Conditional Probability Models for Deep Image Compression

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

ETH Zürich, Switzerland

Abstract

Deep Neural Networks trained as image auto-encoders have recently emerged as a promising direction for advancing the state-of-the-art in image compression. The key challenge in learning such networks is twofold: To deal with quantization, and to control the trade-off between reconstruction error (distortion) and entropy (rate) of the latent image representation. In this paper, we focus on the latter challenge and propose a new technique to navigate the rate-distortion trade-off for an image compression auto-encoder. The main idea is to directly model the entropy of the latent representation by using a context model: A 3D-CNN which learns a conditional probability model of the latent distribution of the auto-encoder. During training, the auto-encoder makes use of the context model to estimate the entropy of its representation, and the context model is concurrently updated to learn the dependencies between the symbols in the latent representation. Our experiments show that this approach, when measured in MS-SSIM, yields a state-of-the-art image compression system based on a simple convolutional auto-encoder.

1. Introduction

Image compression refers to the task of representing images using as little storage (i.e., bits) as possible. While in lossless image compression the compression rate is limited by the requirement that the original image should be perfectly reconstructible, in *lossy* image compression, a greater reduction in storage is enabled by allowing for some distortion in the reconstructed image. This results in a so-called rate-distortion trade-off, where a balance is found between the bitrate R and the distortion d by minimizing $d + \beta R$, where $\beta > 0$ balances the two competing objectives. Recently, deep neural networks (DNNs) trained as image auto-encoders for this task led to promising results, achieving better performance than many traditional techniques for image compression [19, 20, 17, 4, 2, 9]. Another advantage of DNN-based learned compression systems is their adaptability to specific target domains such as areal images or stereo images, enabling even higher compression rates on these domains. A key challenge in training such systems is to optimize the bitrate R of the latent image representation in the auto-encoder. To encode the latent representation using a finite number of bits, it needs to be discretized into symbols (i.e., mapped to a stream of elements from some finite set of values). Since discretization is non-differentiable, this presents challenges for gradient-based optimization methods and many techniques have been proposed to address them. After discretization, information theory tells us that the correct measure for bitrate R is the entropy H of the resulting symbols. Thus the challenge, and the focus of this paper, is how to model H such that we can navigate the trade-off $d + \beta H$ during optimization of the auto-encoder.

Our proposed method is based on leveraging context models, which were previously used as techniques to im-

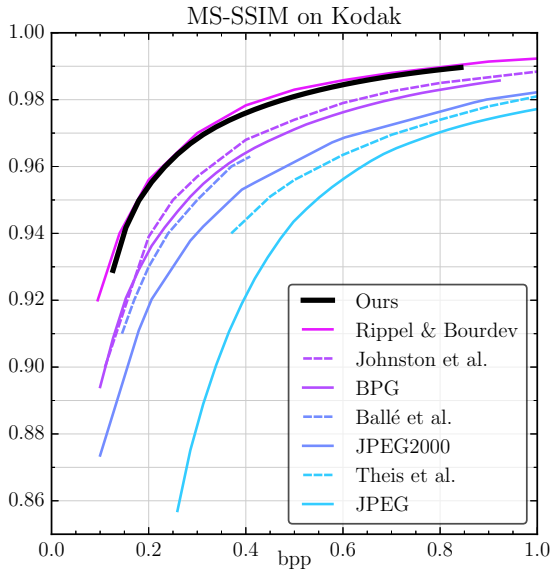

Figure 1: State-of-the-art performance achieved by our simple compression system composed of a standard convolutional auto-encoder and a 3D-CNN-based context model.

*The first two authors contributed equally.

prove coding rates for already-trained models [4, 20, 9, 14], directly as an entropy term in the optimization. We concurrently train the auto-encoder and the context model with respect to each other, where the context model learns a convolutional probabilistic model of the image representation in the auto-encoder, while the auto-encoder uses it for entropy estimation to navigate the rate-distortion trade-off. Furthermore, we generalize our formulation to spatially-aware networks, which use an *importance map* to spatially attend the bitrate representation to the most important regions in the compressed representation. The proposed techniques lead to a simple image compression system[1], which achieves state-of-the-art performance when measured with the popular multi-scale structural similarity index (MS-SSIM) distortion metric [23], while being straightforward to implement with standard deep-learning toolboxes.

2. Related work

Full-resolution image compression using DNNs has attracted considerable attention recently. DNN architectures commonly used for image compression are auto-encoders [17, 4, 2, 9] and recurrent neural networks (RNNs) [19, 20]. The networks are typically trained to minimize the mean-squared error (MSE) between original and decompressed image [17, 4, 2, 9], or using perceptual metrics such as MS-SSIM [20, 14]. Other notable techniques involve progressive encoding/decoding strategies [19, 20], adversarial training [14], multi-scale image decompositions [14], and generalized divisive normalization (GDN) layers [4, 3].

Context models and entropy estimation—the focus of the present paper—have a long history in the context of engineered compression methods, both lossless and lossy [24, 12, 25, 13, 10]. Most of the recent DNN-based lossy image compression approaches have also employed such techniques in some form. [4] uses a binary context model for adaptive binary arithmetic coding [11]. The works of [20, 9, 14] use learned context models for improved coding performance on their trained models when using adaptive arithmetic coding. [17, 2] use non-adaptive arithmetic coding but estimate the entropy term with an independence assumption on the symbols.

Also related is the work of van den Oord *et al*. [22, 21], who proposed PixelRNN and PixelCNN, powerful RNN- and CNN-based context models for modeling the distribution of natural images in a lossless setting, which can be used for (learned) lossless image compression as well as image generation.

