# Residual Flows for Invertible Generative Modeling

**Ricky T. Q. Chen**[1,3], **Jens Behrmann**[2], **David Duvenaud**[1,3], **Jörn-Henrik Jacobsen**[1,3]
University of Toronto[1], University of Bremen[2], Vector Institute[3]
rtqichen@cs.toronto.edu, jensb@uni-bremen.de
duvenaud@cs.toronto.edu, j.jacobsen@vectorinstitute.ai

## Abstract

Flow-based generative models parameterize probability distributions through an invertible transformation and can be trained by maximum likelihood. Invertible residual networks provide a flexible family of transformations where only Lipschitz conditions rather than strict architectural constraints are needed for enforcing invertibility. However, prior work trained invertible residual networks for density estimation by relying on biased log-density estimates whose bias increased with the network's expressiveness. We give a tractable unbiased estimate of the log density, and reduce the memory required during training by a factor of ten. Furthermore, we improve invertible residual blocks by proposing the use of activation functions that avoid gradient saturation and generalizing the Lipschitz condition to induced mixed norms. The resulting approach, called Residual Flows, achieves state-of-the-art performance on density estimation amongst flow-based models, and outperforms networks that use coupling blocks at joint generative and discriminative modeling.

## 1 Introduction

Maximum likelihood is a core machine learning paradigm that poses learning as a distribution alignment problem. However, it is often unclear what family of distributions should be used to fit high-dimensional continuous data. In this regard, the change of variables theorem offers an appealing way to construct flexible distributions that allow tractable exact sampling and efficient evaluation of its density. This class of models is generally referred to as invertible or flow-based generative models (Deco and Brauer, 1995; Rezende and Mohamed, 2015).

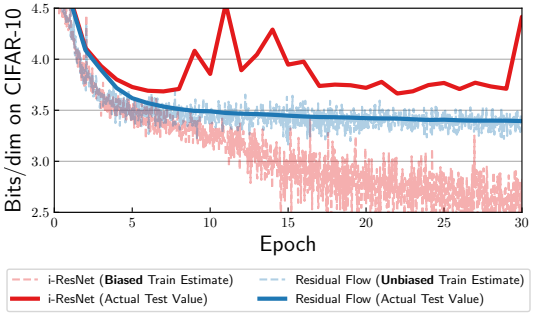



Figure 1: i-ResNets suffer from substantial bias when using expressive networks, whereas Residual Flows principledly perform maximum likelihood with unbiased stochastic gradients.

With invertibility as its core design principle, flow-based models have shown to be capable of generating realistic images (Kingma and Dhariwal, 2018) and can achieve density estimation performance on-par with competing state-of-the-art approaches (Ho et al., 2019). In applications, they have been applied to study adversarial robustness (Jacobsen et al., 2019) and are used to train hybrid models with both generative and classification capabilities (Nalisnick et al., 2019) using a weighted maximum likelihood objective.

However, existing high-dimensional flow-based models rely on specialized architectures such as coupling blocks (Dinh et al., 2014, 2017) or solving a differential equation (Grathwohl et al., 2019).

Preprint. Under review.

Such approaches have a strong inductive bias that can hinder their application in other tasks, such as learning representations that are suitable for both generative and discriminative tasks.

Recent work by Behrmann et al. (2019) showed that residual networks (He et al., 2016) can be made invertible by simply enforcing a Lipschitz constraint, allowing to use a very successful discriminative deep network architecture for unsupervised flow-based modeling. Unfortunately, the density evaluation requires computing an infinite series. The choice of a fixed truncation estimator used by Behrmann et al. (2019) leads to substantial bias that is tightly coupled with the expressiveness of the network, and cannot be said to be performing maximum likelihood as bias is introduced in the objective and gradients.

In this work, we introduce Residual Flows, a flow-based generative model that produces an unbiased estimate of the log density and has memory-efficient backpropagation through the log density computation. This allows us to use expressive architectures and train via maximum likelihood. Furthermore, we propose and experiment with the use of activations functions that avoid gradient saturation and induced mixed norms for Lipschitz-constrained neural networks.

## 2 Background

**Maximum likelihood estimation.** To perform maximum likelihood with stochastic gradient descent, it is sufficient to have an unbiased estimator for the gradient as

$$\nabla_\theta D_{\text{KL}}(p_{\text{data}} \,||\, p_\theta) = \nabla_\theta \mathbb{E}_{x\sim p_{\text{data}}(x)} \left[\log p_\theta(x)\right] = \mathbb{E}_{x\sim p_{\text{data}}(x)} \left[\nabla_\theta \log p_\theta(x)\right], \tag{1}$$

where $p_{\text{data}}$ is the unknown data distribution which can be sampled from and $p_\theta$ is the model distribution. An unbiased estimator of the gradient also immediately follows from an unbiased estimator of the log density function, $\log p_\theta(x)$.

**Change of variables theorem.** With an invertible transformation $f$, the change of variables

$$\log p(x) = \log p(f(x)) + \log \left| \det \frac{df(x)}{dx} \right| \tag{2}$$

captures the change in density of the transformed samples. A simple base distribution such as a standard normal is often used for $\log p(f(x))$. Tractable evaluation of (2) allows flow-based models to be trained using the maximum likelihood objective (1). In contrast, variational autoencoders (Kingma and Welling, 2014) can only optimize a stochastic lower bound, and generative adversial networks (Goodfellow et al., 2014) require an extra discriminator network for training.

**Invertible residual networks (i-ResNets).** Residual networks are composed of simple transformations $y = f(x) = x + g(x)$. Behrmann et al. (2019) noted that this transformation is invertible by the Banach fixed point theorem if $g$ is contractive, i.e. with Lipschitz constant strictly less than unity, which was enforced using spectral normalization (Miyato et al., 2018; Gouk et al., 2018).

Applying i-ResNets to the change-of-variables (2), the identity

$$\log p(x) = \log p(f(x)) + \text{tr}\left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} [J_g(x)]^k\right) \tag{3}$$

was shown, where $J_g(x) = \frac{dg(x)}{dx}$. Furthermore, the Skilling-Hutchinson estimator (Skilling, 1989; Hutchinson, 1990) was used to to estimate the trace in the power series. Behrmann et al. (2019) used a fixed truncation to approximate the infinite series in (3). The Lipschitz constant of $g$ bounds the spectral radius of $J_g$, and therefore determines the convergence rate of this power series. As such, the fixed truncation estimator introduces substantial bias in the log density and is unreliable if the Lipschitz constant is close to one, thus requiring a careful balance between bias and expressiveness. Without decoupling the objective and estimation bias, i-ResNets end up optimizing for the bias without improving the actual maximum likelihood objective (see Figure 1).

## 3 Residual Flows

### 3.1 Unbiased Log Density Estimation for Maximum Likelihood Estimation

Evaluation of the exact log density function $\log p_\theta(\cdot)$ in (3) requires infinite time due to the power series. Instead, we rely on randomization to derive an unbiased estimator that can be computed in finite time (with probability one) based on an existing concept (Kahn, 1955).

To illustrate the idea, let $\Delta_k$ denote the $k$-th term of an infinite series, and suppose we always evaluate the first term then flip a coin $b \sim \text{Bernoulli}(q)$ to determine whether we stop or continue evaluating the remaining terms. By reweighting the remaining terms by $\frac{1}{1-q}$, we obtain an unbiased estimator

$$\Delta_1 + \mathbb{E}\left[\left(\frac{\sum_{k=2}^{\infty}\Delta_k}{1-q}\right)\mathbb{1}_{b=0} + (0)\mathbb{1}_{b=1}\right] = \Delta_1 + \frac{\sum_{k=2}^{\infty}\Delta_k}{1-q}(1-q) = \sum_{k=1}^{\infty}\Delta_k.$$

This unbiased estimator has probability $q$ of being evaluated in finite time. We can obtain an estimator that is evaluated in finite time with probability one by applying this process infinitely many times to the remaining terms. Directly sampling the number of evaluated terms, we obtain the appropriately named "Russian roulette" estimator (Kahn, 1955)

$$\sum_{k=1}^{\infty}\Delta_k = \mathbb{E}_{n\sim p(N)}\left[\sum_{k=1}^{n}\frac{\Delta_k}{\mathbb{P}(N\geq k)}\right]. \tag{4}$$

We note that the explanation above is only meant to be an intuitive guide and not a formal derivation. The peculiarities of dealing with infinite quantities dictate that we must make assumptions on $\Delta_k$, $p(N)$, or both in order for the equality in (4) to hold. While many existing works have made different assumptions depending on specific applications of (4), we state our result as a theorem where the only condition is that $p(N)$ must have support over all of the indices.

