CInC Flow: Characterizable Invertible 3×3 Convolution

Sandeep Nagar[1] **Marius Dufraisse**[2] **Girish Varma**[1]

[1]Machine Learning Lab, International Institute of Information Technology, Hyderabad, India
[2]Computer Science Dept., École Normale Supérieure (ENS), Paris-Saclay, France

Abstract

Normalizing flows are an essential alternative to GANs for generative modelling, which can be optimized directly on the maximum likelihood of the dataset. They also allow computation of the exact latent vector corresponding to an image since they are composed of invertible transformations. However, the requirement of invertibility of the transformation prevents standard and expressive neural network models such as CNNs from being directly used. Emergent convolutions were proposed to construct an invertible 3×3 CNN layer using a pair of masked CNN layers, making them inefficient. We study conditions such that 3×3 CNNs are invertible, allowing them to construct expressive normalizing flows. We derive necessary and sufficient conditions on a padded CNN for it to be invertible. Our conditions for invertibility are simple, can easily be maintained during the training process. Since we require only a single CNN layer for every effective invertible CNN layer, our approach is more efficient than emerging convolutions. We also proposed a new coupling layer for more flexibility and expressiveness, Quad-coupling. We benchmark our approach and show similar performance results to emergent convolutions while improving the model's efficiency. Code available on GitHub[a].

[a]https://github.com/Naagar/Normalizing_Flow_3x3_inv

1 INTRODUCTION

The availability of large datasets has resulted in improved machine learning solutions for more complex problems. However, supervised datasets are expensive to create. Hence unsupervised methods like generative models are increasingly worked on. Generative models that have been proposed can be broadly categorized under Likelihood-based methods and Generative Adversarial Methods. For example, an optimization algorithm could directly minimize the former's negative log-likelihood of the unsupervised examples. At the same time, in the latter, the loss function itself is modelled as a discriminator network that is trained alternatively. Hence likelihood-based methods directly optimize the probability of examples. In contrast, in GANs, the function being optimized is implicit and hard to reason about.

An essential type of Likelihood-based Generative models is normalizing flow-based models. Normalizing flow-based models transform a latent vector usually sampled from a continuous distribution like the Gaussian by a sequence of invertible functions to produce the sample. Hence even though the latent vector distribution is simple, the sample distribution could be highly complex, provided we are using an expressive set of invertible transformations. Also, the invertibility of the model implies that one can find the exact latent vector corresponding to an example from a dataset. All other approaches to generative modelling can compute the latent vector, for example, only approximately.

The ability of a normalizing flow based model to express complex real-world data distributions depends on the expressive power of the invertible transformations used. In supervised models in vision tasks, complex, multilayered CNNs with different window sizes are used. CNN's with larger window size helps in spatial mixing of information about the images, resulting in expressive features. Glow used invertible 1×1 convolutions to build normalizing flow models Kingma and Dhariwal [2018]. For a 1×1 convolution (if it is invertible), the inverse is also a 1×1 convolution. We show that this approach does not generalize to larger window sizes. In particular, the inverse of an invertible 3×3 convolution necessarily depends on all the feature vector dimensions, unlike CNNs, which only require local features.

Emerging convolutions proposed a way of inverting convolutions with large window sizes Hoogeboom et al. [2019]. The inverse is not a convolution and is computed by a linear equa-

Accepted for the 4th Workshop on Tractable Probabilistic Modeling (TPM 2021).

tion system that can efficiently be solved using back substitution. However, for obtaining an invertible convolution, they required 2 CNN filters to be applied. Hence for every effective invertible convolution, they are required to do two convolutions back to back. We propose a simple approach using padding of obtaining invertible convolutions, which only uses a single convolutional filter. Furthermore, we are able to give a characterization (necessary and sufficient conditions) for the convolutions to be invertible. This allows us to optimize over the space of invertible convolutions during training directly. We have compared our method with Emerging convolution (Hoogeboom et al. [2019]) and Autoregressive convolution (Germain et al. [2015]) in Appendix-C

Main Contributions.

