Densely Connected PixelCNN

Mateo Espinosa Zarlenga*
Department of Computer Science
Cornell University
Ithaca, NY 14850
me326 at cornell.edu

Aaron Ferber*
Department of Computer Science
Cornell University
Ithaca, NY 14850
amf272 at cornell.edu

Michael Luo*
Department of Computer Science
Cornell University
Ithaca, NY 14850
mrl233 at cornell.edu

Eyvind Niklasson*
Department of Computer Science
Cornell University
Ithaca, NY 14850
een7 at cornell.edu

Abstract

PixelCNNs are a class of powerful generative models that generate images pixel-by-pixel by using tractable likelihood estimations. In this paper we propose a novel PixelCNN variant called DensePixelCNN, which draws inspiration from the Gated PixelCNN architecture as described by van den Oord et al. [14] and the DenseNet architecture as described by Huang et al. [5]. DensePixelCNN combines the pixel modeling approach of Gated PixelCNN with the feature map reuse of DenseNet to efficiently learn the underlying distribution of natural images. Adding these dense inter-layer connections allows our model to be more expressive than a plain Gated PixelCNN while counterintuitively requiring significantly fewer parameters to achieve equal performance. We experiment with this architecture by comparing it to a vanilla implementation of Gated PixelCNN. In our experiments, DensePixelCNN manages to obtain similar or better results to those obtained by Gated PixelCNN, both in terms of the cross-entropy error and the subjective quality of the generated samples, while using fewer parameters and converging faster than the original Gated PixelCNN.

1 Introduction

Learning the distribution of natural images is currently a challenge that demands improvement. In recent years, a significant amount of work has been done to explore more complex approaches for learning this distribution. While models like Generative Adversarial Networks [2], Adversarial Autoencoders [10], and Variational Autoencoders [7] have proven to generate realistic images even under very complicated and diverse distributions, these models tend to have an intractable inference step that make them unpractical in several circumstances [11]. Furthermore, even though these previous models are able to generate samples from the learned distribution, they are not capable of explicitly computing the likelihood of a given image. Ideally, we would prefer a method that could not only generate random images that belong to the manifold of real images, but also discriminate between real and unlikely images by explicitly computing the likelihood of an image. These two complications are the main motivation behind the architectures proposed by van der Oor et al. in [11] and [14].

The PixelCNN and PixelRNN architectures attempt to learn the distribution of real images by learning instead many conditional probability distributions that could be used to obtain the likelihood of

*All authors contributed equally.

an image. To achieve this, these architectures, like the work done in [13], model the conditional distribution of a pixel given the values of the previous pixels. For both of these architectures, we model the conditional distribution of an arbitrary pixel, given the previous pixels, through a deep neural network whose last layer is a softmax layer that returns the probabilities of all 256 possible values that the pixel could take for a given channel. More generally, given a $N \times N \times c$ image where c is the number of channels, our model will return a $N \times N \times c \times 256$ tensor with the conditional distributions of each pixel (i, j) and every channel $k \in \{1, 2, ..., c\}$. Note that this architecture then models the conditional distributions as a discrete multinomial distribution where we assume that every pixel can only have one of 256 values.

The difference between PixelCNN and PixelRNN is clear from their names. While the architecture in PixelRNN approaches this modeling through a series of LSTM layers [4] before the softmax layer, the architecture in PixelCNN takes a computationally cheaper approach of learning these distributions through a series of convolutional layers [8] before reaching the last layer [11]. Even though PixelRNN outperforms PixelCNN, in this paper we will focus on a variant the latter due to its simplicity and tractability. More specifically, we will focus on the Gated PixelCNN variant that was introduced in [14].

The main contributions of this paper can be summarized as follows:

- We describe a novel architecture called DensePixelCNN which outperforms Gated PixelCNN in terms of both the quality of the generated images and the cross-entropy error on the test images.
- We provide and discuss further evidence that densely connected networks tend to outperform their not-densely-connected counterparts while using significantly fewer parameters.

We will now introduce the Gated PixelCNN architecture, and its conditional form [14], together with a brief discussion of DenseNet [5]. Following this, we will describe DensePixelCNN together with the results of our experiments using this architecture.

