Learned Lossless Image Compression based on Bit Plane Slicing

Supplementary Material

Table 5. Ad hoc test of generalizability by evaluating the dataset compression performance on three low-resolution datasets with a model trained on ImageNet32.

Method	CIFAR10	ImageNet32	ImageNet64
IDF [13]	3.60	4.18	3.94
iVPF [42]	3.49	4.03	3.79
SHVC [28]	3.41	3.98	3.71
iFlow [41]	3.36	3.88	3.65
ArIB-BPS (ours)	3.38	3.91	3.67

A. Details In Experiment

A.1. Implementation Details

We split the last 5,000, 50,000, and 50,000 images as the validation sets for CIFAR10, ImageNet32, and ImageNet64, respectively. We have observed that employing distinct Φ_s and Θ in the same plane can enhance performance, although it tends to be prone to overfitting. Consequently, for the smaller CIFAR10 dataset, the parameters of Φ_s and Θ in the same plane are shared to mitigate this risk. However, for the larger datasets, these parameters are not shared. In alignment with prior studies, we employ dropout in CIFAR10 to prevent overfitting. Furthermore, we note a similar tendency for overfitting in our method when applied to ImageNet32, and as such, we also employ dropout in this context. The models selected for use are those that demonstrated the best performance on their respective validation sets.

A.2. Experiments on Generalizability

Following [13, 28, 41, 42], we use the model trained on ImageNet32 to evaluate the dataset compression performance on CIFAR10 and ImageNet64 as an ad hoc test of generalizability. As illustrated in Table 5, the performance on all datasets behaves similarly to the performance on ImageNet32, indicating that our method has no issues with generalizability.

A.3. Performance Comparison to SHVC-ArIB

We note that precise values for the single image compression performance of SHVC, as well as the compression performance of SHVC-ArIB, are not available. Qualitatively, they are inferior to the dataset performance of SHVC. Moreover, the authors present a figure (Figure 4 in [28]), which depicts the additional overheads of SHVC and SHVC-ArIB for single image compression, using the dataset compression performance of SHVC as a reference. From the figure, it's evident that the overhead of SHVC surpasses 0.8 BPD on three low-resolution datasets. The authors further point out that SHVC's overhead is approximately 20 times that of SHVC-ArIB. Consequently, when it comes to single-image compression, our performance surpasses SHVC-ArIB by an average of more than 0.10 BPD on low-resolution datasets. Furthermore, the authors indicate that the additional overhead of SHVC-ArIB comes from performance cost, implying a similar performance between dataset compression and single-image compression for SHVC-ArIB. Hence, our approach also surpasses SHVC-ArIB in dataset performance, exhibiting a margin that exceeds an average of 0.10 BPD on low-resolution datasets.

B. Details In Method

B.1. Discretized Sampling

In prevalent lossless compression methodologies based on VAEs, the latent variables manifest as continuous variables during the training phase, necessitating discretization for the subsequent compression process, wherein the probability density function $p(z)$ is employed. Consider $\hat{z}$ as the discretized value situated within the interval $[a, a+\delta a]$. The associated probability mass can be determined using the integral $P(\hat{z}) = \int_a^{a+\delta a} p(z)dz$, which, given a sufficiently small δa, can be approximated as $P(\hat{z}) \approx \delta a \cdot p(\hat{z})$. Analogously, $Q(\hat{z})$ is computed in a similar fashion.

Under these circumstances, the bit rate for $\hat{z}$ can be expressed as follows:

$$\begin{aligned} R_{\hat{z}} &= -\log\left(\frac{P(\hat{z})}{Q(\hat{z})}\right) \\ &\approx -\log\left(\frac{p(\hat{z})\cdot\delta a}{q(\hat{z})\cdot\delta a}\right) \\ &= -\log(p(\hat{z})) + \log(q(\hat{z})). \end{aligned} \tag{13}$$

However, discretization precision k is relatively small for a small subset of images, leading to a larger δa. This discrepancy results in a significant bias between the estimated and actual bit rates, adversely affecting the compression performance of single images. To mitigate this issue, we employ a sampling strategy to approximate the discretization process. Due to the gradient bias introduced by the rounding operation, we restrict the application of this method exclusively to fine-tuning the model, rather than incorporating it throughout the entire training process.

