Emerging Convolutions for Generative Normalizing Flows

Emiel Hoogeboom [1 2]   Rianne van den Berg [1]   Max Welling [1 3]

Abstract

Generative flows are attractive because they admit exact likelihood optimization and efficient image synthesis. Recently, Kingma & Dhariwal (2018) demonstrated with Glow that generative flows are capable of generating high quality images. We generalize the 1×1 convolutions proposed in Glow to *invertible* $d \times d$ convolutions, which are more flexible since they operate on both channel and spatial axes. We propose two methods to produce invertible convolutions that have receptive fields identical to standard convolutions: *Emerging* convolutions are obtained by chaining specific autoregressive convolutions, and *periodic* convolutions are decoupled in the frequency domain. Our experiments show that the flexibility of $d \times d$ convolutions significantly improves the performance of generative flow models on galaxy images, CIFAR10 and ImageNet.

1. Introduction

Generative models aim to learn a representation of the data $p(x)$, in contrast with discriminative models that learn a probability distribution of labels given data $p(y|x)$. Generative modeling may be used for numerous applications such as anomaly detection, denoising, inpainting, and super-resolution. The task of generative modeling is challenging, because data is often very high-dimensional, which makes optimization and choosing a successful objective difficult.

Generative models based on normalizing flows (Rippel & Adams, 2013) have several advantages over other generative models: i) They optimize the log likelihood of a continuous distribution exactly, as opposed to Variational Auto-Encoders (VAEs) (Kingma & Welling, 2014; Rezende et al., 2014) which optimize a lower bound to the log-likelihood. ii) Drawing samples has a computational cost comparable to inference, in contrast with PixelCNNs (Van Oord et al., 2016). iii) Generative flows also have the potential for huge
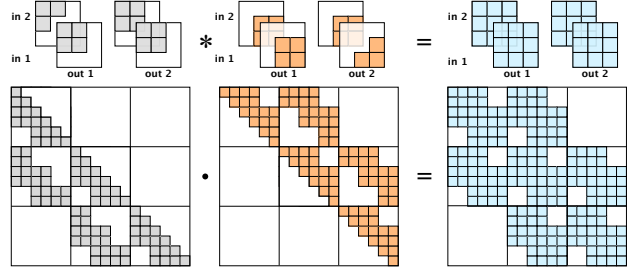
Preprint. Work in progress

Figure 1. Illustration of a square emerging convolution. The input has spatial dimensions equal to 3×3, and two channels. Convolutions use one-pixel-wide zero padding at each border. Two consecutive square autoregressive convolutions with filters k_2 and k_1 have a receptive field identical to a standard convolution, with filter $k_2 *_l k_1$, where $*_l$ denotes a convolution layer. These operations are equivalent to the multiplication of matrices $K_2 \cdot K_1$ and a vectorized input signal $\vec{x}$. Since the filters are learned decomposed, the Jacobian determinant and inverse are straightforward to compute.

memory savings, because activations necessary in the backward pass can be obtained by computing the inverse of layers (Gomez et al., 2017; Li & Grathwohl, 2018).

The performance of density estimation models can be largely attributed to Masked Autoregressive Flows (MAFs) (Papamakarios et al., 2017) and the coupling layers introduced in NICE and RealNVP (Dinh et al., 2014; 2017). MAFs contain flexible autoregressive transformations, but are computationally expensive to invert, which is a disadvantage for sampling high-dimensional data. Coupling layers transform a subset of the dimensions of the data, parameterized by the remaining dimensions. The inverse of coupling layers is straightforward to compute, which makes them suitable for generative flows. However, since coupling layers can only operate on a subset of the dimensions of the data, they may be limited in flexibility.

To improve their effectiveness, coupling layers are alternated with less complex transformations that do operate on all dimensions of the data. Dinh et al. (2017) use a fixed channel permutation in Real NVP, and Kingma & Dhariwal (2018) utilize learnable 1×1 convolutions in Glow.

However, 1×1 convolutions suffer from limited flexibility, and using standard convolutions is not straightforward as they are very computationally expensive to invert. We

propose two methods to obtain easily invertible and flexible convolutions: *emerging* and *periodic* convolutions. Both of these convolutions have receptive fields identical to standard convolutions, resulting in flexible transformations over both the channel *and* spatial axes.

The structure of an emerging convolution is depicted in Figure 1, where the top depicts the convolution filters, and the bottom shows the equivalent matrices of these convolutions. Two autoregressive convolutions are chained to obtain an emerging receptive field identical to a standard convolution. Empirically, we find that replacing 1×1 convolutions with the generalized invertible convolutions produces significantly better results on galaxy images, CIFAR10 and ImageNet, even when correcting for the increase in parameters.

In addition to invertible convolutions, we also propose a QR decomposition for 1×1 convolutions, which resolves flexibility issues of the PLU decomposition proposed by Kingma & Dhariwal (2018).

The main contributions of this paper are: 1) Invertible emerging convolutions using autoregressive convolutions. 2) Invertible periodic convolutions using decoupling in the frequency domain. 3) Numerically stable and flexible 1×1 convolutions parameterized by a QR decomposition. 4) An accelerated inversion module for autoregressive convolutions.

2. Background

2.1. Change of variables formula

Consider a bijective map between variables x and z. The likelihood of the variable x can be written as the likelihood of the transformation $z = f(x)$ evaluated by p_Z, using the change of variables formula:

$$p_X(x) = p_Z(z) \left| \frac{\partial z}{\partial x} \right| \;\; ; \;\; z = f(x). \qquad (1)$$

The complicated probability density $p_X(x)$ is equal to the probability density $p_Z(z)$ multiplied by the Jacobian determinant, where p_Z is chosen to be tractable. The function f can be learned, but the choice of f is constrained by two practical issues: Firstly, the Jacobian determinant should be tractable. Secondly, to draw samples from p_X, the inverse of f should be tractable.