- We give necessary and sufficient conditions for a 3×3 convolution to be invertible by making some modifications to the padding (see Section 3.1).
- We also propose a more expressive coupling mechanism called Quad-coupling (see Section 3.2).
- We use our characterization and Quad-coupling to train flow-based models that give samples of similar quality as previous works while improving upon the run-time compared to the other invertible 3×3 convolutions proposed (see Section 4).

2 RELATED WORKS

Normalizing flows. A normalizing flow aims to model an unknown data distribution (Kingma and Dhariwal [2018], Durkan et al. [2019a,b]), that is, to be able to sample from this distribution and estimate the likelihood of an element for this distribution.

To model the probability density of a random variable x, a normalizing flow apply an invertible change of variable $x = g_\theta(z)$ where z is a random variable following a known distribution for instance $z \sim \mathcal{N}(0, I_d)$. Then we can get the probability of x by applying the change of variable formula

$$p_\theta(x) = p(f_\theta(x)) \left(\left| \frac{\partial f_\theta(x)}{\partial x^T} \right| \right)$$

where f_θ denotes the inverse of g_θ and $\left| \frac{\partial f_\theta(x)}{\partial x^T} \right|$ its Jacobian.

The parameters θ are learned by maximizing the actual likelihood of the dataset. At the same time, the model is designed so that the function g_θ can be inverted and have its Jacobian computed in a reasonable amount of time.

Glow. RealNVP defines a normalizing flow composed of a succession of invertible steps (Dinh et al. [2017]). Each of these steps can be decomposed into layers steps. Improvements for some of these layers where proposed in later articles (Kingma and Dhariwal [2018], Hoogeboom et al. [2019]).

Actnorm: The actnorm layer performs an affine transformation similar to batch normalization. First, its weights are initialized so that the output of the actnorm layer has zero mean and unit variance. Then its parameters are learned without any constraint.

Permutation: RealNVP proposed to use a fixed permutation to shuffle the channels as the coupling layer only acts on half of the channels. Later, Kingma and Dhariwal [2018] replaced this permutation with a 1×1 convolution in Glow. These can easily be inverted by inverting the kernel. Finally, Hoogeboom et al. [2019] replaced this 1x1 convolution with the so-called emerging convolution. These have the same receptive convolution with a kernel of arbitrary size. However, they are computed by convolving with two successive convolutions whose kernel is masked to help the inversion operation.

Coupling layer: The coupling layer is used to provide flexibility to the architecture. The Feistel scheme (Hoang and Rogaway [2010]) inspires its design. They are used to build an invertible layer out of any given function f. Here f is learn as a convolutional neural network.

$$y = [y_1, y_2], \quad y_1 = x1, \quad y_2 = (x_2 + f(x_1)) * \exp(g(x_1))$$

Where we get x_1 and x_2 by splitting the input x along the channel axis.

Invertible Convolutional Networks. Complementary to normalizing flows, there has been some work done designing more flexible invertible networks. For example, Gomez et al. [2017] proposed reversible residual networks (RevNet) to limit the memory overhead of backpropagation, while (Jacobsen et al. [2019]) built modifications to allow an explicit form of the inverse, also studying the ill-conditioning of the local inverse. Ho et al. [2019] proposed a flow-based model that is the non-autoregressive model for unconditional density estimation on standard image benchmarks

Invertible 1×1 Convolution: Kingma and Dhariwal [2018] proposed the invertible 1×1 convolution replacing fixed permutation (Dinh et al. [2017]) that reverses the ordering of the channels. Hoogeboom et al. [2019] proposed normalizing flow method to do the inversion of 1×1 convolution with doing some padding on the kernel and two distinct autoregressive convolutions, which also provide a stable and flexible parameterization for invertible 1×1 convolutions.

