Multi-Scale Continuous Normalizing Flows

Anonymous CVPR 2021 submission

Paper ID ****

Abstract

Recent work has shown that Neural Ordinary Differential Equations (ODEs) can serve as generative models of images using the perspective of Continuous Normalizing Flows. Such models offer exact likelihood calculation, and invertible generation/density estimation. In this work we introduce a multi-scale variant of such models and show that this approach yields state-of-the art likelihood values for various real image datasets in lesser time with fewer parameters using only 1 GPU.

1. Introduction

Reversible generative models derived through the use of the change of variables technique [16, 39, 26, 74] are growing in interest as alternatives to generative models based on Generative Adversarial Networks (GANs) [20] and Variational Autoencoders (VAEs) [38]. While GANs and VAEs have been able to produce visually impressive samples of images, they have a number of limitations. The use of a change of variables approach facilitates the transformation of a simple base probability distribution into a more complex model distribution. Reversible generative models using this technique are attractive because they enable efficient density estimation, efficient sampling, and admit exact likelihoods to be computed. Furthermore, state-of-the art GANs and VAEs exploit the multiscale properties of images, and recently top performing methods also inject noise at each scale. While shaping noise is fundamental to normalizing flows, only recently have normalizing flows exploited the multiscale properties of images, using wavelets [74]. In this work, we consider a direct multi-scale approach to normalizing flows, which we find performs better than the corresponding wavelet approach.

A promising variation of the change of variable approach is based on the use of a continuous time variant of normalizing flows [68], which uses an integral over continuous time dynamics to transform a base distribution into the model distribution [9]. This approach uses ordinary differential equations (ODEs) specified by a neural network, or neural ODEs. Neural ODEs have been shown capable of modelling complex distributions such as those associated with images. While this new paradigm for the generative modelling of images is not as mature as GANs or VAEs in terms of the generated image quality, it is a promising direction of research as it does not have some key shortcomings associated with GANs and VAEs. Specifically, GANs are known to suffer from mode-collapse [47], and are notoriously difficult to train [2] compared to feed forward networks because their adversarial loss seeks a saddle point instead of a local minimum [4]. Neural ODEs are trained by mapping images to noise and their reversible architecture allows images to be generated by going in reverse, from noise to images. This leads to fewer issues related to mode collapse, since any input example in the dataset can be recovered from the flow transforming the input into the latent space using the reverse of the transformation learned during training. VAEs only provide a lower bound on the marginal likelihood whereas neural ODEs provide exact likelihoods.

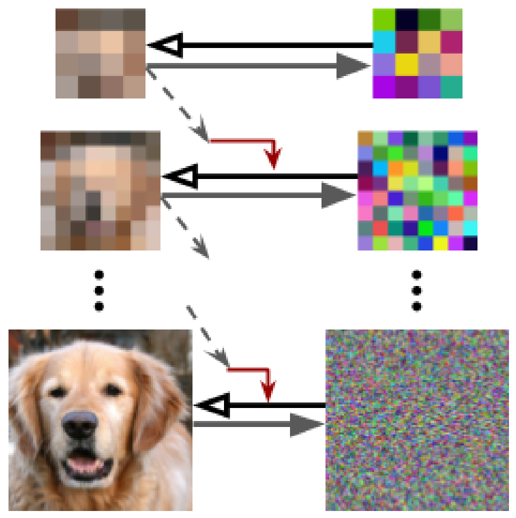


Figure 1: The architecture of our Multiscale Flow (MS-Flow) method. Normalizing flows are used to generate images at each scale, with finer scales being generated conditioned on the coarser image one level above.

Despite the many advantages of reversible generative models built with neural ODEs, quantitatively such methods still do not match the widely used Fréchet Inception Distance (FID) scores of GANs or VAEs. However their other advantages motivate us to explore them in our work here. In this work we focus on two key aspects of reversible normalizing flow based generative modelling techniques: 1) improving the quality of likelihoods for real world images, and 2) reducing computation time. We achieve these goals through a novel multi-scale technique for continuous normalizing flows. A high-level view of our approach is shown in Figure 1.

In this work, we combine a multi-scale image approach with neural ODE based normalizing flows. We use them in a multi-scale fashion to generate an image at finer resolutions conditioned on the immediate coarser resolution image using Continuous Normalizing Flows (CNF)s. Our main contributions consist of:

1. Introducing **Multi-scale Continuous Normalizing Flow (MSFlow)**: Through which we achieve state-of-the-art Bits-per-dimension (BPD) (negative log likelihood per pixel) on CIFAR10, ImageNet32 and ImageNet64 using fewer model parameters relative to comparable methods.

2. The improved BPD comes without additional cost in terms of memory and computation time, and we achieve performance on a par with prior normalizing flow works for other quantities of interest: e.g. image quality and FID scores, as well as out of distribution (OoD) measures.

2. Our method

Since images naturally exhibit structure in scale, images can be decomposed into representations at multiple scales/resolutions. Such multi-scale representations of images have been explored in computer vision for decades [7, 53, 72, 6, 52, 48]. This implies that much of the content of the image at a scale is a composition of low-level information that has been captured at coarser scales, and some high-level information not present in the coarser images. We take advantage of this property by first decomposing an image in *scale space* i.e. into a series of images at coarser scales/resolutions, either by averaging nearby pixels, or using more complex transforms such as the Haar wavelet. We then train an invertible generative model that normalizes this multi-scale image into multi-scale noise.

More formally, an image at scale/resolution S can be expressed as a series of images at progressively finer scales (or resolutions) $\mathbf{x}_{s\leq S}$. This is sometimes called an image pyramid, or a Gaussian Pyramid if the upsampling-downsampling operations include a Gaussian filter [7, 6, 1, 72, 48]. Images could also be decomposed into Wavelets [52].

2.1. Normalizing Flows

We wish to train a generative model on a multi-scale set of true images, i.e. find a probability distribution $p(\mathbf{x}_{s\leq S}) = p(\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_S)$ that matches the true data distribution. Normalizing flows [68, 32, 16, 59, 41] are good candidates for such a model, as they are probabilistic generative models that perform exact likelihood estimates, and can be run in reverse to generate novel data from the model's distribution. This allows model comparison and measuring generalization to unseen data.

Normalizing flows are trained by maximizing the log likelihood of the input image. If a normalizing flow produces output $\mathbf{z}$ from an input image $\mathbf{x}$, the change-of-variables formula provides the likelihood of the image under this transformation as:

$$\log p(\mathbf{x}) = \log p(\mathbf{z}) + \log \left| \det \frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\mathbf{x}} \right| \qquad (1)$$

Since the image data distribution is meant to be normalized to a noise distribution, $\log p(\mathbf{z})$ is computed as the log probability of $\mathbf{z}$ under the noise distribution (typically standard Gaussian $\mathcal{N}$). Hence, by maximizing the likelihood of the image $\mathbf{x}$ using Equation 1, the normalizing flow learns a reversible mapping between the true image distribution and a noise distribution.

