Exploring Generative Image Modeling with Block PixelCNNs

Ekaterina Iakovleva
Thesis supervisor: Jakob Verbeek

Grenoble INP

June 28, 2018

Generative Image Modeling

- Generative Image Modeling is one of the central problems in modern unsupervised learning.
- Potential applications include a wide range of tasks: super-resolution, inpainting, pure image generation, deblurring, compression etc.
- Pros: unlimited pool of data.
- Cons: modelled objects are highly structured and very high-dimensional.

Figure 1: Examples of image completion task. Figure from [Oord et al., 2016b].

Generative Image Modeling

Main approaches:

- Tractable likelihood models
  (Deep GMM, NICE, Real NVP, NADE, MADE, PixelCNN etc.)
- Variational models
  (VAE, NF, IAF, VLAE, PixelVAE, ConvDRAW, Deep Diffusion)
- Generative Adversarial Networks (GANs)

PixelCNN [Oord et al., 2016b]

- Joint pixel distribution is factorized into a product of nested univariate conditionals:

$$p(\mathbf{x}) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<d}), \quad \mathbf{x}_{<d} = [x_1, \ldots, x_{d-1}]^T. \tag{1}$$

- Dependencies are modelled in a raster scan order (Figure 2): each pixel depends on all pixels up and to the left of it.

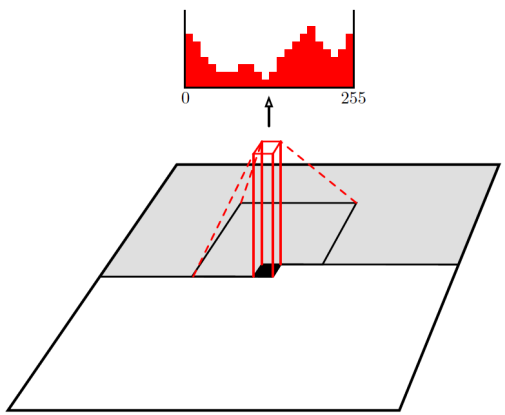


Figure 2: A visualization of the context for image pixels. Figure from [Oord et al., 2016a].

PixelCNN [Oord et al., 2016b]

- Each conditional is factorized into another product over RGB channels:

$$p(x_d|\mathbf{x}_{<d}) = p(x_{d,R}|\mathbf{x}_{<d}) \cdot p(x_{d,G}|\mathbf{x}_{<d}, x_{d,R}) \cdot p(x_{d,B}|\mathbf{x}_{<d}, x_{d,R}, x_{d,G}). \tag{2}$$

- Each pixel is modelled as a discrete variable that takes one of 256 values with the use of softmax.
- During training and evaluation steps pixel distributions are computed in parallel.
- Image generation is a sequential process: each generated pixel is put back into the network as input to produce the next pixel.

PixelCNN [Oord et al., 2016b]

- PixelCNN neural network architecture consists of multiple convolutional layers that preserve spatial resolution.
- To preserve autoregressive property and prevent each pixel from seeing future pixels, each convolutional filter is multiplied by the mask (Figure 3).
- Autoregressive connectivity between colour channels RGB is modelled by splitting feature maps at each layer into three parts and adjusting the centre values of the masked convolutional weights.

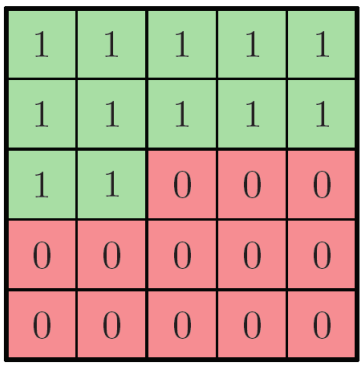


Figure 3: An example of mask used for the 5x5 convolutional kernel. Figure from [Oord et al., 2016a].

Gated PixelCNN and Conditional PixelCNN [Oord et al., 2016a]

- The problem with the blind spots in the receptive field is addressed. The solution is to use two convolutional stacks as in Figure 4.

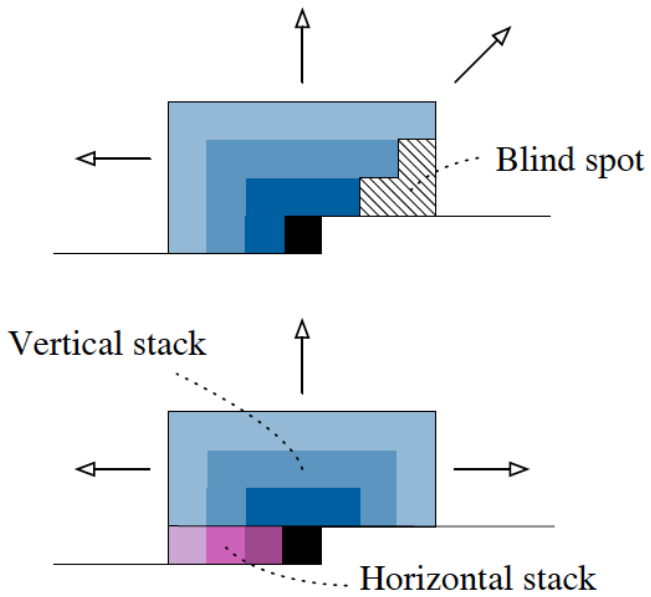


Figure 4: Top: Blind spots in the receptive field of PixelCNNs. Bottom: Two convolutional stacks. Figure from [Oord et al., 2016a].

Gated PixelCNN and Conditional PixelCNN [Oord et al., 2016a]

- *Gated activation units* are introduced and placed after each convolutional layer in the Gated PixelCNN:

$$y = \tanh\left(W_f * x\right) \odot \sigma\left(W_g * x\right). \tag{3}$$

- *Conditional PixelCNN* uses high-level information about the image coded as a latent vector $\mathbf{h}$:

$$p(\mathbf{x}|\mathbf{h}) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<d}, \mathbf{h}), \quad \mathbf{x}_{<d} = [x_1, \ldots, x_{d-1}]^T. \tag{4}$$

- Conditional distribution is modelled by adding $\mathbf{h}$ to the gated activation units:

$$y = \tanh\left(W_f * x + V_f^T \mathbf{h}\right) \odot \sigma\left(W_g * x + V_g^T \mathbf{h}\right). \tag{5}$$

PixelCNN++ [Salimans et al., 2017]

- Univariate conditionals are modelled using a mixture of logistic distributions instead of 256-way softmax:

$$p(x|\pi,\mu,s) = \sum_{k=1}^{K} \pi_k \left[\sigma((x+0.5-\mu_k)/s_k) - \sigma((x-0.5-\mu_k)/s_k)\right]. \tag{6}$$

- Dependencies between colour sub-pixels are modelled as parameterized distributions instead of masked convolutions:

$$\begin{aligned} p(r_d, g_d, b_d|\mathbf{x}_{<d}) =& p(r_d|\mu_r(\mathbf{x}_{<d}), s_r(\mathbf{x}_{<d})) \\ & \times p(g_d|\mu_g(r_d, \mathbf{x}_{<d}), s_g(r_d, \mathbf{x}_{<d})) \\ & \times p(b_d|\mu_b(r_d, g_d, \mathbf{x}_{<d}), s_b(r_d, g_d, \mathbf{x}_{<d})), \qquad (7) \\ \mu_g(r_d, \mathbf{x}_{<d}) =& \mu_g(\mathbf{x}_{<d}) + \alpha(\mathbf{x}_{<d}) \cdot r_d, \\ \mu_b(r_d, g_d, \mathbf{x}_{<d}) =& \mu_b(\mathbf{x}_{<d}) + \beta(\mathbf{x}_{<d}) \cdot r_d + \gamma(\mathbf{x}_{<d}) \cdot g_d. \end{aligned}$$

PixelCNN++ [Salimans et al., 2017]

- Spatial resolution preserving convolutions are substituted for strided convolutions followed by transposed convolutions which results in a U-shaped "downsampling-upsampling" neural network architecture.
- The neural network architecture is structured into ResNet blocks with gated activation units after them.
- Additional skip connections between blocks of the same spatial resolution before and after the bottleneck are added.

