Practical Full Resolution Learned Lossless Image Compression

Fabian Mentzer mentzerf@vision.ee.ethz.ch
Eirikur Agustsson aeirikur@vision.ee.ethz.ch
Michael Tschannen michaelt@nari.ee.ethz.ch
Radu Timofte timofter@vision.ee.ethz.ch
Luc Van Gool vangool@vision.ee.ethz.ch

ETH Zürich, Switzerland

Abstract

We propose the first practical learned lossless image compression system, L3C, and show that it outperforms the popular engineered codecs, PNG, WebP and JPEG2000. At the core of our method is a fully parallelizable hierarchical probabilistic model for adaptive entropy coding which is optimized end-to-end for the compression task. In contrast to recent autoregressive discrete probabilistic models such as PixelCNN, our method i) models the image distribution jointly with learned auxiliary representations instead of exclusively modeling the image distribution in RGB space, and ii) only requires three forward-passes to predict all pixel probabilities instead of one for each pixel. As a result, L3C obtains over three orders of magnitude speedups compared to the fastest PixelCNN variant (Multiscale-PixelCNN). Furthermore, we find that learning the auxiliary representation is crucial and outperforms predefined auxiliary representations such as an RGB pyramid significantly.

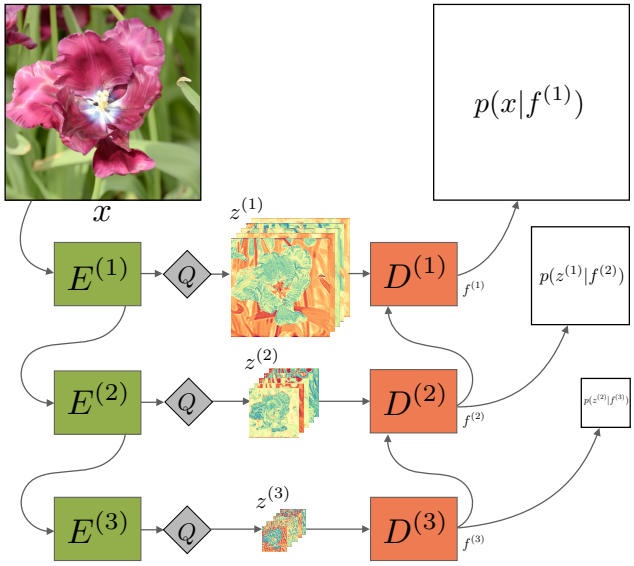

Figure 1. Overview of the architecture of L3C. The feature extractors $E^{(s)}$ compute quantized (by Q) hierarchical feature representation $z^{(1)}, \dots, z^{(S)}$ whose joint distribution with the image x, $p(x, z^{(1)}, \dots, z^{(S)})$, is modeled using the non-autoregressive predictors $D^{(s)}$. The features $f^{(s)}$ summarize the information up to level s.

1. Introduction

While it is known that likelihood-based discrete generative models can in theory be used for lossless compression [37], recent work on learned compression using deep neural networks has solely focused on lossy compression [4, 38, 39, 32, 1, 3]. Indeed, the literature on discrete generative models [42, 41, 33, 30, 18] has largely ignored the application as a lossless compression system, with neither bitrates nor runtimes being compared with classical codecs such as PNG [29], WebP [43], JPEG2000-lossless [35], and FLIF [36]. This is not surprising as (lossless) entropy coding using likelihood-based discrete generative models amounts to a decoding complexity essentially identical to the sampling complexity of the model, which renders many of the recent state-of-the-art autoregressive models such as PixelCNN [42], PixelCNN++ [33], and Multiscale-PixelCNN [30] impractical, requiring minutes or hours on GPU to generate moderately large images, typically $<256 \times 256$px (see Table 2). The computational complexity of these models is mainly caused by the sequential nature of the sampling (and thereby decoding) operation, where a forward pass needs to be computed for every single (sub) pixel of the image in a raster scan order.

In this paper, we address these challenges and develop a fully parallelizeable learned lossless compression system, outperforming the popular classical systems PNG, WebP and JPEG2000.

Our system (see Fig. 1 for an overview) is based on a hierarchy of fully parallel learned feature extractors and predictors which are trained jointly for the compression task. The role of the feature extractors is to build an auxiliary feature representation which helps the predictors to model

both the image, as well as the auxiliary features themselves. Our experiments show that learning the feature representations is crucial, and heuristic (predefined) choices such as a multiscale RGB pyramid lead to suboptimal performance.

In more detail, to encode an image x, we feed it to the feature extractors and predictors, and obtain the predictions of the probability distribution of x as well as all auxiliary variables in parallel in a single forward pass. These predictions are then used with an adaptive arithmetic encoder to obtain a compressed bitstream of x and the auxiliary variables (see Sec. 3.1 for an introduction to adaptive arithmetic coding). However, the arithmetic decoder now needs predictions to be able to decode the bitstream. Starting from the lowest level of auxiliary variables (for which we assume a uniform prior), the decoder obtains a prediction of the distribution of the next set of auxiliary variables, and can thus decode them from the bitstream. Prediction and decoding is alternated until finally the arithmetic decoder obtains the image x.

In practice, we only need to use three feature extractors and predictors for our model, so when decoding we only need to perform three fully parallel forward passes (which has the same complexity as the single forward pass through the predictors when encoding) in combination with the adaptive arithmetic coding.

The parallel nature of our model enables it to be orders of magnitude faster for decoding than autoregressive models, while learning enables us to obtain compression rates competitive with state-of-the-art engineered lossless codecs.

In summary, our contributions are the following:

- We propose a fully parallel hierarchical probabilistic model, learning both the feature extractors that produce an auxiliary feature representation to help the prediction task, as well as the predictors which model the joint distribution of all variables (see Section 3).

