Grenoble Institute of Technology

Master's Educational Program:

Master of Science in Industrial and Applied Mathematics (Data Science)

Exploring Generative Image Modeling with Block PixelCNNs

Author: .....
Ekaterina Iakovleva

Certified by .....
Jakob Verbeek
Senior Researcher at THOTH, INRIA
Thesis Supervisor

Grenoble

June 2018

Exploring Generative Image Modeling

with Block PixelCNNs

by

Ekaterina Iakovleva

Submitted to the Grenoble Institute of Technology
on June, 2018

Abstract

Generative image modeling is one of the central but extremely challenging tasks in the modern unsupervised learning. There are numerous potential applications of probabilistic density models which are interesting both from theoretical and practical point of view. Block Pixel Convolutional Neural Networks, or simply Block PixelCNNs, proposed in this thesis combine approaches of Multiscale PixelCNNs and original PixelCNNs and fill the gap between them with the purpose to explore the impact made by different group structuring of the image pixels. Resulting model demonstrates competitive log-likelihood scores compared to other state-of-the-art generative models on CIFAR-10 and ImageNet datasets. Qualitative performance of different Block PixelCNNs is explored for the upscaling task on CIFAR-10 dataset.

Thesis Supervisor: Jakob Verbeek
Title: Senior Researcher at THOTH, INRIA

Acknowledgments

I would like to wholeheartedly express the gratitude to my supervisor, Jakob Verbeek, for fruitful discussions and productive collaboration. I would also like to thank Roman Klokov for valuable advice and ideas regarding technical part of the research.

Contents

1 Introduction **11**

1.1 Motivation 11

1.2 Contribution 12

1.3 Outline 13

2 Related work **14**

2.1 Tractable likelihood models 14

2.1.1 Tractable likelihood model zoo 14

2.1.2 PixelCNNs and variations 18

2.2 Variational models 25

2.3 Generative adversarial networks 28

3 Block PixelCNNs **30**

3.1 Pixel groups and blocks 30

3.2 Building Block PixelCNNs 31

3.2.1 Dilated block convolutions 32

3.2.2 Spatial up- and downscaling 34

3.3 Generative Image Model 36

4 Experiments **37**

4.1 Datasets and experimental setup 37

4.2 Network architectures and optimizantion 38

4.3 Experimental results 39

4.3.1 Model development and analysis on CIFAR-10 40
4.3.2 Comparison to the state of the art 47

5 Conclusion and future work **53**

List of Figures

2-1 Left: Conventional three hidden layer autoencoder. Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard autoencoder cannot predict or sample new data. Right: MADE. The network has the same structure as the autoencoder, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks (M^{W^1}, M^{W^2}, M^V). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(\mathbf{x}) = p(x_2)p(x_3|x_2)p(x_1|x_2, x_3)$. Picture and caption from [Germain et al., 2015]. 18

2-2 A visualization of the context for image pixels. Each of them depends only on the previous pixels (in terms of raster scan order) and cannot see any of the future pixels. Figure from [Oord et al., 2016b]. 19

2-3 An example of mask used for the 5x5 convolutional kernel to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. Figure from [Oord et al., 2016b]. 20

2-4 Top: PixelCNNs have a blind spot in the receptive field that can not be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field. Figure and caption from [Oord et al., 2016b]. 22

2-5 PixelCNN++ model structure. "Downward" and "Downward and rightward" streams correspond respectively to vertical and horizontal stacks from Gated PixelCNN. Figure from [Salimans et al., 2017]. 23

2-6 Example pixel grouping and ordering for a 4×4 image. Pixels of group 1 are modelled using shallow PixelCNN. Pixels of group 2 depend on all pixels from group 1, pixels of group 3 depend on all pixels from groups 1 and 2, and so forth. Figure from [Reed et al., 2017]. 24

2-7 A simple form of causal upscaling network, mapping from a $K \times K$ image to $K \times 2K$. The same procedure can be applied in the vertical direction to produce a $2K \times 2K$ image. In reference to figure 2, we use this type of network for the group $1 \rightarrow 2$ factor. For the $1, 2 \rightarrow 3$ factor, we apply the same technique but in the vertical direction, and subsample the even columns (starting from zero) to get the group 3 pixels. For the $1, 2, 3 \rightarrow 4$ factor, we first build a $2K \times 2K$ input image with group 4 pixels zeroed. This is fed through a spatial dimension-preserving ResNet, from which we subsample the output group 4 pixels. (**A**) In the simplest version, a deep convolutional network (in our case ResNet) directly produces the right image from the left image, and merges column-wise. (**B**) A more sophisticated version extracts features from a convolutional net, splits the feature map into spatially contiguous blocks, and feeds these in parallel through a shallow PixelCNN. The result is then merged as in (A). Note that the entire pathway is trained end-to-end in both cases. Caption and figure from [Reed et al., 2017]. 25

3-1 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. 31

(a) 2×2 31

(b) 4×4 31

(c) 8×8 31

3-2 Two simple patterns for neighbouring block inclusions in dilated block convolution are square and cross patterns. Cross pattern results in a slower growth of receptive field of Block PixelCNN over the image blocks, however, for small resolution inputs and due to the presence of down- and upscaling convolutions in the network, this effect is neglectable. On the other hand, it allows to spend more GPU memory on higher number of convolution filters / additional convolutions in the network. 32

(a) receptive field with *square* pattern 32

(b) receptive field with *cross* pattern 32

3-3 Dilated block convolution with cross neighbouring block inclusion pattern applied to a red pixel in 16×16 input feature maps divided into square blocks of size four. Gray cells represent blocks of input feature maps. For simplicity, instead of two-streamed, masked version of convolution is shown. White rows and columns correspond to zero wrapping of blocks. Blue cells show application of convolution filter parameters to the input and red cell is the target pixel position for this particular convolution operation. Zero wrapping is required to prevent mixing of autoregressive features coming from different blocks. Simultaneous application of such filters to all cells in the input feature maps result into proposed dilated block convolution. 33

4-1 Upscaling results for random downsampled test images obtained by the Block PixelCNN 2×2 model *without*/*with* within-group interaction. In each image first column 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. . 41

(a) Block PixelCNN 2×2 without within-group interaction 41

(b) Block PixelCNN 2×2 with within-group interaction 41

4-2 Upscaling results for random downsampled test images obtained by the Block PixelCNN 4×4 model *without*/*with* within-group interaction. Columns are aranged in a manner, similar to Figure 4-1. 42

(a) Block PixelCNN 4×4 without within-group interaction 42
(b) Block PixelCNN 4×4 with within-group interaction 42
4-3 Upscaling results for random downsampled test images obtained by the Block PixelCNN 8×8 model *without*/*with* within-group interaction. Columns are aranged in a manner, similar to Figure 4-1. 42
(a) Block PixelCNN 8×8 without within-group interaction 42
(b) Block PixelCNN 8×8 with within-group interaction 42
4-4 Upscaling results for random downsampled test images obtained by the Block PixelCNN 2×2 model with within-group interactions. In each image first column show model input; second - ground truth images; third - mean of predicted pixel distribution; fourth to seventh - random samples from predicted distributions. 46
(a) upscaling inputs downsampled to 8×8 pixels 46
(b) upscaling inputs downsampled to 4×4 pixels 46
4-5 Upscaling results for random downsampled test images obtained by the Block PixelCNN 4×4 model with within-group interactions. Columns are organized in a manner similar to Figure 4-4. 48
(a) upscaling inputs downsampled to 16×16 pixels 48
(b) upscaling inputs downsampled to 4×4 pixels 48
4-6 Upscaling results for random downsampled test images obtained by the Block PixelCNN 8×8 model with within-group interactions. Columns are organized in a manner similar to Figure 4-4. 48
(a) upscaling inputs downsampled to 16×16 pixels 48
(b) upscaling inputs downsampled to 8×8 pixels 48

List of Tables

4.1 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. 40

4.2 Comparison of generation time on CIFAR-10 dataset for a minibatch of 16 images for different Block PixelCNN models on Nvidia Tesla P100 GPU. . . 40

4.3 Bits per dimension on CIFAR-10 test set for different Block PixelCNN models. First column corresponds to bits-per-dim on the whole image. Second column contains bits-per-dim computed only on image pixels that correspond to group 1 in each Block PixelCNN model. Third columns contains bits-per-dim computed on all image pixels except pixels of group 1 in each model. . . 44

4.4 Negative log-likelihood for different generative models on CIFAR-10 test set measured as bits per dimension. 49

4.5 Negative log-likelihood for different generative models on ImageNet 32×32 test set measured as bits per dimension. 50

Chapter 1

Introduction

Generative image modeling is one of the most active topics in the modern deep learning. Its numerous practical applications include image compression, inpainting, denoising, deblurring, super-resolution, colorization, generation of images from text descriptions and many more. Being an unsupervised type of task, image density estimation has access to virtually endless pool of training data. The challenging part is that images are very high-structural and high-dimensional objects to model which makes distribution estimation difficult.

There are different approaches to solve this task, and recent breakthroughs in deep learning allowed to develop models that can generate diverse natural images. PixelCNNs and their modifications are among the most well-known probabilistic density models. They suggest to factorize joint pixel distribution as a product of univariate conditional distributions and order autoregressive pixel dependencies in a raster scan order.

1.1 Motivation

The goal of this work is to improve and generalize PixelCNNs, to explore possible trade-offs between accuracy and sampling complexity in particular and to gain new insights into image modeling in general.

The PixelCNN is an autoregressive probabilistic density model that captures the full generality of pixel inter-dependencies without introducing any independence assumptions. While achieving state-of-the-art results in density estimation for natural images, due to

their autoregressive nature they all require $O(N)$ network evaluations to generate a single image of N pixels which could be costly, especially for images of high resolution (since such images consist of huge number of pixels). Multiscale PixelCNN, introduced later, addresses this problem by suggesting to factorize the joint distribution not into per-pixel conditional factors but rather into four groups of conditionally independent pixels that depend only on the pixels of the previous groups. Such approach deliberately breaks some dependencies between pixels but allows to sample pixels of the same group independently from each other, resulting in $O(\log N)$ network evaluations for sampling and significant speedup.

This leads to a reasonable question: is factorization into four groups optimal? What other trade-offs between quality and generation speed are possible by capturing most, many or some dependencies but still not all of them? In addition to that, from an implementation point of view PixelCNNs and Multiscale PixelCNNs differ significantly, and neither of them could be properly reformulated into another one which is illogical given their common features. All these issues call for a model that can benefit from both approaches and in which various speed-accuracy trade-offs can be made, generalizing PixelCNNs and Multiscale PixelCNNs. This model should be flexible in terms of longevity of pixel dependencies and easy to learn at the same time. Development of such a model, as a result, is the main purpose of this thesis.

1.2 Contribution

With the goals set in Section 1.1, a new PixelCNN-type model, named *Block PixelCNN*, was developed based on PixelCNN and Multiscale PixelCNN. Unlike original PixelCNN and similarly to Multiscale PixelCNN, Block PixelCNN can factorize image pixels into predefined number of equal size groups of conditionally independent pixels to speed up image generation. Similarly to PixelCNN and unlike Multiscale PixelCNN, Block PixelCNN models joint distribution using single neural network. Proposed model is, thus, easily scalable from Multiscale PixelCNN with its 4 groups of pixels to original PixelCNN where the number of groups is equal to the total number of image pixels, and represents a generalization of both approaches to modeling joint pixel distribution.

Development of such a model required conceptually novel architectural components and corresponding implementations. Source code of a well-known modification of the original model, PixelCNN++, was used as a basis for proposed Block PixelCNN. However, in order to introduce group structure of the image pixels ordinary convolutional layers were substituted with new type of convolutions – dilated block convolutions which are also a part of contribution of this work.

1.3 Outline

Related work is reviewed in Chapter 2 to place these efforts in the context of other results in development of generative image models. The model that was used as a basis for this research, Multiscale PixelCNN, is also described in the same chapter. Thereafter its generalization and improvement are presented in Chapter 3. These changes allow to explore the properties of PixelCNN-type models and offer a trade-off between model quality in terms of log-likelihood and image generation time. Chapter 4 presents details and experimental results obtained during application of proposed models to different datasets. Finally, in Chapter 5 conclusions and directions for the future work are discussed.