Invertible n×n Convolution: Reformulating n×n convolution using invertible shift function proposed by Truong et al. [2019] to decrease the number of parameters and remove the additional computational cost while keeping the range of the receptive fields. In our proposed method, there is no need for the reformulation of standard convolutions. Hoogeboom et al. [2019] proposed two different methods to produce the

invertible convolutions : (1) Emerging Convolution and (2) Invertible Periodic Convolutions. Emerging requires two autoregressive convolutions to do a standard convolution, but our method requires only one convolution as compare to the method proposed by Hoogeboom et al. [2019] and increase the flexibility of the invertible n×n convolution.

3 OUR APPROACH

We propose a novel approach for constructing invertible 3×3 convolutions and coupling layers for normalizing flows. We propose two modifications to the existing layers used in previous normalizing flow models:

- convolution layer: instead of using 1×1 convolutions or emerging convolutions, we propose to use standard convolutions with a kernel of any size with a specific padding.
- coupling layer: we propose to use a modified version of the coupling layer designed to have a bigger receptive field.

We also show how invertibility can be used to manipulate images semantically.

3.1 INVERTIBLE 3×3 CONVOLUTION

We give necessary and sufficient conditions for an arbitrary convolution with some simple modifications on the padding to be invertible. Moreover, the inverse can also be computed by an efficient back substitution algorithm.

Definition 1 (Convolution)**.** *The convolution of an input X with shape $H \times W \times C$ with a kernel K with shape $k \times k \times C \times C$ is $Y = X * K$ of shape $(H - k + 1) \times (W - k + 1) \times C$ which is equal to*

$$Y_{i,j,c_o} = \sum_{l,h<k} \sum_{c_i=1}^{C} I_{i+l,j+k,c_i} K_{l,k,c_i,c_o} \tag{1}$$

.

In this setting, the output Y has a smaller size than the input to prevent this input is padded before applying the convolution.

Definition 2 (Padding)**.** *Given an image I with shape $H \times W \times C$, the (t, b, l, r) padding of I is the image $\hat{I}$ of shape $(H + t + b) \times (W + l + r) \times C$ defined as*

$$\hat{I}_{i,j,c} = \begin{cases} I_{i-t,j-l,c} & \text{if } i - t < H \text{ and } j - l < W \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

As zero padding does not add any bias to the input, the convolution between a padded input $\hat{I}$ and a kernel K is still a linear map between the input and the output. As such, it can be described as matrix multiplication.

An image I of shape (H, W, C) can be seen as an vector $\vec{I}$ of $\mathbb{R}^{H \times W \times C}$. In the rest of this paper we will always use the following basis $I_{i,j,c} = \vec{I}_{c+Cj+CHi}$. For any index $i \leq HWC$, let (i_y, i_x, i_c) denote the indexes that satisfy $\vec{I}_i = I_{i_y,i_x,i_c}$. Note that $i < j$ iff $(i_y, i_x, i_c) \prec (j_y, j_x, j_c)$ where $\prec$ denotes the lexicographical order over $\mathbb{R}^3$. If $C = 1$ this means that the pixel (j_y, j_x) is on the right or below the pixel (i_y, i_x).

Definition 3 (Matrix of a convolution.)**.** *Let K be a kernel of shape $k \times k \times C \times C$. The matrix of a convolution of kernel K with input X of size $H \times W \times C$ with padding (t, b, l, r) is a matrix describing the linear map $X \mapsto \hat{X} * K$.*

Characterization of invertible convolutions: We consider convolution with top and left padding only. For such convolutions, we give necessary and sufficient conditions for it to be invertible. Let K be the kernel of the convolution with shape $3 \times 3 \times N \times N$ where 3×3 is the window size, and N is the number of channels. Note that number of input channels should be equal to the number of output channels for it to be invertible.

Lemma 1. *Let $y = M\hat{x}$,*

M is a lower triangular matrix with all diagonal entries $= K_{3,3}$

Where the matrix M is which produces the equivalent result when multiplied with a vectorized input $(\hat{x})$.

