iVPF: Numerical Invertible Volume Preserving Flow for Efficient Lossless Compression

Shifeng Zhang   Chen Zhang   Ning Kang   Zhenguo Li

Huawei Noah's Ark Lab

{zhangshifeng4, chenzhang10, kang.ning2, li.zhenguo}@huawei.com

Abstract

It is nontrivial to store rapidly growing big data nowadays, which demands high-performance lossless compression techniques. Likelihood-based generative models have witnessed their success on lossless compression, where flow based models are desirable in allowing exact data likelihood optimisation with bijective mappings. However, common continuous flows are in contradiction with the discreteness of coding schemes, which requires either 1) imposing strict constraints on flow models that degrades the performance or 2) coding numerous bijective mapping errors which reduces the efficiency. In this paper, we investigate volume preserving flows for lossless compression and show that a bijective mapping without error is possible. We propose Numerical Invertible Volume Preserving Flow (iVPF) which is derived from the general volume preserving flows. By introducing novel computation algorithms on flow models, an exact bijective mapping is achieved without any numerical error. We also propose a lossless compression algorithm based on iVPF. Experiments on various datasets show that the algorithm based on iVPF achieves state-of-the-art compression ratio over lightweight compression algorithms.

1. Introduction

Lossless compression is widely applied to efficiently store a growing amount of data in the big data era. It is indicated by Shannon's source coding theorem that the optimal codeword length is lower bounded by the entropy of the data distribution [25]. Many coding methods [18, 39, 9] succeed in reaching the theoretical codelength, and are widely applied in various compression algorithms [10, 6, 30, 31].

Probabilistic modelling techniques can be adopted for lossless compression with coding algorithms, in which the minimum possible codelength would be essentially bounded by the negative log-likelihood of the probabilistic model. Recent advances of deep generative models
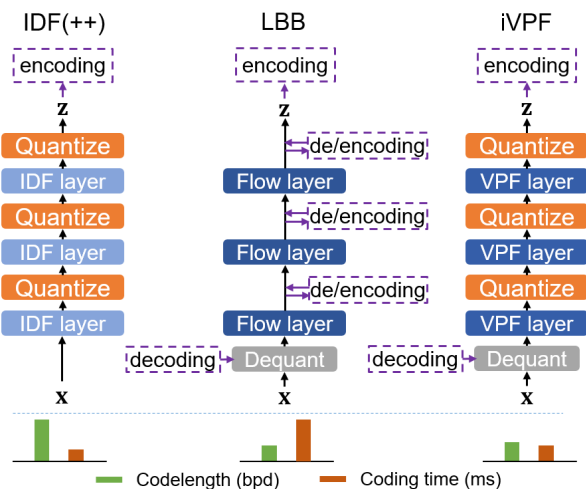

Figure 1. Illustration of IDF(++), LBB and the proposed iVPF model. VPF denotes volume preserving flow. Flow layers with darker colour denotes higher expressive power. The compression codelength and coding time are listed below (lower is better).

have shown great success in maximising the likelihood of various types of data including images [22, 21, 32, 8], videos [24, 19] and audios [29, 28]. Various lossless compression algorithms have been discovered based on different types of generative models [26, 35, 15, 3, 13, 36, 23, 27, 5]. They witnessed better compression ratio compared to those without machine learning techniques.

In this paper, we focus on coding with flow-based generative models [8, 21, 8, 29, 24]. Although variational autoencoders (VAEs) [22, 11] and auto-regressive [32, 37] models are both probabilistic models, they either suffer from inferior likelihood estimation (VAEs), or involve huge computational overhead (auto-regressive). Normalising flow models admit bijective mappings between input data and latent variables, and the latter is adopted for likelihood computation given the prior. However, common flows are continuous and numerical errors always occur for forward and inverse computation [2], while lossless compression involves discrete data and any numerical error is unaccept-

able. To solve the above issue, IDF [15] and IDF++ [3] introduce discrete flows by adding discrete constraints on the model, but the theoretical likelihood is inferior to that of general continuous flows. LBB [13] succeeds in coding with general flows and the codelength is close to the theoretical lower bound of negative log-likelihood, but it is extraordinarily time-consuming.

To circumvent the above issues, we introduce iVPF, a Numerical Invertible Volume Preserving Flow model with high expressivity and efficient coding. We point out that general flows are almost not bijective in practice, because the real-world data are discrete in nature, resulting the difficulty for lossless compression. The proposed iVPF is constructed from volume preserving flow [7, 34] to achieve the bijective mapping on discrete space, and an efficient lossless compression algorithm based on iVPF is developed. [1] The main contributions of this work are summarised as follows:

1. We investigate the importance of volume preserving flow in the lossless compression context, and propose the novel computation algorithm Modular Affine Transformation (MAT) which achieves exact bijective mapping without any numerical error for coupling layers of volume preserving flows.

2. Based on MAT, we propose the Numerical Invertible Volume Preserving Flow (iVPF) model and an efficient lossless compression algorithm for iVPF.

3. Experiments show the effectiveness of the proposed lossless compression algorithm, where the compression performance is superior over the state-of-the-art lightweight methods.

2. Related Work

Entropy Coding for Lossless Compression We put our work in the regime of machine learning techniques for lossless compression. As indicated by Shannon's source coding theorem, the best possible lossless compression performance is given by entropy of the data distribution [25]. Various coding schemes, e.g., Huffman coding [18], Arithmetic coding [39] and Asymmetric Numeral Systems [9], are developed so that symbols with certain distribution density can be encoded with codelength close to the entropy. With recent advances in deep generative models, the machine learning community has been actively investigating learned probabilistic models which represent the data distribution more accurately and achieve state-of-the-art performance for lossless compression [35, 15, 13, 36, 26, 3]. **Explicit Probabilistic Modelling** Due to the fact that entropy coders require explicit evaluation of density functions,

predominant approaches are based on Variational Inference and Maximum Likelihood. VAEs leverage the latent variables for expressive explicit mixture models and employ bits-back techniques [38, 12] to encode the hierarchical distribution structure. The theoretical codelength with VAEs is exactly the variational lower bound. However, there exists a gap between the lower bound and the true data entropy, resulting in inferior performance on compression ratio. L3C [26] augments the data variable with hierarchical latents, and the compression ratio is compromised due to the introduction of latent variables. Autoregressive models [32] stipulates probabilistic models in a fully observable way in which the probability is factorised on the pixel level. They perform well for image generation, but suffer from high computational costs. Compared to the previous explicit probability models, flow models admit direct evaluation of data density rather than the variational lower bound, yielding more efficient coding schemes in theory.