2.1.1. COMPOSITION OF FUNCTIONS

A sequence composed of several applications of the change of variables formula is often referred to as a normalizing flow (Deco & Brauer, 1995; Tabak et al., 2010; Tabak & Turner, 2013; Rezende & Mohamed, 2015). Let $\{h_l\}_{l=1}^L$ be the intermediate representations produced by the layers

of a neural network, where $z = h_L$ and $h_0 = x$. The log-likelihood of x is written as the log-likelihood of z, and the summation of the log Jacobian determinant of each layer:

$$\log p_X(x) = \log p_Z(z) + \sum_{l=1}^L \log \left| \frac{\partial h_l}{\partial h_{l-1}} \right|. \qquad (2)$$

2.1.2. DEQUANTIZATION

We will evaluate our methods with experiments on image datasets, where pixels are discrete-valued from 0 to 255. Since generative flows are continuous density models, they may trivially place infinite mass on discretized bin locations. Therefore, we use the definition of Theis et al. (2016) that defines the relation between a discrete model $\hat{p}(\hat{x})$ and continuous model $p(x)$ as an integration over bins: $\hat{p}(\hat{x}) \equiv \int_{[0,1)^d} p(\hat{x} + u) du$, where $x = \hat{x} + u$. They further derive a lowerbound to optimize this model with Jensen's inequality, resulting in additive uniform noise for the integer valued pixels from the data distribution $\mathcal{D}$:

$$\mathbb{E}_{\hat{x} \sim \mathcal{D}} \left[\log \hat{p}(\hat{x}) \right] = \mathbb{E}_{\hat{x} \sim \mathcal{D}} \left[\log \int_{[0,1)^d} p(\hat{x} + u) du \right] \qquad (3)$$
$$\geq \mathbb{E}_{\hat{x} \sim \mathcal{D}, u \sim \mathcal{U}[0,1)^d} \left[\log p(\hat{x} + u) \right].$$

2.2. Generative flows

Generative flows are bijective functions, often structured as deep learning layers, that are designed to have tractable Jacobian determinants and inverses. An overview of several generative flows is provided in Table 1, and a description is given below:

Coupling layers (Dinh et al., 2017) split the input in two parts. The output is a combination of a copy of the first half, and a transformation of the second half, parametrized by the first part. As a result, the inverse and Jacobian determinant are straightforward to compute.

Actnorm layers (Kingma & Dhariwal, 2018) are data dependent initialized layers with scale and translation parameters. They are initialized such that the distribution of activations has mean zero and standard deviation one. Actnorm layers improve training stability and performance.

1×1 **Convolutions** (Kingma & Dhariwal, 2018) are easy to invert, and can be seen as a generalization of the permutation operations that were used by Dinh et al. (2017). 1×1 convolutions improve the effectiveness of the coupling layers.

Table 1. The definition of several generative normalizing flows. All flow functions have an inverse and determinant that are straightforward to compute. The height h, width w and number of channels n_c of an output remains identical to the dimensions of the input. The symbols $\odot$ and $/$ denote element-wise multiplication and division. Input and output may be denoted as tensors $\mathbf{x}$ and $\mathbf{z}$ with dimensions $n_c \times h \times w$. The inputs and outputs may be denoted as one-dimensional vectors $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$ with dimension $n_c \cdot h \cdot w$. Input and output in frequency domain are denoted with $\hat{\mathbf{x}}$ and $\hat{\mathbf{z}}$, with dimensions $n_c \times h \times w$, where the last two components denote frequencies.

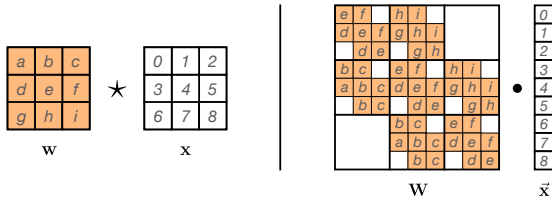| Generative Flow	Function	Inverse	Log Determinant
Actnorm	$\vec{\mathbf{z}} = \vec{\mathbf{x}} \odot \vec{\boldsymbol{\gamma}} + \vec{\boldsymbol{\beta}}$	$\vec{\mathbf{x}} = (\vec{\mathbf{z}} - \vec{\boldsymbol{\beta}})/\vec{\boldsymbol{\gamma}}$	$\text{sum}(\log |\vec{\boldsymbol{\gamma}}|)$
Affine coupling	$[\vec{\mathbf{x}}_a, \vec{\mathbf{x}}_b] = \vec{\mathbf{x}}$ $\vec{\mathbf{z}}_a = \vec{\mathbf{x}}_a \odot f(\vec{\mathbf{x}}_b) + g(\vec{\mathbf{x}}_b)$ $\vec{\mathbf{z}} = [\vec{\mathbf{z}}_a, \vec{\mathbf{x}}_b]$	$[\vec{\mathbf{z}}_a, \vec{\mathbf{z}}_b] = \vec{\mathbf{z}}$ $\vec{\mathbf{z}}_a = (\vec{\mathbf{z}}_a - g(\vec{\mathbf{z}}_b))/f(\vec{\mathbf{z}}_b)$ $\vec{\mathbf{x}} = [\vec{\mathbf{z}}_a, \vec{\mathbf{x}}_b]$	$\text{sum}(\log |f(\vec{\mathbf{x}}_b)|)$
1×1 Conv	$\forall ij : \mathbf{z}_{:,ij} = \mathbf{W}\mathbf{x}_{:,ij}$	$\forall ij : \mathbf{x}_{:,ij} = \mathbf{W}^{-1}\mathbf{z}_{:,ij}$	$h \cdot w \cdot \log |\det \mathbf{W}|$
Emerging Conv	$\mathbf{k} = \mathbf{w}_1 \odot \mathbf{m}_1$ $\mathbf{g} = \mathbf{w}_2 \odot \mathbf{m}_2$ $\mathbf{z} = \mathbf{k} \star_l (\mathbf{g} \star_l \mathbf{x})$	$\forall t : \vec{y}_t = (\vec{z}_t - \sum_{i=t+1} G_{t,i}\,\vec{y}_i)/G_{t,t}$ $\forall t : \vec{x}_t = (\vec{y}_t - \sum_{i=1}^{t-1} K_{t,i}\,\vec{x}_i)/K_{t,t}$	$\sum_c \log |\mathbf{k}_{c,c,m_y,m_x}\, \mathbf{g}_{c,c,m_y,m_x}|$
Periodic Conv	$\forall uv : \hat{\mathbf{z}}_{:,uv} = \hat{\mathbf{W}}_{uv}\hat{\mathbf{x}}_{:,uv}$	$\forall uv : \hat{\mathbf{x}}_{:,uv} = \hat{\mathbf{W}}_{uv}^{-1}\hat{\mathbf{z}}_{:,uv}$	$\sum_{u,v} \log |\det \hat{\mathbf{W}}_{uv}|$

