Supplemental Material

A Out-of-Distribution properties of (MR)CNF

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 Theis et al. (2016); Nalisnick et al. (2019a); Serrà et al. (2020); Nalisnick et al. (2019b) 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 (such as Glow) trained on CIFAR10 images assign higher likelihood to SVHN Netzer et al. (2011) images. This could have serious implications on the practical applicability of these models. Some also note that likelihood-based models do not generate images with good sample quality as they avoid assigning small probability to out-of-distribution (OoD) data points, hence using model likelihood (-BPD) for detecting OoD data is not effective.

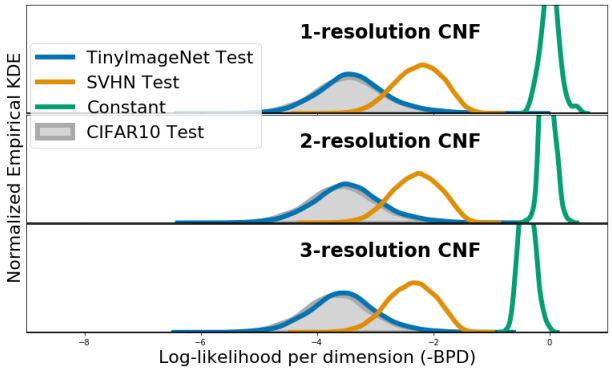


Figure 4: Histogram of log likelihood per dimension of out-of-distribution datasets (TinyImageNet, SVHN, Constant) under (MR)CNF models trained on CIFAR10. As with other likelihood-based generative models such as Glow & PixelCNN, OoD datasets have higher likelihood under (MR)CNFs.

We conduct the same experiments with (MR)CNFs, and find that similar conclusions can be drawn. Figure 4 plots the histogram of log likelihood per dimension (-BPD) of OoD images (SVHN, TinyImageNet) under MRCNF models trained on CIFAR10. It can be observed that the likelihood of the OoD SVHN is higher than CIFAR10 for MRCNF, similar to the findings for Glow, PixelCNN, VAE in earlier works Nalisnick et al. (2019a); Choi et al. (2018); Serrà et al. (2020); Nalisnick et al. (2019b); Kirichenko et al. (2020).

One possible explanation put forward by Nalisnick et al. (2019b) 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 4 shows, constant images have the highest likelihood under MRCNFs, while randomly generated (uniformly distributed) pixels have the least likelihood (not shown in figure due to space constraints).

Choi et al. (2018); Nalisnick et al. (2019b) suggest using "typicality" as a better measure of OoD. However, Serrà et al. (2020) observe that the complexity of an image plays a significant role in the training of likelihood-based generative models. They propose a new metric S as an out-of-distribution detector:

$$S(\mathbf{x}) = \text{bpd}(\mathbf{x}) - L(\mathbf{x}) \tag{16}$$

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 Sneyers & Wuille (2016) following Serrà et al. (2020)) normalized by the number of dimensions.

We perform a similar analysis as Serrà et al. (2020) to test how S compares with -bpd for OoD detection. For different MRCNF models trained on CIFAR10, we compute the area under the receiver operating characteristic curve (auROC) using -bpd and S as standard evaluation for the OoD detection task Hendrycks et al. (2019); Serrà et al. (2020).

Table 4 shows that S does perform better than -bpd in the case of (MR)CNFs, similar to the findings in Serrà et al. (2020) for

Table 4: auROC for OoD detection using -bpd and S (Serrà et al., 2020), for models trained on CIFAR10.

CIFAR10	SVHN		TIN	
(trained)	-bpd	S	-bpd	S
Glow	0.08	0.95	0.66	0.72
1-res CNF	0.07	0.16	0.48	0.60
2-res MRCNF	0.06	0.25	0.46	0.66
3-res MRCNF	0.05	0.25	0.46	0.66