- We show that entropy coding based on our *non-autoregressive* probabilistic model optimized for discrete log-likelihood can obtain compression rates outperforming WebP, JPEG2000 and PNG, the latter by a large margin. We are only marginally outperformed by the state-of-the-art, FLIF, while being conceptually much simpler (see Section 5.1).

- At the same time, our model is practical in terms of runtime complexity and orders of magnitude faster than Pixel-CNN-based approaches. In particular, our model is $1.27 \cdot 10^5\times$ *faster than PixelCNN++*[33] and $1.4 \cdot 10^3\times$ *faster than the highly speed-optimized MS-PixelCNN* [30] (see Section 5.2).

2. Related Work

Likelihood-based generative models As already mentioned, essentially all likelihood-based discrete generative models can be used with an arithmetic coder for lossless compression. A prominent group of models that obtain state-of-the-art performance are variants of the autoregressive PixelRNN/PixelCNN [42, 41]. PixelRNN and PixelCNN organize the pixels of the image distribution as a sequence and predict the distribution of each pixel conditionally on (all) previous pixels using an RNN and a CNN with masked convolutions, respectively. These models hence require a number of network evaluations equal to the number of predicted sub-pixels. PixelCNN++ [33] simplifies the structure of PixelCNN and adds regularization, leading to faster training and better predictive performance. MS-PixelCNN [30] parallelizes PixelCNN by reducing dependencies between blocks of pixels and processing them in parallel with shallow PixelCNNs. [18] equips PixelCNN with auxiliary variables (grayscale version of the image or RGB pyramid) to encourage modeling of high-level features, thereby improving the overall the modeling performance. [6, 27] propose autoregressive models similar to PixelCNN/PixelRNN, but additionally rely on attention mechanisms to increase the receptive field.

Engineered codecs The well-known *PNG* [29] operates in two stages: first the image is reversibly transformed to a more compressible representation with a simple autoregressive filter that updates pixels based on surrounding pixels, then it is compressed with the deflate algorithm [10]. *WebP* [43] uses more involved transformations, including the use of entire image fragments to encode new pixels and a custom entropy coding scheme. *JPEG 2000* [35] includes a lossless mode where tiles are reversibly transformed before the coding step, instead of irreversibly removing frequencies. The current state-of-the-art (non-learned) algorithm is *FLIF* [36]. It relies on powerful preprocessing and a sophisticated entropy coding method based on CABAC [31] called MANIAC, which grows a dynamic decision tree per channel as an adaptive context model during encoding.

Context models in lossy compression In lossy compression, context models have been studied as a way to efficiently (losslessly) encode lossy representations of images. Classical approaches are discussed in [22, 24, 25, 46, 44]. Recent learned approaches include [20, 23, 26], where shallow autoregressive models over latents are learned.

Continuous likelihood models for compression The objective of continuous likelihood models, such as VAEs [17] and RealNVP [11], where $p(x')$ is a continuous distribution, is closely related to its discrete counterpart. In particular,

by setting $x' = x + u$ where x is the discrete image and u is uniform quantization noise, the continuous likelihood of $p(x')$ is a lower bound on the likelihood of the discrete $q(x) = \mathbb{E}_u[p(x')]$ [37]. However, there are two challenges for deploying such models for compression. First, the discrete likelihood $q(x)$ needs to be available (which involves a non-trivial integration step). Additionally, the memory complexity of (adaptive) arithmetic coding depends on the size of the domain of the variables of the factorization of q (see Sec.3.1 on (adaptive) arithmetic coding). Since the domain grows exponentially in the number of pixels in x, unless q is factorizable, it is not feasible to use it with adaptive arithmetic coding.

3. Method

3.1. Lossless Compression

In general, in lossless compression, we are given some stream of symbols x drawn independently and identically distributed (i.i.d.) from a set $\mathcal{X} = \{1, \dots, |\mathcal{X}|\}$ according to the probability mass function $\tilde{p}$. The goal is to encode this stream into a bitstream of minimal length using a "code", s.t. a receiver can decode the symbols from the bitstream. Ideally, an encoder minimizes the expected bits per symbol $\tilde{L} = \sum_{j \in \mathcal{X}} \tilde{p}(j)\ell(j)$, where $\ell(j)$ is the length of encoding symbol j (i.e., more probable symbols should obtain shorter codes). Information theory states (e.g., [8]) that $\tilde{L} \geq H(\tilde{p})$ for *any* possible code, where $H(\tilde{p}) = \mathbb{E}_{j\sim\tilde{p}}[-\log \tilde{p}(j)]$ is the Shannon entropy.

Arithmetic coding A strategy that almost achieves the lower bound $H(\tilde{p})$ (for long enough symbol streams) is arithmetic coding [45].[1] It encodes the entire stream into a single number $a' \in [0, 1)$, by subdividing $[0, 1)$ in each step (encoding one symbol) as follows: Let a, b the bounds of the current step (initialized to $a = 0$ and $b = 1$ for the initial interval $[0, 1)$). We divide the interval $[a, b)$ into $|\mathcal{X}|$ sections where the length of the j-th section is $\tilde{p}(j)/(b-a)$. Then we pick the interval corresponding to the current symbol, i.e., we update a, b to be the boundaries of this interval. We proceed recursively until no symbols are left. Finally, we transmit a', which is a rounded to the smallest number of bits s.t. $a' \geq a$. Receiving a' together with the knowledge of the number of encoded symbols and $\tilde{p}$ uniquely specifies the stream and allows the receiver to decode.

