

Invertible Residual Networks

Jens Behrmann^{*12} Will Grathwohl^{*2} Ricky T. Q. Chen² David Duvenaud² Jörn-Henrik Jacobsen^{*2}

Abstract

We show that standard ResNet architectures can be made invertible, allowing the same model to be used for classification, density estimation, and generation. Typically, enforcing invertibility requires partitioning dimensions or restricting network architectures. In contrast, our approach only requires adding a simple normalization step during training, already available in standard frameworks. Invertible ResNets define a generative model which can be trained by maximum likelihood on unlabeled data. To compute likelihoods, we introduce a tractable approximation to the Jacobian log-determinant of a residual block. Our empirical evaluation shows that invertible ResNets perform competitively with both state-of-the-art image classifiers and flow-based generative models, something that has not been previously achieved with a single architecture.

1. Introduction

One of the main appeals of neural network-based models is that a single model architecture can often be used to solve a variety of related tasks. However, many recent advances are based on special-purpose solutions tailored to particular domains. State-of-the-art architectures in unsupervised learning, for instance, are becoming increasingly domain-specific (Van Den Oord et al., 2016b; Kingma & Dhariwal, 2018; Parmar et al., 2018; Karras et al., 2018; Van Den Oord et al., 2016a). On the other hand, one of the most successful feed-forward architectures for discriminative learning are deep residual networks (He et al., 2016; Zagoruyko & Komodakis, 2016), which differ considerably from their generative counterparts. This divide makes it complicated to choose or design a suitable architecture for a given task. It also makes it hard for discriminative tasks to benefit from unsupervised learning. We bridge this gap with a new class of architectures that perform well in both domains.

^{*}Equal contribution ¹University of Bremen, Center for Industrial Mathematics ²Vector Institute and University of Toronto. Correspondence to: Jens Behrmann <jensb@uni-bremen.de>, Jörn-Henrik Jacobsen <j.jacobsen@vectorinstitute.ai>.

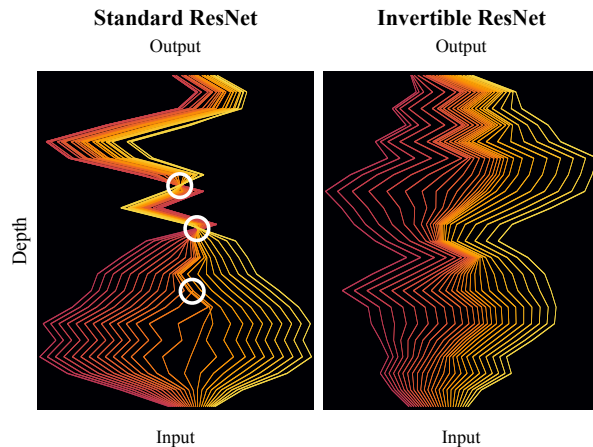


Figure 1. Dynamics of a standard residual network (left) and invertible residual network (right). Invertible ResNets describe a bijective continuous dynamics while regular ResNets result in crossing and collapsing paths (circled in white) which correspond to non-bijective continuous dynamics. Due to collapsing paths, standard ResNets are not a valid density model.

To achieve this, we focus on reversible networks which have been shown to produce competitive performance on discriminative (Gomez et al., 2017; Jacobsen et al., 2018) and generative (Dinh et al., 2014; 2017; Kingma & Dhariwal, 2018) tasks independently, albeit in the same model paradigm. They typically rely on fixed dimension splitting heuristics, but common splittings interleaved with non-volume conserving elements are constraining and their choice has a significant impact on performance (Kingma & Dhariwal, 2018; Dinh et al., 2017). This makes building reversible networks a difficult task. In this work we show that these exotic designs, necessary for competitive density estimation performance, can severely hurt discriminative performance.

To overcome this problem, we leverage the viewpoint of ResNets as an Euler discretization of ODEs (Haber & Ruthotto, 2018; Ruthotto & Haber, 2018; Lu et al., 2017; Ciccone et al., 2018) and prove that invertible ResNets (i-ResNets) can be constructed by simply changing the normalization scheme of standard ResNets. As an intuition, Figure 1 visualizes the differences in the dynamics learned by standard and invertible ResNets.

Work in progress. Copyright 2019 by the author(s).

This approach allows unconstrained architectures for each residual block, while only requiring a Lipschitz constant smaller than one for each block¹. We demonstrate that this restriction negligibly impacts performance when building image classifiers - they perform on par with their non-invertible counterparts on classifying MNIST, CIFAR10 and CIFAR100 images.

We then show how i-ResNets can be trained as maximum likelihood generative models on unlabeled data. To compute likelihoods, we introduce a tractable approximation to the Jacobian determinant of a residual block. Like FFIORD (Grathwohl et al., 2019), i-ResNet flows have unconstrained (free-form) Jacobians, allowing them to learn more expressive transformations than the triangular mappings used in other reversible models. Our empirical evaluation shows that i-ResNets perform competitively with both state-of-the-art image classifiers and flow-based generative models, bringing general-purpose architectures one step closer to reality.

2. Enforcing Invertibility in ResNets

There is a remarkable similarity between ResNet architectures and Euler’s method for ODE initial value problems:

$$\begin{aligned} x_{t+1} &\leftarrow x_t + g_{\theta_t}(x_t) \\ x_{t+1} &\leftarrow x_t + h f_{\theta_t}(x_t) \end{aligned}$$

where $x_t \in \mathbb{R}^d$ represent activations or states, t represents layer indices or time, $h > 0$ is a step size, and g_{θ_t} is a residual block. This connection has attracted research at the intersection of deep learning and dynamical systems (Lu et al., 2017; Haber & Ruthotto, 2018; Ruthotto & Haber, 2018; Chen et al., 2018). However, little attention has been paid to the dynamics backwards in time

$$\begin{aligned} x_t &\leftarrow x_{t+1} - g_{\theta_t}(x_t) \\ x_t &\leftarrow x_{t+1} - h f_{\theta_t}(x_t) \end{aligned}$$

which amounts to the implicit backward Euler discretization. In particular, solving the dynamics backwards in time would implement an inverse of the corresponding ResNet. The following theorem states that a simple condition suffices to make the dynamics solvable and thus renders the ResNet invertible:

Theorem 1 (Sufficient condition for invertible ResNets). *Let $F_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ with $F_\theta = (F_\theta^1 \circ \dots \circ F_\theta^T)$ denote a ResNet with blocks $F_\theta^t = I + g_{\theta_t}$. Then, the ResNet F_θ is invertible if*

$$\text{Lip}(g_{\theta_t}) < 1, \text{ for all } t = 1, \dots, T,$$

where $\text{Lip}(g_{\theta_t})$ is the Lipschitz-constant of g_{θ_t} .

¹Note that this still allows for arbitrarily high Lipschitz constants of the full network.

Algorithm 1. Inverse of i-ResNet layer via fixed-point iteration.

Input: activation y , contractive residual block g , number of fixed-point iterations n
 Init: $x^0 := y$
for $i = 0, \dots, n$ **do**
 $x^{i+1} := y - g(x^i)$
end for

Note that this condition is not necessary for invertibility. Other approaches (Dinh et al., 2014; 2017; Jacobsen et al., 2018; Chang et al., 2018; Kingma & Dhariwal, 2018) rely on partitioning dimensions or autoregressive structures to create analytical inverses.

While enforcing $\text{Lip}(g) < 1$ makes the ResNet invertible, we have no analytic form of this inverse. However, we can obtain it through a simple fixed-point iteration, see Algorithm 1. Note, that the starting value for the fixed-point iteration can be any vector, because the fixed-point is unique. However, using $x^0 = y$ for initializing is a good starting point since y was obtained from x only via a bounded perturbation of the identity. From the Banach fixed-point theorem we have

$$\|x - x^n\|_2 \leq \frac{\text{Lip}(g)^n}{1 - \text{Lip}(g)} \|x^1 - x^0\|_2. \quad (1)$$

Thus, the convergence rate is exponential in the number of iterations n and smaller Lipschitz constants will yield faster convergence.

Additional to invertibility, a contractive residual block also renders the residual layer bi-Lipschitz.

Lemma 2 (Lipschitz constants of Forward and Inverse). *Let $F(x) = x + g(x)$ with $\text{Lip}(g) = L < 1$ denote the residual layer. Then, it holds*

$$\text{Lip}(F) \leq 1 + L \quad \text{and} \quad \text{Lip}(F^{-1}) \leq \frac{1}{1 - L}.$$

Hence by design, invertible ResNets offer stability guarantees for both their forward and inverse mapping. In the following section, we discuss approaches to enforce the Lipschitz condition.