2.2. Multi-scale Normalizing Flows

We now wish to map the joint distribution of multi-scale images $\mathbf{x}_{s\leq S}$ to "joint" multi-scale noise $\mathbf{z}_{s\leq S}$. In this case, the multi-scale change-of-variables formula is:

$$\log p(\mathbf{x}_{s\leq S}) = \log p(\mathbf{z}_0, \cdots, \mathbf{z}_S) + \log \left|\det \left[\tfrac{\partial \mathbf{z}}{\partial \mathbf{x}}\right]\right| \qquad (2)$$

The multi-scale structure of the data results in a simplification of the calculation of the Jacobian determinant. To illustrate this, choose a non-redundant basis of multi-scale variables such that $\mathbf{z}_s$ at any scale is linearly independent of $\mathbf{x}_{s+j}, j > 0$ at finer scales. (Loosely speaking, the span of the coarser scale variables have been projected out of the next finer scale variables). This leads to following lower triangular structure in the variables:

$$\begin{aligned}
&\log p(\mathbf{x}_{s\leq S}) \\
&= \log p(\mathbf{z}_0, \cdots, \mathbf{z}_S) \; + \log \left| \det \begin{bmatrix} \frac{\partial \mathbf{z}_0}{\partial \mathbf{x}_0} & 0 & \cdots & 0 \\ \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_0} & \frac{\partial \mathbf{z}_1}{\partial \mathbf{x}_1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \mathbf{z}_S}{\partial \mathbf{x}_0} & \frac{\partial \mathbf{z}_S}{\partial \mathbf{x}_1} & \cdots & \frac{\partial \mathbf{z}_S}{\partial \mathbf{x}_S} \end{bmatrix} \right| \\
&= \sum_{s=0}^{S} \left(\log p(\mathbf{z}_s) + \log \left| \det \frac{\partial \mathbf{z}_s}{\partial \mathbf{x}_s} \right| \right) \qquad (3)
\end{aligned}$$

We train a normalizing flow *at each scale* to compute the likelihood of the image up to that scale using Equation 3. This allows us to learn normalizing flows at each scale independently, and in parallel.

Since the Jacobian determinant is a lower triangular matrix, the non-zero off-diagonal elements don't contribute to the final log probability. Hence, we can freely condition each normalizing flow on the coarser images, by treating the coarser images as independent variables. This allows us to learn only the higher-level information at each scale. We use this to our advantage, and train each normalizing flow g_s between $\mathbf{x}_s$ and $\mathbf{z}_s$ conditioned on the immediate coarser $\mathbf{x}_{s-1}$ by making a Markovian assumption:

$$\begin{cases} \mathbf{z}_0 = g_0(\mathbf{x}_0) \\ \mathbf{z}_s = g_s(\mathbf{x}_s \mid \mathbf{x}_{s-1}) \quad \forall\, s > 0 \end{cases} \tag{4}$$

Hence, equation 3 can be rewritten as:

$$\log p(\mathbf{x}_{s\le S}) = \log \mathcal{N}(g_s(\mathbf{x}_0); \mathbf{0}, \mathbf{I}) + \log \left| \det \frac{\partial g_0}{\partial \mathbf{x}_0} \right| + \sum_{s=1}^{S} \left(\log \mathcal{N}(g_s(\mathbf{x}_s|\mathbf{x}_{s-1}); \mathbf{0}, \mathbf{I}) + \log \left| \det \frac{\partial g_s}{\partial \mathbf{x}_s} \right| \right) \tag{5}$$

2.3. Multi-scale Continuous Normalizing Flows

We choose to use Continuous Normalizing Flows at each scale (CNF) [9, 21], since they have recently been shown to effectively model image distributions using a fraction of the number of parameters typically used in normalizing flows (and non flow-based approaches). At each scale, each CNF g_s transforms its state (say $\mathbf{v}(t)$) using a Neural ODE [9] with neural network f_s:

$$\frac{\mathrm{d}\mathbf{v}(t)}{\mathrm{d}t} = f_s(\mathbf{v}(t), t, \mathbf{c}) \tag{6}$$

$$\mathbf{v}(t_1) = g_s(\mathbf{v}(t_0) \mid \mathbf{c}) = \mathbf{v}(t_0) + \int_{t_0}^{t_1} f_s(\mathbf{v}(t), t, \mathbf{c})\ \mathrm{d}t$$

This integration is performed by an ODE solver [9]. Chen et al. [9], Grathwohl et al. [21] proposed an instantaneous variant of the change-of-variables formula CNFs, which expresses the change in log-probability of the state of the Neural ODE i.e. $\Delta \log p_{\mathbf{v}}$ as a differential equation:

$$\frac{\partial \log p(\mathbf{v}(t))}{\partial t} = -\mathrm{Tr}\left(\frac{\partial f_s}{\partial \mathbf{v}(t)}\right)$$

$$\implies \Delta \log p_{\mathbf{v}(t_0)\to\mathbf{v}(t_1)} = -\int_{t_0}^{t_1} \mathrm{Tr}\left(\frac{\partial f_s}{\partial \mathbf{v}(t)}\right) \mathrm{d}t \tag{7}$$

Hence, the ODE solver solves for the augmented state with the above differential, to obtain both the final state as well as the change in log probability simultaneously. Thus, Equation 1 can be restated as:

$$\log p(\mathbf{x}_s) = \log p(\mathbf{z}_s) + \Delta \log p_{\mathbf{x}_s \to \mathbf{z}_s} \tag{8}$$

Thus, the log probability at each scale in eqs. (3) and (5) can be computed using equation 8 as:

$$\log p(\mathbf{x}_{s\le S}) = \sum_{s=0}^{S} \left(\log p(\mathbf{z}_s) + \Delta \log p_{\mathbf{x}_s \to \mathbf{z}_s} \right) \tag{9}$$

We call this model Multi-scale Continuous Normalizing Flow - Image (MSFlow-Image).

