Symmetric Convolutional Flow

Mahdi Karami [1] **Jascha Sohl-Dickstein** [2] **Laurent Dinh** [2] **Daniel Duckworth,** [2] **Dale Schuurmans** [1 2]

Abstract

Normalizing flows can be used to construct high quality generative probabilistic models. However, training and generating samples from the flow requires evaluating the determinant of the input-output Jacobian, and inverting the input-output function, respectively. In order to make these computations feasible, existing normalizing flow models have highly constrained architectures, in most cases producing a Jacobian which is a diagonal, triangular, or low-rank matrix, enabling fast Jacobian calculation. In this work, we introduce a set of novel and powerful normalizing flows based on the symmetric convolution transform. We show that the Jacobian of this transform admits efficient determinant computation and inverse mapping (deconvolution) in $\mathcal{O}(N \log N)$ time. We also propose an analytical approach for designing non-linear elementwise bijectors that induce special properties on the intermediate layers by implicitly introducing specific regularizer terms in the loss function.

1. Introduction

Flow-based generative networks have shown tremendous promise for modeling complex observations in high dimensional datasets. In flow-based models, a complex probability density is constructed by transforming a simple base density, such as a standard normal distribution, via a chain of smooth and invertible mappings (bijections), resulting in a *normalizing flow*. Such normalizing flows are employed in various contexts, including approximating a complex posterior distribution in variational inference [14], or for density estimation with generative models [2].

Using a complex transformation to define a normalized density requires the computation of a Jacobian determinant, which is generally impractical for arbitrary neural network transformations. To overcome this difficulty and enabling fast computation of the Jacobian determinant, previous work carefully designs architectures to impose a simple structure in the Jacobian matrix. For example, [14; 15] have considered transformations that with a Jacobian corresponding to low rank perturbations of a diagonal matrix, enabling the use of Sylvester's determinant lemma. Other works, such as [1; 2; 9; 13], use a constrained transformation where the Jacobian has a triangular structure. The latter scheme has proved particularly successful as this constraint is easy to enforce without large sacrifices in expressiveness or computational efficiency. More recently, [8] proposed the use of 1×1 convolutions for cross channel mixing in multi-channels signal, which owes its tractability to a block diagonal Jacobian.

In this work, we propose an alternative non-linear convolution layer we call an *adaptive non-linear convolution filter*. Here, the convolution kernel for a layer adapts to the input of the layer. Broadly speaking, splitting the input of a layer $\boldsymbol{x}$ into two disjoint subsets $\{\boldsymbol{x}_1,\ \boldsymbol{x}_2\}$, the convolution updates a subset of input as $\boldsymbol{w}(\boldsymbol{x}_1) * \boldsymbol{x}_2$, while the convolution kernel $\boldsymbol{w}(\boldsymbol{x}_2)$ is a function of a disjoint subset of input which can be modeled with deep neural networks. We present convolution operations that are invertible and whose Jacobians can be computed efficiently, making them amenable for the normalizing flow. As apposed to the causal convolution employed in [16] to generate audio waveform, or in [18] to approximate the posterior in variational autoencoder, the convolution operations presented here are not constrained to depend only on the preceding input variables.

1.1. Normalizing flows

Given an invertible and differentiable mapping $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ of a random variable $\mathbf{z} \sim p(\mathbf{z})$, with inverse transform $f = g^{-1}$, the probability density function of the transform $\boldsymbol{x} = g(\mathbf{z})$ can be recovered by the *change of variable rule* as : $p(\boldsymbol{x}) = p(\mathbf{z}) \left|\det \boldsymbol{J}_g\right|^{-1} = p(f(\boldsymbol{x})) \left|\det \boldsymbol{J}_f\right|$ where $J_g = \frac{\partial g}{\partial \mathbf{z}^\top}$ and $\boldsymbol{J}_f = \frac{\partial f}{\partial \boldsymbol{x}^\top}$ are the Jacobian matrices of functions g and f, respectively. Furthermore, one can build a complex mapping g by composing a chain of simple bijective maps, $g = g^{(1)} \circ g^{(2)} \circ ... \circ g^{(K)}$, that preserves the invertibility property with the inverse transformation being $f = f^{(K)} \circ$