In our method, a latent variable z is sampled from a logistic distribution $\mathrm{Logistic}(\mu_q, 1)$, and discretized into in-

tervals such that each interval has an equal probability mass under $\text{Logistic}(\mu_p, 1)$. The continuous z is sampled with $z = \epsilon + \mu_q$, where $\epsilon \sim \text{Logistic}(0, 1)$. Given a discretization precision k, the number of intervals is 2^k. The cumulative distribution function of z under $\text{Logistic}(\mu_p, 1)$ is $\text{sigmoid}(z - \mu_p)$, and the lower, middle, and upper cumulative distribution values for the target interval are $\frac{\lceil 2^k \cdot \text{sigmoid}(z-\mu_p) \rceil - x}{2^k}$, where $x = 1, 0.5, 0$, respectively. Thus, we could obtain the lower, middle, and upper points of the interval. We use the middle point of the interval as the discretized latent variable $\hat{z}$. We then calculate the posterior and prior probabilities of $\hat{z}$ using the endpoints under $\text{Logistic}(\mu_q, 1)$ and $\text{Logistic}(\mu_p, 1)$. This approach leads to the derivation of Equation 9.

B.2. Entropy Coder

The entropy coding could be accelerated when intermediate results are precalculated, especially when the number of distributions is small. In PILC [16], the authors employ a single logistic distribution for probability modeling and introduce ANS-AI, which proves to be faster than the range ANS (rANS) [8] used in other methods. ANS-AI is a semi-dynamic entropy coder that precalculates intermediate results stored in a table, whereas rANS is a dynamic coder requiring time-consuming multiplication and division.

In our method, there are three types of symbols following distinct distributions that need to be entropy coded, and we employ different entropy coding methods for each type.

The first type consists of latent variables following a uniform distribution, where different symbols have equal probability mass. Since the number of values is a power of 2, we encode them by simply appending their binary representation to the bitstream, which is fast.

The second type comprises latent variables following a logistic distribution. Although the distribution has only one parameter, we observe that discretizing it adversely affects performance. Moreover, the number of such symbols occupies a small portion (approximately 3.8%), making it non-critical for accelerating entropy coding. Therefore, we use rANS for encoding.

The third type includes bits following a Bernoulli distribution. The number of such symbols occupies a primary portion (approximately 92.4%), and thus is crucial for accelerating entropy coding. The Bernoulli distribution also requires one parameter. Moreover, we find that it does not affect the performance much if we discretize the parameter with relatively small precision. Therefore, we employ a semi-dynamic entropy coder. Though ANS-AI demonstrates very fast coding speed, it incurs an unacceptable performance degradation for our method. Thus, we choose to employ FSE [1], which could achieve a bit rate close to the theoretical bit rate.

FSE is a static entropy coder, compatible with symbols with a fixed distribution. It precalculates an encoding table for encoding and a decoding table for decoding. Each symbol follows a Bernoulli distribution using one of the predefined parameters. Thus, we modify FSE into semi-dynamic FSE for entropy coding bits. Considering there are m probable parameter values, we precalculate m encoding tables and m decoding tables, each with different indexes. To encode or decode a bit, we input the distribution index and use it to obtain the corresponding table, then use the table for FSE encoding and decoding. It could achieve approximately twice the acceleration for entropy encoding with comparable entropy decoding speed compared to a binary rANS. It is noteworthy that in our method, entropy coding constitutes a minor portion of the coding time, whereas inference time emerges as the predominant factor. For example, when compressing a 512x512 image with the ImageNet64-trained model, entropy coding accounts for 5.2% and 4.5% of the encoding and decoding times, respectively. However, various methods, such as pruning, quantization, and the deployment of more powerful accelerators, can be employed to expedite the inference stage. In scenarios where these techniques are implemented, our acceleration in entropy coding stands poised to provide additional acceleration to the overall coding process.

[1] https://github.com/Cyan4973/FiniteStateEntropy