PixelMRF
A Deep Markov Random Field for Image Generation

Joshua Frederick
Departments of Computer Science & Mathematics
California Polytechnic State University
San Luis Obispo, California, USA
jmfreder@calpoly.edu

Jonathan Ventura
Department of Computer Science
California Polytechnic State University
San Luis Obispo, California, USA
jventu09@calpoly.edu

Abstract

In this work we introduce a new graphical model for generative image modeling. The popular PixelCNN has utilized deep graphical models to represent images as directed, acyclic Markov chains. The limitations enforced by the directed, acyclic nature of PixelCNN have restricted the model to conceptualize an image as a sequence of directional dependencies, where each pixel is conditionally dependent on the pixels before it in row-major order. We instead propose PixelMRF, a deep graphical model that fully captures the conditional distributions for each pixel of an image. PixelMRF achieves this goal by representing the pixels in an image as a undirected Markov random field, and so allows the dependencies that PixelCNN lacks.

Introduction

The need for powerful generative modeling is clear from consistent growth in area. One such highly used model is PixelCNN [van den Oord, Kalchbrenner, and Kavukcuoglu, 2016]. The autoregressive network, PixelCNN is a convolutional neural network (CNN) that represents the joint distribution of an image X as the product of conditional distributions

$$p(X) = \prod_{i=1}^{n^2} p(x_i | x_1, \ldots, x_{i-1})$$

where x_i is the ith pixel in row major order and X is a $n \times n$ image (see Figure 1 for a visual representation).

The network utilizes a series of masked convolutional layers with residual connections to guarantee that each pixel is only dependent on the prior pixels. The mask is implemented by taking the Hadamard product between each convolution kernel and a matrix that is 1 in each position prior to the central position and 0 otherwise, an example is given in Figure 2 [van den Oord, Kalchbrenner, and Kavukcuoglu, 2016].

Networks with the structure of purposely obscuring information from the receptive field of each feature are commonly denoted as *blind-spot networks*. To our knowledge, this terminology was constructed in [Krull, Buchholz, and Jug, 2018], and expanded in [Laine, Lehtinen, and Aila, 2019]. Similarly to [Laine, Lehtinen, and Aila, 2019], we adopt this terminology but note that the term "blind-spot" has a contradictory meaning in PixelCNN literature, where it denotes an unintentional obscuring of the receptive field. For the sake of concreteness of the blind-spot concept, note that the blind spot for a pixel x_i in PixelCNN is all posterior pixels of x_i and x_i itself. We question whether this blind-spot formulation given by PixelCNN is limiting its effectiveness, and so through this work suggest a reconsideration that we believe will achieve better performance.

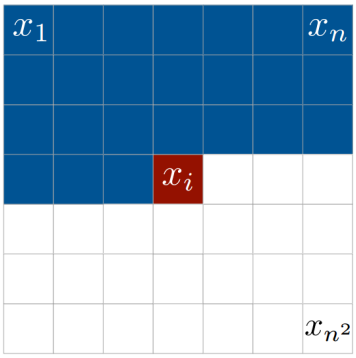

Figure 1: A visual representation of the dependencies of pixel x_i [Harshit Sharma, 2017]

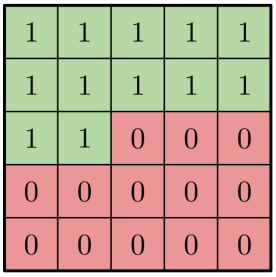

Figure 2: An example of a mask used in PixelCNN [van den Oord et al., 2016b]

Previous Works

Before continuing, we consider some prior improvements made to PixelCNN. Soon after its inception in 2016, two improvements to PixelCNN were suggested. An extension of the team that developed the original model improved their orignal architecture to the Gated PixelCNN model [van den Oord et al., 2016b] by adding gated activation such as in a LSTM. Then a series of improvements was suggested in [Salimans et al., 2017], ultimately leading to the current PixelCNN++.