1.1 Gated PixelCNN

As mentioned above, the PixelCNN architecture attempts to learn the distribution of natural images $p(\mathbf{x})$ by modeling this distribution as a product of several conditional distributions. More explicitly, given a $n \times n$ image $\mathbf{x}$ we will rewrite the probability distribution $p(\mathbf{x})$ as

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i \mid x_1, x_2, ..., x_{i-1}) \tag{1}$$

Where x_i is the i^{th} pixel of image $\mathbf{x}$ in raster scan order starting on the top-left corner: we read each row left-to-right first starting on the top row going towards the bottom row. Intuitively, we want to estimate the conditional probability of pixel x_i given the values of all the pixels "above and to the left" of pixel x_i. The original PixelCNN attempts to simulate this conditioning by using a stack of masked convolutions. This process, however, unavoidably generates a "blind spot" in the receptive field of the pixel that does not allow the network to truly use every pixel before the target pixel as we would want [11]. This is the main motivation for the modifications introduced in the Gated PixelCNN variant.

The first significant difference between the original PixelCNN and the Gated PixelCNN architecture is that the Gated PixelCNN modifies the convolutional layers so that they become multiplicative units (similar to those in LSTM) in order to add expressiveness. In order words, if the convolutional layer k outputs $\mathbf{y} = W_k * \mathbf{x}$ in the original PixelCNN architecture (where W_k are some learnable parameters), the Gated PixelCNN will output the following value:

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x}) \ \odot \ \sigma(W_{k,g} * \mathbf{x}) \tag{2}$$

Where $W_{k,f}$ and $W_{k,g}$ are the parameters we want to learn and $\odot$ indicates entry-wise multiplication.

A second big modification is that Gated PixelCNN has two convolution stacks, one "vertical" stack and one "horizontal" stack. The purpose of having these two stacks is to eliminate the blind spot in the receptive field caused by the masked convolutions in the original PixelCNN. The unmasked vertical stack permits the conditioning on all the rows above the current pixel while the masked

horizontal stack handles the conditioning of our result on the current row. Under this scheme, every layer in the horizontal stack will take as an input the output of the previous horizontal layer and the output of the vertical stack. On the other hand, every layer in the vertical stack will only take as an input the output of the previous layer. This is done so that no information from pixels following our target pixel is leaked into the computation of the conditional probability using these two stacks.

1.2 Conditional Gated PixelCNN

Given some latent encoding vector $\mathbf{h}$ of an image $\mathbf{x}$, one can modify the gated PixelCNN architecture so that it models the distribution $p(\mathbf{x} \mid \mathbf{h})$ [11]. For this, we change the output of the gated convolutions layers described by [14] to take in account the latent encoding as a bias term at every layer. Therefore, in this case the activation of the kth convolutional gated layer is given by

$$\mathbf{y} = \tanh(W_{k,f} * \mathbf{x} \ + \ V_{k,f}^T \mathbf{h}) \ \odot \ \sigma(W_{k,g} * \mathbf{x} \ + \ V_{k,g}^T \mathbf{h}) \tag{3}$$

Where $V_{k,f}$, $V_{k,g}$, $W_{k,f}$ and $W_{k,g}$ are the parameters we want to learn.

Note that in this case the latent encoding must be a description of the content of the image as a whole rather that a description that indicates something about an specific pixel. An example of this conditioning would be to use a one-hot encoding for the class of an image. This would then simply add a class-dependent bias on each layer. While there is a way to modify this activation so that it takes latent encodings that are location-dependent, for the purposes of our experiments we will only use class-dependent latent encodings.

1.3 DenseNet

DenseNet was introduced by Huang et al. as an alternative architecture that circumvents the vanishing gradient problem that is so frequent among very deep neural networks [5]. Several architectures have previously been proposed in order to facilitate the training of very deep networks. Some of these architectures include Residual Networks [3], Highway Networks [12], and stochastic depth networks [6]. Similarly to what those architectures do, DenseNet allows the training of deep networks by facilitating the pass of information between a layer and the layers that follow. In this case, DenseNet does this through a combination of "dense blocks": block of layers where the input of one layer will be concatenated to the inputs of all subsequent layers within the same block. By doing this, DenseNet reduces the amount of information that could get lost when a feature map is fed into a layer in a feed-forward manner. This modification has allowed DenseNet to become the state-of-the-art architecture for image classification on ImageNet while the number of parameters used remains very small compared to other architectures that perform similarly.