**Theorem 1** (Unbiased log density estimator)**.** *Let $f(x) = x + g(x)$ with $\text{Lip}(g) < 1$ and $N$ be a random variable with support over the positive integers. Then*

$$\log p(x) = \log p(f(x)) + \mathbb{E}_{n,v}\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}\frac{v^T[J_g(x)^k]v}{\mathbb{P}(N\geq k)}\right], \tag{5}$$

*where $n \sim p(N)$ and $v \sim \mathcal{N}(0, I)$.*

Here we have also used the Skilling-Hutchinson trace estimator (Skilling, 1989; Hutchinson, 1990) to estimate the trace of the matrices $J_g^k$. A detailed proof is given in Appendix B.

Note that since $J_g$ is constrained to have a spectral radius less than unity, the power series converges exponentially. The variance of the Russian roulette estimator is small when the infinite series exhibits fast convergence (Rhee and Glynn, 2015; Beatson and Adams, 2019), and in practice, we did not have to tune $p(N)$ for variance reduction. Instead, in our experiments, we compute two terms exactly and then use the unbiased estimator on the remaining terms with a single sample from $p(N) = \text{Geom}(0.5)$. This results in an expected compute cost of 4 terms, which is less than the 5 to 10 terms that Behrmann et al. (2019) used for their biased estimator.

Theorem 1 forms the core of Residual Flows, as we can now perform maximum likelihood training by backpropagating through (5) to obtain unbiased gradients. The price we pay for the unbiased estimator is variable compute and memory, as each sample of the log density uses a random number of terms in the power series.

### 3.2 Memory-Efficient Backpropagation

Memory can be a scarce resource, and running out of memory due to a large sample from the unbiased estimator can halt training unexpectedly. To this end, we propose two methods to reduce the memory consumption during training.

To see how naïve backpropagation can be problematic, the gradient w.r.t. parameters $\theta$ by directly differentiating through the power series (5) can be expressed as

$$\frac{\partial}{\partial\theta}\log\det\left(I + J_g(x,\theta)\right) = \mathbb{E}_{n,v}\left[\sum_{k=1}^{n}\frac{(-1)^{k+1}}{k}\frac{\partial v^T(J_g(x,\theta)^k)v}{\partial\theta}\right]. \tag{6}$$

Table 1: Memory usage (GB) per minibatch of 64 samples when computing $n$=10 terms in the corresponding power series. †Uses immediate downsampling before any residual blocks.

| | MNIST | | CIFAR-10† | | CIFAR-10 | | |
|---|---|---|---|---|---|---|---|
| | ELU | LipSwish | ELU | LipSwish | ELU | LipSwish | Relative |
| Naïve Backprop | 92.0 | 192.1 | 33.3 | 66.4 | 120.2 | 263.5 | 100% |
| Neumann Gradient | 13.4 | 31.2 | 5.5 | 11.3 | 17.6 | 40.8 | 15.7% |
| Backward-in-Forward | 8.7 | 19.8 | 3.8 | 7.4 | 11.5 | 26.1 | 10.3% |
| Both Combined | 4.9 | 13.6 | 3.0 | 5.9 | 6.6 | 18.0 | 7.1% |

Unfortunately, this estimator requires each term to be stored in memory because $\partial/\partial\theta$ needs to be applied to each term. The total memory cost is then $\mathcal{O}(n \cdot m)$ where $n$ is the number of computed terms and $m$ is the number of residual blocks in the entire network. This is extremely memory-hungry during training, and a large random sample of $n$ can occasionally result in running out of memory.

**Neumann gradient series.** Instead, we can specifically express the gradients as a power series derived from a Neumann series (see Appendix C). Applying the Russian roulette and trace estimators, we obtain the following theorem.

**Theorem 2** (Unbiased log-determinant gradient estimator)**.** *Let* $\mathrm{Lip}(g) < 1$ *and* $N$ *be a random variable with support over positive integers. Then*

$$\frac{\partial}{\partial\theta}\log\det\left(I + J_g(x,\theta)\right) = \mathbb{E}_{n,v}\left[\left(\sum_{k=0}^{n}\frac{(-1)^k}{\mathbb{P}(N \geq k)}\, v^T J(x,\theta)^k\right)\frac{\partial(J_g(x,\theta))}{\partial\theta}v\right], \tag{7}$$

*where* $n \sim p(N)$ *and* $v \sim \mathcal{N}(0, I)$.

As the power series in (7) does not need to be differentiated through, using this reduces the memory requirement by a factor of $n$. This is especially useful when using the unbiased estimator as the memory will be constant regardless of the number of terms we draw from $p(N)$.

**Backward-in-forward: early computation of gradients.** We can further reduce memory by partially performing backpropagation during the forward evaluation. By taking advantage of $\log\det(I + J_g(x,\theta))$ being a scalar quantity, the partial derivative from the objective $\mathcal{L}$ is

$$\frac{\partial\mathcal{L}}{\partial\theta} = \underbrace{\frac{\partial\mathcal{L}}{\partial\log\det(I + J_g(x,\theta))}}_{\text{scalar}}\underbrace{\frac{\partial\log\det(I + J_g(x,\theta))}{\partial\theta}}_{\text{vector}}. \tag{8}$$

For every residual block, we compute $\partial\log\det(I+J_g(x,\theta))/\partial\theta$ along with the forward pass, release the memory for the computation graph, then simply multiply by $\partial\mathcal{L}/\partial\log\det(I+J_g(x,\theta))$ later during the main backprop. This reduces memory by another factor of $m$ to $\mathcal{O}(1)$ with negligible overhead.

Note that while these two tricks remove the memory cost from backpropagating through the $\log\det$ terms, computing the path-wise derivatives from $\log p(f(x))$ still requires the same amount of memory as a single evaluation of the residual network. Table 1 shows that the memory consumption can be enormous for naïve backpropagation, and using large networks would have been intractable.

### 3.3 Avoiding Gradient Saturation with the LipSwish Activation Function

As the log density depends on the first derivatives through the Jacobian $J_g$, the gradients for training depend on second derivatives. Similar to the phenomenon of saturated activation functions, Lipschitz-constrained activation functions can have a gradient saturation problem. For instance, the ELU activation used by Behrmann et al. (2019) achieves the highest Lipschitz constant when $\mathrm{ELU}'(z) = 1$, but this occurs when the second derivative is exactly zero in a very large region, implying there is a trade-off between a large Lipschitz constant and non-vanishing gradients.

We thus desire two properties from our activation functions $\phi(z)$:

1. The first derivatives must be bounded as $|\phi'(z)| \leq 1$ for all $z$
2. The second derivatives should not asymptotically vanish when $|\phi'(z)|$ is close to one.

While many activation functions satisfy condition 1, most do not satisfy condition 2. We argue that the ELU and softplus activations are suboptimal due to gradient saturation.

We find that good activation functions satisfying condition 2 are *smooth and non-monotonic* functions, such as Swish (Ramachandran et al., 2017). However, Swish by default does not satisfy condition 1 as $\max_z |\frac{d}{dz}\text{Swish}(z)| \lesssim 1.1$. But scaling via

$$\text{LipSwish}(z) := \text{Swish}(z)/1.1 = z \cdot \sigma(\beta z)/1.1, \tag{9}$$

where $\sigma$ is the sigmoid function, results in $\max_z |\frac{d}{dz}\text{LipSwish}(z)| \leq 1$ for all values of $\beta$. LipSwish is a simple modification to Swish that exhibits a less than unity Lipschitz property. In our experiments, we parameterize $\beta$ to be strictly positive by passing it through softplus. One caveat of using LipSwish is it cannot be computed in-place, resulting in double the amount of memory usage as ELU (Table 1).