Figure 2. Illustration of a standard 3×3 convolution layer with one input and output channel. The spatial input size is 3×3, and the input values are $\{0, 1, \ldots, 8\}$. The convolution uses one-pixel-wide zero padding at each border, and the filter has parameters $\{a, b, \ldots, i\}$. Left: the convolution $\mathbf{w} \star \mathbf{x}$. Right: the matrix multiplication $\mathbf{W} \cdot \vec{\mathbf{x}}$ which produces the equivalent result.

Figure 3. A standard 3×3 convolution layer with two input and output channels. The input is 3×3 spatially, and has two channels. The convolution uses one-pixel-wide zero padding at each border. Left: the convolution filter $\mathbf{w}$. Right: the matrix $\mathbf{W}$ which produces the equivalent result when multiplied with a vectorized input.

2.3. Convolutions

Generally, a convolution layer[1] with filter $\mathbf{w}$ and input $\mathbf{x}$ is equivalent to the multiplication of $\mathbf{W}$, a $h\,w\,n_{c_{out}} \times h\,w\,n_{c_{in}}$ matrix, and a vectorized input $\vec{\mathbf{x}}$. An example of a single channel convolution and its equivalent matrix is depicted in Figure 2. The signals $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$ are indexed as $t = i + w \cdot j$, where i is the width index, j is the height index, and w is the total width. Note that the matrix $\mathbf{W}$ becomes sparser as the image dimensions grow and that the parameters of the filter $\mathbf{w}$ occur repeatedly in the matrix $\mathbf{W}$. A two-channel convolution is visualized in Figure 3, where we have omitted parameters inside filters to avoid clutter. Here, $\vec{\mathbf{x}}$ and $\vec{\mathbf{z}}$ are vectorized using indexing $t = c + n_c \cdot i + (n_c \cdot w) \cdot j$, where c denotes the channel index and n_c the number of channels.

Using standard convolutions as a generative flow is ineffi-

---

[1]Note that in deep learning frameworks, convolutions are often actually cross-correlations. In our equations $\star$ denotes a cross-correlation and $*$ denotes a convolution. In addition, a convolution layer is usually implemented as an aggregation of cross-correlations, i.e. a cross-correlation layer. We denote such a layer with the symbol $\star_l$. In the main text we may omit these details, and these operations are referred to as convolutions.
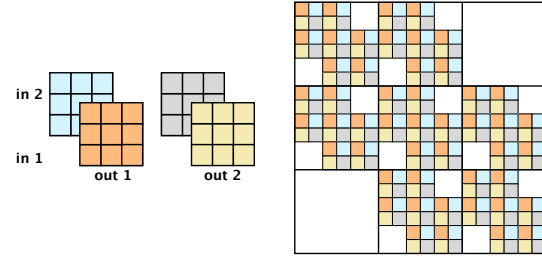
cient. The determinant and inverse can be obtained naïvely by operating directly on the corresponding matrix, but this would be very expensive, corresponding to computational complexity $\mathcal{O}(h^3 \cdot w^3 \cdot n_c^3)$.

2.4. Autoregressive Convolutions

Autoregressive convolutions have been widely used in the field of normalizing flows (Germain et al., 2015; Kingma et al., 2016) because it is straightforward to compute their Jacobian determinant. Although there exist autoregressive convolutions with different input and output dimensions, we let $n_{c_{out}} = n_{c_{in}}$ for invertibility. In this case, autoregressive convolutions can be expressed as a multiplication between a triangular weight matrix and a vectorized input.

In practice, a filter $\mathbf{k} = \mathbf{w} \odot \mathbf{m}$ is constructed from weights $\mathbf{w}$ and a binary mask $\mathbf{m}$ that enforces the autoregressive structure (see Figure 4). The convolution with the masked filter is autoregressive without the need to mask inputs, which allows parallel computation of the convolution layer:

$$\mathbf{z} = \mathbf{k} \star_l \mathbf{x}, \tag{4}$$

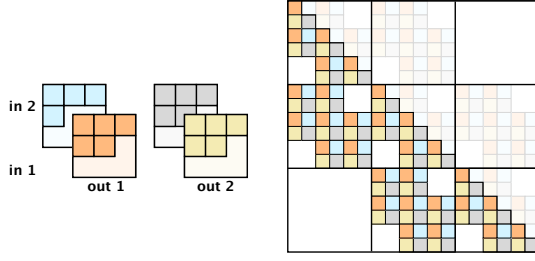where $\star_l$ denotes a convolution layer[1]. The matrix multipli-

Figure 4. An *autoregressive* 3×3 convolution layer with two input and output channels. The input has spatial dimensions 3×3, and two channels. The convolution uses one-pixel-wide zero padding at each border. Left: the autoregressive convolution filter **k**. Right: the matrix **K** which produces the equivalent result on a vectorized input. Note that the equivalent matrix is *triangular*.

