# PIXELVAE: A LATENT VARIABLE MODEL FOR NATURAL IMAGES

**Ishaan Gulrajani**[1,*], **Kundan Kumar**[1,2], **Faruk Ahmed**[1], **Adrien Ali Taiga**[1,3], **Francesco Visin**[1,5], **David Vazquez**[1,4], **Aaron Courville**[1,6]
[1] Montreal Institute for Learning Algorithms, Université de Montréal
[2] Department of Computer Science and Engineering, IIT Kanpur
[3] CentraleSupélec
[4] Computer Vision Center & Universitat Autonoma de Barcelona
[5] Politecnico di Milano
[6] CIFAR Fellow

## ABSTRACT

Natural image modeling is a landmark challenge of unsupervised learning. Variational Autoencoders (VAEs) learn a useful latent representation and model global structure well but have difficulty capturing small details. PixelCNN models details very well, but lacks a latent code and is difficult to scale for capturing large structures. We present PixelVAE, a VAE model with an autoregressive decoder based on PixelCNN. Our model requires very few expensive autoregressive layers compared to PixelCNN and learns latent codes that are more compressed than a standard VAE while still capturing most non-trivial structure. Finally, we extend our model to a hierarchy of latent variables at different scales. Our model achieves state-of-the-art performance on binarized MNIST, competitive performance on $64 \times 64$ ImageNet, and high-quality samples on the LSUN bedrooms dataset.

## 1 INTRODUCTION

Building high-quality generative models of natural images has been a long standing challenge. Although recent work has made significant progress (Kingma & Welling, 2014; van den Oord et al., 2016a;b), we are still far from generating convincing, high-resolution natural images.

Many recent approaches to this problem are based on an efficient method for performing amortized, approximate inference in continuous stochastic latent variables: the variational autoencoder (VAE) (Kingma & Welling, 2014) jointly trains a top-down decoder generative neural network with a bottom-up encoder inference network. VAEs for images typically use rigid decoders that model the output pixels as conditionally independent given the latent variables. The resulting model learns a useful latent representation of the data and effectively models global structure in images, but has difficulty capturing small-scale features such as textures and sharp edges due to the conditional independence of the output pixels, which significantly hurts both log-likelihood and quality of generated samples compared to other models.

PixelCNNs (van den Oord et al., 2016a;b) are another state-of-the-art image model. Unlike VAEs, PixelCNNs model image densities autoregressively, pixel-by-pixel. This allows it to capture fine details in images, as features such as edges can be precisely aligned. By leveraging carefully constructed masked convolutions (van den Oord et al., 2016b), PixelCNNs can be trained efficiently in parallel on GPUs. Nonetheless, PixelCNN models are still very computationally expensive. Unlike typical convolutional architectures they do not apply downsampling between layers, which means that each layer is computationally expensive and that the depth of a PixelCNN must grow linearly with the size of the images in order for it to capture dependencies between far-away pixels. PixelCNNs also do not explicitly learn a latent representation of the data, which can be useful for downstream tasks such as semi-supervised learning.

[*] Corresponding author; `igul222@gmail.com`

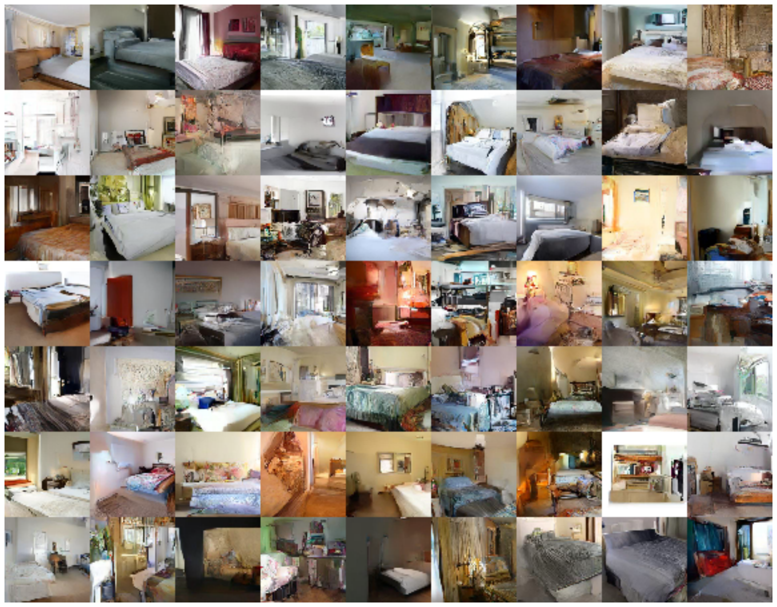


Figure 1: Samples from hierarchical PixelVAE on the LSUN bedrooms dataset.

Our contributions are as follows:

- We present PixelVAE, a latent variable model which combines the largely complementary advantages of VAEs and PixelCNNs by using PixelCNN-based masked convolutions in the conditional output distribution of a VAE.
- We extend PixelVAE to a hierarchical model with multiple stochastic layers and autoregressive decoders at each layer. This lets us autoregressively model not only the output pixels but also higher-level latent feature maps.
- On MNIST, we show that PixelVAE: (1) establishes a new state-of-the-art likelihood, (2) performs comparably to PixelCNN using far fewer computationally expensive autoregressive layers, (3) learns more compressed latent codes than a standard VAE while still accounting for most non-trivial structure, and (4) learns a latent code which separates digits better than a standard VAE.
- We evaluate hierarchical PixelVAE on two challenging natural image datasets ($64 \times 64$ ImageNet and LSUN bedrooms). On $64 \times 64$ ImageNet, we report likelihood competitive with the state of the art at significantly less computational cost. On LSUN bedrooms, we generate high-quality samples and show that hierarchical PixelVAE learns to model different properties of the scene with each of its multiple layers.

## 2 Related Work

There have been many recent advancements in generative modeling of images. We briefly discuss some of these below, especially those that are related to our approach.