### 3.4 Generalizing i-ResNets via Induced Mixed Norms

Behrmann et al. (2019) used spectral normalization (Miyato et al., 2018) (which relies on power iteration to approximate the spectral norm) to enforce the Lipschitz constraint on $g$. Specifically, this bounds the spectral norm of the Jacobian $J_g$ by the sub-multiplicativity property. If $g(x)$ is a neural network with pre-activation defined recursively using $z_l = W_l h_{l-1} + b_l$ and $h_l = \phi(z_l)$, with $x = z_0, g(x) = z_L$, then the data-independent upper bound

$$||J_g||_2 = ||W_L \phi'(z_{L-1}) \cdots W_2 \phi'(z_1) W_1 \phi'(z_0)||_2 \leq ||W_1||_2 \cdots ||W_L||_2 \tag{10}$$

holds, where $\phi'(z)$ are diagonal matrices containing the first derivatives of the activation functions. The inequality in (10) is a result of using a sub-multiplicative norm and assuming that the activation functions have Lipschitz less than unity. However, any induced matrix norm satisfies the sub-multiplicativity property, including mixed norms $||W||_{p\to q} := \sup_{x\neq 0} {}^{||Wx||_q}/_{||x||_p}$, where the input and output spaces have different vector norms.

As long as $g(x)$ maps back to the original normed (complete) vector space, the Banach fixed point theorem used in the proof of invertibility of residual blocks (Behrmann et al., 2019) still holds. As such, we can choose arbitrary $p_0, \ldots, p_{L-2} \in [1, \infty]$ such that

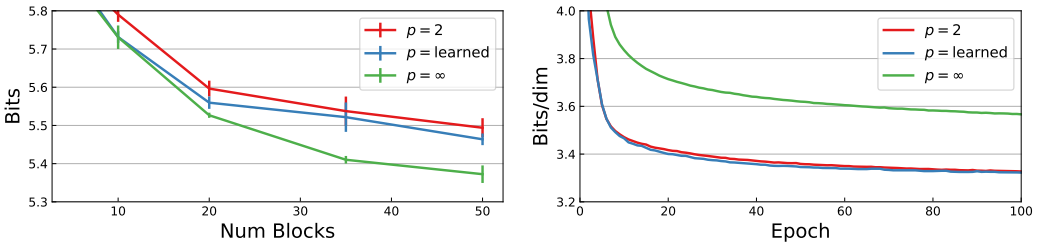
$$||J_g||_{p_0} \leq ||W_1||_{p_0 \to p_1} ||W_2||_{p_1 \to p_2} \cdots ||W_L||_{p_{L-2} \to p_0}. \tag{11}$$

We use a more general form of power iteration (Johnston, 2016) for estimating induced mixed norms, which becomes the standard power iteration for $p = q = 2$. Furthermore, the special cases where $p_l = 1$ or $p_l = \infty$ are of particular interest, as the matrix norms can be computed exactly (Tropp, 2004). Additionally, we can also optimize the norm orders during training by backpropagating through the modified power method. Lastly, we emphasize that the convergence of the infinite series (3) is guaranteed for any induced matrix norm, as they still upper bound the spectral radius (Horn and Johnson, 2012).

Figure 2a shows that we obtain some performance gain by using either learned norms or the infinity norm on a difficult 2D dataset, where similar performance can be achieved by using fewer residual blocks. While the infinity norm works well with fully connected layers, we find that it does not perform as well as the spectral norm for convolutional networks. Instead, Figure 2b shows that learned norms obtain marginal improvement on CIFAR-10. Ultimately, while the idea of generalizing spectral normalization via learnable norm orders is interesting in its own right to be communicated here, we found that the improvements are very marginal. More details are in Appendix D.

## 4 Related Work

**Estimation of Infinite Series.** Our derivation of the unbiased estimator follows from the general approach of using a randomized truncation (Kahn, 1955). This paradigm of estimation has been repeatedly rediscovered and applied in many fields, including solving of stochastic differential equations (McLeish, 2011; Rhee and Glynn, 2012, 2015), ray tracing for rendering paths of light (Arvo and Kirk, 1990), and estimating limiting behavior of optimization problems (Tallec and Ollivier,




(a) Checkerboard 2D

(b) CIFAR-10

Figure 2: **Lipschitz constraints with different induced matrix norms**. (a) On 2D density, both learned and $\infty$-norm improve upon spectral norm, allowing more expressive models with fewer blocks. Standard deviation across 3 random seeds. (b) On CIFAR-10, learning the norm orders give a small performance gain and the $\infty$-norm performs much worse than spectral norm ($p = 2$). Comparisons are made using identical initialization.

2017; Beatson and Adams, 2019), among many other applications. Some recent works use Chebyshev polynomials to estimate the spectral functions of symmetric matrices Han et al. (2018); Adams et al. (2018); Ramesh and LeCun (2018); Boutsidis et al. (2008). These works estimate quantities that are similar to those presented in this work, but a key difference is that the Jacobian in our power series is not symmetric. We also note works that have rediscovered the random truncation approach (McLeish, 2011; Rhee and Glynn, 2015; Han et al., 2018) made assumptions on $p(N)$ in order for it to be applicable to general infinite series. Fortunately, since the power series in Theorems 1 and 2 converge fast enough, we were able to make use of a different set of assumptions requiring only that $p(N)$ has sufficient support, which was adapted from Bouchard-Côté (2018) (details in Appendix B).

**Memory-efficient Backpropagation.** The issue of computing gradients in a memory-efficient manner was explored by Gomez et al. (2017) and Chang et al. (2018) for residual networks with an architecture devised by Dinh et al. (2014), and explored by Chen et al. (2018) for a continuous analogue of residual networks. These works focus on the path-wise gradients from the output of the network, whereas we focus on the gradients from the log-determinant term in the change of variables equation specifically for generative modeling.

**Invertible Deep Networks.** Flow-based generative models are a density estimation approach which has invertibility as its core design principle (Rezende and Mohamed, 2015; Deco and Brauer, 1995). Most recent work on flows focuses on designing maximally expressive architectures while maintaining invertibility and tractable log determinant computation (Dinh et al., 2014, 2017; Kingma and Dhariwal, 2018). An alternative route has been taken by Neural ODEs which show that a Jacobian trace can be computed instead of log determinants, if the transformation is specified by an ordinary differential equation (Chen et al., 2018; Grathwohl et al., 2019). Invertible architectures are also of interest for discriminative problems, as their information-preservation properties make them suitable candidates to analyze and regularize learned representations (Jacobsen et al., 2019).

# 5 Experiments

## 5.1 Density & Generative Modeling

We use a similar architecture as Behrmann et al. (2019), except without the immediate invertible downsampling (Dinh et al., 2017) at the image pixel-level. Removing this substantially increases the amount of memory required (shown in Table 1) as there are more spatial dimensions at every layer, but increases the overall performance. We also increase the bound on the Lipschitz constants of each weight matrix to 0.98, whereas Behrmann et al. (2019) used 0.90 to reduce the error of the biased estimator. More detailed explanations are in Appendix E.

Unlike prior works that use multiple GPUs, large batch sizes, and a few hundred epochs, Residual Flow models are trained with the standard batch size of 64 and converges in roughly 300-350 epochs for MNIST and CIFAR-10. Most network settings can fit on a single GPU (see Table 1), though we use 4 GPUs in our experiments to speed up training. In comparison, Glow was trained for 1800 epochs

Table 2: Results [bits/dim] on standard benchmark datasets for density estimation. In brackets are models that used "variational dequantization" (Ho et al., 2019), which we don't compare against.

| Model | MNIST | CIFAR-10 | ImageNet 32×32 | ImageNet 64×64 |
|---|---|---|---|---|
| Real NVP (Dinh et al., 2017) | 1.06 | 3.49 | 4.28 | 3.98 |
| Glow (Kingma and Dhariwal, 2018) | 1.05 | 3.35 | 4.09 | 3.81 |
| FFJORD (Grathwohl et al., 2019) | 0.99 | 3.40 | — | — |
| Flow++ (Ho et al., 2019) | — | **3.29** (3.09) | — (3.86) | — (3.69) |
| i-ResNet (Behrmann et al., 2019) | 1.05 | 3.45 | — | — |
| Residual Flow (Ours) | **0.97** | **3.29** | **4.02** | **3.78** |

Figure 3: **Qualitative samples.** Real (left) and random samples (right) from a model trained on 5bit 64×64 CelebA. The most visually appealing samples were picked out of 5 random batches.