[1]Department of Computer Science, University of Alberta, Alberta, Canada [2]Google Brain. Correspondence to: Mahdi Karami <karami1@ualberta.ca>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

$f^{(K-1)} \circ ... \circ f^{(1)}$; subsequently, applying the chain rule to the Jacobian of the composition, and using the fact that $\det \boldsymbol{AB} = \det \boldsymbol{A} \det \boldsymbol{B}$, the log-likelihood can be written as $\log p(\boldsymbol{x}) = \log p(\mathbf{z}) + \sum_{k=1}^{K} \log |\det \boldsymbol{J}_{f_k}|$.

This mapping, which enables specifying complex densities by flow of a simple density $p(z)$ through a sequence of bijections, is called a *normalizing flow* [14]. The basic density is chosen from well known distribution families that can be easily evaluated, such as a standard normal distribution. In comparison to variational inference methods that maximize a variational lower bound on the likelihood by learning an approximate posterior distribution on the latent variable, this equation provides a mechanism for exact likelihood maximization and density estimation.

Evaluating the Jacobian determinant is the main computational bottleneck in log-likelihood equation since, in general, it can scale cubicly with the size of input, therfore, previous work has mainly focused on transforms with a block-diagonal or triangular Jacobians. It is natural to seek general classes of structured transformations that can mitigate this cost while retaining useful modeling flexibility. In the following a class of such transformations with computationally efficient determinant-Jacoubian will be introduced.

1.2. Symmetric convolution

Besides the circular convolution, used in [7] as an invertible flow, there exist other type of structured filtering operations, such as symmetric convolution, that can be tailored to the application of interest and, hence, provide a diverse range of properties. A family of symmetric extension (padding) patterns and their corresponding discrete trigonometric transforms (DTT) are outlined in [11], based on which different symmetric convolution filtering can be defined that satisfies the convolution-multiplication property. Among all different types, we pick a simple even-symmetric extension that can be interpreted readily. Let a base sequence be extended by an even-symmetric operation $\varepsilon\{.\}$ around its first element as

$$\hat{\boldsymbol{x}}(n) = \varepsilon\{\boldsymbol{x}(n)\} := \begin{cases} \boldsymbol{x}(n) & n = 0, 1, ..., N \\ \boldsymbol{x}(2N-n) & n = N+1, ..., 2N-1 \end{cases} \tag{1}$$

subsequently, the *symmetric convolution* of two sequences can be defined in terms of the circular convolution of their corresponding even-symmetric extensions as $\boldsymbol{y} = \boldsymbol{w} *_s \boldsymbol{x} = \mathcal{R}\{\hat{\boldsymbol{x}} \circledast \hat{\boldsymbol{w}}\}$ where $\mathcal{R}\{.\}$ is a rectangular window operation to retain the base sequence of interest out of an extended sequence, *i.e.* it inverses the symmetric extension operation (1). Now, since the sequences are extended by an even-symmetric pattern, the cosine functions are the appropriate basis of the Fourier transform, giving rise to the discrete cosine transform of type one (DCT-I) defined as [1]

$$\boldsymbol{x}_c(k) = \mathcal{F}_{dct}\{\boldsymbol{x}\}_k = \sum_{n=0}^{N} \boldsymbol{x}(n) \times 2\alpha_n \cos\left(\frac{\pi k n}{N}\right) \tag{2}$$

where $\alpha_n = \begin{cases} 1/2 & n = 0, N \\ 1 & otherwise \end{cases}$

Then the convolution-multiplication property holds which implies that the symmetric convolution of two sequences in spatial domain can be expressed as a pointwise multiplication in the transform domain after forward DCT of its operands as $\mathcal{F}_{dct}\{\boldsymbol{y}\} = \mathcal{F}_{dct}\{\boldsymbol{w}\} \odot \mathcal{F}_{dct}\{\boldsymbol{x}\}$. Also, the above implies that the symmetric convolution can be alternatively defined as performing the inverse DCT on element-wise multiplication of the forward DCT of its operands [11]. The symmetric convolution-multiplication property implies that the Jacobian of the symmetric convolution operation can be diagonalized by DCT basis with eigenvalues being the DCT of the convolution kernel. On the other hand, since DCT can be defined in terms of DFT of the symmetric extension of the original sequences, thus symmetric convolution operation, its inverse transform and Jacobian determinant evaluations can enjoy the available fast Fourier algorithms with $\mathcal{O}(N \log N)$ complexity.