Coding with Flow Models The idea of normalizing flows is based on change of variables, where the diffeomorphism would introduce a push forward map on the original distribution. In order to have a simple Jacobian matrix determinant evaluation in the push forward probability, the diffeomorphisms are usually composed of successive simple invertible layers, e.g. coupling layers. Even though flows could admit plausible image generation [8, 21], the numeric quantization error is one of the main challenges for lossless compression. Hoogeboom et al [15] approached this problem by proposing a discrete flow for integers (IDF). IDF++ [3] introduces several improvements to IDF. However, this very design imposes strict discrete constraints as well as strong constraints on coupling layer structure and thus undermines the expressive power of the flow model. Ho et al [13] proposed a local bits-back coding technique (LBB) for more flexible flow models. Nevertheless, it suffers from a large quantity of encoding/decoding operations on each layer, which greatly affects the efficiency of lossless compression. Full Jocabian computation may reduce coding times, but computing it is intractable.

The technique introduced in this work enables compression with any volume-preserving flows. Volume preserving flows only introduce limited constraints on general flows without hurting much expressive power, and more types of volume-preserving layers can be considered compared to IDF. With novel quantization methods, the invertible computation of flows without any numeric error is achieved, while the compression ratio can reach the entropy of distribution estimated with flows.

3. Method

In this section, we introduce iVPF, a Numerical Invertible Volume Preserving Flow for efficient lossless compression. Note that the invertible network in a traditional flow

---

$\mathbf{z} = f(\mathbf{x})$ is not exactly a bijective map as numerical errors are encountered in arithmetic computations of floating points [2]. Although numerical errors are allowed in normal data generation modelling, no error shall occur in a lossless compression task. iVPF can be built upon any volume preserving flows and is exactly bijective by eliminating floating-point numeral errors in forward and inverse computation, making it feasible for efficient lossless compression by encoding latent variables $\mathbf{z} = f(\mathbf{x})$.

3.1. Recap: Volume Preserving Flows

Normalising flows are normally constructed between a continuous domain $\mathcal{X}$ and co-domain $\mathcal{Z}$. Let $f : \mathcal{X} \to \mathcal{Z}$ be a bijective map defined by a flow model. The model distribution $p_X(\mathbf{x})$ of input data $\mathbf{x}$ can be expressed with the latent variable $\mathbf{z}$ such that

$$\log p_X(\mathbf{x}) = \log p_Z(\mathbf{z}) + \log \left| \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right| \quad \mathbf{z} = f(\mathbf{x}), \quad (1)$$

where $\left| \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right|$ is the absolute value of the determinant of the Jacobian matrix. The flow model is constructed with composition of flow layers such that $f = f_L \circ ... \circ f_2 \circ f_1$. Denoting $\mathbf{y}_l = f_l(\mathbf{y}_{l-1})$ and $\mathbf{y}_0 = \mathbf{x}, \mathbf{z} = \mathbf{y}_L$, we have $\log p_X(\mathbf{x}) = \log p_Z(\mathbf{z}) + \sum_{i=1}^{L} \log \left| \frac{\partial \mathbf{y}_l}{\partial \mathbf{y}_{l-1}} \right|$. The distribution $p_Z(\mathbf{z})$ can be stipulated from any parametric distribution families, e.g., mixture of Gaussians, mixture of logistic distributions, etc. And the parameters of such distributions can be defined manually or learned jointly with flow parameters. The inverse flow is computed as $f^{-1} = f_1^{-1} \circ ... \circ f_L^{-1}$. As discussed in Sec. 3.2, general non-volume preserving flows will cause practical issues when used for lossless compression, and so we focus on volume preserving flows.

Volume preserving flows have the property that $\left| \frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right| = 1$, and therefore the density of $\mathbf{x}$ can be directly computed as $p_X(\mathbf{x}) = p_Z(\mathbf{z})$. A volume preserving flow model is a composition of volume preserving layers where $\left| \frac{\partial \mathbf{y}_l}{\partial \mathbf{y}_{l-1}} \right| = 1$ for any $\mathbf{y}_l = f_l(\mathbf{y}_{l-1}), l = 1, 2, ..., L$. According to the following section, the volume preserving layers can be constructed from most general flow layers with limited constraints, thus its expressive power is close to the general flow. Besides, volume preserving flows can use broader types of flow layers than IDF, implying better expressive power than IDF.

Next, we briefly review the three typical building layers adopted in the flow model.

(1) Coupling layer. For simplicity of notation, we drop the subscript for layer index in f_l, and use $\mathbf{x}$ and $\mathbf{z}$ as layer input and output. A coupling layer with volume preserving is defined by

$$\mathbf{z} = [\mathbf{z}_a, \mathbf{z}_b] = f(\mathbf{x}) = [\mathbf{x}_a, \mathbf{x}_b \odot \exp(\mathbf{s}(\mathbf{x}_a)) + \mathbf{t}(\mathbf{x}_a)], \quad (2)$$

where $\mathbf{x} = [\mathbf{x}_a, \mathbf{x}_b]$, $\odot$ denotes element-wise multiplication of vectors, and the functions $\mathbf{s}(\cdot)$ and $\mathbf{t}(\cdot)$ can be modelled with neural networks so that

$$\sum_i \mathbf{s}_i(\mathbf{x}_a) = 0 \quad (3)$$

is the volume preserving constraint in which $\mathbf{s}_i$ is the ith element of $\mathbf{s}$. Note that in IDF, all elements of $\mathbf{s}(\mathbf{x}_a)$ are set to be zero, conveying the limited expressive power.

If the dimension of $\mathbf{z}_a$ is zero, the outputs of $\mathbf{s}(\mathbf{x}_a), \mathbf{t}(\mathbf{x}_a)$ are learned linear coefficients irrelevant with $\mathbf{x}$, and it degrades to **invertible linear layer**.

(3) Invertible 1×1 **convolution.** This layer can be converted to matrix multiplication along channels [21]. Let $\mathbf{W}$ be the weight of a 1×1 convolution layer. It is the square matrix with $|\det(\mathbf{W})| = 1$ to ensure volume preserving.

In practice, $\mathbf{W}$ can be parameterised with LU decomposition as

$$\mathbf{W} = \mathbf{PL\Lambda U} \quad (4)$$

where $\mathbf{P}$ is the permutation matrix, $\mathbf{L}, \mathbf{U}$ are the lower and upper triangular matrix respectively whose diagonal are all ones, and $\mathbf{\Lambda}$ is the diagonal matrix with $|\det(\mathbf{\Lambda})| = 1$. For simplicity, we usually use $\mathbf{\Lambda} = \mathbf{I}$.

If $\mathbf{L}, \mathbf{U}$ are both identity matrices, then the 1×1 convolution layer reduces to **permutation layer**.