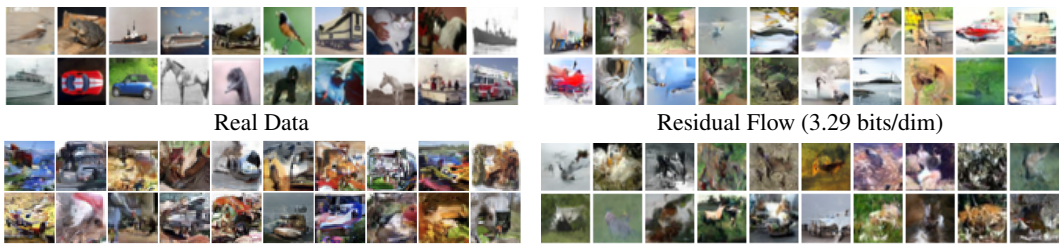
(Kingma and Dhariwal, 2018) and Flow++ reported to not have fully converged after 400 epochs (Ho et al., 2019). We exceed Glow's performance (3.35) on CIFAR-10 at around 50 epochs (Figure 5).

Table 2 reports the bits per dimension ($\log_2 p(x)/d$ where $x \in \mathbb{R}^d$) on standard benchmark datasets MNIST, CIFAR-10, and downsampled ImageNet. We achieve competitive performance to state-of-the-art flow-based models on both datasets. For evaluation, we computed 20 terms of the power series (3) and use the unbiased estimator (5) to estimate the remaining terms. This reduces the standard deviation of the unbiased estimator to a negligible level.

We are also competitive with state-of-the-art flow-based models in regards to sample quality. Figure 3 shows random samples from the model trained on CelebA. Furthermore, samples from Residual Flow trained on CIFAR-10 are slightly more globally coherent (Figure 4) than PixelCNN and variational dequantized Flow++, even though our likelihood is worse. This is only an informal comparison, and it is well-known that visual fidelity and log-likelihood are not necessarily indicative of each other (Theis et al., 2015), but we believe residual networks may have better inductive bias than coupling blocks or autoregressive architectures as generative models. More samples are in Appendix A.

### 5.2 Ablation Experiments

We report ablation experiments for the unbiased estimator and the LipSwish activation function in Table 3. Even in settings where the Lipschitz constant and bias are relatively low, we observe a significant improvement from using the unbiased estimator. Training the larger i-ResNet model on CIFAR-10 results in the biased estimator completely ignoring the actual likelihood objective altogether. In this setting, the biased estimate was lower than 0.8 bits/dim by 50 epochs, but the actual



Real Data | Residual Flow (3.29 bits/dim)

PixelCNN (3.14 bits/dim) | Variational Dequantized Flow++ (3.08 bits/dim)

Figure 4: Random samples from CIFAR-10 models.

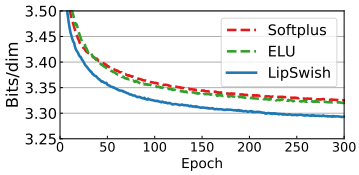



Figure 5: Effect of activation functions on CIFAR-10.

| Training Setting | MNIST | CIFAR-10† | CIFAR-10 |
|---|---|---|---|
| i-ResNet + ELU | 1.05 | 3.45 | 3.66∼4.78 |
| Residual Flow + ELU | 1.00 | 3.40 | 3.32 |
| Residual Flow + LipSwish | **0.97** | **3.39** | **3.29** |

Table 3: Ablation results. †Uses immediate downsampling before any residual blocks.

Table 4: Comparison of residual vs. coupling blocks for the hybrid modeling task.

| | MNIST | | | | | SVHN | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | $\lambda = 0$ | $\lambda = 1/D$ | | $\lambda = 1$ | | $\lambda = 0$ | $\lambda = 1/D$ | | $\lambda = 1$ | |
| Block Type | Acc↑ | BPD↓ | Acc↑ | BPD↓ | Acc↑ | Acc↑ | BPD↓ | Acc↑ | BPD↓ | Acc↑ |
| Nalisnick et al. (2019) | 99.33% | 1.26 | 97.78% | – | – | 95.74% | 2.40 | 94.77% | – | – |
| Coupling | 99.50% | 1.18 | 98.45% | 1.04 | 95.42% | 96.27% | 2.73 | 95.15% | 2.21 | 46.22% |
| + $1 \times 1$ Conv | **99.56%** | 1.15 | 98.93% | 1.03 | 94.22% | **96.72%** | 2.61 | 95.49% | 2.17 | 46.58% |
| Residual | 99.53% | **1.01** | **99.46%** | **0.99** | **98.69%** | **96.72%** | **2.29** | **95.79%** | **2.06** | **58.52%** |

bits/dim wildly oscillates above 3.66 bits/dim and seems to never converge. Using LipSwish not only converges much faster but also results in better performance compared to softplus or ELU, especially in the high Lipschitz settings (Figure 5 and Table 3).

### 5.3 Hybrid Modeling

Next, we experiment on joint training of continuous and discrete data. Of particular interest is the ability to learn both a generative model and a classifier, referred to as a hybrid model (Nalisnick et al., 2019). Let $x$ be the data and $y$ be a categorical random variable. The maximum likelihood objective can be separated into $\log p(x, y) = \log p(x) + \log p(y|x)$, where $\log p(x)$ is modeled using a flow-based generative model and $\log p(y|x)$ is a classifier network that shares learned features from the generative model. However, it is often the case that accuracy is the metric of interest and log-likelihood is only used as a surrogate training objective. In this case, (Nalisnick et al., 2019) suggests a weighted maximum likelihood objective,

$$\mathbb{E}_{(x,y)\sim p_{\text{data}}}[\lambda \log p(x) + \log p(y|x)], \tag{12}$$

where $\lambda$ is a scaling constant. As $y$ is much lower dimensional than $x$, setting $\lambda < 1$ emphasizes classification, and setting $\lambda = 0$ results in a classification-only model which can be compared against.

Table 5: Hybrid modeling results on CIFAR-10.

| | $\lambda = 0$ | $\lambda = 1/D$ | | $\lambda = 1$ | |
|---|---|---|---|---|---|
| Block Type | Acc↑ | BPD↓ | Acc↑ | BPD↓ | Acc↑ |
| Coupling | 89.77% | 4.30 | 87.58% | 3.54 | 67.62% |
| + $1 \times 1$ Conv | 90.82% | 4.09 | 87.96% | 3.47 | 67.38% |
| Residual | **91.78%** | **3.62** | **90.47%** | **3.39** | **70.32%** |

As Nalisnick et al. (2019) performs approximate Bayesian inference and uses different architectures than us, we perform our own experiments to compare residual blocks to coupling blocks Dinh et al. (2017) as well as $1\times1$ convolutions (Kingma and Dhariwal, 2018). We use the same architecture as the density estimation experiments and append a classification branch that takes features at the final output of multiple scales (see details in Appendix E). For comparisons to coupling blocks, we use the same architecture for $g(x)$ except we use ReLU activations and no longer constrain the Lipschitz constant.

Tables 4 & 5 show our experiment results. We outperform Nalisnick et al. (2019) on both pure classification and hybrid modeling. Furthermore, on MNIST we are able to obtain a decent classifier even when $\lambda = 1$. In general, we find that residual blocks perform much better than coupling blocks at learning representations for both generative and discriminative tasks. Coupling blocks have very high bits per dimension when $\lambda = 1/D$ while performing worse at classification when $\lambda = 1$, suggesting that they have restricted flexibility and can only perform one task well at a time.

