Enhancing and Evaluating Probabilistic Circuits for High-Resolution Lossless Image Compression

Daniel Severo*, Jingtong Su†, Anji Liu§, Jeff Johnson*, Brian Karrer*
Guy Van den Broeck§, Matthew J. Muckley*, Karen Ullrich*

*Meta AI {dsevero,jhj,briankarrer,mmuckley,karenu}@meta.com
†NYU js12196@nyu.edu
§UCLA {liuanji,guyvdb}@cs.ucla.edu

Abstract

We propose a set of modifications that improve training time and likelihood estimation of hierarchical mixture models implemented via Probabilistic Circuits (PCs). Our proposal reduces the complexity of mutual information estimation in the structure learning step of PCs from quadratic to linear in the number of inputs, without sacrificing likelihood estimation performance on image datasets. We repurpose invertible transformations from the lossless compression community to improve likelihood estimation by a factor of up to 25% on benchmark image datasets, making PCs competitive with current standard codecs on low-resolution datasets. Despite our improvements, experiments with low- and high-resolution image datasets indicate that the advantage of lossless neural compression and PCs over standard codecs, such as WebP, disappears as the image size increases, motivating future work on practical lossless neural compression.

1 Introduction

In recent years, the machine learning community has brought forward *learnable* discrete density estimators to improve compression rates [11, 27, 25, 26, 10] by learning from data rather than using hand-crafted approaches. However, these rate improvements come at a significant increase in computational complexity when compared to traditional compression codecs.

Probabilistic circuits (PCs) is a model class implementing mixture models that is compute constrained by design, allowing efficient marginalisation over input data. In [14], the authors show this feature allows for efficient encoding and decoding when used together with an entropy coder such as asymmetric numeral systems [8]. In the same work, the authors show the complexities of both encoding and decoding can be reduced from linear to sub-linear in the sequence length, and provide empirical evidence indicating speed enhancements of 5-40 times over other neural models.

In this paper, we build on prior work on PCs [1, 14] to improve scalability, parameter efficiency, and inference speed:

- We demonstrate that **invertible transforms**, commonly found in generic compression schemes, can significantly improve compression rates for PCs with a negligible increase in computational overhead.

- We introduce **local mutual information estimation** for structured sources. Our proposal speeds up the process of learning the graphical model for images from quadratic to linear in the input dimension, and hence enables us to scale to larger models.

- We provide **extensive experiments** ablating the effects of batch and patch size on training of probabilistic circuits, as well as compression results on a recently introduced dataset of high-resolution images for AR/VR applications [28].

Despite our improvements, we provide evidence that lossless neural compressors such as IDF/IDF++ [11, 27] and Bits-back Coding [24, 25] have diminishing gains when evaluated on higher-resolution, real-world, image datasets. In some cases neural compressors even under-perform when compared to standard codecs (e.g., WebP [9]), which are significantly faster, do not require training, and have significantly low memory footprint, motivating future research.

2 Background

Probabilistic Circuits (PCs) [1] define a broad family of tractable probabilistic models, including the earliest proposal of arithmetic circuits (ACs) [5] and Sum-Product Networks (SPNs) [19]. A PC represents a mixture distribution over random variables $X^n := (X_1, \ldots, X_n)$ via a parametrized directed acyclic graph (DAG) with a single root node. The DAG comprises three kinds of nodes: input (leaves), sum, and product. Each node n defines a probability distribution p_n, recursively, as follows [15]:

$$p_n(x) := \begin{cases} f_n(x) & \text{if } n \text{ is an input unit,} \\ \sum_{c \in \text{in}(x)} \theta_{n,c} \cdot p_c(x) & \text{if } n \text{ is a sum unit,} \\ \prod_{c \in \text{in}(x)} p_c(x) & \text{if } n \text{ is a product unit.} \end{cases}$$

where f_n is a univariate input distribution (e.g., boolean, categorical, or discrete logistic), $\theta_{n,c}$ is the mixture component corresponding to edge (n, c), and $\text{in}(x)$ is the set of children nodes of x.

Learning a PC from data is usually a 2-step process: first, the structure of the DAG is learned, which is followed by the estimation of the parameters $\theta_{n,c}$. Parameter estimation follows the same procedures as traditional neural network training (e.g., SGD, train/test/validation splits).