Adaptive arithmetic coding In contrast to the i.i.d. setting we just described, we are interested in losslessly encoding the pixels of an image which are not i.i.d at all. Let x_t be the sub-pixels of an image x, and $\tilde{p}_{\text{img}}(x)$ the joint distribution of all sub-pixels. We can then consider the factorization $\tilde{p}_{\text{img}}(x) = \prod_t \tilde{p}(x_t|x_{t-1}, \dots x_1)$. Now, to encode x, we can consider the sub-pixels x_t as our symbol stream and encode the t-th symbol/sub-pixel using $\tilde{p}(x_t|x_{t-1}, \dots x_1)$. Note that this corresponds to varying the $\tilde{p}(j)$ of the previous paragraph during encoding, and is in general referred to as *adaptive* arithmetic coding (AAC) [45]. For AAC the receiver also needs to know the varying $\tilde{p}$ at every step, i.e., they must either be known a priori or the factorization must be causal (as above) so that the receiver can calculate them from already decoded symbols.

Cross-entropy In practice, the exact $\tilde{p}$ is usually unknown, and instead is estimated by a model p. Thus, instead of using length $\log 1/\tilde{p}(x)$ to encode a symbol x, we need (sub-optimal) length $\log 1/p(x)$. Then

$$\begin{aligned} H(\tilde{p}, p) &= \mathbb{E}_{j\sim\tilde{p}}\left[-\log p(j)\right] \\ &= -\sum_{j\in\mathcal{X}} \tilde{p}(j) \log p(j) \end{aligned} \tag{1}$$

is the resulting expected (sub-optimal) bits per symbol, and is called the **cross-entropy** [8].

Thus, given some p, we can minimize the bitcost needed to encode a symbol stream with symbols distributed according to $\tilde{p}$ by minimizing Eq. (1). This naturally generalizes to the non i.i.d. case described in the previous paragraph by using different $\tilde{p}(x_t)$ and $p(x_t)$ for each symbol x_t and minimizing $\sum_t H(\tilde{p}(x_t), p(x_t))$.

The following sections describe how we model a hierarchical causal factorization of p_{img} for natural images to be able to do learned lossless image compression (L3C) efficiently.

3.2. Architecture

A high-level overview of the architecture is given in Fig. 1. Unlike autoregressive models such as PixelCNN and PixelRNN, which factorize the image distribution autoregressively over sub-pixels x_t as $p(x) = \prod_{t=1}^{T} p(x_t|x_{t-1}, \dots, x_1)$, we jointly model all the sub-pixels and introduce a learned hierarchy of auxiliary feature representations $z^{(1)}, \dots, z^{(S)}$ to simplify the modeling task.[2] Specifically, we model the joint distribution of the image x and the feature representations $z^{(s)}$ as

$$\begin{aligned} &p(x, z^{(1)}, \dots, z^{(S)}) = \\ &\qquad p(x|z^{(1)}, \dots, z^{(S)}) \prod_{s=1}^{S} p(z^{(s)}|z^{(s+1)}, \dots, z^{(S)}) \end{aligned}$$

[1] We use (adaptive) arithmetic coding for simplicity of exposition, but any adaptive entropy-achieving coder can be used with our method.

[2] We fix the dimensions of $z^{(s)}$ to be $C \times H' \times W'$, where the number of channels C is a hyperparameter ($C = 5$ in our reported models), and $H' = H/2^s, W' = W/2^s$ given a $H \times W$-dimensional image. Considering that $z^{(s)}$ is quantized, this conveniently upper bounds the information that can be contained within each $z^{(s)}$, however, other dimensions could be explored.

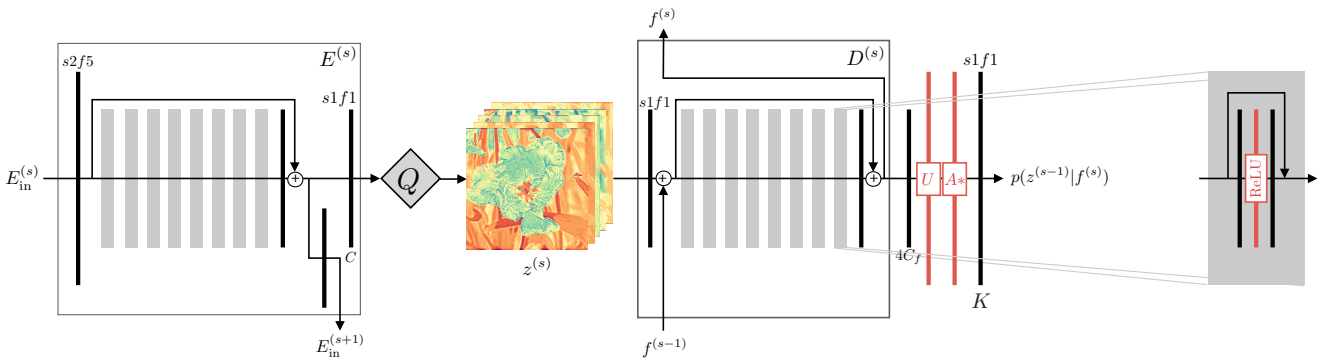

Figure 2. Architecture details for scale s. For $s = 1$, $E^{(1)}_{\text{in}}$ is the RGB image x normalized to $[-1, 1]$. All vertical **black** lines are convolutions, which have $C_f = 64$ filters, except when denoted otherwise beneath. All convolutions are stride 1 with 3×3 filters, except when denoted otherwise above ($sSfF$ = stride s, filter f). We add the features $f^{(s+1)}$ of the predictor $D^{(s+1)}$ to those of the first layer of predictor $D^{(s)}$ (this is essentially a skip connection between scales). The gray blocks are residual blocks, shown once on the right side. Special blocks are denoted in red: U is pixelshuffling upsampling [34]. $A*$ is the "atrous convolution" layer described in the text.

where $p(z^{(S)})$ is a uniform distribution. The feature representations can be hand designed or learned. Specifically, on one side, we consider an RGB pyramid with $z^{(s)} = \mathcal{B}_{2^s}(x)$, where $\mathcal{B}_{2^s}$ is the bicubic (spatial) subsampling operator with subsampling factor 2^s. On the other side, we consider a learned representation $z^{(s)} = F^{(s)}(x)$ using a feature extractor $F^{(s)}$. We use the hierarchical model shown in Fig. 1 using the composition $F^{(s)} = Q \circ E^{(s)} \circ \cdots \circ E^{(1)}$, where the $E^{(s)}$ are feature extractor blocks and Q is a scalar differentiable quantization function (see Section 3.3). The $D^{(s)}$ in Fig. 1 are predictor blocks, and we parametrize $E^{(s)}$ and $D^{(s)}$ as convolutional neural networks.