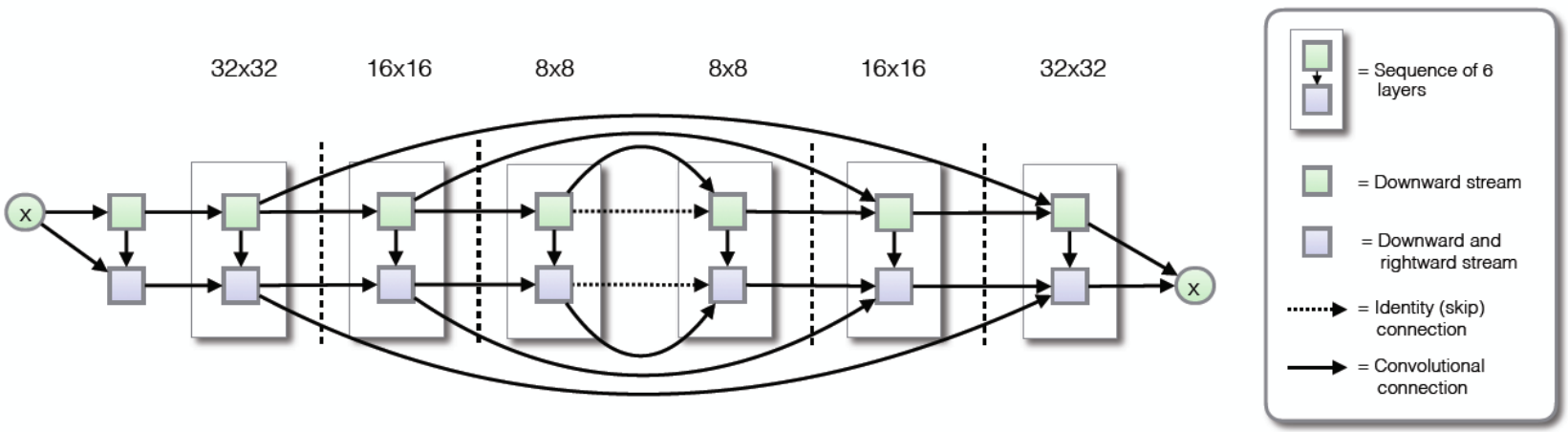


Figure 5: PixelCNN++ architecture. Figure from [Salimans et al., 2017].

Multiscale PixelCNN [Reed et al., 2017]

- Joint pixel distribution is factorized into G equal factors of T pixels:

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p(x_{1:T}^{(g)} | x_{1:T}^{(1:g-1)}). \tag{8}$$

- Pixels within each group depend only on the pixels of the previous groups and can now be generated in parallel. It results in $O(\log N)$ sampling complexity compared to $O(\log N)$ in the original PixelCNN.
- In the paper $G = 4$. The group structure is obtained by tiling the image with 2×2 blocks as in Figure 6.

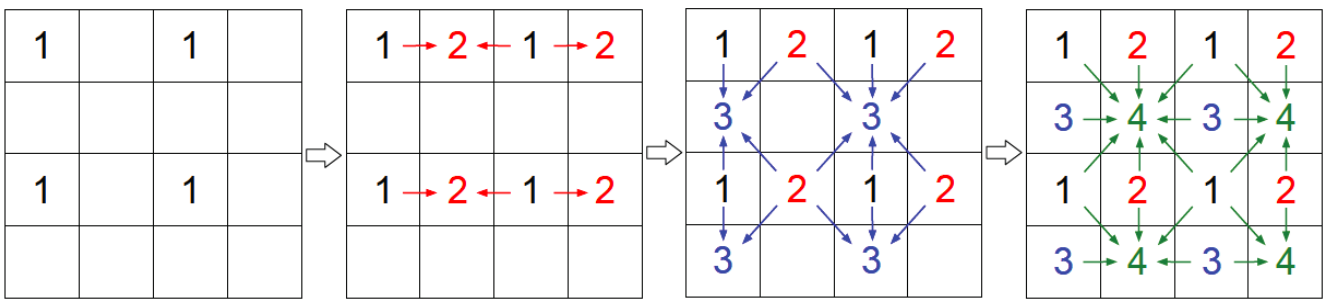


Figure 6: Example pixel grouping and ordering for a 4×4 image. Figure from [Reed et al., 2017].

Multiscale PixelCNN [Reed et al., 2017]

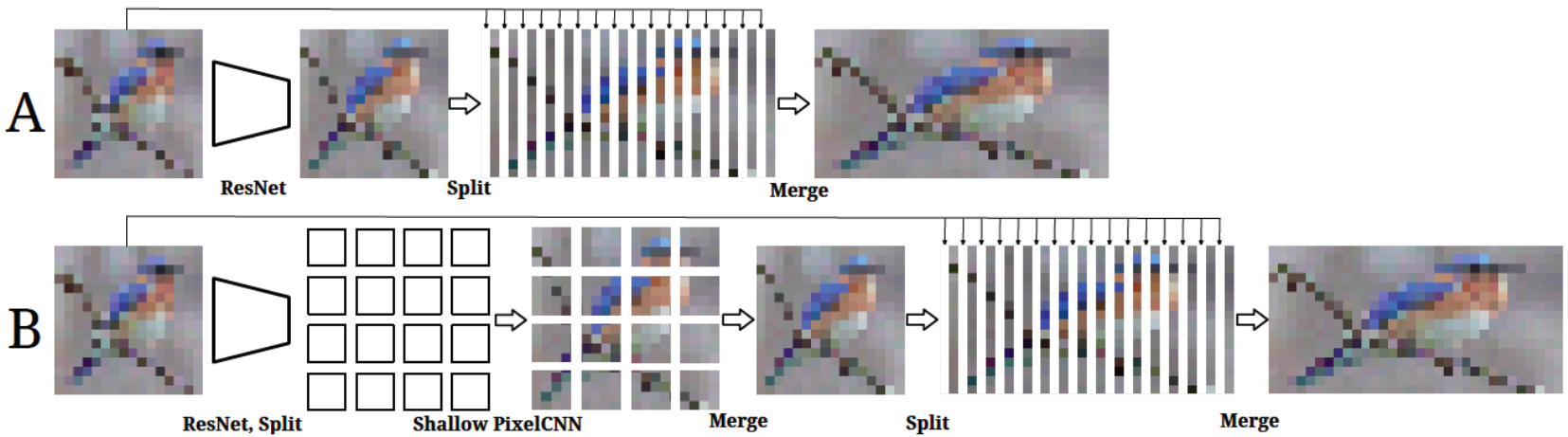


Figure 7: A simple form of causal upscaling network, mapping from a $K \times K$ image to $K \times 2K$. Figure from [Reed et al., 2017].

Block PixelCNN

Similarly to Multiscale PixelCNN, we consider factorization of joint pixel distibution into G disjoint factors of size T:

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p(x_{1:T}^{(g)} | x_{1:T}^{(1:g-1)}). \tag{9}$$

Pixels of the same group are modelled as conditionally independent given context from the previous groups. Each univariate conditional is modelled using a mixture of logistics, similarly to PixelCNN++ model:

$$p(x|\pi, \mu, s) = \sum_{k=1}^{K} \pi_k \left[\sigma((x + 0.5 - \mu_k)/s_k) - \sigma((x - 0.5 - \mu_k)/s_k)\right], \tag{10}$$

where π_k, μ_k, s_k, $k = \overline{1, K}$ are the outputs of the neural network.