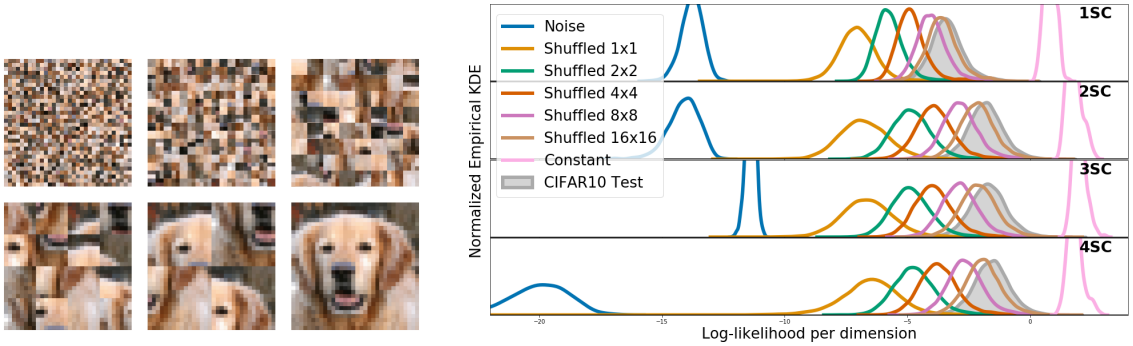
Glow and PixelCNN++. It seems that SVHN is easier to detect as OoD for Glow than MRCNFs. However, OoD detection performance is about the same for TinyImageNet. We also observe that MRCNFs are better at OoD than CNFs.

Other OoD methods Hendrycks & Gimpel (2017); Liang et al. (2018); Lee et al. (2018); Sabeti & Høst-Madsen (2019); Høst-Madsen et al. (2019); Hendrycks et al. (2019) are not as suitable in our case, as identified in Serrà et al. (2020).

A.1 SHUFFLED IN-DISTRIBUTION IMAGES

Kirichenko et al. (2020) 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 5 (a) shows an example of images of shuffled patches of varying size, Figure 5 (b) shows the graph of the their log-likelihoods.

That shuffling pixel patches would render the image semantically meaningless is reflected in the Fréchet Inception Distance (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 shuffled 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 under MRCNFs.



(a) (b)

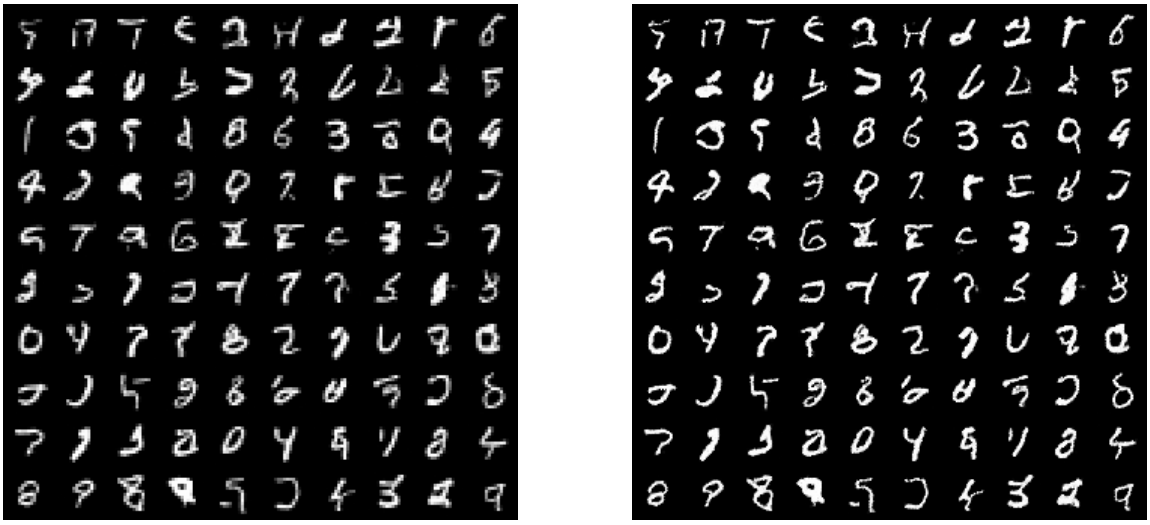
Figure 5: (a) Example of shuffling different-sized patches of a 32×32 image: (left to right, top to bottom) 1×1, 2×2, 4×4, 8×8, 16×16, 32×32 (unshuffled) (b) Bits-per-dim vs Epoch at each resolution for different MRCNF models trained on CIFAR10.