In general, at each scale (except the coarsest), the image $\mathbf{x}_s$ could first be converted to another representation $\mathbf{y}_s$ using a suitable orthogonal bijective transformation T from $\mathbf{x}_s$ to $\mathbf{y}_s$ so that $\Delta \log p_{\mathbf{x}_s \to \mathbf{y}_s} = 0$:

$$\log p(\mathbf{x}_{s\le S}) = \sum_{s=0}^{S} \left(\log p(\mathbf{z}_s) + \Delta \log p_{\mathbf{y}_s \to \mathbf{z}_s} + \cancelto{0}{\Delta \log p_{\mathbf{x}_s \to \mathbf{y}_s}} \right) \tag{10}$$

In the simplest case, $\mathbf{y}_s = \mathbf{x}_s$, which is MSFlow-Image. A more complex orthogonal transform to use is the Haar wavelet transform, we call this model **Multi-scale Continuous Normalizing Flow - Wavelet (MSFlow-Wavelet)**. At each scale, $\mathbf{x}_s$ is transformed into a composition of the 3 wavelet coefficients $\mathbf{w}_s$, and the coarser version $\mathbf{x}_{s-1}$, i.e. $\mathbf{y}_s = (\mathbf{w}_s, \mathbf{x}_{s-1})$. In this case, the conditioning becomes more obvious: each CNF maps the wavelet coefficients $\mathbf{w}_s$ to a noise sample $\mathbf{z}_s$ conditioned on $\mathbf{x}_{s-1}$ (see Figure 2), similar to WaveletFlow[74] which builds on Glow [39].

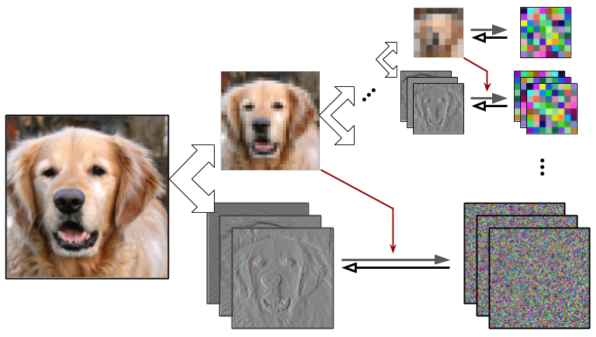


Figure 2: Architecture of MSFlow-Wavelet.

Training: The overall model is trained to maximize the log-probability of the joint multi-scale image, given by equation 9 as the sum of the likelihoods of the images at each scale. Equivalently, our model is trained to minimize the BPD of the image at finest scale S with D_S pixels:

$$\mathrm{bpd}(\mathbf{x}_{s\le S}) = \frac{-\log p(\mathbf{x}_{s\le S})}{D_S \log 2} = \frac{-1}{D_S \log 2} \left[\sum_{s=0}^{S} \left(\log p(\mathbf{z}_s) + \Delta \log p_{\mathbf{x}_s \to \mathbf{z}_s} \right) \right] \tag{11}$$

Since each CNF g_s independently models the conditional distribution of the image at that scale, we train each g_s to minimize each bpd($\mathbf{x}_{s' \leq s}$) step by step from the coarsest scale ($s = 0$) to the finest scale ($s = S$), having frozen $g_j : j \neq s$.

We use FFJORD [21] as the baseline model for our CNFs. In addition, to speed up the training of FFJORD models by stabilizing the learnt dynamics, FFJORD RNODE [18] introduced two regularization terms: the kinetic energy of the flow $\mathcal{K}(\theta)$, and the Jacobian norm $\mathcal{B}(\theta)$:

$$\begin{cases} \mathcal{K}(\theta) = \int_{t_0}^{t_1} \|f(\mathbf{x}(t), t, \theta)\|_2^2 \, \mathrm{d}t \\ \mathcal{B}(\theta) = \int_{t_0}^{t_1} \|\epsilon^\top \nabla_z f(\mathbf{x}(t), t, \theta)\|_2^2 \, \mathrm{d}t, \quad \epsilon \sim \mathcal{N}(0, I) \end{cases} \tag{12}$$

STEER [19] introduced temporal regularization by making the final time of integration stochastic:

$$\begin{cases} \mathbf{v}(t_1) = \mathbf{v}(t_0) + \int_{t_0}^{T} f(\mathbf{v}(t), t, \theta) \, \mathrm{d}t; \\ T \sim \text{Uniform}(t_1 - b, t_1 + b); \; b < t_1 - t_0 \end{cases} \tag{13}$$

Generation: Assuming each g_s is invertible (which CNFs are), we may then generate images using ancestral sampling: we first sample $\mathbf{z}_s$'s from a latent noise distribution, and transform them backwards into image space progressively from coarser to finer scales through the CNFs:

$$\begin{cases} \mathbf{x}_0 = {g_0}^{-1}(\mathbf{z}_0) \\ \mathbf{x}_s = T^{-1}(\mathbf{y}_s) = T^{-1}({g_s}^{-1}(\mathbf{z}_s \mid \mathbf{x}_{s-1})) \quad \forall \, s > 0 \end{cases} \tag{14}$$

3. Related work

Multi-scale approaches already serve as a key component of state-of-the-art GAN [15, 36, 35] and VAE [62, 70] based deep generative models. The idea is to take advantage of the fact that much of the information in an image is contained in a coarsened version, which allows us to deal with simpler problems (coarser images) in a progressive fashion. This can make models more efficient and effective. Deconvolutional CNNs [49, 61] use upsampling layers to generate images more effectively. Modern state-of-the-art generative models have also been using the injection of noise at different levels of the hierarchy to improve sample quality [5, 37, 70]. Several works [58, 63, 54, 62] have also shown how the inductive bias of the multi-scale structure helps alleviate some of the problems of image quality in likelihood-based models.

Several prior works on normalizing flows [39, 28, 29, 67, 50, 17, 8, 26, 45, 74] build on RealNVP [16]. Although they achieve great results in terms of BPD and image quality, they nonetheless report results from significantly higher parameters and several GPU hours for training.

Our MSFlow-Wavelet model is quite similar to the recently published WaveletFlow[74]. However, WaveletFlow builds on the Glow [39] architecture, while ours builds on CNFs [21, 18]. Moreover, WaveletFlow applies certain techniques to obtain better samples from its model. We have so far not used such techniques for generation, but they can possibly help generate better samples from our model as well.

"Multiple scales" in prior normalizing flows: Normalizing flows [16, 39, 21] try to be "multi-scale" by transforming the input in smart ways (squeezing operation) such that the width of the features progressively reduces while maintaining total dimensionality. We instead model multiple normalizing flows at each scale that maintain dimensionality, while conditioning on coarser scales.

4. Experimental results

We train MSFlow-Image and MSFlow-Wavelet models on the CIFAR10 [42] dataset at finest resolution of 32x32, and the ImageNet [14] dataset at 32x32, 64x64, 128x128. We build on top of the code provided in [18][1]. In all cases, we train using *only one* NVIDIA V100 GPU.

We compare our results with prior work in terms of (lower is better in all cases) the number of parameters used by the model, the BPD of the images of the test datasets under the trained models, the time taken to train, and the Fréchet Inception Distance (FID) of the generated images from the models. We achieve significantly lower BPDs in lesser time with fewer parameters than previous normalizing flows (and non flow-based approaches), and lower FID than previous CNFs.

Baselines: The most relevant models for comparison are the 1-scale FFJORD [21] models, and their regularized versions [18, 19], since our model directly converts their architecture into multi-scale. Other relevant comparisons are previous flow-based methods [16, 39, 67, 26, 74], however their core architecture (RealNVP [16]) is quite different from FFJORD.