Block PixelCNN

Dependencies between color channels are modelled in the same way as in the PixelCNN++. Parameters of the mixture for a t-th pixel from group g depend on the context from all the pixels $\mathbf{x}_{1:T}$ of the previous groups $(1 : g-1)$:

$$
\begin{aligned}
p(r_t^{(g)}, g_t^{(g)}, b_t^{(g)} | \mathbf{x}_{1:T}^{(1:g-1)}) =& p(r_t^{(g)} | \mu_r(\mathbf{x}_{1:T}^{(1:g-1)}), s_r(\mathbf{x}_{1:T}^{(1:g-1)})) \\
& \times p(g_t^{(g)} | \mu_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}), s_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)})) \\
& \times p(b_t^{(g)} | \mu_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}), s_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)})), \\
\mu_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}) =& \mu_g(\mathbf{x}_{1:T}^{(1:g-1)}) + \alpha(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot r_t^{(g)}, \\
\mu_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}) =& \mu_b(\mathbf{x}_{1:T}^{(1:g-1)}) + \beta(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot r_t^{(g)} + \gamma(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot g_t^{(g)}.
\end{aligned}
\tag{11}
$$

Block PixelCNN

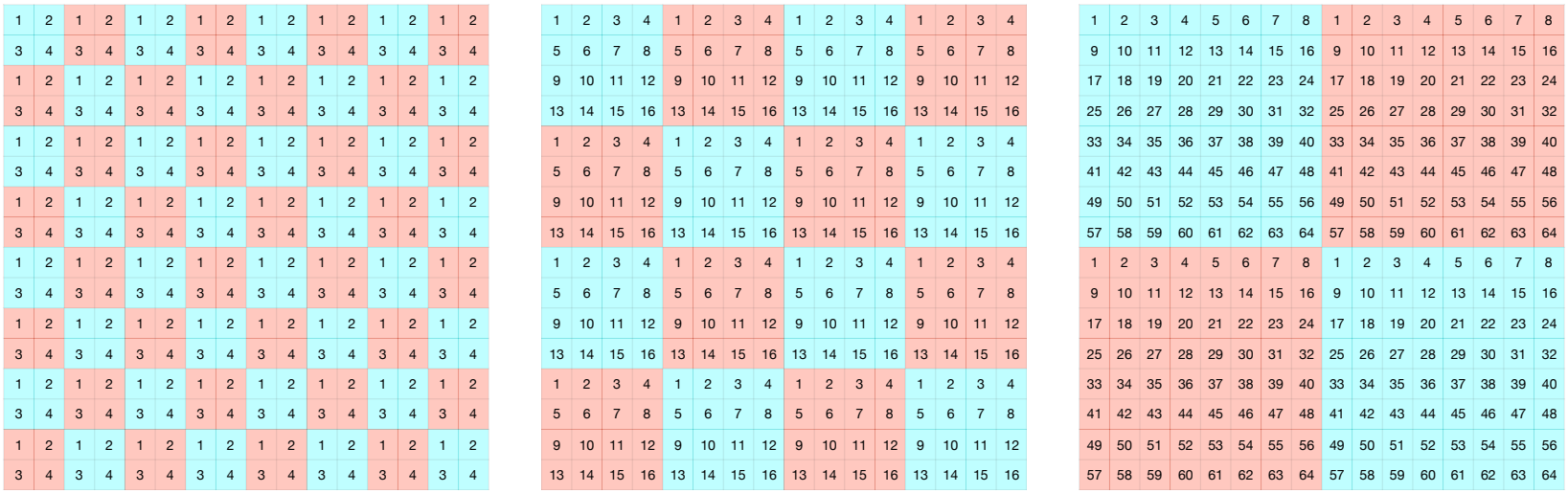
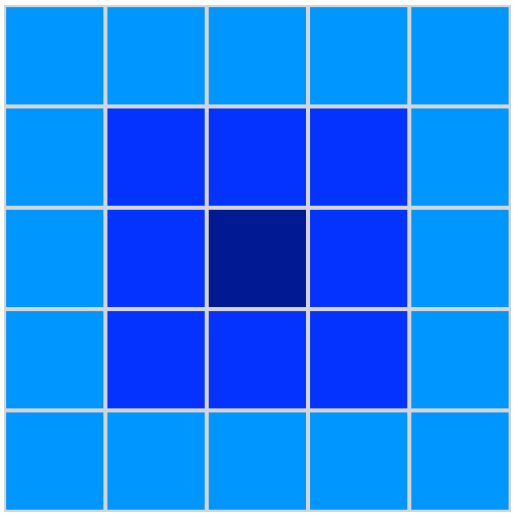


Figure 8: Division of pixel inputs into blocks and groups. Blocks of sizes 2×2, 4×4 and 8×8 accordingly result into 4, 16 and 64 pixel groups. Different colors are added to visually underline the stricture of a single group.

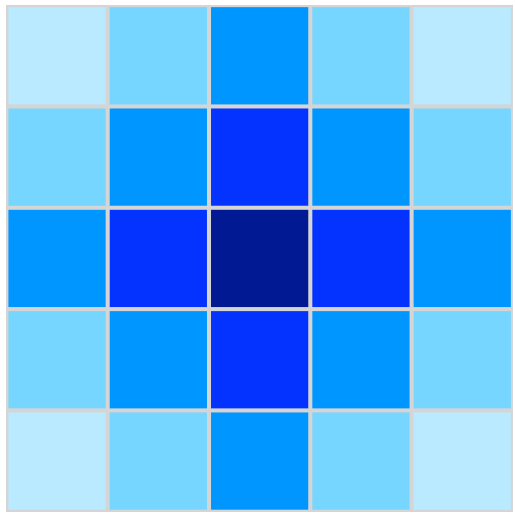
Block PixelCNN

- Two-stream PixelCNN++ convolutions have already demonstrated the ability to correctly cover the receptive field for each pixel within a single fully autoregressive region.
- Now for each considered pixel we need to grow the receptive field not only across the previous pixels within the current block but also across the neighbouring blocks to ensure the autoregressive dependence on all the pixels from the previous groups across the image.
- The filters in the convolution have to be extended to include pixels of the same groups in the neighbouring blocks.

Block PixelCNN



((a)) receptive field with *square* pattern



((b)) receptive field with *cross* pattern

Figure 9: Two simple patterns for neighbouring block inclusions in dilated block convolution are square and cross patterns.

Block PixelCNN

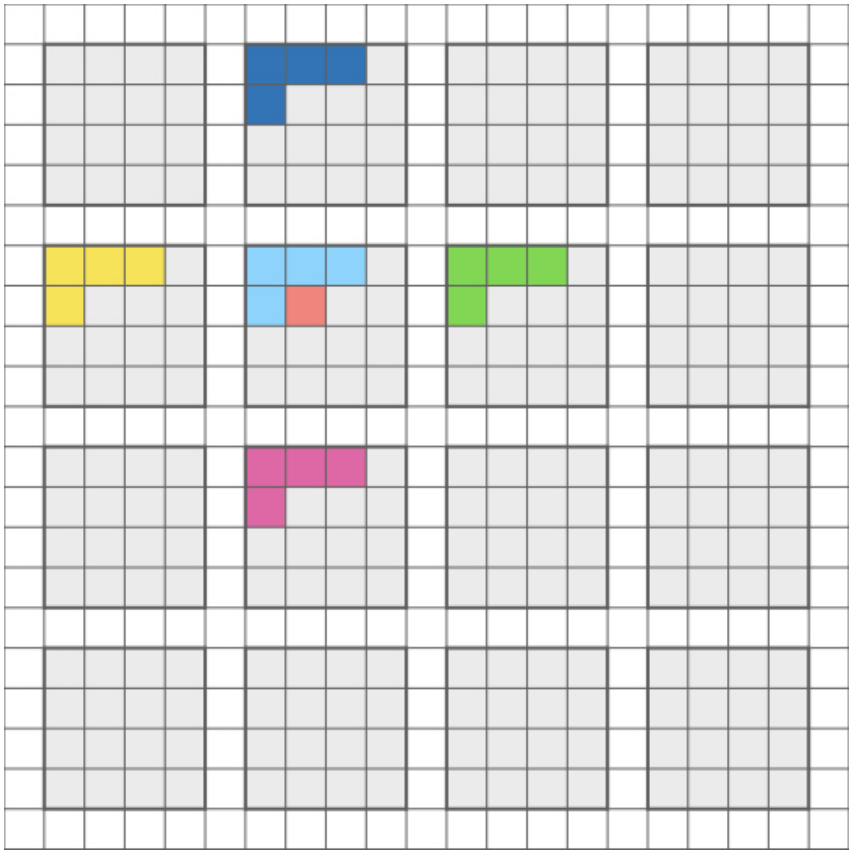


Figure 10: **Dilated block convolution** with cross neighbouring block inclusion pattern applied to a red pixel in 16×16 input feature maps divided into 4×4 blocks.

