LEARNED COMPRESSION OF ENCODING DISTRIBUTIONS

Mateen Ulhaq and Ivan V. Bajić

School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada

ABSTRACT

The entropy bottleneck introduced by Ballé *et al.* [1] is a common component used in many learned compression models. It encodes a transformed latent representation using a static distribution whose parameters are learned during training. However, the actual distribution of the latent data may vary wildly across different inputs. The static distribution attempts to encompass all possible input distributions, thus fitting none of them particularly well. This unfortunate phenomenon, sometimes known as the amortization gap, results in suboptimal compression. To address this issue, we propose a method that dynamically adapts the encoding distribution to match the latent data distribution for a specific input. First, our model estimates a better encoding distribution for a given input. This distribution is then compressed and transmitted as an additional side-information bitstream. Finally, the decoder reconstructs the encoding distribution and uses it to decompress the corresponding latent data. Our method achieves a Bjøntegaard-Delta (BD)-rate gain of −7.10% on the Kodak test dataset when applied to the standard fully-factorized architecture. Furthermore, considering computational complexity, the transform used by our method is an order of magnitude cheaper in terms of Multiply-Accumulate (MAC) operations compared to related side-information methods such as the scale hyperprior.

Index Terms— Learned compression, entropy bottleneck, entropy model, probabilistic modeling, side information

1. INTRODUCTION

In this paper, we explore the compression of probability distributions that are used in compression. Specifically, we will focus on probability distributions that are used to encode the latent representations produced by an image compression model. Entropy models compress latents using *encoding distributions*, which are discrete probability mass functions (pmfs) used for encoding. For example, Ballé *et al.* [1] encode a latent representation $\boldsymbol{y}$ using the following entropy models:

1. A "fully-factorized" *entropy bottleneck*. Each element $y_{c,i}$ within the c-th channel is encoded using a single channel-specific non-parametric encoding distribution $p_c(y)$, which is typically fixed once training completes.
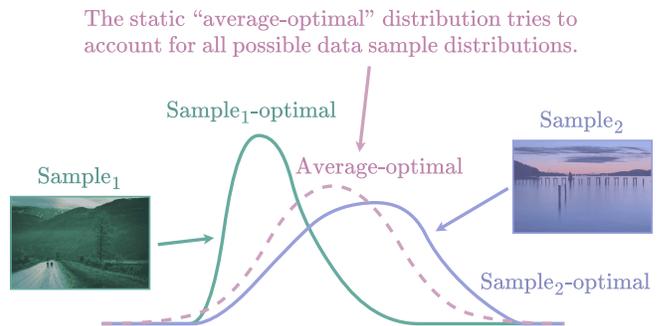

Fig. 1. Visualization of the suboptimality of using a single static encoding distribution. This distribution is optimal, on average, among all static distributions, but it is suboptimal for any specific data instance.

2. A *Gaussian conditional*. For each element y_i, the parameters μ_i and σ_i of its encoding distribution are computed, conditioned on the latent $\hat{\boldsymbol{z}} = \mathrm{Quantize}[h_a(\boldsymbol{y})]$. In later works [2], a Gaussian mixture model (GMM) is sometimes used instead, with parameters $w_i^{(k)}$, $\mu_i^{(k)}$, and $\sigma_i^{(k)}$ for each Gaussian in the mixture.

Of these two, the entropy bottleneck is more flexible at modeling arbitrary distributions. However, because all elements within a given channel are modelled using the same distribution, it is quite poor at specializing and adapting to specific input distributions. Instead, it models a "conservative" distribution that tries to encompass all possible inputs, which is often suboptimal, as visualized in Fig. 1. This is sometimes known as the *amortization gap* [3, 4]. In contrast, the Gaussian conditional is better at adapting to a specific input, though it is limited to modeling Gaussian distributions. Furthermore, it must also pay an additional cost for transmitting the distribution parameters, though this is often worth the trade-off, e.g., a savings of roughly 40% for a 10% increase in rate in [1].

In this paper, we propose a method to address this amortization gap via the compression of an input-specific encoding distribution. This allows our adaptive entropy bottleneck to adapt to the input distribution, rather than using a static dataset-optimized encoding distribution which suffers from the amortization gap. We demonstrate that our low-cost input-adaptive method can be used by models containing the entropy bottleneck in order to achieve better rate-distortion performance.

2. RELATED WORKS

In their "scale hyperprior" architecture, Ballé *et al.* [1] utilize an entropy bottleneck to compress the latent representation $\boldsymbol{z} = h_a(\boldsymbol{y})$. This is known as "side-information," and its compressed bitstream is transmitted before the bitstream for $\boldsymbol{y}$. The reconstructed latent $\hat{\boldsymbol{z}}$ is then used to determine the parameters[1] $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of a Gaussian conditional distribution which is used to reconstruct $\hat{\boldsymbol{y}}$. In many such models that utilize the scale hyperprior, the rate usage of the entropy bottleneck often comprises roughly 10% of the total rate. Evidently, the entropy bottleneck is a key component of many state-of-the-art (SOTA) image compression models, and so it is important to address the amortization gap suffered by the entropy bottleneck.