(4) Factor-out layer. To reduce the computation cost, the latent $\mathbf{z}$ can be modelled with a factored model $p(\mathbf{z}) = p(\mathbf{z}_L)p(\mathbf{z}_{L-1}|\mathbf{z}_L)...p(\mathbf{z}_1|\mathbf{z}_2, ..., \mathbf{z}_L)$, which can be modelled with volume preserving flow. We give an example of $L = 2$ [8, 15] such that

$$[\mathbf{z}_1, \mathbf{y}_1] = f_1(\mathbf{x}), \quad \mathbf{z}_2 = f_2(\mathbf{y}_1), \quad \mathbf{z} = [\mathbf{z}_1, \mathbf{z}_2] \quad (5)$$

where f_1, f_2 are both volume preserving flows. Then the probability of $\mathbf{x}$ is given by $p(\mathbf{x}) = p(\mathbf{z}_2)p(\mathbf{z}_1|\mathbf{y}_1)$.

3.2. Volume Preserving Flows in Discrete Space

With normalizing flows, the input data may be encoded by first converting input data to latents, and then encoding latents with prior distribution. As all data are stored in binary formats, the data and latents should be discrete. However, normalizing flows naturally work for continuous distributions. In what follows, quantization is needed to make the latents discrete, and discrete bijections between input data and latents should be established. We show that volume preserving flows have the potential to hold the property.

3.2.1 k-precision Quantization

In general practice, the continuous space is discretized to bins with volume δ, where certain continuous data $\mathbf{x}$ is quantized to the centre of a certain bin $\bar{\mathbf{x}}$. The probability mass function is approximated by $P(\bar{\mathbf{x}}) \approx p_X(\bar{\mathbf{x}})\delta$ for the corresponding bin. Specifically, we use a similar quantization strategy as in IDF [15] where the floating points are quantized to decimals with certain precision k as

$$\lfloor x \rceil = \frac{\text{round}(2^k \cdot x)}{2^k}, \quad (6)$$

where round is the rounding operation. We refer to such quantization scheme as *k-precision quantization*. It is worth noting that with large k, x and $\lfloor x \rceil$ can be very close, since $|x - \lfloor x \rceil| \leq 2^{-k-1}$. And the bin volume can be computed as $\delta = 2^{-kd}$, where d is the dimension of $\mathbf{x}$.

Due to the discrete nature of electronic data, e.g., 8 bit images, for any specific dataset to compress, there exists an integer K, such that for any $k > K$, k-precision quantization is sufficient to represent all possible data values. As a result, we assume that k is chosen large enough s.t. $\lfloor \mathbf{x} \rceil = \mathbf{x}$ for the input data. Moreover, it is worth noting that the same k-precision quantization scheme is employed for the data $\mathbf{x}$ and all of the latent variables output by each flow layer. And given discrete latents $\bar{\mathbf{z}} = \lfloor f(\mathbf{x}) \rceil$, the inverse flow must recover the inputs such that $\bar{f}^{-1}(\bar{\mathbf{z}}) \equiv \mathbf{x}$.

For a chosen k-precision quantization, the theoretical codelength of $\mathbf{x}$ is $-\log(p_X(\mathbf{x})\delta)$. With change of variables in flow, $\mathbf{x}$ is coded by coding $\bar{\mathbf{z}}$ with theoretical length $-\log(p_Z(\bar{\mathbf{z}})\delta)$, where $p_Z(\bar{\mathbf{z}}) = p_X(\mathbf{x})|\frac{\partial \mathbf{x}}{\partial \mathbf{z}}(\bar{\mathbf{z}})|$.

3.2.2 Invertibility of Volume Preserving Flows

The essential difference of non-volume preserving flows from volume preserving flows is that the absolute Jacobian determinant is not constant 1 for all data points. We will see that this characteristic of non-volume preserving flows can lead to either waste of bits for compression or failure of bijection induction that is needed for lossless compression with flows. As a result, volume preserving flows are preferred for efficient lossless compression.

From Proposition 1, for a data point $\mathbf{x}_0$ with $|\frac{\partial \mathbf{z}}{\partial \mathbf{x}}(\mathbf{x}_0)| > 1$, the codelength by coding in the latent space can be longer than it needs to be which leads to suboptimal compression. From Proposition 2, for a data point $\mathbf{x}_0$ with $|\frac{\partial \mathbf{z}}{\partial \mathbf{x}}(\mathbf{x}_0)| < 1$, the bijection between domain and co-domain with the same discretization scheme cannot be guaranteed, which means an encoded data point cannot be correctly decoded. The proofs of two propositions can be seen in the Appendix B.

Proposition 1. *Let f be a smooth bijection from $\mathcal{X}$ to $\mathcal{Z}$. Assume for $\mathbf{x}_0$, it holds that $|\frac{\partial \mathbf{z}}{\partial \mathbf{x}}(\mathbf{x}_0)| > 1$. Then there exists an integer K, for any k-precision discretisation scheme where $k > K$, we have $-\log(p_Z(\lfloor f(\mathbf{x}_0) \rceil)\delta) > -\log(p_X(\mathbf{x})\delta)$.*

Proposition 2. *Let f be a smooth bijection from $\mathcal{X}$ to $\mathcal{Z}$. Assume for $\mathbf{x}_0$, it holds that $|\frac{\partial \mathbf{z}}{\partial \mathbf{x}}(\mathbf{x}_0)| < 1$. Then there exists an integer K, for any $k > K$, f cannot induce a bijection between the discretised domain $\bar{\mathcal{X}}$ and discretised co-domain $\bar{\mathcal{Z}}$.*

3.3. Numerical Invertible Volume Preserving Flow Layer

The previous section illustrated the necessity of efficient coding with volume preserving flows. In this subsection, we

---

Algorithm 1 Modular Affine Transformation (MAT)

Forward MAT: $\mathbf{z}_b = \bar{f}(\mathbf{x}_b) = \mathbf{s} \odot \mathbf{x}_b + \mathbf{t}$ given $(\mathbf{x}_b, r)$.
1: $\mathbf{x}_b \leftarrow 2^k \cdot \mathbf{x}_b$;
2: Set $m_0 = m_{d_b} = 2^C$, assert $0 \leq r < m_0$;
3: **for** $i = 1, 2, ..., d_b - 1$ **do**
4: $\quad m_i \leftarrow \text{round}(m_0/(\prod_{j=1}^i s_j));$ $\quad //s_i \approx m_{i-1}/m_i$
5: **end for**
6: **for** $i = 1, 2, ..., d_b$ **do**
7: $\quad v \leftarrow x_i \cdot m_{i-1} + r;$
8: $\quad y_i \leftarrow \lfloor v/m_i \rfloor, \quad r \leftarrow v \mod m_i;$ $\quad //y_i \approx x_i \cdot s_i$
9: **end for**
10: $\mathbf{y}_b = [y_1, ..., y_{d_b}]^\top;$
11: $\mathbf{z}_b \leftarrow \mathbf{y}_b/2^k + \lfloor \mathbf{t} \rfloor.$
12: **return** $\mathbf{z}_b, r$.