Ablation study on regularizers: We perform an ablation study using the two regularizations mentioned above: with/without RNODE [18], with/without STEER [19] (Tables 1, 2, 3). We find that consistently in all cases, FFJORD RNODE achieves superior BPD in lesser time. In some cases, FFJORD fails to train.

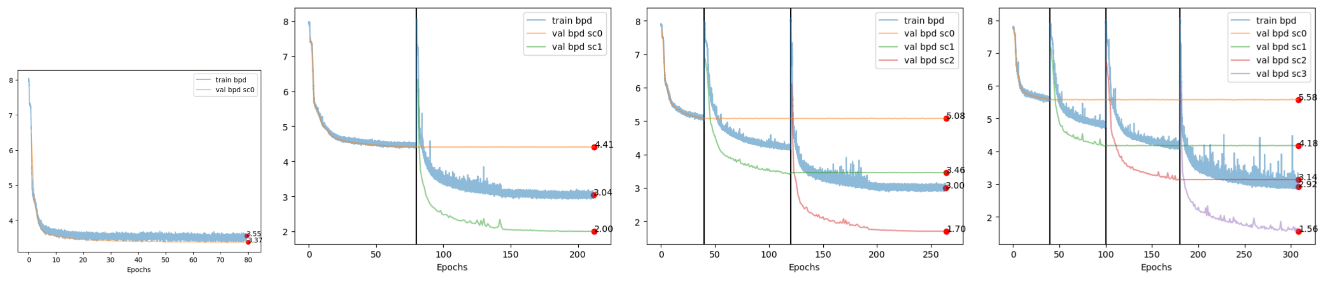
Ablation study on scales: We train models with varying number of total scales. Increasing the number of total scales consistently improves BPD across models with the same number of parameters per scale (see Table 2, Figure 3). We also see that the BPD at each scale starts around 8, which is the true BPD of an 8-bit image.

[1]https://github.com/cfinlay/ffjord-rnode

	CIFAR10			IMAGENET32			IMAGENET64		
	Param	BPD	Time	Param	BPD	Time	Param	BPD	Time
Non Flow-based Prior Work									
PixelRNN [58]		3.00			3.86			3.63	
Gated PixelCNN [71]		3.03			3.83	60		3.57	60
Parallel Multiscale [63]					3.95			3.70	
Image Transformer [60]		2.90			3.77				
PixelSNAIL [11]		2.85			3.80				
SPN [54]				150M	3.85		150M	3.53	
Sparse Transformer [12]	59M	2.80					152M	3.44	7days
Axial Transformer [27]					3.76			3.44	
PixelFlow++ [57]		2.92							
NVAE [70]		2.91	55		3.92	70			
Dist-Aug Sparse Transformer [33]	152M	2.56					152M	3.42	
Flow-based Prior Work									
RealNVP [16]		3.49		46.0M	4.28		96.0M	3.98	
Glow [39]	44.0M	3.35		66.1M	4.09		111.1M	3.81	
Emerging [28]	44.7M	3.34		67.1M	4.09		67.1M	3.81	
IDF [29]		3.34			4.18			3.90	
S-CONF [34]		3.34							
MintNet [67]	17.9M	3.32	≥5days	17.4M	4.06				
Residual Flow [10]		3.28			4.01			3.76	
MaCow [50]	43.5M	3.16					122.5M	3.69	
Neural Spline Flows [17]	11.8M	3.38					15.6M	3.82	
Flow++ [26]	31.4M	3.08		169M	3.86		73.5M	3.69	
ANF [31]		3.05			3.92			3.66	
MEF [73]	37.7M	3.32		37.7M	4.05		46.6M	3.73	
VFlow [8]		2.98			3.83				
Wavelet Flow [74]				64M	4.08		96M	3.78	822
1-scale Continuous Normalizing Flow									
FFJORD [21]	1.36M	3.40	≥5days	2.00M	$3.96^{‡}$	$>$5days‡	2.00M	x	x
FFJORD RNODE [18]		3.38	31.84		$2.36^{‡}$	$30.1^{‡}$		3.83^{*}	64.1^{*}
FFJORD + STEER [19]		3.40	86.34		3.84	>5days			
FFJORD RNODE+STEER [19]		3.397	22.24		2.35	24.9			
(OURS) 2-scale MSFlow-Image									
2sc FFJORD	0.16M	1.87	17.43	0.16M	x	x	0.16M	x	x
2sc FFJORD RNODE		**1.80**	16.53		**1.92**	26.20		**1.54**	51.21
2sc FFJORD + STEER		2.14	17.71		2.36	20.12		x	x
2sc FFJORD RNODE+STEER		1.94	17.67		1.97	65.16		1.58	66.76
(OURS) 3-scale MSFlow-Image									
3sc FFJORD	0.13M	1.76	19.83	0.13M	2.00	30.54	0.13M	x	x
3sc FFJORD RNODE		**1.70**	19.68		**1.66**	41.17		**1.21**	60.89
3sc FFJORD + STEER		2.02	22.57		2.21	21.36		x	x
3sc FFJORD RNODE+STEER		1.72	19.97		1.68	54.05		1.26	59.14
(OURS) 4-scale MSFlow-Image									
4sc FFJORD	0.17M	1.77	19.89	0.17M	1.84	30.63	0.17M	x	x
4sc FFJORD RNODE		**1.56**	15.74		**1.62**	42.60		**1.18**	65.6
4sc FFJORD + STEER		1.86	18.01		x	x		x	x
4sc FFJORD RNODE+STEER		1.65	17.58		1.63	62.82		1.36	66.2
(OURS) 2-scale MSFlow-Wavelet	0.50M	3.56	17.17	0.33M	3.92	15.30 Re!			
(OURS) 3-scale MSFlow-Wavelet	0.76M	3.69	13.99	0.51M	4.00	17.73Re!		4.04	
(OURS) 4-scale MSFlow-Wavelet	1.03M	3.77	13.94	0.69M	4.02	16.83 Re!			

Table 1: Unconditional image generation metrics (lower is better in all cases): number of parameters in the model, bits-per-dimension, time (in hours). Check Tables 2, 3 for a more detailed report. Most models use multiple GPUs for training. All our models were trained on only *one* NVIDIA V100 GPU. ‡As reported in [19]. *FFJORD RNODE [18] used 4 GPUs to train on ImageNet64. 'x': Fails to train.

Although each CNF can be trained independently (potentially on a separate GPU), we train our models on a single GPU one scale after another from coarser to finer, and compute the BPD of images at each scale (see Figure 3). We see that the BPD of the image at the same resolution decreases when total number of scales is increased, due to the conditioning. It is to be noted that the "train bpd" in Figure 3 is computed on noisy train images.