Balcilar *et al.* [3] propose non-learned methods for the compression and transmission of encoding distributions. For the factorized entropy model, they first measure a discrete histogram of the c-th latent channel. Then, they construct an approximation of the histogram using a K-Gaussian mixture model parameterized by $w^{(k)}$, $\mu^{(k)}$, and $\sigma^{(k)}$. The aforementioned parameters are compactly encoded into 8 bits each, and transmitted. That is, for a model using C activated channels, $(3K - 1) \cdot C \cdot 8$ bits are required to transmit the parameters. (Typically, $K \in [1, 3]$ is used, and $C \in [16, 192]$ for low-rate models.) The approximation model is then used as the encoding distribution $p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}_c) = \sum_{k=1}^{K} w_k \ \hat{\mathcal{N}}(\hat{\boldsymbol{y}}_c; \mu_k, \sigma_k{}^2)$. For the Gaussian conditional entropy model, they instead transmit a factor $\beta^{(c)}$ that corrects the error in the center (and most likely) bin's probability. A key difference between our work and this prior work is that we formulate a learnable method for the compression of the encoding distribution.

Galpin *et al.* [6] introduce a lightweight non-learned context model for raster-scan ordered encoding, similar to Minnen *et al.* [5]. They compare their method in conjunction with [7, 3] in various configurations on both the standard factorized model and its GDN $\rightarrow$ ReLU replacement variant. Our work is also broadly related to source instance overfitting for compression [8, 9]. However, unlike these approaches, we do not overfit the neural network, but rather the latent distribution of the source instance, which is computationally more feasible.

3. PROPOSED METHOD

3.1. Compression of encoding distributions

Let $\boldsymbol{x}$ be an input image, and $\boldsymbol{y} = g_a(\boldsymbol{x})$ be its transformed latent representation, and $\hat{\boldsymbol{y}} = \text{Quantize}[\boldsymbol{y}]$ be its quantized version. Consider a fully-factorized entropy model [1], which models the j-th channel of $\hat{\boldsymbol{y}}$ (denoted as $\hat{\boldsymbol{y}}_j$) using a single channel-specific encoding distribution $\hat{\boldsymbol{p}}_j$. In this setting, all elements within a given channel are modelled using the same encoding distribution. Then, the "true" distribution $\boldsymbol{p}_j$ can be

---

[1] In their initial publication, $\boldsymbol{\mu}$ is fixed to 0, though later results [5] show that it is better to predict $\boldsymbol{\mu}$ as well.
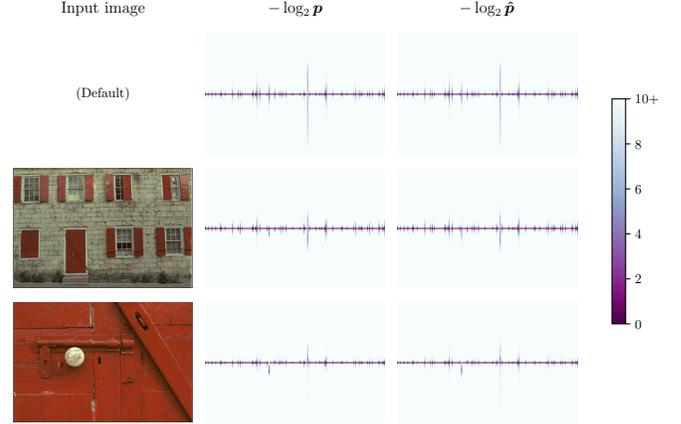

Fig. 2. Visualization of target ($\boldsymbol{p}$) and reconstructed ($\hat{\boldsymbol{p}}$) encoding distributions. Our proposed method reconstructs $\hat{\boldsymbol{p}}$, which is then used by the fully-factorized entropy model to encode the latent derived from a given input image. Each collection of distributions is visualized as a color plot, with channels varying along the x-axis, bins varying along the y-axis, and negative log-likelihoods represented by the z-axis (i.e., color).

determined exactly by computing the normalized histogram of $\hat{\boldsymbol{y}}_j$. Notably, $\boldsymbol{p}_j$ is also the distribution with the minimum rate cost for encoding $\hat{\boldsymbol{y}}_j$ using the entropy model. That is, the ideal encoding distribution $\hat{\boldsymbol{p}}_j^*$ is given by

$$\hat{\boldsymbol{p}}_j^* = \operatorname*{arg\,min}_{\hat{\boldsymbol{p}}_j} \left[H(\boldsymbol{p}_j) + D_{\text{KL}}(\boldsymbol{p}_j \parallel \hat{\boldsymbol{p}}_j) \right] \implies \hat{\boldsymbol{p}}_j^* = \boldsymbol{p}_j.$$

Unfortunately, using the true distribution $\boldsymbol{p}_j$ for encoding is infeasible, since the decoder does not have access to it. Instead, we propose a method that generates a reconstructed approximation $\hat{\boldsymbol{p}}$ that targets $\boldsymbol{p}$, subject to a rate trade-off.