2 Densely Connected PixelCNN

In this paper we propose a novel architecture called DensePixelCNN that combines DenseNet's densely connected blocks with the conditional approach taken by Gated PixelCNN in order to model the distribution of natural images by computing the conditional probabilities as in [11]. For this, we form a network where in each layer, similarly to gated PixelCNN, there is both a gated "horizontal" convolutional unit and a gated "vertical" convolutional unit that act like the ones used in Gated PixelCNN. The main difference is that now we divide our network into one or more densely connected blocks where, for any layer L_i of a densely connected block with l layers in total, we concatenate the input of the horizontal convolutional unit of L_i to the input of the horizontal convolutional units of layers $L_{i+1}, L_{i+2}, ..., L_l$. Similarly, we concatenate the input of the vertical convolutional unit of L_i to the vertical convolutional units of all the subsequent layers. Furthermore, in order to facilitate the reuse of information in the network, we add a "residual connection" between the input image and the beginning of each dense block. This means that every layer has the input image as an input feature map, rather than just the layers in the first dense block.

The architecture of a single layer in a dense block is summarized in Figure 1. Furthermore, the high-level architecture of DensePixelCNN can be seen in Figure 2. Note that, differently from DenseNet, between any two densely connected blocks we simply pass the output of one densely connected block as an input to the following densely connected block without any intermediate pooling layers.

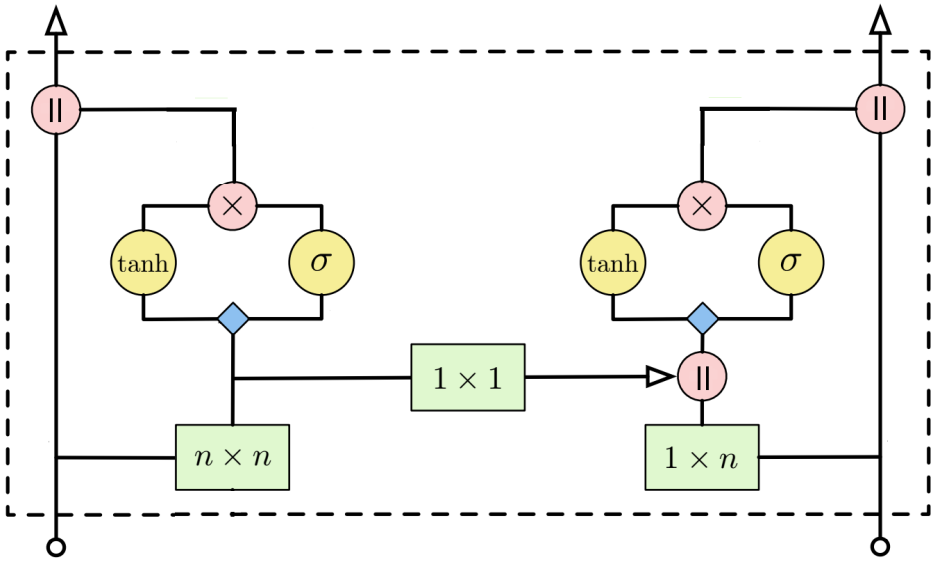

Figure 1: Diagram of the architecture of one layer of a densely connected block in DensePixelCNN. Yellow circles represent non-linearities, green boxes represent convolutions, the pink circle with the two lines represent feature maps concatenation, the pink circle with an "X" represents entry-wise multiplication, and the blue diamond represents a split of the feature maps into two equally sized sets.

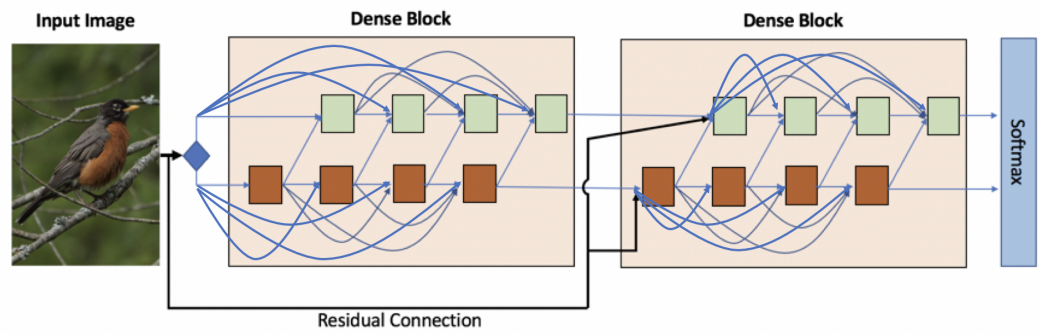

Figure 2: Diagram of the high-level architecture of DensePixelCNN. Blue diamond represents the partition of the image into two copies of itself, the green boxes represent gated units in the horizontal stack, and the orange boxes represent gated units in the vertical stack. Converging arrows represent concatenations.