## 6 Conclusion

We have shown that invertible residual networks can be turned into powerful generative models. The proposed unbiased flow-based generative model, coined Residual Flow, achieves competitive or better performance compared to alternative flow-based models in density estimation, sample quality, and hybrid modeling. More generally, we gave a recipe for introducing stochasticity in order to construct tractable flow-based models with a different set of constraints on layer architectures than competing approaches, which rely on exact log-determinant computations. This opens up a new design space of expressive but Lipschitz-constrained architectures that has yet to be explored.

## Acknowledgments

Jens Behrmann gratefully acknowledges the financial support from the German Science Foundation for RTG 2224 "$\pi^3$: Parameter Identification - Analysis, Algorithms, Applications"

## References

Ryan P Adams, Jeffrey Pennington, Matthew J Johnson, Jamie Smith, Yaniv Ovadia, Brian Patton, and James Saunderson. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.

James Arvo and David Kirk. Particle transport and image synthesis. *ACM SIGGRAPH Computer Graphics*, 24 (4):63–66, 1990.

Alex Beatson and Ryan P. Adams. Efficient optimization of loops and limits with randomized telescoping sums. In *International Conference on Machine Learning*, 2019.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Jörn-Henrik Jacobsen. Invertible residual networks. *International Conference on Machine Learning*, 2019.

Alexandre Bouchard-Côté. Topics in probability assignment 1. https://www.stat.ubc.ca/~bouchard/courses/stat547-fa2018-19//files/assignment1-solution.pdf, 2018. Accessed: 2019-05-22.

Christos Boutsidis, Michael W Mahoney, and Petros Drineas. Unsupervised feature selection for principal components analysis. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 61–69. ACM, 2008.

Bo Chang, Lili Meng, Eldad Haber, Lars Ruthotto, David Begert, and Elliot Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI Conference on Artificial Intelligence*, 2018.

Ricky T. Q. Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.

Gustavo Deco and Wilfried Brauer. Nonlinear higher-order statistical decorrelation by volume-conserving neural architectures. *Neural Networks*, 8(4):525–535, 1995.

Laurent Dinh, David Krueger, and Yoshua Bengio. NICE: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *International Conference on Learning Representations*, 2017.

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*, pages 2214–2224, 2017.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

Henry Gouk, Eibe Frank, Bernhard Pfahringer, and Michael Cree. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. *International Conference on Learning Representations*, 2019.

Insu Han, Haim Avron, and Jinwoo Shin. Stochastic chebyshev gradient descent for spectral optimization. In *Conference on Neural Information Processing Systems*. 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Jonathan Ho, Xi Chen, Aravind Srinivas, Yan Duan, and Pieter Abbeel. Flow++: Improving flow-based generative models with variational dequantization and architecture design. *International Conference on Machine Learning*, 2019.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge University Press, 2012.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 19(2):433–450, 1990.

Jörn-Henrik Jacobsen, Jens Behrmann, Richard Zemel, and Matthias Bethge. Excessive invariance causes adversarial vulnerability. *International Conference on Learning Representations*, 2019.

Nathaniel Johnston. QETLAB: A MATLAB toolbox for quantum entanglement, version 0.9. http://qetlab.com, January 2016.

Herman Kahn. Use of different monte carlo sampling techniques. 1955.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *International Conference on Learning Representations*, 2014.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

Don McLeish. A general method for debiasing a monte carlo estimator. *Monte Carlo Methods and Applications*, 17(4):301–315, 2011.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.

Eric Nalisnick, Akihiro Matsukawa, Yee Whye Teh, Dilan Gorur, and Balaji Lakshminarayanan. Hybrid models with deep and invertible features. *International Conference on Machine Learning*, 2019.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *International Conference on Machine Learning*, 2016.

K. B. Petersen and M. S. Pedersen. The matrix cookbook, 2012.

Boris T. Polyak and Anatoli Juditsky. Acceleration of stochastic approximation by averaging. 1992.

Prajit Ramachandran, Barret Zoph, and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

Aditya Ramesh and Yann LeCun. Backpropagation for implicit spectral densities. *Conference on Neural Information Processing Systems*, abs/1806.00499, 2018.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538, 2015.

Chang-han Rhee and Peter W Glynn. A new approach to unbiased estimation for sde's. In *Proceedings of the Winter Simulation Conference*, page 17. Winter Simulation Conference, 2012.

Chang-han Rhee and Peter W Glynn. Unbiased estimation with square root convergence for sde models. *Operations Research*, 63(5):1026–1043, 2015.

John Skilling. The eigenvalues of mega-dimensional matrices. In *Maximum Entropy and Bayesian Methods*, pages 455–466. Springer, 1989.

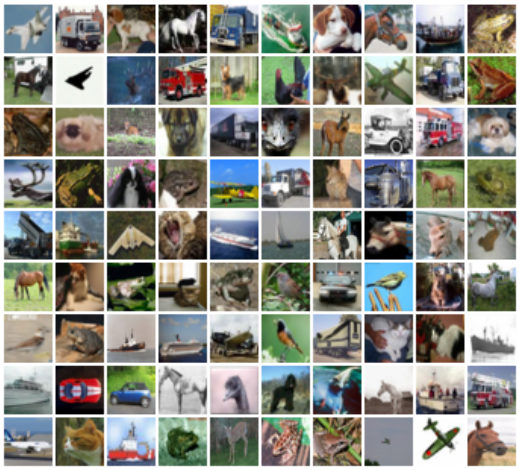
Corentin Tallec and Yann Ollivier. Unbiasing truncated backpropagation through time. *arXiv preprint arXiv:1705.08209*, 2017.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

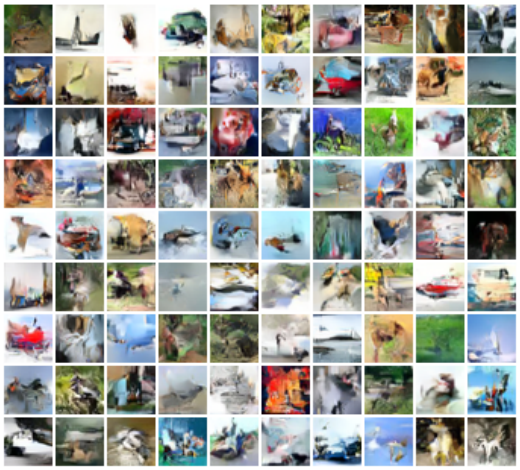
Joel Aaron Tropp. Topics in sparse approximation. In *PhD thesis*. University of Texas, 2004.

Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization. In *International Conference on Learning Representations*, 2019.