Fig. 2 visualizes various collections of encoding distributions used for encoding different input images using a fully-factorized entropy model. The left column shows various input images — except for the "(Default)" image, which is an abstract amortized representation of randomly sampled images from the training dataset. The middle and right columns visualize the negative log-likelihoods (in bits) of the true ($\boldsymbol{p}$) and reconstructed ($\hat{\boldsymbol{p}}$) encoding distributions, respectively. Each collection of distributions is visualized as a color plot, where

(i) the x-coordinate is the channel index j of the latent $\boldsymbol{y}$,

(ii) the y-coordinate is the bin index i of the encoding distribution, and

(iii) the z-coordinate (i.e., color intensity) is the negative log-likelihood of $(\boldsymbol{p}_j)_i$ or $(\hat{\boldsymbol{p}}_j)_i$, clipped within $[0, 10]$.

As shown, the default amortized encoding distributions account for a wide range of possible input distributions. However, the ideal encoding distributions for any given image often have much narrower ranges. The average cost of encoding a randomly drawn sample from a distribution $\boldsymbol{p}$ is equal to the

cross-entropy between that distribution and the one used for encoding, $\hat{p}$. Thus, a mismatch between the two leads to excessive rate usage. In contrast, with our proposed method, the encoding distributions more closely approximate the ideal distributions for encoding a specific image. Our method balances the trade-off between the rate cost of encoding the encoding distributions $\hat{p}$ and the rate cost of encoding the latent representation $\hat{y}$ using the encoding distributions. Indeed, our method accurately reconstructs regions of high probability, which contribute the most towards saving rate.
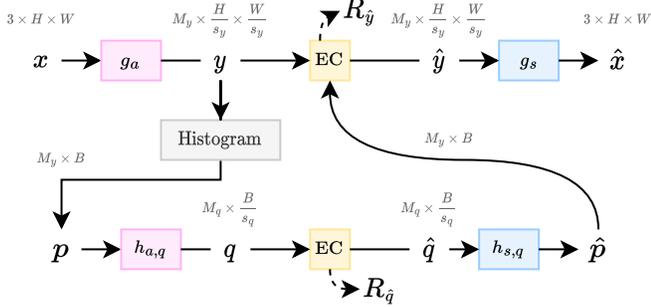
3.2. Architecture overview

Fig. 3. Adaptive encoding distribution architecture.

Our proposed method applied to the traditional "fully-factorized" architecture [1] is visualized in Fig. 3. In this configuration, the latent representation $\boldsymbol{y}$ is first passed through the histogram layer described in Section 3.3, which estimates the target distribution $\boldsymbol{p}$. Then, $\boldsymbol{p}$ is passed through a distribution compression architecture consisting of the analysis transform $h_{a,q}$ which generates a latent representation $\boldsymbol{q}$, followed by an entropy model (e.g., entropy bottleneck), and then finally a synthesis transform $h_{s,q}$ which reconstructs the distribution $\hat{\boldsymbol{p}}$. The rate cost of encoding $\boldsymbol{q}$ is labelled R_q, and the rate cost of encoding $\boldsymbol{y}$ using $\hat{\boldsymbol{p}}$ is labelled R_y.

Like the scale hyperprior, our proposed method also requires the transmission of side-information from which encoding distributions are reconstructed. However, the scale hyperprior does most of its computation within the spatial dimensions, whereas our method initially collapses the spatial dimensions into a histogram. This allows our method to be more computationally efficient when the number of bins and encoding distributions is small.

3.3. Histogram estimation

To adapt the encoding distribution to the input, we must first determine the ideal target encoding distribution. We do this by estimating the histogram for a given latent channel. To ensure differentiability backwards through the histogram module, we use the method of kernel density estimation (KDE) [10].

Let $[y_{\min}, y_{\max}]$ be a supporting domain that encompasses all values $y_1, y_2, \ldots, y_N$ in a specific channel. Let $\{b_1, b_2, \ldots, b_B\}$ denote the bin centers of the B-bin histogram, where $b_i = y_{\min} + i - 1$. In the method of kernel density estimation, which is used to estimate the probability density function from a set of observations, a kernel function is placed centered at each observed sample, and those functions are summed to create the overall density function. To construct the histogram, we evaluate the kernel density estimate at each b_i. Or rather, we use an equivalent formulation where we instead place a kernel function at each bin center. Then, the probability mass in the i-th bin can be estimated as

$$(\text{Hist}(\boldsymbol{y}))_i = \frac{1}{\text{Normalization}} \sum_{n=1}^{N} K\left(\frac{y_n - b_i}{\Delta b}\right),$$

where $\Delta b = b_{i+1} - b_i$ is the bin width, and $K(u)$ is the kernel function, which we choose to be the triangular function

$$K_{\text{soft}}(u) = \max\{0, 1 - |u|\}.$$

Conveniently, the triangular kernel function ensures that the total mass of a single observation sums to 1. Thus, to ensure that the total mass of the histogram under all N observations sums to 1, we may simply choose $\text{Normalization} = N$.