Inverse MAT: $\mathbf{x}_b = \bar{f}^{-1}(\mathbf{z}_b) = (\mathbf{z}_b - \mathbf{t}) \oslash \mathbf{s}$ given $(\mathbf{z}_b, r)$.
1: $\mathbf{y}_b \leftarrow 2^k \cdot (\mathbf{z}_b - \lfloor \mathbf{t} \rfloor);$
2: Set $m_0 = m_{d_b} = 2^C$, assert $0 \leq r < m_0$;
3: **for** $i = 1, 2, ..., d_b - 1$ **do**
4: $\quad m_i \leftarrow \text{round}(m_0/(\prod_{j=1}^i s_j));$ $\quad //s_i \approx m_{i-1}/m_i$
5: **end for**
6: **for** $i = d_b, ..., 2, 1$ **do**
7: $\quad v \leftarrow y_i \cdot m_i + r;$
8: $\quad x_i \leftarrow \lfloor v/m_{i-1} \rfloor, r \leftarrow v \mod m_{i-1}; //x_i \approx y_i/s_i$
9: **end for**
10: $\mathbf{x}_b \leftarrow \mathbf{x}_b/2^k.$
11: **return** $\mathbf{x}_b, r$.

---

introduce Numerical Invertible Volume Preserving Flows (iVPF), which are derived from existing volume preserving flows with k-precision quantization and novel arithmetic computation algorithms of flow layers. Denote by $\bar{f}$ the iVPF derived from the volume preserving flow f. An iVPF model is constructed with composition of numerical invertible volume preserving flow layers such that $\bar{f} = \bar{f}_L \circ ... \circ \bar{f}_1$, where $\mathbf{y}_{l-1} \equiv \bar{f}_l^{-1}(\bar{f}_l(\mathbf{y}_{l-1}))$. It holds that $\mathbf{x} \equiv \bar{f}^{-1}(\bar{f}(\mathbf{x}))$ for all quantized input data $\mathbf{x}$ with the small error between $\bar{\mathbf{z}} = \bar{f}(\mathbf{x})$ and $\mathbf{z} = f(\mathbf{x})$, therefore the efficient coding algorithm with iVPF is expected. As discussed below, the inputs and outputs of each flow layer in iVPF should be quantized such that $\mathbf{y}_l \equiv \lfloor \mathbf{y}_l \rfloor$ for all $l = 0, 1, ..., L$, and so as $\mathbf{z}$.

3.3.1 Numerical Invertible Coupling Layer with Modular Affine Transformation

Without loss of generality, denote by $\mathbf{s} = \exp(\mathbf{s}(\mathbf{x}_a)), \mathbf{t} = \mathbf{t}(\mathbf{x}_a)$. Given $\mathbf{x}_b \in \mathbb{R}^{d_b}$, $\mathbf{z}_b \in \mathbb{R}^{d_b}$ is computed by affine transformation such that $\mathbf{z}_b = \mathbf{s} \odot \mathbf{x}_b + \mathbf{t}$, and the inverse is $\mathbf{x}_b = (\mathbf{z}_b - \mathbf{t}) \oslash \mathbf{s}$[2]. It is clear that traditional multiplication and division do not ensure invertibility as $\lfloor (\lfloor \mathbf{s} \odot \mathbf{x}_b + \mathbf{t} \rfloor - \mathbf{t}) \oslash \mathbf{s} \rceil \neq \mathbf{x}_b$. Denote by z_i, x_i, s_i and t_i the ith element of $\mathbf{z}_b, \mathbf{x}_b, \mathbf{s}$ and $\mathbf{t}$ respectively. Note that $\prod_{i=1}^{d_b} s_i = 1$ in iVPF. By introducing an auxiliary variable

---
[2]$\oslash$ denotes element-wise division.

$\mathbf{x} = [40, 50, 60]$, $\boldsymbol{s} = \left[\frac{8}{6}, \frac{6}{11}, \frac{11}{8}\right]$. Expected $\mathbf{z} \approx [53.3, \ 27.2, \ 82.5]$

Forward: auxiliary data $r = 3$

- $v_1 = 8 \ \times 40 + 3 = 323$
  $z_1 = 323 \div 6 \ \ \ \ = \mathbf{53}\ R\ 5$

- $v_2 = 6 \ \times 50 + 5 = 305$
  $z_2 = 305 \div 11 \ \ \ = \mathbf{27}\ R\ 8$

- $v_3 = 11 \times 60 + 8 = 668$
  $z_3 = 668 \div 8 \ \ \ \ = \mathbf{83}\ R\ 4$

Forward output: $r = 4$, $\mathbf{z} = [53, 27, 83]$

Inverse: recovered x and $r = 3$

- $x_1 = 323 \div 8 \ \ \ \ = \mathbf{40}\ R\ 3$
  $v_1 = 6 \ \times 53 + 5 = 323$

- $x_2 = 305 \div 6 \ \ \ \ = \mathbf{50}\ R\ 5$
  $v_2 = 11 \times 27 + 8 = 305$

- $x_3 = 668 \div 11 \ \ \ = \mathbf{60}\ R\ 8$
  $v_3 = 8 \ \times 83 + 4 = 668$

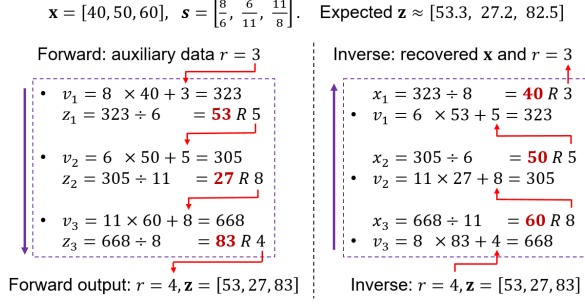**Inverse:** $r = 4$, $\mathbf{z} = [53, 27, 83]$

Figure 2. Illustration of how MAT works with Alg. 1. $a \div b = q\ R\ r$ denotes the division with remainder such that $a = b \cdot q + r$. For simplicity, the inputs are rescaled to integers as Line 1.

r, we give a novel numerical invertible affine transformation algorithm shown in Alg. 1. Since the numerical invertibility of the affine transformation is achieved by modular arithmetic, we refer to the induced numerical invertible affine transformation as *Modular Affine Transformation* (MAT).

The proof of inveribility of MAT is shown in Appendix C. Since the input to both forward and inverse MAT are k-precision floating numbers, they are converted to integers in Line 1. In Line 2-3, s_i is approximated with fraction m_{i-1}/m_i, and therefore $m_{d_b} = m_0$ as the volume preserving constraint holds. It is worth noting that round is the normal rounding operation except for outputting 1 for the input in $(0, 1]$ to avoid m_i being 0. Lines 6-9 are implemented by division with remainders. In Line 7, $r \in [0, m_{i-1})$, so we have $y_i = \lfloor (x_i \cdot m_{i-1} + r)/m_i \rfloor \approx s_i \cdot x_i$ and $x_i = \lfloor (y_i \cdot m_i + r)/m_{i-1} \rfloor \approx x_i/s_i (r \in [0, m_i))$. Formally, the error of computation of $\mathbf{z}_b$ with Alg. 1 is

$$|z_i - (s_i x_i + t_i)| = O(2^{-k}, 2^{-C}) \qquad (7)$$

and the detailed derivation of Eq. (7) is shown in Appendix D. A simple illustration on the correctness of the algorithm is shown in Fig. 2.