cation $\vec{z} = \mathbf{K}\vec{x}$ produces the equivalent result, where $\vec{x}$ and $\vec{z}$ are the vectorized signals, and **K** is a sparse triangular matrix constructed from **k** (see Figure 4). The Jacobian is triangular by design and its determinant can be computed in $\mathcal{O}(n_c)$ since it only depends on the diagonal elements of the matrix **K**:

$$\log \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right| = h \cdot w \sum_{c}^{n_c} \log \left| k_{c,c,m_y,m_x} \right|, \qquad (5)$$

where index c denotes the channel and (m_y, m_x) denotes the spatial center of the filter. The inverse of an autoregressive convolution can theoretically be computed using $\vec{x} = \mathbf{K}^{-1}\vec{z}$. In reality this matrix is large and impractical to invert. Since **K** is triangular, the solution for $\vec{x}$ can be found through forward substitution:

$$\vec{x}_t = \frac{\vec{z}_t - \sum_{i=1}^{t-1} K_{t,i} \cdot \vec{x}_i}{K_{t,t}}. \qquad (6)$$

The inverse can be computed by sequentially traversing through the input feature map in the imposed autoregressive order. The computational complexity of the inverse is $\mathcal{O}(h \cdot w \cdot n_c^2)$ and computation can be parallelized across examples in the minibatch.

3. Method

We present two methods to generalize 1×1 convolutions to invertible $d \times d$ convolutions, improving the flexibility of generative flow models. Emerging convolutions are obtained by chaining autoregressive convolutions (section 3.1), and periodic convolutions are decoupled in frequency domain (section 3.2). In section 3.3, we provide a stable and flexible parameterization for invertible 1×1 convolutions.

3.1. Emerging convolutions

Although autoregressive convolutions are invertible, their transformation is restricted by the imposed autoregressive
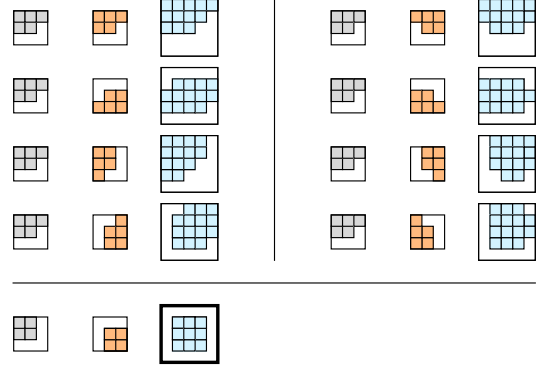

Figure 5. Achievable emerging receptive fields that consist of two distinct auto-regressive convolutions. Grey areas denote the first convolution filter and orange areas denote the second convolution filter. Blue areas denote the emerging receptive field, and white areas are masked. The convolution in the bottom row is a special case, which has a receptive field identical to a standard convolution.

order, enforced through masking of the filters (as depicted in Figure 4). To alleviate this restriction, we propose emerging convolutions, which are more flexible and nevertheless invertible. Emerging convolutions are obtained by chaining specific autoregressive convolutions, invertible via the autoregressive inverses. To some extent this resembles the combination of stacks used to resolve the blind spot problem in conditional image modeling with PixelCNNs (van den Oord et al., 2016), with the important difference that we do not constrain the resulting convolution itself to be autoregressive.

The emerging receptive field can be controlled by chaining autoregressive convolutions with variations in the imposed order. A collection of achievable receptive fields for emerging convolutions is depicted in Figure 5, based on commonly used autoregressive masking.

The autoregressive inverse requires the solution to a sequential problem, and as a result, it inevitably suffers some additional computational cost. In emerging convolutions we minimize this cost through the use of an accelerated parallel inversion module, implemented in Cython, and by maintaining relatively small dimensionality in the emerging convolutions compared to the internal size of coupling layers.

3.1.1. SQUARE EMERGING CONVOLUTIONS

Deep learning applications tend to use square filters, and libraries are specifically optimized for these shapes. Since most of the receptive fields in Figure 5 are unusually shaped, these would require masking to fit them in rectangular arrays, leading to unnecessary computation.

However, there is a special case in which the emerging re-

ceptive field of two specific autoregressive convolutions is identical to a standard convolution. These *square* emerging convolutions can be obtained by combining off center square convolutions, depicted in the bottom row of Figure 5 (also Figure 1). Our square emerging convolution filters are more efficient since they require fewer masked values in rectangular arrays.

There are two approaches to efficiently compute square emerging convolutions during optimization and density estimation: either a $d \times d$ emerging convolution is expressed as two smaller consecutive $\frac{d+1}{2} \times \frac{d+1}{2}$ convolutions. Alternatively, the order of convolution can be changed: first the smaller $\frac{d+1}{2}$ filters (k_2 and k_1) are convolved to obtain a single equivalent convolution filter. Then, the output of the emerging convolution is obtained by convolving the equivalent filter, $k = k_2 * k_1$, with the feature map f:

$$k_2 \star (k_1 \star f) = (k_2 * k_1) \star f. \tag{7}$$

This equivalence follows from the associativity of convolutions and the time reversal of real discrete signals in cross-correlations.

When $d = 1$, two autoregressive convolutions simplify to an LU decomposed 1×1 convolution. To ensure that emerging convolutions are flexible, we use emerging convolutions that consists of: a single 1×1 convolution, and two square autoregressive convolutions with different masking as depicted in the bottom row of Figure 1. Again, the individual convolutions may all be combined into a single emerging convolution filter using the associativity of convolutions (Equation 7).

3.2. Invertible Periodic Convolutions

In some cases, data may be periodic or boundaries may contain roughly the same values. In these cases it may be advantageous to use invertible *periodic* convolutions, which assume that boundaries wrap around. When computed in the frequency domain, this alternative convolution has a tractable determinant Jacobian and inverse. The method leverages the convolution theorem, which states that the Fourier transform of a convolution is given by the element-wise product of the Fourier transformed signals. Specifically, the input and filter are transformed using the Discrete Fourier Transform (DFT) and multiplied element-wise, after which the inverse DFT is taken. By considering the transformation in the frequency domain, the computational complexity of the determinant Jacobian and the inverse are considerably reduced. In contrast with emerging convolutions, which are very specifically parameterized, the filters of periodic convolutions are completely unconstrained.