One important attribute of PixelCNN is that it formally optimizes the log-likelihood of the training data unlike other forms of generative modeling such as GANs. Prior to PixelCNN, other generative models sought to generate images by optimizing the log-likelihood [Theis and Bethge, 2015; van den Oord and Schrauwen, 2014; Dinh, Krueger, and Bengio, 2015].

After PixelCNN in 2018, a Berkeley team used self-attention together with the traditional convolutional networks of PixelCNN to achieve state-of-the-art results on a number of standard data sets [Chen et al., 2018]. Most recently an OpenAI team took the "attention is all you need" philosophy of [Vaswani et al., 2017] and used sparse transformers to achieve the current state-of-the-art performance on several data sets [Child et al., 2019]. On the topic of Markov random fields in image generation, work in [Wu, Lin, and Tang, 2016] explored the utilization of Markov random fields for local image modeling and super-resolution.

Motivation

The chain rule for probability motivates and validates PixelCNN's representation of the joint distribution of an image. However, the representation leads to certain restriction. For an individual pixel x_i, PixelCNN restricts x_i to be viewed as conditionally dependent on only the pixels prior to itself, and so fails to represent the underlying conditional distribution of each x_i. Indeed, to fully express the dependencies of x_i we would desire to model each pixel distribution as

$$p(x_i|x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_{n^2}) := p(x_i|x_{-i})$$

and so model the joint distribution of the image as

$$p(X) = \frac{1}{Z} \cdot \tilde{p}(X) = \frac{1}{Z} \prod_{i=1}^{n^2} p(x_i|x_{-i})$$

where $\tilde{p}(x)$ is the pseudo-likelihood and Z is a normalizing constant.

Implementation

The representation of $p(X)$ presented previously is that of a Markov random field (MRF). Indeed, this realization provides the motivation for the name PixelMRF. We follow the following explicit implementation. Jointly, use two PixelCNNs to represent the conditional distribution of x_i, one model for the pixels prior to x_i in X and another reversed PixelCNN to represent the dependencies of x_i on later pixels in X. This joint model is visually represented in Figure 3. We additionally notice that this architecture is a blind-spot

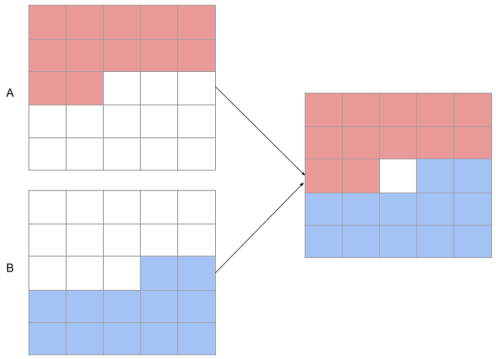

Figure 3: A visual representation of the receptive field of two PixelCNNs and the resulting receptive field of PixelMRF, where two PixelCNNs are working in unison.

network where the receptive field does not contain the center pixel. As such, we consider that for the sake of further exploration we could also consider the architecture presented in [Laine, Lehtinen, and Aila, 2019]. One issue with this representation is that computing the normalizing constant Z is intractable. As noted in [Goodfellow, Bengio, and Courville, 2016], one could represent the distribution using the pseudo-likelihood accepting that

$$p(X) \approx \prod_{i=1}^{n^2} p(x_i|x_{-i})$$

essentially ignoring the normalizing constant. Instead, we use the Markov chain Monte Carlo (MCMC) methods of contrastive divergence to confront the intractability of Z.