While the above provides a piecewise differentiable estimate of the histogram, it is also possible to determine the exact *hard* histogram by using a rectangular kernel function, which allocates y_n to its nearest bin:

$$K_{\text{hard}}(u) = \text{rect}(u).$$

We can then use the "straight-through" estimator (STE) [11] trick to get the benefits of both:

$$\text{Hist}(\boldsymbol{y}) = \text{Hist}_{\text{soft}}(\boldsymbol{y}) + \text{detach}(\text{Hist}_{\text{hard}}(\boldsymbol{y}) - \text{Hist}_{\text{soft}}(\boldsymbol{y})).$$

Thus, the hard histogram is used for the forward pass, and the soft histogram for the backward pass (i.e., backpropagation).

3.4. Loss function

Our model follows the variational formulation based on [1]; further details on the construction may be found in earlier works such as [12, 13, 14]. Briefly, for a variable $\boldsymbol{y}$, its counterparts $\tilde{\boldsymbol{y}}$ and $\hat{\boldsymbol{y}}$ denote the noise-modelled (for training) and quantized (for inference) versions of $\boldsymbol{y}$, respectively. Furthermore, the variational encoder $q_\phi(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{q}} \mid \boldsymbol{x})$ with trainable parameters ϕ is a probabilistic model of the joint distribution for $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{q}}$, for a given input $\boldsymbol{x}$. Then, the loss function that we seek to minimize over the input data distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$ is

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x})} \mathbb{E}_{\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{q}} \sim q_\phi(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{q}} | \boldsymbol{x})} \begin{bmatrix} -\log p_{\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{q}}}(\tilde{\boldsymbol{y}} \mid \tilde{\boldsymbol{q}}) \\ -\lambda_q \log p_{\tilde{\boldsymbol{q}}}(\tilde{\boldsymbol{q}}) \\ +\lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \end{bmatrix}$$

$$= \mathbb{E}_{\boldsymbol{x} \sim p_{\boldsymbol{x}}(\boldsymbol{x})} \left[R_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}) + \lambda_q R_{\tilde{\boldsymbol{q}}}(\boldsymbol{x}) + \lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \right].$$

For a given input $\boldsymbol{x}$, $R_{\tilde{\boldsymbol{y}}}(\boldsymbol{x})$ and $R_{\tilde{\boldsymbol{q}}}(\boldsymbol{x})$ are the rate costs of $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{q}}$, respectively. Furthermore, $D(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ is a distortion metric — typically mean-squared error (MSE) — between the input $\boldsymbol{x}$ and its reconstruction $\tilde{\boldsymbol{x}}$. Additionally, λ_x is a trade-off hyperparameter between the rate and distortion. And lastly, λ_q is a trade-off hyperparameter between the rate cost of encoding $\tilde{\boldsymbol{q}}$ against the amount of rate saved when using $\tilde{\boldsymbol{q}}$ to encode $\tilde{\boldsymbol{y}}$.

The choice for λ_q depends upon the ratio between the target and trained-upon input dimensions:

$$\lambda_q = \frac{H_{\boldsymbol{x},\text{trained}} W_{\boldsymbol{x},\text{trained}}}{H_{\boldsymbol{x},\text{target}} W_{\boldsymbol{x},\text{target}}}.$$

For instance, for the target dimension of 768×512 (i.e., images from the Kodak test dataset) and a trained-upon dimension of 256×256 (i.e., image patches from the Vimeo-90K dataset), we set $\lambda_q = \frac{1}{6}$.

Let $\boldsymbol{p}_j = (p_{j1}, \ldots, p_{jB})$ refer to the discrete B-bin probability distribution used to encode the j-th channel of $\tilde{\boldsymbol{y}}$. Then concretely, the rate cost of $\tilde{\boldsymbol{y}}$ is the cross-entropy between the true and reconstructed[2] discretized distributions,

$$R_{\tilde{\boldsymbol{y}}} = \sum_{j=1}^{M} \sum_{i=1}^{B} -p_{ji} \log \hat{p}_{ji}.$$

Similarly, let $\tilde{\boldsymbol{q}}_j$ denote the j-th channel of $\tilde{\boldsymbol{q}}$. Then, the rate cost of $\tilde{\boldsymbol{q}}$ is measured by the typical calculation,

$$R_{\tilde{\boldsymbol{q}}} = \sum_{j=1}^{M} \sum_{i=1}^{\text{length}(\boldsymbol{l}_j)} -l_{ji} \log l_{ji}, \quad \text{where } \boldsymbol{l}_j = p_{\tilde{\boldsymbol{q}}_j}(\tilde{\boldsymbol{q}}_j).$$