Letting $z^{(0)} = x$, we parametrize the conditional distributions for all $s \in \{0, \dots, S\}$ as

$$p(z^{(s)}|z^{(s+1)}, \dots, z^{(S)}) = p(z^{(s)}|f^{(s+1)}),$$

using the features of the predictor

$$f^{(s)} = D^{(s)}(f^{(s+1)}, z^{(s)}),$$

where $f^{(S+1)} = 0$, i.e., the final predictor only sees $z^{(S)}$.

A detailed description of the feature extractor and predictor architecture is given in Fig. 2. The predictor is based on the super-resolution architecture from EDSR [21], motivated by the fact that our prediction tasked is somewhat related to super-resolution in that both are dense prediction tasks involving spatial upsampling. We mirror the predictor to obtain the feature extractor, and follow [21] in not using BatchNorm [15]. Inspired by the "atrous spatial pyramid pooling" from [5], we insert a similar layer before predicting the K channels of $p(z^{(s-1)}|f^{(s)})$: In $A*$, we use three atrous convolutions in parallel, with rates 1, 2, and 4, then concatenate the resulting feature maps to a $3C_f$-dimensional feature map.

3.3. Quantization

We use the scalar quantization approach proposed in [23] to quantize the output of $E^{(s)}$: Given levels $\mathbb{L} = \{\ell_1, \dots, \ell_L\} \subset \mathbb{R}$, we use nearest neighbor assignments to quantize each entry $z' \in z^{(s)}$ as

$$z = Q(z') := \arg\min_j \|z' - \ell_j\|, \tag{2}$$

but use differentiable "soft quantization"

$$\tilde{Q}(z') = \sum_{j=1}^{L} \frac{\exp(-\sigma_q \|z' - \ell_j\|)}{\sum_{l=1}^{L} \exp(-\sigma_q \|z' - \ell_l\|)} \ell_j \tag{3}$$

to compute gradients for the backward pass, where σ_q is a hyperparameter relating to the "softness" of the quantization. For simplicity, we fix $\mathbb{L}$ to be $L = 25$ evenly spaced values in $[-1, 1]$.

3.4. Mixture Model

Letting again $z^{(0)} = x$, we model the conditional distributions $p(z^{(s)}|z^{(s-1)}, \dots, z^{(S)})$ using a generalization of the discretized logistic mixture model with K components proposed in [33]. Specifically, consider the $C \times H' \times W'$-dimensional feature representation $z^{(s)}$ whose entries take values in $\{0, \dots, 255\}$ for $s = 0$ and in $\mathbb{L}$ otherwise. Let c index the channel and u, v the spatial location. We assume the entries of $z^{(s)}_{c,u,v}$ to be independent across u, v but we share mixture weights across channels c, i.e.,

$$p(z^{(s)}|f^{s+1}) = \prod_{u,v} p(z^{(s)}_{1,u,v}, \dots, z^{(s)}_{c,u,v}|f^{(s+1)}) \tag{4}$$

using the mixture distribution

$$p(z^{(s)}_{1,u,v}, \dots, z^{(s)}_{c,u,v} | f^{(s+1)}) = \sum_k \Big[\pi^k_{u,v}(f^{(s+1)}) \cdot \prod_c p_{\text{mix}}(z^{(s)}_{c,u,v} | \mu^k_{c,u,v}(f^{(s+1)}), \sigma^k_{c,u,v}(f^{(s+1)})) \Big],$$

where the mixture weights π^k and the parameters of the mixture components μ^k and σ^k are predicted from the features $f^{(s+1)}$. The mixture components follow a logistic distribution,

$$p_{\text{mix}}(z|\mu,\sigma) = \big(\text{sigmoid}((z + b/2 - \mu)/\sigma) - \text{sigmoid}((z - b/2 - \mu)/\sigma)\big).$$

Here, b is the bin width of the quantization grid ($b = 1$ for $s = 0$ and $b = 1/12$ otherwise; the edge-cases $z = 0$ and $z = 255$ occurring for $s = 0$ are handled as described in [33, Sec. 2.1]). For the distribution of $z^{(0)} = x$, we follow [33, Sec. 2.2] by allowing μ^k to linearly depend on the RGB pixels of previous channels (for simplicity, this is not reflected in our notation). We emphasize that in contrast to [33], our model is *not* autoregressive over pixels, i.e., $z^{(s)}_{c,u,v}$ is modelled as independent across u, v.