3. Proposed method

Given a set of training images $\mathcal{X}$, we wish to learn a compression system which consists of an encoder, a quantizer, and a decoder. The encoder $E : \mathbb{R}^d \to \mathbb{R}^m$ maps an image $\mathbf{x}$ to a latent representation $\mathbf{z} = E(\mathbf{x})$. The quantizer $Q : \mathbb{R} \to \mathcal{C}$ discretizes the coordinates of $\mathbf{z}$ to $L = |\mathcal{C}|$ centers, obtaining $\hat{\mathbf{z}}$ with $\hat{z}_i := Q(z_i) \in \mathcal{C}$, which can be losslessly encoded into a bitstream. The decoder D then forms the reconstructed image $\hat{\mathbf{x}} = D(\hat{\mathbf{z}})$ from the quantized latent representation $\hat{\mathbf{z}}$, which is in turn (losslessy) decoded from the bitstream. We want the encoded representation $\hat{\mathbf{z}}$ to be compact when measured in bits, while at the same time we want the distortion $d(\mathbf{x}, \hat{\mathbf{x}})$ to be small, where d is some measure of reconstruction error, such as MSE or MS-SSIM. This results in the so-called rate-distortion trade-off

$$d(\mathbf{x}, \hat{\mathbf{x}}) + \beta H(\hat{\mathbf{z}}), \tag{1}$$

where H denotes the cost of encoding $\hat{\mathbf{z}}$ to bits, i.e., the entropy of $\hat{\mathbf{z}}$. Our system is realized by modeling E and D as convolutional neural networks (CNNs) (more specifically, as the encoder and decoder, respectively, of a convolutional auto-encoder) and minimizing (1) over the training set $\mathcal{X}$, where a large/small β draws the system towards low/high average entropy H. In the next sections, we will discuss how we quantize $\mathbf{z}$ and estimate the entropy $H(\hat{\mathbf{z}})$. We note that as E, D are CNNs, $\hat{\mathbf{z}}$ will be a 3D feature map, but for simplicity of exposition we will denote it as a vector with equally many elements. Thus, $\hat{z}_i$ refers to the i-th element of the feature map, in raster scan order (row by column by channel).

3.1. Quantization

We adopt the scalar variant of the quantization approach proposed in [2] to quantize $\mathbf{z}$, but simplify it using ideas from [17]. Specifically, given centers $\mathcal{C} = \{c_1, \cdots, c_L\} \subset \mathbb{R}$, we use nearest neighbor assignments to compute

$$\hat{z}_i = Q(z_i) := \arg\min_j \|z_i - c_j\|, \tag{2}$$

but rely on (differentiable) soft quantization

$$\tilde{z}_i = \sum_{j=1}^{L} \frac{\exp(-\sigma\|z_i - c_j\|)}{\sum_{l=1}^{L} \exp(-\sigma\|z_i - c_l\|)} c_j \tag{3}$$

to compute gradients during the backward pass. This combines the benefit of [2] where the quantization is restricted to a finite set of learned centers $\mathcal{C}$ (instead of the fixed (non-learned) integer grid as in [17]) and the simplicity of [17], where a differentiable approximation of quantization is only used in the backward pass, avoiding the need to choose an annealing strategy (i.e., a schedule for σ) as in [2] to drive the soft quantization (3) to hard assignments (2) during training. In TensorFlow, this is implemented as

$$\bar{z}_i = \text{tf.stopgradient}(\hat{z}_i - \tilde{z}_i) + \tilde{z}_i. \tag{4}$$

We note that for forward pass computations, $\bar{z}_i = \hat{z}_i$, and thus we will continue writing $\hat{z}_i$ for the latent representation.

[1] https://github.com/fab-jul/imgcomp-cvpr

3.2. Entropy estimation

To model the entropy $H(\hat{\mathbf{z}})$ we build on the approach of PixelRNN [22] and factorize the distribution $p(\hat{\mathbf{z}})$ as a product of conditional distributions

$$p(\hat{\mathbf{z}}) = \prod_{i=1}^{m} p(\hat{z}_i|\hat{z}_{i-1}, \dots, \hat{z}_1), \tag{5}$$

where the 3D feature volume $\hat{\mathbf{z}}$ is indexed in raster scan order. We then use a neural network $P(\hat{\mathbf{z}})$, which we refer to as a *context model*, to estimate each term $p(\hat{z}_i|\hat{z}_{i-1}, \dots, \hat{z}_1)$:

$$P_{i,l}(\hat{\mathbf{z}}) \approx p(\hat{z}_i = c_l|\hat{z}_{i-1}, \dots, \hat{z}_1), \tag{6}$$

where $P_{i,l}$ specifies for every 3D location i in $\hat{\mathbf{z}}$ the probabilites of each symbol in $\mathcal{C}$ with $l = 1, \dots, L$. We refer to the resulting approximate distribution as $q(\hat{\mathbf{z}}) := \prod_{i=1}^{m} P_{i,I(\hat{z}_i)}(\hat{\mathbf{z}})$, where $I(\hat{z}_i)$ denotes the index of $\hat{z}_i$ in $\mathcal{C}$.

Since the conditional distributions $p(\hat{z}_i|\hat{z}_{i-1}, \dots, \hat{z}_1)$ only depend on previous values $\hat{z}_{i-1}, \dots, \hat{z}_1$, this imposes a *causality* constraint on the network P: While P may compute $P_{i,l}$ in parallel for $i = 1, \dots, m, l = 1, \dots, L$, it needs to make sure that each such term only depends on previous values $\hat{z}_{i-1}, \dots, \hat{z}_1$.