Contrastive Divergence

To motivate the process of contrastive divergence, notice that

$$\begin{aligned} \log p(X) &= \log\left(\frac{1}{Z} \cdot \tilde{p}(X)\right) \\ &= \log \tilde{p}(X) - \log Z \\ &= \log \tilde{p}(X) - \log \sum_{X'} \tilde{p}(X'). \end{aligned} \tag{0}$$

Thus to maximize the log-probability of our training data, we can maximize the pseudo-log-likelihood the training data while minimizing the weight of the total distribution

$$\sum_{X'} \tilde{p}(X').$$

Notice that this negative term is the only difference from simply minimizing the pseudo-log-likelihood. The process to approximate the objective given in equation (0) optimization is contrastive divergence, which we now state in general terms:

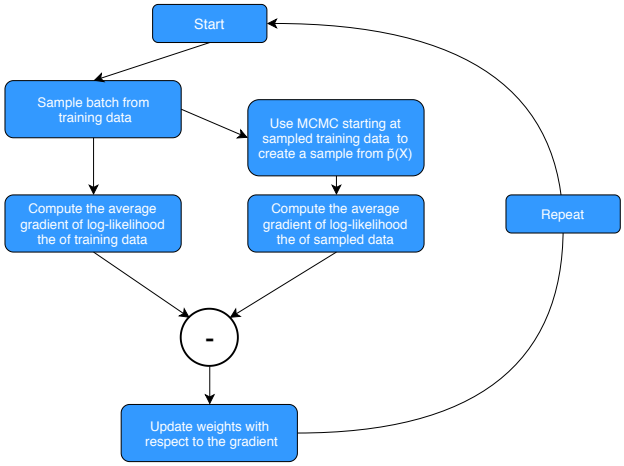

Figure 4: A visual representation of the contrastive divergence algorithm

Algorithm 1 Contrastive Divergence

1: **for** $k \leftarrow 1$ **to** maximum of iterations **do**
2: Sample batch of "real" data X from training data
3: Sample batch of "fake" data X' from model
4: Compute mean gradient ∇ for log-likelihood of X
5: Compute mean gradient ∇' for log-likelihood of X'
6: Update weight with respect to $\nabla - \nabla'$
7: **end for**

In Figure 4, we give a visual representation of the contrastive divergence algorithm that we believe better displays the process. After attempting to train PixelMRF by optimizing the object given in equation (0), we relaxed the objective to

$$\log \tilde{p}(X) - \alpha \log \sum_{X'} \tilde{p}(X') \tag{1}$$

where

$$0 \leq \alpha \leq 1.$$

In the context of equation (1), the variable α can be seen as a term to compromise between optimizing the psuedo-loglikelihood and the contrastive divergence objective. To see why this change was made, note in figures 12 and 13 the instability and subsequent inability to train PixelMRF using the objective given by equation (0).

Hamiltonian Monte Carlo

For the "fake" sampling portion of contrastive divergence (see line 3), we use the Hamiltonian dynamic based sampling method discussed in [Neal, 2010]. Hamiltonian Monte Carlo sampling utilizes the gradient of the log-likelihood to take informed steps, and so allows a shorter number of steps for a fully converged sample. Additionally, we note that for further consideration, future work could consider other state-of-the-art MCMC methods such as the non-trivial improvements to Hamiltonian MC: DeepHMC [Levy, Sohl-dickstein, and Hoffman, 2018], NUTS (No U-Turn Sampler)

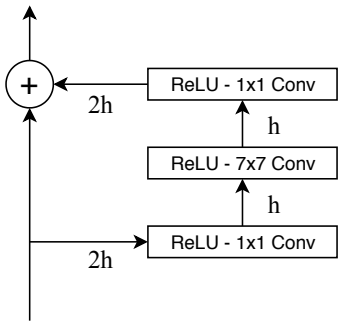

Figure 5: A layer in a PixelCNN. A total of 16 such layers were used in each PixelCNN.

[Homan and Gelman, 2014], or the replica exchange (parallel tempering). In a later section we discuss why exploring different sampling methods may be needed, but the need can be seen in the poor quality of the samples in Figure 11.