(a) 1-scale (b) 2-scale (c) 3-scale (d) 4-scale

Figure 3: Bits-per-dim vs Epoch at each scale for different MSFlow-Image models (FFJORD RNODE) trained on CIFAR10.

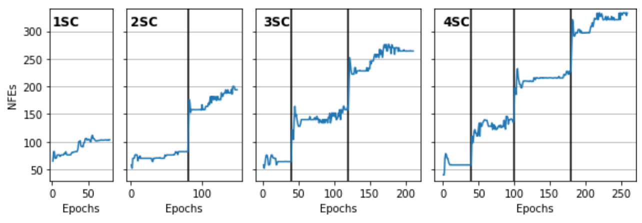


Figure 4: Number of function evaluations (NFEs) during training for MSFlow-Image models trained on (left to right) 1-scale, 2-scales, 3-scales, 4-scales

Ablation study on parameters: At each scale, there is an f_s consisting of a stack of CNN blocks with 3 layers each without any squeezing operators. We conducted an ablation study on the total number of parameters by tweaking the number of scales (2, 3, 4, 5), each f_s's channel width per block (32, 64), and the number of blocks (1, 2, 3, 4, 5) per scale (Table 4). We find that higher parameters achieve lower BPDs, but only upto a point. All our best models have $<$ 1M parameters.

Training time: (Tables 1, 2, 3) To make a fair comparison with previous methods, we report the total time taken to train the CNFs of all scales one after another on a single GPU. We also maintained the batch size of the finest scale the same as that in the previous CNF works, but used bigger batch sizes to train coarser scales. We find the inductive bias of multi-scale architecture train CNFs faster than their single-scale counterparts.

Since all the CNFs can be trained in parallel, the maximum training time could be that of only that scale which takes the longest, presumably the finest. Hence, the actual training time in practice could be much lower than those reported. Other non flow-based methods typically report their training time as several days using multiple GPUs. In contrast, all our experiments used only 1 GPU, all our CIFAR10 experiments took $<$1 day to complete, ImageNet32 a little more than 1 day, and ImageNet64 $<$3 days.

Number of function evaluations (NFE): We also note that the total NFE at scale s is the sum of the NFEs of the ODE solvers at every scale $i \leq s$ (Figure 4).

Image quality: For each of the trained models, we also report the standard image quality metric FID [25] for the generated images, and show that our method offers a marked improvement in FID relative to other CNFs. However, it does not match the image quality of other generative models such as GANs or VAEs. We find that Glow [39] and WaveletFlow [74] also reported similar FID values on non-face datasets. Unlinke those methods, we do not use any additional techniques to generate better images, such as sampling from a reduced temperature model. We believe such techniques help here, and include an ablation study on the variation in FID with temperature in the appendix.

Progressive training: Since each scale can be trained independently, we train an MSFlow-Image model on ImageNet128 by training only the finest scale (128x128) conditioned on 64x64 images for 1 epoch, and then attach that to a model trained on ImageNet64 from scratch. This model gives a BPD of ?? on ImageNet128.

Adversarial Loss: Several works have found it useful to add an adversarial loss to pre-existing losses to generate images that better resemble the true data distribution [51, 22, 43, 3]. Similar to [22], we conducted experiments with an additional adversarial loss at each scale. However in our experiments so far, we could achieve neither better BPDs nor better FIDs. Since likelihood-based models tend to cover all the modes by minimizing KL-divergence while GAN-based methods tend to mode collapse by minimizing JS-divergence [69], it is possible that the two approaches are fundamentally incompatible.

4.1. Examining out-of-distribution behaviour

The derivation of likelihood-based models suggests that the density of an image under the model is an effective measure of its likelihood of being in distribution. However, recent works [69, 55, 65, 56] have pointed out that it is possible that images drawn from other distributions have higher model likelihood. Examples have been shown where normalizing flow models trained on CIFAR10 images assign higher likelihood to SVHN images. Some also note that likelihood-based models do not generate images with

(CIFAR10)	Param	BPD	Time	FID	SVHN	TIN
1-scale Continuous Normalizing Flow						
1sc F	0.16M*	3.40	≥5days			
1sc F R		3.38	31.84	139.70‡	2.21‡	3.33‡
1sc F + S		3.40	86.34	152.60‡	2.29‡	3.44‡
1sc F R + S		3.397	22.24	148.77‡	2.24‡	3.38‡
1sc F R	0.24M	3.35	26.73	138.82	2.19	3.30
1sc F R	0.40M	3.32	25.88	132.60	2.18	3.28
(OURS) 2-scale MSFlow-Image						
2sc F R	0.09M	2.00	14.97	107.30	0.49	1.97
2sc F	0.16M	1.87	17.43	99.69	0.44	1.84
2sc F R		1.80	17.22	103.52	0.38	1.77
2sc F + S		2.14	17.71	102.29	0.58	2.08
2sc F R + S		1.94	14.81	104.13	0.51	1.93
2sc F	0.32M	1.85	17.89	97.96	0.73	
2sc F R		1.69	16.37	93.38	0.31	1.63
2sc F + S		2.04	18.76	121.58	0.50	1.99
2sc F R + S		1.74	18.43	100.78	0.32	1.70
2sc F R	0.48M	**1.59**	24.76	88.52	0.25	1.52
2sc F R	0.64M	1.61	22.85	89.55	0.25	1.53
2sc F R	0.80M	1.62	22.36	88.40	0.29	1.60
(OURS) 3-scale MSFlow-Image						
3sc F	0.13M	1.76	19.83	110.86	0.16	1.70
3sc F R		1.70	19.68	106.32	0.16	1.67
3sc F + S		2.02	22.57	105.24	0.39	2.02
3sc F R + S		1.72	19.97	103.92	0.18	1.72
3sc F	0.48M	1.54	21.51	91.80	0.06	1.46
3sc F R		1.32	21.48	91.82	0.00	1.33
3sc F + S		1.72	21.09	104.08	0.27	1.70
3sc F R + S		1.44	23.44	95.21	0.03	1.41
3sc F R	0.72M	**1.28**	21.04	88.51	-0.14	1.23
3sc F R	0.96M	1.34	21.84	88.99	-0.02	1.29
3sc F R	1.20M	1.32	20.73	91.24	-0.12	1.28
(OURS) 4-scale MSFlow-Image						
4sc F R	0.09M	2.02	9.86	122.16	0.40	1.97
4sc F	0.17M	1.77	19.89	119.34	0.43	1.80
4sc F R		1.56	15.74	107.11	0.08	1.50
4sc F + S		1.86	18.01	111.37	0.25	1.76
4sc F R + S		1.65	17.58	117.64	0.13	1.58
4sc F	0.64M	1.42	19.95	93.89	-0.01	1.44
4sc F R		**1.28**	19.08	92.19	-0.14	1.22
4sc F + S		1.88	17.73	98.24	0.2	1.78
4sc F R + S		1.44	17.60	109.68	-0.04	1.39
(OURS) 5-scale MSFlow-Image						
5sc F R						
(OURS) 6-scale MSFlow-Image						
6sc F R						

Table 2: Metrics of MSFlow-Image models trained on CIFAR10 (lower is better in all cases): number of parameters, BPD of CIFAR10 Val images, time in hours, FID, BPD of SVHN Test and TinyImageNet Test (TIN) images. F=FFJORD [21], R=RNODE [18], S=STEER [19]. *Number of parameters used in model to reproduce results. ‡Unreported in original paper, computed by us.

good sample quality as they avoid assigning small probability to out-of-distribution (OoD) data points. Hence, using the model likelihood (-BPD) for detecting OoD images is not effective.