Chapter 2

Related work

There are a number of different approaches to the generative image modeling task. The majority of these methods can be divided into three main categories. First category contains models that tractably model joint pixel distribution. Second category consists of stochastic latent variable models which do some intractable inference steps for density estimation. Finally, the third category is represented by generative adversarial networks (GANs), which do not directly model pixel distributions but provide a way to generate supports for data distributions. Details regarding PixelCNN and Multiscale PixelCNN models which served as a basis for the proposed Block PixelCNN, are described below in the Section 2.1.

2.1 Tractable likelihood models

To underline the differences, approaches in this category are divided into two groups: PixelCNNs plus their variations and several conceptually different methods.

2.1.1 Tractable likelihood model zoo

One of the first models that used deep learning approach to solve image modeling task is **Deep Gaussian Mixture Model** [Oord and Schrauwen, 2014a]. **Deep GMM** is a generalization of Gaussian Mixture Model, and it is constructed by stacking multiple GMM-layers on top of each other which resembles the structure of deep learning architectures. Each

GMM-layer is a set of linear transformations applied to input feature vectors in a fully-connected manner. During sampling a single path through all the layers and according series of linear transformations is considered to obtain a single component in the mixture, thus every path in the network correspond to a single component in the mixture. These components are weighted to obtain the final GMM. The authors claim that these layers are able to efficiently factorize different variations present in natural images (changes in brightness, color, rotations of objects etc.) without the necessity to use an exponential number of components to model all the combinations of variations, as would do a conventional Gaussian Mixture model. A special version of Expectation Maximization (EM) algorithm for training Deep GMM was also proposed in the paper.

Another deep learning framework called **Non-linear Independent Component Estimation** [Dinh et al., 2014], or **NICE**, suggests to learn a transformation f of the data such that in the new space the distribution of the mapping had independent components and, thus, was factorizable:

$$p_H(f(x)) = \prod_d p_{H_d}(f_d(x)). \tag{2.1}$$

Function f should satisfy two criteria: its Jacobian should be easy to compute and the function itself should be easily invertible. The change of the variables and the factorization property result in the following log likelihood of the model:

$$\log p_X(x) = \sum_{d=1}^{D} \log \left(p_{H_d}(f_d(x))\right) + \log \left| \det \frac{df}{dx^T} \right|. \tag{2.2}$$

If one considers $p_H(h)$ to be a factorizable prior and chooses its specific form, maximized log likelihood (2.2) turns into non-linear independent components estimation (NICE) criterion, and the generation procedure becomes as follows:

$$h \sim p_H(h), \quad x = f^{-1}(h). \tag{2.3}$$

Transformation f, proposed in the paper, consists of splitting x into two blocks $(x_{1:d}, x_{d+1:D})$ and applying a transformation that the authors refer to as *additive coupling layer*:

$$h_{1:d} = x_{1:d}, \tag{2.4}$$

$$h_{d+1:D} = x_{d+1:D} + m(x_{1:d}), \tag{2.5}$$

where m is an arbitrary complex function implemented via neural network and referred to as *coupling function*. The Jacobian of such transformation is a triangular matrix for any m, so its determinant is easily computed as a product of diagonal elements (in this case, it is further simplified since diagonal elements are identities). Inverse transformation is trivial and looks as follows:

$$x_{1:d} = h_{1:d}, \tag{2.6}$$

$$x_{d+1:D} = h_{d+1:D} - m(h_{1:d}). \tag{2.7}$$

At least three coupling layers are required to allow all dimensions influence each other. Overall, the model NICE can be viewed as learning an invertible preprocessing transformation of the dataset.

This approach was further developed by the same authors in the **Real-valued Non-Volume Preserving** transformations, or **RealNVP** [Dinh et al., 2016]. The "change of variable" formula is still the center of the model, as well as coupling layers. However, in this work, instead of additive coupling layers, the authors introduce *affine coupling layers* which look as follows:

$$h_{1:d} = x_{1:d}, \tag{2.8}$$

$$h_{d+1:D} = x_{d+1:D} \odot \exp\left(s(x_{1:d})\right) + t(x_{1:d}), \tag{2.9}$$

where s and t are scale and translation functions. This transformation has the same benefits as NICE: Jacobian is a triangular matrix with easy-to-compute determinant, and the inverse transformation can be easily computed for arbitrary complex and possibly difficult to invert

functions s and t:

$$x_{1:d} = h_{1:d}, \tag{2.10}$$

$$x_{d+1:D} = (h_{d+1:D} - t(h_{1:d})) \odot \exp\left(-s(h_{1:d})\right). \tag{2.11}$$

Two types of masks – checkerboard pattern and channel-wise mask – are proposed for partitioning of x which resulted in masked convolutions. Methods for combining coupling layers and multi-scale architecture are also discussed in the paper.

Neural Autoregressive Distribution Estimation [Larochelle and Murray, 2011] and **DeepNADE** [Uria et al., 2016] are among the first models that considered factorization of joint pixel distribution into the product of nested univariate conditionals:

$$p(\mathbf{x}) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<d}), \tag{2.12}$$

$$\mathbf{x}_{<d} = [x_1, \ldots, x_{d-1}]^T, \tag{2.13}$$

making each output pixel dependent only on the previous input units in terms of arbitrary fixed order. This property is often referred to as the *autoregressive property*. All conditionals in (2.12) are modeled using a single feed-forward neural network, and autoregressive property is preserved by masking corresponding input units before computing each output unit. As a result, evaluation of probability $p(x)$ for a single D-dimensional vector requires $O(D)$ feed-forward passes of the neural network. In order to be able to model arbitrary conditional probabilities, in DeepNADE the ordering of the input was treated as a stochastic variable with a uniform distribution and sampled for each update of the model. Training was done by maximization of the expected likelihood over the ordering of variables. Such network defines a factorial number of different models corresponding to different order of dependencies but with shared parameters.

Authors of **Masked Autoencoder for Distribution Estimation** [Germain et al., 2015] also considered autoregressive decomposition of joint pixel distribution (2.12) and suggested to incorporate it into autoencoders. Autoencoders are feed-forward neural networks designed for unsupervised learning of low-dimensional latent representation $\mathbf{z}$ of initial object $\mathbf{x}$, usu-

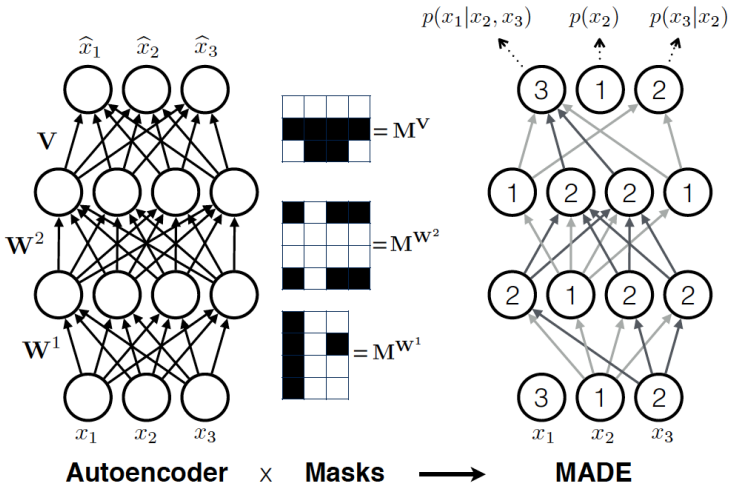


Figure 2-1: Left: Conventional three hidden layer autoencoder. Input in the bottom is passed through fully connected layers and point-wise nonlinearities. In the final top layer, a reconstruction specified as a probability distribution over inputs is produced. As this distribution depends on the input itself, a standard autoencoder cannot predict or sample new data. Right: MADE. The network has the same structure as the autoencoder, but a set of connections is removed such that each input unit is only predicted from the previous ones, using multiplicative binary masks (M^{W^1}, M^{W^2}, M^V). These masks ensure that MADE satisfies the autoregressive property, allowing it to form a probabilistic model, in this example $p(\mathbf{x}) = p(x_2)p(x_3|x_2)p(x_1|x_2, x_3)$. Picture and caption from [Germain et al., 2015].

ally with the purpose of dimensionality reduction. The model is trained so that the output variables regress the input data, while constraining the hidden layer which leads to learning of compressed representation of data. To make each output pixel x_d dependent only on previous input pixels $\mathbf{x}_{<d}$ one must break computational path between x_d and $\mathbf{x}_{\geq d}$. The authors suggest to do it by masking the weights of autoencoder, with different masks corresponding to different ordering over image pixels. The main advantage of MADE is that evaluating probabilities requires single feed-forward pass through the network, while additional cost for simple masking operations is negligible.

2.1.2 PixelCNNs and variations

Pixel Recurrent Neural Networks [Oord et al., 2016c] proposed in 2016 gave rise to a whole new direction in the area of generative image models. The basis of the model, once again, is a factorized joint distribution of image pixels (2.12). However, in this model dependencies between image pixels are modelled in a fixed raster scan order: each pixel

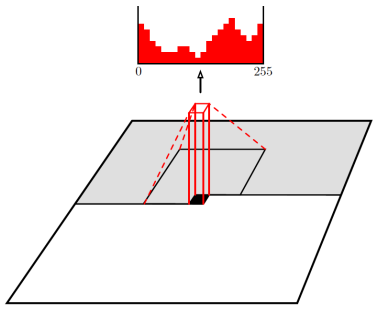


Figure 2-2: A visualization of the context for image pixels. Each of them depends only on the previous pixels (in terms of raster scan order) and cannot see any of the future pixels. Figure from [Oord et al., 2016b].

depends on all pixels above and to the left of it. The ordering of the dependencies is shown in Figure 2-2.

Another novelty is that all conditionals are modelled only by a convolutional neural network. Since natural images are represented by RGB (Red, Blue, Green) channels, dependencies between them also need be taken into consideration. The authors suggest to define an order over channels and to model each conditional in (2.12) as follows:

$$p(x_d|\mathbf{x}_{<d}) = p(x_{d,R}|\mathbf{x}_{<d}) \cdot p(x_{d,G}|\mathbf{x}_{<d}, x_{d,R}) \cdot p(x_{d,B}|\mathbf{x}_{<d}, x_{d,R}, x_{d,G}). \tag{2.14}$$

In addition to that, each pixel value is modelled as a discrete variable that can take one of 256 different values. Importantly, during training and evaluation stages the distributions over the image pixels are computed in parallel, while sampling of new images is a sequential process where each generated pixel is put back into the network as input to produce the next pixel. Two types of architectures are proposed in the paper: **PixelRNN** and **PixelCNN**. **PixelRNN** uses one of two types of LSTM layers: unidirectional Row LSTM or bidirectional Diagonal BiLSTM. The Row LSTM processes the image row-wise from top to bottom, computing features for the whole row at a time using one-dimensional convolution masked to prevent it from seeing future pixels. In the Diagonal BiLSTM each of two directions of the layer processes the image from one corner to the diagonally opposite one, and the features are computed for the whole diagonal at once. To preserve autoregressive property and to include only allowed context, skewing and unskewing operations for feature maps are proposed. The

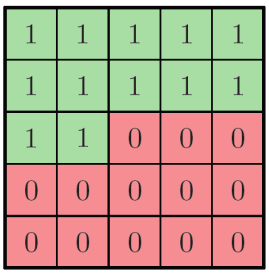


Figure 2-3: An example of mask used for the 5x5 convolutional kernel to make sure the model cannot read pixels below (or strictly to the right) of the current pixel to make its predictions. Figure from [Oord et al., 2016b].

main difference between these types of LSTM layers is their context: Row LSTM captures triangular context above the pixel of interest, whereas Diagonal BiLSTM captures the whole available context (everything above and to the left). In any case PixelRNN has potentially unbounded dependency field of view, however, this comes with a significant computational cost. Solution to this would be to make the receptive field large but not unbounded. With this purpose a **PixelCNN** version of the model was proposed. PixelCNN uses deep convolutional architecture that preserves the spatial resolution. All convolutional kernels are multiplied by the masks, depicted in Figure 2-3, to prevent pixels from seeing future context.