3.5. Loss

We are now ready to define the loss, which is a generalization of the discrete logistic mixture loss introduced in [33]. Recall form Sec. 3.1 that our goal is to model the true joint distribution of x and the representations $z^{(s)}$, i.e., $\tilde{p}(x, z^{(1)}, \dots, z^{(s)})$ as accurately as possible using our model $p(x, z^{(1)}, \dots, z^{(s)})$. Thereby, the $z^{(s)} = F^{(s)}(x)$ are defined using the learned feature extractor blocks $E^{(s)}$, and $p(x, z^{(1)}, \dots, z^{(s)})$ is a product of discretized (conditional) logistic mixture models with parameters defined through the $f^{(s)}$, which are in turn computed using the learned predictor blocks $D^{(s)}$. As discussed in Sec. 3.1, the expected coding cost incurred by coding $x, z^{(1)}, \dots, z^{(s)}$ w.r.t. our model $p(x, z^{(1)}, \dots, z^{(s)})$ is the cross entropy $H(\tilde{p}, p)$.

We therefore directly minimize $H(\tilde{p}, p)$ w.r.t. the parameters of the feature extractor blocks $E^{(s)}$ and predictor blocks $D^{(s)}$ over samples. Specifically, given N training samples $x_1, \dots, x_N$, let $F^{(s)}_i = F^{(s)}(x_i)$. We minimize

$$\begin{aligned}
&\mathcal{L}(E^{(1)}, \dots, E^{(S)}, D^{(1)}, \dots, D^{(S)}) \\
&= -\sum_{i=1}^N \log \Big(p\big(x_i, F^{(1)}_i, \dots, F^{(S)}_i\big) \Big) \\
&= -\sum_{i=1}^N \log \Big(p\big(x_i | F^{(1)}_i, \dots, F^{(S)}_i\big) \cdot \prod_{s=1}^S p\big(F^{(s)}_i | F^{(s+1)}_i, \dots, F^{(S)}_i\big) \Big) \\
&= -\sum_{i=1}^N \Big(\log p(x_i | F^{(1)}_i, \dots, F^{(S)}_i) + \sum_{s=1}^S \log p(F^{(s)}_i | F^{(s+1)}_i, \dots, F^{(S)}_i) \Big). \qquad (5)
\end{aligned}$$

Note that the loss decomposes into the sum of the cross-entropies of the different representations. Also note that this loss corresponds to the negative log-likelihood of the data w.r.t. our model which is typically the perspective taken in the generative modeling literature (see, e.g., [42]).

Propagating gradients through targets We emphasize that in contrast to the generative model literature we learn the representations, propagating gradients to both $E^{(s)}$ and $D^{(s)}$, since each component of our loss depends on $D^{(s+1)}, \dots, D^{(S)}$ though the parametrization of the logistic distribution and on $E^{(s)}, \dots, E^{(1)}$ because of the differentiable Q. Thereby, our network can autonomously learn to navigate the trade-off between a) making the output $z^{(s)}$ of feature extractor $E^{(s)}$ more easily estimable for the predictor $D^{(s+1)}$ and b) putting enough information into $z^{(s)}$ for predictor $D^{(s)}$ to predict $z^{(s-1)}$.

3.6. Relationship to MS-PixelCNN

When the feature extractors in our approach are restricted to a non-learned multiscale RGB pyramid as auxiliary variables (see baselines in the next section), it shares some similarities with MS-PixelCNN [30]. In particular, [30] combines such a pyramid with upscaling networks which play the same role as the predictors in our architecture. Crucially however, they rely on i) combining such predictors with a shallow PixelCNN and ii) upscaling one dimension at a time ($W \times H \to 2W \times H \to 2W \times 2H$. While their complexity is reduced from $O(WH)$ forward passes needed for PixelCNN [42] to $O(\log WH)$, their approach is in practice still an order of magnitude slower than ours (See Table 2). Furthermore, we stress that these similarities only apply for our RGB baseline model, whereas our best models are obtained using learned feature extractors trained jointly with the predictors.

[bpsp]	Method	Open Images	RAISE1K	DIV2K
Ours	L3C	2.629	2.952	3.159
Learned Baselines	RGB Shared	3.060 +16%	3.588 +22%	3.870 +23%
	RGB	3.008 +14%	3.517 +19%	3.751 +19%
Non-Learned Approaches	PNG	3.779 +44%	4.167 +41%	4.527 +43%
	JPEG2000	2.778 +5.7%	3.094 +4.8%	3.331 +5.4%
	WebP	2.666 +1.4%	2.972 +0.7%	3.234 +2.4%
	FLIF	2.473 −5.9%	2.822 −4.4%	3.046 −3.6%

Table 1. Compression performance of our method (L3C) and learned baselines (RGB Shared and RGB) to previous (non-learned) approaches, in bits per sub-pixel (bpsp). We emphasize the difference in percentage to our method for each other method in *green* if L3C outperforms the other method and in *red* otherwise.

4. Experiments

Data sets The main models are trained on 213 487 images randomly selected from the *Open Images Train* data set [19]. We evaluate on 500 randomly selected images from *Open Images Test*, 500 randomly selected images from *RAISE1k* [9], as well as 100 images from the *DIV2K* super-resolution data set [2]. All data sets are downscaled to 768 pixels on the longer side to remove potential artifacts from previous compression, where we discarded images were rescaling did not result in at least $1.25\times$ downscaling. We also discarded any high saturation images, i.e., with mean $S > 0.9$ or $V > 0.8$ in the HSV color space (this removes 3 images from DIV2K, for the other test data sets we do this before selecting 500 images). We additionally investigate the ImageNet64 data set [7] with 1 281 151 training images and 50 000 validation images, each 64×64 pixels.

Models We compare our main model (L3C) to two learned baselines: For the **RGB Shared** baseline we use bicubic subsampling as feature extractors, i.e., $z^{(s)} = \mathcal{B}_{2^s}(x)$, and only train one predictor $D^{(1)}$. During testing, we can obtain multiple $z^{(s)}$ using $\mathcal{B}$ and apply the predictor $D^{(1)}$ as often as needed.[3] The **RGB** baseline also uses bicubic subsampling, however, we train $S = 3$ predictors, one for each scale, to capture the different distributions of different RGB scales.