The authors of PixelCNN [22, 21] study the use of 2D-CNNs as causal conditional models over 2D images in a lossless setting, i.e., treating the RGB pixels as symbols. They show that the causality constraint can be efficiently enforced using masked filters in the convolution. Intuitively, the idea is as follows: If for each layer the causality condition is satisfied with respect to the spatial coordinates of the layer before, then by induction the causality condition will hold between the output layer and the input. Satisfying the causality condition for each layer can be achieved with proper masking of its weight tensor, and thus the entire network can be made causal *only through the masking of its weights*. Thus, the entire set of probabilities $P_{i,l}$ for all (2D) spatial locations i and symbol values l can be computed in parallel with a fully convolutional network, as opposed to modeling each term $p(\hat{z}_i|\hat{z}_{i-1}, \cdots, \hat{z}_1)$ separately.

In our case, $\hat{\mathbf{z}}$ is a 3D symbol volume, with as much as $K = 64$ channels. We therefore generalize the approach of PixelCNN to 3D convolutions, using the same idea of masking the filters properly in every layer of the network. This enables us to model P efficiently, with a light-weight[2] 3D-CNN which slides over $\hat{z}$, while properly respecting the causality constraint. We refer to the supplementary material[3] for more details.

As in [21], we learn P by training it for maximum likelihood, or equivalently (see [16]) by training $P_{i,:}$ to classify the index $I(\hat{z}_i)$ of $\hat{z}_i$ in $\mathcal{C}$ with a cross entropy loss:

$$CE := \mathbb{E}_{\hat{z}\sim p(\hat{\mathbf{z}})}[\sum_{i=1}^{m} -\log P_{i,I(\hat{z}_i)}]. \tag{7}$$

Using the well-known property of cross entropy as the coding cost when using the wrong distribution $q(\hat{\mathbf{z}})$ instead of the true distribution $p(\hat{\mathbf{z}})$, we can also view the CE loss as an estimate of $H(\hat{\mathbf{z}})$ since we learn P such that $P = q \approx p$. That is, we can compute

$$\begin{aligned} H(\hat{\mathbf{z}}) &= \mathbb{E}_{\hat{z}\sim p(\hat{\mathbf{z}})}[-\log(p(\hat{\mathbf{z}}))] && (8)\\ &= \mathbb{E}_{\hat{z}\sim p(\hat{\mathbf{z}})}[\sum_{i=1}^{m} -\log p(\hat{z}_i|\hat{z}_{i-1}, \cdots, \hat{z}_1)] && (9)\\ &\approx \mathbb{E}_{\hat{z}\sim p(\hat{\mathbf{z}})}[\sum_{i=1}^{m} -\log q(\hat{z}_i|\hat{z}_{i-1}, \cdots, \hat{z}_1)] && (10)\\ &= \mathbb{E}_{\hat{z}\sim p(\hat{\mathbf{z}})}[\sum_{i=1}^{m} -\log P_{i,I(\hat{z}_i)}] && (11)\\ &= CE && (12) \end{aligned}$$

Therefore, when training the auto-encoder we can indirectly minimize $H(\hat{\mathbf{z}})$ through the cross entropy CE. We refer to argument in the expectation of (7),

$$C(\hat{\mathbf{z}}) := \sum_{i=1}^{m} -\log P_{i,I(\hat{z}_i)}, \tag{13}$$

as the *coding cost* of the latent image representation, since this reflects the coding cost incurred when using P as a context model with an adaptive arithmetic encoder [11]. From the application perspective, minimizing the coding cost is actually more important than the (unknown) true entropy, since it reflects the bitrate obtained in practice.

To backpropagate through $P(\hat{\mathbf{z}})$ we use the same approach as for the encoder (see (4)). Thus, like the decoder D, P only sees the (discrete) $\hat{\mathbf{z}}$ in the forward pass, whereas the gradient of the soft quantization $\tilde{\mathbf{z}}$ is used for the backward pass.

3.3. Concurrent optimization

Given an auto-encoder (E, D), we can train P to model the dependencies of the entries of $\hat{\mathbf{z}}$ as described in the previous section by minimizing (7). On the other hand, using the model P, we can obtain an estimate of $H(\hat{\mathbf{z}})$ as in (12) and use this estimate to adjust (E, D) such that $d(\mathbf{x}, D(Q(E(\mathbf{x}))))+\beta H(\hat{\mathbf{z}})$ is reduced, thereby navigating the rate distortion trade-off. Therefore, it is natural to concurrently learn P (with respect to its own loss), and (E, D) (with respect to the rate distortion trade-off) during training, such that all models which the losses depend on are continuously updated.

[2] We use a 4-layer network, compared to 15 layers in [22].

[3] Available at https://arxiv.org/abs/1801.04260

3.4. Importance map for spatial bit-allocation

Recall that since E and D are CNNs, $\hat{\mathbf{z}}$ is a 3D feature-map. For example, if E has three stride-2 convolution layers and the bottleneck has K channels, the dimensions of $\hat{\mathbf{z}}$ will be $\frac{W}{8} \times \frac{H}{8} \times K$. A consequence of this formulation is that we are using equally many symbols in $\hat{\mathbf{z}}$ for each spatial location of the input image $\mathbf{x}$. It is known, however, that in practice there is great variability in the information content across spatial locations (e.g., the uniform area of blue sky vs. the fine-grained structure of the leaves of a tree).

This can in principle be accounted for automatically in the trade-off between the entropy and the distortion, where the network would learn to output more predictable (i.e., low entropy) symbols for the low information regions, while making room for the use of high entropy symbols for the more complex regions. More precisely, the formulation in (7) already allows for variable bit allocation for different spatial regions through the context model P.