Successive modelling of colour channels is achieved by splitting the feature maps at every layer into three parts and adjusting the centre values of the mask tensors. For computational reasons PixelCNN became much more popular than PixelRNN, even though the latter demonstrated better log-likelihood results on all datasets.

Further development of the PixelCNN resulted in **Gated PixelCNN** and **Conditional PixelCNN** [Oord et al., 2016b] models. In Gated PixelCNN gated activation units are added after each masked convolution to imitate LSTM gates which, as the authors assume, may help to model more complex interactions. Gated activation unit looks as follows:

$$y = \tanh\left(W_f * x\right) \odot \sigma\left(W_g * x\right). \tag{2.15}$$

Here $*$ denotes convolutional operation and $\odot$ – element-wise product, W_* denote weights. Conditional PixelCNN goes one step further and adds a high-level image information (e.g. keypoints, class labels, etc.) coded as a latent vector $\mathbf{h}$ to model the conditional distribution

in a similar way to (2.12):

$$p(\mathbf{x}|\mathbf{h}) = \prod_{d=1}^{D} p(x_d|\mathbf{x}_{<d}, \mathbf{h}), \tag{2.16}$$

$$\mathbf{x}_{<d} = [x_1, \ldots, x_{d-1}]^T. \tag{2.17}$$

Conditionals are modelled by adding these latent vectors $\mathbf{h}$ to the gated units:

$$y = \tanh\left(W_f * x + V_f^T \mathbf{h}\right) \odot \sigma\left(W_g * x + V_g^T \mathbf{h}\right). \tag{2.18}$$

Another important issue addressed in this paper was the presence of blind spots in the receptive field of the original PixelCNNs. Application of a series of masked convolutions, used in PixelCNNs, resulted in a receptive field which did not fully captured all the pixels available for prediction of a certain pixel in an autoregressive manner. To solve this problem, the authors suggest to combine two convolutional network stacks. The first stack (vertical) models all rows above the current pixel, while the second one (horizontal) models current row to the left of the current pixel. The stacks are combined by providing output from the vertical stack of the current layer and the output from the previous layer as input to the horizontal stack. Blind spots and two-stack structure of the receptive field are shown in Figure 2-4.

Authors of **PixelCNN++** [Salimans et al., 2017] developed their own implementation of Gated PixelCNN and proposed some modifications that resulted in significant performance gain in terms of log-likelihood. These modifications can be described as follows:

- Conditionals in (2.12) or (2.16) are modelled using mixture of logistic distributions instead of 256-way softmax:

$$p(x|\pi, \mu, s) = \sum_{k=1}^{K} \pi_i \left[\sigma((x + 0.5 - \mu_k)/s_k) - \sigma((x - 0.5 - \mu_k)/s_k)\right]. \tag{2.19}$$

  This approach uses less parameters to model distributions and results in a smooth optimization objective, compared to discrete distributions obtained by a linear transformation and application of softmax nonlinearity.

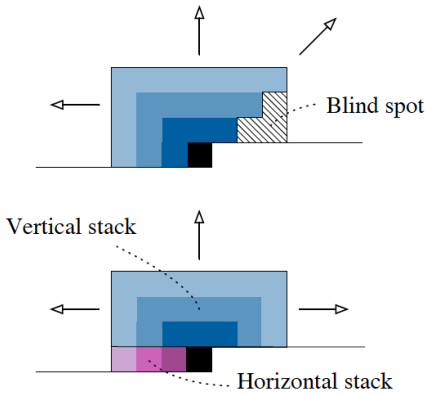


Figure 2-4: Top: PixelCNNs have a blind spot in the receptive field that can not be used to make predictions. Bottom: Two convolutional stacks (blue and purple) allow to capture the whole receptive field. Figure and caption from [Oord et al., 2016b].

- Dependencies between colour sub-pixels are also modelled like a factorized joint distribution instead of masking:

$$\begin{aligned} p(r_d, g_d, b_d|\mathbf{x}_{<d}) =& p(r_d|\mu_r(\mathbf{x}_{<d}), s_r(\mathbf{x}_{<d})) \times p(g_d|\mu_g(r_d, \mathbf{x}_{<d}), s_g(r_d, \mathbf{x}_{<d})) \\ & \times p(b_d|\mu_b(r_d, g_d, \mathbf{x}_{<d}), s_b(r_d, g_d, \mathbf{x}_{<d})), \\ \mu_g(r_d, \mathbf{x}_{<d}) =& \mu_g(\mathbf{x}_{<d}) + \alpha(\mathbf{x}_{<d}) \cdot r_d, \\ \mu_b(r_d, g_d, \mathbf{x}_{<d}) =& \mu_b(\mathbf{x}_{<d}) + \beta(\mathbf{x}_{<d}) \cdot r_d + \gamma(\mathbf{x}_{<d}) \cdot g_d. \end{aligned} \tag{2.20}$$

  If logistic distributions in the model had been replaced by mixtures of univariate Gaussians, pixels would have been modelled by mixtures of 3-dimensional Gaussians with full covariance matrices.

- Spatial resolution preserving convolutions are substituted for strided convolutions followed by transposed convolutions. Resulting "downsampling-upsampling" U-shaped neural network architecture helps to grow receptive field much faster than in Gated PixelCNN which contributes to performance gain in terms of both quality and training speed.

- Finally, the neural network architecture is structured into ResNet blocks [He et al., 2016] with gated activation units after them. Additional skip connections between blocks of

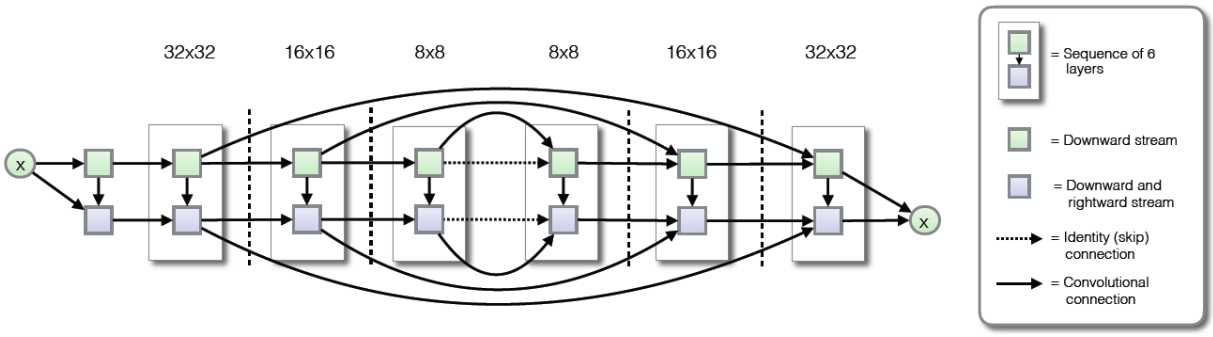


Figure 2-5: PixelCNN++ model structure. "Downward" and "Downward and rightward" streams correspond respectively to vertical and horizontal stacks from Gated PixelCNN. Figure from [Salimans et al., 2017].

the same spatial resolution before and after the bottleneck are added similarly to U-nets [Ronneberger et al., 2015]. Model structure can be found in Figure 2-5.

All of these PixelCNN-type architectures mentioned above have one common drawback: generation speed. The necessity to sample images sequentially, pixel by pixel, is a major bottleneck of this type of generative models. There are works ([Ramachandran et al., 2017, Oord et al., 2016a]) that suggest caching some activations in the computational graph to gain certain acceleration, but even with this optimization generation suffers from the same bottleneck. As pointed out in **Multiscale PixelCNN**[Reed et al., 2017], ideally one would prefer to sample image pixels in parallel which would correspond to modeling them as independent. But such fully-factorizable models tend to be quite weak in terms of their ability to capture fine details. The solution could be to deliberately break some (preferably weak) dependencies between pixels. Instead of using nested univariate conditionals, as in (2.12), the authors of Multiscale PixelCNN suggest to factorize joint pixel distribution into G disjoint equal factors of conditionally independent pixels. Assuming that each factor contains T pixels, the probabilistic density model turns into

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p(x_{1:T}^{(g)} | x_{1:T}^{(1:g-1)}). \tag{2.21}$$

Pixels within each group depend only on the pixels of the previous groups and can now be

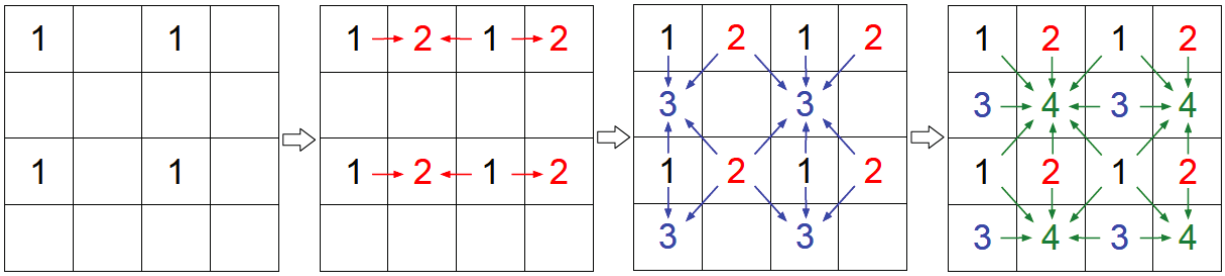


Figure 2-6: Example pixel grouping and ordering for a 4×4 image. Pixels of group 1 are modelled using shallow PixelCNN. Pixels of group 2 depend on all pixels from group 1, pixels of group 3 depend on all pixels from groups 1 and 2, and so forth. Figure from [Reed et al., 2017].

sampled in parallel. In the paper the authors focused on factorization into $G = 4$ autoregressive groups of pixels. To create group structure, they tile the image with 2×2 blocks, as in Figure 2-6. Importantly, now pixels can see some of the context below and to the right of it. In addition to that, none of the dependencies between neighbouring pixels are modelled as independent which helps to preserve local image structure. As in previous models, training and evaluation are computed in parallel, while generation is now autoregressive with respect to groups. Each of the factors in (2.21) is modelled using a neural network. Two types of upscaling networks were proposed in the paper. Their details can be found in Figure 2-7.

Greyscale PixelCNN and **Pyramid PixelCNN** [Kolesnikov and Lampert, 2017] are among the most recent PixelCNN-type models. The basic idea these two models share is to introduce auxiliary variables $\hat{X}$ to the model with data X and to consider their joint distribution:

$$p(X, \hat{X}) = p(\hat{X})p(X|\hat{X}). \tag{2.22}$$

Factors $p(\hat{X})$ and $p(X|\hat{X})$ are modelled using PixelCNN++ architecture. In Greyscale PixelCNN 4-bit per pixel quantized greyscale versions of original images are used as auxiliary variables $\hat{X}$ that the true colour images X become conditioned on. This way the authors decouple shape and colour modeling. In Pyramid PixelCNN $\hat{X}$ corresponds to a twice lower resolution view of X which decouples the image model into a sequence of low-res image model and upscaling model.