To reduce error, we set $C = 16$ (in Line 3). Note that an auxiliary value r is involved in the algorithm. When using the iVPF model, we can simply initialise $r = 0$, update r at each coupling layer and finally store r. When using the inverse iVPF model, the stored r is fed into the model and is updated at each inverse coupling layer. When the original data are restored, r is reduced to zero. As $r \in [0, 2^C)$, storing r just needs additional C bits, which is negligible in compressing high-dimensional data.

3.3.2 Numerical Invertible 1×1 Convolution Layer

Let $\mathbf{x}, \mathbf{z} \in \mathbb{R}^c$ be input and output of the convolution layer along the feature map at a certain pixel. Then we have $\mathbf{z} = \mathbf{Wx} = \mathbf{PL\Lambda Ux}$.

(1) Matrix multiplication with upper triangular matrix U. Denote by x_i and z_i the ith element of $\mathbf{x}$ and

$\mathbf{z}$ respectively. And let u_{ij} be the elements of $\mathbf{U}$. In iVPF, the forward mode matrix multiplication is $z_i = x_i + \lfloor \sum_{j=i+1}^c u_{ij} x_j \rfloor$, $i = 1, 2, ..., c - 1$ and $z_c = x_c$. And the inverse matrix multiplication is computed in an autoregressive manner in which $x_c, x_{c-1}, ..., x_1$ are computed recursively such that

$$x_c = z_c,$$
$$x_i = z_i - \lfloor \sum_{j=i+1}^c u_{ij} x_j \rfloor, \quad i = c - 1, ..., 1. \qquad (8)$$

(2) Matrix multiplication with lower triangular matrix U. Denote by l_{ij} the elements of $\mathbf{L}$. Then the forward mode matrix multiplication is $z_1 = x_1, z_i = x_i + \lfloor \sum_{j=1}^{i-1} l_{ij} x_j \rfloor$, $i = 2, ..., c$. And the inverse mode is performed by recursively computing $x_1, x_2, ..., x_c$ such that

$$x_1 = z_1,$$
$$x_i = z_i - \lfloor \sum_{j=1}^{i-1} l_{ij} x_j \rfloor, \quad i = 2, ..., c. \qquad (9)$$

(3) Matrix multiplication with $\mathbf{\Lambda}$. Denote λ_i as the i-th diagonal element of $\mathbf{\Lambda}$, we have $z_i = \lambda_i x_i$. It is clear that $|\prod_{i=1}^c \lambda_i| = 1$, thus we use MAT shown in Alg. 1.

(4) Matrix multiplication with P. As $\mathbf{P}$ is the permutation matrix, the elements in $\mathbf{x}$ can be directly permuted to $\mathbf{z}$, and $\mathbf{x}$ can be recovered by applying the inverse permutation $\mathbf{P}^{-1}$ on $\mathbf{z}$.

3.4. Error Analysis

Consider the continuous flow layer f_l and the corresponding iVPF layer $\bar{f}_l$. With the same input $\mathbf{y}_{l-1}$, for coupling layer, $|\bar{f}_l(\mathbf{y}_{l-1}) - f_l(\mathbf{y}_{l-1})| = O(2^{-k}, 2^{-C})$; for other layers, $|\bar{f}_l(\mathbf{y}_{l-1}) - f_l(\mathbf{y}_{l-1})| = O(2^{-k})$. If we further assume f is Lipschitz continuous, it holds that $|f_l(\bar{\mathbf{y}}_{l-1}) - f_l(\mathbf{y}_{l-1})| = O(|\bar{\mathbf{y}}_l - \mathbf{y}_l|)$. Thus for the volume preserving flow model with $\mathbf{y}_0 = \mathbf{x}, \mathbf{z} = \mathbf{y}_L, \mathbf{y}_l = f_l(\mathbf{y}_{l-1})$, and the corresponding iVPF model with $\bar{\mathbf{y}}_0 = \mathbf{x}, \bar{\mathbf{z}} = \bar{\mathbf{y}}_L, \bar{\mathbf{y}}_l = \bar{f}_l(\bar{\mathbf{y}}_{l-1})$, we have $|\bar{\mathbf{y}}_l - \mathbf{y}_l| = |\bar{\mathbf{y}}_{l-1} - \mathbf{y}_{l-1}| + O(2^{-k}, 2^{-C})$ and therefore $|\bar{\mathbf{z}} - \mathbf{z}| = O(L 2^{-k}, L 2^{-C})$. Detailed derivation can be found in the supplementary material. If $L, 2^{-k}, 2^{-C}$ are relatively small, the latents generated with the iVPF model are able to approximate the true distribution.

3.5. Bits-Back Coding with Dequantization

Consider the case when elements of the d-dimensional input data are all integers, i.e., $\mathbf{x}_0 \in \{0, 1, ..., 2^h - 1\}^d$ (for byte data or 8-bit images, $h = 8$). In practical applications, $\mathbf{x}_0$ should be pre-processed by normalisation and quantization such that $\mathbf{x} = \lfloor \mathbf{x}_0/2^h - 0.5 \rfloor$. To avoid information loss during quantization, it is set that $k \geq h$. After normalisation, the encoding process is (1) getting latent variable $\bar{\mathbf{z}} = \bar{f}(\mathbf{x})$ with iVPF model; (2) encoding $\bar{\mathbf{z}}$ with prior
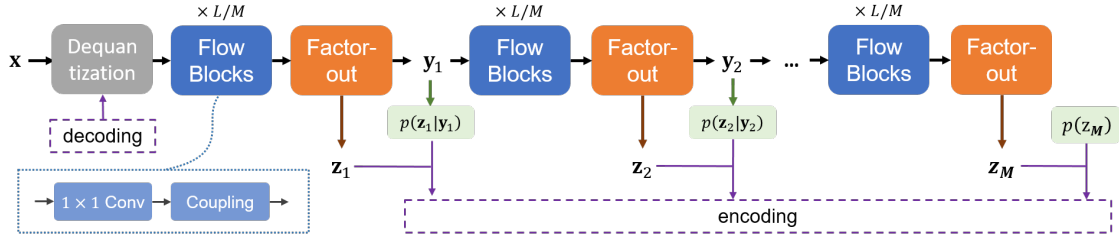
Figure 3. Illustration of the iVPF architecture. We just show the encoding phase with iVPF model and the decoding phase is performed inversely. The latents in the model are $\mathbf{z} = [\mathbf{z}_1, ..., \mathbf{z}_M]$ and the density function is $p(\mathbf{z}) = p(\mathbf{z}_M) \prod_{l=1}^{M-1} p(\mathbf{z}_l | \mathbf{y}_l)$.