However, this arguably requires a quite sophisticated (and hence computationally expensive) context model, and we find it beneficial to follow Li *et al*. [9] instead by using an *importance map* to help the CNN attend to different regions of the image with different amounts of bits. While [9] uses a separate network for this purpose, we consider a simplified setting. We take the last layer of the encoder E, and add a second single-channel output $\mathbf{y} \in \mathbb{R}^{\frac{W}{8} \times \frac{H}{8} \times 1}$. We expand this single channel $\mathbf{y}$ into a mask $\mathbf{m} \in \mathbb{R}^{\frac{W}{8} \times \frac{H}{8} \times K}$ of the same dimensionality as $\mathbf{z}$ as follows:

$$m_{i,j,k} = \begin{cases} 1 & \text{if } k < y_{i,j} \\ (y_{i,j} - k) & \text{if } k \leq y_{i,j} \leq k+1 \\ 0 & \text{if } k+1 > y_{i,j} \end{cases}, \qquad (14)$$

where $y_{i,j}$ denotes the value of $\mathbf{y}$ at spatial location (i, j). The transition value for $k \leq y_{i,j} \leq k+1$ is such that the mask smoothly transitions from 0 to 1 for non-integer values of $\mathbf{y}$.

We then mask $\mathbf{z}$ by pointwise multiplication with the binarization of $\mathbf{m}$, i.e., $\mathbf{z} \leftarrow \mathbf{z} \odot \lceil \mathbf{m} \rceil$. Since the ceiling operator $\lceil \cdot \rceil$ is not differentiable, as done by [17, 9], we use identity for the backward pass.

With this modification, we have simply changed the architecture of E slightly such that it can easily "zero out" portions of columns $\mathbf{z}_{i,j,:}$ of $\mathbf{z}$ (the rest of the network stays the same, so that (2) still holds for example). As suggested by [9], the so-obtained structure in $\mathbf{z}$ presents an alternative coding strategy: Instead of losslessly encoding the entire symbol volume $\hat{\mathbf{z}}$, we could first (separately) encode the mask $\lceil \mathbf{m} \rceil$, and then for each column $\hat{z}_{i,j,:}$ only encode the first $\lceil m_{i,j} \rceil + 1$ symbols, since the remaining ones are the constant $Q(0)$, which we refer to as the *zero symbol*.

Work [9] uses binary symbols (i.e., $\mathcal{C} = \{0, 1\}$) and assumes independence between the symbols and a uniform prior during training, i.e., costing each 1 bit to encode. The importance map is thus their principal tool for controlling the bitrate, since they thereby avoid encoding all the bits in the representation. In contrast, we stick to the formulation in (5) where the dependencies between the symbols are modeled during training. We then use the importance map as an architectural constraint and use their suggested coding strategy to obtain an alternative estimate for the entropy $H(\hat{\mathbf{z}})$, as follows.

We observe that we can recover $\lceil \mathbf{m} \rceil$ from $\hat{\mathbf{z}}$ by counting the number of consecutive zero symbols at the end of each column $\hat{\mathbf{z}}_{i,j,:}$.[4] $\lceil \mathbf{m} \rceil$ is therefore a function of the masked $\hat{\mathbf{z}}$, i.e., $\lceil \mathbf{m} \rceil = g(\hat{\mathbf{z}})$ for g recovering $\lceil \mathbf{m} \rceil$ as described, which means that we have for the conditional entropy $H(\lceil \mathbf{m} \rceil | \hat{\mathbf{z}}) = 0$. Now, we have

$$\begin{aligned} H(\hat{\mathbf{z}}) &= H(\lceil \mathbf{m} \rceil | \hat{\mathbf{z}}) + H(\hat{\mathbf{z}}) && (15) \\ &= H(\hat{\mathbf{z}}, \lceil \mathbf{m} \rceil) && (16) \\ &= H(\hat{\mathbf{z}} | \lceil \mathbf{m} \rceil) + H(\lceil \mathbf{m} \rceil). && (17) \end{aligned}$$

If we treat the entropy of the mask, $H(\lceil \mathbf{m} \rceil)$, as constant during optimization of the auto-encoder, we can then indirectly minimize $H(\hat{\mathbf{z}})$ through $H(\hat{\mathbf{z}} | \mathbf{m})$.

To estimate $H(\hat{\mathbf{z}} | \mathbf{m})$, we use the same factorization of p as in (5), but since the mask $\lceil \mathbf{m} \rceil$ is known we have $p(\hat{z}_i = c_0) = 1$ deterministic for the 3D locations i in $\hat{\mathbf{z}}$ where the mask is zero. The logs of the corresponding terms in (9) then evaluate to 0. The remaining terms, we can model with the same context model $P_{i,l}(\hat{\mathbf{z}})$, which results in

$$H(\hat{\mathbf{z}} | \lceil \mathbf{m} \rceil) \approx E_{\hat{z} \sim p(\hat{\mathbf{z}})} [\sum_{i=1}^{m} -\lceil m_i \rceil \log P_{i, I(\hat{z}_i)}], \qquad (18)$$

where m_i denotes the i-th element of $\mathbf{m}$ (in the same raster scan order as $\hat{z}$).

Similar to the coding cost (13), we refer to the argument in the expectation in (18),

$$MC(\hat{\mathbf{z}}) := \sum_{i=1}^{m} -\lceil m_i \rceil \log P_{i, I(\hat{z}_i)} \qquad (19)$$

as the *masked coding cost* of $\hat{\mathbf{z}}$.

While the entropy estimate (18) is almost estimating the same quantity as (7) (only differing by $H(\lceil \mathbf{m} \rceil)$), it has the benefit of being weighted by m_i. Therefore, the encoder E has an obvious path to control the entropy of $\hat{\mathbf{z}}$, by simply increasing/decreasing the value of $\mathbf{y}$ for some spatial location of $\mathbf{x}$ and thus obtaining fewer/more zero entries in $\mathbf{m}$.