Model Configuration

Our model architecture closely follows the model implementation given in [van den Oord, Kalchbrenner, and Kavukcuoglu, 2016]. Of course our implementation doubles that of PixelCNN, as PixelMRF contains two PixelCNNs working in unison. For each PixelCNN, all but the first layer and the final layer take the form given in Figure 5, where $h = 32$ and $2h = 64$ are the number the filters and the convolution is appropriately masked. The first layer of the PixelCNNs is a simple masked convolutional layer that additionally masks the center pixel to obscure it from the receptive field. Due to the everywhere-differentiable regularity requirements of HMC, unlike PixelCNN and later models, we enforce a Gaussian prior on PixelMRF. As such, the last layers are convolutional layers to correctly shape the output to produce a mean and standard deviation for each dimension in the input image.

For the configuration of Hamiltonian Monte Carlo we used 2 leapfrog integrator steps and an adaptive step size with an initial value of 0.1. Additionally due to computational limitations, we had only 10 burnin steps, and so to handle a zero HMC acceptance rate, noise sampled from $\mathcal{N}(0, 0.05)$ was added to the "fake" data. Moreover, we used a non-standard training scheme. We start by training each PixelCNN in the traditional form as in [van den Oord, Kalchbrenner, and Kavukcuoglu, 2016] for 400 epochs. We follow this by optimizing the pseudo-loglikelihood for 100 epochs and then the full contrastive divergence loss with $\alpha = 10^{-6}$ for another 100 epochs. We additionally tested the PixelSNAIL architecture of [Chen et al., 2018]; however, due to our computational limit this did not lead to a noticeable improvement, and so was not included in the final results.

Evaluation

Finally, on the discussion of evaluating PixelMRF, we evaluated PixelMRF on the Frey data set. To properly evalu-

Figure 6: Sampling from PixelCNN using ancestral sampling.

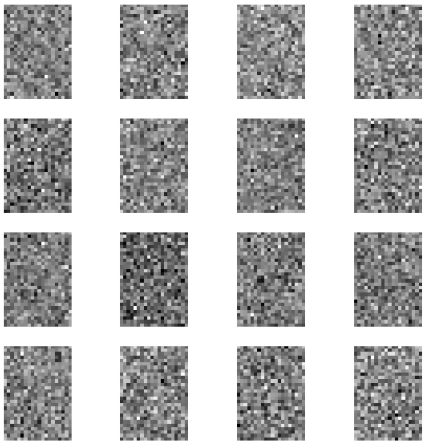

Figure 7: Sampling from PixelCNN using HMC sampling.

ate the performance of or MCMC method, we use the following process: train a PixelCNN in the traditional fashion and evaluate the sampling performance of the the MCMC method for our selection of hyperprameters. This allows us to decouple the evaluation of our contrastive divergence parameters and PixelMRF's architecture from the evaluation of MCMC methods. As mentioned previously, one consequent obstacle we had to overcome is that all of PixelCNN, GatedPixelCNN, and PixelCNN++ model some portion of $p(X)$ as a discrete distribution, and most advanced MCMC methods, such as Hamiltonian MC and NUTS, have everywhere-differentiable regularity requirements. We handled this obstacle by instead enforcing a Gaussian prior on both PixelCNN and PixelMRF when testing MCMC methods.

As both PixelCNN and PixelMRF are attempting to represent the joint distribution $p(X)$, it would be desirable to directly compare their performance. A hurdle to this comparison is realization that due to the intractability of computing Z, computing the exact negative log-likelihood for PixelMRF is impossible. At the current time, we instead use qualitative, visual evaluation of the generated samples. In future work we hope to overcome this limitation, and so we consider the use of annealed importance sampling (AIS) to estimate Z [Neal, 2001], and so compare approximate negative log-likelihood as in [van den Oord, Kalchbrenner, and Kavukcuoglu, 2016].