3.5. Optimization

We will now compute the derivatives for our proposed method. To simplify the notation, we will focus on a single channel — the j-th channel — and directly denote $\tilde{\boldsymbol{y}}_j$ as $\boldsymbol{y}$, and $\boldsymbol{p}_j$ as $\boldsymbol{p}$. From the perspective of the backward pass,

$$p_i = (\text{Hist}(\boldsymbol{y}))_i = \frac{s^2}{HW} \sum_n K_{\text{soft}} \left(\frac{y_n - b_i}{\Delta b} \right)$$
$$= \frac{s^2}{HW} \sum_n \max \left\{ 0, 1 - \left| \frac{y_n - b_i}{\Delta b} \right| \right\},$$

where s is the downscale factor (e.g., $s = 2^4$), and the dimension of the input image $\boldsymbol{x}$ is $H \times W$. And so,

$$\frac{\partial p_i}{\partial y_k} = \frac{s^2}{HW} \sum_n \frac{\partial}{\partial y_k} \left[\max \left\{ 0, 1 - \left| \frac{y_n - b_i}{\Delta b} \right| \right\} \right]$$
$$= \frac{s^2}{HW} \frac{\partial}{\partial y_k} \left[\max \left\{ 0, 1 - \left| \frac{y_k - b_i}{\Delta b} \right| \right\} \right]$$
$$= \frac{s^2}{HW} \frac{-1}{\Delta b} \cdot \underbrace{H\left(1 - \left| \frac{y_k - b_i}{\Delta b} \right| \right)}_{1 \text{ if } |y_k - b_i| < \Delta b \text{ else } 0} \cdot \underbrace{\left(2 \cdot H\left(\frac{y_k - b_i}{\Delta b} \right) - 1 \right)}_{-1 \text{ if } y_k < b_i \text{ else } 1}.$$

---
[2]Since the reconstructed distribution is determined directly from $\boldsymbol{y}$ rather than from $\tilde{\boldsymbol{y}}$, we denote it by $\hat{\boldsymbol{p}}$, not $\tilde{\boldsymbol{p}}$.

The corresponding derivative for the reconstructed distribution $\hat{\boldsymbol{p}}$ may be determined as

$$\frac{\partial \hat{p}_i}{\partial y_k} = \sum_l \frac{\partial \hat{p}_i}{\partial p_l} \frac{\partial p_l}{\partial y_k} \approx \sum_l \delta_{il} \frac{\partial p_l}{\partial y_k} = \frac{\partial p_i}{\partial y_k},$$

where we have assumed $\frac{\partial \hat{p}_i}{\partial p_l} \approx \delta_{il}$, and that the dominant term in the sum is the term where $l = i$.

The rate cost of encoding the channel $\boldsymbol{y}$ with its associated encoding distribution $\boldsymbol{p}$ is given by

$$R_y = \frac{HW}{s^2} \sum_i -p_i \log \hat{p}_i.$$

Then, we may compute the derivative as follows:

$$\frac{\partial R_y}{\partial y_k} = \frac{-HW}{s^2} \sum_i \frac{\partial}{\partial y_k} [p_i \log \hat{p}_i]$$
$$= \frac{-HW}{s^2} \sum_i \left[\frac{p_i}{\hat{p}_i} \frac{\partial \hat{p}_i}{\partial y_k} + \frac{\partial p_i}{\partial y_k} \log \hat{p}_i \right]$$
$$\approx \frac{-HW}{s^2} \sum_i \left[\frac{p_i}{\hat{p}_i} \frac{\partial p_i}{\partial y_k} + \frac{\partial p_i}{\partial y_k} \log \hat{p}_i \right] \quad \text{assume } \frac{\partial \hat{p}_i}{\partial y_k} \approx \frac{\partial p_i}{\partial y_k}$$
$$= \frac{-HW}{s^2} \sum_i \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i \right] \frac{\partial p_i}{\partial y_k}$$
$$= \frac{-HW}{s^2} \sum_{i \in \{\lfloor y_k - y_{\min} \rfloor + 1, \lceil y_k - y_{\min} \rceil + 1\}} \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i \right] \frac{\partial p_i}{\partial y_k}$$
$$= \frac{-1}{\Delta b} \left(\left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i \right]_{i = \lceil y_k - y_{\min} \rceil + 1} - \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i \right]_{i = \lfloor y_k - y_{\min} \rfloor + 1} \right)$$
$$\approx \frac{-1}{\Delta b} \left(\log \hat{p}_{\lceil y_k - y_{\min} \rceil + 1} - \log \hat{p}_{\lfloor y_k - y_{\min} \rfloor + 1} \right) \quad \text{assume } \hat{p}_i \approx p_i. \tag{1}$$

Thus, assuming that $\hat{p}_i \approx p_i$ and $\frac{\partial \hat{p}_i}{\partial y_k} \approx \frac{\partial p_i}{\partial y_k}$, the gradient of the rate cost R_y is directly proportional to the difference in code lengths of the two bins whose centers are nearest to the value y_k. Intuitively, this makes sense, since the rate cost of encoding the k-th element is directly proportional to the linear interpolation between the code lengths of the two nearest bins:

$$R_{y_k} = -\alpha \cdot \log \hat{p}_{\lfloor y_k - y_{\min} \rfloor + 1} - (1 - \alpha) \cdot \log \hat{p}_{\lceil y_k - y_{\min} \rceil + 1},$$

where $\alpha = (y_k - b_{\lfloor y_k - y_{\min} \rfloor + 1}) / \Delta b$. The linear interpolation may be justified by the fact that $\hat{y}_k$ inhabits only a single bin at a time. The probability of $\hat{y}_k$ inhabiting the left bin is α, and the probability of inhabiting the right bin is $1 - \alpha$. This aligns with Shannon's measure of entropy, which is the expected value of the code length.