A standard convolution layer in deep learning is conventionally implemented as an aggregation of *cross-correlations* for every output channel. The convolution layer with input
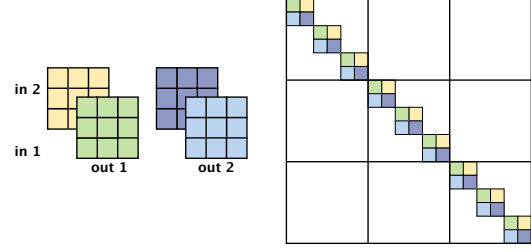

Figure 6. Visualization of a *periodic* 3×3 convolution layer in the *frequency domain*. The input and output have height 3, width 3 and channels 2. The shape of the filter in the frequency domain determined by the shape of the image, which is also 3×3 spatially in this specific example. Left: the convolution filter transformed to the frequency domain $\hat{\mathbf{w}}$. Right: the matrix $\hat{\mathbf{W}}$ in the frequency domain, which produces the equivalent result on a vectorized input. The equivalent matrix in the frequency domain is *partitioned*.

$\mathbf{x}$ and filter $\mathbf{w}$ outputs the feature map $\mathbf{z} = \mathbf{w} \star_l \mathbf{x}$, which is computed as:

$$\mathbf{z}_{c_{out}} = \sum_{c_{in}} \mathbf{w}_{c_{out}, c_{in}} \star \mathbf{x}_{c_{in}}. \tag{8}$$

Let $\mathcal{F}(\cdot)$ denote the Fourier transform and let $\mathcal{F}^{-1}(\cdot)$ denote the inverse Fourier transform. The Fourier transform can be moved inside the channel summation, since it is distributive over addition. Let $\hat{\mathbf{z}}_{c_{out}} = \mathcal{F}(\mathbf{z}_{c_{out}})$, $\hat{\mathbf{w}}_{c_{out}, c_{in}} = \mathcal{F}(\mathbf{w}^*_{c_{out}, c_{in}})$ and $\hat{\mathbf{x}}_{c_{in}} = \mathcal{F}(\mathbf{x}_{c_{in}})$, which are indexed by frequencies u and v. Because a convolution differs from a cross-correlation by a time reversal for real signals, let $\mathbf{w}^*_{c_{out}, c_{in}}$ denote the reflection of filter $\mathbf{w}_{c_{out}, c_{in}}$ in both spatial directions. Using these definitions, each cross-correlation is written as an element-wise multiplication in the frequency domain:

$$\hat{\mathbf{z}}_{c_{out}} = \sum_{c_{in}} \hat{\mathbf{w}}_{c_{out}, c_{in}} \odot \hat{\mathbf{x}}_{c_{in}}, \tag{9}$$

which can be written as a sum of products in scalar form:

$$\hat{z}_{c_{out}, uv} = \sum_{c_{in}} \hat{w}_{c_{out}, c_{in}, uv} \cdot \hat{x}_{c_{in}, uv}. \tag{10}$$

The summation of multiplications can be reformulated as a matrix multiplication over the channel axis by viewing the output $\hat{\mathbf{z}}_{:,uv}$ at frequency u, v as a multiplication of the matrix $\hat{\mathbf{W}}_{uv} = \hat{\mathbf{w}}_{:,:,u,v}$ and the input vector $\hat{\mathbf{x}}_{:,uv}$:

$$\hat{\mathbf{z}}_{:,uv} = \hat{\mathbf{W}}_{uv} \hat{\mathbf{x}}_{:,uv}. \tag{11}$$

The matrix $\hat{\mathbf{W}}_{uv}$ has dimensions $c_{out} \times c_{in}$, the input $\hat{\mathbf{x}}_{:,uv}$ and output $\hat{\mathbf{z}}_{:,uv}$ are vectors with dimension c_{in} and c_{out}. The output in the original domain $\mathbf{z}_{c_{out}}$ can simply be retrieved by taking the inverse Fourier transform, $\mathcal{F}^{-1}(\hat{\mathbf{z}}_{c_{out}})$.

The perspective of matrix multiplication in the frequency domain decouples the convolution transformation (see Figure 6). Therefore, the log determinant of a periodic convolution layer is equal to the sum of determinants of individual frequency components:

$$\log \left| \det \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right| = \log \left| \det \frac{\partial \hat{\mathbf{z}}}{\partial \hat{\mathbf{x}}} \right| = \sum_{u,v} \log \left| \det \hat{\mathbf{W}}_{uv} \right|. \quad (12)$$

The determinant remains unchanged by the Fourier transform and its inverse, since these are unitary transformations. The inverse operation requires an inversion of the matrix $\hat{\mathbf{W}}_{uv}$ for every frequency u, v:

$$\hat{\mathbf{x}}_{:,uv} = \hat{\mathbf{W}}_{uv}^{-1} \hat{\mathbf{z}}_{:,uv}. \quad (13)$$

The solution of $\mathbf{x}$ in the original domain is obtained by the inverse Fourier transform, $\mathbf{x}_{c_{in}} = \mathcal{F}^{-1}(\hat{\mathbf{x}}_{c_{in}})$, for every channel c_{in}.

Recall that a standard convolution layer is equivalent to a matrix multiplication with a $h\,w\,n_{c_{out}} \times h\,w\,n_{c_{in}}$ matrix, where we let $n_{c_{out}} = n_{c_{in}}$ for invertibility. The Fourier transform decouples the transformation of the convolution layer at each frequency, which divides the computation into $h \cdot w$ separate matrix multiplications with $n_c \times n_c$ matrices. Therefore, the computational cost of the determinant is reduced from $\mathcal{O}(h^3 \cdot w^3 \cdot n_c^3)$ to $\mathcal{O}(h \cdot w \cdot n_c^3)$ in the frequency domain, and computation can be parallelized since the matrices are independent across frequencies and independent of the data. Furthermore, the inverse matrices $\hat{\mathbf{W}}_{uv}^{-1}$ only need to be computed once after the model has converged, which reduces the inverse convolution to an efficient matrix multiplication with computational complexity[2] $\mathcal{O}(h \cdot w \cdot n_c^2)$.