2.1. Satisfying the Lipschitz Constraint

We implement residual blocks as a composition of contractive nonlinearities ϕ (e.g. ReLU, ELU, tanh) and linear mappings. For example, in our convolutional networks $g = W_3\phi(W_2\phi(W_1))$, where W_i are convolutional layers. Hence,

$$\text{Lip}(g) < 1, \quad \text{if } \|W_i\|_2 < 1,$$

where $\|\cdot\|_2$ denotes the spectral norm. Note, that regularizing the spectral norm of the Jacobian of g (Sokoli et al., 2017) only reduces it locally and does not guarantee the above condition. Thus, we will enforce $\|W_i\|_2 < 1$ for each layer.

Unlike (Miyato et al., 2018), we directly estimate the spectral norm of W_i by performing power-iteration using W_i and W_i^T as proposed in Gouk et al. (2018). Note, that a power-iteration on the parameter matrix (Miyato et al., 2018) only gives a bound on $\|W_i\|_2$ when the filter kernel is larger than 1×1 , see (Tsuzuku et al., 2018). The power-iteration yields an under-estimate $\tilde{\sigma}_i \leq \|W_i\|_2$. Using this estimate, we normalize via

$$\tilde{W}_i = \begin{cases} c W_i / \tilde{\sigma}_i, & \text{if } c / \tilde{\sigma}_i < 1 \\ W_i, & \text{else} \end{cases}, \quad (2)$$

where the hyper-parameter $c < 1$ is a scaling coefficient. Since $\tilde{\sigma}_i$ is an under-estimate, $\|W_i\|_2 \leq c$ is not guaranteed. However, after training (Sedghi et al., 2019) offer an approach to inspect $\|W_i\|_2$ exactly using the SVD on the Fourier transformed parameter matrix, which will allow us to show $\text{Lip}(g) < 1$ holds in all cases.

3. Generative Modelling with i-ResNets

We can define a simple generative model for data $x \in \mathbb{R}^d$ by first sampling $z \sim p_z(z)$ where $z \in \mathbb{R}^d$ and then defining $x = \Phi(z)$ for some function $\Phi : \mathbb{R}^d \rightarrow \mathbb{R}^d$. If Φ is invertible and we define $F = \Phi^{-1}$, then we can compute the likelihood of any x under this model using the change of variables formula

$$\ln p_x(x) = \ln p_z(z) + \ln |\det J_F(x)|, \quad (3)$$

where $J_F(x)$ is the Jacobian of F evaluated at x . Models of this form are known as Normalizing Flows (Rezende & Mohamed, 2015). They have recently become a popular model for high-dimensional data due to the introduction of powerful bijective function approximators whose Jacobian log-determinant can be efficiently computed (Dinh et al., 2014; 2017; Kingma & Dhariwal, 2018; Chen et al., 2018) or approximated (Grathwohl et al., 2019).

Since i-ResNets are guaranteed to be invertible we can use them to parameterize F in Equation (3). Samples from this model can be drawn by first sampling $z \sim p(z)$ and then computing $x = F^{-1}(z)$ with Algorithm 1. In Figure 2 we show an example of using an i-ResNet to define a generative model on some two-dimensional datasets compared to Glow (Kingma & Dhariwal, 2018).

3.1. Scaling to Higher Dimensions

While the invertibility of i-ResNets allows us to use them to define a Normalizing Flow, we must compute

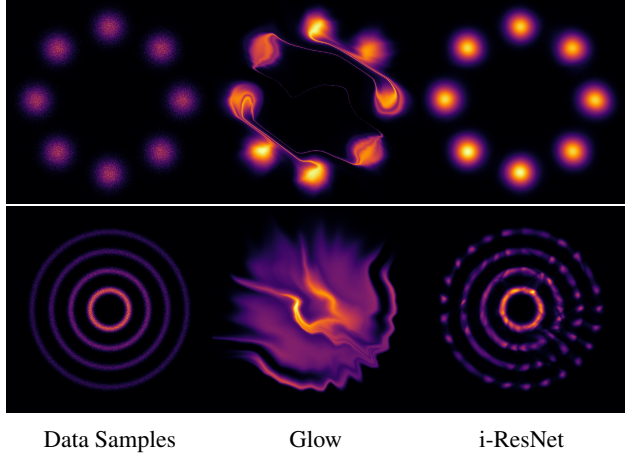


Figure 2. Visual comparison of i-ResNet flow and Glow. Details of this experiment can be found in Appendix C.2.

$\ln |\det J_F(x)|$ to evaluate the data-density under the model. Computing this quantity has a time cost of $\mathcal{O}(d^3)$ in general which makes naïvely scaling to high-dimensional data impossible.

To bypass this constraint we present a tractable approximation to the log-determinant term in Equation (3), which will scale to high dimensions d . First, we note that the Lipschitz constrained perturbations $x + g(x)$ of the identity yield positive determinants, hence

$$|\det J_F(x)| = \det J_F(x),$$

see Lemma 6 in Appendix A. Combining this result with the matrix identity $\ln \det(A) = \text{tr}(\ln(A))$ for non-singular $A \in \mathbb{R}^{d \times d}$ (see e.g. Withers & Nadarajah (2010)), we have

$$\ln |\det J_F(x)| = \text{tr}(\ln J_F),$$

where tr denotes the matrix trace and \ln the matrix logarithm. Thus for $z = F(x) = (I + g)(x)$, it is

$$\ln p_x(x) = \ln p_z(z) + \text{tr}(\ln(I + J_g(x))).$$

The trace of the matrix logarithm can be expressed as a power series (Hall, 2015)

$$\text{tr}(\ln(I + J_g(x))) = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{\text{tr}(J_g^k)}{k}, \quad (4)$$

which converges if $\|J_g\|_2 < 1$. Hence, due to the Lipschitz constraint, we can compute the log-determinant via the above power series with guaranteed convergence.

Before we present a stochastic approximation to the above power series, we observe following properties of i-ResNets: Due to $\text{Lip}(g_t) < 1$ for the residual block of each layer t , we

can provide a lower and upper bound on its log-determinant with

$$\begin{aligned} d \sum_{t=1}^T \ln(1 - \text{Lip}(g_t)) &\leq \ln |\det J_F(x)| \\ d \sum_{t=1}^T \ln(1 + \text{Lip}(g_t)) &\geq \ln |\det J_F(x)|, \end{aligned}$$

for all $x \in \mathbb{R}$, see Lemma 7 in Appendix A. Thus, both the number of layers T and the Lipschitz constant affect the contraction and expansion bounds of i-ResNets and must be taken into account when designing such an architecture.

3.2. Stochastic Approximation of log-determinant

Expressing the log-determinant with the power series in (4) has three main computational drawbacks: 1) Computing $\text{tr}(J_g)$ exactly costs $\mathcal{O}(d^2)$, or approximately needs d evaluations of g as each entry of the diagonal of the Jacobian requires the computation of a separate derivative of g (Grathwohl et al., 2019). 2) Matrix powers J_g^k are needed, which requires the knowledge of the full Jacobian. 3) The series is infinite.

Fortunately, drawback 1) and 2) can be alleviated. First, vector-Jacobian products $v^T J_g$ can be computed at approximately the same costs as evaluating g through reverse-mode automatic differentiation. Second, a stochastic approximation of the matrix trace of $A \in \mathbb{R}^{d \times d}$

$$\text{tr}(A) = \mathbb{E}_{p(v)} [v^T A v],$$

known as the Hutchinsons trace estimator, can be used to estimate $\text{tr}(J_g^k)$. The distribution $p(v)$ needs to fulfill $\mathbb{E}[v] = 0$ and $\text{Cov}(v) = I$, see (Hutchinson, 1990; Avron & Toledo, 2011).

While this allows for an unbiased estimate of the matrix trace, to achieve bounded computational costs, the power series (4) will be truncated at index n to address drawback 3). Algorithm 2 summarizes the basic steps. The truncation turns the unbiased estimator into a biased estimator, where the bias depends on the truncation error. Fortunately, this error can be bounded as we demonstrate below.

To improve the stability of optimization when using this estimator we recommend using nonlinearities with continuous derivatives such as ELU (Clevert et al., 2015) or softplus instead of ReLU (See Appendix C.2).