For comparison, the standard "entropy bottleneck" represents the likelihood of a symbol by

$$p(y) = c(y + 1/2) - c(y - 1/2) = \int_{y-1/2}^{y+1/2} f(t) \, dt,$$

where c is a cumulative distribution function and $f(y) = \frac{d}{dy}c(y)$ is a probability density function, from which the encoding distribution can be determined by integrating over the bin. Then, the rate cost for the "entropy bottleneck" is merely the sum of the negative log-likelihoods (i.e., code lengths),

$$R_y = \sum_i -\log p(y_i).$$

We may then compute its derivative as follows:

$$\frac{\partial R_y}{\partial y_k} = -\sum_i \frac{\partial}{\partial y_k}\left[\log p(y_i)\right] = -\frac{\partial}{\partial y_k}\left[\log p(y_k)\right] \quad (2)$$
$$= -[p(y_k)]^{-1} \cdot [f(y_k + 1/2) - f(y_k - 1/2)].$$

Interestingly, whereas the derivative for the proposed method (under the assumption that $\hat{p}_i \approx p_i$) computed in (1) contains the difference between the "right" and "left" log-likelihoods, the derivative for the entropy bottleneck computed in (2) contains the difference between the "right" and "left" evaluations of the probability density function.

4. EXPERIMENTAL SETUP

4.1. Architecture details

As shown in Fig. 4, our $h_{a,q}$ and $h_{s,q}$ are implemented using a simple five-layer convolutional neural network. In between each of the convolutional layers shown is a ReLU activation function, as well as a channel shuffle operation, as is done in ShuffleNet [15]. There are two strides of length 2, resulting in a total downscaling factor of $s_q = 4$. For all our models, we set $K = 15$ to control the kernel sizes, and $G = 8$ to control the number of channel groups. Furthermore, we set $(N_q, M_q) = (32, 16)$ for low-resolution models, and $(N_q, M_q) = (64, 32)$ for high-resolution models. The number of bins B is set between 128 and 1024 across different models. We have elected to use an entropy bottleneck design for simplicity, though one can likely further improve the compression performance of the encoding distribution compression architecture by using more powerful entropy modeling techniques (e.g., a scale hyperprior). Since λ_q is a parameter that depends on the ratio between the target and trained-upon input dimensions, it may also be advisable to train a single model that supports λ-rate control strategies (e.g., G-VAE [16, 17] and QVRF [18]) for both latent representations y, q. However, we have not yet explored this possibility.

4.2. Training details

Our models are trained on 256×256 image patches from the Vimeo-90K triplet dataset [19]. A training batch size of 16 was used, along with the Adam optimizer [20] with an initial learning rate of 10^{-4} that was decayed by a factor of 0.1 whenever the validation loss plateaued. Specifically, we loaded
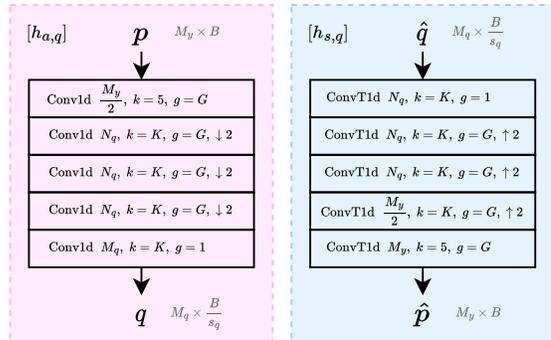

Fig. 4. Architecture layer diagram for $h_{a,q}$ and $h_{s,q}$ transforms. k denotes kernel size, g denotes number of channel groups, and $\downarrow, \uparrow$ denote stride.

the weights of the pretrained models from CompressAI [21], which were also trained using the same setup as above. We replaced the static distributions of the `EntropyBottleneck` module with the dynamically generated adaptive distributions from our proposed encoding distribution compression module. Then, we froze the weights for g_a and g_s, and trained only the weights for our encoding distribution compression model (i.e., for $h_{a,q}$ and $h_{s,q}$, and the entropy model for q). Finally, we evaluated our models on the standard Kodak test dataset [22] containing 24 images of size 768×512.

5. EXPERIMENTAL RESULTS

Fig. 5 shows the rate-distortion (RD) curves comparing a given base model against the same model enhanced with an encoding distribution compression method.