B FULL TABLE 1

	CIFAR10			IMAGENET32			IMAGENET64		
	BPD	PARAM	TIME	BPD	PARAM	TIME	BPD	PARAM	TIME
Non Flow-based Prior Work									
PixelRNN (Oord et al., 2016)	3.00			3.86			3.63		
Gated PixelCNN (Van den Oord et al., 2016)	3.03			3.83		60	3.57		60
Parallel Multiscale (Reed et al., 2017)				3.95			3.70		
Image Transformer Parmar et al. (2018)	2.90			3.77					
PixelSNAIL (Chen et al., 2018b)	2.85			3.80					
SPN (Menick & Kalchbrenner, 2019)				3.85	150.0M		3.53	150.0M	
Sparse Transformer (Child et al., 2019)	2.80	59.0M					3.44	152.0M	7days
Axial Transformer (Ho et al., 2019b)				3.76			3.44		
PixelFlow++ (Nielsen & Winther, 2020)	2.92								
NVAE (Vahdat & Kautz, 2020)	2.91		55	3.92		70			
Dist-Aug Sparse Transformer (Jun et al., 2020)	2.56	152.0M					3.42	152.0M	
Flow-based Prior Work									
IAF (Kingma et al., 2016)				3.11					
RealNVP (Dinh et al., 2017)	3.49			4.28	46.0M		3.98	96.0M	
Glow (Kingma & Dhariwal, 2018)	3.35	44.0M		4.09	66.1M		3.81	111.1M	
i-ResNets (Behrmann et al., 2019)									
Emerging (Hoogeboom et al., 2019a)	3.34	44.7M		4.09	67.1M		3.81	67.1M	
IDF (Hoogeboom et al., 2019b)	3.34			4.18			3.90		
S-CONF (Karami et al., 2019)	3.34								
MintNet (Song et al., 2019)	3.32	17.9M	$\geq$5days	4.06	17.4M				
Residual Flow (Chen et al., 2019)	3.28			4.01			3.76		
MaCow (Ma et al., 2019)	3.16	43.5M					3.69	122.5M	
Neural Spline Flows (Durkan et al., 2019)	3.38	11.8M					3.82	15.6M	
Flow++ (Ho et al., 2019a)	3.08	31.4M		3.86	169.0M		3.69	73.5M	
ANF (Huang et al., 2020)	3.05			3.92			3.66		
MEF (Xiao & Liu, 2020)	3.32	37.7M		4.05	37.7M		3.73	46.6M	
VFlow (Chen et al., 2020)	2.98			3.83					
Woodbury NF (Lu & Huang, 2020)	3.47			4.20			3.87		
NanoFlow (Lee et al., 2020)	3.25								
ConvExp (Hoogeboom et al., 2020)	3.218								
Wavelet Flow (Yu et al., 2020)				4.08	64.0M		3.78	96.0M	822
TayNODE (Kelly et al., 2020)	1.039								
1-resolution Continuous Normalizing Flow									
FFJORD (Grathwohl et al., 2019)	3.40	0.9M	$\geq$5days	‡3.96	‡2.0M	‡>5days	x		x
RNODE (Finlay et al., 2020)	3.38	1.4M	31.84	‡2.36 §3.49	2.0M §1.6M	‡30.1 §40.39	*3.83	2.0M	*256.4
FFJORD + STEER (Ghosh et al., 2020)	3.40	1.4M	86.34	3.84	2.0M	>5days			
RNODE + STEER (Ghosh et al., 2020)	3.397	1.4M	22.24	2.35 §3.49	2.0M §1.6M	24.90 §30.07			
(OURS) Multi-Resolution Continuous Normalizing Flow (MRCNF)									
2-resolution MRCNF	3.65	1.3M	19.79	3.77	1.3M	18.18	3.44	2.0M	42.30
2-resolution MRCNF	3.54	3.3M	36.47	3.78	6.7M	17.98	x	6.7M	x
3-resolution MRCNF	3.79	1.5M	17.44	3.97	1.5M	13.78	3.55	2.0M	35.39
3-resolution MRCNF	3.60	5.1M	38.27	3.93	10.2M	41.20	x	7.6M	x