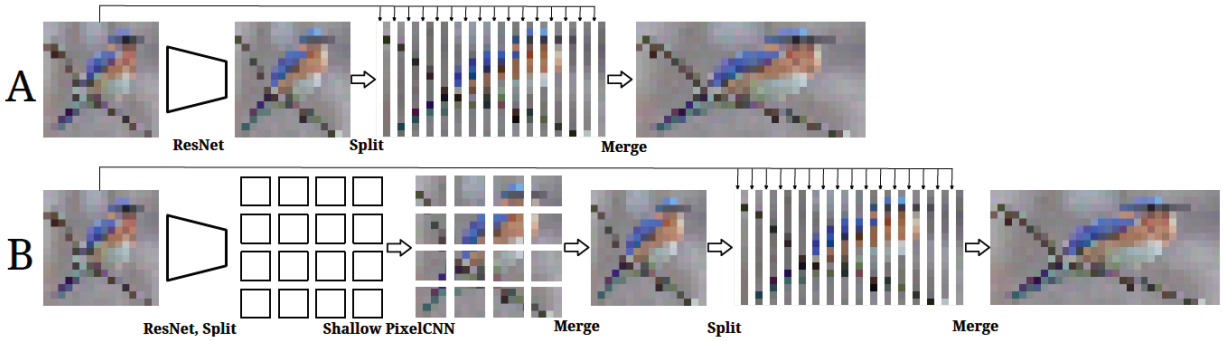


Figure 2-7: A simple form of causal upscaling network, mapping from a $K \times K$ image to $K \times 2K$. The same procedure can be applied in the vertical direction to produce a $2K \times 2K$ image. In reference to figure 2, we use this type of network for the group $1 \to 2$ factor. For the $1, 2 \to 3$ factor, we apply the same technique but in the vertical direction, and subsample the even columns (starting from zero) to get the group 3 pixels. For the $1, 2, 3 \to 4$ factor, we first build a $2K \times 2K$ input image with group 4 pixels zeroed. This is fed through a spatial dimension-preserving ResNet, from which we subsample the output group 4 pixels. (**A**) In the simplest version, a deep convolutional network (in our case ResNet) directly produces the right image from the left image, and merges column-wise. (**B**) A more sophisticated version extracts features from a convolutional net, splits the feature map into spatially contiguous blocks, and feeds these in parallel through a shallow PixelCNN. The result is then merged as in (A). Note that the entire pathway is trained end-to-end in both cases. Caption and figure from [Reed et al., 2017].

2.2 Variational models

Compared to the previously reviewed works, methods in this section are united by an idea to avoid direct modeling of the data distribution. This is achieved by assuming there is some latent process that generates the data $\mathbf{x}$ and representing it in terms of latent variables $\mathbf{z}$ in a joint generative model $p(\mathbf{x}, \mathbf{z})$. Careful design of latent structure allows to build rich models that can be efficiently sampled from but require intractable inference which hinders the performance.

Variational autoencoders (VAE) were introduced in [Kingma and Welling, 2013]. Its authors propose a generative model $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x}|\mathbf{z})p_\theta(\mathbf{z})$ with observed data $\mathbf{x}$, its latent representation $\mathbf{z}$, and θ are parameters of the model. Posterior $p_\theta(\mathbf{z}|\mathbf{x})$ is called a probabilistic *encoder* and likelihood $p_\theta(\mathbf{x}|\mathbf{z})$ is called a probabilistic *decoder*. Usually true posterior is intractable and requires some approximation $q_\phi(\mathbf{z}|\mathbf{x})$ with variational parameters ϕ. Posterior

inference is done by maximization of variational lower bound:

$$\mathcal{L}\left(\theta, \phi; \mathbf{x}\right) = -D_{\mathrm{KL}}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z})\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left(\log p_\theta(\mathbf{x}|\mathbf{z})\right). \tag{2.23}$$

Initially probabilistic decoders $p_\theta(\mathbf{x}|\mathbf{z})$ modelled pixels $\mathbf{x}$ as conditionally independent given latent representation $\mathbf{z}$. Such approach turned out to be efficient enough for capturing global image structure, however, it failed to detect small-scale features (texture, sharpness etc.). This affected both likelihood and sample quality. Another limitation of the original model was simplicity of posterior approximations $q_\phi(\mathbf{z}|\mathbf{x})$ which also impacted the quality of inference.

In [Rezende and Mohamed, 2015] the authors propose **Normalizing Flows** – a powerful framework for construction of flexible and complex posterior distributions through iterative procedure. In particular, applying a chain of invertible, smooth mappings $\mathbf{f}_t$, one transforms a random variable $\boldsymbol{z}_0$ with a simple distribution into a random variable $\boldsymbol{z}_T$ with arbitrary complex density:

$$\mathbf{z}_0 \sim q(\mathbf{z}_0|\mathbf{x}), \quad \mathbf{z}_t = \mathbf{f}_t(\mathbf{z}_{t-1}, \mathbf{x}) \quad \forall t = \overline{1, T} \tag{2.24}$$

Probability density function of the resulting variable $\mathbf{z}_T$ is then computed using Jacobians:

$$\log q(\mathbf{z}_T|\mathbf{x}) = \log q(\mathbf{z}_0|\mathbf{x}) - \sum_{t=1}^{T} \log \det \left| \frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}} \right|. \tag{2.25}$$

A similar formula of change of variables has already been seen in NICE and Real NVP models. In fact, both models represent some normalizing flow. NICE is an example of a finite volume-preserving flow which turned out to be computationally efficient but less powerful than other types of flows. Real NVP, as implied in the name of the method, is an instance of a non-volume preserving normalizing flow. The main reason why NICE and Real NVP were considered in the Section 2.1 and not here is that encoding into the latent variable z is deterministic, the likelihood remains tractable and is directly optimized, so the training procedure does not require sampling and variational inference.

Authors of **VAE with Inverse Autoregressive Flow** [Kingma et al., 2016] further develop this idea and consider a series of inverse autoregressive transformations applied to the

latent variable $\mathbf{z}_0$. The chain is initialized with $\boldsymbol{z}_0 = \boldsymbol{\mu}_0 + \boldsymbol{\sigma}_0 \cdot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon}$ is sampled from $\mathcal{N}(0, I)$. The flow is constructed by applying T consecutive transformations:

$$\mathbf{z}_t = \boldsymbol{\mu}_t + \boldsymbol{\sigma}_t \odot \mathbf{z}_{t-1}, \tag{2.26}$$

and is called *inverse autoregressive flow* (**IAF**). Each pair of $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$ is modelled using a separate neural network structured to be autoregressive w.r.t. $\mathbf{z}_{t-1}$, so that the Jacobians $\frac{d\boldsymbol{\mu}_t}{d\mathbf{z}_{t-1}}$, $\frac{d\boldsymbol{\sigma}_t}{d\mathbf{z}_{t-1}}$ and, as a result, $\frac{d\mathbf{z}_t}{d\mathbf{z}_{t-1}}$ were triangular. First two Jacobians have zeros on the diagonal, while the last one has $\boldsymbol{\sigma}_t$. It all leads to the following density of the final D-dimensional variable $\mathbf{z}_t$:

$$\log q(\mathbf{z}_T|\mathbf{x}) = -\sum_{i=1}^{D}\left(\frac{1}{2}\epsilon_i^2 + \frac{1}{2}\log 2\pi + \sum_{t=0}^{T}\log\sigma_t, i\right). \tag{2.27}$$

Obtained distribution is very flexible and is able to closely fit true posterior. Autoregressive neural networks used for computing $\boldsymbol{\mu}_t$ and $\boldsymbol{\sigma}_t$ include MADE [Germain et al., 2015] and PixelCNN [Oord et al., 2016c, Oord et al., 2016b].

In **Variational Lossy Autoencoder** [Chen et al., 2016] authors argue that the latent code $\mathbf{z}$ is generally not used when the decoder $p(\mathbf{x}|\mathbf{z})$ is too expressive and explain it by inefficiency in Bits-Back coding [Hinton and Van Camp, 1993, Honkela and Valpola, 2004]. They also explain why decoder weakening is required and propose a way to control what kind of information is placed into latent variables. Proposed model **VLAE** has learnable prior parametrized by autoregressive flow (AF) and posterior parametrized by inverse autoregressive flow (IAF) introduced in (2.27), both modelled using PixelCNN [Oord et al., 2016c, Oord et al., 2016b]. PixelCNN is also used as autoregressive decoder in VLAE. Autoregressive flow for prior distribution is obtained through transformation of continuous noise $\boldsymbol{\epsilon}$ with density $u(\boldsymbol{\epsilon})$ using invertible, smooth function (or composition of functions) $\mathbf{f}$: $\mathbf{z} = \mathbf{f}(\boldsymbol{\epsilon})$. Obtained density is $\log p(\mathbf{z}) = \log u(\boldsymbol{\epsilon}) + \log\det\frac{d\boldsymbol{\epsilon}}{d\mathbf{z}}$.

Concurrently with VLAE, **PixelVAE** [Gulrajani et al., 2016] was proposed. Similarly to VLAE, this model considers a VAE with conditional PixelCNN decoder and autoregressive prior over latent variables.

Authors of **ConvDRAW** [Gregor et al., 2016] also use VAE setting as a basis for development of their model. Similarly to VLAE, ConvDRAW is a latent variable model of a lossy image compressor. The difference between these two models lies in their approaches to compression: VLAE uses autoregressive model to explicitly control what kind of information should be lost, while ConvDRAW uses so called "conceptual compression". The model learns an hierarchy of latent variables – progressively more abstract representations – and only uses information stored in its topmost levels. The authors argue that this way the model stores only high-level information from the image and remains low-level details for generation. One can change the level of compression by changing the number of stored hierarchy levels. As for neural network architecture, it also differs a lot from VLAE. No PixelCNN-type models are used in any part of the network, LSTMs [Hochreiter and Schmidhuber, 1997] are used for modeling latent representations instead. Decoder models image pixels conditionally independent given latent variables.

Finally, **Deep Diffusion** [Sohl-Dickstein et al., 2015] utilizes the idea that remotely resembles Normalizing Flows [Rezende and Mohamed, 2015]. Inspired by non-equilibrium statistical physics, the authors suggest to systematically and slowly destroy structure in a complex data distribution through an iterative forward diffusion process, modelled with generative Markov chain, to obtain a simple distribution. In addition to that a finite-time reverse diffusion process with the same trajectory is learned to restore data structure and define generative distribution. Both direct and reverse Markov transitions are modelled with Gaussian distributions for continuous data and with binomial for binary data. Training consists of finding the reverse Markov transitions which maximize the lower bound on the log likelihood, and the task of density estimation is in fact reduced to regression on the functions that compute mean and covariance for Gaussians sequence or flip probability for Bernoulli trials.

2.3 Generative adversarial networks

Since their introduction by [Goodfellow et al., 2014] GANs successfully became a very active topic related to generative image modelling in general and various tasks associated with

it. In some sense, they fall in the category of latent variable models discussed above, since the initial step of generation procedure is also sampling a lower-dimensional latent variable from a chosen, usually simple distribution. However, compared to all the previously mentioned methods GANs do not directly model and predict any data distributions. In GANs framework two networks are considered: generator G and discriminator D. Generator takes random samples of latent variables as inputs and outputs a data sample. This sample is fed to the discriminator network which returns probability of this example being sampled from the true data distribution. The discriminator is trained to maximize this probability, and in order to achieve that the generator has to produce data samples that are indistinguishable from samples from the true data distribution. This objective was reformulated by [Goodfellow et al., 2014] as the following two-player minimax game with value function $V(G, D)$:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{data}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim p_z(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))]. \tag{2.28}$$

Global optimum of this objective is reached when the generator starts to sample images from true data distribution.

GAN framework gives an instrument to learn supports of true data distributions without explicitly modelling them, which is the main conceptual difference of these strategies compared to methods reviewed earlier. This difference leads to various issues both within GANs and outside of them, including inability to properly compare GANs to probability density modelling approaches. For these reasons GANs models were left out of scope in this work.

