Appendix - Compression with Bayesian Implicit Neural Representations

Anonymous Author(s)
Affiliation
Address
`email`

In addition to the four appendix sections mentioned in our main paper, we would like to draw attention to two additional experiments: one evaluating the practical training and coding time, and the other investigating the impact of the number of training samples. These two experiments, especially the later one, offer **crucial** insights and are detailed in Appendix E1 and Appendix E2, respectively.

A Relative Entropy Coding with A* Coding

Algorithm 1 A* encoding

Require: Proposal distribution $p_{\mathbf{w}}$ and target distribution $q_{\mathbf{w}}$.

Initialize : $N, G_0, \boldsymbol{w}^*, N^*, L \leftarrow 2^{|C|}, \infty, \perp, \perp, -\infty$
for $i = 1, \ldots, N$ **do** ▷ N samples from proposal distribution
  $\boldsymbol{w}_i \sim p_{\mathbf{w}}$
  $G_i \sim \mathrm{TruncGumbel}(G_{i-1})$
  $L_i \leftarrow G_i + \log\left(q_{\mathbf{w}}(\boldsymbol{w}_i)/p_{\mathbf{w}}(\boldsymbol{w}_i)\right)$ ▷ Perturbed importance weight
  if $L_i \leqslant L$ **then**
    $L \leftarrow L_i$
    $\boldsymbol{w}^*, N^* \leftarrow \boldsymbol{w}_i, i$
  end if
end for
return $\boldsymbol{w}^*, N^*$ ▷ Transmit the index N^*

Recall that we would like to communicate a sample from the variational posterior distribution $q_{\mathbf{w}}$ using the proposal distribution $p_{\mathbf{w}}$. In our experiments, we used *global-bound depth-limited A* coding* to achieve this [1]. We describe the encoding procedure in Algorithm 1 and the decoding procedure in Algorithm 2. For brevity, we refer to this particular variant of the algorithm as *A* coding* for the rest of the appendix.

A* coding is an importance sampler that draws N samples $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_N \sim p_{\mathbf{w}}$ from the proposal distribution $p_{\mathbf{w}}$, where N is a parameter we pick. Then, it computes the importance weights $r(\boldsymbol{w}_n) = q_{\mathbf{w}}(\boldsymbol{w}_n)/p_{\mathbf{w}}(\boldsymbol{w}_n)$, and sequentially perturbs them with truncated Gumbel[1] noise:

$$\tilde{r}_n = r(\boldsymbol{w}_n) + G_n, \quad G_n \sim \mathrm{TruncGumbel}(G_{n-1}),\, G_0 = \infty \tag{1}$$

Then, it can be shown that by setting

$$N^* = \operatorname*{arg\,max}_{n \in [1:N]} \tilde{r}_n, \tag{2}$$

[1]The PDF of a standard Gumbel random variable truncated to $(-\infty, b)$ is given by $\mathrm{TruncGumbel}(x \mid b) = \mathbf{1}[x \leqslant b] \cdot \exp(-x - \exp(-x) + \exp(-b))$.

Algorithm 2 A* decoding

Simulate $\{\boldsymbol{w}_i\} = \{\boldsymbol{w}_1, \cdots, \boldsymbol{w}_N\}$ ▷ Simulate N samples from $p_{\mathbf{w}}$ with the shared seed
Receive N^*

return $\boldsymbol{w}^* \leftarrow \boldsymbol{w}_{N^*}$ ▷ Receive the approximate posterior sample

we have that $\boldsymbol{w}_{N^*} \sim \tilde{q}_{\mathbf{w}}$ is approximately distributed according to the target, i.e. $\tilde{q}_{\mathbf{w}} \approx q_{\mathbf{w}}$. More preciesly, we have the following result:

Lemma A.1 (Bound on the total variation between $\tilde{q}_{\mathbf{w}}$ and $q_{\mathbf{w}}$ (Lemma D.1 in [2])). *Let us set the number of proposal samples simulated by Algorithm 1 to $N = 2^{D_{\mathrm{KL}}[q_{\mathbf{w}} \| p_{\mathbf{w}}]+t}$ for some parameter $t \geqslant 0$. Let $\tilde{q}_{\mathbf{w}}$ denote the approximate distribution of the algorithm's output for this choice of N. Then,*

$$D_{TV}(q_{\mathbf{w}}, \tilde{q}_{\mathbf{w}}) \leqslant 4\epsilon, \tag{3}$$

where

$$\epsilon = \left(2^{-t/4} + 2\sqrt{\mathbb{P}_{Z\sim q_{\mathbf{w}}}[\log_2 r(Z) \geqslant D_{\mathrm{KL}}[Q\|P] + t]}\right)^{1/2}. \tag{4}$$

This result essentially tells us that we should draw at least around $2^{D_{\mathrm{KL}}[q_{\mathbf{w}}\|p_{\mathbf{w}}]}$ samples to ensure low sample bias, and beyond this, the bias decreases exponentially quickly as $t \to \infty$. However, note that the number of samples we need also increases exponentially quickly with t. In practice, we observed that when $D_{\mathrm{KL}}[q_{\mathbf{w}}\|p_{\mathbf{w}}]$ is sufficiently large (around 16-20 bits), setting $t = 0$ already gave good results. To encode N^*, we built an empirical distribution over indices using our training datasets and used it for entropy coding to find the optimal variable-length code for the index.

In short, on the encoder side, N random samples are obtained from the proposal distribution $p_{\mathbf{w}}$, and we select the sample $\boldsymbol{w}_i$ and transmit its index N^* that has the greatest perturbed importance weight. On the decoder side, those N random samples can be simulated with the same seed held by the encoder. The decoder only needs to find the sample with the index N^*. Therefore, the decoding process of our method is very fast. We also provide the specific coding time in Appendix E1.

B Closed-Form Solution for Updating Model Prior

In this section, we derive the analytic expressions for the prior parameter updates in our iterative prior learning procedure when both the prior and the posterior are Gaussian distributions. Given a set of training data $\{\mathcal{D}_i\} = \{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M\}$, we fit a variational distribution $q_{\mathbf{w}}^{(i)}$ to represent each of the $\mathcal{D}_i$s. To do this, we minimize the loss (abbreviated as $\mathcal{L}$ later)

$$\begin{aligned}
\bar{\mathcal{L}}_\beta(\boldsymbol{\theta}_p, \{q_{\mathbf{w}}^{(i)}\}) &= \frac{1}{M}\sum_{i=1}^{M} \mathcal{L}_\beta(\mathcal{D}_i, q_{\mathbf{w}}^{(i)}, p_{\mathbf{w};\boldsymbol{\theta}_p}) && (5)\\
&= \frac{1}{M}\sum_{i=1}^{M}\{ \sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \mathbb{E}_{\boldsymbol{w}\sim q_{\mathbf{w}}}[\Delta(\boldsymbol{y}, f(\boldsymbol{x} \mid \boldsymbol{w})] + \beta \cdot D_{\mathrm{KL}}[q_{\mathbf{w}}\|p_{\mathbf{w};\boldsymbol{\theta}_p}] \ \}. && (6)
\end{aligned}$$

Now calculate the derivative w.r.t. the prior distribution parameter $p_{\mathbf{w};\boldsymbol{\theta}_p}$,

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_p} = \frac{1}{M}\sum_{i=1}^{M} \frac{\partial D_{\mathrm{KL}}[q_{\mathbf{w}}\|p_{\mathbf{w},\boldsymbol{\theta}_p}]}{\partial \boldsymbol{\theta}_p} \tag{7}$$

Considering we choose factorized Gaussian as variational distributions, the KL divergence is

$$\begin{aligned}
D_{\mathrm{KL}}[q_{\mathbf{w}}^{(i)}\|p_{\mathbf{w},\boldsymbol{\theta}_p}] &= D_{\mathrm{KL}}[\mathcal{N}(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i))\|\mathcal{N}(\boldsymbol{\mu}_i, \mathrm{diag}(\boldsymbol{\sigma}_i))] && (8)\\
&= \frac{1}{2}\log\frac{\boldsymbol{\sigma}_p}{\boldsymbol{\sigma}_q^{(i)}} + \frac{\boldsymbol{\sigma}_q^{(i)} + (\boldsymbol{\mu}_q^{(i)} - \boldsymbol{\mu}_p)^2}{\boldsymbol{\sigma}_p} - \frac{1}{2} && (9)
\end{aligned}$$