When the context model $P(\hat{\mathbf{z}})$ is trained, however, we still train it with respect to the formulation in (8), so it does

[4] If $\mathbf{z}$ contained zeros before it was masked, we might overestimate the number of 0 entries in $\lceil \mathbf{m} \rceil$. However, we can redefine those entries of $\mathbf{m}$ as 0 and this will give the same result after masking.

not have direct access to the mask $\mathbf{m}$ and needs to learn the dependencies on the entire masked symbol volume $\hat{\mathbf{z}}$. This means that when encoding an image, we can stick to standard adaptive arithmetic coding over the entire bottleneck, without needing to resort to a two-step coding process as in [9], where the mask is first encoded and then the remaining symbols. We emphasize that this approach hinges critically on the context model P and the encoder E being trained concurrently as this allows the encoder to learn a meaningful (in terms of coding cost) mask with respect to P (see the next section).

In our experiments we observe that during training, the two entropy losses (7) and (18) converge to almost the same value, with the latter being around $\approx 3.5\%$ smaller due to $H(\lceil \mathbf{m} \rceil)$ being ignored.

While the importance map is not crucial for optimal rate-distortion performance, if the channel depth K is adjusted carefully, we found that we could more easily control the entropy of $\hat{\mathbf{z}}$ through β when using a fixed K, since the network can easily learn to ignore some of the channels via the importance map. Furthermore, in the supplementary material we show that by using multiple importance maps for a single network, one can obtain a single model that supports multiple compression rates.

3.5. Putting the pieces together

We made an effort to carefully describe our formulation and its motivation in detail. While the description is lengthy, when putting the resulting pieces together we get a quite straightforward pipeline for learned image compression, as follows.

Given the set of training images $\mathcal{X}$, we initialize (fully convolutional) CNNs E, D, and P, as well as the centers $\mathcal{C}$ of the quantizer Q. Then, we train over minibatches $\mathcal{X}_B = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(B)}\}$ of crops from $\mathcal{X}$. At each iteration, we take one gradient step for the auto-encoder (E, D) and the quantizer Q, with respect to the rate-distortion trade-off

$$\mathcal{L}_{E,D,Q} = \frac{1}{B}\sum_{j=1}^{B} d(\mathbf{x}^{(j)}, \hat{\mathbf{x}}^{(j)}) + \beta MC(\hat{\mathbf{z}}^{(j)}), \qquad (20)$$

which is obtained by combining (1) with the estimate (18) & (19) and taking the batch sample average. Furthermore, we take a gradient step for the context model P with respect to its objective (see (7) & (13))

$$\mathcal{L}_P := \frac{1}{B}\sum_{j=1}^{B} d(\mathbf{x}^{(j)}, \hat{\mathbf{x}}^{(j)}) + \beta C(\hat{\mathbf{z}}^{(j)}). \qquad (21)$$

To compute these two batch losses, we need to perform the following computation for each $\mathbf{x} \in \mathcal{X}_B$:

1. Obtain compressed (latent) representation $\mathbf{z}$ and importance map $\mathbf{y}$ from the encoder: $(\mathbf{z}, \mathbf{y}) = E(\mathbf{x})$
2. Expand importance map $\mathbf{y}$ to mask $\mathbf{m}$ via (14)
3. Mask $\mathbf{z}$, i.e., $\mathbf{z} \leftarrow \mathbf{z} \odot \lceil \mathbf{m} \rceil$
4. Quantize $\hat{\mathbf{z}} = Q(\mathbf{z})$
5. Compute the context $P(\hat{\mathbf{z}})$
6. Decode $\hat{\mathbf{x}} = D(\hat{\mathbf{z}})$,

which can be computed in parallel over the minibatch on a GPU since all the models are fully convolutional.

3.6. Relationship to previous methods

We are not the first to use context models for adaptive arithmetic coding to improve the performance in learned deep image compression. Work [20] uses a PixelRNN-like architecture [22] to train a recurrent network as a context model for an RNN-based compression auto-encoder. Li *et al*. [9] extract cuboid patches around each symbol in a binary feature map, and feed them to a convolutional context model. Both these methods, however, only learn the context model *after* training their system, as a post-processing step to boost coding performance.

In contrast, our method directly incorporates the context model as the entropy term for the rate-distortion term (1) of the auto-encoder, and trains the two concurrently. This is done at little overhead during training, since we adopt a 3D-CNN for the context model, using PixelCNN-inspired [21] masking of the weights of each layer to ensure causality in the context model. Adopting the same approach to the context models deployed by [20] or [9] would be non-trivial since they are not designed for fast feed-forward computation. In particular, while the context model of [9] is also convolutional, its causality is enforced through masking the *inputs* to the network, as opposed to our masking of the weights of the networks. This means their context model needs to be run separately with a proper input cuboid for each symbol in the volume (i.e., not fully convolutionally).

4. Experiments

Architecture Our auto-encoder has a similar architecture as [17] but with more layers, and is described in Fig. 2. We adapt the number of channels K in the latent representation for different models. For the context model P, we use a simple 4-layer 3D-CNN as described in Fig. 3.

Distortion measure Following [7, 14], we use the multi-scale structural similarity index (MS-SSIM) [23] as measure of distortion $d(\mathbf{x}, \hat{\mathbf{x}}) = 100 \cdot (1 - \text{MS-SSIM}(\mathbf{x}, \hat{\mathbf{x}}))$ for our models. MS-SSIM reportedly correlates better with human perception of distortion than mean squared error (MSE). We train and test all our models using MS-SSIM.