Block PixelCNN

Implementation of dilated block convolutions includes the following steps:

- Internal padding with pixel-size columns and rows of zeros.
- Division of feature maps into 5 equal parts along the last axis after each convolution followed by shifting in pixels by padding and cropping.
- Summation of the shifted feature maps.

Block PixelCNN

Downsampling options:

- Pixel-strided block convolutions (changes number of groups inside each block).
- Block-strided block convolutions (changes number of blocks across the image).

Upsampling options:

- Pixel-strided transposed convolutions (changes number of groups inside each block).
- Block-wise upsampling followed by block convolutions (changes number of blocks across the image).

Block PixelCNN

Proposed strategy:

- Use pixel-strided block convolutions until block size reaches 2×2.
- After that switch to block-strided block convolutions.
- Upsampling trajectory mimics the downsampling: at each resolution upsampling changes the number of groups inside each block if the corresponding downsampling also changed the number of groups inside each block, otherwise it changes the number of blocks.

Block PixelCNN

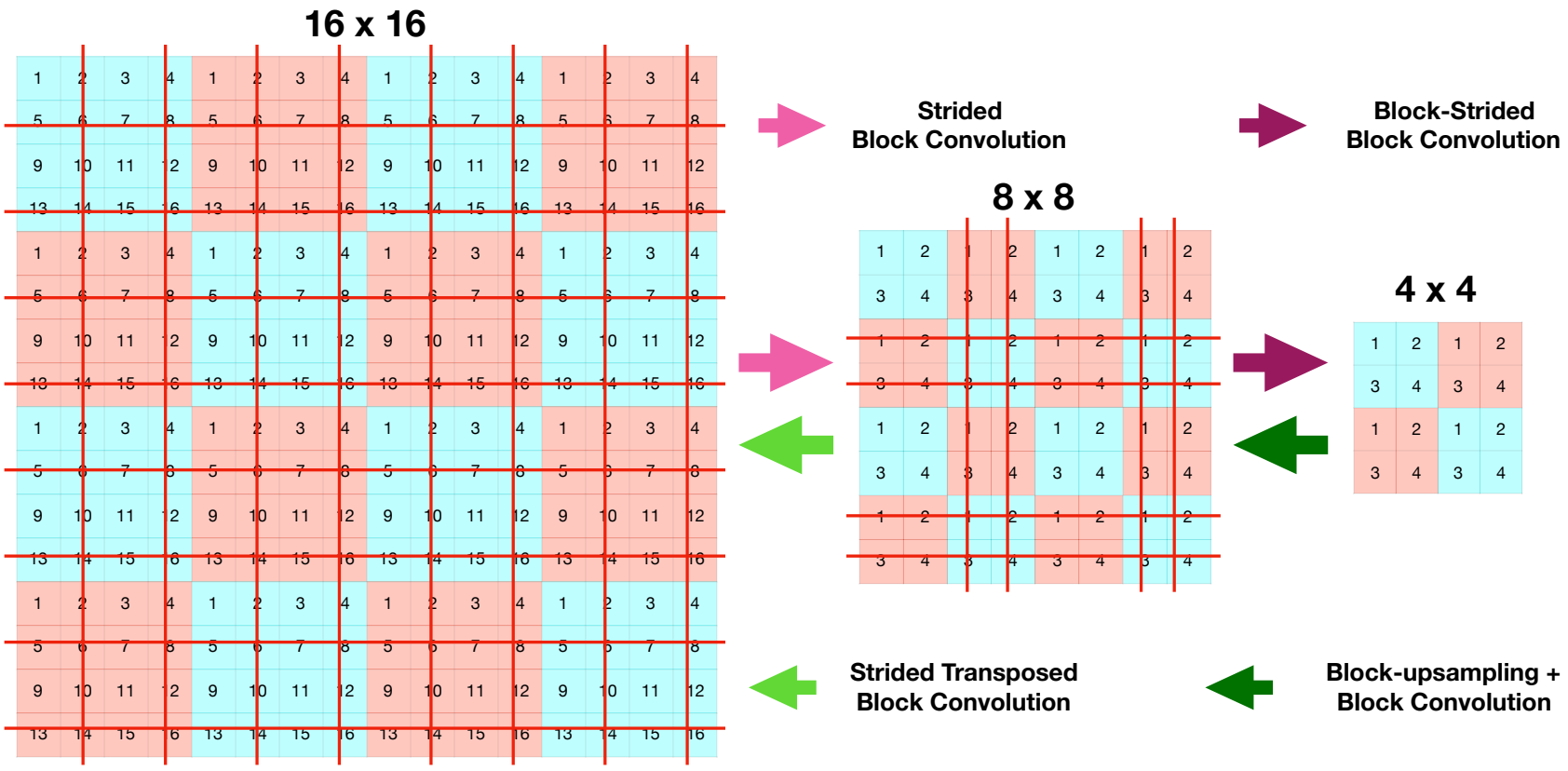


Figure 11: Example for 4×4 blocks over 16×16 image with 2 down-/upsampling layers.

Block PixelCNN

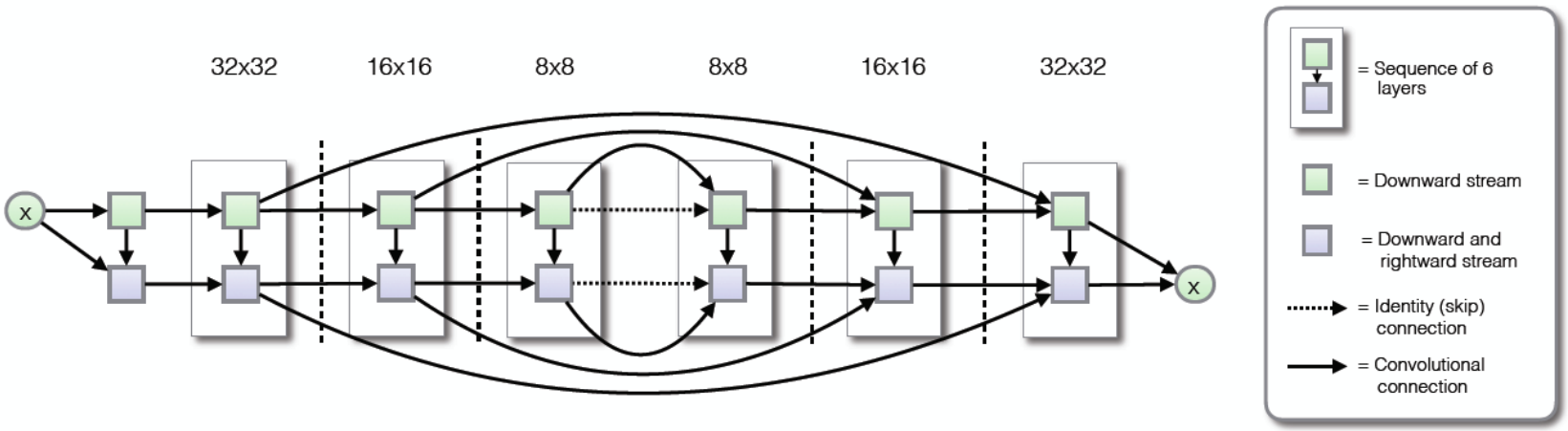


Figure 12: Neural network architecture of the Block PixelCNN *without* within-group interaction. Figure from [Salimans et al., 2017].