The Variational Autoencoder (VAE) (Kingma & Welling, 2014) is a framework to train neural networks for generation and approximate inference jointly by optimizing a variational bound on the data log-likelihood. The use of normalizing flows (Rezende & Mohamed, 2015) improves the flexibility of the VAE approximate posterior. Based on this, Kingma et al. (2016) develop an efficient formulation of an autoregressive approximate posterior model using MADE (Germain et al., 2015). In our work, we avoid the need for such flexible inference models by using autoregressive priors.

The idea of using autoregressive conditional likelihoods in VAEs has been explored in the context of language modeling in (Bowman et al., 2016), however in that work the use of latent variables fails to improve likelihood over a purely autoregressive model.

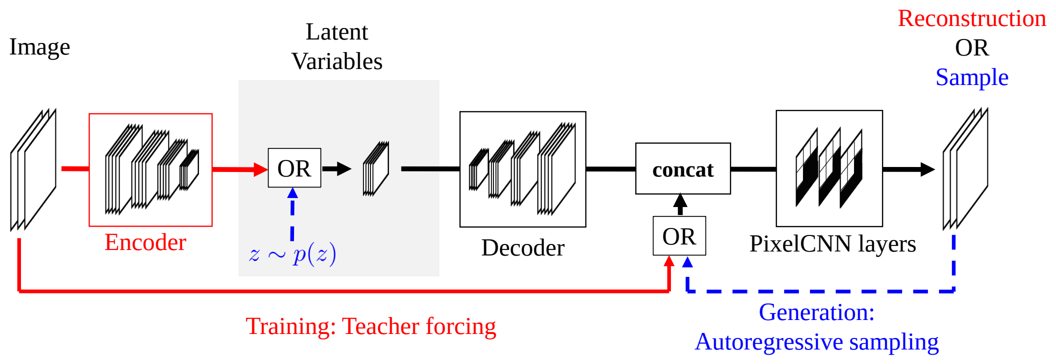


Figure 2: Our proposed model, PixelVAE, makes use of PixelCNN to model an autoregressive decoder for a VAE. VAEs, which assume (conditional) independence among pixels, are known to suffer from blurry samples, while PixelCNN, modeling the joint distribution, produces sharp samples, but lack a latent representation that might be more useful for downstream tasks. PixelVAE combines the best of both worlds, providing a meaningful latent representation, while producing sharp samples.

Simultaneously to our work, Chen et al. (2016) present a VAE model for images with an an autoregressive output distribution. In constrast to Chen et al. (2016), who focus on models with a single layer of latent variables, we also investigate models with a hierarchy of latent variables (and corresponding autoregressive priors) and show that they enable us to scale our model to challenging natural image datasets.

Another promising recent approach is Generative Adversarial Networks (GANs) (Goodfellow et al., 2014), which pit a generator network and a discriminator network against each other. Recent work has improved training stability (Radford et al., 2015; Salimans et al., 2016) and incorporated inference networks into the GAN framework (Dumoulin et al., 2016; Donahue et al., 2016). GANs generate compelling samples compared to our work, but still exhibit unstable training dynamics and are known to underfit by ignoring modes of the data distribution (Dumoulin et al., 2016). Further, it is difficult to accurately estimate the data likelihood in GANs.

## 3 PixelVAE Model

Like a VAE, our model jointly trains an "encoder" inference network, which maps an image $x$ to a posterior distribution over latent variables $z$, and a "decoder" generative network, which models a distribution over $x$ conditioned on $z$. The encoder and decoder networks are composed of a series of convolutional layers, respectively with strided convolutions for downsampling in the encoder and transposed convolutions for upsampling in the decoder.

As opposed to most VAE decoders that model each dimension of the output independently (for example, by modeling the output as a Gaussian with diagonal covariance), we use a conditional PixelCNN in the decoder. Our decoder models $x$ as the product of each dimension $x_i$ conditioned on all previous dimensions and the latent variable $z$:

$$p(x|z) = \prod_i p(x_i|x_1, \ldots, x_{i-1}, z)$$

We first transform $z$ through a series of convolutional layers into feature maps with the same spatial resolution as the output image and then concatenate the resulting feature maps with the image. The resulting concatenated feature maps are then further processed by several PixelCNN masked convolutional layers and a final PixelCNN 256-way softmax output.

Unlike typical PixelCNN implementations, we use very few PixelCNN layers in our decoder, relying on the latent variables to model the structure of the input at scales larger than the combined receptive

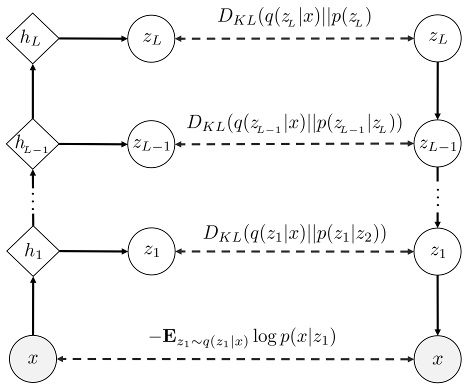

$h_L$
$z_L$
$D_{KL}(q(z_L|x)||p(z_L))$
$z_L$
$h_{L-1}$
$z_{L-1}$
$D_{KL}(q(z_{L-1}|x)||p(z_{L-1}|z_L))$
$z_{L-1}$
$h_1$
$z_1$
$D_{KL}(q(z_1|x)||p(z_1|z_2))$
$z_1$
$-\mathbf{E}_{z_1 \sim q(z_1|x)} \log p(x|z_1)$
$x$
$x$

Figure 3: We generate top-down through a hierarchical latent space decomposition. The inference network generates latent variables by composing successive deterministic functions to compute parameters of the stochastic random variables. Dotted lines denote contributions to the cost.

field of our PixelCNN layers. As a result of this, our architecture captures global structure at a much lower computational cost than a standard PixelCNN implementation.

### 3.1 Hierarchical architecture

The performance of VAEs can be improved by stacking them to form a hierarchy of stochastic latent variables: in the simplest configuration, the VAE at each level models a distribution over the latent variables at the level below, with generation proceeding downward and inference upward through each level (i.e. as in Fig. 3). In convolutional architectures, the intermediate latent variables are typically organized into feature maps whose spatial resolution decreases toward higher levels.