Computing the probability $p_n(x)$, of some data point x, can be done similar to a forward-pass in a neural network, which can benefit from hardware accelerators such as GPUs. Note that the computational complexity of a forward-pass in a PC, as well as most neural networks, depends only on the structure of the DAG, and not the values of parameters $\theta_{n,c}$.

The PCs used in this work are latent variable models constructed from Chow-Liu Trees [2] (CLT) known as Hidden CLTs (HCLT) [14]. For $X^n \sim P_{X^n}$, a CLT

is a probabilistic graphical model (PGM) defined by the maximum spanning tree of the graph with nodes $\{X_1, \ldots, X_n\}$ and edge weights the mutual information [3] between two endpoint variables. The resulting PGM is a directed tree in the observation space (each node has exactly one parent), implying the model Q_{X^n} factorizes into a product of first-order (i.e. Markov) conditionals $Q_{X_i \mid X_{\sigma(i)}}$, where $\sigma(i)$ is the parent of i. If the conditionals are computed from the true data distribution, i.e., $Q_{X_i \mid X_{\sigma(i)}} = P_{X_i, X_{\sigma(i)}} / P_{X_{\sigma(i)}}$, then the resulting model minimizes the KL divergence, $\mathrm{KL}[P_{X^n} || Q_{X^n}]$, over all models Q_{X^n} with first-order Markov factorization [13].

An HCLT is a PGM, constructed by adding latent variable nodes to the PGM of some CLT, where each observed node is assigned a corresponding (discrete) latent variable with K categories, denoted as the latent size of the HCLT. Each observation is conditionally independent from the rest given its latent variable. HCLTs allow for efficient marginalization, as well as computing of conditionals, which can be leveraged for efficient lossless compression [1].

3 Improving Training and Efficiency for PCs

This section discusses 2 modifications to PCs which together improve compression performance while enabling scaling to large images.

3.1 Local MI Estimation

Learning the structure of an HCLT requires estimating the mutual information (MI) of all pairs (X_i, X_j) for $1 \leq i < j < n$. Formally, let $M \in \mathbb{R}^{n \times n}$ be a matrix with elements $M_{ij} \geq 0$ being an estimate of the MI $I(X_i; X_j)$ computed from the dataset D of images. In [14], MI is estimated from the empirical distribution over a single channel-element of a pixel. For $x, x' \in [256) := \{0, \ldots, 255\}$ and $d \in [256)^n$ a flattened image in the dataset, define the following empirical distributions

$$P_i(x) := \frac{1}{|D|} \cdot \sum_{d \in D} \mathbf{1}\{d_i = x\} \qquad P_{ij}(x, x') := \frac{1}{|D|} \cdot \sum_{d \in D} \mathbf{1}\{d_i = x\} \cdot \mathbf{1}\{d_j = x'\}, \quad (1)$$

which count the marginal occurrences of x and x', as well as the joint occurrence, in the dataset. An estimate for MI is given by

$$M_{ij} = \sum_{x \in [256)} \sum_{x' \in [256)} P_{ij}(x, x') \cdot \log_2 \left(\frac{P_{ij}(x, x')}{P_i(x) \cdot P_j(x')} \right). \quad (2)$$

Estimating MI for all pairs scales quadratically with n, the number of input variables, due to the double-summation.

The DAG of the HCLT [14], discussed in Section 2, is constructed by computing the maximum-spanning tree (MST) of the graph with weighted-adjacency matrix equal to M. The MST graph can be found through a greedy procedure known as Kruskal's algorithm [12] that visits edges, in the order of highest to lowest weight, and adds them to the MST graph as long as the addition does not result in the formation of a cycle (the absence of cycles guarantees the graph will be a tree at every step).

Pairs (X_i, X_j) that have small MI are unlikely to be added to the graph, as they will be visited last. For images, we expect the MI to correlate with the distance L1 in the coordinate space, as pixels tend to be more similar to their neighbors. We can therefore replace the second summation in Equation (2) for a summation over a small neighborhood of x, which avoids computing most elements in M_{ij}, which are then set to 0. This reduces structure learning complexity from quadratic to linear in n.

As mentioned in Section 2, the computational complexitey of the PC is determined solely by its structure (i.e., DAG). Our ablations indicate that defining the neighborhood to be all elements within an L1 distance of 3 was enough to learn *exactly* the same HCLT structure as the unconstrained case across all datasets, hyperparameters, and random seeds. This implies our method does not alter the complexity of computing the probabilities $p_n(x)$. Furthermore, the parameter learning step is not affected in any way, as it is agnostic to how the structure was learned.