(CIFAR10)	Param	BPD	Time	FID
(OURS) 2-scale MSFlow-Wavelet				
2sc F	0.17M	3.78	12.61	108.92
2sc F R		3.78	6.67	114.89
2sc F + S		4.13	7.31	150.91
2sc F R + S		3.78	8.18	123.94
2sc F	0.33M	3.72	21.85	100.31
2sc F R		3.64	10.61	96.42
2sc F + S		4.14	11.53	126.09
2sc F R + S		3.76	11.85	99.83
2sc F	0.50M	3.72	22.69	93.69
2sc F R		3.56	17.17	91.19
2sc F + S		3.96	17.38	117.48
2sc F R + S		3.75	15.66	98.58
(OURS) 3-scale MSFlow-Wavelet				
3sc F	0.14M	4.13	9.12	173.00
3sc F R		4.11	6.55	176.38
3sc F + S		4.16	12.82	187.60
3sc F R + S		4.21	5.06	186.50
3sc F R	0.76M	3.69	13.99	103.76
(OURS) 4-scale MSFlow-Wavelet				
4sc F	0.19M	4.10	12.23	204.95
4sc F R		4.46	3.20	199.11
4sc F + S		4.28	9.44	205.06
4sc F R + S		4.17	8.53	216.87
4sc F R	1.03M	3.77	13.94	121.80
(OURS) 5-scale MSFlow-Wavelet				
5sc F R	1.29M	3.87	10.73	131.74
(OURS) 6-scale MSFlow-Wavelet				
6sc F R	1.55M			

Table 3: CIFAR10 unconditional image generation metrics using the MSFlow-Wavelet model. F=FFJORD [21], R=RNODE [18], S=STEER [19] similar to Table 2.

MSFlow-Image

sc	ch	bl	Param
1	64	2	0.16M
	64	3	0.24M
	64	5	0.40M
2	32	2	0.09M
	64	1	0.16M
	64	2	0.32M
	64	3	0.48M
	64	4	0.64M
	64	5	0.80M

MSFlow-Image

sc	ch	bl	Param
3	32	2	0.13M
	64	2	0.48M
	64	3	0.72M
	64	4	0.96M
	64	5	1.20M
4	32	1	0.08M
	32	2	0.17M
	64	2	0.64M

MSFlow-Wavelet

sc	ch	bl	Param
2	64	1	0.17M
	64	2	0.33M
	64	3	0.50M
3	32	2	0.14M
	64	2	0.51M
	64	3	0.76M
4	32	2	0.19M
	64	2	0.69M
	64	3	1.03M
5	64	3	1.29M
6	64	3	1.55M

Table 4: Number of parameters for different models with different total number of scales (sc), and the number of channels (ch) and number of blocks (bl) per scale.

We find similar conclusions can be drawn for CNFs. Figure 5 plots the log likelihood per dimension (-BPD) of OoD images under MSFlow-Image models trained on CIFAR10. We observe that the likelihood of the OoD SVHN is higher, similar to the findings for Glow, PixelCNN, VAE in earlier works. We observe similar plots for MSFlow-Wavelet, as well as for the case of ImageNet being in-distribution.

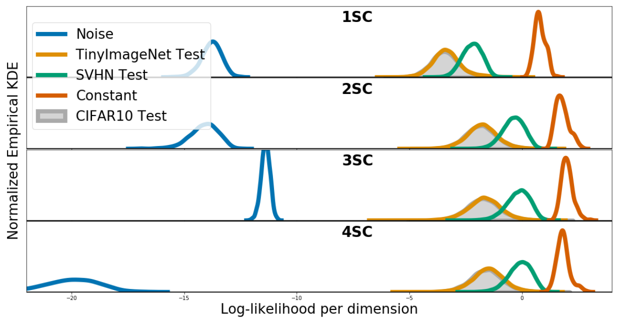


Figure 5: Histogram of -BPD, i.e. log likelihood per dimension of various out-of-distribution datasets under different MSFlow-Image models trained on CIFAR10.

One possible explanation put forward by Nalisnick et al. [56] is that "typical" images are less "likely" than constant images, which is a consequence of the distribution of a Gaussian in high dimensions. Indeed, as our Figure 5 shows, constant images have the highest likelihood under these models. Randomly generated (uniformly distributed) pixels have the least likelihood.

Choi et al. [13], Nalisnick et al. [56] suggest using "typicality" as a better measure of OoD. However, Serrà et al. [65] observe that the complexity of an image plays a significant role in the training of likelihood-based generative models. They propose to a new metric S to be used as an out-of-distribution detector:

$$S(\mathbf{x}) = \text{bpd}(\mathbf{x}) - L(\mathbf{x}) \tag{15}$$

where $L(\mathbf{x})$ is the complexity of an image $\mathbf{x}$ measured as the length of the best compressed version of $\mathbf{x}$ (we use FLIF [66]) normalized by the number of dimensions.

We perform a similar analysis as Serrà et al. [65] to test how S compares with $-$bpd for OoD detection. For different MSFlow-Image models trained on CIFAR10, we compute the area under the receiver operating characteristic curve (auROC) for each of these measures as a standard evaluation for the OoD detection task [24, 65]. It can be seen from Table 5 that S does perform better than $-$bpd in the case of CNFs, similar to the findings in Serrà et al. [65] for Glow and PixelCNN++. Serrà et al. [65] also identifies how other OoD methods [23, 46, 44, 64, 30, 24] are not as suitable in our case.

	CIFAR 10				IMAGENET32			
	SVHN		TIN		CIFAR10		SVHN	
	-bpd	S	-bpd	S	-bpd	S	-bpd	S
1sc	0.08	0.18	0.48	0.60				
2sc	0.06	0.44	0.47	0.63	0.50	0.44	0.06	0.43
3sc	0.06	0.44	0.47	0.63	0.52	0.39	0.06	0.54
4sc	0.05	0.50	0.46	0.65	0.52	0.39	0.06	0.49

Table 5: auROC for OoD detection using -bpd and S[65]

4.2. Shuffled in-distribution images

Kirichenko et al. [40] conclude that normalizing flows do not represent images based on their semantic contents, but rather directly encode their visual appearance. We verify this for continuous normalizing flows by estimating the density of in-distribution test images, but with patches of pixels randomly shuffled. Figure 6 shows an example of an image shuffled patches of varying size. Figure 7 shows the graph of the log-likelihoods of shuffled images with varying patch sizes.