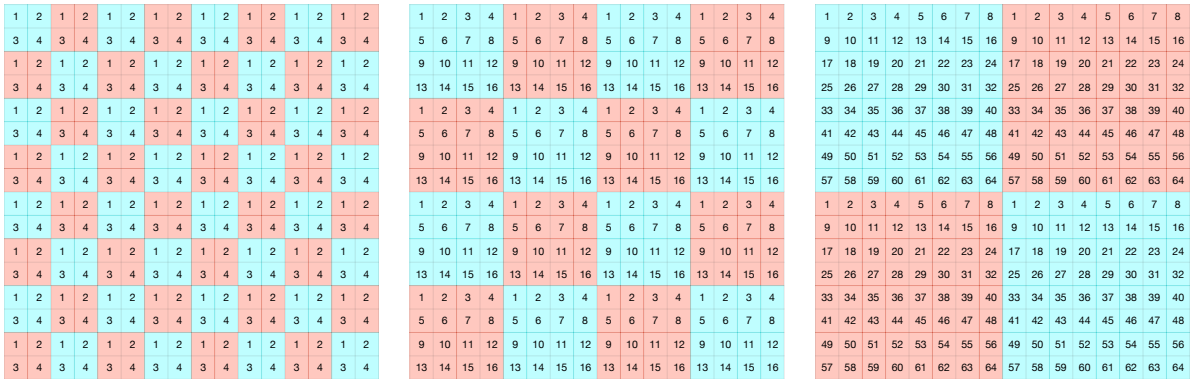
Chapter 3

Block PixelCNNs

In this chapter an adaptation uniting Multiscale PixelCNNs and PixelCNNs++ is introduced. This adaptation involves development of a specific convolutional operation which preserves autoregressive nature of feature extraction along different groups of pixels while being able to successfully grow receptive field both inside each pixel block and across all blocks in the image. Dilated block convolution is specifically designed to achieve this purpose.

3.1 Pixel groups and blocks

Groups and blocks of pixels were initially introduced by Multiscale PixelCNNs. To allow parallel sampling of several pixels in an image while still preserving autoregressive properties they considered dividing images into square *blocks* each containing 4 pixels. Each union of pixels, corresponding to the same position in blocks, form a *group* of pixels (see Figure 2-6 for an example of image being divided into 2×2 blocks and according 4 groups of pixels). Raster scan order is used to enumerate groups of pixels inside blocks similarly to standard PixelCNNs. In fact, original PixelCNN can be viewed as an extreme case of image grouping when the number of groups is equal to the total number of pixels in the image. Autoregressive property is preserved by conditioning pixels of each group on all the pixels in the previous groups. However, the division into smallest possible 2×2 blocks and corresponding 4 groups of pixels is not properly justified. Is it possible that image distribution modelling



(a) 2×2 (b) 4×4 (c) 8×8

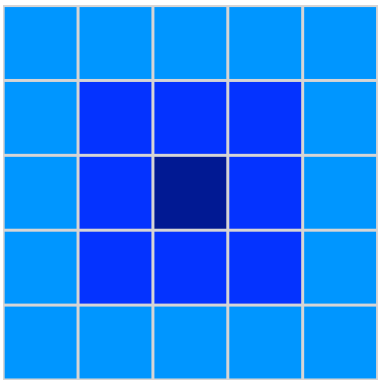
Figure 3-1: Division of pixel inputs into blocks and groups. Blocks of sizes 2×2, 4×4 and 8×8 accordingly result into 4, 16 and 64 pixel groups. Different colors are added to visually underline the stricture of a single group.

might benefit from bigger blocks and higher number of groups, which will form longer autoregressive dependencies?

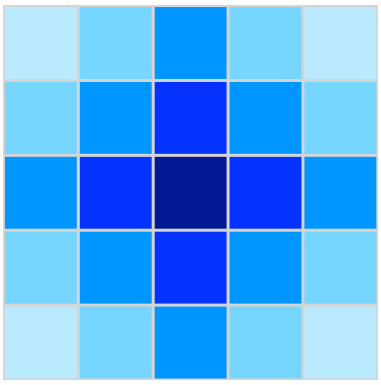
In fact, images of arbitrary size $N \times N$ might be divided into regular blocks using $O(\log^2 N)$ different options (including size options along both spatial dimensions). All those options result into corresponding number of pixel groupings that can be used for introduction of autoregressive dependencies of different lengths. In this work, for simplicity of experiments, we assumed that images are square with spatial sizes equal to powers of two and we assumed that all blocks are square as well. In particular, 2×2, 4×4 and 8×8 blocks were considered, image structure in these cases can be seen in Figure 3-1. However, generalization to rectangular images and blocks is straightforward.

3.2 Building Block PixelCNNs

To preserve autoregressive dependencies of each pixel on all the pixels in the previous groups, the model should consider all the blocks of the image, but leave out pixels of higher group number in them. This could be achieved by a two-stream Pixel Convolutional Network from PixelCNN++ described in Figure 2-5 with all the convolutions in the network being substituted with the convolutions which consider current and neighbouring image blocks while computing output feature maps. Careful construction of operations changing spatial reso-



(a) receptive field with *square* pattern



(b) receptive field with *cross* pattern

Figure 3-2: Two simple patterns for neighbouring block inclusions in dilated block convolution are square and cross patterns. Cross pattern results in a slower growth of receptive field of Block PixelCNN over the image blocks, however, for small resolution inputs and due to the presence of down- and upscaling convolutions in the network, this effect is neglectable. On the other hand, it allows to spend more GPU memory on higher number of convolution filters / additional convolutions in the network.

lutions of the feature maps is also required to allow spatial down- and upscaling of feature maps throughout the network.

3.2.1 Dilated block convolutions

Two-stream PixelCNN++ convolutions have already demonstrated the ability to correctly cover the receptive field for each pixel within a single fully autoregressive pixel patch. In the paper [Salimans et al., 2017] this fully autoregressive pixel patch was the whole image, and in our case this is simply one of the blocks depicted in Figure 3-1. However, 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 coverage of all the pixels from the previous groups across the image. To ensure the required coverage, the filters in the convolution have to be extended to include pixels of the same groups in the neighbouring blocks. Two simple patterns (square and cross) for neighbouring block inclusion and corresponding growth of receptive fields of Block PixelCNNs are presented in Figure 3-2, and an example of the convolution with a cross block pattern is shown in Figure 3-3. Proposed

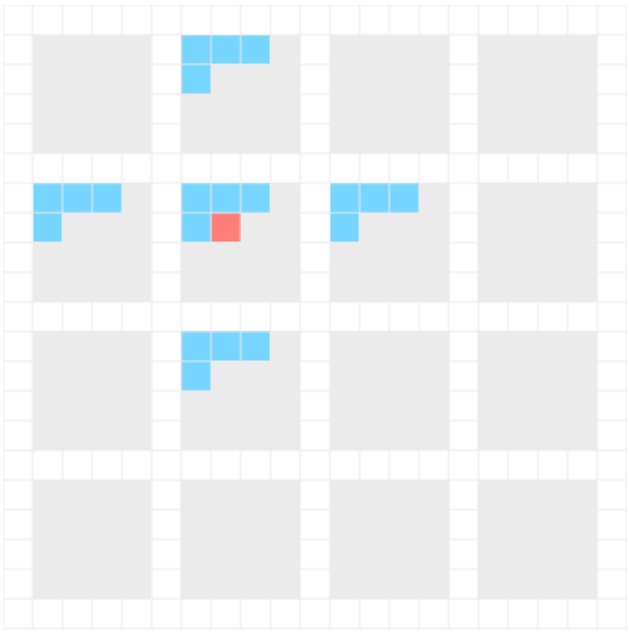


Figure 3-3: Dilated block convolution with cross neighbouring block inclusion pattern applied to a red pixel in 16×16 input feature maps divided into square blocks of size four. Gray cells represent blocks of input feature maps. For simplicity, instead of two-streamed, masked version of convolution is shown. White rows and columns correspond to zero wrapping of blocks. Blue cells show application of convolution filter parameters to the input and red cell is the target pixel position for this particular convolution operation. Zero wrapping is required to prevent mixing of autoregressive features coming from different blocks. Simultaneous application of such filters to all cells in the input feature maps result into proposed dilated block convolution.

convolutions visually resemble dilated convolutions introduced in [Yu and Koltun, 2015] but have blocks instead of single elements in the nodes of dilated convolutional filter. For this resemblance we will further refer to it as *block dilated convolutions*.

A naive implementation of such a convolution would consider a dense filter of the size equal to three block sizes and with weights zeroed out for all the pixels irrelevant to the target pixel. Such computations would include significant amount of redundant operations. This complication can be dealt with by considering another implementation. In fact, the whole operation can be represented as a simple two-stream convolution, as in PixelCNN++, but with a specific aggregation of the output feature maps. In particular, block dialated convolutions can be quite efficiently implemented with ordinary convolutions but with nine times more filters in case of square block pattern, and with five times more filters in case of cross pattern. After each convolution the intermediate feature map is divided across the last

dimension into nine or five equal parts, depending on the pattern, and each part is shifted in its own direction by the number of pixels equal to block size to be eventually aggregated in the target pixel position. Out of two patterns mentioned in this work, the cross seems to be a better choice. It efficiently grows the receptive field across blocks while requiring a considerably less inflation of the number of convolutional filters.

Straightforward application of two-stream convolutions for any intermediate feature maps will hinder the autoregressive connectivity since it will mix pixel representations residing near block borders which can accidentally leak information from the pixels of higher groups. In order to avoid such a mistake, all the blocks have to be interleaved with a single pixel-size rows and columns of zeros prior to the first convolution. Pixel-size rows and columns are enough to prevent undesirable spread of information because all convolutional filters used in the network do not exceed the spatial size 3×3. Further, this internal padding has to be preserved and intentionally zeroed out after each convolution to prevent erroneous mixing of autoregressive block features.

Thus, to obtain the desired dilated block convolutions with cross block inclusion pattern we firstly apply (or enforce) zero wrapping to pixel blocks in the input feature maps, then we compute two-stream PixelCNN++ convolution with the number of output feature maps equal to the $5\times$number of input feature maps. After that the output feature maps are divided into 5 parts corresponding to current and each of the 4 neighbouring blocks in the dialated cross convolutional filter, and for each pixel in the image these parts are combined by shifting and summation to obtain the final values of convolution of the input with the sparse filters mentioned in the naive implementation scheme (see Figure 3-3 for visual explanation of the proposed convolution).

3.2.2 Spatial up- and downscaling

To ensure rapid growth of the receptive field over the image blocks, it is preferable to use down- and upscaling operations in the network. However, these operations have to be designed accordingly not to break the autoregressive nature of features. There are two possible scenarios for spatial downscaling and upscaling when the size of blocks is bigger than two pixels: first scenario down-/upscale the number or groups, second - the number of blocks

(when block size reaches 2 pixels).

To obtain downscaling along the number of groups we used dilated block convolution with stride equal to two. In order to prevent block features mixing and obtain a result with valid zero wrapping it is required to increase zero wrapping thickness to two pixels. In that case, after stride two convolution we again obtain feature maps with a single pixel-wide row/column zero wrapping. Corresponding upscaling in the architecture is obtained by transposed dilated block convolution.

When the size of the group reaches two such scaling is no longer helpful, since its application will destroy autoregressive features. Thus, to continue downscaling, block sizes have to be preserved. This is achieved by standard dilated block convolution which considers every second row and column of blocks of previous feature maps as an input. To upscale properly further in the network each input block is upsampled by factor two before application of dilated block convolution (different weights in the blocks of convolution filters ensure different outputs in neighbouring blocks of pixels).

Proposed techniques for down-/upscaling of spatial resolutions of feature maps in the network are independent and can be used in any order. However, it is preferable to maintain downscaling (and corresponding upscaling) in the following order: feature maps are downsampled only by reducing the block size (the first scenario) until it reaches the minimal 2×2 size, and after that downscaling is performed by leaving out every second block column-wise and row-wise (the second scenario). Such order ensures that the receptive field within each block covers the whole relevant part of this block first, and only after that aggressive expansion of the block-wise receptive field over the whole image is applied.

Presented neural network building components allow to construct Block PixelCNNs on the basis of an existing PixelCNN++ architecture. Resulting neural network can model autoregressive pixel dependencies of arbitrary length in an efficient manner, allowing the exploration of models with different ranges of connectivity.

3.3 Generative Image Model

Previous sections in this chapter were devoted to the implementation of Block PixelCNN. In this section we define the probabilistic density model used in this thesis.