Our model can be extended in the same way. At each level, the generator is a conditional PixelCNN over the latent features in the level below. This lets us autoregressively model not only the output distribution over pixels but also the prior over each set of latent feature maps. The higher-level PixelCNN decoders use diagonal Gaussian output layers instead of 256-way softmax, and model the dimensions within each spatial location (i.e. across feature maps) independently. This is done for simplicity, but is not a limitation of our model.

The output distributions over the latent variables for the generative and inference networks decompose as follows (see Fig. 3).

$$\begin{aligned} p(z_1, \cdots, z_L) &= p(z_L) p(z_{L-1}|z_L) \cdots p(z_1|z_2) \\ q(z_1, \cdots, z_L|x) &= q(z_1|x) \cdots q(z_L|x) \end{aligned}$$

We optimize the negative of the evidence lower bound (sum of data negative log-likelihood and KL-divergence of the posterior over latents with the prior).

$$\begin{aligned} -L(x,q,p) &= -E_{z_1 \sim q(z_1|x)} \log p(x|z_1) + D_{KL}(q(z_1, \cdots z_L|x)||p(z_1, \cdots, z_L)) \\ &= -E_{z_1 \sim q(z_1|x)} \log p(x|z_1) + \int\limits_{z_1,\cdots,z_L} \prod_{j=1}^{L} q(z_j|x) \sum_{i=1}^{L} \log \frac{q(z_i|x)}{p(z_i|z_{i+1})} dz_1 ... dz_L \\ &= -E_{z_1 \sim q(z_1|x)} \log p(x|z_1) + \sum_{i=1}^{L} \int\limits_{z_1,\cdots,z_L} \prod_{j=1}^{L} q(z_j|x) \log \frac{q(z_i|x)}{p(z_i|z_{i+1})} dz_1 ... dz_L \\ &= -E_{z_1 \sim q(z_1|x)} \log p(x|z_1) + \sum_{i=1}^{L} \int\limits_{z_i, z_{i+1}} q(z_{i+1}|x) q(z_i|x) \log \frac{q(z_i|x)}{p(z_i|z_{i+1})} dz_i dz_{i+1} \end{aligned}$$

$$= -E_{z_1 \sim q(z_1|x)} \log p(x|z_1) + \sum_{i=1}^{L} \mathbf{E}_{z_{i+1} \sim q(z_{i+1}|x)} \left[ D_{KL}(q(z_i|x) || p(z_i|z_{i+1})) \right]$$

Note that when specifying an autoregressive prior over each latent level $z_i$, we can leverage masked convolutions (van den Oord et al., 2016b) and samples drawn independently from the approximate posterior $q(z_i \mid x)$ (i.e. from the inference network) to train efficiently in parallel on GPUs.

## 4 Experiments

### 4.1 MNIST

| **Model** | **NLL Test** |
|---|---|
| DRAW (Gregor et al., 2016) | $\leq 80.97$ |
| Discrete VAE (Rolfe, 2016) | $= 81.01$ |
| IAF VAE (Kingma et al., 2016) | $\approx 79.88$ |
| PixelCNN (van den Oord et al., 2016a) | $= 81.30$ |
| PixelRNN (van den Oord et al., 2016a) | $= 79.20$ |
| VLAE (Chen et al., 2016) | $= 79.03$ |
| Convolutional VAE | $\leq 87.41$ |
| PixelVAE | $\leq 80.64$ |
| Gated PixelCNN (our implementation) | $= 80.10$ |
| Gated PixelVAE | $\approx 79.48$ ($\leq 80.02$) |
| Gated PixelVAE without upsampling | $\approx$ **78.96** ($\leq 79.58$) |

Table 1: We compare performance of different models on binarized MNIST. "PixelCNN" is the model described in van den Oord et al. (2016a). Our corresponding latent variable model is "PixelVAE". "Gated PixelCNN" and "Gated PixelVAE" use the gated activation function in van den Oord et al. (2016b). In "Gated PixelVAE without upsampling", a linear transformation of latent variable conditions the (gated) activation in every PixelCNN layer instead of using upsampling layers.

We evaluate our model on the binarized MNIST dataset (Salakhutdinov & Murray, 2008; Lecun et al., 1998) and report results in Table 1. We also experiment with a variant of our model in which each PixelCNN layer is directly conditioned on a linear transformation of latent variable, $z$ (rather than transforming $z$ first through several upsampling convolutional layers) (as in (van den Oord et al., 2016b) and find that this further improves performance, achieving an NLL upper bound comparable with the current state of the art. We estimate the marginal likelihood of our MNIST model using the importance sampling technique in Burda et al. (2015), which computes a lower bound on the likelihood whose tightness increases with the number of importance samples per datapoint. We use $N = 5000$ samples per datapoint (higher values don't appear to significantly affect the likelihood estimate) and achieve state-of-the-art likelihood.

#### 4.1.1 Using Few PixelCNN Layers

The masked convolutional layers in PixelCNN are computationally expensive because they operate at the full resolution of the image and in order to cover the full receptive field of the image, PixelCNN typically needs a large number of them. One advantage of our architecture is that we can achieve strong performance with very few PixelCNN layers, which makes training and sampling from our model significantly faster than PixelCNN. To demonstrate this, we compare the performance of our model to PixelCNN as a function of the number of PixelCNN layers (Fig. 4a). We find that with fewer than 10 autoregressive layers, our PixelVAE model performs much better than PixelCNN. This is expected since with few layers, the effective receptive field of the PixelCNN output units is too small to capture long-range dependencies in the data.

