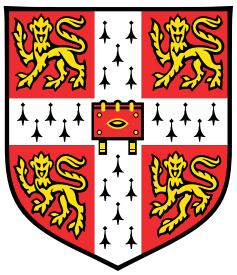
Data Compression with Variational Implicit Neural Representations



Jiajun He

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

Fitzwilliam College August 2023

To wisdom.

Declaration

I, Jiajun He of Fitzwilliam College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Software Declaration: All of this work is implemented using standard Python packages.

Word count: 14996

Jiajun He
August 2023

Acknowledgements

I would like to acknowledge my supervisors, Gergely Flamich and Prof. José Miguel Hernández-Lobato, first and foremost. Their passion, enthusiasm, and patience are extremely helpful throughout this project. I feel very fortunate to have had the opportunity to be their supervisee.

I would also like to extend my thanks to Zongyu Guo, whose expertize in this field has been a constant source of inspiration, and his initial works greatly facilitated my experiments in this project. It is amazing to work with him.

I would like to thank my loving parents, grandparents, my girlfriend Yang, and all relatives for always being there, consistently supporting and caring for me. I am such a lucky boy from a lucky family.

Last but certainly not least, a huge thank you to Qiaosong, Murray, and many others, to whom I am so honored to refer as my friends. They have greatly colored my life.

Abstract

Title: Data Compression with Variational Implicit Neural Representations
Name: Jiajun He

The growing need for data transmission and storage highlights the requirement for efficient compression techniques. The recent surge in deep learning has propelled remarkable progress in the field of neural compression. While the mainstream of studies is centered around variational autoencoder architectures, which enable end-to-end rate-distortion optimization with entropy models, there is a recent emerging trend in research involving treating individual data points as continuous functions. Known as Implicit Neural Representations (INRs), this framework has demonstrated the potential to develop codecs with transferability across various data modalities.

Our work in this thesis is rooted in this methodology. We present a general framework, dubbed as COMBINER (**Com**pression with **B**ayesian **I**mplicit **Ne**ural **R**epresentations), addressing a significant challenge in existing INR-based codecs: their lack of support for joint rate-distortion optimization, which potentially degrades the compression performance. On top of COMBINER, we further propose COMBINER+, employing simple yet efficient approaches enhancing performance. Fundamentally different from most INR compression researches, which adopt increasingly complicated parameterizations to enhance compression, our approaches take a novel direction that we believe holds significant potential. Experimental results demonstrate that our methods achieve strong performance on image compression while retaining simplicity.

Table of contents

Nomenclature

Codes(■) (the color scheme is inspried by Havasi (2021))

$\mathcal{M}(X)$ Binary message encoding X

Matrices

$\text{diag}(\boldsymbol{x})$ A diagonal matrix with $\boldsymbol{x}$ as the diagonal elements

$\text{Vec}(\boldsymbol{W})$ A vector containing elements of matrix $\boldsymbol{W}$

Distributions and Variables(■)

$\mathbf{x}, \mathrm{X}$ Symbolic representation of one continuous / discrete variable

$\boldsymbol{x}, X$ Realization of one continuous / discrete variable

$p(\boldsymbol{x})$ The density of a continuous random variable $\mathbf{x}$ at point $\boldsymbol{x}$, i.e., $p(\mathbf{x} = \boldsymbol{x})$

$P(X)$ The probability of a discrete random variable X being X, i.e., $P(\mathrm{X} = X)$

$\boldsymbol{x} \sim p$ A random continuous sample from p

$X \sim P$ A random discrete sample from P

Sets(■)

$\emptyset$ Empty set

$\{a, b, c\}$ A set contains elements a, b, c

$\{x_i\}_{i=1}^{N}$ A set contains elements $x_1, x_2, \ldots, x_N$

$\mathcal{A}$ Set $\mathcal{A}$

$x \in \mathcal{A}$ The element x in set $\mathcal{A}$

$\{\mathcal{A}, \mathcal{B}\}$ A set contains set $\mathcal{A}$ and set $\mathcal{B}$ as elements

Chapter 1

Introduction

Compression plays a pivotal role in modern digital era by reducing the data storage space and enabling swifter data transmission. While the history of apply machine learning to data compression can be traced back to earlier ages [Jiang (1999)], the recent surge in deep learning, especially the development of variational autoencoders (VAEs) [Kingma and Welling (2014)], has propelled significant advancements in neural compression, particularly within the realm of lossy image compression [Ballé et al. (2017, 2018); Cheng et al. (2020); Guo et al. (2022); Theis et al. (2017)]. However, the success of these methods largely thanks to their elaborately designed modality-specific architectures, highly limiting their transferability across data modalities.

A recent line of studies has shown the potential of designing modality-agnostic codecs by treating individual data points as continuous functions mapping coordinates to signal values. Such functions, parameterized by compact neural networks, are known as Implicit Neural Representations (INRs). Moreover, thanks to the small size of INRs, these codecs typically feature a remarkably shorter decoding times in comparison to VAE-based codecs.

However, existing INR-based codecs experience limitations in performance. A major reason for this disparity is that these methods are unable to directly optimize compression costs, and merely quantize parameters to a fixed precision. In contrast, VAE-based methods support end-to-end rate-distortion optimization with powerful entropy models.

To this end, in this thesis, we propose a general framework, dubbed as COMBINER (**Com**pression with **B**ayesian **I**mplicit **Ne**ural **R**epresentations), which naturally supports joint rate-distortion optimization, and adaptively controls its bit-rates. On top of this framework, we further propose COMBINER+, enhancing its performance by linear transforming INR parameters and introducing learnable positional encodings. Experiments show that our methods achieve strong performance among INR-based codecs.