Proof. Consider any entry in the upper right half of M. That is (i, j) such that $i < j$ according to the ordering given in the definition of M. $M_{i,j}$ is nothing but the scalar weight that needs to be multiplied to the jth pixel of input when computing ith pixel of the output. The linear equation relating these two variable is as follows:

$$y_i = \sum_{l=0}^{3} \sum_{k=0}^{3} K_{3-l,3-k} x_{i_x-l,i_y-k}$$

From this equations follows that if $j_x > i_x$ or $j_y > i_y$ then the ith pixel of the output does not depend on the jth pixel of the input and thus $M_{i,j} = 0$. This also justifies that all diagonal coefficients of M are equal to $K_{3,3}$ □

We first describe our conditions for the case when $N = 1$. We prove the following theorem.

Theorem 1 (Characterization for $N = 1$)**.**

$$M \text{ is invertible iff } K_{3,3} \neq 0.$$

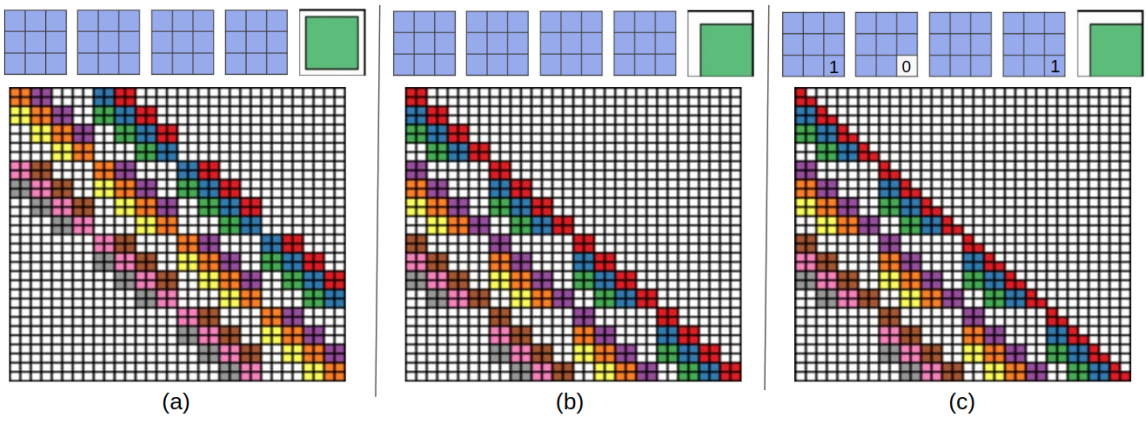

Figure 1: (a).Top: first four is kernel matrix and fifth is input matrix with the *standard* padding that give the bottom convolution matrix, Bottom: the convolution matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded on both sides and with 2 channels. Zero coefficients are drawn in white, other coefficient are drawn using the same color if they are applied to the same spatial location albeit on different channels. (b) Top: an *alternative* padding scheme that results in a block triangular matrix M, Bottom: The matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded only on one side and with 2 channels. (c) Top: an *masked alternative* padding scheme that results in a triangular matrix M, Bottom: the matrix corresponding to a convolution with kernel of size 3 applied to an input of size 4×4 padded only on one side and with 2 channel. One of the weight of the kernel is masked. Note that the equivalent matrix M is *triangular*.

Proof. The proof of the theorem uses Lemma 1. Since M is lower triangular, the determinant is nothing but the product of diagonal entries, which is $= K_{3,3}^{h*w}$ where h, w is the dimensions of the input/output image. □

At its core, the convolution layer is a linear operation. However, we have no guarantees regarding it as invertibility. The result z of the convolution of input x with kernel k can be expressed as the product as x with a matrix M. When zero-padding is used around the input so that x and z have the same shape, the matrix M is not easily invertible because the determinant of M can be zero (see matrix M in Figure 1(a)).

However, when padding only on two sides (left and top), the corresponding M is blocked triangular (see Figure 1(b)). To further ensure invertibility and speed up the inversion process, we also mask part of the kernel so that the matrix corresponding to the convolution is triangular, see in Figure 1(c) and for more details which $K_(n, n)$ we need to mask see Appendix-B. In this configuration, the Jacobian of the convolution can also be easily computed. For further details of the padding the input, see Appendix-A.