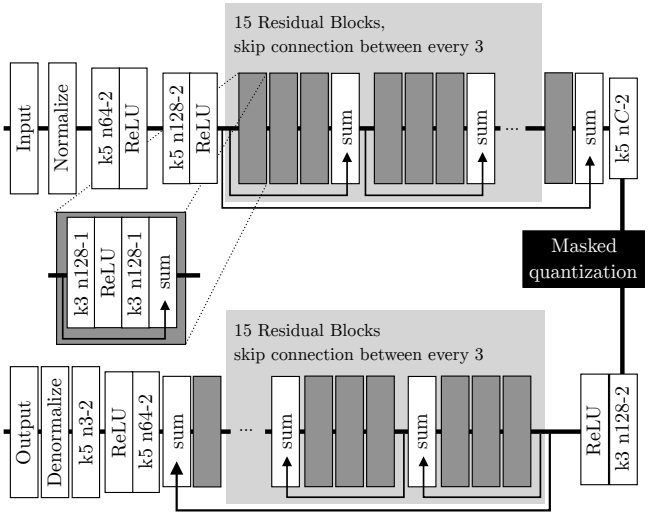

Figure 2: The architecture of our auto-encoder. Dark gray blocks represent residual units. The upper part represents the encoder E, the lower part the decoder D. For the encoder, "k5 n64-2" represents a convolution layer with kernel size 5, 64 output channels and a stride of 2. For the decoder it represents the equivalent deconvolution layer. All convolution layers are normalized using batch norm [6], and use SAME padding. *Masked quantization* is the quantization described in Section 3.4. *Normalize* normalizes the input to $[0, 1]$ using a mean and variance obtained from a subset of the training set. *Denormalize* is the inverse operation.

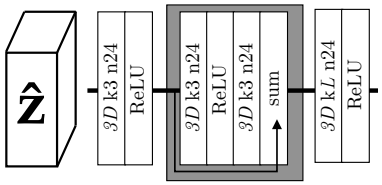

Figure 3: The architecture of our context model. "3D k3 n24" refers to a 3D *masked* convolution with filter size 3 and 24 output channels. The last layer outputs L values for each voxel in $\hat{\mathbf{z}}$.

Training We use the Adam optimizer [8] with a mini-batch size of 30 to train seven models. Each model is trained to maximize MS-SSIM directly. As a baseline, we used a learning rate (LR) of $4 \cdot 10^{-3}$ for each model, but found it beneficial to vary it slightly for different models. We set $\sigma = 1$ in the smooth approximation (3) used for gradient backpropagation through Q. To make the model more predictably land at a certain bitrate t when optimizing (1), we found it helpful to clip the rate term (i.e., replace the entropy term βH with $\max(t, \beta H)$), such that the entropy term is "switched off" when it is below t. We found this did not hurt performance. We decay the learning rate by a factor 10 every two epochs. To obtain models for different bitrates, we adapt the target bitrate t and the number of channels K, while using a moderately large $\beta = 10$. We use a small regularization on the weights and note that we achieve very stable training. We trained our models for 6 epochs, which took around 24h per model on a single GPU. For P, we use a LR of 10^{-4} and the same decay schedule.

Datasets We train on the the **ImageNet** dataset from the Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) [15]. As a preprocessing step, we take random 160×160 crops, and randomly flip them. We set aside 100 images from ImageNet as a testing set, **ImageNetTest**. Furthermore, we test our method on the widely used **Kodak** [1] dataset. To asses performance on high-quality full-resolution images, we also test on the datasets **B100** [18] and **Urban100** [5], commonly used in super-resolution.

Other codecs We compare to JPEG, using libjpeg[5], and JPEG2000, using the Kakadu implementation[6]. We also compare to the lesser known BPG[7], which is based on HEVC, the state-of-the-art in video compression, and which outperforms JPEG and JPEG2000. We use BPG in the non-default 4:4:4 chroma format, following [14].

Comparison Like [14], we proceed as follows to compare to other methods. For each dataset, we compress each image using all our models. This yields a set of (bpp, MS-SSIM) points for each image, which we interpolate to get a curve for each image. We fix a grid of bpp values, and average the curves for each image at each bpp grid value (ignoring those images whose bpp range does not include the grid value, i.e., we do not extrapolate). We do this for our method, BPG, JPEG, and JPEG2000. Due to code being unavailable for the related works in general, we digitize the Kodak curve from Rippel & Bourdev [14], who have carefully collected the curves from the respective works. With this, we also show the results of Rippel & Bourdev [14], Johnston *et al*. [7], Ballé *et al*. [4], and Theis *et al*. [17]. To validate that our estimated MS-SSIM is correctly implemented, we independently generated the BPG curves for Kodak and verified that they matched the one from [14].

Results Fig. 1 shows a comparison of the aforementioned methods for Kodak. Our method outperforms BPG, JPEG, and JPEG2000, as well as the neural network based approaches of Johnston *et al*. [7], Ballé *et al*. [4], and Theis *et al*. [17]. Furthermore, we achieve performance comparable to that of Rippel & Bourdev [14]. This holds for all bpps we tested, from 0.3 bpp to 0.9 bpp. We note that while Rippel & Bourdev and Johnston *et al*. also train to maximize (MS-)SSIM, the other methods minimize MSE.

[5] http://libjpeg.sourceforge.net/
[6] http://kakadusoftware.com/
[7] https://bellard.org/bpg/

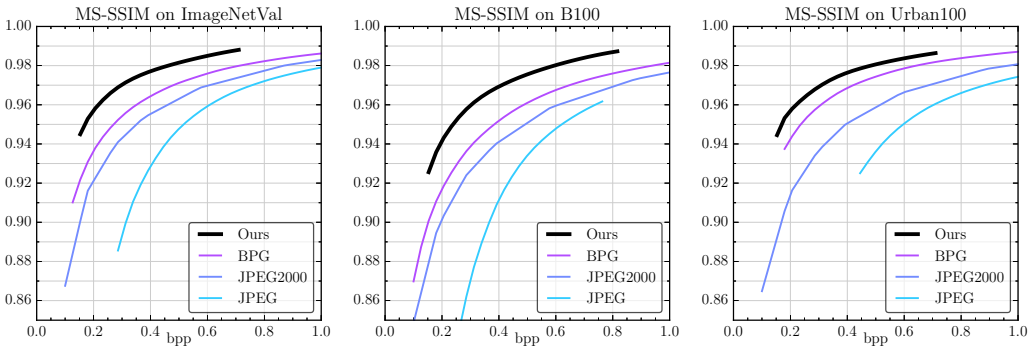

Figure 4: Performance of our approach on ImageNetTest, B100, Urban100, where we outperform BPG, JPEG and JPEG2000 in MS-SSIM.