Block PixelCNN

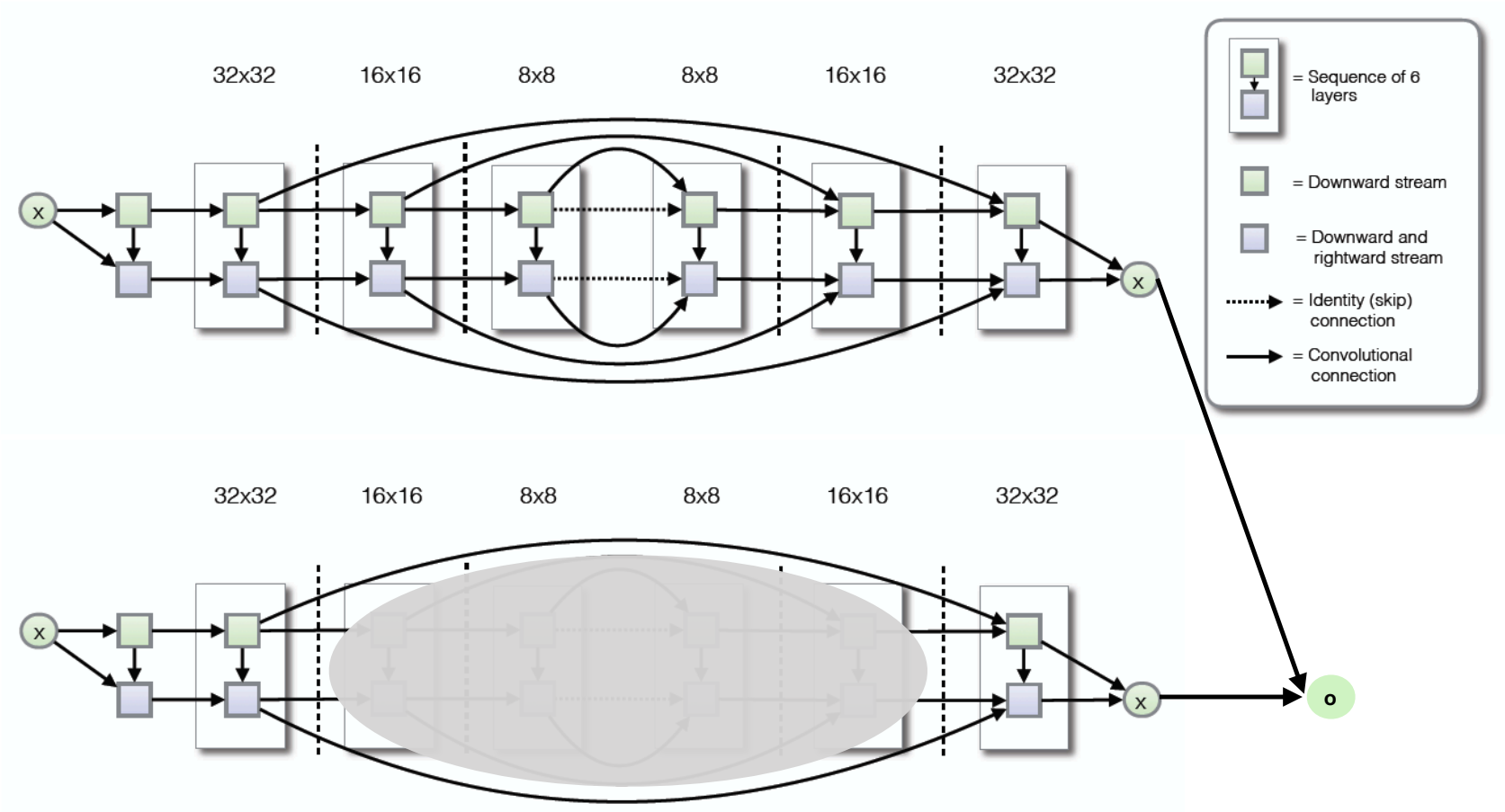


Figure 13: Neural network architecture of the Block PixelCNN *with* within-group interaction.

Experiments: training setup

- **Data**: CIFAR-10 and ImageNet downsized to 32×32.
- **Implementation**: Python, TensorFlow.
- **Loss function**: Log-likelihood measured in bits per dimension.
- **Generative task**: Upscaling.

Experiments: model comparison

	without within-group interaction	*with* within-group interaction
Block PixelCNN 2×2	3.91	3.33
Block PixelCNN 4×4	3.34	3.25
Block PixelCNN 8×8	3.25	3.25

Table 1: Comparison of negative log-likelihood on the CIFAR-10 test set measured in bits-per-dim for Block PixelCNNs of different scale and with different type of within-group interaction.

Experiments: model comparison

	sampling speed, sec	speed-up compared to PixelCNN++
Block PixelCNN 2×2 *without* within-group interaction	2.6	$\times 58$
Block PixelCNN 2×2 *with* within-group interaction	3.0	$\times 50$
Block PixelCNN 4×4 *without* within-group interaction	8.7	$\times 17$
Block PixelCNN 4×4 *with* within-group interaction	10.3	$\times 15$
Block PixelCNN 8×8 *without* within-group interaction	28.8	$\times 5$
Block PixelCNN 8×8 *with* within-group interaction	35.1	$\times 4$
PixelCNN++ [Salimans et al., 2017]	150	$\times 1$

Table 2: Comparison of generation time on CIFAR-10 dataset for a minibatch of 16 images for different Block PixelCNN models on Nvidia Tesla P100 GPU.

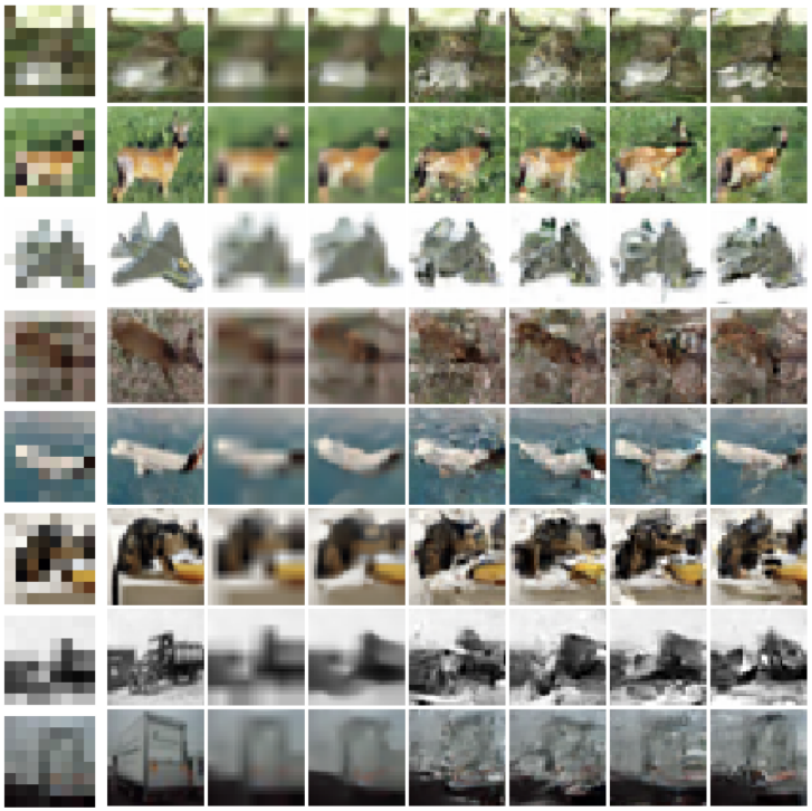
Experiments: upscaling task

((a)) Block PixelCNN 2×2 without within-group interaction

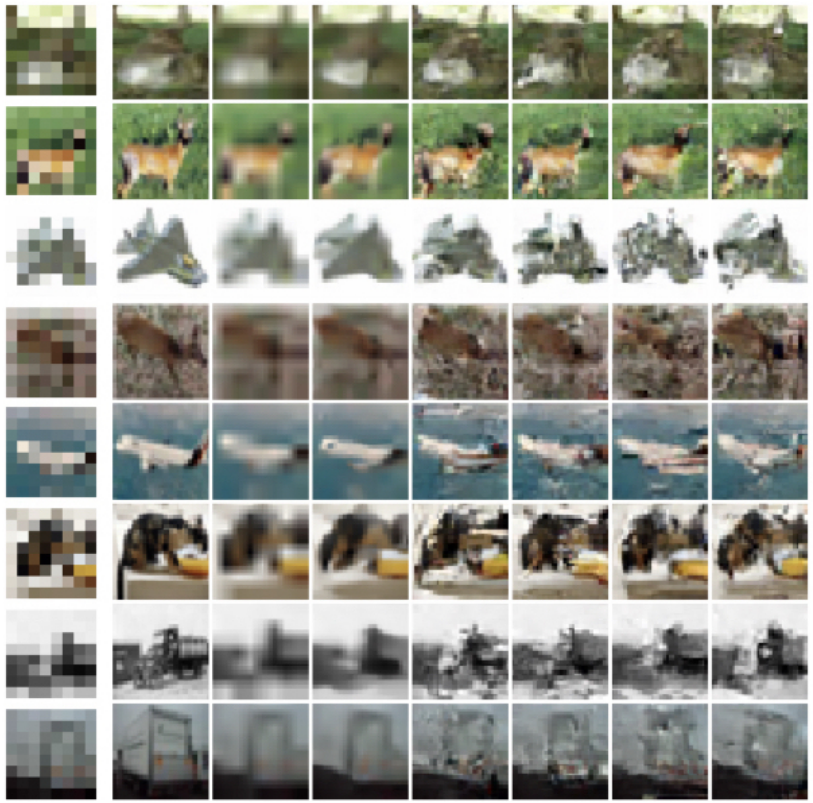
((b)) Block PixelCNN 2×2 with within-group interaction

Figure 14: In each image first columns show model input; second - ground truth images; third - upsampled image obtained by bilinear interpolation; fourth - mean of predicted pixel distribution; fifth to eighth - random samples from predicted distributions.