To compute the analytical solution, let

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_p} = \frac{1}{M}\sum_{i=1}^{M} \frac{\partial D_{\mathrm{KL}}[q_{\mathbf{w}} \| p_{\mathbf{w},\boldsymbol{\theta}_p}]}{\partial \boldsymbol{\theta}_p} = 0. \tag{10}$$

Note here $\boldsymbol{\sigma}$ refers to variance rather than standard deviation. The above equation is equivalent to

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_p} &= \sum_{i=1}^{M} \frac{\boldsymbol{\mu}_p - \boldsymbol{\mu}_q^{(i)}}{\boldsymbol{\sigma}_p} = 0, \\ \frac{\partial \mathcal{L}}{\partial \boldsymbol{\sigma}_p} &= \sum_{i=1}^{M} [\frac{1}{\boldsymbol{\sigma}_p} - \frac{\boldsymbol{\sigma}_q^{(i)} + (\boldsymbol{\mu}_q^{(i)} - \boldsymbol{\mu}_p)^2}{\boldsymbol{\sigma}_p^2}] = 0. \end{aligned} \tag{11}$$

We finally can solve these equations and get

$$\boldsymbol{\mu}_p = \frac{1}{M}\sum_{i=1}^{M} \boldsymbol{\mu}_q^{(i)}, \quad \boldsymbol{\sigma}_p = \frac{1}{M}\sum_{i=1}^{M} [\boldsymbol{\sigma}_q^{(i)} + (\boldsymbol{\mu}_q^{(i)} - \boldsymbol{\mu}_p)^2] \tag{12}$$

as the result of Equation (5) in our main text. In short, this closed-form solution provides an efficient way to update the model prior from a bunch of variational posteriors. It makes our method simple in practice, unlike some previous methods [3, 4] that require expensive meta-learning loops.

C Dynamic Adjustment of β

When learning the model prior, the value of β that controlling the rate-distortion trade-off is defined in advance to train the model prior at a specific bitrate point. After obtaining the model prior, we will first partition the network parameters into K groups $\mathbf{w}_{1:K} = \{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$ according to the average approximate coding cost of training data, as described in Section 3.3 of the main text. Now for training the variational posterior for a given test datum, to ensure the coding cost of each group is close to $\kappa = 16$ bits, we adjust the value of β dynamically when optimizing the posteriors. The detailed algorithm is illustrated here in Algorithm 3.

Algorithm 3 Dynamic β adjustment for optimizing the posteriors

Require: $\beta, \mathbf{w}_{1:K} = \{\mathbf{w}_1, \ldots, \mathbf{w}_K\}$

- **Initialize:** $\lambda_k = \beta, k = 1, \cdots, K$
- **Initialize:** variational posterior $q_{\mathbf{w}_k}, k = 1, \cdots, K$

for $i \leftarrow$ NumberIter **do**
- $\delta_k = D_{\mathrm{KL}}[q_{\mathbf{w}_k} \| p_{\mathbf{w}_k}], k = 1, \cdots, K$
- $q_{\mathbf{w}_{1:K}} \leftarrow$ `VariationalUpdate(`$\mathcal{L}_{\lambda_{1:K}}$`)` $\quad \triangleright$ $\mathcal{L}_{\lambda_{1:K}}$ is defined in Equation 8 in the main text
- **if** $(i \mod 15) = 0$ **then**
  - **if** $\delta_k > \kappa$ **then** $\lambda_k = \lambda_k \cdot 1.05$
  - **end if**
  - **if** $\delta_k < \kappa - 0.4$ **then** $\lambda_k = \lambda_k$ / 1.05
  - **end if**
- **end if**

end for

return $q_{\mathbf{w}_k}, \lambda_k, k = 1, \cdots, K$

The algorithm is improved from Havasi et al. [5] to stabilize training, in the way that we set an interval $[\kappa - 0.4, \kappa]$ as buffer area where we do not change the value of λ_k. Here we only adjust λ_k every 15 iterations to avoid frequent changes at the initial training stage.