Algorithm 2 Coding with iVPF

Procedure ENCODE($\mathbf{x}_0$)

1: Pre-process: $\mathbf{x} \leftarrow \lfloor \mathbf{x}_0 / 2^h - 0.5 \rceil$, set $r = 0$;
2: Decode $\mathbf{u} \sim U(0, 2^{-h})\delta$;
3: $\bar{\mathbf{x}} \leftarrow \mathbf{x} + \mathbf{u}$;
4: Compute $\bar{\mathbf{z}} \leftarrow \bar{f}(\bar{\mathbf{x}})$ with the iVPF model, in which r is updated with forward MAT;
5: Encode $\bar{\mathbf{z}} \sim p_Z(\bar{\mathbf{z}})\delta$; store r.

Procedure DECODE()

1: Decode $\bar{\mathbf{z}} \sim p_Z(\bar{\mathbf{z}})\delta$; get r.
2: Compute $\bar{\mathbf{x}} \leftarrow \bar{f}^{-1}(\bar{\mathbf{x}})$ with the inverse iVPF model, in which r is updated with inverse MAT;
3: $\mathbf{x} \leftarrow \lfloor 2^k \cdot \bar{\mathbf{x}} \rfloor / 2^h$, $\mathbf{u} \leftarrow \bar{\mathbf{x}} - \mathbf{x}$;
4: Encode $\mathbf{u} \sim U(0, 2^{-h})\delta$;
5: Post-process: $\mathbf{x}_0 \leftarrow 2^h \cdot (\mathbf{x} + 0.5)$;
6: **return** $\mathbf{x}_0$.

$P(\bar{\mathbf{z}}) \approx p_Z(\bar{\mathbf{z}})\delta$ using rANS coding algorithm [9]; and (3) saving auxiliary data generated by the iVPF model (see Sec. 3.3.1). The decoding process is (1) loading auxiliary data; (2) decoding $\bar{\mathbf{z}}$ with $P(\bar{\mathbf{z}}) \approx p_Z(\bar{\mathbf{z}})\delta$; and (3) recovering $\mathbf{x}$ with the inverse iVPF given $\bar{\mathbf{z}}$. In the encoding and decoding process, the volume of quantization bins is $\delta = 2^{-kd}$. As a result, the expected entropy coding length of $\mathbf{x}$ is approximately $-\log p_Z(\bar{\mathbf{z}}) + kd + C$, where $C = 16$ denotes the bit length of the auxiliary data (see Sec. 3.3.1).

However, the coding length with the above coding technique is significantly affected by the decimal precision k. If $k > h$, the coding length is dramatically increased with larger k. If $k(= h)$ is relatively small, the error between $\bar{\mathbf{z}} = \bar{f}(\mathbf{x})$ and the latents with continuous volume preserving flow $\mathbf{z} = f(\mathbf{x})$ are large. Then the density estimation of $p_X(\mathbf{x})$ will be far from the true distribution. In this case, the expected codelength will be large.

To resolve this issue, we adopt the idea of LBB [13] by dequantizing $\mathbf{x}$ to certain decimal precision k with the bitsback coding scheme. By using $k - h$ auxiliary bits, we can decode $\mathbf{u} \in \mathbb{R}^d$ from the uniform distribution $P(\mathbf{u}|\mathbf{x}) = U(0, 2^{-h})\delta^3$ with quantization bin size $\delta = 2^{-kd}$, and put

$\mathbf{x} + \mathbf{u}$ in the iVPF model such that $\bar{\mathbf{z}} = \bar{f}(\mathbf{x} + \mathbf{u})$. Then expected codelength of $\mathbf{x}$ is

$$
\begin{aligned}
&\mathbb{E}_{q(\mathbf{u}|\mathbf{x})}[-\log(p_X(\mathbf{x} + \mathbf{u})\delta) + \log(q(\mathbf{u}|\mathbf{x})\delta)] + C \\
&= \mathbb{E}_{q(\mathbf{u}|\mathbf{x})}[-\log p_Z(\bar{\mathbf{z}})] + hd + C
\end{aligned}
\tag{10}
$$

which is irrelevant with the decimal precision k. Thus we can set higher k to ensure accurate density estimation of $\mathbf{x}$.

The overall lossless compression algorithm with iVPF is summarised in Alg. 2.

4. Experiments

In this section, we perform extensive experiments to evaluate the effectiveness of lossless compression with iVPF model. We first introduce the continuous volume preserving flow architecture used for iVPF, and test the performance of iVPF on both low resolution images and real-world high resolution ones.

4.1. Flow Architecture

The architecture is a continuous flow derived from IDF [15] and IDF++ [3], which is shown in Fig. 3. The architecture consists of L stacked flow layer blocks, which are split into M levels.

The flow input is normalised to $[-0.5, 0.5]$. Each flow block is composition of volume preserving flow layers including Coupling $- 1 \times 1$ Conv. For Coupling layer, we follow [3] and split the input by $3 : 1$ where the dimension of $\mathbf{x}_b$ is $d_b = d/4$, while $1 : 1$ split reaches similar results. $\mathbf{s}(\mathbf{x}_a)$ and $\mathbf{t}(\mathbf{x}_a)$ are predicted with DenseNets [17], and the weights are shared except for the last layer. The dense blocks in the Densenet are the same as IDF++ in which Swish activation [16] and group norm [40] are used. Denote by $\mathbf{s}_0$ and $\mathbf{t}_0$ the outputs of the DenseNet, the final coupling coefficients are computed by $\mathbf{s}(\mathbf{x}_a) = \alpha[\tanh(\mathbf{s}_0) - \text{mean}(\tanh(\mathbf{s}_0))]$, $\mathbf{t}(\mathbf{x}_a) = \alpha \mathbf{t}_0$, where α is a learned parameter and mean denotes the mean of vector elements to make the flow volume-preserving. We initialise α to be zero so that the coupling layer starts with identity transformation (see Eq. (2)) for training [3]. $\tanh$ is used to constrain the