Experiments: upscaling task



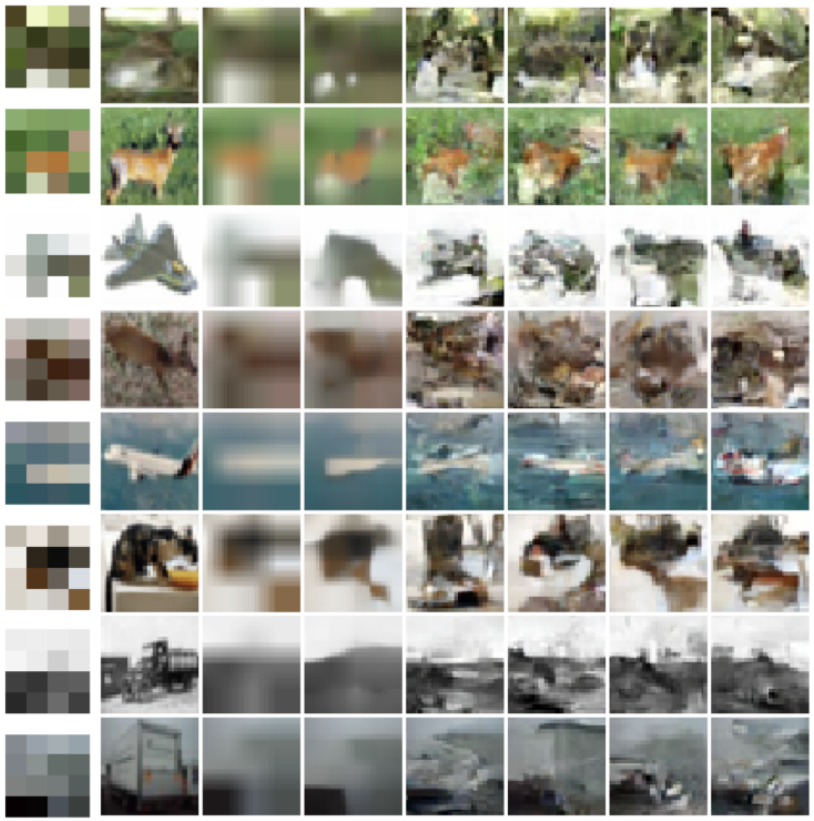
((a)) Block PixelCNN 4×4 without within-group interaction



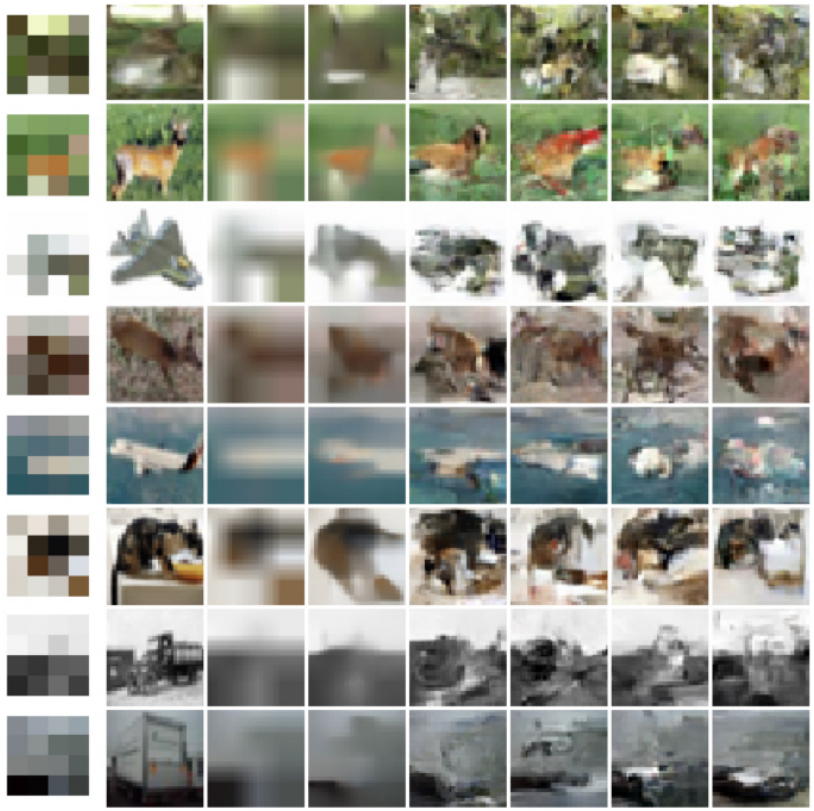
((b)) Block PixelCNN 4×4 with within-group interaction

Figure 15: In each image first columns show model input; second - ground truth images; third - upsampled image obtained by bilinear interpolation; fourth - mean of predicted pixel distribution; fifth to eighth - random samples from predicted distributions.

Experiments: upscaling task



((a)) Block PixelCNN 8×8 without within-group interaction



((b)) Block PixelCNN 8×8 with within-group interaction

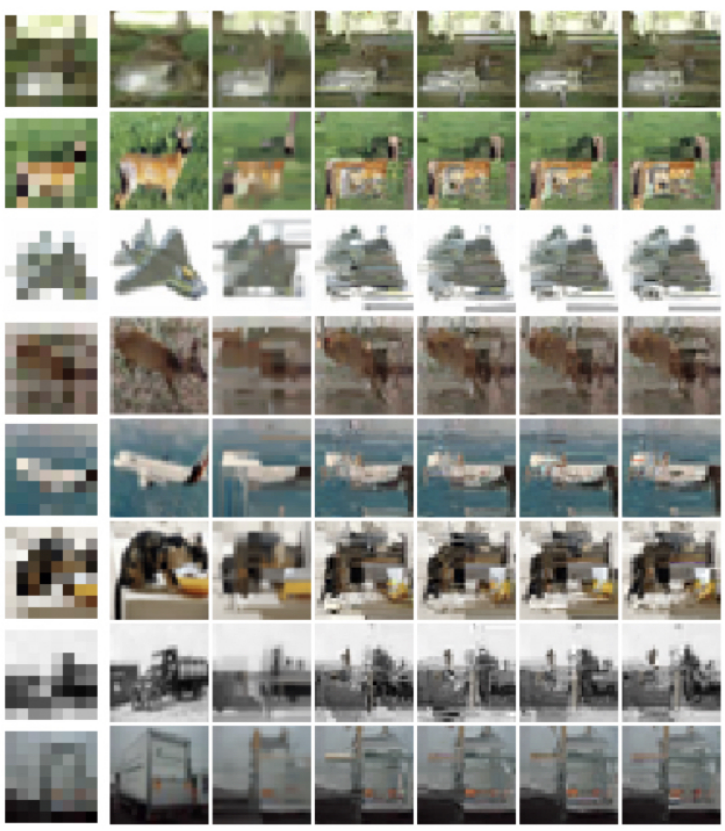
Figure 16: In each image first columns show model input; second - ground truth images; third - upsampled image obtained by bilinear interpolation; fourth - mean of predicted pixel distribution; fifth to eighth - random samples from predicted distributions.