For our main model, **L3C**, we additionally learn $S = 3$ feature extractors. We emphasize that the only difference to the **RGB** baseline is that the representations $z^{(s)}$ are learned. We train all these models for 700k iterations, where they converge. Additionally, in order to compare to the PixelCNN literature, we train L3C also on ImageNet64. Due to the smaller images and bigger data set, we increase the batch size to 120 and decay λ every epoch.

[3]We choose 4 applications as this results in the final $z^{(s)}$ requiring a negligible number of bits.

Training We use the RMSProp optimizer [14], with a batch size of 30, minimizing Eq. (5) directly (no regularization). We start with a learning rate $\lambda = 1 \cdot 10^{-4}$ and decay it by a factor of 0.75 every 5 epochs. We preprocess on the fly with random 128×128 crops and random horizontal flips.

Architecture ablations We find that adding BatchNorm slightly degrades performance. Furthermore, replacing the stacked atrous convolutions $A*$ with a single convolution, slightly degrades performance as well. By stopping gradients being propagated through the targets of our loss, we get significantly worse performance—in fact, the optimizer does not manage to pull down the cross-entropy of any of the learned representations $z^{(s)}$ significantly.

We find the choice of σ_q for Q has impacts on training: [23] suggests setting it s.t. $\tilde{Q}$ resembles identity, which we found a good starting point. However, we found it beneficial to let σ_q be slightly smoother, yielding better gradients for the encoder. We use $\sigma_q = 2$.

Additionally, we explored the impact of varying C (number of channels of $z^{(s)}$) and the number of levels L and found it more beneficial to increase L instead of increasing C, i.e., it is beneficial for training to have a finer quantization grid.

Other codecs We compare to FLIF and WebP using the respective official implementations [36, 43], for PNG we use the implementation of Pillow [28], and for the lossless (LL) mode of JPEG2000 we use the Kakadu implementation [16]. See Section 2 for a description these codecs.

5. Results

5.1. Compression

Table 1 shows a comparison of our approach (L3C) and the learned baselines to the other codecs, on our testsets. All of our methods outperform the widely-used PNG, which is at least 40% larger on all data sets. We also outperform WebP and JPEG2000-LL everywhere by a smaller margin of up to 5%. We note that FLIF still marginally outperforms our model but remind the reader of the many hand-engineered highly specialized techniques involved in FLIF (see Section 2). In contrast, we use a simple convolutional feed-forward neural network architecture. The RGB baseline with $S = 3$ learned predictors outperforms the RGB Shared baseline on all data sets, showing the importance of learning a predictor for each scale. Using our main model (L3C), where we additionally learn the feature extractors, we outperform both baselines: The outputs are at least 14% larger everywhere, showing the benefits of learning the representation.

5.2. Comparison with PixelCNN

Runtimes Table 2 shows a speed comparison to three PixelCNN-based approaches. The original PixelCNN [42] conditions the probability of seeing some sub-pixel value on all previously seen pixels in the current channel (R, G, or B) and all pixels from all previously seen channels. To decode a full image, $3{\cdot}W{\cdot}H$ forward passes are needed. PixelCNN++ [33] improves on this in various ways, including modeling the joint distribution of each pixel, thereby eliminating conditioning on previous channels and reducing to $W \cdot H$ forward passes, whereas MS-PixelCNN [30] needs $O(\log WH)$ forward passes (see Section 3.6).

	Method	32×32px	320×320px
BS=1	PixelCNN++[a]	47.4 s	$\approx$ 80 min
	L3C (Ours)[a]	0.0142 s	0.0214 s
BS=30	PixelCNN++[a]	11.3 s	$\approx$ 18 min
	L3C (Ours)[a]	0.000514 s	0.00850 s
	PixelCNN[b] [42]	120 s	$\approx$ 3 hours
	MS-PixelCNN[b] [30]	1.17 s	$\approx$ 2 min

Table 2. Comparison of our method (L3C) to the PixelCNN literature in terms of sampling complexity. The times in the first two rows were obtained with batch size (BS) 1, the other times with BS=30, since this is what is reported in [30]. We compare crops of 32×32 pixels to crops of 320×320 pixels, and report the time needed when sampling/decoding from the model. [a]: times obtained by us with code released of PixelCNN++ [33], on a Titan X (Pascal) GPU. [b]: times reported in [30], calculated on a Nvidia Quadro M4000 GPU. Note that for 320×320 resolution, times from other approaches are interpolated due to the long runtimes or unavailability of code.

Method	bpsp	Time for p	Learned
L3C (Ours)	4.52	0.0021 s	✓
PixelCNN [42]	3.57	$\approx$ 8 min	✓
MS-PixelCNN [30]	3.70	$\approx$ 4.7 s	
PNG	5.74		
JPEG2000-LL	4.93		
WebP	4.64		
FLIF	4.54		

Table 3. Comparison of bpsp on the 64×64 images from ImageNet64 of our method (L3C) vs. PixelCNN-based approaches and classical approaches. We report bits per sub-pixel (bpsp) and time needed when sampling/decoding from the model. Times for the PixelCNN approaches are interpolated from Table 2.

We note that the reported times only account for the time spent evaluating p when sampling/decoding. For all approaches, a pass with an entropy coder is needed to actually encode/decode images. Note, however, that state-of-the-art adaptive entropy coders typically require on the order of milliseconds per MB (see [12] and in particular [13] for benchmarks on adaptive entropy coding), and the time spent for entropy coding should thus be in the same order as computing p using L3C.