We also observe that adding even a single PixelCNN layer has a dramatic impact on the NLL bound of PixelVAE. This is not surprising since the PixelCNN layer helps model local characteristics which

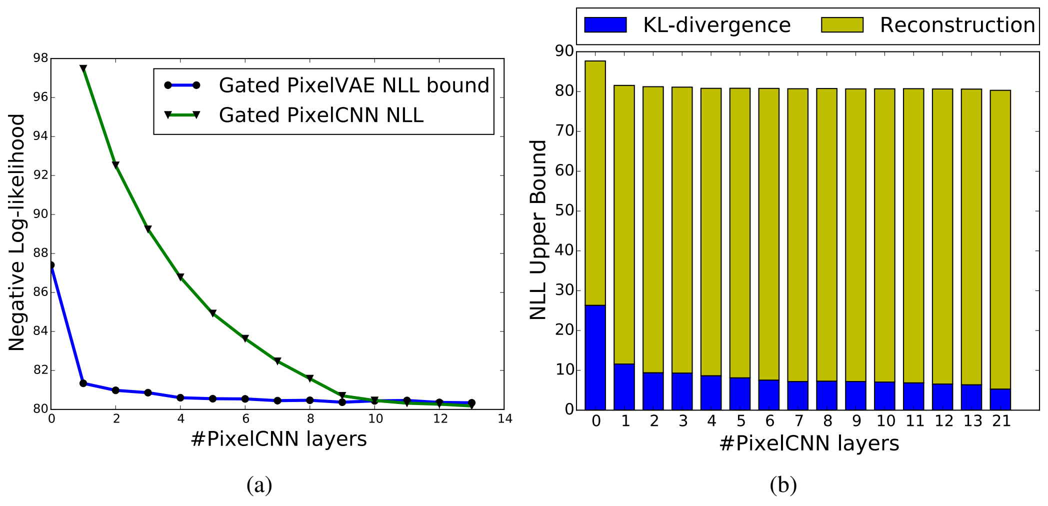


Figure 4: (a) Comparison of Negative log-likelihood upper bound of PixelVAE and NLL for PixelCNN as a function of the number of PixelCNN layers used. (b) Cost break down into KL divergence and reconstruction cost.

are complementary to the global characteristics which a VAE with a factorized output distribution models.

### 4.1.2 Latent Variable Information Content

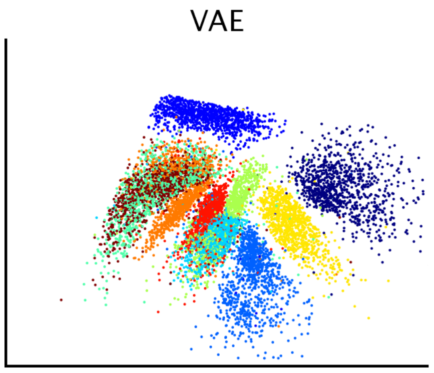
Because the autoregressive conditional likelihood function of PixelVAE is expressive enough to model some properties of the image distribution, it isn't forced to account for those properties through its latent variables as a standard VAE is. As a result, we can expect PixelVAE to learn latent representations which are invariant to textures, precise positions, and other attributes which are more efficiently modeled by the autoregressive decoder. To empirically validate this, we train PixelVAE models with different numbers of autoregressive layers (and hence, different PixelCNN receptive field sizes) and plot the breakdown of the NLL bound for each of these models into the reconstruction term $\log p(x|z)$ and the KL divergence term $D_{KL}(q(z|x)||p(z))$ (Fig. 4b). The KL divergence term can be interpreted as a measure of the information content in the posterior distribution $q(z|x)$ (in the sense that in expectation, samples from $q(z|x)$ require $KL(q||p)$ fewer bits to code under a code optimized for $q$ than under one optimized for $p$ (Burnham & Anderson, 2003)) and hence, models with smaller KL terms encode less information in their latent variables.

We observe a sharp drop in the KL divergence term when we use a single autoregressive layer compared to no autoregressive layers, indicating that the latent variables have been freed from having to encode small-scale details in the images. Since the addition of a single PixelCNN layer allows the decoder to model interactions between pixels which are at most 2 pixels away from each other (since our masked convolution filter size is $5 \times 5$), we can also say that most of the non-trivial (long-range) structure in the images is still encoded in the latent variables.

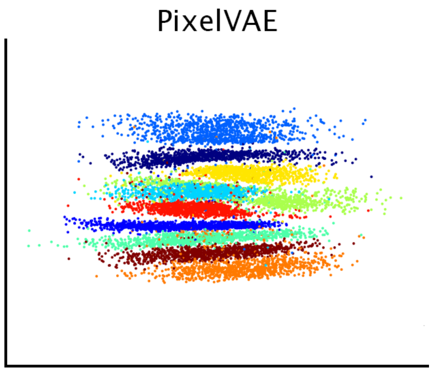
### 4.1.3 Latent Representations

On MNIST, given a sufficiently high-dimensional latent space, VAEs have already been shown to learn representations in which digits are well-separated (Sønderby et al., 2016). However, this task becomes more challenging as the capacity of the latent space is decreased. PixelVAE's flexible output distribution should allow it to learn a latent representation which is invariant to small details and thus better models global factors of variation given limited capacity.

To test this, we train a PixelVAE with a two-dimensional latent space, and an equivalent VAE. We visualize the distribution of test set images in latent space and observe that PixelVAE's latent representation separates digits significantly better than VAE (Figure 5). To quantify this difference, we train a K-nearest neighbors classifier in the latent space of each model and find that PixelVAE



(a)



(b)

Figure 5: Visualization of the MNIST test set in the latent space of (a) convolutional VAE and (b) PixelVAE with two latent dimensions. PixelVAE separates classes more completely than VAE.

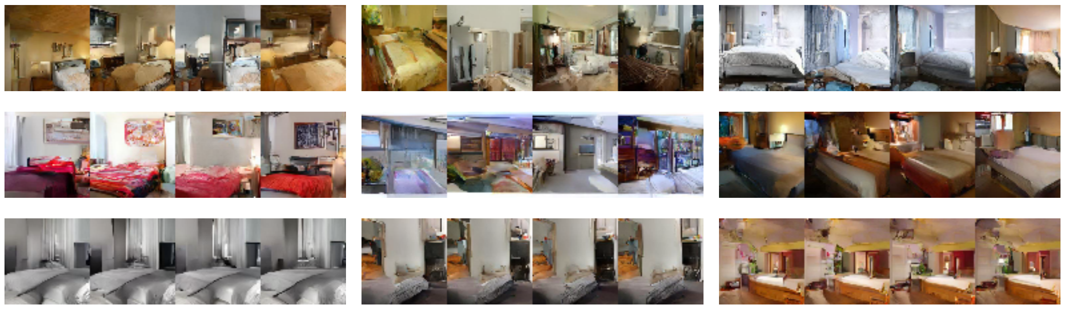


Figure 6: We visually inspect the variation in image features captured by the different levels of stochasticity in our model. For the two-level latent variable model trained on $64 \times 64$ LSUN bedrooms, we vary only the top-level sampling noise *(top)* while holding the other levels constant, vary only the middle-level noise *(middle)*, and vary only the bottom (pixel-level) noise *(bottom)*. It appears that the top-level latent variables learn to model room structure and overall geometry, the middle-level latents model color and texture features, and the pixel-level distribution models low-level image characteristics such as texture, alignment, shading.

significantly outperforms VAE, achieving a test error of 7.2% compared to VAE's 22.9%. We also note that unlike VAE, PixelVAE learns a representation in which digit identity is largely disentangled from other generative factors.