Experiments: bits-per-dim analysis

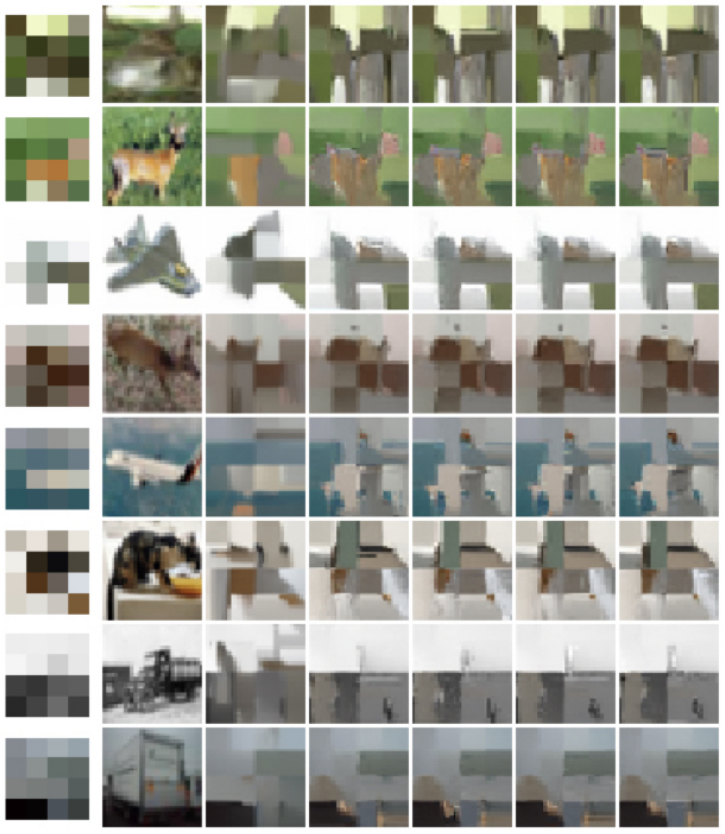
	total bits per dim	bits per dim on the first group	bits per dim on all groups except the first
Block PixelCNN 2×2 without within-group interaction	3.91	6.84	2.94
Block PixelCNN 2×2 with within-group interaction	3.33	4.72	2.87
Block PixelCNN 4×4 without within-group interaction	3.33	6.84	3.10
Block PixelCNN 4×4 without within-group interaction	3.25	5.61	3.09
Block PixelCNN 8×8 without within-group interaction	3.25	6.83	3.19
Block PixelCNN 8×8 without within-group interaction	3.25	6.71	3.19

Table 3: Bits per dimension on CIFAR-10 test set for different Block PixelCNN models.

Experiments: cross-upscaling



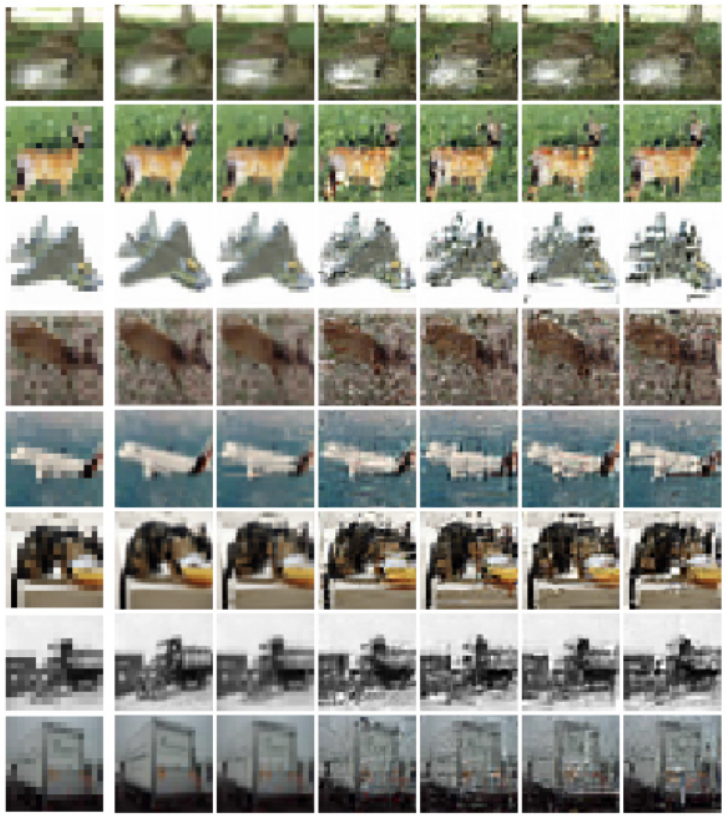
((a)) upscaling inputs downsampled to 8×8 pixels



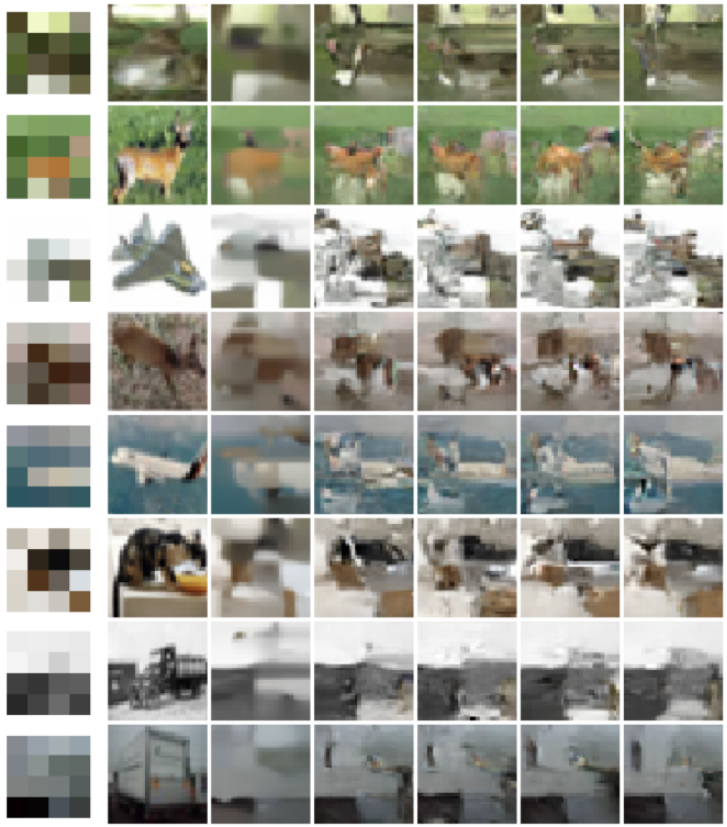
((b)) upscaling inputs downsampled to 4×4 pixels

Figure 17: Upscaling results for random downsampled test images obtained by the Block PixelCNN 2×2 model with within-group interactions.

Experiments: cross-upscaling



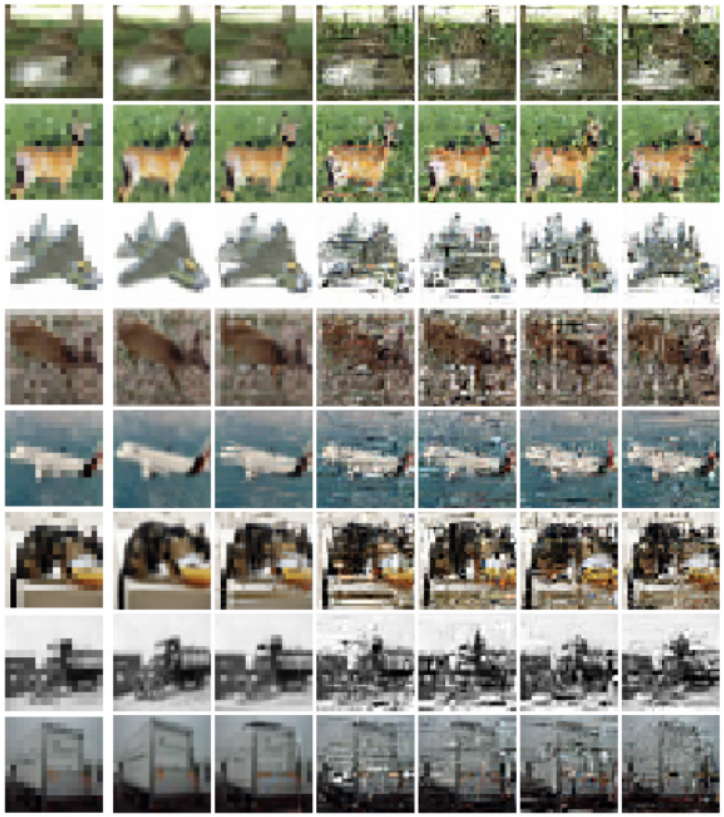
((a)) upscaling inputs downsampled to 16×16 pixels