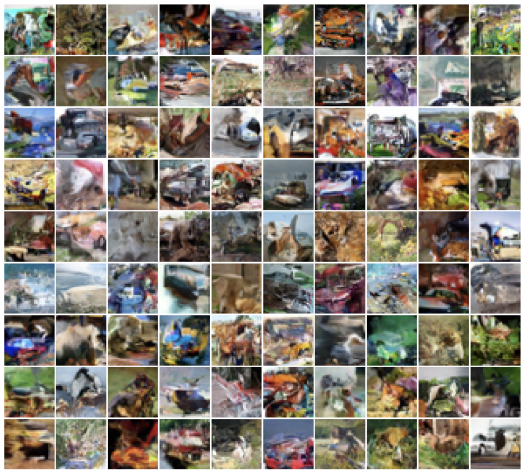
# A Random Samples



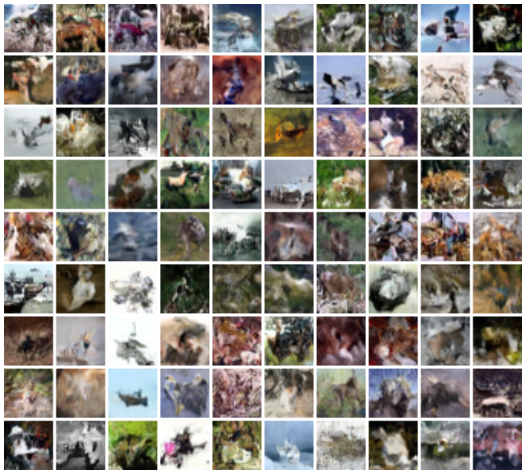
Real Data



Residual Flow

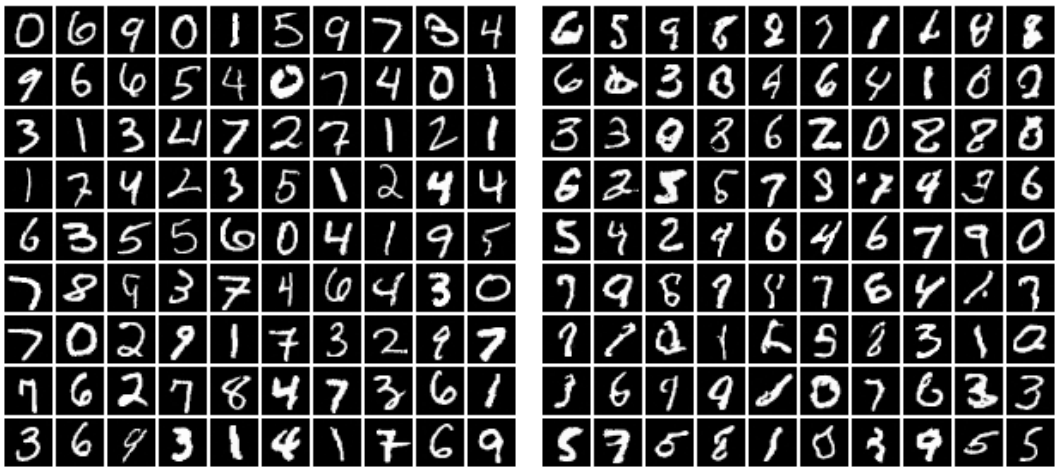


PixelCNN



Flow++

Figure 6: Random samples from CIFAR-10 models. PixelCNN (Oord et al., 2016) and Flow++ samples reprinted from Ho et al. (2019), with permission.

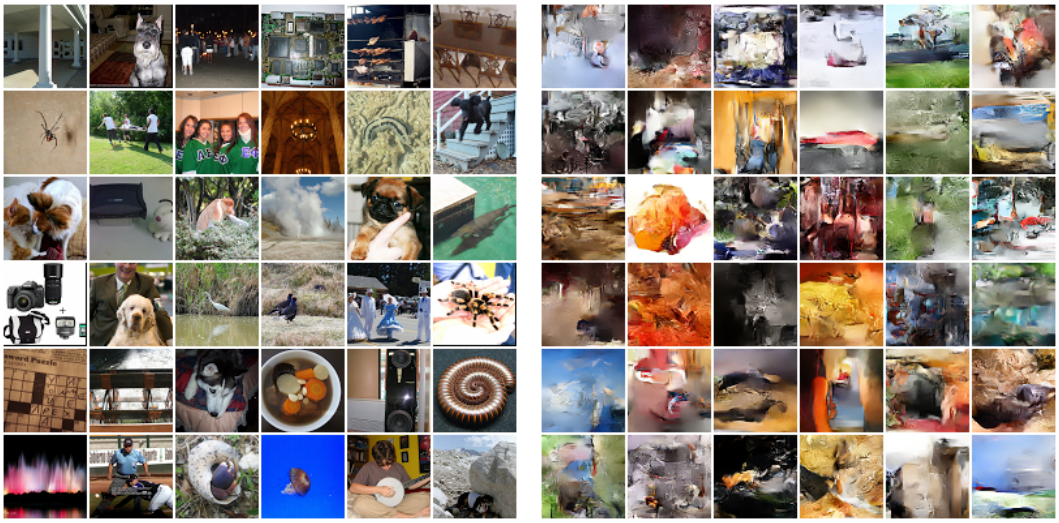


Real Data Residual Flow

Figure 7: Random samples from MNIST.

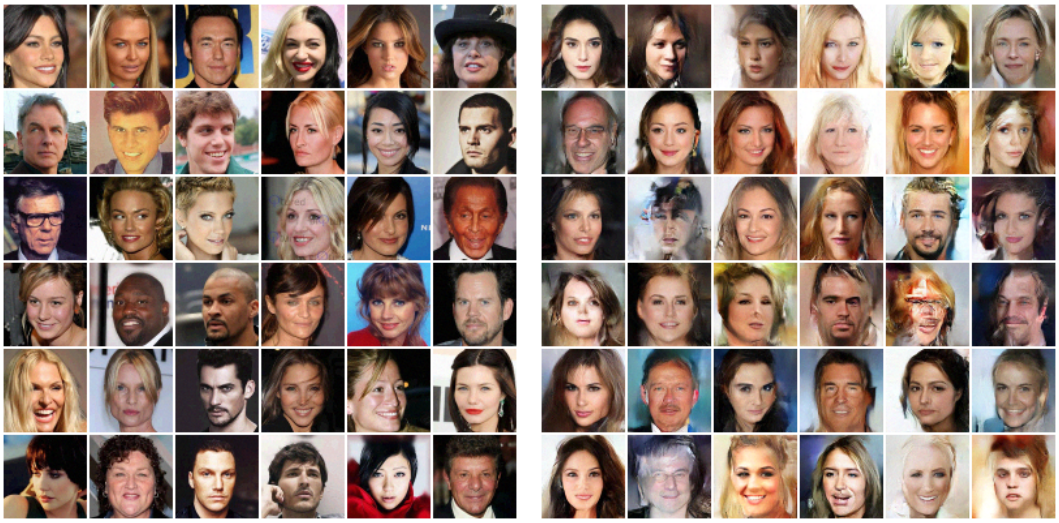
Real Data Residual Flow

Figure 8: Random samples from ImageNet 32×32.



Real Data Residual Flow

Figure 9: Random samples from ImageNet 64×64.



Real Data Residual Flow

Figure 10: Random samples from 5bit CelebA 64×64.

## B Proofs

We start by formulating a Lemma, which gives the condition when the randomized truncated series is an unbiased estimator in a fairly general setting. Afterwards, we study our specific estimator and prove that the assumption of the Lemma is satisfied.

Note, that similar conditions have been stated in previous works, e.g. in McLeish (2011) and Rhee and Glynn (2012). However, we use the condition from (Bouchard-Côté, 2018), which only requires $p(N)$ to have sufficient support.

To make the derivations self-contained, we reformulate the conditions from (Bouchard-Côté, 2018) in the following way:

**Lemma 3** (Unbiased randomized truncated series)**.** *Let $Y_k$ be a real random variable with $\lim_{k\to\infty} \mathbb{E}[Y_k] = a$ for some $a \in \mathbb{R}$. Further, let $\Delta_0 = Y_0$ and $\Delta_k = Y_k - Y_{k-1}$ for $k \geq 1$.*

*Assume*

$$\mathbb{E}\left[\sum_{k=0}^{\infty}|\Delta_k|\right] < \infty$$

*and let $N$ be a random variable with support over the positive integers and $n \sim p(N)$. Then for*

$$Z = \sum_{k=0}^{n}\frac{\Delta_k}{\mathbb{P}(N \geq k)},$$

*it holds*

$$a = \lim_{k\to\infty}\mathbb{E}[Y_k] = \mathbb{E}_{n\sim p(N)}[Z] = a.$$

*Proof.* First, denote

$$Z_M = \sum_{k=0}^{M}\frac{\mathbb{1}[N \geq k]\Delta_k}{\mathbb{P}(N \geq k)} \quad \text{and} \quad B_M = \sum_{k=0}^{M}\frac{\mathbb{1}[N \geq k]|\Delta_k|}{\mathbb{P}(N \geq k)},$$

where $|Z_M| \leq B_M$ by the triangle inequality. Since $B_M$ is non-decreasing, the monotone convergence theorem allows swapping the expectation and limit as $\mathbb{E}[B] = \mathbb{E}[\lim_{M\to\infty} B_M] = \lim_{M\to\infty}\mathbb{E}[B_M]$. Furthermore, it is