3.3. Error of Power Series Truncation

We estimate $\ln |\det(I + J_g)|$ with the finite power series

$$PS(J_g, n) := \sum_{k=1}^n (-1)^{k+1} \frac{\text{tr}(J_g^k)}{k}, \quad (5)$$

Algorithm 2. Forward pass of an invertible ResNets with Lipschitz constraint and log-determinant approximation, SN denotes spectral normalization based on (2).

Input: data point x , network F , residual block g , number of power series terms n
for Each residual block **do**
 Lip constraint: $\hat{W}_j := \text{SN}(W_j, x)$ for linear Layer W_j .
 Draw v from $\mathcal{N}(0, I)$
 $w^T := v^T$
 $\ln \det := 0$
 for $k = 1$ **to** n **do**
 $w^T := w^T J_g$ (vector-Jacobian product)
 $\ln \det := \ln \det + (-1)^{k+1} w^T v / k$
 end for
end for

where we have (with some abuse of notation) $PS(J_g, \infty) = \text{tr}(\ln(I + J_g))$. We are interested in bounding the truncation error of the log-determinant as a function of the data dimension d , the Lipschitz constant $\text{Lip}(g)$ and the number of terms in the series n .

Theorem 3 (Approximation error of Loss). *Let g denote the residual function and J_g the Jacobian as before. Then, the error of a truncated power series at term n is bounded as*

$$\begin{aligned} &|PS(J_g, n) - \ln \det(I + J_g)| \\ &\leq -d \left(\ln(1 - \text{Lip}(g)) + \sum_{k=1}^n \frac{\text{Lip}(g)^k}{k} \right). \end{aligned}$$

While the result above gives an error bound for evaluation of the loss, during training the error in the gradient of the loss is of greater interest. Similarly, we can obtain the following bound. The proofs are given in Appendix A.

Theorem 4 (Convergence Rate of Gradient Approximation). *Let $\theta \in \mathbb{R}^p$ denote the parameters of network F , let g, J_g be as before. Further, assume bounded inputs and a Lipschitz activation function with Lipschitz derivative. Then, we obtain the convergence rate*

$$\|\nabla_{\theta} (\ln \det(I + J_g)) - PS(J_g, n)\|_{\infty} = \mathcal{O}(c^n)$$

where $c := \text{Lip}(g)$ and n the number of terms used in the power series.

In practice, only 5-10 terms must be taken to obtain a bias less than .001 bits per dimension, which is typically reported up to .01 precision (See Appendix E).

Invertible Residual Networks

Method	ResNet	NICE/ i-RevNet	Real-NVP	Glow	FFJORD	i-ResNet
Free-form	✓	✗	✗	✗	✓	✓
Analytic Forward	✓	✓	✓	✓	✗	✓
Analytic Inverse	N/A	✓	✓	✗	✗	✗
Non-volume Preserving	N/A	✗	✓	✓	✓	✓
Exact Likelihood	N/A	✓	✓	✓	✗	✗
Unbiased Stochastic Log-Det Estimator	N/A	N/A	N/A	N/A	✓	✗

Table 1. Comparing i-ResNet and ResNets to NICE (Dinh et al., 2014), Real-NVP (Dinh et al., 2017), Glow (Kingma & Dhariwal, 2018) and FFJORD (Grathwohl et al., 2019). Non-volume preserving refers to the ability to allow for contraction and expansions and exact likelihood to compute the change of variables (3) exactly. The unbiased estimator refers to a stochastic approximation of the log-determinant, see section 3.2.

4. Related Work

4.1. Reversible Architectures

We put our focus on invertible architectures with efficient inverse computation, namely NICE (Dinh et al., 2014), i-RevNet (Jacobsen et al., 2018), Real-NVP (Dinh et al., 2017), Glow (Kingma & Dhariwal, 2018) and Neural ODEs (Chen et al., 2018) and its stochastic density estimator FFJORD (Grathwohl et al., 2019). A summary of the comparison between different reversible networks is given in Table 1.

The dimension-splitting approach used in NICE, i-RevNet, Real-NVP and Glow allows for both analytic forward and inverse mappings. However, this restriction required the introduction of additional steps like invertible 1×1 convolutions in Glow (Kingma & Dhariwal, 2018). These 1×1 convolutions need to be inverted numerically, making Glow altogether not analytically invertible. In contrast, i-ResNet can be viewed as an intermediate approach, where the forward mapping is given analytically, while the inverse can be computed via a fixed-point iteration.

Furthermore, an i-ResNet block has a Lipschitz bound both for forward and inverse (Lemma 2), while other approaches do not have this property by design. Hence, i-ResNets could be an interesting avenue for stability-critical applications like inverse problems (Ardizzone et al., 2019) or invariance-based adversarial vulnerability (Jacobsen et al., 2019).

Neural ODEs (Chen et al., 2018) allow free-form dynamics similar to i-ResNets, meaning that any architecture could be used as long as the input and output dimensions are the same. To obtain discrete forward and inverse dynamics, Neural ODEs rely on adaptive ODE solvers, which allows for an accuracy vs. speed trade-off. Yet, scalability to very high input dimension such as high-resolution images remains unclear.

4.2. Ordinary Differential Equations

Due to the similarity of ResNets and Euler discretizations, there are many connections between the i-ResNet and ODEs, which we review in this section.

Relationship of i-ResNets to Neural ODEs: The view of deep networks as dynamics over time offers two fundamental learning approaches: 1) Direct learning of dynamics using discrete architectures like ResNets (Haber & Ruthotto, 2018; Ruthotto & Haber, 2018; Lu et al., 2017; Ciccone et al., 2018). 2) Indirect learning of dynamics via parametrizing an ODE with a neural network as in Chen et al. (2018); Grathwohl et al. (2019).

The dynamics $x(t)$ of a fixed ResNet F_θ are only defined at time points t_i corresponding to each block $g_{\theta_{t_i}}$. However, a linear interpolation in time can be used to generate continuous dynamics. See Figure 1, where the continuous dynamics of a linearly interpolated invertible ResNet are shown against those of a standard ResNet. Invertible ResNets are bijective along the continuous path while regular ResNets may result in crossing or merging paths. The indirect approach of learning an ODE, on the other hand, adapts the discretization based on an ODE-solver, but does not have a fixed computational budget compared to an i-ResNet.

Stability of ODEs: There are two main approaches to study the stability of ODEs, 1) behavior for $t \rightarrow \infty$ and 2) Lipschitz stability over finite time intervals $[0, T]$. Based on time-invariant dynamics $f(x(t))$, (Ciccone et al., 2018) constructed asymptotically stable ResNets using anti-symmetric layers such that $\text{Re}(\lambda(J_x)) < 0$ (with $\text{Re}(\lambda(\cdot))$ denoting the real-part of eigenvalues, $\rho(\cdot)$ spectral radius and $J_x g$ the Jacobian at point x). By projecting weights based on the Gershgorin circle theorem, they further fulfilled $\rho(J_x g) < 1$, yielding asymptotically stable ResNets with shared weights over layers. On the other hand, (Haber & Ruthotto, 2018; Ruthotto & Haber, 2018) considered time-dependent dynamics $f(x(t), \theta(t))$ corresponding to standard ResNets. They induce stability by using anti-symmetric layers and

projections of the weights. Contrarily, initial value problems on $[0, T]$ are well-posed for Lipschitz continuous dynamics (Ascher, 2008). Thus, the invertible ResNet with $\text{Lip}(f) < 1$ can be understood as a stabilizer of an ODE for step size $h = 1$ without a restriction to anti-symmetric layers as in Ruthotto & Haber (2018); Haber & Ruthotto (2018); Ciccone et al. (2018).

4.3. Spectral Sum Approximations

The approximation of spectral sums like the log-determinant is of broad interest for many machine learning problems such as Gaussian Process regression (Dong et al., 2017). Among others, Taylor approximation (Boutsidis et al., 2017) of the log-determinant similar to our approach or Chebyshev polynomials (Han et al., 2016) are used. In Boutsidis et al. (2017), error bounds on the estimation via truncated power series and stochastic trace estimation are given for symmetric positive definite matrices. However, $I + J_g$ is not symmetric and thus, their analysis does not apply here.

Recently, unbiased estimates (Adams et al., 2018) and unbiased gradient estimators (Han et al., 2018) were proposed for symmetric positive definite matrices. Furthermore, Chebyshev polynomials have been used to approximate the log-determinant of Jacobian of deep neural networks in Ramesh & LeCun (2018) to evaluate the likelihood of GANs.