The architecture described above works similarly to Gated PixelCNN in the sense that it takes as an input a $N \times N \times c$ image and it returns a $N \times N \times c \times 256$ tensor that represent the conditional probabilities of all pixels in all channels.

3 Experimentation

3.1 Setup

For our experiments we compare our architectures with a vanilla implementation of the Gated PixelCNN architecture as proposed by [14]. We train both models on the MNIST hand written digit dataset [9] and compare their convergence times, cross-entropy loss on the training data, and the quality of the generated images during intermediate training steps. We implement the Gated

PixelCNN architecture in TensorFlow [1] by modifying the Anan Gupta's implementation[2] to follow the implementation described in [14] more closely. From there, we implemented DensePixelCNN by modifying our existing implementation so that feature maps are passed between layers as explained above. In our experiments we try different number of dense blocks, layers, and feature map in order to compare the sensitivity of our algorithm with respect to these hyper-parameters. Our full implementation can be found in our github repository[3].

3.2 Quantitative Results

In order to quantitatively determine the benefits of DensePixelCNN over the original Gated PixelCNN architecture, we compare results of each model over varying number of parameters. Specifically, we test three parameter settings, low, medium, and high, in order to compare how effectively the two models reuse parameters at different scales. We utilize the training loss and 95% confidence intervals for the test loss as comparison metrics. In 1, we see that for high parameter settings, DensePixelCNN performs on par with the original Gated PixelCNN as their test loss confidence intervals overlap. However, comparing the models in the low setting, consisting of around 110k parameters, reveals a lower training and test loss for the Densely connected variant. Additionally, the confidence interval for the test loss of DensePixelCNN is lower than that of the original Gated PixelCNN model. This means that in cases where the number of parameters is heavily constrained, DensePixelCNN is able to better utilize the parameters it has available to generate a probabilistic model with lower cross-entropy loss.

model	params	no. layers	no. blocks	no. feature maps	train loss	test loss
Gated PixelCNN	~ 110k	10	N/A	20	0.1111	0.1073 ± 0.0008
DensePixelCNN	~ 110k	9	3	8	0.1104	$\mathbf{0.1042 \pm 0.0008}$
Gated PixelCNN	~ 300k	28	N/A	20	0.0995	0.0992 ± 0.0008
DensePixelCNN	~ 300k	15	5	12	0.0995	0.1004 ± 0.0008
Gated PixelCNN	~ 500k	18	N/A	32	0.0995	0.0992 ± 0.0008
DensePixelCNN	~ 500k	15	5	12	0.0992	0.1005 ± 0.0008

Table 1: Cross-entropy loss on MNIST dataset. We vary the total number of layers, the number of layers in each dense block, and the number of feature maps produced at each layer to ensure that the number of parameters is comparable. We mark in **bold** entries whose confidence intervals do not overlap with other entries in the parameter setting and which do better in terms of loss.

3.3 Qualitative Results

The advantages of Dense PixelCNN become most clear when working with a highly constrained number of parameters. Both models were constrained to roughly ~ 110k parameters and hand-tuned. The below training snapshots in Figure 3 show the sampled images from Gated PixelCNN and Dense PixelCNN at 5, 10 and 20 epochs, respectively. It is clear to see that DensePixelCNN produces more convincing generated images than Gated PixelCNN, when both are given the same tight parameter and training constraints. This supports the idea that feature reusability allows our model to be less dependent on the correct behavior of all layers and thus makes it more reliable than the gated PixelCNN architecture with the same number of training epochs and parameters.