2. Convolutional normalizing flow

2.1. Data dependent convolution layer

For modeling flexibility, we propose a data dependent convolution layer called *adaptive non-linear convolution filter*. Inspired by the idea of the coupling layer in [2], we can build a modular bijection by splitting the input $\boldsymbol{x} \in \mathbb{R}^d$ into two disjoint parts $\{\boldsymbol{x}_1 \in \mathbb{R}^{d_1}, \boldsymbol{x}_2 \in \mathbb{R}^{d_2} : d_1 + d_2 = d\}$ termed as the *base input* and *update input*, respectively, and only update the update input $\boldsymbol{x}_2$ by an invertible convolution operation where the data-parameterized kernel of this transform is a function of base input $\boldsymbol{x}_1$. Thus, the adaptive convolution sub-flow can be expressed as

$$f_*(\boldsymbol{x}_2; \boldsymbol{x}_1) = \boldsymbol{w}(\boldsymbol{x}_1) * \boldsymbol{x}_2 \tag{3}$$

2.2. Nonlinear invertible transformations

Adding elementwise nonlinear bijection $y_i = \sigma(x_i)$ in the chain of normalizing flow can enhance the expressiveness of the network , also the Jacobian determinant term introduced by the nonlinearities can be interpreted as regularizers that capture specific structures on the output of middle layers of the flow. In the following, an analytical approach is driven to design a nonlinear invertible gate that inherently introduces an specific regularizer term in the loss function.

[1] $\boldsymbol{x}_f$, $\boldsymbol{x}_c$ and $\boldsymbol{x}_t$ denote DFT, DCT and trigonometric transform of sample $\boldsymbol{x}$, respectively.

Lemma 1 *Let $\boldsymbol{y}^{(k)}$ be the output of the k^{th} transformation in the chain of normalizing flows, $f^{(k)}$, and this transformation be an elementwise operation,* i.e. $\boldsymbol{y}_i^{(k)} = f^{(k)}(\boldsymbol{x}_i^{(k)})$ [2]. *Dropping the indices, this transformation can be simply written as $y = f(x)$ with inverse $x = g(y) = f^{-1}(y)$. Assume we want to induce a specific structure, captured and formulated by the regularizer term $\gamma(y)$, on the latent variable $y := \boldsymbol{y}_i^{(k)}$. Then the elementwise transformation can be defined as the solution to the differential equation $\left|\frac{\partial f^{-1}}{\partial y}\right| = \left|\frac{\partial g}{\partial y}\right| = e^{\gamma(y)}$.*

Solving above equation for $l1$ regularizer, $\gamma(y) = \alpha|y|$ which corresponds to Laplace distribution assumption on y, we can derive the nonlinear bijection

$$g(y) = \tfrac{\text{sign}(y)}{\alpha}(e^{\alpha|y|} - 1), \;\; f(x) = \tfrac{\text{sign}(x)}{\alpha}\ln(\alpha|x| + 1).$$

Due to its symmetric logarithmic shape, we term the forward transform $f(x)$ as *S-Log* gate. It is worth mentioning that these gate are not only differentiable by construction but also their domain and range are unbounded, the properties that make them suitable choices for designing normalizing flows.