5. Experiments

We complete a thorough experimental survey of invertible ResNets. First, we numerically verify the invertibility of i-ResNets. Then, we investigate their discriminative abilities on a number of common image classification datasets. Furthermore, we compare the discriminative performance of i-ResNets to other invertible networks. Finally, we study how i-ResNets can be used to define generative models.

5.1. Validating Invertibility and Classification

To compare the discriminative performance and invertibility of i-ResNets with standard ResNet architectures, we train both models on CIFAR10, CIFAR100, and MNIST. The CIFAR and MNIST models have models have 54 and 21 residual blocks, respectively and we use identical settings for all other hyperparameters. We replace strided downsampling with “invertible downsampling” operations (Jacobsen et al., 2018) to ensure bijectivity, see Appendix C.1 for training and architectural details. We increase the number of input channels to 16 by padding with zeros. This is analogous to the standard practice of projecting the data into a higher-dimensional space using a standard convolutional layer at the input of a model, but this mapping is reversible. To obtain the numerical inverse, we apply 100 fixed point iterations (Equation (1)) for each block. This



Figure 3. Original images (top) and reconstructions from i-ResNet with $c = 0.9$ (middle) and a standard ResNet with the same architecture (bottom), showing that the fixed point iteration does not recover the input without the Lipschitz constraint.

number is chosen to ensure that the poor reconstructions for vanilla ResNets (see Figure 3) are not due to using too few iterations. In practice far fewer iterations suffice, as the trade-off between reconstruction error and number of iterations analyzed in Appendix D shows.

Classification and reconstruction results for a baseline pre-activation ResNet-164, a ResNet with architecture like i-ResNets without Lipschitz constraint (denoted as vanilla) and five invertible ResNets with different spectral normalization coefficients are shown in Table 2. The results illustrate that for larger settings of the layer-wise Lipschitz constant c , our proposed invertible ResNets perform competitively with the baselines in terms of classification performance, while being provably invertible. When applying very conservative normalization (small c), the classification error becomes higher on all datasets tested.

To demonstrate that our normalization scheme is effective and that standard ResNets are not generally invertible, we reconstruct inputs from the features of each model using Algorithm 1. Intriguingly, our analysis also reveals that unconstrained ResNets are invertible after training on MNIST (see Figure 7 in Appendix B), whereas on CIFAR10/100 they are not. Further, we find ResNets with and without BatchNorm are not invertible after training on CIFAR10, which can also be seen from the singular value plots in Appendix B (Figure 6). The runtime on 4 GeForce GTX 1080 GPUs with 1 spectral norm iteration was 0.5 sec for a forward and backward pass of batch with 128 samples, while it took 0.2 sec without spectral normalization.

The reconstruction error decays quickly and the errors are already imperceptible after 5-20 iterations, which is the cost of 5-20 times the forward pass and empirically corresponds to 0.15-0.75 seconds for reconstructing 100 CIFAR10 images. Computing the inverse is fast even for the largest normalization coefficient, but becomes faster with stronger normalization. The number of iterations needed for full convergence is approximately cut in half when reducing the spectral normalization coefficient by 0.2, see Figure 8 (Appendix D) for a detailed plot. We also ran an i-RevNet

Invertible Residual Networks

		ResNet-164	Vanilla	$c = 0.9$	$c = 0.8$	$c = 0.7$	$c = 0.6$	$c = 0.5$
Classification	MNIST	-	0.38	0.40	0.42	0.40	0.42	0.86
	Error %		6.69	6.78	6.86	6.93	7.72	8.71
	CIFAR100	24.30	23.97	24.58	24.99	25.99	27.30	29.45
Guaranteed Inverse		No	No	Yes	Yes	Yes	Yes	Yes

Table 2. Comparison of i-ResNet to a ResNet-164 baseline architecture of similar depth and width with varying Lipschitz constraints via coefficients c . Vanilla shares the same architecture as i-ResNet, without the Lipschitz constraint.

(Jacobsen et al., 2018) with comparable hyperparameters as ResNet-164 and it performs on par with ResNet-164 with 5.6%. Note however, that i-RevNets are similar to volume-conserving NICE (Dinh et al., 2014) which performs significantly worse than its non-volume conserving counterparts Real-NVP and Glow.

In summary, we observe that invertibility without additional constraints is unlikely, but possible, whereas it is hard to predict if networks will have this property. In our proposed model, we can guarantee the existence of an inverse without significantly harming classification performance.

5.2. Comparison with Other Invertible Architectures

In this section we compare i-ResNet classifiers to the state-of-the-art invertible flow-based model Glow. We take the implementation of Kingma & Dhariwal (2018) and modify it to classify CIFAR10 images (with no generative modeling component). We create an i-ResNet that is as close as possible in structure to the default Glow model on CIFAR10 (denoted as i-ResNet Glow-style) and compare it to two variants of Glow, one that uses learned (1×1 convolutions) and affine block structure, and one with reverse permutations (like Real-NVP) and additive block structure. Results of this experiment can be found in Table 3. We can see that i-ResNets outperform all versions of Glow on this discriminative task, even when adapting the network depth and width to that of Glow. This indicates that i-ResNets have a more suitable inductive bias in their block structure for discriminative tasks than Glow.

We also find that i-ResNets are considerably easier to train

Affine Glow 1×1 Conv	Additive Glow Reverse	i-ResNet Glow-Style	i-ResNet 164
12.63	12.36	8.03	6.69

Table 3. CIFAR10 classification results compared to state-of-the-art flow Glow as a classifier. We compare two versions of Glow, as well as an i-ResNet architecture as similar as possible to Glow in its number of layers and channels, termed “i-ResNet, Glow-Style”.

than these other models. We are able to train i-ResNets using SGD with momentum and a learning rate of 0.1 whereas all version of Glow we tested needed Adam or Adamax (Kingma & Ba, 2014) and much smaller learning rates to avoid divergence.

5.3. Generative Modeling

We run a number of experiments to verify the utility of i-ResNets in building generative models. First, we compare i-ResNet Flows with Glow (Kingma & Dhariwal, 2018) on simple two-dimensional datasets. Figure 2 qualitatively shows the density learned by a Glow model with 100 coupling layers and 100 invertible linear transformations. We compare against an i-ResNet where the coupling layers are replaced by invertible residual blocks with the same number of parameters and the invertible linear transformations are replaced by actnorm (Kingma & Dhariwal, 2018). This results in the i-ResNet model having slightly fewer parameters, while maintaining an equal number of layers. In this experiment we train i-ResNets using the brute-force computed log-determinant since the data is two-dimensional. We find that i-ResNets are able to more accurately fit these simple densities. As stated in Grathwohl et al. (2019), we believe this is due to our model’s ability to avoid partitioning dimensions.

Next we evaluate i-ResNets as a generative model for images on MNIST and CIFAR10. Our models consist of multiple i-ResNet blocks followed by invertible downsampling or di-

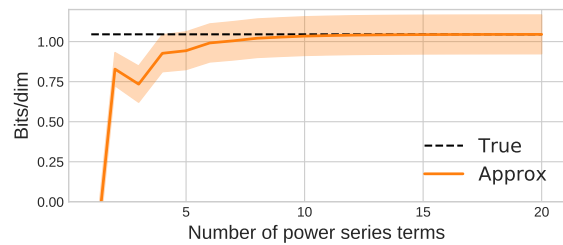


Figure 4. Bias and standard deviation of our log-determinant estimator as the number of power series terms increases. Variance is due to the stochastic trace estimator.



Figure 5. CIFAR10 samples from our i-ResNet flow. More samples can be found in Appendix F.

Method	MNIST	CIFAR10
NICE (Dinh et al., 2014)	4.36	4.48†
MADE (Germain et al., 2015)	2.04	5.67
MAF (Papamakarios et al., 2017)	1.89	4.31
Real NVP (Dinh et al., 2017)	1.06	3.49
Glow (Kingma & Dhariwal, 2018)	1.05	3.35
FFJORD (Grathwohl et al., 2019)	0.99	3.40
i-ResNet	1.06	3.45

Table 4. MNIST and CIFAR10 bits/dim results. † Uses ZCA pre-processing making results not directly comparable.

mension “squeezing” to downsample the spatial dimensions. We use multi-scale architectures like those of Dinh et al. (2017); Kingma & Dhariwal (2018). In these experiments we train i-ResNets using the log-determinant approximation, see Algorithm 2. Full architecture, experimental, and evaluation details can be found in Appendix C.2. Samples from our CIFAR10 model are shown in Figure 5 and samples from our MNIST model can be found in Appendix F. Compared to the classification model, the log-determinant approximation with 5 series terms roughly increased the computation times by a factor of 4. The bias and variance of our log-determinant estimator is shown in Figure 4.