In Table 1, we compare the rate savings of our proposed method against the base model at various quality levels. Additionally, we also list the maximum rate savings that can be
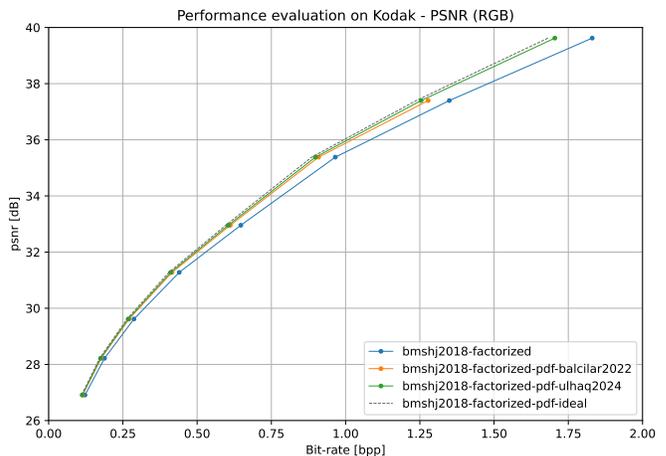

Fig. 5. RD curves for the Kodak dataset.

theoretically achieved by perfectly eliminating the amortization gap between the true and reconstructed distributions, i.e., by using the true distribution directly as the encoding distribution with zero additional transmission cost. As shown, our approach achieves a 7.10% BD-rate reduction over the base model, in comparison to a maximum possible reduction of 8.21% BD-rate reduction achievable by perfectly eliminating the amortization gap.

Table 1. Potential and achieved rate savings for the bmshj2018-factorized model [1] when equipped with our proposed distribution compression method.

Quality	Original (bpp)	Potential gain (bpp)	Potential gain (%)	Our (bpp)	Our gain (bpp)	Our gain (%)
1	0.122	-0.012	-9.45	0.113	-0.010	-8.06
2	0.188	-0.016	-8.66	0.174	-0.014	-7.53
3	0.287	-0.024	-8.20	0.267	-0.020	-7.13
4	0.440	-0.035	-7.90	0.409	-0.031	-6.95
5	0.647	-0.051	-7.83	0.602	-0.045	-6.98
6	0.965	-0.080	-8.25	0.898	-0.067	-6.95
7	1.349	-0.111	-8.24	1.254	-0.095	-7.04
8	1.830	-0.149	-8.14	1.705	-0.126	-6.88
*			-8.21			-7.10

In order to compute the potential savings, we first ran the base model on each image from the Kodak test dataset [22], giving us a collection of true distributions $\boldsymbol{p}$ for each image. Furthermore, the base model provides a static encoding distribution $\boldsymbol{p}_{\text{default}}$. Then, we computed the potential savings (in bpp) for each image as $(\Delta R)_{\max} = \frac{1}{s^2} \sum_j D_{\text{KL}}(\boldsymbol{p}_j \parallel (\boldsymbol{p}_{\text{default}})_j)$. We then averaged these values across the dataset.

In Table 2, we compare the rate savings for various base models equipped with a distribution compression method. "Ratio" is the fraction of the total rate occupied by the original unmodified model. "Gap" is the maximum potential rate gain with perfect distribution reconstruction and zero additional transmission cost. "Gain" is the actual rate gain achieved by the model. All models are evaluated on the Kodak dataset [22].

Table 2. Comparison of rate savings for various models.

Model	Quality	Factorized Ratio (%)	Factorized Gap (%)	Factorized Gain (%)	Total Gain (%)
bmshj2018-factorized [1] + Balcilar2022 [3]	1	100	-9.45	-6.79	-6.79
bmshj2018-factorized [1] + ours	1	100	-9.45	-8.06	-8.06
bmshj2018-factorized [1] + Balcilar2022 [3]	*	100	-8.21	-5.74	-5.74
bmshj2018-factorized [1] + ours	*	100	-8.21	-7.10	-7.10

Table 3 reports the number of trainable parameters and the number of multiply-accumulate operations (MACs) per pixel for various model configurations. The results are calculated assuming an input image size of 768×512, and a latent representation size of 48×32. As shown, our model requires far fewer parameters and MACs/pixel than the comparable scale

hyperprior [1] model. In particular, for comparable configurations, our method's $h_{a,q}$ and $h_{s,q}$ transforms require 96–97% fewer parameters and 96–99% fewer MACs/pixel than the scale hyperprior's h_a and h_s transforms.

Table 3. Trainable parameter counts and number of multiply-accumulate operations (MACs) per pixel.

Model configuration	Params	MAC/px	Params	MAC/px
(M_y, N_q, M_q, K, G, B)	$h_{a,q}$		$h_{s,q}$	
Ours $(192, 32, 16, 15, 8, 256)$	0.029M	10	0.029M	10
Ours $(320, 64, 32, 15, 8, 1024)$	0.097M	126	0.097M	126
(N, M)	h_a		h_s	
bmshj2018-hyperprior [1] $(128, 192)$	1.040M	1364	1.040M	1364
bmshj2018-hyperprior [1] $(192, 320)$	2.396M	3285	2.396M	3285

6. CONCLUSION

In this paper, we proposed a learned method for the compression of encoding distributions. Our method effectively measures and compresses the encoding distributions used by the entropy bottleneck. The experiments we performed where we only trained the distribution compression component show that this method is effective at significantly reducing the amortization gap. Since many learned compression models use the entropy bottleneck component, our method provides them with a low-cost improvement in bitrate. Furthermore, our work opens up the possibility of using learned distribution compression as a paradigm for correcting encoding distributions.