$$\begin{aligned}\mathbb{E}[B] &= \lim_{M\to\infty}\mathbb{E}[B_M] = \lim_{M\to\infty}\sum_{k=0}^{M}\mathbb{E}\left[\frac{\mathbb{1}[N \geq k]|\Delta_k|}{\mathbb{P}(N \geq k)}\right] \\ &= \lim_{M\to\infty}\sum_{k=0}^{M}\frac{\mathbb{P}(N \geq k)\mathbb{E}|\Delta_k|}{\mathbb{P}(N \geq k)} = \mathbb{E}\left[\lim_{M\to\infty}\sum_{k=0}^{M}|\Delta_k|\right] < \infty,\end{aligned}$$

where the assumption is used in the last step. Using the above, the dominated convergence theorem can be used to swap the limit and expectation for $Z_M$. Using similar derivations as above, it is

$$\mathbb{E}[Z] = \lim_{M\to\infty}\mathbb{E}[Z_M] = \lim_{M\to\infty}\mathbb{E}\left[\sum_{k=0}^{M}\Delta_k\right] = \lim_{M\to\infty}\mathbb{E}\left[Y_k\right] = a,$$

where we used the definition of $Y_M$ and $\lim_{k\to\infty}\mathbb{E}[Y_k] = a$. □

*Proof.* **(Theorem 1)**
To simplify notation, we denote $J := J_g(x)$. Furthermore, let

$$Y_N = \mathbb{E}_v\left[\sum_{k=1}^{N}\frac{(-1)^{k+1}}{k}v^T J^k v\right]$$

denote the real random variable and let $\Delta_0 = Y_0$ and $\Delta_k = Y_k - Y_{k-1}$ for $k \geq 1$, as in Lemma 3. To prove the claim of the theorem, we can use Lemma 3 and we only need to prove that the assumption $\mathbb{E}_v[\sum_{k=1}^{\infty}|\Delta_k|] < \infty$ holds for this specific case.

In order to exchange summation and expectation via Fubini's theorem, we need to proof that $\sum_{k=1}^{\infty}|\Delta_k| < \infty$ for all $v \sim \mathcal{N}(0, I)$. Using the assumption $\mathrm{Lip}(g) < 1$, it is

$$\begin{aligned}\sum_{k=1}^{\infty}|\Delta_k| &= \sum_{k=1}^{\infty}\left|\frac{(-1)^{k+1}}{k}v^T J^k v\right| = \sum_{k=1}^{\infty}\frac{\|v^T J^k v\|_2}{k} \leq \sum_{k=1}^{\infty}\frac{\|v^T\|_2\|J^k\|_2\|v\|_2}{k} \\ &\leq 2\|v\|_2\sum_{k=1}^{\infty}\frac{\|J\|_2^k}{k} \leq 2\|v\|_2\sum_{k=1}^{\infty}\frac{\mathrm{Lip}(g)_2^k}{k} = 2\|v\|_2\log\big(1-\mathrm{Lip}(g)\big) < \infty,\end{aligned}$$

for an arbitrary $v$. Hence,

$$\mathbb{E}_v\left[\sum_{k=1}^{\infty}|\Delta_k|\right] = \sum_{k=1}^{\infty}\mathbb{E}_v[|\Delta_k|]. \tag{13}$$

Since $\operatorname{tr}(A) = \mathbb{E}_v[v^T A v]$ for $v \sim \mathcal{N}(0, I)$ via the Skilling-Hutchinson trace estimator (Hutchinson, 1990; Skilling, 1989), it is

$$\mathbb{E}_v[|\Delta_k|] = \left| \frac{\operatorname{tr}(J^k)}{k} \right|.$$

To show that (13) is bounded, we derive the bound

$$\frac{1}{k} |\operatorname{tr}(J^k)| \leq \frac{1}{k} \left| \sum_{i=d}^{d} \lambda_i(J^k) \right| \leq \frac{1}{k} \sum_{i=d}^{d} |\lambda_i(J^k)| \leq \frac{d}{k} \rho(J^k) \leq \frac{d}{k} \|J^k\|_2 \leq \frac{d}{k} \operatorname{Lip}(g)^k,$$

where $\lambda(J^k)$ denote the eigenvalues and $\rho(J^k)$ the spectral radius. Inserting this bound into (13) and using $\operatorname{Lip}(g) < 1$ yields

$$\mathbb{E}_v[|\Delta_k|] \leq d \sum_{k=1}^{\infty} \frac{\operatorname{Lip}(g)^k}{k} = -d \log\left(1 - \operatorname{Lip}(g)\right) < \infty.$$

Hence, the assumption of Lemma 3 is verified. □

*Proof.* **(Theorem 2)**
The result can be proven in an analogous fashion to the proof of Theorem 1, which is why we only present a short version without all steps.

By obtaining the bound

$$\begin{aligned} \sum_{k=0}^{\infty} \left| (-1)^k v^T \left( J(x,\theta)^k \frac{\partial (J_g(x,\theta))}{\partial \theta} \right) v \right| &\leq 2\|v\|_2 \left\| \frac{\partial (J_g(x,\theta))}{\partial \theta} \right\| \sum_{k=0}^{\infty} \operatorname{Lip}(g)^k \\ &= 2\|v\|_2 \left\| \frac{\partial (J_g(x,\theta))}{\partial \theta} \right\| \frac{1}{1 - \operatorname{Lip}(g)} < \infty, \end{aligned}$$

Fubini's theorem can be applied to swap the expection and summation. Furthermore, by using the trace estimation and similar bounds as in the proof of Theorem 1, the assumption $\mathbb{E}\left[\sum_{k=0}^{\infty} |\Delta_k|\right] < \infty$ from Lemma 3 can be proven.

□

## C Memory-Efficient Gradient Estimation of Log-Determinant

Derivation of gradient estimator via differentiating power series:

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \log\det\left(I + J_g(x,\theta)\right) &= \frac{\partial}{\partial \theta_i} \left( \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\operatorname{tr}(J_g(x,\theta)^k)}{k} \right) \\ &= \operatorname{tr}\left( \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \frac{\partial (J_g(x,\theta)^k)}{\partial \theta_i} \right) \end{aligned}$$

Derivation of memory-efficient gradient estimator:

$$
\begin{aligned}
&\frac{\partial}{\partial \theta_i} \log \det \left(I + J_g(x,\theta)\right) \\
&= \frac{1}{\det(I + J_g(x,\theta))} \left[\frac{\partial}{\partial \theta_i} \det \left(I + J_g(x,\theta)\right)\right] && (14) \\
&= \frac{1}{\det(I + J_g(x,\theta))} \left[\det(I + J_g(x,\theta)) \operatorname{tr}\left((I + J(x,\theta))^{-1} \frac{\partial (I + J_g(x,\theta))}{\partial \theta_i}\right)\right] && (15) \\
&= \operatorname{tr}\left((I + J(x,\theta))^{-1} \frac{\partial (I + J_g(x,\theta))}{\partial \theta_i}\right) \\
&= \operatorname{tr}\left((I + J(x,\theta))^{-1} \frac{\partial (J_g(x,\theta))}{\partial \theta_i}\right) \\
&= \operatorname{tr}\left(\left[\sum_{k=0}^{\infty} (-1)^k J(x,\theta)^k\right] \frac{\partial (J_g(x,\theta))}{\partial \theta_i}\right) && (16)
\end{aligned}
$$

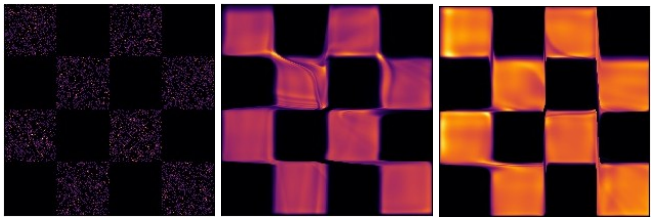
Note, that (14) follows from the chain rule of differentiation, for the derivative of the determinant in (15), see (Petersen and Pedersen, 2012) (eq. 46). Furthermore, (16) follows from the properties of a Neumann-Series which converges due to $\|J_g(x,\theta)\| < 1$.