Results and comparisons to other generative models can be found in Table 4. While our models did not perform as well as Glow and FFJORD, we find it intriguing that ResNets, with very little modification, can create a generative model competitive with these highly engineered models. We believe the gap in performance is mainly due to our use of a biased log-determinant estimator and that the use of an unbiased method (Han et al., 2018) can help close this gap.

6. Other Applications

In many applications, a secondary unsupervised learning or generative modeling objective is formulated in combination with a primary discriminative task. i-ResNets are appealing here, as they manage to achieve competitive performance on both discriminative and generative tasks. We summarize some application areas to highlight that there is a wide

variety of tasks for which i-ResNets would be promising to consider:

- Hybrid density and discriminative models for joint classification and detection or fairness applications (Nalisnick et al., 2018; Louizos et al., 2016)
- Unsupervised learning for downstream tasks (Hjelm et al., 2019; Van Den Oord et al., 2018)
- Semi-supervised learning from few labeled examples (Oliver et al., 2018; Kingma et al., 2014)
- Solving inverse problems with hybrid regression and generative losses (Ardizzone et al., 2019)
- Adversarial robustness with likelihood-based generative models (Schott et al., 2019; Jacobsen et al., 2019)

Finally, it is plausible that the Lipschitz bounds on the layers of the i-ResNet could aid with the stability of gradients for optimization, as well as adversarial robustness.

7. Conclusions

We introduced a new architecture, i-ResNets, which allow free-form layer architectures while still providing tractable density estimates. The unrestricted form of the Jacobian allows expansion and contraction via the residual blocks, while partitioning-based models (Dinh et al., 2014; 2017; Kingma & Dhariwal, 2018) must include affine blocks and scaling layers to be non-volume preserving.

Several challenges remain to be addressed in future work. First, our estimator of the log-determinant is biased. However, there have been recent advances in building unbiased estimators for the log-determinant (Han et al., 2018), which we believe could improve the performance of our generative model. Second, learning and designing networks with a Lipschitz constraint is challenging. For example, we need to constrain each linear layer in the block instead of being able to directly control the Lipschitz constant of a block, see Anil et al. (2018) for a promising approach for addressing this problem.

Acknowledgments

We thank Rich Zemel for very helpful comments on an earlier version of the manuscript. We thank Yulia Rubanova for spotting a mistake in one of the proofs. We also thank everyone else at Vector for helpful discussions and feedback.

We gratefully acknowledge the financial support from the German Science Foundation for RTG 2224 ” π^3 : Parameter Identification - Analysis, Algorithms, Applications”

References

- Adams, R. P., Pennington, J., Johnson, M. J., Smith, J., Ovadia, Y., Patton, B., and Saunderson, J. Estimating the spectral density of large implicit matrices. *arXiv preprint arXiv:1802.03451*, 2018.
- Anil, C., Lucas, J., and Grosse, R. Sorting out lipschitz function approximation. *arXiv preprint arXiv:1811.05381*, 2018.
- Ardizzone, L., Kruse, J., Rother, C., and Köthe, U. Analyzing inverse problems with invertible neural networks. In *International Conference on Learning Representations*, 2019.
- Ascher, U. *Numerical methods for evolutionary differential equations*. Computational science and engineering. Society for Industrial and Applied Mathematics, 2008.
- Avron, H. and Toledo, S. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *J. ACM*, 58(2):8:1–8:34, 2011.
- Boutsidis, C., Drineas, P., Kambadur, P., Kontopoulou, E.-M., and Zouzias, A. A randomized algorithm for approximating the log determinant of a symmetric positive definite matrix. *Linear Algebra and its Applications*, 533: 95 – 117, 2017.
- Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. Reversible architectures for arbitrarily deep residual neural networks. *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- Chen, T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. Neural ordinary differential equations. *Advances in Neural Information Processing Systems*, 2018.
- Ciccone, M., Gallieri, M., Masci, J., Osendorfer, C., and Gomez, F. Nais-net: Stable deep networks from non-autonomous differential equations. In *Advances in Neural Information Processing Systems 31*, pp. 3029–3039. 2018.
- Clevert, D., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.
- Dinh, L., Krueger, D., and Bengio, Y. Nice: Non-linear independent components estimation. *arXiv preprint arXiv:1410.8516*, 2014.
- Dinh, L., Sohl-Dickstein, J., and Bengio, S. Density estimation using real nvp. *International Conference on Learning Representations*, 2017.
- Dong, K., Eriksson, D., Nickisch, H., Bindel, D., and Wilson, A. G. Scalable log determinants for gaussian process kernel learning. In *Advances in Neural Information Processing Systems 30*, pp. 6327–6337. 2017.
- Drucker, H. and Lecun, Y. Improving generalization performance using double backpropagation. *IEEE transactions on neural networks*, 3:991–7, 02 1992.
- Germain, M., Gregor, K., Murray, I., and Larochelle, H. MADE: Masked autoencoder for distribution estimation. In *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 881–889. JMLR.org, 2015.
- Gomez, A. N., Ren, M., Urtasun, R., and Grosse, R. B. The reversible residual network: Backpropagation without storing activations. *Advances in Neural Information Processing Systems*, 2017.
- Gouk, H., Frank, E., Pfahringer, B., and Cree, M. Regularisation of neural networks by enforcing lipschitz continuity. *arXiv preprint arXiv:1804.04368*, 2018.
- Grathwohl, W., Chen, R. T. Q., Bettencourt, J., and Duvenaud, D. Ffjord: Scalable reversible generative models with free-form continuous dynamics. In *International Conference on Learning Representations*, 2019.
- Haber, E. and Ruthotto, L. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2018.
- Hall, B. C. Lie groups, lie algebras, and representations: An elementary introduction. *Graduate Texts in Mathematics*, 222 (2nd ed.), Springer, 2015.
- Han, I., Malioutov, D., Avron, H., and Shin, J. Approximating the spectral sums of large-scale matrices using chebyshev approximations. *SIAM Journal on Scientific Computing*, 39, 06 2016.
- Han, I., Avron, H., and Shin, J. Stochastic chebyshev gradient descent for spectral optimization. In *Advances in Neural Information Processing Systems 31*, pp. 7397–7407. 2018.
- He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.
- Hjelm, R. D., Fedorov, A., Lavoie-Marchildon, S., Grewal, K., Bachman, P., Trischler, A., and Bengio, Y. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.
- Hutchinson, M. 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.

- Jacobsen, J.-H., Smeulders, A. W., and Oyallon, E. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*, 2018.
- Jacobsen, J.-H., Behrmann, J., Zemel, R., and Bethge, M. Excessive invariance causes adversarial vulnerability. In *International Conference on Learning Representations*, 2019.
- Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. *arXiv preprint arXiv:1812.04948*, 2018.
- Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. *Advances in Neural Information Processing Systems*, 2018.
- Kingma, D. P., Mohamed, S., Rezende, D. J., and Welling, M. Semi-supervised learning with deep generative models. In *Advances in neural information processing systems*, pp. 3581–3589, 2014.
- Louizos, C., Swersky, K., Li, Y., Welling, M., and Zemel, R. The variational fair autoencoder. *International Conference on Learning Representations*, 2016.
- Lu, Y., Zhong, A., Li, Q., and Dong, B. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. *arXiv preprint arXiv:1710.10121*, 2017.
- Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018.
- Nalisnick, E., Matsukawa, A., Teh, Y. W., Gorur, D., and Lakshminarayanan, B. Hybrid models with deep and invertible features. *NeurIPS workshop on Bayesian deep learning*, 2018.
- Oliver, A., Odena, A., Raffel, C. A., Cubuk, E. D., and Goodfellow, I. Realistic evaluation of deep semi-supervised learning algorithms. In *Advances in Neural Information Processing Systems 31*. 2018.
- Papamakarios, G., Murray, I., and Pavlakou, T. Masked autoregressive flow for density estimation. In *Advances in Neural Information Processing Systems*, 2017.
- Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. Image transformer. In *ICML*, volume 80 of *JMLR Workshop and Conference Proceedings*, pp. 4052–4061. JMLR.org, 2018.
- Ramesh, A. and LeCun, Y. Backpropagation for implicit spectral densities. *arXiv preprint arXiv:1806.00499*, 2018.
- Rezende, D. J. and Mohamed, S. Variational inference with normalizing flows. *Proceedings of the 32nd International Conference on International Conference on Machine Learning*, 2015.
- Ruthotto, L. and Haber, E. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.
- Schott, L., Rauber, J., Bethge, M., and Brendel, W. Towards the first adversarially robust neural network model on MNIST. In *International Conference on Learning Representations*, 2019.
- Sedghi, H., Gupta, V., and Long, P. M. The singular values of convolutional layers. In *International Conference on Learning Representations*, 2019.
- Sokoli, J., Giryes, R., Sapiro, G., and Rodrigues, M. R. D. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.
- Tsuzuku, Y., Sato, I., and Sugiyama, M. Lipschitz-margin training: Scalable certification of perturbation invariance for deep neural networks. *Advances in Neural Information Processing Systems*, 2018.
- Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016a.
- Van Den Oord, A., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, pp. 1747–1756, 2016b.
- Van Den Oord, A., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.
- Withers, C. S. and Nadarajah, S. $\log \det a = \text{tr} \log a$. *International Journal of Mathematical Education in Science and Technology*, 41(8):1121–1124, 2010.
- Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.
- Zhao, T., Zhang, D., Sun, Z., and Lee, H. Information regularized neural networks, 2019. URL <https://openreview.net/forum?id=BJgvg30ctX>.