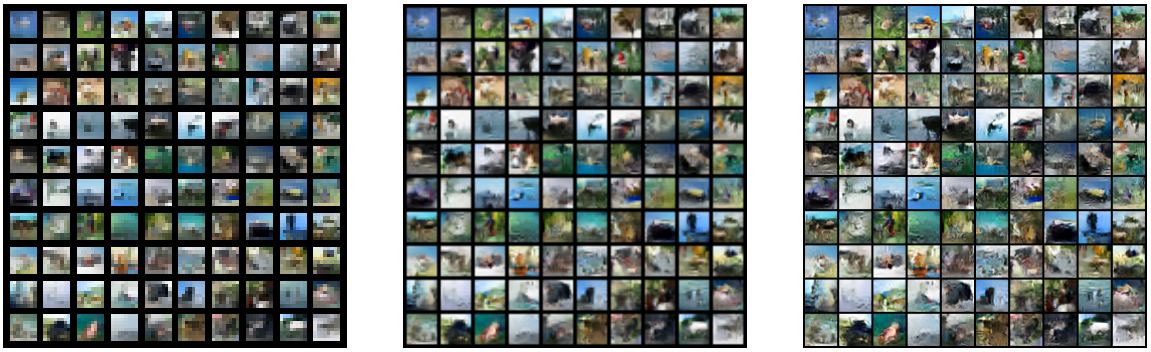
Table 5: Unconditional image generation metrics (lower is better in all cases): number of parameters in the model, bits-per-dimension, time (in hours). Most previous models use multiple GPUs for training, all our models were trained on only *one* NVIDIA V100 GPU. ‡As reported in Ghosh et al. (2020). *FFJORD RNODE Finlay et al. (2020) used 4 GPUs to train on ImageNet64. 'x': Fails to train.

C QUALITATIVE SAMPLES



(a) Generated samples at 16×16 (b) Corresponding generated samples at 32×32

Figure 6: Generated samples from MNIST.



(a) Generated samples at 8×8 (b) Generated samples at 16×16 (c) Generated samples at 32×32

Figure 7: Generated samples from CIFAR10.

D SIMPLE EXAMPLE OF DENSITY ESTIMATION

For example, if we use Euler method as our ODE solver, for density estimation Equation 2 reduces to:

$$\mathbf{v}(t_1) = \mathbf{v}(t_0) + (t_1 - t_0) f_s(\mathbf{v}(t_0), t_0 \mid \mathbf{c}) \tag{17}$$

where f_s is a neural network, t_0 represents the "time" at which the state is image $\mathbf{x}$, and t_1 is when the state is noise $\mathbf{z}$. We start at scale S with an image sample $\mathbf{x}_S$, and assume t_0 and t_1 are 0 and 1 respectively:

$$\begin{cases} \mathbf{z}_S = \mathbf{x}_S + f_S(\mathbf{x}_S,\ t_0 \mid \mathbf{x}_{S-1}) \\ \mathbf{z}_{S-1} = \mathbf{x}_{S-1} + f_{S-1}(\mathbf{x}_{S-1},\ t_0 \mid \mathbf{x}_{S-2}) \\ \vdots \\ \mathbf{z}_1 = \mathbf{x}_1 + f_1(\mathbf{x}_1,\ t_0 \mid \mathbf{x}_0) \\ \mathbf{z}_0 = \mathbf{x}_0 + f_0(\mathbf{x}_0,\ t_0) \end{cases} \tag{18}$$