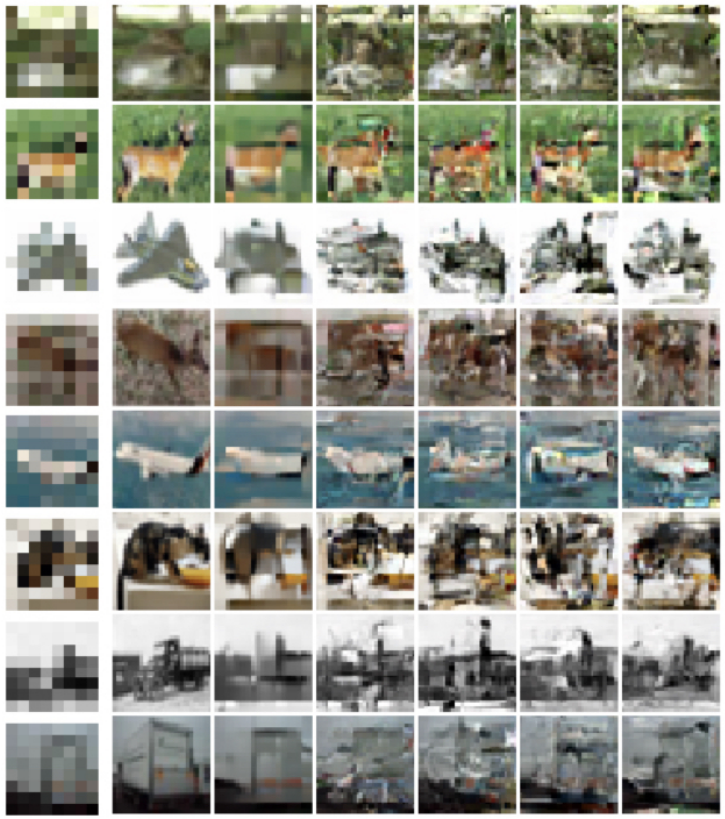
((b)) upscaling inputs downsampled to 4×4 pixels

Figure 18: Upscaling results for random downsampled test images obtained by the Block PixelCNN 4×4 model with within-group interactions.

Experiments: cross-upscaling



((a)) upscaling inputs downsampled to 16×16 pixels



((b)) upscaling inputs downsampled to 8×8 pixels

Figure 19: Upscaling results for random downsampled test images obtained by the Block PixelCNN 8×8 model with within-group interactions.

Experiments: comparison to the state of the art

Model	bits per dim
Deep Diffusion [Sohl-Dickstein et al., 2015]	≤ 5.40
NICE [Dinh et al., 2014]	4.48
DRAW [Gregor et al., 2015]	≤ 4.13
Deep GMMs [Oord and Schrauwen, 2014]	4.00
ConvDRAW [Gregor et al., 2016]	≤ 3.58
Real NVP [Dinh et al., 2016]	3.49
Block PixelCNN 2×2	$\leq \mathbf{3.33}$
Pyramid PixelCNN [Kolesnikov and Lampert, 2017]	3.32
Block PixelCNN 4×4	$\leq \mathbf{3.25}$
Block PixelCNN 8×8	$\leq \mathbf{3.25}$
PixelCNN [Oord et al., 2016b]	3.14
VAE with IAF [Kingma et al., 2016]	≤ 3.11
Gated PixelCNN [Oord et al., 2016a]	3.03
PixelRNN [Oord et al., 2016b]	3.00
Grayscale PixelCNN [Kolesnikov and Lampert, 2017]	≤ 2.98
DenseNet VLAE [Chen et al., 2016]	≤ 2.95
PixelCNN++ [Salimans et al., 2017]	2.92

Table 4: Negative log-likelihood for different generative models on CIFAR-10 test set measured as bits per dimension.

Experiments: comparison to the state of the art

Model	bits per dim
ConvDRAW [Gregor et al., 2016]	4.40
Real NVP [Dinh et al., 2016]	4.28
Block PixelCNN $\mathbf{2 \times 2}$	$\mathbf{\leq 4.12}$
Multiscale PixelCNN [Reed et al., 2017]	3.95
PixelRNN [Oord et al., 2016b]	3.86
PixelCNN [Oord et al., 2016b]	3.83

Table 5: Negative log-likelihood for different generative models on ImageNet 32×32 test set measured as bits per dimension.

Contribution

- A model that factorizes the joint pixel distribution into arbitrary number of equal groups was developed.
- *Block dilated convolutions* were introduced to model dependency of each pixel on all the pixels in the previous groups.
- A special procedure for downsamling and upsampling was suggested to efficiently grow the receptive field.
- Proposed convolutional operations are applicable to arbitrary number of groups G that the joint pixel distribution is factorized into. It allowed to model all the dependencies using a single network.
- Resulting model *Block PixelCNN* was trained with different numbers of groups, and the impact of the length of the dependency range was analyzed.

Summary

- Proposed model, Block PixelCNN, has the desired property of flexibility and allows different trade-offs between log-likelihood and generation speed.
- Obtained results on CIFAR-10 and downsized ImageNet are comparable to the state-of-the-art generative image models.
- Analysis of the impact of different groups on the total log-likelihood showed that modeling of the first group is the most expensive part of the loss.
- Further analysis showed that shorter connections present in the 2×2 blocks are more beneficial than longer connections from the visual perspective, while log-likelihood is inversely related.

Future work

Further model development includes the following steps:

- Modeling pixels of the first group separately and fully autoregressively.
- Development of a conditional Block PixelCNN and introduction of the high-level image information (e.g. labels, keypoints etc.).
- Optimized implementation of the neural network architecture.
- Training on multiple GPUs.

Thank you!

Bibliography I


Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. (2016).
Variational lossy autoencoder.
arXiv preprint arXiv:1611.02731.


Dinh, L., Krueger, D., and Bengio, Y. (2014).
Nice: Non-linear independent components estimation.
arXiv preprint arXiv:1410.8516.


Dinh, L., Sohl-Dickstein, J., and Bengio, S. (2016).
Density estimation using real nvp.
arXiv preprint arXiv:1605.08803.


Gregor, K., Besse, F., Rezende, D. J., Danihelka, I., and Wierstra, D. (2016).
Towards conceptual compression.
In *Advances In Neural Information Processing Systems*, pages 3549–3557.

Bibliography II


Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015).
Draw: A recurrent neural network for image generation.
arXiv preprint arXiv:1502.04623.


Kingma, D. P., Salimans, T., Jozefowicz, R., Chen, X., Sutskever, I., and Welling, M. (2016).
Improved variational inference with inverse autoregressive flow.
In *Advances in Neural Information Processing Systems*, pages 4743–4751.


Kolesnikov, A. and Lampert, C. H. (2017).
Pixelcnn models with auxiliary variables for natural image modeling.
In *International Conference on Machine Learning*, pages 1905–1914.


Oord, A. v. d., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. (2016a).
Conditional image generation with pixelcnn decoders.
In *Advances in Neural Information Processing Systems*, pages 4790–4798.

Bibliography III


Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016b).
Pixel recurrent neural networks.
arXiv preprint arXiv:1601.06759.


Oord, A. v. d. and Schrauwen, B. (2014).
Factoring variations in natural images with deep gaussian mixture models.
In *Advances in Neural Information Processing Systems*, pages 3518–3526.


Reed, S., Oord, A. v. d., Kalchbrenner, N., Colmenarejo, S. G., Wang, Z., Belov, D., and de Freitas, N. (2017).
Parallel multiscale autoregressive density estimation.
arXiv preprint arXiv:1703.03664.


Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. (2017).
Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications.
arXiv preprint arXiv:1701.05517.

Bibliography IV


Sohl-Dickstein, J., Weiss, E. A., Maheswaranathan, N., and Ganguli, S. (2015).
Deep unsupervised learning using nonequilibrium thermodynamics.
arXiv preprint arXiv:1503.03585.