We now move on to concrete examples of performance on the Frey data set. In Figure 6 and Figure 7, we provide a comparison of images generated by PixelCNN. We have sharp but sometimes distorted images produced by standard ancestral sampling and complete noise produced Hamiltonian Monte Carlo sampling. This absolute failure of Hamiltonian Monte Carlo sampling is not ideal, but the success of the ancestral sampling does confirm that a Gaussian prior can adequately perform on the Frey data set.

With that knowledge we can assume the noise is a result of the Hamiltonian Monte Carlo sampling. In figures 8 and 9, we provide a display of the effect of five steps of HMC

Figure 8: Five steps of the PixelCNN based Hamiltonian Monte Carlo initialized to training examples.

Figure 9: Five steps of the PixelMRF based Hamiltonian Monte Carlo initialized to training examples.

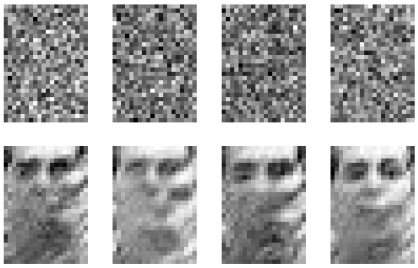

Figure 10: Samples from PixelMRF using the Gibbs sampling procedure. Top: the starting noise. Bottom: the produced sample

sampling initialized at training examples for both PixelCNN and PixelMRF. We see that the HMC process appears to be adding noise to the examples and is failing to step to reasonable, consistent examples. In Figure 11 we see images generated by a trained PixelMRF and Hamiltonian Monte Carlo. Similar to the previous PixelCNN example, these images have no clear relation to the training data and appear to be noise, suggesting that Hamiltonian Monte Carlo is to blame for the noisy generation by PixelMRF. In contrast, in Figure 10 we have samples from PixelMRF produced by the Gibbs sampling procedure. These images are lower in quality than the PixelCNN samples, but have clear learned structure.

Issues and Improvements

As we discussed previously and can see in Figure 11, PixelMRF with Hamiltonian Monte Carlo sampling currently has extremely poor performance. We have reasonable explanations as to why this may be the case. To begin, in

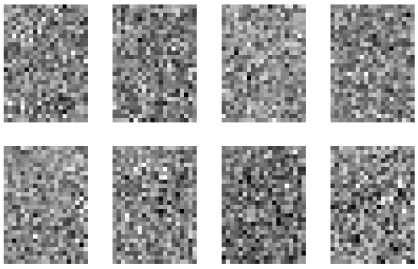

Figure 11: Samples from PixelMRF using the HMC sampling procedure. Top: the starting noise. Bottom: the produced sample

[Neal, 2010], Neal states that the performance of Hamiltonian Monte Carlo decreases heavily in a high dimensional setting. Even in 28x20 grayscale images such as in the Frey data set our image space still has 560 dimensions. This realization could explain a portion of our poor Hamiltonian Monte Carlo performance. This is further supported by the success of Gibbs sampling in Figure 10. This comparison and the eventual zero acceptance rate of HMC suggests that one possible solution would be utilize Gibbs sampling in place of HMC sampling in the training procedure. The main issue with using Gibbs instead HMC is time; Gibbs sampling is sequential and so cannot parallelized like HMC sampling, and so becomes computational unfeasible with large images.

Additionally, the log-likelihood estimate utilized by Hamiltonian Monte Carlo is given by the model. Thus, as the performance of the model on the training data increases, the log-probability distribution becomes flatter at training examples and the gradients of log-probability goes to zero. Consequently, the acceptance rate of Hamiltonian Monte Carlo at the training data goes to zero, and so the difference in gradients given in line 6 goes to zero. Once this occurs, the PixelCNN models underlying PixelMRF fail to learn. In the case of our training, this zeroing of the acceptance rate occurs only a few batches into training. Additionally, the acceptance rate naturally decreases exponentially in the number of dimension, and as discussed previously, image data is necessarily high dimensional [Neal, 2001]. This fact means that even without a trained model we would expect some acceptance rate issues with HMC on our data.