6.1. Future work

A few steps remain towards making encoding distribution corrective methods viable parts of more advanced entropy models such as [1, 2, 23]. These include the following:

- The proposed adaptive entropy bottleneck needs to be formulated in such a way that it can be trained fully end-to-end. When training a pretrained base model equipped with our adaptive distribution compression module, we found that unfreezing the pretrained transform weights led to a degradation in RD performance.

- Application of our adaptive distribution method to the Gaussian conditional component of entropy models. This component (along with the entropy bottleneck) is used to construct entropy modeling methods such as those used in [1, 2, 23]. Most SOTA learned image compression models predict the location and scale parameters of the Gaussian encoding distributions. There are potential rate improvements to be made by adapting the shapes of the encoding distributions to those that more closely match the data distribution.

7. REFERENCES

[1] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston, "Variational image compression with a scale hyperprior," in *Proc. ICLR*, 2018.

[2] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Proc. IEEE/CVF CVPR*, 2020, pp. 7939–7948.

[3] Muhammet Balcilar, Bharath Damodaran, and Pierre Hellier, "Reducing the amortization gap of entropy bottleneck in end-to-end image compression," in *Proc. Picture Coding Symposium (PCS)*. IEEE, 2022, pp. 115–119.

[4] Chris Cremer, Xuechen Li, and David Duvenaud, "Inference suboptimality in variational autoencoders," in *Int. Conf. on Machine Learning (ICML)*. PMLR, 2018, pp. 1078–1086.

[5] David Minnen, Johannes Ballé, and George D Toderici, "Joint autoregressive and hierarchical priors for learned image compression," *Advances in neural information processing systems*, vol. 31, 2018.

[6] Franck Galpin, Muhammet Balcilar, Frédéric Lefebvre, Fabien Racapé, and Pierre Hellier, "Entropy coding improvement for low-complexity compressive autoencoders," in *Proc. IEEE DCC*, 2023, pp. 338–338.

[7] Joaquim Campos, Simon Meierhans, Abdelaziz Djelouah, and Christopher Schroers, "Content adaptive optimization for neural image compression," in *Proc. IEEE/CVF CVPR Workshops*, 2019.

[8] Ties van Rozendaal, Iris A. M. Huijben, and Taco S. Cohen, "Overfitting for fun and profit: Instance-adaptive data compression," in *Proc. ICLR*, 2021, pp. 1–18.

[9] Honglei Zhang, Francesco Cricri, Hamed Rezazadegan Tavakoli, Maria Santamaria, Yat-Hong Lam, and Miska M. Hannuksela, "Learn to overfit better: finding the important parameters for learned image compression," in *Proc. IEEE VCIP*, 2021, pp. 1–5.

[10] Emanuel Parzen, "On Estimation of a Probability Density Function and Mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065–1076, 1962.

[11] Yoshua Bengio, Nicholas Léonard, and Aaron Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013.

[12] Diederik P Kingma and Max Welling, "Auto-encoding variational bayes," 2013.

[13] Johannes Ballé, Valero Laparra, and Eero P. Simoncelli, "End-to-end optimized image compression," in *Proc. ICLR*, 2017.

[14] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár, "Lossy image compression with compressive autoencoders," in *Proc. ICLR*, 2017.

[15] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF CVPR*, 2018, pp. 6848–6856.

[16] Ze Cui, Jing Wang, Shangyin Gao, Bo Bai, Tiansheng Guo, and Yihui Feng, "G-VAE: A continuously variable rate deep image compression framework," 2020.

[17] Ze Cui, Jing Wang, Shangyin Gao, Tiansheng Guo, Yihui Feng, and Bo Bai, "Asymmetric gained deep image compression with continuous rate adaptation," in *Proc. IEEE/CVF CVPR*, 2022, pp. 10532–10541.

[18] Kedeng Tong, Yaojun Wu, Yue Li, Kai Zhang, Li Zhang, and Xin Jin, "QVRF: A quantization-error-aware variable rate framework for learned image compression," in *Proc. IEEE ICIP*, 2023.

[19] Tianfan Xue, Baian Chen, Jiajun Wu, Donglai Wei, and William T Freeman, "Video enhancement with task-oriented flow," *Int. J. Comput. Vis. (IJCV)*, vol. 127, pp. 1106–1125, 2019.

[20] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2015.

[21] Jean Bégaint, Fabien Racapé, Simon Feltman, and Akshay Pushparaja, "CompressAI: a PyTorch library and evaluation platform for end-to-end compression research," 2020.

[22] Eastman Kodak, "Kodak lossless true color image suite (PhotoCD PCD0992)," .

[23] Dailan He, Ziming Yang, Weikun Peng, Rui Ma, Hongwei Qin, and Yan Wang, "ELIC: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding," in *Proc. IEEE/CVF CVPR*, 2022, pp. 5718–5727.