3.2 QUAD-COUPLING

The coupling layer is used to have some flexibility as its functions can be of any form. However, it only combines the effects of half channels. To overcome this issue we designed a new coupling layer inspired from generalized Feistel (Hoang and Rogaway [2010]) schemes. Instead of dividing the input x into two blocks we divide it into four $x = [x_1, x_2, x_3, x_4]$ along the feature axis. Then we keep x_1 unchanged and use it to modify the other blocks in an autoregressive manner (see Figure 2):

$$y_1 = x_1 \tag{3}$$
$$y_2 = (x_2 + f_1(x_1)) * \exp(g_1(x_1)) \tag{4}$$
$$y_3 = (x_3 + f_2(x_1, x_2)) * \exp(g_2(x_1, x_2)) \tag{5}$$
$$y_4 = (x_4 + f_3(x_1, x_2, x_3)) * \exp(g_3(x_1, x_2, x_3)) \tag{6}$$

where $(f_i)_{i \leq 3}$ and $(g_i)_{i \leq 3}$ are learned. The output of the layer is obtained by concatenating the $(y_i)_{i \leq 4}$.

4 EXPERIMENTAL RESULTS

The architecture is based on Hoogeboom et al. [2019]. We modified the emerging convolution layer to use our standard convolution. We also introduced the Quad-coupling layer in place of the affine coupling layer. Finally, we evaluate the model on a variety of models and provide images sampled from the model. For detailed overview of the architecture see Figure 3.

Training setting: To train the model on Cifar10, we used the 3 level (L) and depth (D) of 32 and lr 0.001 for the 500 epochs. To train on ImageNet32, $L = 3$, $D = 48$, lr 0.001 for the 600 epochs and for ImageNet64, $L = 4$, $D = 48$, lr 0.001 for the 1000 epochs. See Figure 3 for the model architecture.

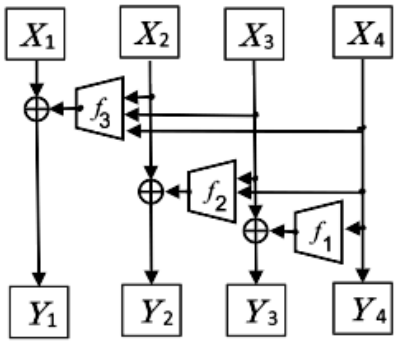

Figure 2: The Quad-coupling layer, each input block X_i has the same spatial dimension as the input X but only one quarter of the channels. Each of the function f_1, f_2 and f_3 is a 3 layer convolutional network. $\bigoplus$ symbolizes a component-wise addition. The multiplicative actions are not represented here.

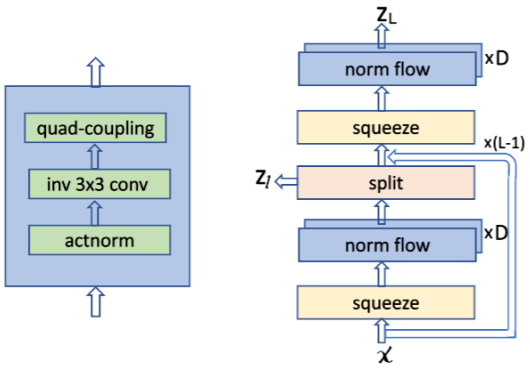

Figure 3: Overview of the model architecture. Left, the flow modules we propose: containing inv 3×3 convolution. The diagram on the right shows the entire model architecture, where the flow module is now grouped. The squeeze module reorders pixels by reducing the spatial dimensions by a half, and increasing the channel depth by four. A hierarchical prior is placed on part of the intermediate representation using the split module as in (Kingma and Dhariwal [2018]). x and z denote input and output. The model has L levels, and D flow modules per level.