Finally, another issue is the instability of contrastive divergence. Recalling the objective given in (0), we see that contrastive divergence training is inherently adversarial. In cases where the PixelMRF successfully trains, we have noisy loss functions such as in Figure 12. However, more frequently the model chooses to minimize the log-probability of the entire distribution including the training data, instead of correctly maximizing the log-probability of the training data while minimizing the mass of the total distribution. We can see an example of this failed training in Figure 13.

In terms of solutions for these issues, we believe that the

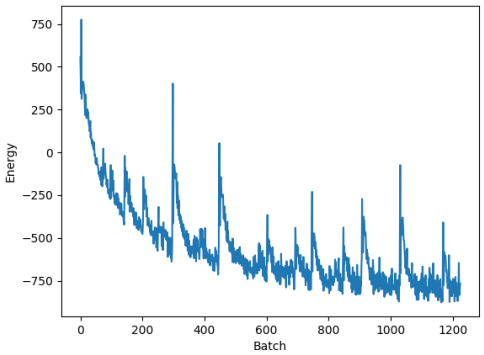

Figure 12: A plot of the loss over time in the early stages of a successful PixelMRF training.

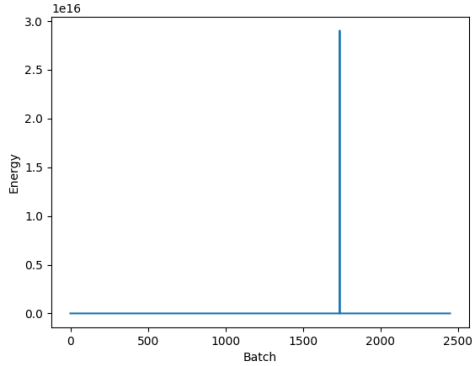

Figure 13: A plot of the loss over time in an unsuccessful PixelMRF training.

MCMC methods discussed previously could solve these issues. DeepHMC parameters Hamiltonian Monte Carlo in the form of a deep network that learns the appropriate parameters to optimize proposals and the acceptance rate for a given data set. Replica exchange Monte Carlo adds a temperature parameter to Hamiltonian Monte Carlo which adds a probability for Hamiltonian Monte Carlo to accept a worse state. This could help solve the issue of the model failing to learn after a small amount of time. Finally, NUTS would provide a fully adaptive sampler no u-turns sampler that would possibly solve all but the problems associated with the dimensionality of our data.

Conclusion

The autoregressive model PixelCNN has been used in many various applications [Kolesnikov and Lampert, 2016][Fauw, Dieleman, and Simonyan, 2019]. Additionally, variants of PixelCNN have been developed for domains such as audio [van den Oord et al., 2016a] and text [Dauphin et al., 2016]. However, PixelCNN has limitation that stem from its acyclic conceptualization of pixel dependencies. Through PixelMRF we represent pixel dependencies completely and confront the resulting normalizing constant through contrastive divergence. At the current time, the performance of PixelMRF is poor. However by exploring the samplers discussed in the document, we believe this novel representation can be made to improve the performance of the already powerful model, PixelCNN.

References

[Chen et al., 2018] Chen, X.; Mishra, N.; Rohaninejad, M.; and Abbeel, P. 2018. Pixelsnail: An improved autoregressive generative model.

[Child et al., 2019] Child, R.; Gray, S.; Radford, A.; and Sutskever, I. 2019. Generating long sequences with sparse transformers. *CoRR* abs/1904.10509.

[Dauphin et al., 2016] Dauphin, Y. N.; Fan, A.; Auli, M.; and Grangier, D. 2016. Language modeling with gated convolutional networks. *CoRR* abs/1612.08083.