3.3. QR 1×1 convolutions

Standard 1×1 convolutions are flexible but may be numerically unstable during optimization, causing crashes in the training procedure. Kingma & Dhariwal (2018) propose to learn a PLU decomposition, but since the permutation matrix $\mathbf{P}$ is fixed during optimization, their flexibility is limited.

In order to resolve the stability issues while retaining the flexibility of the transformation, we propose to use a *QR* decomposition. Any real square matrix can be decomposed into a multiplication of an orthogonal and a triangular matrix. In a similar fashion to the PLU parametrization, we stabilize the decomposition by choosing $\mathbf{W} = \mathbf{Q}(\mathbf{R} + \text{diag}(\mathbf{s}))$, where $\mathbf{Q}$ is orthogonal, $\mathbf{R}$ is strictly triangular, and elements in $\mathbf{s}$ are nonzero. Any $n \times n$ orthogonal matrix $\mathbf{Q}$ can be constructed from at most n Householder reflections through

---

[2]The inverse also incurs some overhead due to the Fourier transform of the feature maps which corresponds to a computational complexity $\mathcal{O}(h \cdot w \cdot n_c \cdot \log h\,w)$.

$\mathbf{Q} = \mathbf{Q}_1 \mathbf{Q}_2 \ldots \mathbf{Q}_n$, where $\mathbf{Q}_i$ is a Householder reflection:

$$\mathbf{Q}_i = \mathbf{I} - 2 \frac{\mathbf{v}_i \mathbf{v}_i^T}{\mathbf{v}_i^T \mathbf{v}_i}. \quad (14)$$

$\{\mathbf{v}_i\}_{i=1}^n$ are learnable parameters. Note that in our case $n = n_c$. In practice, arbitrary flexibility of $\mathbf{Q}$ may be redundant, and we can trade off computational complexity and flexibility by using a smaller number of Householder reflections. The log determinant of the QR decomposition is $h \cdot w \cdot \text{sum}(\log |\mathbf{s}|)$ and can be computed in $\mathcal{O}(n_c)$. The computational complexity to construct $\mathbf{Q}$ is between $\mathcal{O}(n_c^2)$ and $\mathcal{O}(n_c^3)$ depending on the desired flexibility. The QR parametrization has two main advantages: in contrast with the straightforward parameterization it is numerically stable, and it can be completely flexible in contrast with the PLU parametrization.

4. Related Work

The field of generative modeling has been approached from several directions. This work mainly builds upon generative flow methods developed in (Rippel & Adams, 2013; Dinh et al., 2014; 2017; Papamakarios et al., 2017; Kingma & Dhariwal, 2018). In (Papamakarios et al., 2017) autoregressive convolutions are also used for density estimation, but both its depth and number of channels makes drawing samples computationally expensive.

Normalizing flows have also been used to perform flexible inference in variational auto-encoders (Rezende & Mohamed, 2015; Kingma et al., 2016; Tomczak & Welling, 2016; van den Berg et al., 2018; Huang et al., 2018) and Bayesian neural networks (Louizos & Welling, 2017). Instead of designing discrete sequences of transformations, continuous-time normalizing flows can also be designed by drawing a connection with ordinary differential equations (Chen et al., 2018; Grathwohl et al., 2018).

Other likelihood-based methods such as PixelCNNs (Van Oord et al., 2016) impose a specific order on the dimensions of the image, which may not reflect the actual generative process. Furthermore, drawing samples tends to be computationally expensive. Alternatively, VAEs (Kingma & Welling, 2014) optimize a lower bound of the likelihood. The likelihood can be evaluated via an importance sampling scheme, but the quality of the estimate depends on the number of samples and the quality of the proposal distribution.

Many non likelihood-based methods that can generate high resolution image samples utilize Generative Adversarial Networks (GAN) (Goodfellow et al., 2014). Although GANs tend to generate high quality images, they do not directly optimize a likelihood. This makes it difficult to obtain likelihoods and to measure their coverage of the dataset.
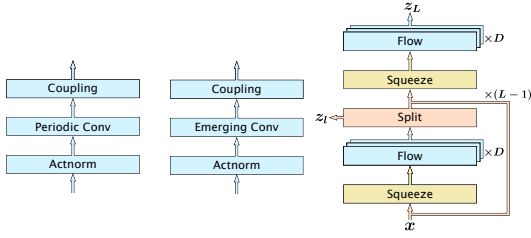
Figure 7. Overview of the model architecture. Left and center depict the flow modules we propose: containing either a periodic convolution or an emerging convolution. The diagram on the right shows the entire model architecture, where the flow module is now grouped. The squeeze module reorders pixels by reducing the spatial dimensions by a half, and increasing the channel depth by four. A hierarchical prior is placed on part of the intermediate representation using the split module as in (Kingma & Dhariwal, 2018). x and z denote input and output. The model has L levels, and D flow modules per level.

5. Results

The architecture of (Kingma & Dhariwal, 2018) is the starting point for the architecture in our experiments. In the flow module, the invertible 1×1 convolution can simply be replaced with a $d \times d$ periodic or emerging convolution. For a detailed overview of the architecture see Figure 7. We quantitatively evaluate models on a variety of datasets in bits per dimension, which is equivalent to the negative $\log_2$-likelihood. We do not use inception based metrics, as they do not generalize to different datasets, and they do not report overfitting (Barratt & Sharma, 2018). In addition, we provide image samples generated with periodic convolutions trained on galaxy images, and samples generated with emerging convolutions trained on CIFAR10.

Note that generative models are very computationally expensive in general, and we do not have the computational budget to run extremely high-dimensional image modeling tasks.