3.2 Invertible Transforms

We applied invertible transformations to the input of PCs to decorrelate the input pixels and improve likelihood estimation. The transforms selected are inspired by those employed by standard codecs such as WebP [9].

WebP [9] offers a wide range of all 3 types of decorrelation functions. The simplest option, which we adopted, is to transform the red (R) and blue (B) channels according to the value of the green (G) channel, which is kept fixed. This is done by subtracting the green channel from the remaining 2 (Subtract-Green transform), which is easily reversible by adding back the green channel during decoding time.

Spatial decorrelation is performed via the transform described next, which is similar to the Delta transform [3]. For an image of height H, width W, and C channels, let $x_i \in [0, d-1]$ (typically, $d = 256$) be a single channel element of a pixel. The image is assumed to be flattened using some procedure known both to the encoder and decoder (i.e., in raster-scan or row-major order) such that pixels are ordered $x_1, \ldots, x_{H \times W \times C}$. The transformed elements y_i are defined as

$$y_i = x_i - x_{i-1} \text{ for } i > 0, \text{ and } y_0 = x_0. \tag{3}$$

Transmitting the first pixel (top-left, $i = 0$) unchanged guarantees this transform is lossless,

$$x_i = \sum_{j=0}^{i} y_j \text{ for } i > 0, \text{ and } x_0 = y_0. \tag{4}$$

Finally, lossless image compression precludes modelling in the YCbCr colorspace as the transform between RGB and YCbCr is lossy. Instead, we adopt a *lossless* image color transform, YCoCg-R [18], which has similar properties to YCbCr but is lossless with respect to RGB.

4 Experiments

In this section we experimentally validate the modifications to PCs discussed in Section 3 through a series of ablations, as well as compare it to baseline standard

codecs and recently introduced neural codecs.

We experimented on well-known benchmark datasets of low-resolution images, CIFAR-10 [23] and ImageNet [6], as well as high-resolution datasets, CLIC [22] and (the recently introduced) Multiface (MF) [28] AR/VR dataset.

4.1 Models and Training Details

We benchmark PCs against standard codecs (PNG [7] and WebP [9]) as well as neural codecs HiLLoC [25] (i.e., an implemention of bits-back coding [26]) and integer discrete flow [11, 27]. For PCs, We used the hidden chow-liu tree (HCLT) model present in [14]. The improvements discussed in Section 3, i.e., invertible transforms and local MI estimation, were applied to HCLTs to create an additional model which we named HCLT++.

For PC results on CLIC and MF the learning rate and batch size were fixed to 0.035, after an initial sweep between 10^{-5} and 1, with batch size $2^{14} = 16,384$; while batch size was fixed to 128 and learning rate sweeped from 0.1 to 0.01 for CIFAR, IM32, and IM64. We experimented changing colorspaces from RGB to YCoCg-R [18] and found no significant change in performance for neural compressors.

4.2 Computational Requirements

The computational resource required to run HCLT++ is very close to that of HCLT, as the invertible transformations are significantly faster than performing the forward pass in the network. Local MI estimation does not affect the compute needed for HCLT++, as computing the likelihood in a PC depends only on its structure (see Section 2).

As expected, standard codecs on CPU are orders of magnitude faster than both PCs and Neural Codecs on GPU. For example, on the largest dataset, Multiface, WebP with YCoCg encodes at 43.8 Mbps, while the fastest IDF++ implementation reaches only 13.8 kbps on an NVIDIA Tesla V100 GPU. For PCs, reported throughputs are roughly 20x that of IDF [14], which is still significantly slower than standard codecs.

4.3 Main Results

Compression results across all models and datasets are shown in Table 1. As expected, IDF++ [27] achieves the lowest bitrate across all datasets, including large datasets where patch-level compression must be used due to limited memory constraints. Probabilistic circuit methods achieve lower bitrates than the best traditional codec (WebP) on all datasets other than Multiface, but we note that WebP requires far less compute to achieve these compression ratios.

The performance of neural compression methods diminishes significantly when moving from low-to-high resolution datasets. On CIFAR, IDF++ achieves a bitrate 29.2% lower than WebP, while HCLT++ achieves a bitrate 10% lower. However, on CLIC the IDF++ bitrate is only 5.8% better than WebP, while the HCLT++ bitrate is 4.2% better, at the expense of a large increase in both compute and memory resources. This illustrates a property of traditional compressors where they perform

Table 1: PC results compared to standard codecs and neural compressors. PCs provide a gain in compression performance relative to standard codecs, while not requiring as much compute as neural compressors. The advantage of both PCs and neural codecs diminishes as the average image size of the dataset increases (left-to-right). All units are bits-per-dimension.