A. Additional Lemmas and Proofs

Proof. (Theorem 1)

Since ResNet F_θ is a composition of functions, it is invertible if each block F_θ^t is invertible. Let $x_{t+1} \in \mathbb{R}^d$ be arbitrary and consider the backward Euler discretization $x_t = x_{t+1} - hf_{\theta_t}(x_t) = x_{t+1} - g_{\theta_t}(x_t)$. Re-writing as a iteration yields

$$x_t^0 := x_{t+1} \text{ and } x_t^{k+1} := x_{t+1} - g_{\theta_t}(x_t^k), \quad (6)$$

where $\lim_{k \rightarrow \infty} x_t^k = x_t$ is the fixed point if the iteration converges. As $g_{\theta_t} : \mathbb{R}^d \rightarrow \mathbb{R}^d$ is an operator on a Banach space, the contraction condition $\text{Lip}(g_{\theta_t}) < 1$ guarantees convergence due to the Banach fixed point theorem. \square

Remark 5. The condition above was also stated in [Zhao et al. \(2019\)](#) (Appendix D), however, their proof restricts the domain of the residual block g to be bounded and applies only to linear operators g , because the inverse was given by a convergent Neumann-series.

Proof. (Lemma 2)

First note, that $\text{Lip}(F) \leq 1 + L$ follows directly from the addition of Lipschitz constants. For the inverse, consider

$$\begin{aligned} \|F(x) - F(y)\|_2 &= \|x - y + g(x) - g(y)\|_2 \\ &= \|x - y - (-g(x) + g(y))\|_2 \\ &\geq \|x - y\|_2 - \|-g(x) + g(y)\|_2 \\ &\geq \|x - y\|_2 - \|(-1)(g(x) - g(y))\|_2 \\ &\geq \|x - y\|_2 - \|g(x) - g(y)\|_2 \\ &\geq \|x - y\|_2 - L\|x - y\|_2, \end{aligned} \quad (7)$$

where we apply the reverse triangular inequality in (7) and apply the Lipschitz constant of g . Denote $x = F^{-1}(z)$ and $y = F^{-1}(w)$ for $z, w \in \mathbb{R}^d$, which is possible since F^{-1} is surjective. Inserting above yields

$$\begin{aligned} \|F(F^{-1}(z)) - F(F^{-1}(w))\|_2 &\geq (1 - L)\|F^{-1}(z) - F^{-1}(w)\|_2 \\ \iff \frac{1}{1 - L}\|z - w\|_2 &\geq \|F^{-1}(z) - F^{-1}(w)\|_2, \end{aligned}$$

which holds for all z, w . \square

Lemma 6 (Positive Determinant of Jacobian of Residual Layer). *Let $F(x) = (I + g(\cdot))(x)$ denote a residual layer and $J_F(x) = I + J_g(x)$ its Jacobian at $x \in \mathbb{R}^d$. If $\text{Lip}(g) < 1$, then it holds λ_i of $J_F(x)$ are positive for all x and thus*

$$|\det[J_F(x)]| = \det[J_F(x)],$$

where λ_i denotes the eigenvalues.

Proof. (Lemma 6)

First, we have $\lambda_i(J_F) = \lambda_i(J_g) + 1$ and $\|J_g(x)\|_2 < 1$ for all x due to $\text{Lip}(g) < 1$. Since the spectral radius $\rho(J_g) \leq \|J_g\|_2$, it is $|\lambda_i(J_g)| < 1$. Hence, $\text{Re}(\lambda_i(J_F)) > 0$ and thus $\det J_F = \prod_i (\lambda_i(J_g) + 1) > 0$. \square

Lemma 7 (Lower and Upper Bounds of an invertible ResNet on log-determinant). *Let $F_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^d$ with $F_\theta = (F_\theta^1 \circ \dots \circ F_\theta^T)$ denote an invertible ResNet with blocks $F_\theta^t = I + g_{\theta_t}$. Then, we can obtain the following bounds*

$$\begin{aligned} d \sum_{t=1}^T \ln(1 - \text{Lip}(g_t)) &\leq \ln |\det J_F(x)| \\ d \sum_{t=1}^T \ln(1 + \text{Lip}(g_t)) &\geq \ln |\det J_F(x)|, \end{aligned}$$

for all $x \in \mathbb{R}^d$.

Proof. (**Lemma 7**)

First, the sum over the layers is due to the function composition, because $J_F(x) = \prod_t J_{F^t}(x)$ and

$$\ln |\det J_F(x)| = \ln \left(\prod_{t=1}^T \det J_{F^t}(x) \right) = \sum_{t=1}^T \ln \det J_{F^t}(x),$$

where we use the positivity of the determinant, see Lemma 6. Furthermore, note that

$$\sigma_d(A)^d \leq \prod_i \sigma_i(A) = |\det A| \leq \sigma_1(A)^d$$

for a matrix A and largest singular values σ_1 and smallest σ_d . Furthermore, we have $\sigma_i(J_{F^t}) \leq (1 + \text{Lip}(g_t))$ and $\sigma_d(J_{F^t}) \leq (1 - \text{Lip}(g_t))$, which follows from Theorem 2. Inserting this and applying the logarithm rules finally yields the claimed bounds. \square

Proof. (**Theorem 3**)

We begin by noting that

$$\begin{aligned} |PS(J_g, n) - \text{tr} \ln(J_g)| &= \left| \sum_{k=n+1}^{\infty} (-1)^{k+1} \frac{\text{tr}(J_g^k)}{k} \right| \\ &\leq \sum_{k=n+1}^{\infty} \left| (-1)^{k+1} \frac{\text{tr}(J_g^k)}{k} \right| \\ &\leq \sum_{k=n+1}^{\infty} \left| \frac{\text{tr}(J_g^k)}{k} \right| \\ &\leq d \sum_{k=n+1}^{\infty} \frac{\text{Lip}(g)^k}{k}, \end{aligned} \tag{8}$$

where inequality (8) follows from

$$\begin{aligned} |\text{tr}(J^k)| &\leq \left| \sum_{i=1}^d \lambda_i(J^k) \right| \leq \sum_{i=1}^d |\lambda_i(J^k)| \leq d \rho(J^k) \\ &\leq d \|J^k\|_2 \leq d \|J\|_2^k \leq d \text{Lip}(g)^k. \end{aligned} \tag{9}$$

We note that the full series $\sum_{k=1}^{\infty} \frac{\text{Lip}(g)^k}{k} = -\ln(1 - \text{Lip}(g))$ thus we can bound the approximation error by

$$|PS(J_g, n) - \text{tr} \ln(J_g)| \leq -d \left(\ln(1 - \text{Lip}(g)) + \sum_{k=1}^n \frac{\text{Lip}(g)^k}{k} \right)$$

\square

Proof. (**Theorem 4**)

First, we derive the by differentiating the power series and using the linearity of the trace operator. We obtain

$$\begin{aligned} \frac{\partial}{\partial \theta_i} \ln \det (I + J_g(x, \theta)) &= \frac{\partial}{\partial \theta_i} \left(\sum_{k=1}^{\infty} (-1)^{k+1} \frac{\text{tr}(J_g^k(x, \theta))}{k} \right) \\ &= \text{tr} \left(\sum_{k=1}^{\infty} \frac{k(-1)^{k+1}}{k} J_g^{k-1}(x, \theta) \frac{\partial(J_g(x, \theta))}{\partial \theta_i} \right) \\ &= \text{tr} \left(\sum_{k=0}^{\infty} (-1)^k J_g^k(x, \theta) \frac{\partial(J_g(x, \theta))}{\partial \theta_i} \right). \end{aligned}$$