---

[3]Entries in $\mathbf{x}_0$ are integers, and $\mathbf{x} = \lfloor \mathbf{x}_0 / 2^h - 0.5 \rceil$ is discretized with bin size 2^{-h}.
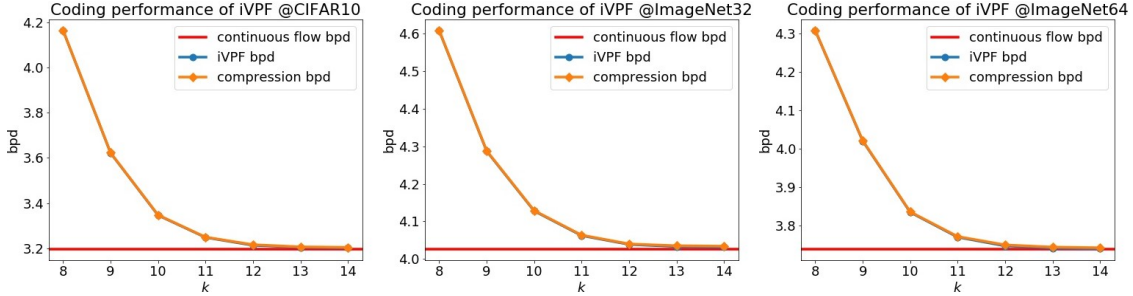
Figure 4. Compression performance of iVPF in terms of bpd. The negative log-likelihood (the lower bound of compression length) of continuous flow and the iVPF are also shown. Orange lines with diamond overlaps with the blue with circle ones as they are very close.

scale for stability. For 1×1Conv layer, the weights are decomposed to $\mathbf{W} = \mathbf{PLU}$. $\mathbf{P}$ is the permutation matrix which is initialised once and kept fixed; $\mathbf{L}$ and $\mathbf{U}$ are lower and upper triangular matrices respectively to learn.

At the end of each level, a factor-out layer is introduced, except for the final level. The architecture of factor-out layer is exactly the same as IDF++ [3].

In the training phase, the continuous flow is trained with variational dequantization to make the input data continuous [14]. To be specific, a dequantization noise $\mathbf{u}$ is sampled with $q(\mathbf{u}|\mathbf{x}) = U(0, 2^{-h})$ and the training objective is minimising $\mathbb{E}_{q(\mathbf{u}|\mathbf{x})}[-\log p_X(\mathbf{x}+\mathbf{u}) + \log q(\mathbf{u}|\mathbf{x})]$. After training, The iVPF model is derived from the continuous flow, in which the flow layer are computed with Sec. 3.3, and rANS [9] is adopted for encoding and decoding. Mindspore tool[4] is used for implementation.

4.2. Dataset and Training Details

Like general lossless compression algorithms [15, 3, 36, 13], we use CIFAR10, ImageNet32 and ImageNet64 for training and evaluation. For the CIFAR10 and ImageNet32 datasets, the number of flow blocks is $L = 24$ with $M = 3$ levels, while in ImageNet64, $L = 32$ and $M = 4$. The depth of Densenet is 12 in each coupling and factor-out layer. Adamax [20] is used for training, and the initial learning rate of each dataset is 2×10^{-3} and gradually decreasing after each epoch. We train approximately 500 epochs for CIFAR10, 70 epochs for ImageNet32 and 20 epochs for ImageNet64. For better convergence, we adopt exponential moving average with decay rate 0.9995 on model weights to get ensembled model for evaluation.

To test the generation performance, we evaluate the above datasets on ImageNet32 trained model, in which ImageNet64 is split into four 32×32 patches. We also evaluate compression on natural high-resolution image datasets including CLIC.mobile, CLIC.pro[5] and DIV2k [1]. These images are cropped to 32×32 patches and feed into the Ima-

---

[4] https://www.mindspore.cn/
[5] https://www.compression.cc/challenge/

Table 1. Coding performance of LBB and iVPF on CIFAR10 in terms of bits per dimension and coding efficiency. $k = 14$ is used in iVPF. Infer time denotes the inference time with batch size 256.

	bpd	aux. bits	time (ms) infer	coding	# coding
LBB [13]	**3.12**	39.86	13.8	97	188
iVPF (Ours)	3.20	**6.00**	**6.1**	**6.6**	**2**

geNet32 trained model for evaluation. We also feed 64×64 patches to ImageNet64 trained model for comparison.

The evaluation protocol is the average bits per dimension (bpd), which can be derived with the negative log-likelihood divided by the dimension of the inputs. For no compression, the bpd is 8. We also investigate the coding efficiency of the proposed algorithm.

4.3. Compression Performance

In this subsection, we investigate the compression performance of the proposed iVPF model on different datasets and different settings. We first evaluate the compression performance in terms of bits per dimension (bpd) under different quantization precision k. The analytical log-likelihood of continuous volume preserving flow and iVPF model are also reported for comparison. Fig. 4 shows the results. It can be seen that the compression codelength is very close to the analytical log-likelihood of the original volume preserving flow model. To be specific, in CIFAR10, when $k = 14$, the difference of bpd between the iVPF model and the continuous volume preserving flow is 0.005, which is the bpd of the auxiliary data introdued in Alg. 1 ($16/3072 \approx 0.005$). While the final entropy coding only introduces another 0.003 bpd compared with the iVPF model. Similar conclusions can be arrived on ImageNet32 and ImageNet64 dataset. We therefore set $k = 14$ in the rest of the section.

As compression with iVPF only involves a single decoding and encoding process, model inference contributes the main latency of coding, which is expected to be effective. Table 1 conveys that compression with iVPF can be per-

Table 2. Compression performance in bits per dimension (bpd) on benchmarking datasets. [†] denote the generation performance in which the models are trained on ImageNet32 and tested on other datasets.

	ImageNet32	ImageNet64	CIFAR10
PNG [4]	6.39	5.71	5.87
FLIF [33]	4.52	4.19	4.19
L3C [26]	4.76	-	-
Bit-Swap [23]	4.50	-	3.82
HiLLoC [36][†]	4.20	3.90	3.56
LBB [13]	3.88	3.70	3.12
IDF [15]	4.18	3.90	3.34
IDF [15][†]	4.18	3.94	3.60
IDF++ [3]	4.12	3.81	3.26
iVPF (Ours)	4.03	3.75	3.20
iVPF (Ours)[†]	4.03	3.79	3.49

Table 3. Compression performance in bpd of different algorithms on real-world high resolution images. [‡] denotes compression with ImageNet64 trained model.

	CLIC.mobile	CLIC.pro	DIV2k
PNG [4]	3.90	4.00	3.09
FLIF [33]	2.49	2.78	2.91
L3C [26]	2.64	2.94	3.09
RC [27]	2.54	2.93	3.08
iVPF (Ours)	**2.47**	**2.63**	**2.77**
iVPF (Ours)[‡]	**2.39**	**2.54**	**2.68**

formed in milliseconds. More discussions on the coding time can be found in the supplementary material. On the other hand, as the coding scheme has to sequentially encode each dimension of data, the latency of coding is negligible when the quantity of coding is large. LBB [13] is such an algorithm with large quantity of coding operations, and our re-implementation on LBB implies that coding takes about 90% of the compression latency. Table 1 shows the coding performance between LBB and iVPF on CIFAR10 in terms of bpd, auxiliary bits used per dimension and the number of times for coding. Although LBB achieves slightly lower bpd, it suffers from large quantity of coding. Moreover, our algorithm could significantly reduce the auxiliary bits used[6]. As a result, compared to LBB, our algorithm is competitive with notable space/time efficiency and limited compression ratio loss.