2.3. Combined convolution multiplication layer

The convolution operation slides a filter spatially and applies the same weighted summation on all locations of its input resulting in a location invariant filtering. To achieve a more flexible and richer filtering scheme, we can combine an element-wise multiplication and convolution so that the filtering scheme varies over the location. All in all, the above components can be deployed to compose a *complex convolutional flow* as

$$\begin{aligned} f_{\boldsymbol{w},\boldsymbol{s}}(\boldsymbol{x}_2;\boldsymbol{x}_1) &= (g_{\alpha'} \circ f_{\odot} \circ f_{\alpha} \circ f_{*})(\boldsymbol{x}_2;\boldsymbol{x}_1) \\ &= g_{\alpha'}\,(\; \boldsymbol{s}(\boldsymbol{x}_1) \odot f_{\alpha}(\boldsymbol{w}(\boldsymbol{x}_1) * \boldsymbol{x}_2)\;) \end{aligned} \tag{4}$$

with the inverse transform

$$g_{\boldsymbol{w},\boldsymbol{s}}(\boldsymbol{y}_2;\boldsymbol{x}_1) = \boldsymbol{w}^{inv}(\boldsymbol{x}_1) * g_{\alpha}(\boldsymbol{s}^{inv}(\boldsymbol{x}_1) \odot f_{\alpha'}(\boldsymbol{y}_2))$$

initialization of the parameters The pair of nonlinear bijectors, $\{f_{\alpha}, g_{\alpha'}\}$, act as a sequence of contraction and expansion functions and while they are parameterized independently, their parameters $\{\alpha, \alpha'\}$ are initialized similarly so that they become inverse of each other initially. Furthermore, the conditioning networks are initialized such that the convolution kernels at the frequency domain, $\boldsymbol{w}_c$, and the scaling filters, $\boldsymbol{s}$, all being set to value one, making them identity filter at the beginning. This initialization scheme renders the whole combined flow to act as an identity mapping at initialization which in turn expected to improves data propagation for very deep networks.

We found that a more expressive network can be achieved by stacking multiples of the complex convolutional flows and an additive coupling transform in each step of the network. Therefore, the *mixed* convolutional normalizing flow (*CONF*) can be written as

$$\begin{aligned} &\boldsymbol{y}_1 = \boldsymbol{x}_1 \\ &\boldsymbol{y}_2 = (f^{(1)}_{\boldsymbol{w},\boldsymbol{s}} \circ ... \circ f^{(m)}_{\boldsymbol{w},\boldsymbol{s}})(\boldsymbol{x}_2;\boldsymbol{x}_1) + \boldsymbol{t}(\boldsymbol{x}_1). \end{aligned} \tag{5}$$

The parameters of the flow $\{\boldsymbol{w}_1, \boldsymbol{s}_1, ..., \boldsymbol{w}_m, \boldsymbol{s}_m, \boldsymbol{b}\}$ can be any nonlinear functions of the base input $\boldsymbol{x}_1$ and are not required to be invertible, hence they can be modeled by deep neural networks with an arbitrary number of hidden units, offering flexibility and rich representation capacity while preserving an efficient learning algorithm. To make the neural network simpler... This network can more powerfully warp the input while all the parameters sharing the same neural networks

Due to its modular structure, the Jacobian of (5) can be expressed in terms of the Jacobian of its sub-flow. More precisely, its Jacobian is

$$\boldsymbol{J}_y = \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{x}^\top} = \begin{bmatrix} \boldsymbol{I}_{d_1} & \boldsymbol{0} \\ \frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{x}_1^\top} & \frac{\partial \boldsymbol{y}_2}{\partial \boldsymbol{x}_2^\top} \end{bmatrix}. \tag{6}$$

Noticeably, the Jacobian is a block triangular matrix, so its determinant can be readily computed as the product of determinant of the square diagonal blocks, therefore

$$\log|\det \boldsymbol{J}_y| = \sum_{i=1}^{m} \log\left|\det \boldsymbol{J}^{(i)}_{w,s}\right| =$$

$$\sum_{i=1}^{m} \log\left|\det \boldsymbol{J}^{(i)}_{g_{\alpha'}}\right| + \log\left|\det \boldsymbol{J}^{(i)}_{\odot}\right| + \log\left|\det \boldsymbol{J}^{(i)}_{f_{\alpha}}\right| + \log\left|\det \boldsymbol{J}^{(i)}_{*}\right|$$

where $\boldsymbol{J}^{(i)}_{w,s}$ denotes the Jacobian of $f^{(i)}_{w,s}$. According to the results presented for invertible convolutions in section 1, $\log\left|\det \boldsymbol{J}^{(i)}_{*}\right|$ can be computed efficiently in $\mathcal{O}(N \log N)$ times using the fast Fourier transform algorithm. Also, it is worth noting that this term plays the role of a log barrier in the final loss function that prevents the eigenvalues of the Jacobian from falling to zero hence guarantees the invertibility of the convolution transform.