By definition of $\|\cdot\|_\infty$,

$$\|\nabla_\theta PS(J_g(\theta), \infty) - \nabla_\theta PS(J_g(\theta), n)\|_\infty = \max_{i=1, \dots, p} \left| \frac{\partial}{\partial \theta_i} PS(J_g(\theta), \infty) - \frac{\partial}{\partial \theta_i} PS(J_g(\theta), n) \right|,$$

which is why, we consider an arbitrary i from now on. It is

$$\begin{aligned} \left| \frac{\partial}{\partial \theta_i} PS(J_g(\theta), \infty) - \frac{\partial}{\partial \theta_i} PS(J_g(\theta), n) \right| &= \sum_{k=n+1}^{\infty} (-1)^k \text{tr} \left(J_g^k(x, \theta) \frac{\partial(J_g(x, \theta))}{\partial \theta_i} \right) \\ &\leq d \sum_{k=n+1}^{\infty} \text{Lip}(g)^K \left\| \frac{\partial J_g(x, \theta)}{\partial \theta_i} \right\|_2, \end{aligned} \quad (10)$$

where we used the same arguments as in estimation (9).

In order to bound $\left\| \frac{\partial J_g(x, \theta)}{\partial \theta_i} \right\|_2$, we need to look into the design of the residual block. We assume contractive and element-wise activation functions (hence $\phi'(\cdot) < 1$) and N linear layers W_i in a residual block. Then, we can write the Jacobian as a matrix product

$$J_g(x, \theta) = W_N^T D_N \cdots W_1^T D_1,$$

where $D_i = \text{diag}(\phi'(z_{i-1}))$ with pre-activations $z_{i-1} \in \mathbb{R}^d$.

Since we need to bound the derivative of the Jacobian with respect to weights θ_i , double backpropagation (Drucker & Lecun, 1992) is necessary. In general, the terms $\|W_i^T\|_2$, $\|D_i\|_2$, $\|D_i^*\|_2 := \|\text{diag}(\phi''(z_{i-1}))\|_2$, $\left\| \left(\frac{\partial W_i}{\partial \theta_i} \right)^T \right\|_2$ and $\|x\|_2$ appear in the bound of the derivative. Hence, in order to bound $\left\| \frac{\partial J_g(x, \theta)}{\partial \theta_i} \right\|_2$, we bound the previous terms as follows

$$\|W_i^T\|_2 \leq \text{Lip}(g) \quad (11)$$

$$\|D_i\|_2 \leq \text{const}, \quad (12)$$

$$\|D_i^*\|_2 \leq \text{const} \quad (13)$$

$$\|x\|_2 \leq \text{const} \quad (14)$$

$$\left\| \left(\frac{\partial W_i}{\partial \theta_i} \right)^T \right\|_2 \leq \|W_i\|_F + s. \quad (15)$$

In particular, (12) is due to the assumption of a Lipschitz activation function and (13) due to assuming a Lipschitz derivative of the activation function. Note, that we are using continuously differentiable activations functions (hence, not ReLU), where this assumptions holds for common functions like ELU, softplus and tanh. Furthermore, (14) holds by assuming bounded inputs and due to the network being Lipschitz. To understand the bound (15), we denote s as the amount of parameter sharing of θ_i . For example, if θ_i is a entry from a convolution kernel, $s = w * h$ with w spatial width and h spatial height. Then

$$\left\| \left(\frac{\partial W_i}{\partial \theta_i} \right)^T \right\|_2 \leq \|W_i\|_F + s,$$

since

$$\frac{\partial W_{lm}(x, \theta)}{\partial \theta_i} = \begin{cases} 1, & \text{if } W_{lm} = \theta_i \\ 0, & \text{else} \end{cases}.$$

Hence, as each term appearing in the second derivative $\left\| \frac{\partial J_g(x, \theta)}{\partial \theta_i} \right\|_2$ is bounded, we can introduce the constant $a(g, \theta, x) < \infty$ which depends on the parameters, the implementation of g and the inputs x . Note, that we do not give an exact bound on $\left\| \frac{\partial J_g(x, \theta)}{\partial \theta_i} \right\|_2$, since we are only interesting in the existence of such a bound in order to proof the convergence in the claim.

Inserting above in (10) and denoting $c := \text{Lip}(g)$, yields

$$\left| \frac{\partial}{\partial \theta_i} PS(J_g(\theta), \infty) - \frac{\partial}{\partial \theta_i} PS(J_g(\theta), n) \right| \leq d a(g, \theta, x) \sum_{k=n+1}^{\infty} c^K = \tilde{a}(d, g, \theta, x) \left(\frac{1}{1-c} - \left(\frac{1-c^n}{1-c} + c^n \right) \right).$$

Letting $f(n) := \tilde{a}(d, g, \theta, x) \left(\frac{1}{1-c} - \left(\frac{1-c^n}{1-c} + c^n \right) \right)$ and $g(n) = c^n$, then

$$\lim_{n \rightarrow \infty} \left| \frac{f(n)}{g(n)} \right| = \text{const} < \infty,$$

which proves the claimed convergence rate. \square

B. Verification of Invertibility

Invertibility of Learned Mappings In this experiment we train standard ResNets and i-ResNets with various layer-wise Lipschitz coefficients ($c \in \{.3, .5, .7, .9\}$). After training, we inspect the learned transformations at each layer by computing the largest singular value of each linear mapping based on the approach in [Sedghi et al. \(2019\)](#). It can be seen clearly (Figure 6 left) that the standard and BatchNorm models have many singular values above 1, making their residual connections non-invertible. Conversely, in the i-ResNet models (Figure 6 right), all singular values are below 1 (and roughly equal to c) indicating their residual connections are invertible.

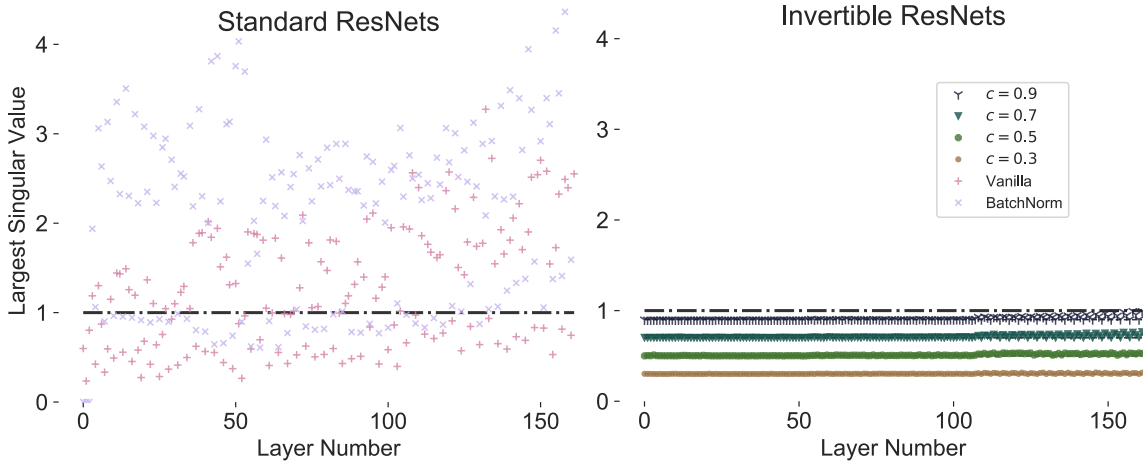


Figure 6. Maximal singular value of each layer’s convolutional operator for various trained ResNets on CIFAR10. *Left*: Vanilla and Batchnorm ResNet singular values. It is likely that the baseline ResNets are not invertible as roughly two thirds of their layers have singular values fairly above one, making the blocks non-contractive. *Right*: Singular values for our 4 spectrally normalized ResNets. The regularization is effective and in every case the single ResNet block remains a contraction.

Computing Inverses with Fixed-Point Iteration Here we numerically compute inverses in our trained models using the fixed-point iteration, see Algorithm 1. We invert each residual connection using 100 iterations (to ensure convergence). We see that i-ResNets can be inverted using this method whereas with standard ResNets this is not guaranteed (Figure 7 top). Interestingly, on MNIST we find that standard ResNets are indeed invertible after training on MNIST (Figure 7 bottom).