4.4. Comparison with State-of-the-Art

To show the competitiveness of our proposed iVPF model, we compare its performance with state-of-the-art learning based lossless compression methods and represen-

---

[6]The quantization precision of the original byte data is $h = 8$. We use 6 auxiliary bits for dequantization to make the precision $k = 14$.

tative conventional lossless coders, i.e., PNG [4] and FLIF [33], on small and large scale datasets.

Experiments on Low Resolution Images. We compare iVPF with L3C [26], Bit-Swap [23] and HiLLoC [36], which represent cutting-edge performance of non-flow based latent variable models. Moreover, we compare the results with a recent advance on lossless compression with continuous flows including LBB [13], IDF [15] and IDF++ [3]. The results are shown with bpd to encode one image averaged on the datasets, where lower bpd shows the advantage of the method. From Table 2, one can see that our proposed method outperforms its predecessors, i.e., IDF and IDF++, which validates the improvement of our method in the line of integer flows. Although LBB preforms slightly better than iVPF, our method is competitive as the former suffers from large number of auxiliary bits and low coding efficiency, as discussed in Sec. 4.3.

Generation. We test the generation performance of iVPF, with HiLLoc [36], IDF [15] and LBB [13] for compression. The results is shown in Table 2. It can be seen that most compression algorithms achieves desirable generation performance, and ours achieves the best performance under all generation experiments.

Experiments on High Resolution Images. We compare our method with representative practical image lossless compression algorithms, i.e., L3C [26] and RC [27]. We use patch-based evaluation in which images are cropped to 32×32 for ImageNet32 model and 64×64 for ImageNet64 model. From Table 3, one can observe that for high resolution images, conventional coders, e.g., FLIF, still have advantage over learning based methods, while they tend to underperform learning based methods on small scale images. Nevertheless, our proposed iVPF outperform all the benchmarks shown for high resolution images and thus validates its competitiveness.

5. Conclusions

In this paper, we present iVPF, the Numerical Invertible Volume Preserving Flow Model for efficient lossless compression. We investigate the potential of volume preserving flow for efficient lossless compression. Then we introduce iVPF with novel numeral computing algorithms to the flow model to achieve exact bijective mapping between data and latent variables, so that efficient lossless compression is available by directly coding the latents. The novel lossless compression algorithm based on iVPF is proposed by combining the bits-back coding scheme. Experiments show that the compression codelength reaches the theoretical lower bound of negative log-likelihood. The proposed model achieves high compression ratio with desired coding efficiency on image datasets with both low and high resolution images.

References

[1] Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 126–135, 2017.

[2] Jens Behrmann, Paul Vicol, Kuan-Chieh Wang, Roger Grosse, and Jörn-Henrik Jacobsen. Understanding and mitigating exploding inverses in invertible neural networks. *arXiv preprint arXiv:2006.09347*, 2020.

[3] Rianne van den Berg, Alexey A Gritsenko, Mostafa Dehghani, Casper Kaae Sønderby, and Tim Salimans. Idf++: Analyzing and improving integer discrete flows for lossless compression. *arXiv preprint arXiv:2006.12459*, 2020.

[4] Thomas Boutell and T Lane. Png (portable network graphics) specification version 1.0. *Network Working Group*, pages 1–102, 1997.

[5] Sheng Cao, Chao-Yuan Wu, and Philipp Krähenbühl. Lossless image compression through super-resolution. *arXiv preprint arXiv:2004.02872*, 2020.

[6] Yann Collet and Chip Turner. Smaller and faster data compression with zstandard. *Facebook Code [online]*, 2016.

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

[8] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.

[9] Jarek Duda. Asymmetric numeral systems: entropy coding combining speed of huffman coding with compression rate of arithmetic coding. *arXiv preprint arXiv:1311.2540*, 2013.

[10] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[11] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.

[12] GE Hinton and Drew van Camp. Keeping neural networks simple by minimising the description length of weights. 1993. In *Proceedings of COLT-93*, pages 5–13.

[13] Jonathan Ho, Evan Lohn, and Pieter Abbeel. Compression with flows via local bits-back coding. In *Advances in Neural Information Processing Systems*, pages 3879–3888, 2019.

[14] Emiel Hoogeboom, Taco S Cohen, and Jakub M Tomczak. Learning discrete distributions by dequantization. *arXiv preprint arXiv:2001.11235*, 2020.

[15] Emiel Hoogeboom, Jorn Peters, Rianne van den Berg, and Max Welling. Integer discrete flows and lossless compression. In *Advances in Neural Information Processing Systems*, pages 12134–12144, 2019.

[16] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[17] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[18] David A Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952.

[19] Nal Kalchbrenner, Aäron Oord, Karen Simonyan, Ivo Danihelka, Oriol Vinyals, Alex Graves, and Koray Kavukcuoglu. Video pixel networks. In *International Conference on Machine Learning*, pages 1771–1779. PMLR, 2017.

[20] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[21] Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in neural information processing systems*, pages 10215–10224, 2018.

[22] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[23] Friso H Kingma, Pieter Abbeel, and Jonathan Ho. Bit-swap: Recursive bits-back coding for lossless compression with hierarchical latent variables. *arXiv preprint arXiv:1905.06845*, 2019.

[24] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A flow-based generative model for video. *arXiv preprint arXiv:1903.01434*, 2(5), 2019.

[25] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[26] Fabian Mentzer, Eirikur Agustsson, Michael Tschannen, Radu Timofte, and Luc Van Gool. Practical full resolution learned lossless image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10629–10638, 2019.

[27] Fabian Mentzer, Luc Van Gool, and Michael Tschannen. Learning better lossless compression using lossy compression. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6638–6647, 2020.

[28] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[29] Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 3617–3621. IEEE, 2019.

[30] Majid Rabbani. Jpeg2000: Image compression fundamentals, standards and practice. *Journal of Electronic Imaging*, 11(2):286, 2002.

[31] Greg Roelofs and Richard Koman. *PNG: the definitive guide*. O'Reilly & Associates, Inc., 1999.

[32] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[33] Jon Sneyers and Pieter Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 66–70. IEEE, 2016.

[34] Peter Sorrenson, Carsten Rother, and Ullrich Köthe. Disentanglement by nonlinear ica with general incompressible-flow networks (gin). *arXiv preprint arXiv:2001.04872*, 2020.

[35] James Townsend, Tom Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. *arXiv preprint arXiv:1901.04866*, 2019.

[36] James Townsend, Thomas Bird, Julius Kunze, and David Barber. Hilloc: Lossless image compression with hierarchical latent variable models. *arXiv preprint arXiv:1912.09953*, 2019.

[37] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in neural information processing systems*, pages 4790–4798, 2016.

[38] Chris S Wallace. Classification by minimum-message-length inference. In *International Conference on Computing and Information*, pages 72–81. Springer, 1990.

[39] Ian H Witten, Radford M Neal, and John G Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987.

[40] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.