cross-channel mapping (mixing) For multi-channel inputs, the invertible convolution operation is performed in depthwise fashion *i.e.* each input channel is filtered by a separate convolution kernel. Then, cross channel information

[2]Throughout this paper, in general, $\boldsymbol{y}$ and $\boldsymbol{x}$ indicate the output and input of a flow, respectively. When referring to k^{th} flow in the chain, we use $\boldsymbol{y}^{(k)}$ and $\boldsymbol{x}^{(k)}$ where $\boldsymbol{x}^{(k)} = \boldsymbol{y}^{(k-1)}$.

Table 1: Average test negative log-likelihood (in nats) for tabular datasets (lower is better). Error bars correspond to 2 standard deviations measured 3 trials.

	POWER	GAS	HEPMASS	MINIBOONE	BSDS300
MADE	3.08 $\pm$ 0.03	-3.56 $\pm$ 0.04	20.98 $\pm$ 0.02	15.59 $\pm$ 0.50	-148.85 $\pm$ 0.28
MAF	-0.24 $\pm$ 0.01	-10.08 $\pm$ 0.02	17.70 $\pm$ 0.02	11.75 $\pm$ 0.44	-155.69 $\pm$ 0.28
TAN	-0.48 $\pm$ 0.01	-11.19 $\pm$ 0.02	15.12 $\pm$ 0.02	11.01 $\pm$ 0.48	-157.03 $\pm$ 0.07
MAF-DDSF	**-0.62** $\pm$ 0.01	**-11.96** $\pm$ 0.33	**15.09** $\pm$ 0.40	**8.86** $\pm$ 0.15	**-157.73** $\pm$ 0.04
Real NVP	-0.17 $\pm$ 0.01	-8.33 $\pm$ 0.14	18.71 $\pm$ 0.02	13.55 $\pm$ 0.49	-153.28 $\pm$ 1.78
Glow	-0.17 $\pm$ 0.01	-8.15 $\pm$ 0.40	18.92 $\pm$ 0.08	11.35 $\pm$ 0.07	-155.07 $\pm$ 0.03
FFJORD	-0.46 $\pm$ 0.01	-8.59 $\pm$ 0.12	14.92 $\pm$ 0.08	10.43 $\pm$ 0.04	-157.40 $\pm$ 0.19
Conv-Flow	**-0.49** $\pm$ .003	**-10.92** $\pm$ 0.10	**11.46** $\pm$ 0.46	**7.03** $\pm$ 0.01	**-164.72** $\pm$ 0.37

flow can be complemented by applying channel shuffling or a point-wise convolution acting on the channels similar to GLOW [8].

3. Experiments

3.1. Density estimation

We first perform some experiments to compare the expressivity of the proposed flow (CONF). As observed in [6] expressivity of the affine coupling flows and affine autoregressive flows stems from the complexity of the conditioning neural network modeling the flow parameters and successive applying the flows, therefore to make a fair comparison we follow the line of [13] and use the general-purpose neural network composed of fully connected layers in the design of conditioning networks. This way we can highlight the capacity of the flow itself without relying on very complex data dependent neural networks such as those used in [2] [8] and [5].

First the proposed flow is evaluated for density estimation task on the tabular datasets, consist of four UCI datasets (POWR, GAS, HEPMASS, MINIBOONE) and a natural image patches dataset, that was used in [13]. Description of these datasets and the preprocessing procedure applied on them can be found therein. We also perform unconditional density estimation on two image datasets; MNIST consisting of handwritten digits [17] and CIFAR-10 consisting of natural images [10]. In order to simplify the the cross channel mixing after convolution flows, having splits with equal size is beneficial consequently, a single dimension with value equal to the mean of input sample is appended to the end of each sample of HEPMASS and MINIBOONE datasets to make the size of the input vector an even number. In BSDS300, the value of bottom-right pixel is replaced with the average of its immediate neighbor pixels (neighbors with single step distance) resulting in monochrome patches of size 8×8 pixels. In image data, 2D invertible convolution is used as flow.