		CIFAR	IM32	IM64	CLIC	MF
Standard Codecs	PNG(RGB)	5.87	6.05	5.34	3.49	2.87
	PNG(YCoCg)	5.23	5.54	4.88	3.13	3.01
	WebP(YCoCg)	4.87	5.20	4.51	2.68	2.66
	WebP(RGB)	**4.61**	**4.98**	**4.30**	**2.59**	**2.61**
PCs	HCLT	6.04	6.16	5.92	4.10	3.12
	HCLT++	**4.13**	**4.72**	**4.29**	**2.48**	**2.75**
Neural Codecs	HiLLoC	3.56	4.20	3.90	2.63	-
	IDF	3.32	3.95	3.66	2.43	2.57
	IDF++	**3.26**	**3.94**	**3.62**	**2.44**	**2.54**
Adv. of best Prob. Circuit over WebP		10.8%	5.8%	1.0%	4.0%	-5.4%
Adv. of best Neural Codec over WebP		29.2%	20.8%	15.9%	6.1%	2.7%

Table 2: Lossless compression results for our method (HCLT++) and the HCLTs of [14]. All units are bits-per-dimension.

	CIFAR		IM32		IM64		CLIC		MF	
# H. dim.	HCLT	HCLT++	HCLT	HCLT++	HCLT	HCLT++	HCLT	HCLT++	HCLT	HCLT++
1	7.89	**4.93**	7.89	**5.27**	7.91	**4.93**	7.94	**3.42**	5.81	**3.40**
8	6.39	**4.22**	6.47	**4.73**	6.28	**4.31**	5.67	**2.71**	3.52	**2.81**
16	6.14	**4.12**	6.25	**4.69**	6.02	**4.26**	5.09	**2.68**	3.17	**2.78**
32	6.04	**4.11**	6.16	**4.69**	5.92	**4.26**	4.65	**2.56**	3.12	**2.77**
64	6.04	**4.12**	6.17	**4.71**	5.93	**4.27**	4.41	**2.51**	3.12	**2.76**
128	6.08	**4.13**	6.24	**4.72**	6.01	**4.29**	4.10	**2.48**	3.14	**2.75**

far better on high-resolution data than low-resolution data. We discuss this property further in Section 5.

4.4 HCLTs with Invertible Transforms (HCLT++)

Table 2 shows the performance gained from HCLT++: it outperforms the HCLT on all datasets for all hidden dimension sizes.[1] The difference in performance between the two methods slightly diminishes as the hidden dimension size grows. We experimented changing the order of the two operations that define HCLT++ (i.e., color and delta transforms), as well as interleaving them with the YCoCg-R color transform, but found no difference in performance.

To understand better how each transform affects the overall performance, we ablated each transform (Subtract-Green and Delta) on the CLIC. We focused the ablation on the CLIC dataset as it is the closest dataset representing to real-world high-resolution images. The hidden dimension size was fixed to 128 to match that of Table 1. Adding Subtract-Green to the HCLT reduces the bits-per-dimension (bpd)

[1]Prior work [16, 17] reports PC likelihoods on natural image datasets such as CIFAR, IM32, and IM64 assuming that the data is given directly in the YCbCr or the YCoCg space without accounting for the lossy conversion from RGB. Hence, those numbers are not directly comparable to our likelihoods in the RGB space.

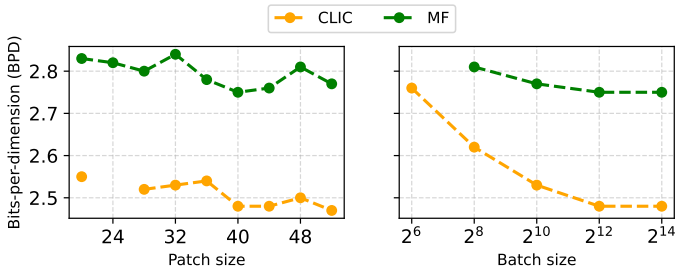

Figure 1: **Left:** Compressing performance (lower is better) of probabilistic circuits for varying patch sizes. Increasing patch size beyond a certain limit increases BPD for probabilistic circuits. **Right:** Compression performance on the test set improves consistently with larger batch sizes for probabilistic circuits.