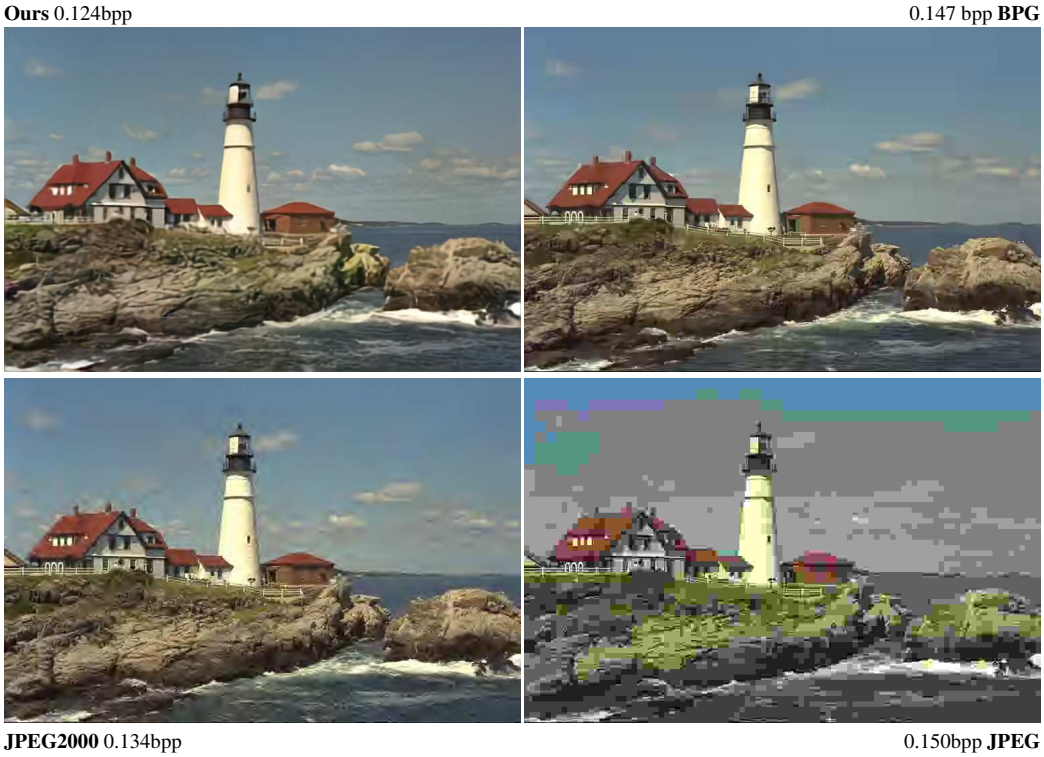

Figure 5: Example image (*kodim21*) from the Kodak testing set, compressed with different methods.

In each of the other testing sets, we also outperform BPG, JPEG, and JPEG2000 over the reported bitrates, as shown in Fig. 4.

In Fig. 5, we compare our approach to BPG, JPEG, and JPEG2000 visually, using very strong compression on *kodim21* from Kodak. It can be seen that the output of our network is pleasant to look at. Soft structures like the clouds are very well preserved. BPG appears to handle high frequencies better (see, e.g., the fence) but loses structure in the clouds and in the sea. Like JPEG2000, it produces block artifacts. JPEG breaks down at this rate. We refer to the supplementary material for further visual examples.

Ablation study: Context model In order to show the effectiveness of the context model, we performed the following ablation study. We trained the auto-encoder without entropy loss, i.e., $\beta = 0$ in (20), using $L = 6$ centers and $K = 16$ channels. On Kodak, this model yields an average MS-SSIM of 0.982, at an average rate of 0.646 bpp (calculated assuming that we need $\log_2(L) = 2.59$ bits per symbol). We then trained three different context models for this auto-encoder, while keeping the auto-encoder fixed: A *zeroth order* context model which uses a histogram to estimate the probability of each of the L symbols; a *first order* (one-step prediction) context model, which uses a conditional histogram to estimate the probability of each of the L symbols given the previous symbol (scanning $\hat{\mathbf{z}}$ in raster order); and P, i.e., our proposed context model. The resulting average rates are shown in Table 1. Our context model

reduces the rate by 10 %, even though the auto-encoder was optimized using a uniform prior (see supplementary material for a detailed comparison of Table 1 and Fig. 1).

Model	rate
Baseline (Uniform)	0.646 bpp
Zeroth order	0.642 bpp
First order	0.627 bpp
Our context model P	0.579 bpp

Table 1: Rates for different context models, for the same architecture (E, D).

Importance map As described in detail in Section 3.4, we use an importance map to dynamically alter the number of channels used at different spatial locations to encode an image. To visualize how this helps, we trained two auto-encoders M and M', where M uses an importance map and at most $K = 32$ channels to compress an image, and M' compresses without importance map and with $K = 16$ channels (this yields a rate for M' similar to that of M). In Fig. 6, we show an image from ImageNetTest along with the same image compressed to 0.463 bpp by M and compressed to 0.504 bpp by M'. Furthermore, Fig. 6 shows the importance map produced by M, as well as ordered visualizations of all channels of the latent representation for both M and M'. Note how for M, channels with larger index are sparser, showing how the model can spatially adapt the number of channels. M' uses all channels similarly.