### 4.2 LSUN Bedrooms

To evaluate our model's performance with more data and complicated image distributions, we perform experiments on the LSUN bedrooms dataset (Yu et al., 2015). We use the same preprocessing as in Radford et al. (2015) to remove duplicate images in the dataset. For quantitative experiments we use a $32 \times 32$ downsampled version of the dataset, and we present samples from a model trained on the $64 \times 64$ version.

We train a two-level PixelVAE with latent variables at $1 \times 1$ and $8 \times 8$ spatial resolutions. We find that this outperforms both a two-level convolutional VAE with diagonal Gaussian output and a single-level PixelVAE in terms of log-likelihood and sample quality. We also try replacing the PixelCNN layers at the higher level with a diagonal Gaussian decoder and find that this hurts log-likelihood, which suggests that multi-scale PixelVAE uses those layers effectively to autoregressively model latent features.

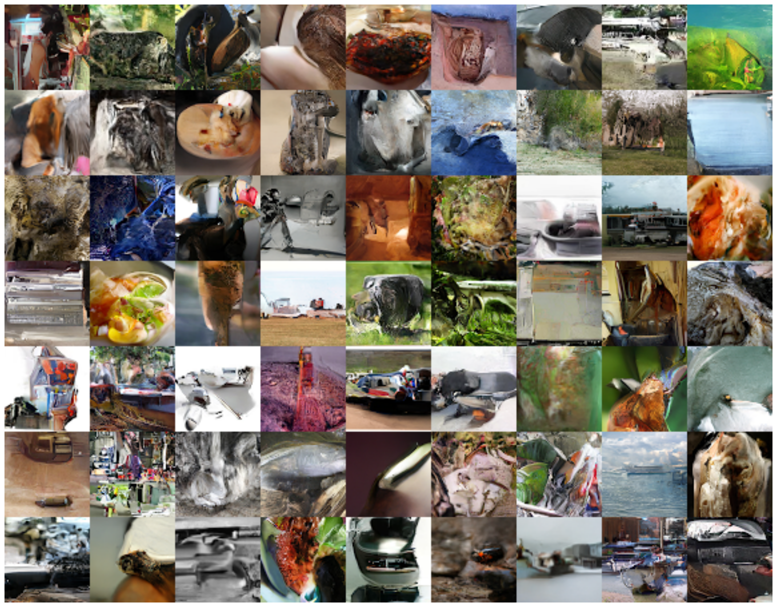


Figure 7: Samples from hierarchical PixelVAE on the 64x64 ImageNet dataset.

#### 4.2.1 FEATURES MODELED AT EACH LAYER

To see which features are modeled by each of the multiple layers, we draw multiple samples while varying the sampling noise at only a specific layer (either at the pixel-wise output or one of the latent layers) and visually inspect the resulting images (Fig. 6). When we vary only the pixel-level sampling (holding $z_1$ and $z_2$ fixed), samples are almost indistinguishable and differ only in precise positioning and shading details, suggesting that the model uses the pixel-level autoregressive distribution to model only these features. Samples where only the noise in the middle-level ($8 \times 8$) latent variables is varied have different objects and colors, but appear to have similar basic room geometry and composition. Finally, samples with varied top-level latent variables have diverse room geometry.

### 4.3 $64 \times 64$ IMAGENET

The $64 \times 64$ ImageNet generative modeling task was introduced in (van den Oord et al., 2016a) and involves density estimation of a difficult, highly varied image distribution. We trained a heirarchical PixelVAE model (with a similar architecture to the model in section 4.2) on $64 \times 64$ ImageNet and report validation set likelihood in Table 2. Our model achieves a likelihood competitive with van den Oord et al. (2016a;b), despite being substantially less computationally complex. A visual inspection of ImageNet samples from our model (Fig. 7) also reveals them to be significantly more globally coherent than samples from PixelRNN.

| **Model** | **NLL Validation (Train)** | **FLOPs** |
|---|---|---|
| Convolutional DRAW (Gregor et al., 2016) | $\leq 4.10$ (4.04) | — |
| Real NVP (Dinh et al., 2016) | $= 4.01$ (3.93) | — |
| PixelRNN (van den Oord et al., 2016a) | $= 3.63$ (3.57) | $154 \times 10^9$ |
| Gated PixelCNN (van den Oord et al., 2016b) | $=$ **3.57** (3.48) | $134 \times 10^9$ |
| Hierarchical PixelVAE | $\leq 3.62$ (3.55) | $63 \times 10^9$ |

Table 2: Model performance on $64 \times 64$ ImageNet. We achieve competitive NLL at a fraction of the computational complexity of other leading models.

## 5 Conclusions

In this paper, we introduced a VAE model for natural images with an autoregressive decoder that achieves strong performance across a number of datasets. We explored properties of our model, showing that it can generate more compressed latent representations than a standard VAE and that it can use fewer autoregressive layers than PixelCNN. We established a new state-of-the-art on binarized MNIST dataset in terms of likelihood on $64 \times 64$ ImageNet and demonstrated that our model generates high-quality samples on LSUN bedrooms.

The ability of PixelVAE to learn compressed representations in its latent variables by ignoring the small-scale structure in images is potentially very useful for downstream tasks. It would be interesting to further explore our model's capabilities for semi-supervised classification and representation learning in future work.