[Dinh, Krueger, and Bengio, 2015] Dinh, L.; Krueger, D.; and Bengio, Y. 2015. NICE: non-linear independent components estimation. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings*.

[Fauw, Dieleman, and Simonyan, 2019] Fauw, J. D.; Dieleman, S.; and Simonyan, K. 2019. Hierarchical autoregressive image models with auxiliary decoders. *CoRR* abs/1903.04933.

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

[Harshit Sharma, 2017] Harshit Sharma, S. M. 2017. Auto-regressive generative models (pixelrnn, pixelcnn++). https://towardsdatascience.com/ auto-regressive-generative-models-pixelrnn-pixelcnn-32d192911173. Accessed: 2019-06-07.

[Homan and Gelman, 2014] Homan, M. D., and Gelman, A. 2014. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.* 15(1):1593–1623.

[Kolesnikov and Lampert, 2016] Kolesnikov, A., and Lampert, C. H. 2016. Deep probabilistic modeling of natural images using a pyramid decomposition. *CoRR* abs/1612.08185.

[Krull, Buchholz, and Jug, 2018] Krull, A.; Buchholz, T.; and Jug, F. 2018. Noise2void - learning denoising from single noisy images. *CoRR* abs/1811.10980.

[Laine, Lehtinen, and Aila, 2019] Laine, S.; Lehtinen, J.; and Aila, T. 2019. Self-supervised deep image denoising. *CoRR* abs/1901.10277.

[Levy, Sohl-dickstein, and Hoffman, 2018] Levy, D.; Sohl-dickstein, J.; and Hoffman, M. 2018. Generalizing hamiltonian monte carlo with neural networks.

[Neal, 2001] Neal, R. M. 2001. Annealed importance sampling. *Statistics and Computing* 11(2):125–139.

[Neal, 2010] Neal, R. M. 2010. MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo* 54:113–162.

[Salimans et al., 2017] Salimans, T.; Karpathy, A.; Chen, X.; and Kingma, D. P. 2017. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *CoRR* abs/1701.05517.

[Theis and Bethge, 2015] Theis, L., and Bethge, M. 2015. Generative image modeling using spatial lstms. In Cortes, C.; Lawrence, N. D.; Lee, D. D.; Sugiyama, M.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc. 1927–1935.

[van den Oord and Schrauwen, 2014] van den Oord, A., and Schrauwen, B. 2014. Factoring variations in natural images with deep gaussian mixture models. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *Advances in Neural Information Processing Systems 27*. Curran Associates, Inc. 3518–3526.

[van den Oord et al., 2016a] van den Oord, A.; Dieleman, S.; Zen, H.; Simonyan, K.; Vinyals, O.; Graves, A.; Kalchbrenner, N.; Senior, A. W.; and Kavukcuoglu, K. 2016a. Wavenet: A generative model for raw audio. *CoRR* abs/1609.03499.

[van den Oord et al., 2016b] van den Oord, A.; Kalchbrenner, N.; Vinyals, O.; Espeholt, L.; Graves, A.; and Kavukcuoglu, K. 2016b. Conditional image generation with pixelcnn decoders. *CoRR* abs/1606.05328.

[van den Oord, Kalchbrenner, and Kavukcuoglu, 2016] van den Oord, A.; Kalchbrenner, N.; and Kavukcuoglu, K. 2016. Pixel recurrent neural networks. *CoRR* abs/1601.06759.

[Vaswani et al., 2017] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L. u.; and Polosukhin, I. 2017. Attention is all you need. In Guyon, I.; Luxburg, U. V.; Bengio, S.; Wallach, H.; Fergus, R.; Vishwanathan, S.; and Garnett, R., eds., *Advances in Neural Information Processing Systems 30*. Curran Associates, Inc. 5998–6008.

[Wu, Lin, and Tang, 2016] Wu, Z.; Lin, D.; and Tang, X. 2016. Deep markov random field for image modeling. *CoRR* abs/1609.02036.