5.1. Galaxy density modeling

Since periodic convolutions assume that image boundaries are connected, they are suited for data where pixels along the boundaries are roughly the same, or are actually connected. An example of such data is pictures taken in space, as they tend to contain some scattered light sources, and boundaries are mostly dark. Ackermann et al. collected a small classification dataset of galaxies with images of merging and non-merging galaxies. On the non-merging galaxy images, we compare the bits per dimension of three models, constrained by the same parameter budget: 1×1 convolutions (Glow), 3×3 Periodic and 3×3 Emerging convolutions (see Table 2). Experiments show that both our periodic and emerging convolutions significantly outperform 1×1 convolutions, and their performance is less

Table 2. Comparison of 1×1, periodic and emerging convolutions on the galaxy images dataset. Performance is measured in bits per dimension. Results are obtained by running 3 times with different random seeds, $\pm$ reports standard deviation.

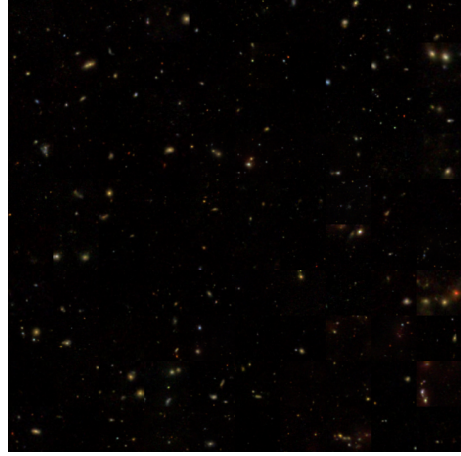|	Galaxy
1×1 (Glow)	2.03 $\pm$0.026
Periodic 3×3	**1.98** $\pm$0.003
Emerging 3×3	**1.98** $\pm$0.007

Figure 8. 100 samples from a generative flow model utilizing periodic convolutions, trained on the galaxy images dataset.

sensitive to initialization. Samples of the model using periodic convolutions are depicted in Figure 8.

5.2. Emerging convolutions

The performance of emerging convolution is extensively tested on CIFAR10 (Krizhevsky & Hinton, 2009) and ImageNet (Russakovsky et al., 2015), with different architectural sizes. The experiments in Table 3 use the architecture from Kingma & Dhariwal (2018), where emerging convolutions replace the 1×1 convolutions. Emerging convolutions perform either on par or better than Glow[3], which may be caused by the overparameterization of these large models. Samples of the model using emerging convolutions are depicted in Figure 9.

In some cases, it may not be feasible to run very large models in production because of the large computational cost. Therefore, it is interesting to study the behavior of models when they are constrained in size. We compare 1×1 and emerging convolutions with the same number of flows per level (D), for $D = 8$ and $D = 4$. Both on CIFAR10 and ImageNet, we observe that models using emerging convolutions perform significantly better. Furthermore, for smaller models the contribution of emerging convolutions becomes

---

[3]The CIFAR10 performance of Glow was obtained by running the code from the original github repository.

Figure 9. 100 samples from a generative flow model utilizing emerging convolutions, trained on CIFAR10.

Table 3. Performance of Emerging convolutions on CIFAR10, ImageNet 32x32 and ImageNet 64x64 in bits per dimension (negative $\log_2$-likelihood), and $\pm$ reports standard deviation.

	CIFAR10	ImageNet 32x32	ImageNet 64x64
Real NVP	3.51	4.28	3.98
Glow	3.36 $\pm$0.002	**4.09**	**3.81**
Emerging	**3.34** $\pm$0.002	**4.09**	**3.81**

more important, as evidenced by the increasing performance gap (see Table 4).

5.3. Modeling and sample time comparison with MAF

Recall that the inverse of autoregressive convolutions requires solving a sequential problem, which we have accelerated with an inversion module that uses Cython and parallelism across the minibatch. Considering CIFAR-10 and the same architecture as used in Table 3, it takes 39ms to sample an image using our accelerated emerging inverses, 46 times faster than the naïvely obtained inverses using tensorflow bijectors (see Table 5). As expected, sampling from models using 1×1 convolutions remains faster and takes 5ms.

Table 4. Performance of generative flows models with different architectures on CIFAR10, ImageNet 32x32 and ImageNet 64x64 in bits per dimension. Results are obtained by running 3 times with different random seeds, $\pm$ reports standard deviation.

	CIFAR10	ImageNet 32x32	D
1×1 (Glow)	3.46 $\pm$0.005	4.18 $\pm$0.003	8
Emerging	**3.43** $\pm$0.004	**4.16** $\pm$0.004	8
1×1 (Glow)	3.56 $\pm$0.008	4.28 $\pm$0.008	4
Emerging	**3.51** $\pm$0.001	**4.25** $\pm$0.002	4

Table 5. Comparison of 1×1, MAF and Emerging convolutions on CIFAR-10. Performance is measured in bits per dimension, and the time required to sample a datapoint, when computed in minibatches of size 100. The naïve implementation uses Tensorflow bijectors, and our accelerated implementation uses Cython with MPI parallelization.

CIFAR10	bits/dim	Naïve sample (ms)	Accelerated sample (ms)
1×1 (Glow)	3.36	5	5
MAF & 1×1	**3.33**	3000	650
Emerging	3.34	1800	39

Masked Autoregressive Flows (MAFs) are a very flexible method for density estimation, and they improve performance over emerging convolutions slightly, 3.33 versus 3.34 bits per dimension. However, the width and depth of MAFs makes them a poor choice for sampling, because it considerably increases the time to compute their inverse: 3000ms per sample using a naïve solution, and 650ms per sample using our inversion module. Since emerging convolutions operate on lower dimensions of the data, they are 17 times faster to invert than the MAFs.