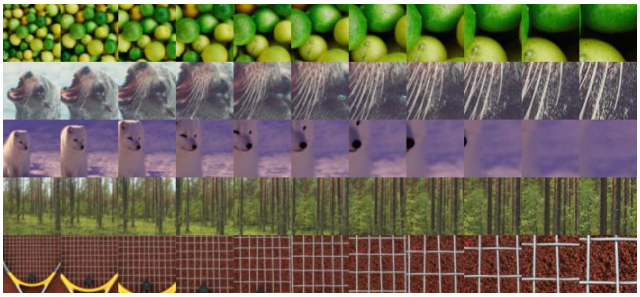

Figure 2: Pixels are more similar to their neighbors as images increase in size. See the text for details and discussions.

from 4.10 to 2.98, while adding Delta (without Subtract-Green) reduces bpd to 2.65. Using both transforms gives further gains: 2.48 bpd.

4.5 Scaling Batch and Patch Sizes

The right plot of Figure 1 shows results, on the test set, when increasing batch size during training of probabilistic circuits with the EM algorithm. Large batch sizes were implemented by accumulating EM computations, similar to how gradient accumulation is performed, before updating the parameters. Larger batch sizes consistently improve generalization to the test set, but plateau for both types of data between 2^{12} and 2^{14}.

The left plot in Figure 1 contains results for patch size ablations. Images were partitioned into non-overlapping patches with 0-padding at the borders. An HCLT probabilistic circuit with 128 hidden dimensions was trained for each patch size for 150 epochs. Increasing patch size beyond 24 gave a small performance boost peeking at 40. This pattern repeated for any number of hidden dimensions, learning rates, or batch sizes.

5 Limitations of Lossless Neural Compressors

Table 1 shows the compression performance of standard codecs, probabilistic circuits, and neural codecs, across 5 datasets. Models are ordered, from top-to-bottom, by

both compute and memory usage, with standard codecs being least intensive. Neural codecs, such as IDF and bits-back, can significantly outperform standard codecs on low-resolution datasets, but at a significant increase in compute and memory resources compared to WebP or PNG [14]. Probabilistic circuits (PC) are a candidate for a good middle ground between standard and neural codecs.

WebP in RGB space outperformed PNG on all datasets. Results for probabilistic circuits improve as the number of hidden latent dimensions grows, as expected, but eventually plateau. IDF/IDF++ achieved the best results across all datasets but are significantly more resource intensive than WebP. The performance across all algorithms improves as image size increases.

WebP improves significantly faster and eventually performs on par with, or better, than probabilistic circuits. Average image size increases across columns from left-to-right in Table 1 as the gap between the best neural and standard codec diminishes (last row). There are many conjectures as to why the performance of standard codecs improves drastically, of which we discuss two. First, pixels are more similar to their neighbours as the image size increases, improving the performance of the simple prediction algorithms used by standard codecs. This is illustrated in Figure 2 on patches of 5 images taken from the CLIC dataset [22]. The center of each patch coincides with the center of the image they were taken from. Images grow in size from left to right but patch size remains constant at 64x64, which corresponds intuitively to a "zoom in" effect, reducing variability drastically. Finally, standard codecs employ dictionary coding methods from the Lempel-Ziv (LZ) family [29]. Increasing the image size is equivalent to increasing the number of samples, as modelling happens at the pixel-level. The compression rate of the LZ family is known to approach the entropy rate of any stationary and ergodic source [3], and have been empirically observed to perform well in practice [21], when given access to enough samples.

6 Conclusion

In this work we provided improvements to training and likelihood estimation with probabilistic circuits (PCs). Repurposing invertible transforms from standard codecs gives a significant boost in performance for PCs. This suggests more research needs to be done at the input layer of PCs to increase the expressivity of these models while still guaranteeing lossless decodability.

Our findings indicate that the performance gain observed for lossless neural compression on small-resolution datasets does not simply transfer to the high-resolution setting. This result, coupled with the large increase in compute when going from standard to neural codecs, highlights the limited practicality of current lossless neural compression models based on likelihood estimation.

We conclude with some remarks and potential research directions. While this work brings probabilistic circuits closer to being competitive for lossless compression, the amount of computational and memory resources needed, compared to standard codecs, is high for many applications. One direction to reducing these costs is pruning based solutions such as those presented in [4]. Pruning addresses both computational and memory concerns by removing computational units from the PCs.