We observe that our approach is considerably faster than the three PixelCNN approaches. When comparing on 320×320 crops, we observe massive speedups of $>2.24 \cdot 10^5\times$ for batch size (BS) 1 and $>1.27 \cdot 10^5\times$ for BS 30 compared to the original PixelCNN [42], and for BS 30 we are $>1.4 \cdot 10^3\times$ faster than MS-PixelCNN, the fastest PixelCNN-based approach.[4]

Bitcost To put the runtimes reported in Table 2 into perspective, we further evaluate the bitcost on ImageNet64, for which PixelCNN and MS-PixelCNN were trained, in Table 3. The reported times are interpolated from the 32×32 results in Table 2 and again refer only to the time needed to evaluate p when sampling/decoding. We observe our outputs to be 22.2% larger than MS-PixelCNN and 26.6% larger than the original PixelCNN, but smaller than all classical approaches. However, we note again that our model (L3C) is over 3 orders of magnitude faster than MS-PixelCNN.

[4] We estimate the speedup of L3C compared to MS-PixelCNN from the results reported in [30] for PixelCNN and MS-PixelCNN, assuming PixelCNN++ is not slower than PixelCNN to get a conservative estimate (PixelCNN++ is in fact around $3\times$ faster than PixelCNN).

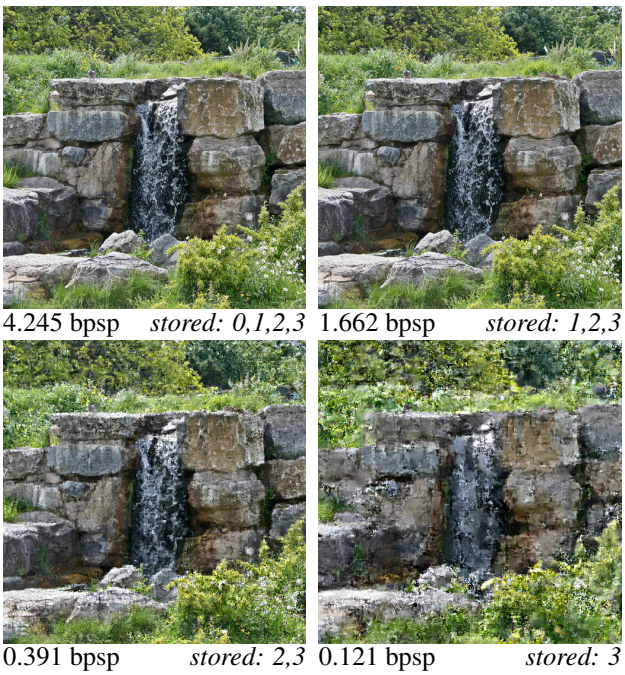

Figure 3. Effect of generating representations instead of storing them, on an image from the Open Images testset. Below each image, we show the required bitcost and which scales are stored.

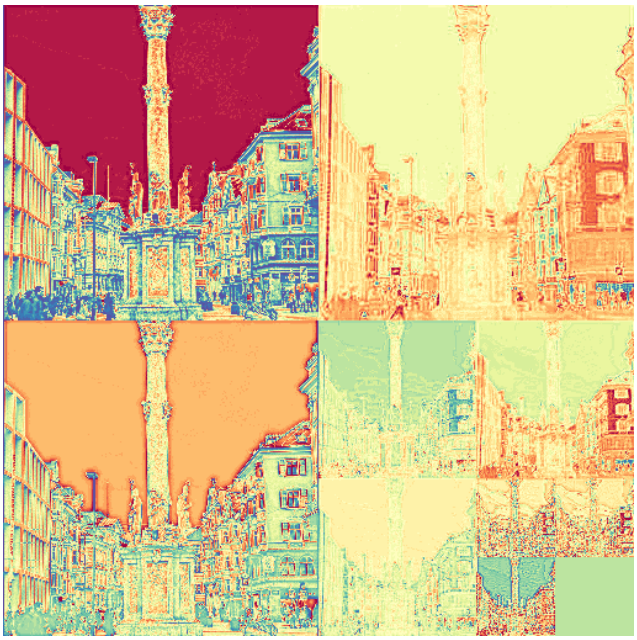

Figure 4. Heatmap visualization of the first three channels for each of the representations $z^{(1)}$, $z^{(2)}$, $z^{(3)}$, each containing values in $\mathbb{L}$.

5.3. Sampling Representations

We stress that we study image compression and not image generation. Nevertheless, our method produces models from which x and $z^{(s)}$ can be sampled. Therefore, we visualize the output when sampling part of the representations from our model in Fig. 3: the top left shows an image from the RAISE1k test set, when we store all scales (losslessly). When we store $z^{(1)}, z^{(2)}, z^{(3)}$ but not x and instead sample from $p(x|f^{(1)})$, we only need 39.2% of the total bits without noticeably degrading visual quality. Sampling $z^{(1)}$ and x leads to some blur while reducing the number of stored bits to 9.21% of the full bitcost. Finally, only storing $z^{(3)}$ (containing $64 \times 64 \times 5$ values from $\mathbb{L}$ and 2.85% of the full bitcost) and sampling $z^{(2)}$, $z^{(1)}$, and x produces significant artifacts. However, the original image is still recognizable, showing the ability of our networks to learn a hierarchical representation capturing global image structure.

5.4. Visualizing Representations

We visualize the representations $z^{(1)}, z^{(2)}, z^{(3)}$ in Fig. 4. It can be seen that the global image structure is preserved over scales, with representations corresponding to smaller s modeling more detail. This shows potential for efficiently performing image understanding tasks on partially decoded images similarly as described in [40] for lossy learned compression: instead of training a feature extractor for a given task on x, one could directly use the features $z^{(s)}$ from our network.