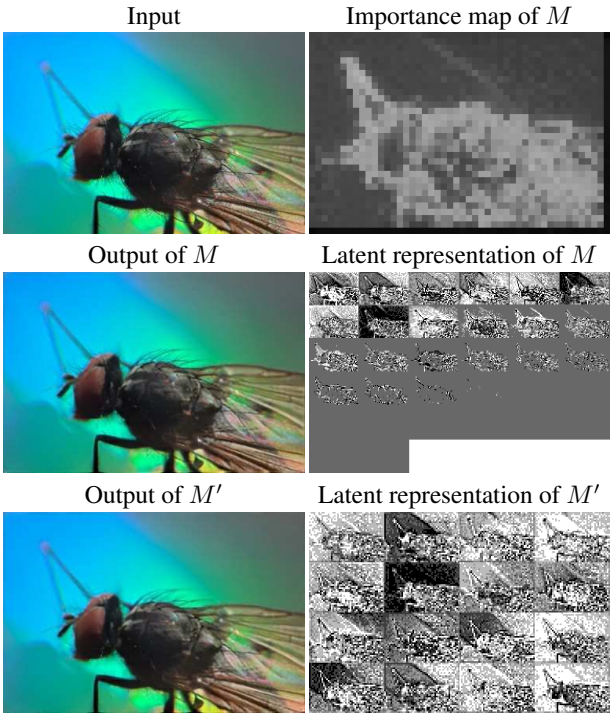

Figure 6: Visualization of the latent representation of the auto-encoder for a high-bpp operating point, with (M) and without (M') incorporating an importance map.

5. Discussion

Our experiments showed that combining a convolutional auto-encoder with a lightweight 3D-CNN as context model and training the two networks concurrently leads to a highly effective image compression system. Not only were we able to clearly outperform state-of-the-art engineered compression methods including BPG and JPEG2000 in terms of MS-SSIM, but we also obtained performance competitive with the current state-of-the-art learned compression method from [14]. In particular, our method outperforms BPG and JPEG2000 in MS-SSIM across four different testing sets (ImageNetTest, Kodak, B100, Urban100), and does so significantly, i.e., the proposed method generalizes well. We emphasize that our method relies on elementary techniques both in terms of the architecture (standard convolutional auto-encoder with importance map, convolutional context model) and training procedure (minimize the rate-distortion trade-off and the negative log-likelihood for the context model), while [14] uses highly specialized techniques such as a pyramidal decomposition architecture, adaptive codelength regularization, and multiscale adversarial training.

The ablation study for the context model showed that our 3D-CNN-based context model is significantly more powerful than the first order (histogram) and second order (one-step prediction) baseline context models. Further, our experiments suggest that the importance map learns to condensate the image information in a reduced number of channels of the latent representation without relying on explicit supervision. Notably, the importance map is learned as a part of the image compression auto-encoder concurrently with the auto-encoder and the context model, without introducing any optimization difficulties. In contrast, in [9] the importance map is computed using a separate network, learned together with the auto-encoder, while the context model is learned separately.

6. Conclusions

In this paper, we proposed the first method for learning a lossy image compression auto-encoder concurrently with a lightweight context model by incorporating it into an entropy loss for the optimization of the auto-encoder, leading to performance competitive with the current state-of-the-art in deep image compression [14].

Future works could explore heavier and more powerful context models, as those employed in [22, 21]. This could further improve compression performance and allow for sampling of natural images in a "lossy" manner, by sampling $\hat{\mathbf{z}}$ according to the context model and then decoding.

Acknowledgements This work was supported by ETH Zürich and by NVIDIA through a GPU grant.

References

[1] Kodak PhotoCD dataset. http://r0k.us/graphics/kodak/. 6

[2] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. *arXiv preprint arXiv:1704.00648*, 2017. 1, 2

[3] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimization of nonlinear transform codes for perceptual quality. *arXiv preprint arXiv:1607.05006*, 2016. 2

[4] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*, 2016. 1, 2, 6

[5] J.-B. Huang, A. Singh, and N. Ahuja. Single image super-resolution from transformed self-exemplars. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5197–5206, 2015. 6

[6] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 6

[7] N. Johnston, D. Vincent, D. Minnen, M. Covell, S. Singh, T. Chinen, S. Jin Hwang, J. Shor, and G. Toderici. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. *arXiv preprint arXiv:1703.10114*, 2017. 5, 6

[8] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 6

[9] M. Li, W. Zuo, S. Gu, D. Zhao, and D. Zhang. Learning convolutional networks for content-weighted image compression. *arXiv preprint arXiv:1703.10553*, 2017. 1, 2, 4, 5, 8

[10] 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

[11] 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, 3

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

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

[14] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2922–2930, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. 2, 5, 6, 8, 11

[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014. 6

[16] C. Shalizi. Lecture notes on stochastic processes. http://www.stat.cmu.edu/~cshalizi/754/2006/notes/lecture-28.pdf, 2006. [Online; accessed 15-Nov-2017]. 3

[17] L. Theis, W. Shi, A. Cunningham, and F. Huszar. Lossy image compression with compressive autoencoders. In *ICLR 2017*, 2017. 1, 2, 4, 5, 6

[18] R. Timofte, V. De Smet, and L. Van Gool. *A+: Adjusted Anchored Neighborhood Regression for Fast Super-Resolution*, pages 111–126. Springer International Publishing, Cham, 2015. 6

[19] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*, 2015. 1, 2

[20] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. *arXiv preprint arXiv:1608.05148*, 2016. 1, 2, 5

[21] 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. 2, 3, 5, 8

[22] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural networks. In *International Conference on Machine Learning*, pages 1747–1756, 2016. 2, 3, 5, 8

[23] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Asilomar Conference on Signals, Systems Computers, 2003*, volume 2, pages 1398–1402 Vol.2, Nov 2003. 2, 5

[24] 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

[25] 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