## Acknowledgments

The authors would like to thank the developers of Theano (Theano Development Team, 2016) and Blocks and Fuel (van Merriënboer et al., 2015). We acknowledge the support of the following agencies for research funding and computing support: Ubisoft, Nuance Foundation, NSERC, Calcul Quebec, Compute Canada, CIFAR, MEC Project TRA2014-57088-C2-1-R, SGR project 2014-SGR-1506 and TECNIOspring-FP7-ACCI grant.

## References

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. 2016.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

Kenneth P. Burnham and David R. Anderson. Model selection and multi-model inference, 2nd ed. *A Practical information-theoretic approach. Springer-Verlag*, pp. 78, 2003.

Xi Chen, Diederik P Kingma, Tim Salimans, Yan Duan, Prafulla Dhariwal, John Schulman, Ilya Sutskever, and Pieter Abbeel. Variational Lossy Autoencoder. *arXiv.org*, November 2016.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using Real NVP. *arXiv.org*, May 2016.

Jeff Donahue, Philipp Krähenbühl, and Trevor Darrell. Adversarial feature learning. *CoRR*, abs/1605.09782, 2016. URL http://arxiv.org/abs/1605.09782.

Vincent Dumoulin, Ishmael Belghazi, Ben Poole, Alex Lamb, Martin Arjovsky, Olivier Mastropietro, and Aaron Courville. Adversarially learned inference. *CoRR*, abs/1606.00704, 2016.

Matthieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. *CoRR*, abs/1502.03509, 2015. URL https://arxiv.org/abs/1502.03509.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards Conceptual Compression. *arXiv.org*, April 2016.

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

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016.

Yann Lecun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2015.

Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *International Conference on Machine Learning (ICML)*, 2015.

Jason Tyler Rolfe. Discrete variational autoencoders. *arXiv preprint arXiv:1609.02200*, 2016.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *In Proceedings of the 25th international conference on Machine learning*, 2008.

Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder Variational Autoencoders. *arXiv.org*, February 2016.

Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

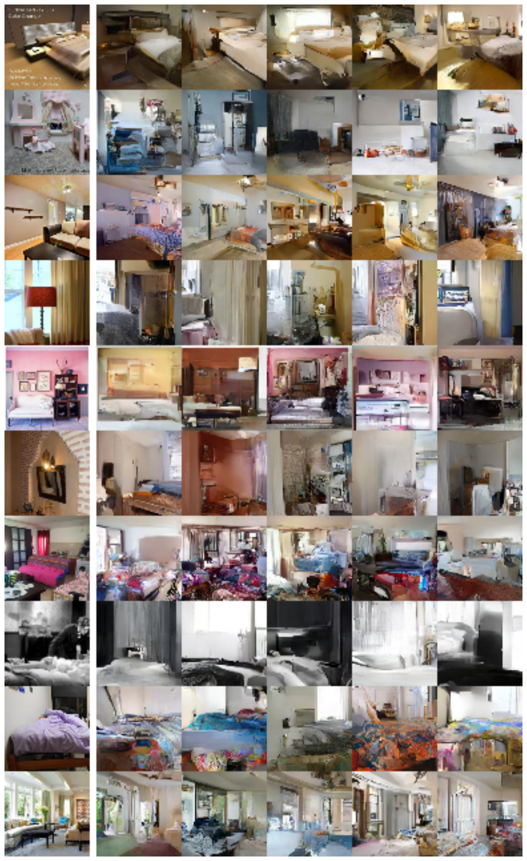
Aäron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning (ICML)*, 2016a.

Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. *CoRR*, abs/1606.05328, 2016b. URL `http://arxiv.org/abs/1606.05328`.

Bart van Merriënboer, Dzmitry Bahdanau, Vincent Dumoulin, Dmitriy Serdyuk, David Warde-Farley, Jan Chorowski, and Yoshua Bengio. Blocks and fuel: Frameworks for deep learning. *arXiv preprint*, abs/1506.00619, 2015. URL `http://arxiv.org/abs/1506.00619`.

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.

## A LSUN BEDROOMS AND 64×64 IMAGENET RECONSTRUCTIONS


(a)

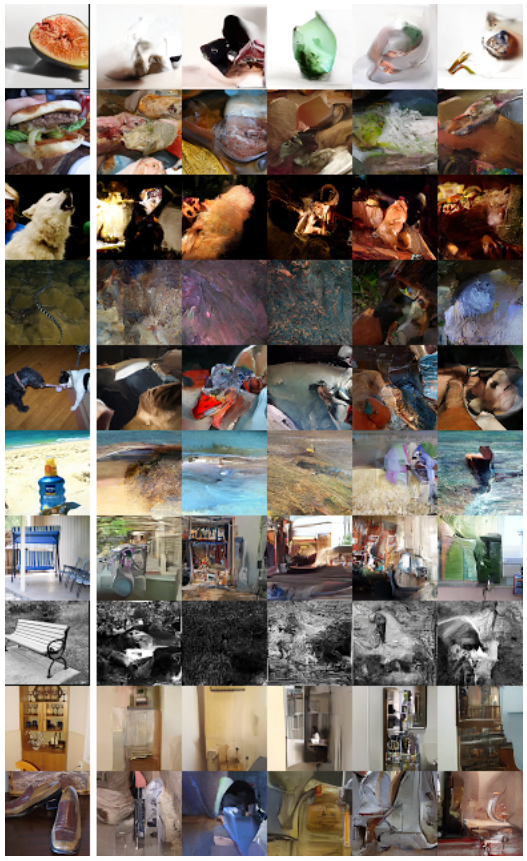

(b)

Figure 8: Reconstructions for (a) LSUN Bedrooms and (b) 64×64 ImageNet. Left-most columns are images from the test set, and the following 5 columns are top-down generations from the highest level of latent variables. We see that the reconstructions capture high-level semantic properties of the original images while varying in most of the details. We also visualized similar reconstructions by generations from the lower level of latent variables, and in this case the reconstructions were visually indistinguishable from the original images.