It is expected that shuffling pixel patches would render the image semantically meaningless. Indeed, this is reflected in the FID between CIFAR10-Train and these sets of shuffled images:= 1x1: 340.42, 2x2: 299.99, 4x4: 235.22, 8x8: 101.36, 16x16: 33.06, 32x32 (i.e. CIFAR10-Test): 3.15. However, we see that images with large pixel patches are quite close in likelihood to the unshuffled images, suggesting that since their visual content has not changed much they are almost as likely as unshuffled images.

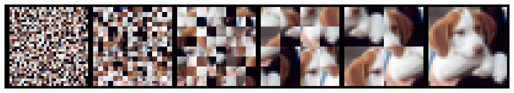


Figure 6: Examples of shuffling different-sized patches of an image: (left to right) 1x1, 2x2, 4x4, 8x8, 16x16, 32x32 (unshuffled)

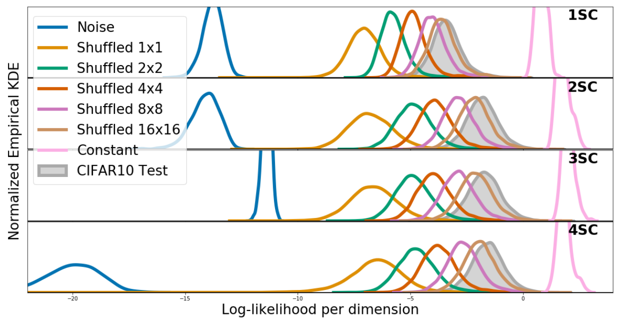


Figure 7: Histogram of -BPD, i.e. log likelihood per dimension of shuffled in-distribution datasets under different MSFlow-Image models trained on CIFAR10.

5. Conclusion

We have presented a multi-scale approach to Continuous Normalizing Flows, which provides state-of-the-art exact likelihood calculations on several benchmark datasets of images by training a single GPU in lesser time with a fraction of the number of parameters of other competitive models. In addition, we show that the multi-scale approach has similar out-of-distribution properties as other Normalizing Flows.

References

[1] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, "Pyramid methods in image processing," *RCA engineer*, vol. 29, no. 6, pp. 33–41, 1984. 2

[2] M. Arjovsky and L. Bottou, "Towards principled methods for training generative adversarial networks," *arXiv preprint arXiv:1701.04862*, 2017. 1

[3] C. Beckham, S. Honari, V. Verma, A. M. Lamb, F. Ghadiri, R. D. Hjelm, Y. Bengio, and C. Pal, "On adversarial mixup resynthesis," in *Advances in neural information processing systems*, 2019, pp. 4346–4357. 6

[4] H. Berard, G. Gidel, A. Almahairi, P. Vincent, and S. Lacoste-Julien, "A closer look at the optimization landscapes of generative adversarial networks," in *International Conference on Machine Learning*, 2020. 1

[5] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," in *International Conference on Learning Representations*, 2019. 4

[6] P. Burt and E. Adelson, "The laplacian pyramid as a compact image code," *IEEE Transactions on communications*, vol. 31, no. 4, pp. 532–540, 1983. 2

[7] P. J. Burt, "Fast filter transform for image processing," *Computer graphics and image processing*, vol. 16, no. 1, pp. 20–51, 1981. 2

[8] J. Chen, C. Lu, B. Chenli, J. Zhu, and T. Tian, "Vflow: More expressive generative flows with variational data augmentation," in *International Conference on Machine Learning*, 2020. 4, 5

[9] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud, "Neural ordinary differential equations," *Advances in Neural Information Processing Systems*, 2018. 1, 3

[10] R. T. Chen, J. Behrmann, D. K. Duvenaud, and J.-H. Jacobsen, "Residual flows for invertible generative modeling," in *Advances in Neural Information Processing Systems*, 2019, pp. 9916–9926. 5

[11] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel, "Pixelsnail: An improved autoregressive generative model," in *International Conference on Machine Learning*. PMLR, 2018, pp. 864–872. 5

[12] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019. 5

[13] H. Choi, E. Jang, and A. A. Alemi, "Waic, but why? generative ensembles for robust anomaly detection," *arXiv preprint arXiv:1810.01392*, 2018. 8

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. IEEE, 2009, pp. 248–255. 4

[15] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494. 4

[16] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," in *International Conference on Learned Representations*, 2017. 1, 2, 4, 5

[17] C. Durkan, A. Bekasov, I. Murray, and G. Papamakarios, "Neural spline flows," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 7511–7522. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/7ac71d433f282034e088473244df8c02-Paper.pdf 4, 5

[18] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. Oberman, "How to train your neural ode: the world of jacobian and kinetic regularization," *International Conference on Machine Learning*, 2020. 4, 5, 7

[19] A. Ghosh, H. S. Behl, E. Dupont, P. H. Torr, and V. Namboodiri, "Steer: Simple temporal regularization for neural odes," in *Advances in Neural Information Processing Systems*, 2020. 4, 5, 7

[20] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*. MIT Press, 2016, vol. 1. 1

[21] W. Grathwohl, R. T. Q. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud, "Ffjord: Free-form continuous dynamics for scalable reversible generative models," *International Conference on Learning Representations*, 2019. 3, 4, 5, 7

[22] A. Grover, M. Dhar, and S. Ermon, "Flow-gan: Combining maximum likelihood and adversarial learning in generative models," in *AAAI Conference on Artificial Intelligence*, 2018. 6

[23] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *International Conference on Learning Representations*, 2017. 8

[24] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," in *International Conference on Learning Representations*, 2019. 8

[25] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "Gans trained by a two time-scale update rule converge to a local nash equilibrium," in *Advances in neural information processing systems*, 2017, pp. 6626–6637. 6

[26] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving flow-based generative models with variational dequantization and architecture design," in *International Conference on Machine Learning*, 2019. 1, 4, 5

[27] J. Ho, N. Kalchbrenner, D. Weissenborn, and T. Salimans, "Axial attention in multidimensional transformers," *arXiv preprint arXiv:1912.12180*, 2019. 5

[28] E. Hoogeboom, R. v. d. Berg, and M. Welling, "Emerging convolutions for generative normalizing flows," in *International Conference on Machine Learning*, 2019. 4, 5