4 Conclusions

Our work presented an improved Gated PixelCNN architecture, called DensePixelCNN. The underlying assumptions and probabilistic model remain the same as in Gated PixelCNN. The inspiration for the architectural changes comes from DenseNets. Applying these dense inter-layer connections allowed for our model, DensePixelCNN, to achieve better performance than the original Gated PixelCNN, but with fewer parameters and faster convergence. Alternatively, when equally size constrained,

[2]https://github.com/anantzoid/Conditional-PixelCNN-decoder

[3]https://github.com/mluop/DensePixelCNN

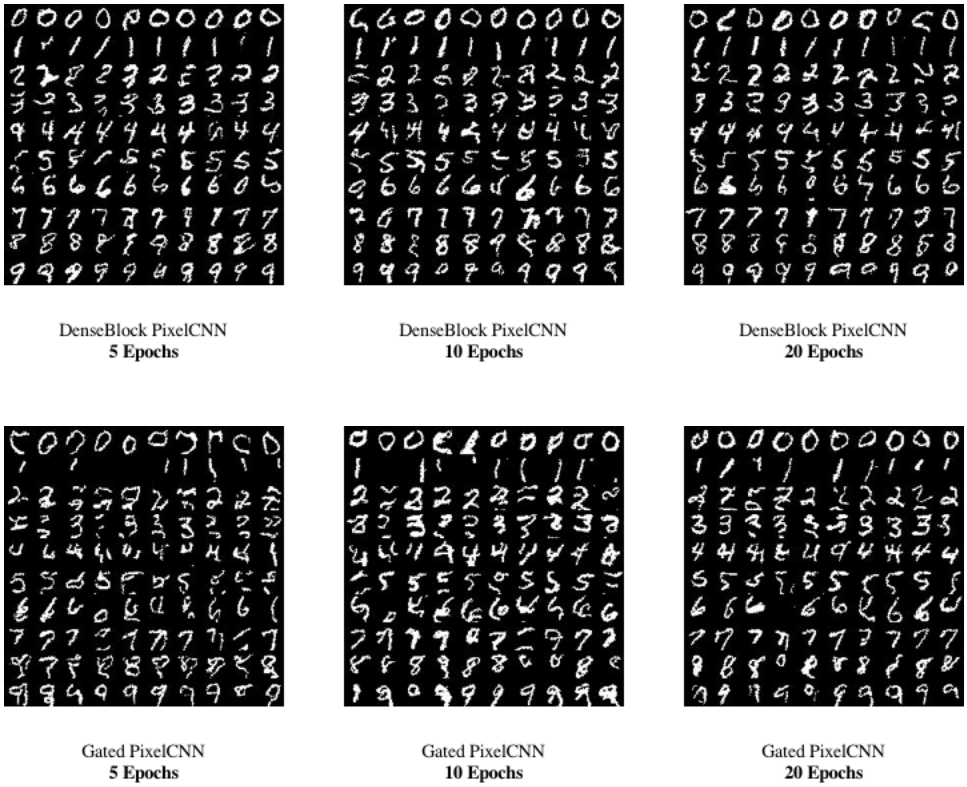

Figure 3: Generated images after 5, 10 and 20 epochs respectively, with 110k parameters.

it provided better performance with the same number of training epochs. The applications which can benefit from reducing parameter count include any device with limited memory attempting to evaluate on the model, or situations in which faster training convergence is necessary. The main contribution of this paper is to show that the advantages of dense inter-layer connections apply to improving Gated PixelCNN.

As an extension of our work, it would be interesting to apply both the DensePixelCNN architecture and the Gated PixelCNN architecture to the challenges of generating convincing CIFAR and ImageNet images, especially in the limited-parameter regime. We did attempt to run these experiments ourselves, but could not fit a sufficiently expressive model into memory. If one has resources equivalent to 60 hours of compute time on 32 GPUs as in [14], we highly recommend attempting this experiment. We suspect that the improvements in convergence time, parameter efficiency, and test error achieved by DensePixelCNN will become especially clear on these datasets, due to the complexity of their images.

Acknowledgments

We want to thank Kilian Weinberger, Yu Sun, and Chuan Guo for giving us access to a GPU cluster for training and for all their support and help in this project.

References

[1] Martın Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. *Software available from tensorflow. org*, 1, 2015.

[2] 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*, pages 2672–2680, 2014.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[5] Gao Huang, Zhuang Liu, and Kilian Q Weinberger. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

[6] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth. *arXiv preprint arXiv:1603.09382*, 2016.

[7] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[8] Yann LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[9] Yann LeCun, Corinna Cortes, and Christopher JC Burges. The mnist database of handwritten digits, 1998.

[10] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.

[11] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[12] Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In *Advances in neural information processing systems*, pages 2377–2385, 2015.

[13] Lucas Theis and Matthias Bethge. Generative image modeling using spatial lstms. In *Advances in Neural Information Processing Systems*, pages 1927–1935, 2015.

[14] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances In Neural Information Processing Systems*, pages 4790–4798, 2016.