Quantitative results: The Comparison of the performance of 3×3 invertible convolution with the emerging convolution (Hoogeboom et al. [2019]) for the cifar10 dataset in Table 1. The performance of our layers was tested on CIFAR10 (Krizhevsky et al. [2009]), ImageNet (Russakovsky et al. [2015]) as well as on the galaxy dataset (Ackermann et al. [2018]) see Table 2. We also tested our architecture on networks with a smaller depth ($D = 4$ or $D = 8$) see Table 3 which could be used when computational resources are limited as their sampling time is much lower. In this case, using standard convolution and Quad-coupling offers a more considerable performance improvement than with bigger models (see Table 3).

	Emerging 3×3 Inv. conv	Our 3×3 Inv. conv
Affine	3.3851	3.4209
Quad	**3.3612**	**3.3879**

Table 1: Comparison of the performance (in bits per dimension) achieved on the Cifar10 dataset with different coupling architectures.

	Glow	Emerging	3×3	Quad
Cifar10	3.35	3.34	**3.3498**	**3.3471**
ImageNet32	4.09	4.09	**4.0140**	4.0377
ImageNet64	3.81	3.81	3.8946	3.8514
Galaxy	—	2.2722	2.2739	**2.2591**

Table 2: Performance achieved on the Cifar10 and Imagenet datasets after a limited number of epochs (500 for Cifar10, 600 for ImageNet32,ImageNet64 and 1000 for Galaxy). Emerging results were obtained by using the code provided in Hoogeboom et al. [2019], 3×3 is replacing the emerging convolutions by our 3×3 invertible convolutions and Quad uses Quad-coupling on top of this.

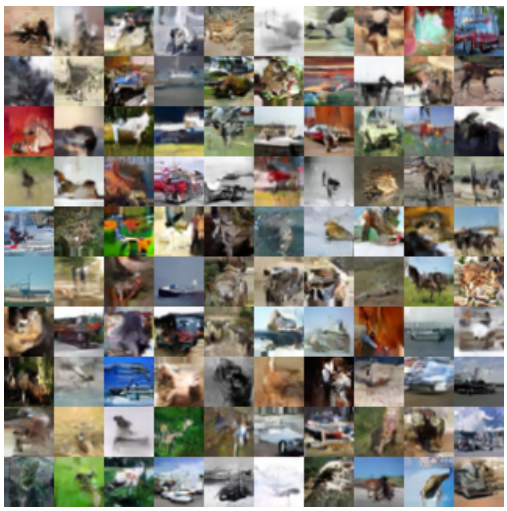

Figure 4: Sample images generated after training on the cifar dataset.

Sampling Times: We compared our method's sampling time (Table 4) against Glow (Kingma and Dhariwal [2018]) and Emerging Convolutions (Hoogeboom et al. [2019]). Our convolution still requires solving a sequential problem to be inverted and, as such, need to be inverted on CPU, unlike Glow that can be inverted while performing all the computation on GPU. This explains the gap between the sampling time of our model compared to Glow. However, it is still roughly two times faster than emerging convolutions; this comes from the need to solve two inversion problems to invert one emerging convolution layer. The Quad-coupling layer does not affect sampling time too much.

Dataset	Emerging		Ours		Depth
	Performance	Sampling time	Performance	Sampling time	
Cifar10	3.52	2.45	**3.49**	**1.31**	4
Imagenet32	4.30		**4.25**		
Cifar10	3.47	4.94	**3.46**	**2.76**	8
Imagenet32	4.20		**4.18**		

Table 3: Performance with smaller networks, when computational resources are limited. The performance is expressed in bits per dimension and the sampling time is the time in seconds needed to sample 100 images. All networks were trained for 600 epochs.

