for segmentation is expensive, we believe developing approaches for segmentation without labeled data can have significant impact.

This work opens much future directions for improvement and research. A topic that should be studied rigorously is why the different layers learn different objects. Although we offer an ablation study that suggests it could be due to random initialization, it is not conclusive. Furthermore, if we have an explicit method of causing each layer to be unique, we could possibly add many more layers to the model to capture even more detailed object segmentations. Another extension is making the probabilities of each layer a learnable parameter. With this additional degree of freedom, the model could automatically learn object frequencies, and possibly also lead to more robust models that extend to many scenes.

We believe that this could also open up a new application of GANs. By replacing the GAN generator with a network with a certain structure, as we did by replacing it with a layered network, we could learn to generate samples with the same structure. Along with an accurate inference mechanism we could reconstruct original examples from a dataset to fit the structure in an unsupervised fashion.

Bibliography

[1] Lei Jimmy Ba, Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016.

[2] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *arXiv preprint arXiv:1511.00561*, 2015.

[3] Aayush Bansal, Xinlei Chen, Bryan Russell, Abhinav Gupta, and Deva Ramanan. Pixelnet: Towards a general pixel-level architecture. *arXiv preprint arXiv:1609.06694*, 2016.

[4] Adela Barriuso and Antonio Torralba. Notes on image annotation. *arXiv preprint arXiv:1210.3448*, 2012.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on PAMI*, (6), 1986.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *CVPR*, 2015.

[8] Piotr Dollar, Zhuowen Tu, and Serge Belongie. Supervised learning of edges and object boundaries. In *CVPR*, volume 2. IEEE, 2006.

[9] Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *arXiv preprint arXiv:1606.00704*, 2016.

[10] Kiana Ehsani, Roozbeh Mottaghi, and Ali Farhadi. Segan: Segmenting and generating the invisible. *arXiv preprint arXiv:1703.10239*, 2017.

[11] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In *NIPS*, 2016.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NIPS*, 2014.

[13] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*. Springer, 2016.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[15] Phillip Isola, Daniel Zoran, Dilip Krishnan, and Edward H Adelson. Crisp boundary detection using pointwise mutual information. In *ECCV*. Springer, 2014.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 05 2015.

[19] Ke Li and Jitendra Malik. Amodal instance segmentation. In *ECCV*. Springer, 2016.

[20] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE transactions on pattern analysis and machine intelligence*, 26(5):530–549, 2004.

[21] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *NIPS*, 2013.

[22] Alec Radford, Rafal Jozefowicz, and Ilya Sutskever. Learning to generate reviews and discovering sentiment. *arXiv preprint arXiv:1704.01444*, 2017.

[23] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[24] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[26] Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *JMLR*, 15(1):1929–1958, 2014.

[27] Erik B Sudderth and Michael I Jordan. Shared segmentation of natural scenes using dependent pitman-yor processes. In *NIPS*, 2009.

[28] Deqing Sun, Erik B Sudderth, and Hanspeter Pfister. Layered rgbd scene flow estimation. In *CVPR*, 2015.

[29] Joseph Tighe and Svetlana Lazebnik. Understanding scenes on many levels. In *ICCV*. IEEE, 2011.

[30] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Generating videos with scene dynamics. *NIPS*, 2016.

[31] John YA Wang and Edward H Adelson. Representing moving images with layers. *IEEE Transactions on Image Processing*, 3(5):625–638, 1994.

[32] Jianwei Yang, Anitha Kannan, Dhruv Batra, and Devi Parikh. Lr-gan: Layered recursive generative adversarial networks for image generation. *arXiv preprint arXiv:1703.01560*, 2017.

[33] Jimei Yang, Brian Price, Scott Cohen, and Ming-Hsuan Yang. Context driven scene parsing with attention to rare classes. In *CVPR*, 2014.

[34] Yi Yang, Sam Hallman, Deva Ramanan, and Charless C Fowlkes. Layered object models for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(9):1731–1743, 2012.

[35] Fisher Yu, Yinda Zhang, Shuran Song, Ari Seff, and Jianxiong Xiao. LSUN: construction of a large-scale image dataset using deep learning with humans in the loop. *CoRR*, abs/1506.03365, 2015.

[36] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Object detectors emerge in deep scene cnns. *arXiv preprint arXiv:1412.6856*, 2014.

[37] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.

[38] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *arXiv preprint arXiv:1608.05442*, 2016.