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

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p(x_{1:T}^{(g)} | x_{1:T}^{(1:g-1)}). \tag{3.1}$$

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:

$$p(x|\pi, \mu, s) = \sum_{k=1}^{K} \pi_k \left[\sigma((x + 0.5 - \mu_k)/s_k) - \sigma((x - 0.5 - \mu_k)/s_k)\right]. \tag{3.2}$$

Parameters of this 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 $(1 : g-1)$:

$$\begin{aligned}
p(r_t^{(g)}, g_t^{(g)}, b_t^{(g)} | \mathbf{x}_{1:T}^{(1:g-1)}) =& p(r_t^{(g)} | \mu_r(\mathbf{x}_{1:T}^{(1:g-1)}), s_r(\mathbf{x}_{1:T}^{(1:g-1)})) \\
&\times p(g_t^{(g)} | \mu_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}), s_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)})) \\
&\times p(b_t^{(g)} | \mu_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}), s_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)})), \\
\mu_g(r_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}) =& \mu_g(\mathbf{x}_{1:T}^{(1:g-1)}) + \alpha(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot r_t^{(g)}, \\
\mu_b(r_t^{(g)}, g_t^{(g)}, \mathbf{x}_{1:T}^{(1:g-1)}) =& \mu_b(\mathbf{x}_{1:T}^{(1:g-1)}) + \beta(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot r_t^{(g)} + \gamma(\mathbf{x}_{1:T}^{(1:g-1)}) \cdot g_t^{(g)}.
\end{aligned} \tag{3.3}$$

In this thesis $G = 4$, 16, and 64 corresponding to block sizes 2×2, 4×4 and 8×8 respectively were considered. Experimental results obtained with the proposed model will be discussed in the next chapter.

Chapter 4

Experiments

4.1 Datasets and experimental setup

The generative image model, proposed in Chapter 3 and referred to as *Block PixelCNN*, was evaluated on two datasets: CIFAR-10 [Krizhevsky and Hinton, 2009] and ILSVRC ImageNet [Russakovsky et al., 2014].

- **CIFAR-10** contains 50,000 train samples and 10,000 test samples of 32×32 colour images, with equal number of examples from each of 10 classes, both in train and test sets.

- **ILSVRC ImageNet** is an image dataset organized according to the WordNet hierarchy. On average, 1000 images are provided to illustrate each meaningful concept in WordNet that is described by multiple words or word phrases. In this work, due to computational limitations and for comparison purposes, Small ImageNet dataset was used. Small Imagenet includes downsampled ImageNet images, which can be used for density estimation and generative modeling experiments. This dataset contains 1,281,150 train and 50,000 test samples of 32×32 colour images. Labels are not provided for these images.

The first dataset, CIFAR-10, was used to explore two variants of the proposed model, striking different speed-accuracy trade-offs. The first model is referred to as *Block PixelCNN without*

within-group interaction and the second model is referred to as *Block PixelCNN with within-group interaction*. In addition to that, the second model was evaluated on ILSVRC Small ImageNet. Proposed models are compared to the state-of-the-art generative models on both datasets, including latent-variable models and autoregressive ones.

Similarly to all PixelCNN-type models, training and evaluation are performed fully in parallel for all pixel positions. During sampling stage Block PixelCNN without within-group interaction samples pixels sequentially over groups and fully in parallel for all pixels of the same group. As for Block PixelCNN with within-group interaction, pixel generation is sequential over groups and also sequential over pixels of the same group that are situated in the same quarter of the image.

Loss function used for training and evaluation of the model is negative logistic mixture likelihood. As commonly done in practice, results for CIFAR-10 and ImageNet datasets are reported in *bits-per-dimension*. It corresponds to normalization of negative log-likelihood by the dimensionality of the images which is $32 \times 32 \times 3$ for both datasets. Bits per dimension can be interpreted as the number of bits required to losslessly compress every RGB value for an entropy coding scheme optimized for the proposed model [Oord and Schrauwen, 2014b, Shannon, 2001, Theis et al., 2015].

4.2 Network architectures and optimizantion

The architecture of Block PixelCNN without within-group interaction mimics the structure of PixelCNN++ architecture, described in Figure 2-5. It consists of six Gated ResNet blocks with skip connections between every k and $7-k$ block outputs (k starts from one). To capture autoregressive interactions among different groups all the convolutions in this architecture were substituted by dilated block convolutions described in Section 3.2.1. Moreover, spatial downsampling and upsampling techniques are modified according to Section 3.2.2 to preserve autoregressive nature of the outputs.

Block PixelCNN with within-group interaction extends this architecture by adding an extra component which introduces autoregressive dependencies among pixels of a single group in the regular non-overlapping parts of images. This component is a shallow variation of the

same PixelCNN++ with two Gated ResNet blocks and regular convolutions being substituted by dilated convolutions for correct aggregation of features describing the pixels in the same part of the image and in the same group. Input images are divided into four regular non-overlapping parts which are used as inputs to this additional sub-network. The output feature maps are concatenated to the output feature maps of the larger sub-network and additional 1×1 convolution is applied to obtain final feature maps which are used for prediction of the distribution parameters. The number of convolutional filters (and, as a result, the number of intermediate feature maps) remains constant across all the layers and across both Block PixelCNN sub-networks except for the last layer which has double the number of feature maps due to concatenation.

The PixelCNN++ implementation provided by the authors of the paper [Salimans et al., 2017] was used as a basis for all the experiments. Thus, all the models described above were implemented in Python in TensorFlow deep learning framework [Abadi et al., 2015].

All models were trained with Adam [Kingma and Ba, 2014] optimizer with the learning rate $5 * 10^{-4}$ and with the hyperparameters $\beta_1 = 0.95$, $\beta_2 = 0.9995$. Models used for CIFAR-10 dataset were trained on minibatches of the size 16 and used 85 convolutional filters per layer in 2×2 model and 90 convolutional filters per layer in 4×4 and 8×8 models. Block PixelCNN 2×2 with within-block interaction used for Small ImageNet dataset was trained on minibatches of the size 32 and used 70 convolutional filters per layer. This choice of hyperparameters was based solely on hardware limitations. Models on CIFAR-10 datset were trained for 100 epochs, while the ImageNet model was trained for 10 epochs.

4.3 Experimental results

This section is devoted to the analysis of Block Pixel CNN models and their performance in terms of metrics, generation speed and sampled images. Image upscaling was selected as an example of a generative task.

	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 4.1: 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.

	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	$\times 1$

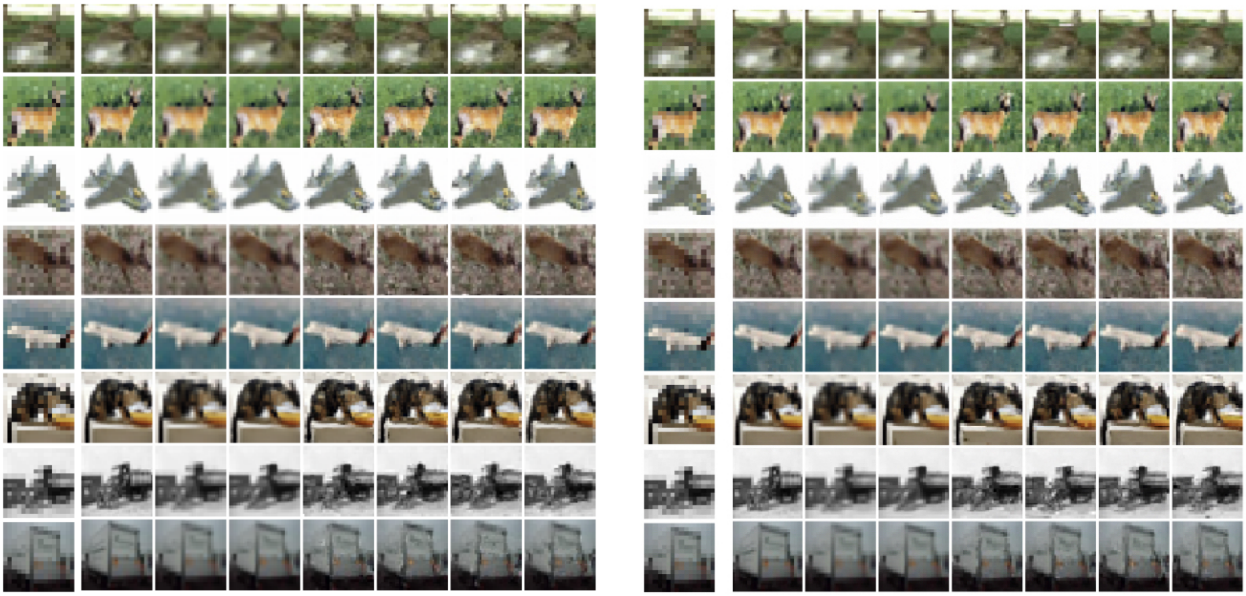
Table 4.2: Comparison of generation time on CIFAR-10 dataset for a minibatch of 16 images for different Block PixelCNN models on Nvidia Tesla P100 GPU.

4.3.1 Model development and analysis on CIFAR-10

Comparison of Block PixelCNN models

As it has been stated above, two types of Block PixelCNN models have been explored: Block PixelCNN without within-group interaction and Block PixelCNN with within-group interaction. Both models were trained with 2×2, 4×4 and 8×8 group blocks and with architecture described in Section 4.2. Bits per dimension achieved on CIFAR-10 test set and sampling speed for a minibatch of 16 images can be found in Table 4.1 and Table 4.2.

From these tables we can see that, overall, Block PixelCNN with within-group interaction demonstrates better log-likelihood results than Block PixelCNN without within-group interaction. This is an expected behaviour since the first model captures longer dependencies



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

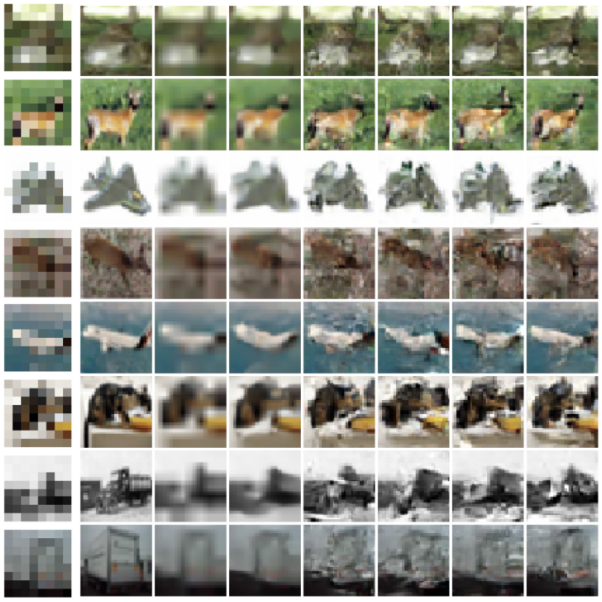
(b) Block PixelCNN 2×2 with within-group interaction

Figure 4-1: Upscaling results for random downsampled test images obtained by the Block PixelCNN 2×2 model *without*/*with* within-group interaction. In each image first column 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.

than the second. Another expected outcome is that with the growth in the number of groups the difference between models in terms of bits-per-dim becomes more and more negligible since there are fewer within-group interactions left for the model to capture inside each image quadrant, and the models become identical. In particular, for the block size 2×2 within-group dependency has the length 64 in a 32×32 image, while for 8×8 block size it is only 4. At the same time the sampling speed of the models with the same block structure but with different within-group interaction is of the same order which makes Block PixelCNN with within-group interactions a better trade-off between log-likelihood and generation speed.