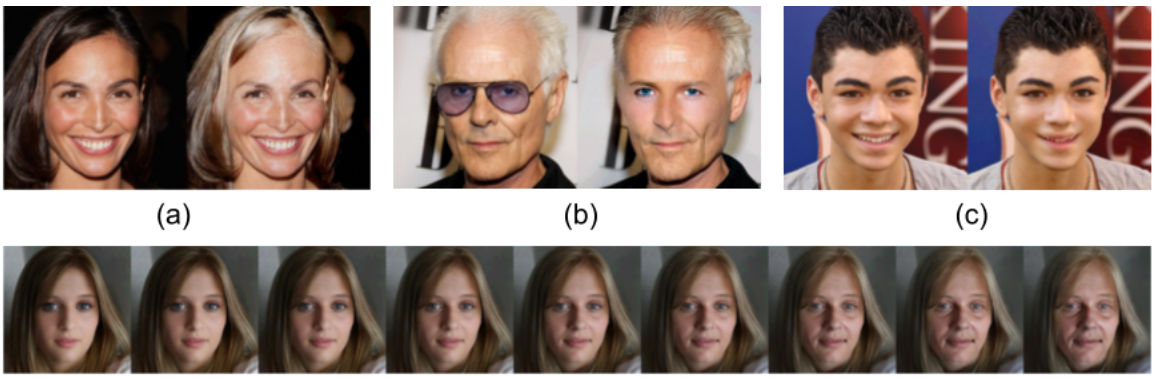

Figure 5: From left to right : the result obtained when using the network to change hair colour (a), remove glasses (b), and visage shape (c). For every example, the original image is shown on the left. Fig.(d) here, we can see the result of gradually modifying the age parameter. The original image is the fourth from the left (middle one).

	Glow	Emerging	3×3	Quad
Cifar10	0.58	18.4	9.3	10.8
Imagenet32	0.86	27.6	14.015	16.1
Imagenet64	0.50	160.72	82.04	84.06

Table 4: Time in seconds to sample 100 images. Results were obtained with Glow running on GPU and the other methods running on one CPU core.

Interpretability results: To show the interpretability of our invertible network, we used the Celeba dataset (Liu et al. [2015]) which provides images of faces and attributes corresponding to these faces. In Figure 4 are the randomly generated fake sample images for the cifar10 dataset. The covariance matrix between the attributes of images in the dataset and their latent representation indicates how to modify the latent representation of an image to add or remove features. Examples of such modifications can be seen in Figures 5(a, b, c) and 5(d).

5 CONCLUSION

In this paper, we propose a new method for Invertible n×n Convolution. Coupling layers solve two problems for normalizing flows: they have a tractable Jacobian determinant and can be inverted in a single pass. We propose a new type of coupling method, Quad-coupling. Our method shows consistent improvement over the Emerging convolutions method, and we only need a single CNN layer for every effective invertible convolution. This paper shows that we can invert a convolution with only one effective convolution, and additionally, the inference time and sampling time improved notably. We show the inversion of 3×3 convolution and the generalization of the inversion for the n×n kernel. Furthermore, we demonstrate improved quantitative performance in terms of log-likelihood on standard image modelling benchmarks.

References

Sandro Ackermann, Kevin Schawinski, Ce Zhang, Anna K Weigel, and M Dennis Turp. Using transfer learning to detect galaxy mergers. *Monthly Notices of the Royal Astronomical Society*, 479(1):415–425, 2018.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real NVP. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Conor Durkan, Artur Bekasov, Iain Murray, and George Papamakarios. Cubic-spline flows. *arXiv preprint arXiv:1906.02145*, 2019a.

Conor Durkan, Artur Bekasov, Iain Murray, and George

Papamakarios. Neural spline flows. *arXiv preprint arXiv:1906.04032*, 2019b.

Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97, pages 2722–2730. PMLR, 09–15 Jun 2019.

Viet Tung Hoang and Phillip Rogaway. On generalized feistel networks. In *Annual Cryptology Conference*, pages 613–630. Springer, 2010.

Emiel Hoogeboom, Rianne van den Berg, and Max Welling. Emerging convolutions for generative normalizing flows. *arXiv preprint arXiv:1901.11137*, 2019.