## B MNIST SAMPLES

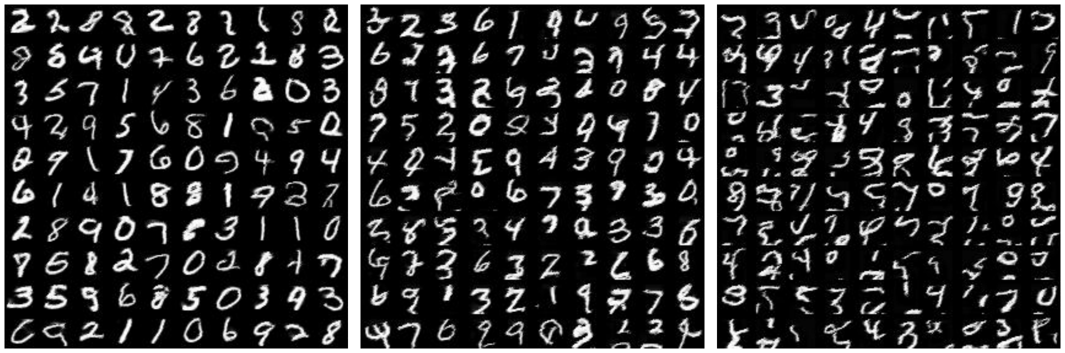


Figure 9: Samples from a PixelVAE with a receptive field of 7 pixels (left), a PixelCNN with an 11-pixel receptive field (middle; roughly the same computational complexity as the PixelVAE), and a PixelCNN with a 7-pixel receptive field (right).

## C MNIST RECONSTRUCTIONS

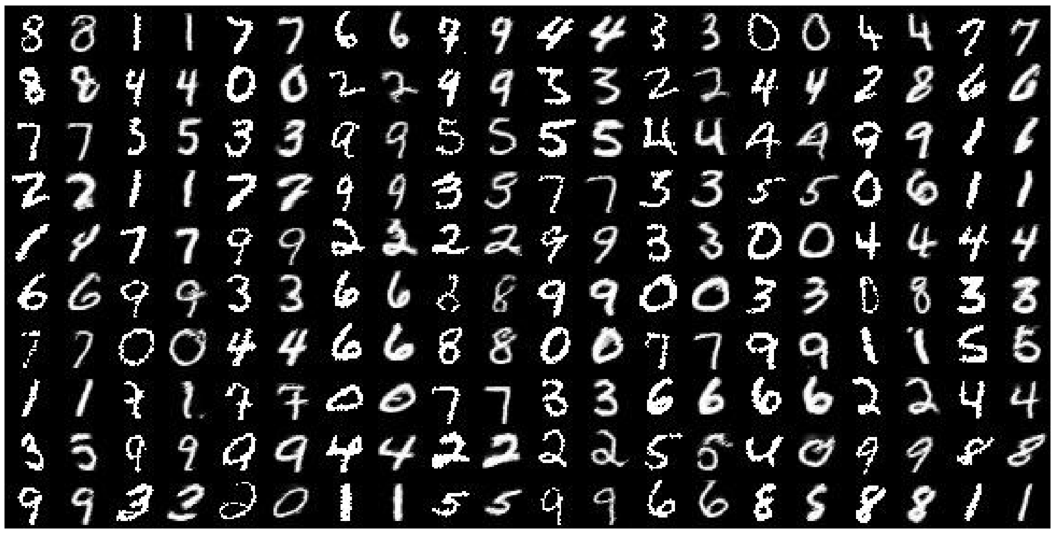


Figure 10: Reconstructions from the MNIST test set. Alternate columns are original (left) and reconstructed images (right).