Upscaling task

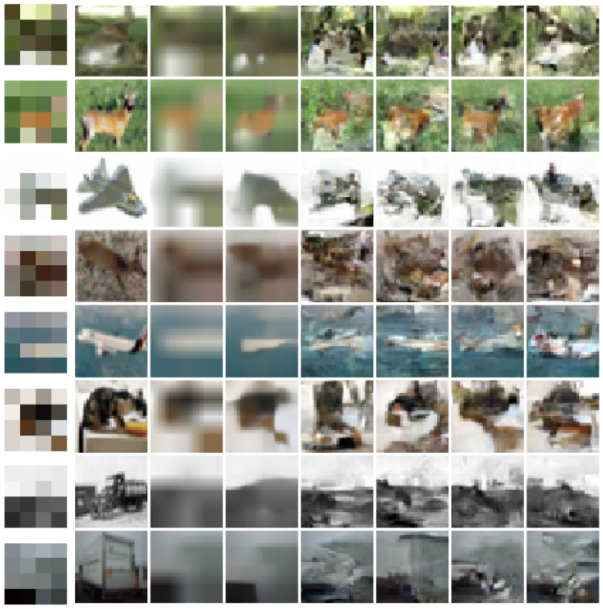
As we have mentioned in the Chapter 1, generative image models have a lot of possible applications. Due to the block structure of the Block PixelCNNs, image upscaling seems to be the most relevant task to test the proposed models on. In our work we used the following



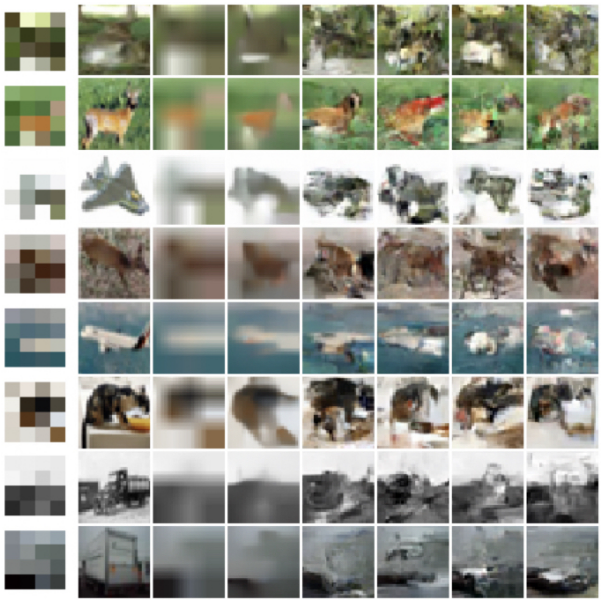
(a) Block PixelCNN 4×4 without within-group interaction

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

Figure 4-2: Upscaling results for random downsampled test images obtained by the Block PixelCNN 4×4 model *without*/*with* within-group interaction. Columns are aranged in a manner, similar to Figure 4-1.



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



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

Figure 4-3: Upscaling results for random downsampled test images obtained by the Block PixelCNN 8×8 model *without*/*with* within-group interaction. Columns are aranged in a manner, similar to Figure 4-1.

generation procedure: random images were sampled from the CIFAR-10 test set, and for each model all pixels, except those corresponding to group 1, were zeroed. These objects were used as input to the tested model which generated the remaining groups starting from the second. It means that Block PixelCNN 2×2 upscales from a 16×16 image, Block PixelCNN 4×4 – from a 8×8 image and Block PixelCNN 8×8 – from a 4×4 image. Figure 4-1, Figure 4-2 and Figure 4-3 show images that were generated by models with and without within-group interactions using the described procedure. Bilinear interpolation was used as a baseline for model comparison.

Samples from both versions of Block PixelCNNs 2×2 shown in Figure 4-1 seem to be equally good. The explanations can be that the initial image is very informative and that the short dependencies are easier to model once there is no need to pay the price for modeling the first group. Bilinear interpolation also demonstrates good results, however, the image is quite blurry and lacks fine details and object texture.

Samples from 4×4 models are not as good as from 2×2 models but some of the objects remain recognizable (e.g. the plane in the fifth row). Here the model with within-group interaction is a little bit less noisy and seems to preserve local structure slightly better than the model without interaction. For example, the body of an animal in the second row and the plane in the fifth row have sharper edges and seem to be less deformed in the right picture compared to the left one. Bilinear interpolation at this scale gets very blurry and even considerably worse than mean prediction which is also blurry. It is necessary to admit, though, that objects' shape can to some extent be seen in these two columns, in some cases even better than in sampled images (e.g. in the third row).

Finally, samples from 8×8 do not seem to preserve any information about objects and look more like purely generated patches with some local correlation. Similarly to previous scale, the right picture seems to be less noisy. Neither bilinear interpolation nor mean prediction can offer anything decent in this case. Such a poor performance is understandable if we take a look at the initial images for Block PixelCNNs 4×4 and 8×8, especially on the latter. One may argue that Multiscale PixelCNNs are able to upscale very decently even from 4×4 images, however, all of these models are conditioned on some external information, e.g. object keypoints. For a lot of PixelCNN-type models conditioning merely on labels already

	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 4.3: Bits per dimension on CIFAR-10 test set for different Block PixelCNN models. First column corresponds to bits-per-dim on the whole image. Second column contains bits-per-dim computed only on image pixels that correspond to group 1 in each Block PixelCNN model. Third columns contains bits-per-dim computed on all image pixels except pixels of group 1 in each model.

drastically improves the sample quality, even if the effect on the log-likelihood is negligible [Oord et al., 2016b, Reed et al., 2017, Salimans et al., 2017]. We thus assume that Block PixelCNN models could also be visually improved by introducing conditioning on some global information about the image.

Analysis of group interactions

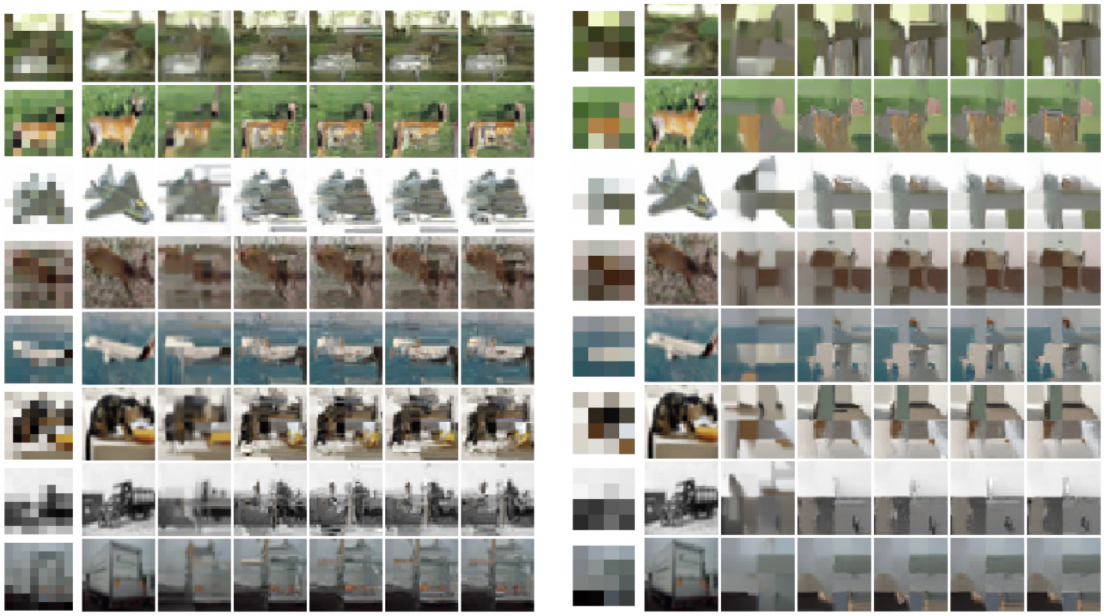
In this section we are exploring the dependencies among pixels of different groups and among pixels of the same group in the attempt to understand how block size affects the performance and if there is a preferable pixel structure that can be induced over the image.

It is reasonable to assume that groups defined over the image pixels are not equivalent in terms of the impact they have on the model. In particular, pixels of the first group play an important role of defining the image content and structure, especially during the pure image generation. Since all subsequent groups depend directly on information provided by the first group, while the pixels of the first group have no other pixels to rely on, the quality of modeling the first group of pixels is in direct relation to the overall model quality. All

Block PixelCNN models that have been considered so far in this thesis factorize pixels of the first group (as well as pixels of any other group) into independent non-overlapping subsets. The size of these subsets depends on the model's block size, the image spatial resolution and the presence of within-group interactions. This number gets smaller with the increase in the number of groups or, equivalently, with larger block sizes. Thus, it would be useful to consider not only the total log-likelihood of the model but also what part of it is contributed by the pixels of the first group and compare it to contribution from other groups.

Table 4.3 shows negative log-likelihood computed for each Block PixelCNN in bits per dimension in three modes: all pixels, pixels of the first group only and all pixels except the first group. The first thing that catches attention is that all models without within-group interaction have the same bits-per-dim accounted for the pixels of the first group. It is an interesting yet expected result. There are no previous groups to get context from, and there are no connections between pixels inside the group as well, so a price has to be paid for storing extra bits of information to model the pixels that cannot be modelled in any other way but remembering. The situation is completely different for pixels of the first group in models with within-group interaction. First group pixels in the 2×2 model have the lowest bits-per-dim which gets larger with an increase in the block size. This can be explained by the length of partial autoregressive connectivity inside the group. In particular, 2×2 model has 64 pixels of the first group in each image quadrant that are modelled autoregressively, while 8×8 model has only 4 such pixels in each quadrant. What is important, bits-per dim for remaining groups of pixels also grows with the number of groups, even though the weighted average is decreasing. This might either be a direct consequence of poor modeling of the first group in larger models or a signal that the shorter connections corresponding to smaller block sizes are locally more preferable than longer connections, even though *on average* bits-per-dim in the latter are lower.

Another angle to look from at the problem of choosing the right number of groups is image sampling. To compare the models with different block sizes during upscaling we used input images for other block sizes as inputs for current block size model. Figure 4-4, Figure 4-5 and Figure 4-6 show results of such cross-upsampling. In Figure 4-4 one can see how Block PixelCNN 2×2 with within-group interactions upscales 8×8 and 4×4 initial images. In



(a) upscaling inputs downsampled to 8×8 pixels

(b) upscaling inputs downsampled to 4×4 pixels

Figure 4-4: Upscaling results for random downsampled test images obtained by the Block PixelCNN 2×2 model with within-group interactions. In each image first column show model input; second - ground truth images; third - mean of predicted pixel distribution; fourth to seventh - random samples from predicted distributions.

both cases the generation procedure is divided into two steps: during the first step the model upsamples to 16×16 image by sampling from the first group in the necessary image positions, and during the second step an ordinary upscaling from 16×16 input image is performed. Moderate performance on the 8×8 is somewhat comparable to upscaling performance of 4×4 model with within-group interaction on the same input in Figure 4-2. Some objects look better upsampled by 4×4 model (e.g. the plane in the fifth row) while some look better upsampled by 2×2 model (e.g. the truck in the last row). As for the right figure, the 2×2 model failed completely to capture any content in the subsampled inputs. Obtained images represent 4 disjoint patches which is a direct result of factorization approach used in Block PixelCNNs where pixels of the same group are modelled autoregressively within each distinct quadrant, while the quadrants themselves stay independent from each other.

Similar artifacts can be found in the Figure 4-5(b) where 4×4 model upsamples the same 4×4 input. However, in this case the models demonstrates the attempts to generate some

object (e.g. in the second and fifth rows), even though some of these attempts fail completely as well (the last two rows). Two-step generation procedure described above which involves partial sampling from the first group of pixels is also used in this particular case. The left figure, however, corresponds to upscaling from a more informative 16×16 input, so only missing image pixels were sampled. Unexpectedly, the 4×4 model, which has better bits-per-dim score and which is expected to model the ranges of connectivity that are longer than in 2×2 model, demonstrates much noisier performance with occasional artifacts.

The same claims and even more can be made against the Figure 4-6(a) where 8×8 model upsamples 16×16 input image. The noisiness gets even worse which, again, seems strange since the model performance in terms of negative log-likelihood is the best out of all considered Block PixelCNNs. Figure 4-6(b) also suffers from extreme noisiness in addition to poor object reconstruction.

Insights obtained both from the log-likelihood tests in Table 4.3 and generation tests in Figure 4-4, Figure 4-5 and Figure 4-6 give a lot to think about. In particular, upscaling of 16×16 subsampled images rises the question why the models that are supposed to be stronger and be able to model long dependencies between image pixels visually perform worse than the model that captures mostly local dependencies. The possible answer may be that good log-likelihood does not always correspond to generation of plausible images [Theis et al., 2015]. The larger the block size gets, the more the model seems to overfit and benefit not from modeling local connections but rather from modeling global connections on average. From this point of view 2×2 block structure can be seen as a form of a model regularization and, as long as the pixels of the first group are modeled well enough, can potentially win from the visual perspective.