6. Conclusion

We proposed and evaluated a fully parallel hierarchical probabilistic model with auxiliary feature representations. Our L3C model outperformed PNG and JPEG-2000 on all data sets and reached performance comparable to WebP. Furthermore, it significantly outperformed the RGB Shared and RGB baselines which rely on predefined heuristic feature representations, showing that learning the representations is crucial. Additionally, we observed that using PixelCNN-based methods for losslessly compressing full-resolution images takes three to five orders of magnitude longer than L3C.

To further improve L3C, future work could investigate weak forms of autoregression across pixels and/or dynamic adaptation of the model network to the current image. Furthermore, it would be interesting to explore domain-specific applications, e.g., for medical image data.

References

[1] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. V. Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1141–1151, 2017. 1

[2] E. Agustsson and R. Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 6

[3] E. Agustsson, M. Tschannen, F. Mentzer, R. Timofte, and L. Van Gool. Generative adversarial networks for extreme learned image compression. *arXiv preprint arXiv:1804.02958*, 2018. 1

[4] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *International Conference on Learning Representations (ICLR)*, 2016. 1

[5] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017. 4

[6] X. Chen, N. Mishra, M. Rohaninejad, and P. Abbeel. Pixelsnail: An improved autoregressive generative model. *arXiv preprint arXiv:1712.09763*, 2017. 2

[7] P. Chrabaszcz, I. Loshchilov, and F. Hutter. A downsampled variant of imagenet as an alternative to the cifar datasets. *arXiv preprint arXiv:1707.08819*, 2017. 6

[8] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012. 3

[9] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato. Raise: a raw images dataset for digital image forensics. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 219–224. ACM, 2015. 6

[10] P. Deutsch. DEFLATE compressed data format specification version 1.3. Technical report, 1996. 2

[11] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 2

[12] J. Duda, K. Tahboub, N. J. Gadgil, and E. J. Delp. The use of asymmetric numeral systems as an accurate replacement for huffman coding. In *Picture Coding Symposium (PCS), 2015*, pages 65–69. IEEE, 2015. 7

[13] F. Giesen. Interleaved entropy coders. *arXiv preprint arXiv:1402.3392*, 2014. 7

[14] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. 6

[15] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. In *Advances in Neural Information Processing Systems*, pages 1945–1953, 2017. 4

[16] Kakadu JPEG2000 implementation. http://kakadusoftware.com. 6

[17] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 2

[18] A. Kolesnikov and C. H. Lampert. Pixelcnn models with auxiliary variables for natural image modeling. In *International Conference on Machine Learning*, pages 1905–1914, 2017. 1, 2

[19] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, S. Kamali, M. Malloci, J. Pont-Tuset, A. Veit, S. Belongie, V. Gomes, A. Gupta, C. Sun, G. Chechik, D. Cai, Z. Feng, D. Narayanan, and K. Murphy. Openimages: A public dataset for large-scale multi-label and multi-class image classification. *Dataset available from https://storage.googleapis.com/openimages/web/index.html*, 2017. 6

[20] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang. Learning convolutional networks for content-weighted image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2

[21] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017. 4

[22] D. Marpe, H. Schwarz, and T. Wiegand. Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):620–636, 2003. 2

[23] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 2, 4, 6

[24] B. Meyer and P. Tischer. Tmw-a new method for lossless image compression. *ITG FACHBERICHT*, pages 533–540, 1997. 2

[25] B. Meyer and P. E. Tischer. Glicbawls-grey level image compression by adaptive weighted least squares. In *Data Compression Conference*, volume 503, 2001. 2

[26] D. Minnen, J. Ballé, and G. Toderici. Joint autoregressive and hierarchical priors for learned image compression. *arXiv preprint arXiv:1809.02736*, 2018. 2

[27] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, and A. Ku. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018. 2

[28] Pillow Library for Python. https://python-pillow.org. 6

[29] Portable Network Graphics (PNG). http://libpng.org/pub/png/libpng.html. 1, 2

[30] S. Reed, A. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. Freitas. Parallel multiscale autoregressive density estimation. In *International Conference on Machine Learning*, pages 2912–2921, 2017. 1, 2, 5, 7

[31] I. E. Richardson. *H. 264 and MPEG-4 video compression: video coding for next-generation multimedia*. John Wiley & Sons, 2004. 2

[32] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2922–2930, 2017. 1

[33] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic

mixture likelihood and other modifications. In *ICLR*, 2017. 1, 2, 4, 5, 7

[34] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1874–1883, 2016. 4

[35] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. *IEEE Signal processing magazine*, 18(5):36–58, 2001. 1, 2

[36] J. Sneyers and P. Wuille. Flif: Free lossless image format based on maniac compression. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 66–70, Sept 2016. 1, 2, 6

[37] L. Theis, A. v. d. Oord, and M. Bethge. A note on the evaluation of generative models. In *International Conference on Learning Representations (ICLR)*, 2016. 1, 3

[38] L. Theis, W. Shi, A. Cunningham, and F. Huszar. Lossy image compression with compressive autoencoders. In *International Conference on Learning Representations (ICLR)*, 2017. 1

[39] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5435–5443. IEEE, 2017. 1

[40] R. Torfason, F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool. Towards image understanding from deep compression without decoding. *arXiv preprint arXiv:1803.06131*, 2018. 8

[41] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 1, 2

[42] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756, 2016. 1, 2, 5, 7

[43] WebP Image format. https://developers.google.com/speed/webp/. 1, 2, 6

[44] M. J. Weinberger, J. J. Rissanen, and R. B. Arps. Applications of universal context modeling to lossless compression of gray-scale images. *IEEE Transactions on Image Processing*, 5(4):575–586, 1996. 2

[45] I. H. Witten, R. M. Neal, and J. G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6):520–540, 1987. 3

[46] X. Wu, E. Barthel, and W. Zhang. Piecewise 2d autoregression for predictive image coding. In *Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on*, pages 901–904. IEEE, 1998. 2