Jörn-Henrik Jacobsen, Jens Behrmann, Richard S. Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Diederik P. Kingma, Tim Salimans, and Max Welling. Improving variational inference with inverse autoregressive flow. *CoRR*, abs/1606.04934, 2016. URL `http://arxiv.org/abs/1606.04934`.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. volume 1, page 7. In Technical report, 2009.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

Thanh-Dat Truong, Khoa Luu, Chi Nhan Duong, Ngan Le, and Minh-Triet Tran. Generative flow via invertible nxn convolution. *CoRR*, abs/1905.10170, 2019. URL `http://arxiv.org/abs/1905.10170`.

A INPUT PADDING

To make sure the matrix (M) block triangular, padding of $(k+1)/2$ on top and left of input (x) is applied where k is the size of kernels.

B MASKING OF KERNELS

Considering the assumption that the numbers of input channels is equals to the output channels (N) then masking of the kernels depends only on the numbers of channels (N), let the kernel size is $k \times k$ and the $K_{k,k}^{a,b}$ are the entries corresponding to block ($N \times N$) diagonals entire in the convolution matrix (M), where a and b are $a, b \in 1, 2, ..., N$. $K_{k,k}^{a,b}$ is a square matrix (C) of size $N \times N$) and the diagonal entries of the block of matrix M are the diagonal of C and to ensure the invariability of the M, we have to set all the entries $C_{a,b}$ to zero when $a > b$ and one when $a = b$.

C COMPARISON OF OUR METHOD WITH THE EXISTING INVERTIBLE NORMALIZING FLOW METHODS

See figure 6.

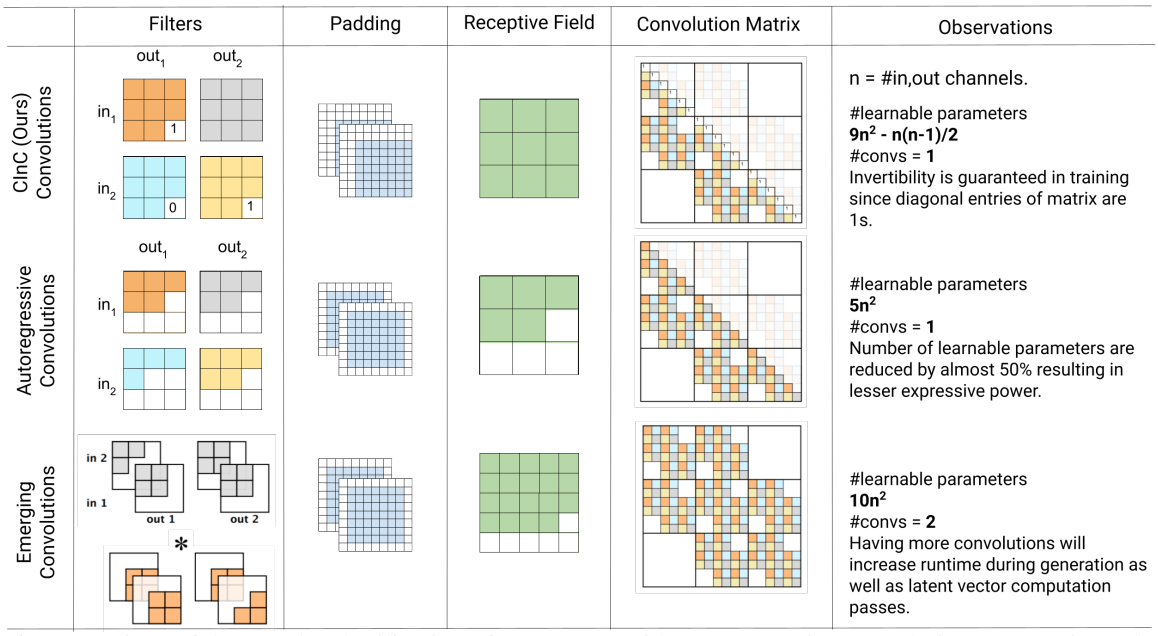

Figure 6: Comparison of the speed and utilization of parameters with Autoregressive convolutions (Germain et al. [2015], Kingma et al. [2016]) and Emerging (Hoogeboom et al. [2019])