4.3.2 Comparison to the state of the art

Since Block PixelCNN with within-group interaction demonstrated better log-likelihood, we chose it for comparison with other generative image models. For simplicity of notation, we will no longer explicitly state that within-group interaction will be used.

The results for the CIFAR-10 and ImageNet datasets can be found in Table 4.4 and Table 4.5 respectively.

(a) upscaling inputs downsampled to 16×16 pixels

(b) upscaling inputs downsampled to 4×4 pixels

Figure 4-5: Upscaling results for random downsampled test images obtained by the Block PixelCNN 4×4 model with within-group interactions. Columns are organized in a manner similar to Figure 4-4.

(a) upscaling inputs downsampled to 16×16 pixels

(b) upscaling inputs downsampled to 8×8 pixels

Figure 4-6: Upscaling results for random downsampled test images obtained by the Block PixelCNN 8×8 model with within-group interactions. Columns are organized in a manner similar to Figure 4-4.

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, 2014a]	4.00
ConvDRAW [Gregor et al., 2016]	≤ 3.58
Real NVP [Dinh et al., 2016]	3.49
Block PixelCNN 2×2	$\leq$ **3.33**
Pyramid PixelCNN [Kolesnikov and Lampert, 2017]	3.32
Block PixelCNN 4×4	$\leq$ **3.25**
Block PixelCNN 8×8	$\leq$ **3.25**
PixelCNN [Oord et al., 2016c]	3.14
VAE with IAF [Kingma et al., 2016]	≤ 3.11
Gated PixelCNN [Oord et al., 2016b]	3.03
PixelRNN [Oord et al., 2016c]	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 4.4: Negative log-likelihood for different generative models on CIFAR-10 test set measured as bits per dimension.

CIFAR-10

As one can see in Table 4.4, except for VAE with IAF [Kingma et al., 2016], all models that demonstrated better log-likelihood results are either PixelCNN-type models or they use one for generating image pixels. Most importantly, they do not make any independence assumptions and, thus, are predictably stronger than the models that deliberately break some of the pixel connections. At the same time all Block PixelCNN models are considerably better than any other non-pixel-autoregressive generative model, except for VAE with IAF which uses generator $p(x|z_{1:T})$ conditioned on an autoregressive hierarchy of latent variables $z_{1:T}$ modelled by PixelCNNs (see Chapter 2).

Since the score for Multiscale PixelCNN on CIFAR-10 was not provided, this model currently cannot be compared to Block PixelCNN 2×2 which has the same group structure over the image pixels. However, the Pyramid PixelCNN which models upscaling with factor 2 and which is very similar to Multiscale PixelCNN, demonstrates results that are on par with Block PixelCNN 2×2 and worse than Block PixelCNN 4×4 and Block PixelCNN 8×8. It is escpecially important since neither of Block PixelCNNs considers pixels of the first group to

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

Table 4.5: Negative log-likelihood for different generative models on ImageNet 32×32 test set measured as bits per dimension.

be fully autoregressive, while both Multiscale PixelCNN and Pyramid PixelCNN use shallow full PixelCNN to model images of the base resolution. These base resolution images from which they start upscaling, in fact, correspond to pixels of the first group in Block PixelCNN models. In addition to that, Pyramid PixelCNN, similarly to Multiscale PixelCNN, uses multiple PixelCNN-type networks for upscaling, while all our Block PixelCNN models are learned using a single PixelCNN-type neural network.

ImageNet 32×32

Table 4.5 demonstrates bits per dimension obtained by generative image models which were trained on 32×32 version of ImageNet dataset. Similarly to CIFAR-10, our Block PixelCNN outperforms all non-pixel-autoregressive models but it is expectedly worse than PixelCNN and PixelRNN models. Its negative log-likelihood is also worse than that of Multiscale PixelCNN, however, some aspects need to be taken into consideration. First of all, as it has been mentioned above, Block PixelCNN in its current version factorizes pixels of the first group in the same way it factorizes pixels in other groups. It means that there is a limited autoregressive dependency between pixels of the first group in each of the non-overlapping quadrants of the image. On the contrary, Multiscale PixelCNN starts from a fully autoregressive model over the pixels of the first group followed by upscaling model that breaks some dependencies in subsequent groups. As it has been demonstrated in Table 4.3, pixels of the first group contribute significantly to the total log-likelihood estimation. It implies that the log-likelihood of Block PixelCNNs can be further improved by inducing a fully-dependant autoregressive structure over the pixels of the first group. Secondly, results reported in

Table 4.5 for the Multiscale PixelCNN were obtained by learning a class-conditional image model. While in another work [Oord et al., 2016b] the authors claimed that they did not observe "big differences" in log-likelihood results between unconditional and class-conditional Gated PixelCNN models on ImageNet, it is not clear what was the exact difference then and how big would be the difference for Multiscale PixelCNN now that some dependencies get broken. Finally, it is not clear what was the hardware setup used for training this model. In the paper [Oord et al., 2016b] that preceded Multiscale PixelCNN and has common authors, it was stated that 32 GPUs were used with only 4 training examples per GPU (minibatch of size 128 in total). In this work, due to limited amount of available hardware, Block PixelCNN was trained on a single GPU and with minibatch size 32. Smaller minibatches led to unstable training which can be explained by high variability in data that causes poor gradients if the number of processed objects per optimization step is too small. As a result, the number of parameters in the ImageNet model was even smaller than the number of parameters in any of the CIFAR-10 models.

Discussion

As one can notice, bits per dimension for all Block PixelCNN models in Table 4.4 and Table 4.5 are mentioned with the sign "$\leq$". This was done because all these results reflect capabilities of the proposed models but not their full potential. Due to the limiting amount of available GPUs and necessity to train a lot of models simultaneously, none of the mentioned Block PixelCNNs was trained to the state when it would have started to overfit noticeably. All of the models discussed in this chapter continue to improve their results, very slowly but steadily. For reference, on CIFAR-10 PixelCNN++, that was used for implementation of Block PixelCNN, requires 10 hours of training on 8 Maxwell TITAN X GPUs to achieve 3.0 bits per dimension and 5 more days to fall by the remaining 0.08.

Several steps for improving the current version of Block PixelCNN could be taken to boost its performance. One of them is complete autoregressive modeling of pixels of the first group since it is the most expensive part of the proposed generative model, as has been shown in Table 4.3. This step is crucial for a pure generation task since broken dependencies within the first group will result in generation of uncorrelated image patches which was already seen in

Figure 4-4(b). For improvement of the quality of the generated samples conditional version of Block PixelCNN could be trained, as it seems to benefit other PixelCNN-type models a lot [Oord et al., 2016b, Salimans et al., 2017]. Architecture optimization, especially in the implementation of the proposed block dilated convolutions, could possibly improve the training of the proposed models considerably, since in the current implementation there are a lot of extra computations and unused intermediate feature maps due to internal paddings and shifting of feature maps. Finally, another step could be to use multiple GPUs for training, especially for ImageNet dataset, since preliminary experiments demonstrate positive responses from the models after an increase in the number of network parameters.

Chapter 5

Conclusion and future work

In this thesis a Block PixelCNN model has been proposed that combines approaches of Multiscale PixelCNN and PixelCNN models. The main reason the development of such a model became a topic of interest was that the regular block structure induced over the image pixels by Multiscale PixelCNN could not be easily reformulated in terms of the original PixelCNN, where all pixels were modelled as a sequence in a raster scan order. This has led to a gap between these models. The main goal we pursued during this research was to obtain a flexible educational model that would allow active exploration of the properties of PixelCNN-type models with different connectivity lengths and would possibly give us an insight whether there is a preferable structure over the image pixels.

Resulting Block PixelCNN model has the desired property of flexibility, both in terms of group structure and training simplicity. This allowed us to consider connectivity patterns structured into regular square blocks of the sizes 2×2, 4×4 and 8×8. Other block sizes can also be considered by this model but they were left out of scope due to limited time and hardware. The trained models demonstrated competitive log-likelihood on CIFAR-10 and downsized ImageNet datasets. Generation capabilities of Block PixelCNNs were tested on the upscaling task. Visual quality of the samples generated by Block PixelCNNs of different block sizes from different input resolutions, in addition to analysis of contribution to log-likelihood by the groups for different block sizes, gave us some insights into how the number of groups and, as a result, the length of pixel dependency affect the model performance. At this point it seems possible to conclude that shorter local dependencies allow for a more

accurate and detailed image generation, as long as image context and structure is captured well enough by the pixels of the first group. Log-likelihood performance of such model at the same time might be good enough but not the best comparing to other models. Multiscale approach in Multiscale PixelCNN and Pyramid PixelCNN that also exploits these short-range dependencies but learns them separately for each image scale seems from this point of view to be a quite efficient strategy, especially since at the lowest resolution the image is modelled in a fully autoregressive manner at a relatively cheap price.

Current version of Block PixelCNN can be further improved taking into consideration the outcome of the experiments and the insights gained in this research. It would be interesting to see how bits-per-dim on the first group and in total would change if the first group was modelled without any independence assumptions, would it change the generation performance for models with larger number of groups. Another direction for model modification could be an introduction of global image information to condition on, to see whether it could make up for broken dependencies in the first group. These two approaches can be applied separately and compared to each other and after that combined to get a more powerful model. Finally, a number of improvements could be made in the implementation of the model to optimize the training performance. All these ideas for further model development are left for future work.

Bibliography

[Abadi et al., 2015] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

[Chen et al., 2016] 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 et al., 2014] Dinh, L., Krueger, D., and Bengio, Y. (2014). Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*.

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

[Germain et al., 2015] Germain, M., Gregor, K., Murray, I., and Larochelle, H. (2015). Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Gregor et al., 2016] 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.

[Gregor et al., 2015] 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*.

[Gulrajani et al., 2016] Gulrajani, I., Kumar, K., Ahmed, F., Taiga, A. A., Visin, F., Vazquez, D., and Courville, A. (2016). Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Hinton and Van Camp, 1993] Hinton, G. E. and Van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 5–13. ACM.

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

[Honkela and Valpola, 2004] Honkela, A. and Valpola, H. (2004). Variational learning and bits-back coding: an information-theoretic view to bayesian learning. *IEEE Transactions on Neural Networks*, 15(4):800–810.

[Kingma and Ba, 2014] Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma et al., 2016] 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.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *ArXiv e-prints*.

[Kolesnikov and Lampert, 2017] 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.

[Krizhevsky and Hinton, 2009] Krizhevsky, A. and Hinton, G. (2009). Learning multiple layers of features from tiny images.

[Larochelle and Murray, 2011] Larochelle, H. and Murray, I. (2011). The neural autoregressive distribution estimator. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 29–37.

[Oord et al., 2016a] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016a). Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*.

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

[Oord et al., 2016c] Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. (2016c). Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*.

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

[Oord and Schrauwen, 2014b] Oord, A. v. d. and Schrauwen, B. (2014b). The student-t mixture as a natural image patch prior with application to image compression. *Journal of Machine Learning Research*, 15(1):2061–2086.

[Ramachandran et al., 2017] Ramachandran, P., Paine, T. L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., Hasegawa-Johnson, M. A., Campbell, R. H., and Huang, T. S. (2017). Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*.

[Reed et al., 2017] 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*.

[Rezende and Mohamed, 2015] Rezende, D. J. and Mohamed, S. (2015). Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[Russakovsky et al., 2014] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Li, F. (2014). Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575.

[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. *arXiv preprint arXiv:1701.05517*.

[Shannon, 2001] Shannon, C. E. (2001). A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55.

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

[Theis et al., 2015] Theis, L., Oord, A. v. d., and Bethge, M. (2015). A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*.

[Uria et al., 2016] Uria, B., Côté, M.-A., Gregor, K., Murray, I., and Larochelle, H. (2016). Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37.

[Yu and Koltun, 2015] Yu, F. and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*.