Hence, if we are able to compute the trace exactly, both approaches will return the same values for a given truncation $n$. However, when estimating the trace via the Hutchinson trace estimator the estimation is not equal in general:

$$
v^T \left(\sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{k} \frac{\partial (J_g(x,\theta)^k)}{\partial \theta_i}\right) v \neq v^T \left(\left[\sum_{k=0}^{\infty} (-1)^k J_g^k(x,\theta)\right] \frac{\partial (J_g(x,\theta))}{\partial \theta_i}\right) v.
$$

Another difference between both approaches is their memory consumption of the corresponding computational graph. The summation $\sum_{k=0}^{\infty} (-1)^k J_g^k(x,\theta)$ is not being tracked for the gradient, which allows to compute the gradient with constant memory (constant with respect to the truncation $n$).

## D Generalized Spectral Norm



Data $p = 2$ (5.13 bits) $p = \infty$ (5.09 bits)

Figure 11: Learned densities on Checkerboard 2D.

**Using different induced $p$-norms on Checkerboard 2D.** We experimented with the checkerboard 2D dataset, which is a rather difficult two-dimensional data to fit a flow-based model on due to the discontinuous nature of the true distribution. We used brute-force computation of the log-determinant for change of variables (which, in the 2D case, is faster than the unbiased estimator). In the 2D case, we found that $\infty$-norm always outperforms or at least matches the $p = 2$ norm (ie. spectral norm). Figure 11 shows the learned densities with 200 residual blocks. The color represents the magnitude of $p_\theta(x)$, with brighter values indicating larger values. The $\infty$-norm model produces density estimates that are more evenly spread out across the space, whereas the spectral norm model focused its density to model between-density regions.

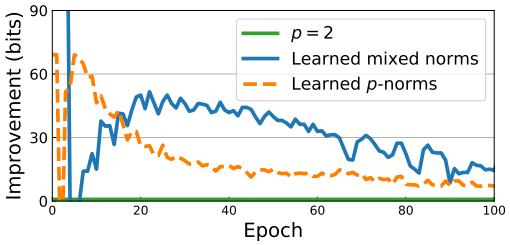



Figure 12: Improvement from using generalized spectral norm on CIFAR-10.

**Learning norm orders on CIFAR-10.** We used $1 + \tanh(s)/2$ where $s$ is a learned weight. This bounds the norm orders to $(1.5, 2.5)$. We tried two different setups. One where all norm orders are free to change (conditioned on them satisfying the constraints (11)), and another setting where each states within each residual block share the same order. Figure 12 shows the improvement in bits from using learned norms. The gain in performance is marginal, and the final models only outperformed spectral norm by around $0.003$ bits/dim. Interestingly, we found that the learned norms stayed around $p = 2$, shown in Figure 13, especially for the input and output spaces of $g$, ie. between blocks. This may suggest that spectral norm, or a norm with $p = 2$ is already optimal in this setting.

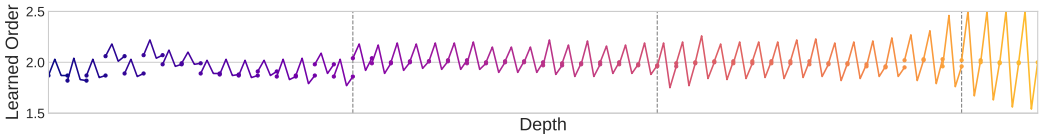



Figure 13: **Learned norm orders on CIFAR-10.** Each residual block is visualized as a single line. The input and two hidden states for each block use different normed spaces. We observe multiple trends: (i) the norms for the first hidden states are consistently higher than the input, and lower for the second. (ii) The orders for the hidden states drift farther away from 2 as depth increases. (iii) The ending order of one block and the starting order of the next are generally consistent and close to 2.

## E Experiment Setup

We use the standard setup of passing the data through a "unsquashing" layer (we used the logit transform (Dinh et al., 2017)), followed by alternating multiple blocks and squeeze layers (Dinh et al., 2017). We use activation normalization (Kingma and Dhariwal, 2018) before and after every residual block. Each residual connection consists of

$$\text{LipSwish} \to 3{\times}3 \text{ Conv} \to \text{LipSwish} \to 1{\times}1 \text{ Conv} \to \text{LipSwish} \to 3{\times}3 \text{ Conv}$$

with hidden dimensions of $512$. Below are the architectures for each dataset.

**MNIST.** With $\alpha =$1e-5.

$$\text{Image} \to \text{LogitTransform}(\alpha) \to 16{\times}\text{ResBlock} \to \left[\, \text{Squeeze} \to 16{\times}\text{ResBlock} \,\right]{\times}2$$

**CIFAR-10.** With $\alpha = 0.05$.

$$\text{Image} \to \text{LogitTransform}(\alpha) \to 16{\times}\text{ResBlock} \to \left[\, \text{Squeeze} \to 16{\times}\text{ResBlock} \,\right]{\times}2$$

**SVHN** With $\alpha = 0.05$.

$$\text{Image} \to \text{LogitTransform}(\alpha) \to 16{\times}\text{ResBlock} \to \left[\, \text{Squeeze} \to 16{\times}\text{ResBlock} \,\right]{\times}2$$

**ImageNet 32×32.** With $\alpha = 0.05$.

$$\text{Image} \to \text{LogitTransform}(\alpha) \to 32{\times}\text{ResBlock} \to \left[\, \text{Squeeze} \to 32{\times}\text{ResBlock} \,\right]{\times}2$$

**ImageNet 64×64.** With $\alpha = 0.05$.

$$\text{Image} \rightarrow \text{Squeeze} \rightarrow \text{LogitTransform}(\alpha) \rightarrow 32\times\text{ResBlock} \rightarrow \left[\ \text{Squeeze} \rightarrow 32\times\text{ResBlock}\ \right]\times 2$$

**CelebA 5bit 64×64.** With $\alpha = 0.05$.

$$\text{Image} \rightarrow \text{Squeeze} \rightarrow \text{LogitTransform}(\alpha) \rightarrow 16\times\text{ResBlock} \rightarrow \left[\ \text{Squeeze} \rightarrow 16\times\text{ResBlock}\ \right]\times 3$$

For density modeling on MNIST and CIFAR-10, we added 4 fully connected residual blocks at the end of the network, with intermediate hidden dimensions of 128. These residual blocks were not used in the hybrid modeling experiments or on other datasets.

For hybrid modeling on CIFAR-10, we replaced the logit transform with normalization by the standard preprocessing of subtracting the mean and dividing by the standard deviation across the training data. The MNIST and SVHN architectures for hybrid modeling were the same as those for density modeling.

For augmenting our flow-based model with a classifier in the hybrid modeling experiments, we added an additional branch after every squeeze layer and at the end of the network. Each branch consisted of

$$3\times 3 \text{ Conv} \rightarrow \text{ActNorm} \rightarrow \text{ReLU} \rightarrow \text{AdaptiveAveragePooling}((1, 1))$$

where the adaptive average pooling averages across all spatial dimensions and resulted in a vector of dimension $256$. The outputs at every scale were concatenated together and fed into a linear softmax classifier.

**Adaptive number of power iterations.** We used spectral normalization for convolutions (Gouk et al., 2018). To account for variable weight updates during training, we implemented an adaptive version of spectral normalization where we performed as many iterations as needed until the relative change in the estimated spectral norm was sufficiently small. As this also reduced the number of iterations when weight updates are small, this did not result in higher time complexity.

**Optimization.** For stochastic gradient descent, we used Adam (Kingma and Ba, 2015) with a learning rate of $0.001$ and weight decay of $0.0005$ applied outside the adaptive learning rate computation (Loshchilov and Hutter, 2019; Zhang et al., 2019). We used Polyak averaging (Polyak and Juditsky, 1992) for evaluation with a decay of $0.999$.

**Preprocessing.** For density estimation experiments, we used random horizontal flipping for CIFAR-10 and CelebA. For hybrid modeling and classification experiments, we used random cropping after reflection padding with 4 pixels for SVHN and CIFAR-10; CIFAR-10 also included random horizontal flipping.