Finally, it is possible to go beyond the i.i.d. setting and target lower entropy rates by, for example, modelling batches of data where patches are not treated as i.i.d. observations. Many real-world applications have easily-accessible side information [3, 20], available at both encoder and decoder, which can be used to condition models and potentially improve compression rates. The gain seen from using invertible transforms with probabilistic circuits suggests incorporating sub-routines of standard compression algorithms into neural codecs might be a way to reduce compute while maintaining performance.

References

[1] Y Choi, Antonio Vergari, and Guy Van den Broeck. Probabilistic circuits: A unifying framework for tractable probabilistic models. *UCLA. URL: http://starai. cs. ucla. edu/papers/ProbCirc20. pdf*, 2020.

[2] CKCN Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[3] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[4] Meihua Dang, Anji Liu, and Guy Van den Broeck. Sparse probabilistic circuits via pruning and growing. *Advances in Neural Information Processing Systems*, 35:28374–28385, 2022.

[5] Adnan Darwiche. A differential approach to inference in bayesian networks. *Journal of the ACM (JACM)*, 50(3):280–305, 2003.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] David Duce. Portable network graphics (png) specification (second edition). World Wide Web Consortium, Recommendation REC-PNG-20031110, November 2003.

[8] Jarek Duda. Asymmetric numeral systems. *arXiv preprint arXiv:0902.0271*, 2009.

[9] Google Developers. Webp lossless bitstream specification. `https://developers.google.com/speed/webp/docs/webp_lossless_bitstream_specification`, 2023. Accessed: 2024-02-21.

[10] Jonathan Ho, Evan Lohn, and Pieter Abbeel. Compression with flows via local bits-back coding. *Advances in Neural Information Processing Systems*, 32, 2019.

[11] Emiel Hoogeboom, Jorn Peters, Rianne Van Den Berg, and Max Welling. Integer discrete flows and lossless compression. *Advances in Neural Information Processing Systems*, 32, 2019.

[12] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[13] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[14] Anji Liu, Stephan Mandt, and Guy Van den Broeck. Lossless compression with probabilistic circuits. *arXiv preprint arXiv:2111.11632*, 2021.

[15] Anji Liu and Guy Van den Broeck. Tractable regularization of probabilistic circuits. *Advances in Neural Information Processing Systems*, 34:3558–3570, 2021.

[16] Anji Liu, Honghua Zhang, and Guy Van den Broeck. Scaling up probabilistic circuits by latent variable distillation. In *The Eleventh International Conference on Learning Representations*, 2022.

[17] Xuejie Liu, Anji Liu, Guy Van den Broeck, and Yitao Liang. Understanding the distillation process from deep generative models to tractable probabilistic circuits. In *International Conference on Machine Learning*, pages 21825–21838. PMLR, 2023.

[18] Henrique Malvar and Gary Sullivan. Ycocg-r: A color space with rgb reversibility and low dynamic range. *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q*, 6, 2003.

[19] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[20] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[21] Christian Steinruecken. *Lossless Data Compression*. PhD thesis, University of Cambridge, 2014.

[22] George Toderici, Lucas Theis, Nick Johnston, Eirikur Agustsson, Fabian Mentzer, Johannes Ballé, Wenzhe Shi, and Radu Timofte. CLIC 2020: Challenge on learned image compression, 2020.

[23] Antonio Torralba, Rob Fergus, and William T Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 30(11):1958–1970, 2008.

[24] James Townsend, Thomas Bird, and David Barber. Practical lossless compression with latent variables using bits back coding. In *International Conference on Learning Representations (ICLR)*, 2019.

[25] James Townsend, Thomas Bird, Julius Kunze, and David Barber. Hi{ll}oc: lossless image compression with hierarchical latent variable models. In *International Conference on Learning Representations*, 2020.

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

[27] 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. In *International Conference on Learning Representations*, 2020.

[28] Cheng-hsin Wuu, Ningyuan Zheng, Scott Ardisson, Rohan Bali, Danielle Belko, Eric Brockmeyer, Lucas Evans, Timothy Godisart, Hyowon Ha, Xuhua Huang, et al. Multiface: A dataset for neural face rendering. *arXiv preprint arXiv:2207.11243*, 2022.

[29] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory*, 24(5):530–536, 1978.