E SIMPLE EXAMPLE OF GENERATION

For example, if we use Euler method as our ODE solver, for generation Equation 2 reduces to:

$$\mathbf{v}(t_0) = \mathbf{v}(t_1) + (t_0 - t_1) f_s(\mathbf{v}(t_1), t_1 \mid \mathbf{c}) \tag{19}$$

i.e. the state is integrated backwards from t_1 (i.e. $\mathbf{z}_s$) to t_0 (i.e. $\mathbf{x}_s$). We start at scale 0 with a noise sample $\mathbf{z}_0$, and assume t_0 and t_1 are 0 and 1 respectively:

$$\begin{cases} \mathbf{x}_0 = \mathbf{z}_0 - f_0(\mathbf{z}_0,\ t_1) \\ \mathbf{x}_1 = \mathbf{z}_1 - f_1(\mathbf{z}_1,\ t_1 \mid \mathbf{x}_0) \\ \vdots \\ \mathbf{x}_{S-1} = \mathbf{z}_{S-1} - f_{S-1}(\mathbf{z}_{S-1},\ t_1 \mid \mathbf{x}_{S-2}) \\ \mathbf{x}_S = \mathbf{z}_S - f_S(\mathbf{z}_S,\ t_1 \mid \mathbf{x}_{S-1}) \end{cases} \tag{20}$$

F MODELS

We used the same neural network architecture as in RNODE Finlay et al. (2020). The CNF at each resolution consists of a stack of bl blocks of a 4-layer deep convolutional network comprised of 3x3 kernels and softplus activation functions, with 64 hidden dimensions, and time t concatenated to the spatial input. In addition, except at the coarsest resolution, the immediate coarser image is also concatenated with the state. The integration time of each piece is [0, 1]. The number of blocks bl and the corresponding total number of parameters are given in Table 6.

Table 6: Number of parameters for different models with different total number of resolutions (res), and the number of channels (ch) and number of blocks (bl) per resolution.

MRCNF			
resolutions	ch	bl	Param
1	64	2	0.16M
	64	4	0.32M
	64	14	1.10M
2	64	8	1.33M
	64	20	3.34M
	64	40	6.68M
3	64	6	1.53M
	64	8	2.04M
	64	20	5.10M

G GRADIENT NORM

In order to avoid exploding gradients, We clipped the norm of the gradients Pascanu et al. (2013) by a maximum value of 100.0. In case of using adversarial loss, we first clip the gradients provided by the adversarial loss by 50.0, sum up the gradients provided by the log-likelihood loss, and then clip the summed gradients by 100.0.

H 8-BIT TO UNIFORM

The change-of-variables formula gives the change in probability due to the transformation of $\mathbf{u}$ to $\mathbf{v}$:

$$\log p(\mathbf{u}) = \log p(\mathbf{v}) + \log \left| \det \frac{\mathrm{d}\mathbf{v}}{\mathrm{d}\mathbf{u}} \right|$$

Specifically, the change of variables from an 8-bit image to an image with pixel values in range [0, 1] is:

$$\begin{aligned}
\mathbf{b}_S^{(p)} &= \frac{\mathbf{a}_S^{(p)}}{256} \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) + \log \left| \det \frac{\mathrm{d}\mathbf{b}}{\mathrm{d}\mathbf{a}} \right| \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) + \log \left(\frac{1}{256} \right)^{D_S} \\
&\implies \log p(\mathbf{a}_S) = \log p(\mathbf{b}_S) - D_S \log 256 \\
\implies \text{bpd}(\mathbf{a}_S) &= \frac{-\log p(\mathbf{a}_S)}{D_S \log 2} \\
&= \frac{-(\log p(\mathbf{b}_S) - D_S \log 256)}{D_S \log 2} \\
&= \frac{-\log p(\mathbf{b}_S)}{D_S \log 2} + \frac{\log 256}{\log 2} \\
&= \text{bpd}(\mathbf{x}) + 8
\end{aligned}$$

where $\text{bpd}(\mathbf{x})$ is given from Equation 13.

I FID V/S TEMPERATURE

Table 7 lists the FID values of generated images from MRCNF models trained on CIFAR10, with different temperature settings on the Gaussian.

	Temperature					
	1.0	**0.9**	**0.8**	**0.7**	**0.6**	**0.5**
1-resolution CNF	138.82	147.62	175.93	284.75	405.34	466.16
2-resolution MRCNF	89.55	106.21	171.53	261.64	370.38	435.17
3-resolution MRCNF	88.51	104.39	152.82	232.53	301.89	329.12
4-resolution MRCNF	92.19	104.35	135.58	186.71	250.39	313.39

Table 7: FID v/s temperature for MRCNF models trained on CIFAR10.