D Experiment Details

We introduce the experimental settings here and summarize the settings in Table 1.

	CIFAR-10	Kodak		LibriSpeech
		Smaller Model	Larger Model	
Network Structure				
number of MLP layer	4	6	7	6
hidden unit	16	48	56	48
Fourier embedding	32	64	96	64
number of parameters	1123	12675	21563	12675
Learning Model Prior from Training Data				
number of training data	2048	512	512	1024
epoch number	128	96	96	128
learning rate	0.0002	0.0001	0.0001	0.0002
iteration / epoch (except the first epoch)	100	200	200	100
iteration number in the first epoch	250	500	500	250
initialization of posterior variance	9×10^{-6}	4×10^{-6}	$4 \times 10^{-6}, 4 \times 10^{-10}$	4×10^{-9}
β	$2 \times 10^{-5}, 5 \times 10^{-6}, 2 \times 10^{-6}$ $1 \times 10^{-6}, 5 \times 10^{-7}$	$10^{-7}, 10^{-8}, 4 \times 10^{-8}$	4×10^{-6}	$10^{-7}, 3 \times 10^{-8}$ $10^{-8}, 10^{-9}$
Optimize the Posterior of a Test Datum				
iteration number	25000	25000	25000	25000
learning rate	0.0002	0.0001	0.0001	0.0002
training with 1/4 the points (pixels)	✗	✓	✓	✗
number of group (KL budget = 16 bits / group)	(58, 89, 146, 224, 285)	(1729, 2962, 3264)	(5503, 7176)	(1005, 2924, 4575, 6289)
bitrate, (bpp for images, Kbps for audios)	(0.91, 1.39, 2.28, 3.50, 4.45)	(0.070, 0.110, 0.132)	(0.224, 0.293)	(5.36, 15.59, 24.40, 33.54)
PSNR, dB	(0.91, 1.39, 2.28, 3.50, 4.45)	(0.070, 0.110, 0.132)	(0.224, 0.293)	(5.36, 15.59, 24.40, 33.54)

Table 1: Hyper parameters in our experiments.

D.1 CIFAR-10

We use a 4-layer MLP with 16 hidden units and 32 Fourier embeddings for the CIFAR-10 dataset. The model prior is trained with 128 epochs to ensure convergence. Here, the term “epoch” is used to refer to optimizing the posteriors and updating the prior in the Algorithm 1 in the main text. For each epoch, the posteriors of all 2048 training data are optimized for 100 iterations using the local reparameterization trick [6], except the first epoch that contains 250 iterations. We use the Adam optimizer with learning rate 0.0002. The posterior variances are initialized as 9×10^{-6}.

After obtaining the model prior, given a specific test CIFAR-10 image to be compressed, the posterior of this image is optimized for 25000 iterations, with the same optimizer. When we finally progressively compress and finetune the posterior, the posteriors of the uncompressed parameter groups are finetuned for 15 iterations with the same optimizer once a previous group is compressed.

D.2 Kodak

For Kodak dataset, since training on high-resolution image takes much longer time, the model prior is learned using fewer training data, i.e., only 512 cropped CLIC images [7]. We also reduce the learning rate of the Adam optimizer to 0.0001 to stabilize training. In each epoch, the posterior of each image is trained for 200 iterations, except the first epoch that contains 500 iterations. We also reduce the total epoch number to 96 which is empirically enough to learn the model prior.