The density estimation performance of CONF is compared against affine coupling flow models real-NVP [2] and Glow [8] and the recent continuous-time invertible generative model FFJORD [4]. These reversible models admit efficient sampling with one single pass of the generative model. CONF offers significant performance gain over these reversible models.

We also compare the density estimation capacity of the proposed model against autoregressive based methods, MADE [3], MAF [13], TAN [12] and MAF-DDSF [6], and observe that CONF outperforms MADE and MAF on all datasets with a wide margin while could also achieve better performance on most of the datasets than the recent models TAN and MAF-DDSF. These family of autoregressive normalizing flows require $\mathcal{O}(D)$ times evaluation of the generative function to sample from the model, making them prohibitively expensive for high dimensional applications, while the most recent and powerful one, MAF-DDSF, does not provide an analytic expression for the generative function.

The results are summarized in Table 1 that highlight the rich capacity of the proposed convolutional flow for both image data and non-image data.

3.2. Variational inference

We also compare the performance of the proposed normalizing flow in designing flexible inference network for variational auto-encoder (VAE) [14]. The network architecture of [15] is used and the results of the available methods are adopted from this paper. In the initial experiments reported in this paper, a simplified model consisting of 4 complex convolutional flows with $m = 8$ that is comparable to orthogonal Sylvester flows (O-SNF) with bottleneck of size 8 in terms of number of model parameters related to the normalizing flow. The results in Table 2 demonstrate that the simple CONF model with reduced number of flows can outperform O-SNF with similar setting by a wide margin and also it can outperform the more complex O-SNF for Caltech dataset.

Table 2: Test negative log-likelihood (in nats) for MNIST and Caltech datasets (lower is better). Reported error bars correspond to 2 standard deviations calculated over 3 trials.

	MNIST		Caltech Silhouettes	
	-ELBO	NLL	-ELBO	NLL
VAE	86.55 $\pm$ 0.06	82.14 $\pm$ 0.07	110.80 $\pm$ 0.46	99.62 $\pm$ 0.74
Planar	86.06 $\pm$ 0.31	81.91 $\pm$ 0.22	109.66 $\pm$ 0.42	98.53 $\pm$ 0.68
IAF	84.20 $\pm$ 0.17	80.79 $\pm$ 0.12	111.58 $\pm$ 0.38	99.92 $\pm$ 0.30
O-SNF (F=16, M=32)	83.32 $\pm$ 0.06	80.22 $\pm$ 0.03	106.08 $\pm$ 0.39	94.61 $\pm$ 0.83
O-SNF (F=4, M=8)	84.83	80.94	109.37 $\pm$ 0.94	97.78 $\pm$ 0.47
CONF (F=4, m=8)	**83.79**	**80.71**	**104.09** $\pm$ 1.03	**94.56** $\pm$ .29

References

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

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

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

[4] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, and D. Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2019.

[5] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design, 2019.

[6] C.-W. Huang, D. Krueger, A. Lacoste, and A. Courville. Neural autoregressive flows. In *International Conference on Machine Learning*, pages 2083–2092, 2018.

[7] M. Karami, L. Dinh, D. Duckworth, J. Sohl-Dickstein, and D. Schuurmans. Generative convolutional flow for density estimation. In *Workshop on Bayesian Deep Learning NeurIPS 2018*, 2018.

[8] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv:1807.03039*, 2018.

[9] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow.

[10] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[11] S. A. Martucci. Symmetric convolution and the discrete sine and cosine transforms. *IEEE Transactions on Signal Processing*, 42(5):1038–1051, 1994.

[12] J. Oliva, A. Dubey, M. Zaheer, B. Poczos, R. Salakhutdinov, E. Xing, and J. Schneider. Transformation autoregressive networks. In *International Conference on Machine Learning*, pages 3895–3904, 2018.

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

[14] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

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

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

[17] C. C. Y. LeCun. The mnist database of handwritten digit. 1998.

[18] G. Zheng, Y. Yang, and J. Carbonell. Convolutional normalizing flows. *arXiv preprint arXiv:1711.02255*, 2017.