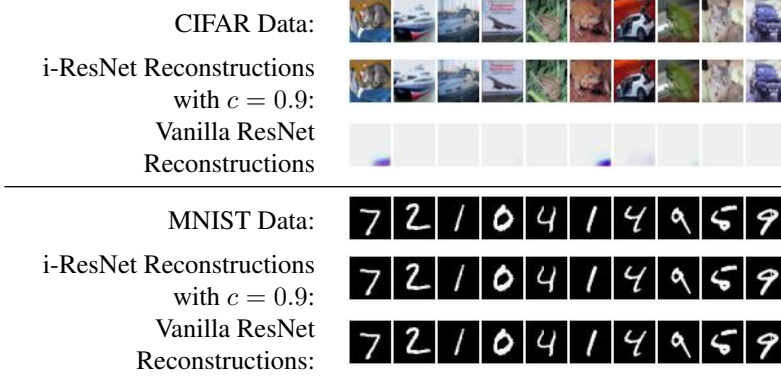


Figure 7. Original images (top), i-ResNets with $c = 0.9$ (middle) and reconstructions from vanilla (bottom). Surprisingly, MNIST reconstructions are close to exact for both models, even without explicitly enforcing the Lipschitz constant. On CIFAR10 however, reconstructions completely fail for the vanilla ResNet, but are qualitatively and quantitatively exact for our proposed network.

C. Experimental Details

C.1. Classification

Architecture We use pre-activation ResNets with 39 convolutional bottleneck blocks with 3 convolution layers each and kernel sizes of 3×3 , 1×1 , 3×3 respectively. All models use the ELU nonlinearity (Clevert et al., 2015). In the BatchNorm version, we apply batch normalization before every nonlinearity and in the invertible models we use ActNorm (Kingma & Dhariwal, 2018) before each residual block. The network has 2 down-sampling stages after 13 and 26 blocks where a dimension squeezing operation is used to decrease the spatial resolution. This reduces the spatial dimension by a factor of two in each direction, while increasing the number of channels by a factor of four. All models transform the data to a $8 \times 8 \times 256$ tensor to which we apply BatchNorm, a nonlinearity, and average pooling to a 256-dimensional vector. A linear classifier is used on top of this representation.

Injective Padding Since our invertible models are not able to increase the dimension of their latent representation, we use injective padding (Jacobsen et al., 2018) which concatenates channels of 0’s to the input, increasing the size of the transformed tensor. This is analogous to the standard practice of projecting the data into a higher-dimensional space using a non-ResNet convolution at the input of a model, but this mapping is reversible. We add 13 channels of 0’s to all models tested, thus the input to our first residual block is a tensor of size $32 \times 32 \times 16$. We experimented with removing this step but found it led to approximately a 2% decrease in accuracy for our CIFAR10 models.

Training We train for 200 epochs with momentum SGD and a weight decay of $5e-4$. The learning rate is set to 0.1 and decayed by a factor of 0.2 after 60, 120 and 160 epochs. For data-augmentation, we apply random shifts of up to two pixels for MNIST and shifts/ random horizontal flips for CIFAR(10/100) during training. The inputs for MNIST are normalized to $[-0.5, 0.5]$ and for CIFAR(10/100) normalize by subtracting the mean and dividing by the standard deviation of the training set.

C.2. Generative Modeling

Toy Densities We used 100 residual blocks, where each residual connection is a multilayer perceptron with state sizes of $2-64-64-64-2$ and ELU nonlinearities (Clevert et al., 2015). We used ActNorm (Kingma & Dhariwal, 2018) after each residual block. The change in log density was computed exactly by constructing the full Jacobian during training and visualization.

MNIST and CIFAR The structure of our generative models closely resembles that of Glow. The model consists of “scale-blocks” which are groups of i-ResNet blocks that operate at different spatial resolutions. After each scale-block, apart from the last, we perform a squeeze operation which decreases the spatial resolution by 2 in each dimension and multiplies the number of channels by 4 (invertible downsampling).

Our MNIST and CIFAR10 models have three scale-blocks. Each scale-block has 32 i-ResNet blocks. Each i-ResNet block consists of three convolutions of 3×3 , 1×1 , 3×3 filters with ELU (Clevert et al., 2015) nonlinearities in between. Each convolutional layer has 32 filters in the MNIST model and 512 filters in the CIFAR10 model.

We train for 200 epochs using the Adamax (Kingma & Ba, 2014) optimizer with a learning rate of .003. Throughout training we estimate the log-determinant in Equation (3) using the power-series approximation (Equation (4)) with ten terms for the MNIST model and 5 terms for the CIFAR10 model.

Evaluation During evaluation we use the bound presented in Section 3.3 to determine the number of terms needed to give an estimate with bias less than .0001 bit/dim. We then average over enough samples from Hutchinson’s estimator such that the standard error is less than .0001 bit/dim, thus we can safely report our model’s bit/dim accurate up to a tolerance of .0002.

Choice of Nonlinearity Differentiating our log-determinant estimator requires us to compute second derivatives of our neural network’s output. If we were to use a nonlinearity with discontinuous derivatives (i.e. ReLU), then these values are not defined in certain regions. This can lead to unstable optimization. To guarantee the quantities required for optimization always exist, we recommend using nonlinearities which have continuous derivatives such as ELU (Clevert et al., 2015) or softplus. In all of our experiments we use ELU.

D. Fixed Point Iteration Analysis

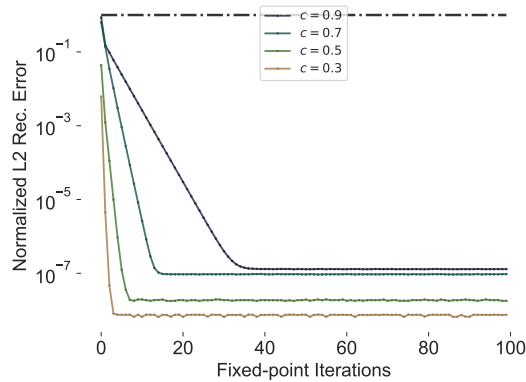


Figure 8. Trade-off between number of fixed point iterations and reconstruction error (log scale) for computing the inverse for different normalization coefficients of trained invertible ResNets (on CIFAR10). The reconstruction error decays quickly. 5-20 iterations are sufficient respectively to obtain visually perfect reconstructions. Note that one iteration corresponds to the time for one forward pass, thus inversion is approximately 5-20 times slower than inference. This corresponds to a reconstruction time of 0.15-0.75 seconds for a batch of 100 CIFAR10 images with 5-20 iterations and 4.3 seconds with 100 iterations.

E. Evaluating the Bias of Our Log-determinant Estimator

Here we numerically evaluate the bias of the log-determinant estimator used to train our generative models (Equation (4)). We compare the true value (computed via brute-force) with the estimator’s mean and standard deviation as the number of terms in the power series is increased. After 10 terms, the estimator’s bias is negligible and after 20 terms it is numerically 0. This is averaged over 1000 test examples.

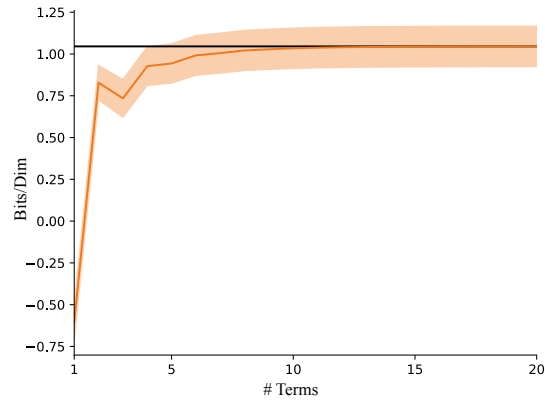
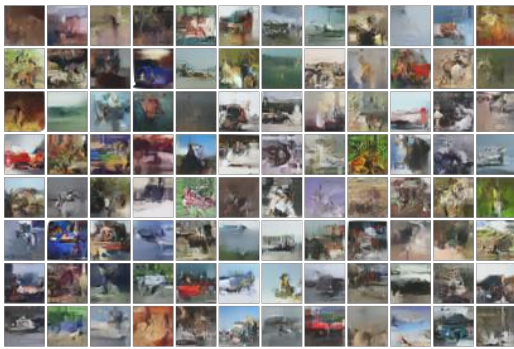


Figure 9. Convergence of approximation error of log-determinant estimator when varying the number of terms used in the power series. The variance is due to the stochastic trace estimator.

F. Additional Samples of i-ResNet flow



CIFAR10 samples.



MNIST samples.