We use two models with different capacity for compressing high-resolution Kodak images. The smaller model is a 6-layer SIREN with 48 hidden units and 64 Fourier embeddings. This model is used to get the three low-bitrate points in Figure 2b in our main text, where the corresponding beta is set as $\{10^{-7}, 10^{-8}, 4\times10^{-8}\}$. Another larger model comprises a 7-layer MLP with 56 hidden units and 96 Fourier embeddings, which is used for evaluation at the two relatively higher bitrate points in Figure 2b in our main text. The betas of these two models have the same value 2×10^{-9}. We empirically adjust the variance initialization from the set $\{4\times10^{-6}, 4\times10^{-10}\}$ and find they can affect the converged bitrate and achieve good performance. In particular, the posterior variance is initialized as 4×10^{-10} to reach the highest bitrate point in the rate-distortion curve. The posterior variance of other bitrate-points on Kodak dataset are all initialized as 4×10^{-6}.

Important note: It required significant empirical effort to find the optimal parameter settings we described above, hence our note in the Conclusion and Limitations section that Bayesian neural networks are inherently sensitive to initialization [8].

D.3 LibriSpeech

We randomly crop 1024 audio samples from LibriSpeech "train-clean-100" set [9] for learning the model prior and randomly crop 24 test samples from "test-clean" set for evaluation. The model structure is the same as the small model used for compressing Kodak images. We evaluate on four bitrate points by setting $\beta = \{10^{-7}, 3\times10^{-8}, 10^{-8}, 10^{-9}\}$. There are 128 epochs, and each epoch has 100 iterations with learning rate as 0.0002. The first epoch has 250 iterations. In addition, the posterior variance is initialized as 4×10^{-9}. The settings for optimizing and finetuning posterior of a test datum are the same as the experiments on Kodak dataset.

E Supplementary Experimental Results

E.1 Evaluation of Training and Practical Coding Time

Training Time. The most time-consuming part of setting up COMBINER for a data modality is running the iterative prior learning algorithm we propose in the main text. Furthermore, optimizing the variational posteriors takes up the bulk of the learning process since updating the prior parameters given the variational posteriors can be done efficiently using the formulae we derive in Appendix B. However, such posterior optimization can be done in parallel, especially given that our INRs have very few parameters. To train the model prior on the CIFAR-10 dataset, we can train the posteriors of 2048 images together in a single V100 GPU. It only takes around **20 minutes** to train for 128 epochs with almost 100 iterations per epoch. For training the model prior with 512 cropped CLIC images, due to the limit of GPU memory, we run multiple processes simultaneously on 4 GTX 1080 GPUs, where each process runs for a single image. The entire training time on CLIC dataset consumes around 30 hours. We note that the training time on CLIC dataset could be significantly reduced with additional engineering effort, but we have not had the opportunity to do so due to time constraints.

Coding Time. To compress a test datum, we first optimize its INR's variational posterior for 25,000 iterations. Such optimization process should also be included as a part of encoding time, similar to COIN [10]. In addition, the progressive posterior refinement process also takes a long time. Therefore, the encoding time of our method is very long. Note that the encoding time of relative entropy coding is negligible compared with the optimization process because our model is very small, and there are not so many parameter groups. As a result, we are able to evaluate all the 10,000 images from the CIFAR-10 test set in parallel using a CPU cluster. To decode the network parameters, the decoder only needs to search for the sample according to the received index, which is very fast. The practical encoding and decoding time is shown in Table 2.

	CIFAR, bpp = 0.91	CIFAR, bpp = 4.45	Kodak, bpp = 0.070	Kodak, bpp = 0.293
encoding time	$\sim$10 minutes	$\sim$20 minutes	$\sim$2 hours	$\sim$4 hours
decoding time	0.051 second	0.075 second	0.410 second	0.542 second

Table 2: Practical encoding and decoding time of a specific image on 1080Ti GPU.

We show both the encoding and decoding time of our method on different datasets at different bitrates. In fact, the decoding time is mainly consumed for relative entropy decoding and inference

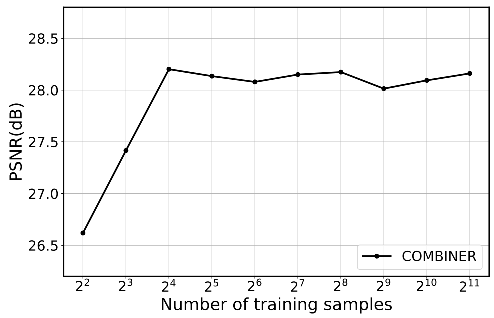

Figure 1: Impact of the number of training data.