5.4. QR 1×1 convolutions

QR 1×1 convolutions are compared with standard and PLU convolutions on the CIFAR10 dataset. The models have 3 levels and 8 flows per level. Experiments confirm that our stable QR decomposition achieves the same performance as the standard parameterization, as shown in Table 6. This is expected, since any real square matrix has a QR decomposition. Furthermore, the experiments confirm that the less flexible PLU parameterization leads to worse performance, which is caused by the fixed permutation matrix.

Table 6. Comparison of standard, PLU and QR 1×1 convolutions. Performance is measured in bits per dimension (negative $\log_2$-likelihood). Results are obtained by running 3 times with different random seeds, $\pm$ reports standard deviation.

Parametrization	CIFAR10
W	**3.46** $\pm$0.005
PLU	3.47 $\pm$0.006
QR	**3.46** $\pm$0.004

6. Conclusion

We have introduced three generative flows: i) $d \times d$ emerging convolutions as invertible standard zero-padded convolutions, ii) $d \times d$ periodic convolutions for periodic data or data with minimal boundary variation, and iii) stable and flexible 1×1 convolutions using a QR parametrization. Our methods show consistent improvements over various datasets using the same parameter budget, especially when considering models constrained in size.

References

Ackermann, S., Schawinksi, K., Zhang, C., Weigel, A. K., and Turp, M. D. Using transfer learning to detect galaxy mergers. *Monthly Notices of the Royal Astronomical Society*.

Barratt, S. and Sharma, R. A note on the inception score. *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018.

Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pp. 6572–6583, 2018.

Deco, G. and Brauer, W. Higher Order Statistical Decorrelation without Information Loss. In Tesauro, G., Touretzky, D. S., and Leen, T. K. (eds.), *Advances in Neural Information Processing Systems 7*, pp. 247–254. MIT Press, 1995.

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

Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using Real NVP. *International Conference on Learning Representations, ICLR*, 2017.

Germain, M., Gregor, K., Murray, I., and Larochelle, H. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pp. 881–889, 2015.

Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, pp. 2214–2224, 2017.

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

Grathwohl, W., Chen, R. T., Betterncourt, J., Sutskever, I., and Duvenaud, D. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *arXiv preprint arXiv:1810.01367*, 2018.

Huang, C.-W., Krueger, D., Lacoste, A., and Courville, A. Neural autoregressive flows. *arXiv preprint arXiv:1804.00779*, 2018.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10236–10245, 2018.

Kingma, D. P. and Welling, M. Auto-Encoding Variational Bayes. In *Proceedings of the 2nd International Conference on Learning Representations*, 2014.

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

Krizhevsky, A. and Hinton, G. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Li, X. and Grathwohl, W. Training Glow with Constant Memory Cost. *NIPS Workshop on Bayesian Deep Learning*, 2018.

Louizos, C. and Welling, M. Multiplicative normalizing flows for variational bayesian neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2218–2227. JMLR. org, 2017.

Papamakarios, G., Murray, I., and Pavlakou, T. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, pp. 2338–2347, 2017.

Rezende, D. and Mohamed, S. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1530–1538. PMLR, 2015.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286. PMLR, 2014.

Rippel, O. and Adams, R. P. High-dimensional probability estimation with deep density models. *arXiv preprint arXiv:1302.5125*, 2013.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Tabak, E. and Turner, C. V. A family of nonparametric density estimation algorithms. *Communications on Pure and Applied Mathematics*, 66(2):145–164, 2013.

Tabak, E. G., Vanden-Eijnden, E., et al. Density estimation by dual ascent of the log-likelihood. *Communications in Mathematical Sciences*, 8(1):217–233, 2010.

Theis, L., van den Oord, A., and Bethge, M. A note on the evaluation of generative models. In *International Conference on Learning Representations*, 2016.

Tomczak, J. M. and Welling, M. Improving variational auto-encoders using householder flow. *arXiv preprint arXiv:1611.09630*, 2016.

van den Berg, R., Hasenclever, L., Tomczak, J. M., and Welling, M. Sylvester normalizing flows for variational inference. *arXiv preprint arXiv:1803.05649*, 2018.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al. Conditional Image Generation with PixelCNN Decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Van Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pp. 1747–1756, 2016.

A. Experimental details

Models are optimized with settings identical to (Kingma & Dhariwal, 2018). The optimizer Adamax is used with a learning rate of 0.001. Coupling layers contain neural networks with three convolution layers. The first and last convolution are 3×3 and the center convolution is 1×1. The two hidden layers have W (width) channels and ReLU activations. A flow module consists of an actnorm layer, a mixing layer and a coupling layer. A mixing layer is either a 1×1 convolution (Glow) or a 3×3 (emerging or periodic) convolution. A forward pass of the entire model uses D flows per level L. At the end of a level the squeeze operation reduces the spatial dimensions by two, and increases the channel dimensions by four.

Table 7. Experimental setup on galaxy image modeling.

Model	L	D	W	batchsize
1×1	3	8	512	256
Periodic 3×3	3	8	510	256
Emerging 3×3	3	8	510	256

In the MAF model, 1×1 convolutions are succeeded (not replaced) by two MAF layers with opposite autoregressive order. MAF layers have the same structure as coupling layers, with the difference that the two hidden layers have 96 channels.

Table 8. Experimental setup on CIFAR10

Model	L	D	W	batchsize
Glow	3	32	512	256
MAF	3	32	462	256
Emerging	3	32	512	256

The models constrained in size are tested on CIFAR10 and have a different number of flows (D) per level. The other hyperparameters remain unchanged.

Table 9. Experimental setup with constrained model size

Model	L	W	batchsize
Glow	3	512	256
Emerging	3	507	256

Table 10. Experimental setup on ImageNet

Experiment	Dataset	L	D	W	batchsize
Glow	ImageNet 32	3	48	512	256
Emerging	ImageNet 32	3	48	512	256
Glow	ImageNet 64	3	48	512	128
Emerging	ImageNet 64	3	48	512	128

The experiments that compare parameterizations of the 1×1 convolutions all use the same hyperparameters, $L = 3$, $D = 8$, $W = 512$, and batchsize $= 256$.