## D More samples for hierarchical latent space visualizations

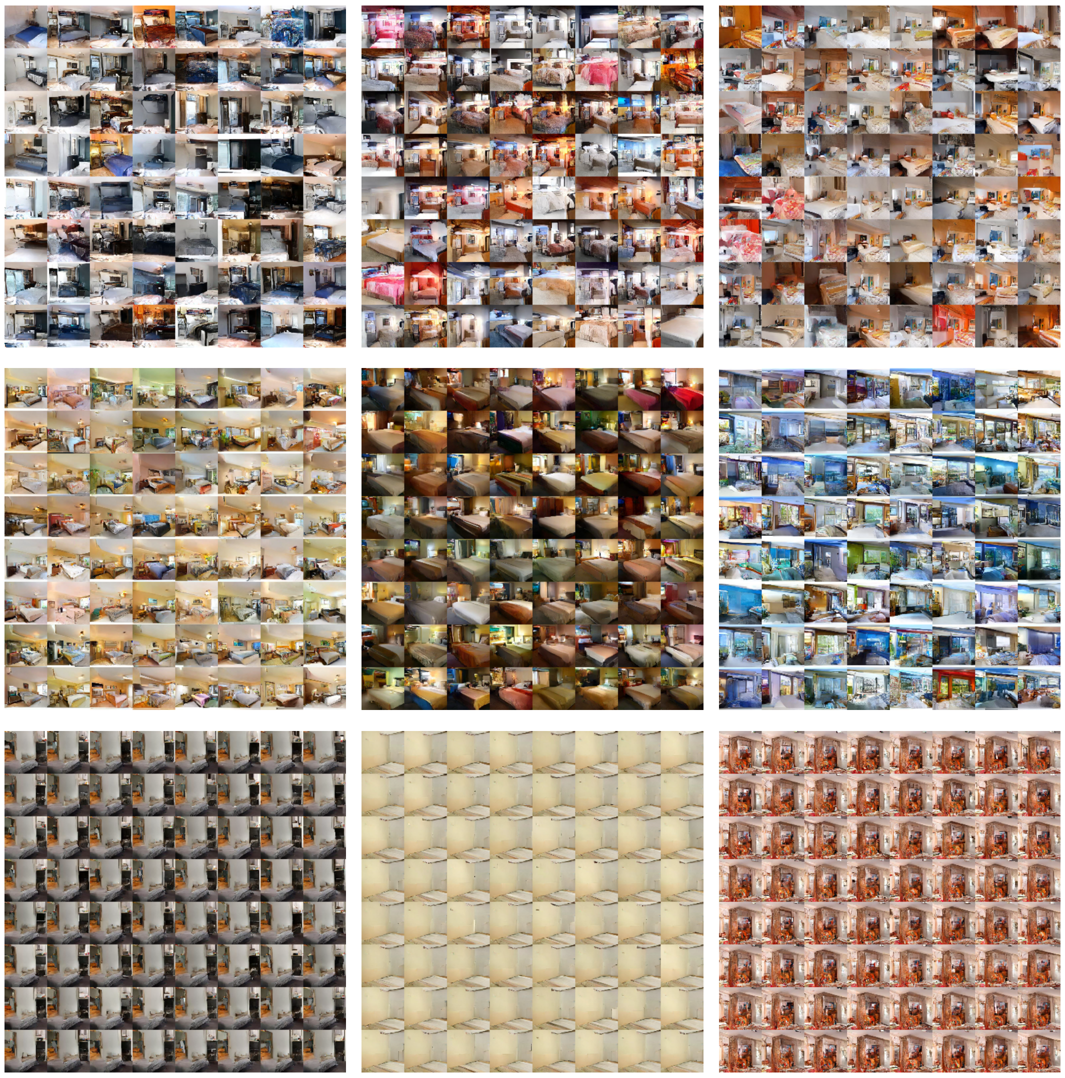


Figure 11: More examples for visualizations of the variation in image features captured at different levels of stochasticity. Holding the other levels constant, we vary only the top-level sampling noise *(top)*, only the middle-level noise *(middle)*, and only the bottom (pixel-level) noise *(bottom)*.

## E Model Architecture

### E.1 MNIST

For our quantitative MNIST experiments, the architectures of our encoder and decoder are as follows. Unless otherwise specified, all convolutional layers use ReLU nonlinearity. We also make an open-source implementation of this model available at `https://github.com/igul222/PixelVAE`.

| **Encoder** $x \to (\mu, \sigma)$ | | | |
|---|---|---|---|
| | Kernel size | Stride | Output channels |
| Convolution | 3x3 | 1 | 32 |
| Convolution | 3x3 | 2 | 32 |
| Convolution | 3x3 | 1 | 32 |
| Convolution | 3x3 | 2 | 64 |
| Pad 7×7 feature maps to 8×8 | | | |
| Convolution | 3x3 | 1 | 64 |
| Convolution | 3x3 | 2 | 64 |
| Convolution | 3x3 | 1 | 64 |
| Convolution | 3x3 | 1 | 64 |
| Convolution | 3x3 | 1 | 64 |
| Flatten | | | |
| Linear | - | - | 2×latent dimensionality |

| **Decoder** $z \to x$ | | | |
|---|---|---|---|
| | Kernel size | Stride | Output channels |
| Linear | - | - | 4×4×64 |
| Reshape to (64, 4, 4) | | | |
| Convolution | 3x3 | 1 | 64 |
| Convolution | 3x3 | 1 | 64 |
| Transposed convolution | 3x3 | 2 | 64 |
| Convolution | 3x3 | 1 | 64 |
| Crop 8×8 feature maps to 7×7 | | | |
| Transposed convolution | 3x3 | 2 | 32 |
| Convolution | 3x3 | 1 | 32 |
| Transposed convolution | 3x3 | 2 | 32 |
| Convolution | 3x3 | 1 | 32 |
| PixelCNN gated residual block | 7x7 | 1 | 32 |
| PixelCNN gated residual block(s) | [ 5x5 ] $\times N$ | 1 | 32 |
| PixelCNN gated convolution | 1x1 | 1 | 32 |
| PixelCNN gated convolution | 1x1 | 1 | 32 |
| Convolution | 1x1 | 1 | 1 |

### E.2 LSUN BEDROOMS AND 64×64 IMAGENET

The LSUN and ImageNet models use the same architecture: all encoders and decoders are residual networks; we use pre-activation residual blocks with two $3 \times 3$ convolutional layers each and ELU nonlinearity. Some residual blocks perform downsampling, using a $2 \times 2$ stride in the second convolutional layer, or upsampling, using subpixel convolution in the first convolutional layer. Weight normalization is used in masked convolutional layers; in all other layers, batch normalization is used. We optimize using Adam with learning rate 1e-3. Training proceeds for 400K iterations using batch size 48.

For further architectural details, please refer to our open-source implementation at `https://github.com/igul222/PixelVAE`.

**Bottom-level Encoder** $x \to h_1$

| | Kernel size | Resample | Output channels |
|---|---|---|---|
| Embedding | - | - | 48 |
| Convolution | 1x1 | - | 192 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 192 |
| Residual block | [ 3x3 ] $\times$ 2 | Down $\times$2 | 256 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 256 |
| Residual block | [ 3x3 ] $\times$ 2 | Down $\times$2 | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |

**Bottom-level Decoder** $z_1 \to x$

| | Kernel size | Resample | Output channels |
|---|---|---|---|
| Convolution | 1x1 | 1 | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | Up $\times$2 | 256 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 256 |
| Residual block | [ 3x3 ] $\times$ 2 | Up $\times$2 | 192 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 192 |
| Embedding | - | - | 48 |
| PixelCNN gated residual block | [ 3x3 ] $\times$ 2 | - | 384 |
| PixelCNN gated residual block | [ 3x3 ] $\times$ 2 | - | 384 |
| PixelCNN gated residual block | [ 3x3 ] $\times$ 2 | - | 384 |

**Top-level Encoder** $h_1 \to h_2$

| | Kernel size | Resample | Output channels |
|---|---|---|---|
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | Down $\times$2 | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | Down $\times$2 | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |
| Residual block | [ 3x3 ] $\times$ 2 | - | 512 |

| **Top-level Decoder** $z_2 \to z_1$ | | | |
|---|---|---|---|
| | Kernel size | Resample | Output channels |
| Linear | - | - | 4×4×512 |
| Reshape to (512, 4, 4) | | | |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| Residual block | [ 3x3 ] × 2 | Up ×2 | 512 |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| Residual block | [ 3x3 ] × 2 | Up ×2 | 512 |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| Residual block | [ 3x3 ] × 2 | - | 512 |
| PixelCNN convolution | 5x5 | - | 512 |
| PixelCNN gated residual block | [ 3x3 ] × 2 | - | 512 |
| PixelCNN gated residual block | [ 3x3 ] × 2 | - | 512 |
| PixelCNN gated residual block | [ 3x3 ] × 2 | - | 512 |
| Convolution | 1x1 | - | 256 |