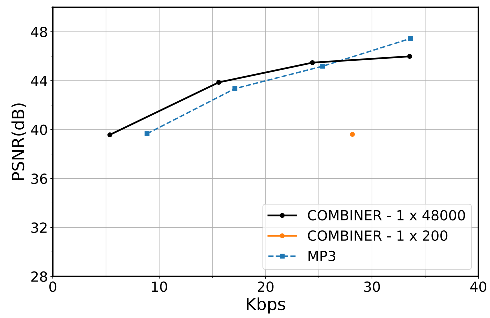

Figure 2: Compressing audios.

of the received MLP network. Usually, if there are more parameter groups, the coding time will be longer. Therefore, decoding a Kodak image at our highest bitrate (0.293 bpp) consumes the most decoding time (0.542 second), but is still very fast.

E.2 Number of Training Samples

Since the model prior is learned from a few training data, the number of training data may influence the quality of the learned model prior. We train the model prior with a different number of training images from the CIFAR-10 training set and evaluate the performance on 100 randomly selected test images from the CIFAR-10 test set. Surprisingly, as shown in Figure 1, we found that even merely 16 training images can help to learn a good model prior. Considering the randomness of training and testing, the performance on this test subset is almost the same when the number of training data exceeds 16. This demonstrates that the model prior is quite robust and generalizes well to test data. In our final experiments, the number of training samples is set to 2048 (on CIFAR-10 dataset) to ensure the prior converges to a good optimum.

E.3 Compressing Audios with Small Chunks

The proposed approach does not need to compute the second-order gradient during training, which helps to learn the model prior of the entire datum. Hence, compression with a single Bayesian INR network helps to fully capture the global dependencies of a datum. That is the reason for our strong performance on Kodak and LibriSpeech datasets. Here, we also conduct a group of experiment to compare the influence of cropping audios into chunks. Unlike the experimental setting in our main text that compresses every 3-second audio (1×48000) with a single MLP network, here we try to crop all the 24 audios into small chunks, each of the chunk has the shape of 1×200. We use the same network used for compressing CIFAR-10 images for our experiments here. As shown in Figure 2, if we do not compress the audio as an entire entity, the performance will drops for around 5 dB. It demonstrates the importance of compressing with a single MLP network to capture the inherent redundancies within the entire data.

E.4 Additional Figures

We provide some examples of the decoded Kodak images in Figure 3.

Figure 3: Decoded Kodak images.

References

[1] Gergely Flamich, Stratis Markou, and José Miguel Hernández-Lobato. Fast relative entropy coding with a* coding. In *International Conference on Machine Learning*, pages 6548–6577. PMLR, 2022. 1

[2] L. Theis and E. Agustsson. On the advantages of stochastic encoders. In *Neural Compression Workshop at ICLR*, 2021. URL `https://arxiv.org/abs/2102.09270`. 2

[3] Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Golinski, Y Whye Teh, and Arnaud Doucet. Coin++: Neural compression across modalities. *Transactions on Machine Learning Research*, 2022(11), 2022. 3

[4] Jonathan Richard Schwarz and Yee Whye Teh. Meta-learning sparse compression networks. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. 3

[5] Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *7th International Conference on Learning Representations, ICLR 2019*, 2019. 3

[6] Diederik P. Kingma, Tim Salimans, and Max Welling. In *Variational Dropout and the Local Reparameterization Trick*, volume 28, 2015. 4

[7] 5th Workshop and Challenge on Learned Image Compression. `http://compression.cc`, 2022. 4

[8] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015. 5

[9] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An ASR corpus based on public domain audio books. In *ICASSP*, pages 5206–5210. IEEE, 2015. 5

[10] Emilien Dupont, Adam Golinski, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. COIN: compression with implicit neural representations. *arXiv preprint arXiv:2103.03123*, 2021. 5