[29] E. Hoogeboom, J. Peters, R. van den Berg, and M. Welling, "Integer discrete flows and lossless compression," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 12 134–12 144. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/9e9a30b74c49d07d8150c8c83b1ccf07-Paper.pdf 4, 5

[30] A. Høst-Madsen, E. Sabeti, and C. Walton, "Data discovery and anomaly detection using atypicality: Theory," *IEEE Transactions on Information Theory*, vol. 65, no. 9, pp. 5302–5322, 2019. 8

[31] C.-W. Huang, L. Dinh, and A. Courville, "Augmented normalizing flows: Bridging the gap between generative flows and latent variable models," *arXiv preprint arXiv:2002.07101*, 2020. 5

[32] D. Jimenez Rezende and S. Mohamed, "Variational inference with normalizing flows," in *International Conference on Machine Learning*, 2015, pp. 1530—-1538. 2

[33] H. Jun, R. Child, M. Chen, J. Schulman, A. Ramesh, A. Radford, and I. Sutskever, "Distribution augmentation for generative modeling," in *International Conference on Machine Learning*, 2020, pp. 10 563–10 576. 5

[34] M. Karami, D. Schuurmans, J. Sohl-Dickstein, L. Dinh, and D. Duckworth, "Invertible convolutional flow," in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 5635–5645. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/b1f62fa99de9f27a048344d55c5ef7a6-Paper.pdf 5

[35] A. Karnewar and O. Wang, "Msg-gan: Multi-scale gradients for generative adversarial networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 7799–7808. 4

[36] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of gans for improved quality, stability, and variation," in *International Conference on Learned Representations*, 2018. 4

[37] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, "Analyzing and improving the image quality of stylegan," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119. 4

[38] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013. 1

[39] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," in *Advances in neural information processing systems*, 2018, pp. 10 215–10 224. 1, 3, 4, 5, 6

[40] P. Kirichenko, P. Izmailov, and A. G. Wilson, "Why normalizing flows fail to detect out-of-distribution data," in *Advances in neural information processing systems*, vol. 33, 2020. 8

[41] I. Kobyzev, S. Prince, and M. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 2

[42] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," *Technical Report, University of Toronto*, 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html 4

[43] A. X. Lee, R. Zhang, F. Ebert, P. Abbeel, C. Finn, and S. Levine, "Stochastic adversarial video prediction," *ArXiv*, vol. abs/1804.01523, 2018. 6

[44] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," in *Advances in Neural Information Processing Systems*, 2018, pp. 7167–7177. 8

[45] S.-g. Lee, S. Kim, and S. Yoon, "Nanoflow: Scalable normalizing flows with sublinear parameter complexity," in *Advances in Neural Information Processing Systems*, 2020. 4

[46] S. Liang, Y. Li, and R. Srikant, "Enhancing the reliability of out-of-distribution image detection in neural networks," in *International Conference on Learning Representations*, 2018. 8

[47] Z. Lin, A. Khetan, G. Fanti, and S. Oh, "Pacgan: The power of two samples in generative adversarial networks," in *Advances in neural information processing systems*, 2018, pp. 1498–1507. 1

[48] T. Lindeberg, "Scale-space for discrete signals," *IEEE transactions on pattern analysis and machine intelligence*, vol. 12, no. 3, pp. 234–254, 1990. 2

[49] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440. 4

[50] X. Ma, X. Kong, S. Zhang, and E. Hovy, "Macow: Masked convolutional generative flow," in *Advances in Neural Information Processing Systems*, 2019, pp. 5893–5902. 4, 5

[51] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015. 6

[52] S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 11, no. 7, pp. 674–693, 1989. 2

[53] D. Marr, *Vision: A computational investigation into the human representation and processing of visual information*. MIT press, 2010. 2

[54] J. Menick and N. Kalchbrenner, "Generating high fidelity images with subscale pixel networks and multidimensional upscaling," in *International Conference on Learning Representations*, 2019. 4, 5

[55] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" in *International Conference on Learning Representations*, 2019. 6

[56] E. Nalisnick, A. Matsukawa, Y. W. Teh, and B. Lakshminarayanan, "Detecting out-of-distribution inputs to deep generative models using a test for typicality," *arXiv preprint arXiv:1906.02994*, vol. 5, 2019. 6, 8

[57] D. Nielsen and O. Winther, "Closing the dequantization gap: Pixelcnn as a single-layer flow," in *Advances in Neural Information Processing Systems*, 2020. 5

[58] A. v. d. Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *International Conference on Machine Learning*, 2016. 4, 5

[59] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *arXiv preprint arXiv:1912.02762*, 2019. 2

[60] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," in *International Conference on Machine Learning*, 2018. 5

[61] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *arXiv preprint arXiv:1511.06434*, 2015. 4

[62] A. Razavi, A. van den Oord, and O. Vinyals, "Generating diverse high-fidelity images with vq-vae-2," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 866–14 876. 4

[63] S. Reed, A. v. d. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, D. Belov, and N. De Freitas, "Parallel multiscale autoregressive density estimation," in *International Conference on Machine Learning*, 2017. 4, 5

[64] E. Sabeti and A. Høst-Madsen, "Data discovery and anomaly detection using atypicality for real-valued data," *Entropy*, vol. 21, no. 3, p. 219, 2019. 8

[65] J. Serrà, D. Álvarez, V. Gómez, O. Slizovskaia, J. F. Núñez, and J. Luque, "Input complexity and out-of-distribution detection with likelihood-based generative models," in *International Conference on Learning Representations*, 2020. 6, 8

[66] J. Sneyers and P. Wuille, "Flif: Free lossless image format based on maniac compression," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 66–70. 8

[67] Y. Song, C. Meng, and S. Ermon, "Mintnet: Building invertible neural networks with masked convolutions," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 004–11 014. 4, 5

[68] E. G. Tabak and C. V. Turner, "A family of nonparametric density estimation algorithms," *Communications on Pure and Applied Mathematics*, vol. 66, no. 2, pp. 145–164, 2013. 1, 2

[69] L. Theis, A. v. d. Oord, and M. Bethge, "A note on the evaluation of generative models," in *International Conference on Learning Representations*, 2016. 6

[70] A. Vahdat and J. Kautz, "Nvae: A deep hierarchical variational autoencoder," in *Advances in Neural Information Processing Systems*, 2020. 4, 5

[71] 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*, 2016, pp. 4790–4798. 5

[72] A. P. Witkin, "Scale-space filtering," in *Readings in Computer Vision*. Elsevier, 1987, pp. 329–332. 2

[73] C. Xiao and L. Liu, "Generative flows with matrix exponential," in *International Conference on Machine Learning*, 2020. 5

[74] J. Yu, K. Derpanis, and M. Brubaker, "Wavelet flow: Fast training of high resolution normalizing flows," in *Advances in Neural Information Processing Systems*, 2020. 1, 3, 4, 5, 6