1.1 Contributions and Publication

Contributions

- **a comparative review** of recent INR-based codecs;
- **COMBINER**, a novel INR-based codec naturally supporting joint rate-distortion optimization;
- **COMBINER+**, a codec developed on top of COMBINER, demonstrating significantly enhanced performance.

Publication

The following paper presents COMBINER. It has been submitted as a conference paper to NeurIPS. It receives a high average rating[1], and thus is likely to be accepted. A pre-print version is available at https://arxiv.org/abs/2305.19185.

Title: Compression with Bayesian Implicit Neural Representations

Authors: Zongyu Guo*, Gergely Flamich*, Jiajun He, Zhibo Chen and José Miguel Hernández-Lobato

My contribution: I implemented the practical coding algorithms, performed the image compression experiments with Zongyu, and conducted ablation studies and supplementary experiments. I also wrote the paper together with Zongyu Guo and Gergely Flamich.

Other authors' contribution: Zongyu Guo handled preliminary experiments, including all hyperparameter tuning, collaborated with me on image compression, drafted the paper, and was fully responsible for experiments on audio compression, which falls beyond the scope of the COMBINER part in this thesis. Gergely Flamich provided the idea of this work.

Besides, we are planning another paper on COMBINER+ for ICLR.

1.2 Outline

The thesis is organized as follows:

- in Chapter 2, we offer the related background in compression and deep learning, followed by a review of related works;
- in Chapter 3, we introduce COMBINER;
- in Chapter 4, we introduce COMBINER+;
- in Chapter 5, we conclude the thesis with a discussion on limitations and future works.

[1] We receive 8, 7, 6, 6, 3. The last comes with a low confidence, and is likely to be raised after rebuttal.

Chapter 2

Background

The chapter begins by outlining the core principles of lossless and lossy compression, followed by a concise introduction to deep learning. While both areas are extensive, the chapter primarily focuses on topics most relevant to this thesis. The chapter ends with a review of related works.

2.1 Compression

In this section, we look at compression. There are two scenarios in compression, known as lossless and lossy compression. Lossless compression aims to encode data while guaranteeing a perfect reconstruction. On the other hand, lossy compression encodes data, permitting the loss of information. In the following, we introduce these two scenarios respectively.

2.1.1 Lossless Compression

Lossless compression, also known as source coding, aims to encode all information in data with as few bits as possible. It involves first estimating the underlying distribution of the data, and performing entropy coding according to this distribution.

Entropy Coding

Assuming a discrete random variable $\mathrm{X} \in \mathbb{X}$ whose probability mass function (PMF) is denoted by $P(\mathrm{X})$, the goal of Lossless compression is to find an invertible mapping $\mathcal{M}$ from $\mathbb{X}$ to binary sequences, so that the codelength $|\mathcal{M}(X)|$ is minimized.

The core concept in lossless compression is the entropy of a distribution, introduced by Shannon (1948) to measure the information content of a distribution. Specifically, the

entropy of distribution $P(\mathrm{X})$ is defined as

$$H[\mathrm{X}] = \mathbb{E}_{X\sim P(\mathrm{X})}[-\log_2 P(X)]. \tag{2.1}$$

Shannon (1948) showed that the minimal average codelength of a random variable X is lower-bounded by the entropy of its distribution in lossless compression, which is known as Shannon's source coding theorem.

This bound on the codelength is tight. The basic strategy to approach this bound is to replace common symbols with shorter codewords, and rarer symbols with longer ones. In fact, the optimal codelength of X should be close to its negative log-probability, i.e.,

$$|\mathcal{M}(X)| \approx -\log_2 P(X). \tag{2.2}$$

Methods close to this limit are referred to as *entropy coding* methods. The most prominent entropy coding approaches include Huffman coding [Huffman (1952)] and arithmetic coding [Witten et al. (1987)].

Approximating P

However, in practice, the underlying distribution P is typically unknown but need to be approximated by Q. The average number of bits required to encode a sample $X \sim P(\mathrm{X})$ using a code optimized for Q is given by the *cross-entropy* between P and Q:

$$H[P,Q] = \mathbb{E}_{X\sim P(\mathrm{X})}[-\log_2 Q(X)]. \tag{2.3}$$

The overhead of this approximation is called *relative entropy* or Kullback-Leibler (KL) divergence, given by

$$D_{\mathrm{KL}}[P||Q] = \mathbb{E}_{X\sim P(\mathrm{X})}\left[\log \frac{P(X)}{Q(X)}\right]. \tag{2.4}$$

2.1.2 Lossy Compression

Lossy compression allows data to have minor distortions after compression. Lossy compression is necessary when working with continuous data, as storing any continuous value requires an infinite number of bits. Also, it is sometimes acceptable to discard non-essential information, such as imperceptible textures in images or audio frequencies beyond the range of human hearing, during data transmission. In general, allowing information loss brings huge flexility, in comparison with lossless compression.

Rate-Distortion Theory

The lossy compression is typically evaluated by two metrics, a distortion measure $\Delta(X,\hat{X})$, and the (bit-)rate $|\mathcal{M}(X)|$. Note that in lossy compression, we do not require the data to be discrete any more. The choice of distortion measure depends on the data modality. For instance, in image compression, the Mean Square Error (MSE) is commonly used. Additionally, perceptual losses are sometimes employed to assess visual similarity between images, including multi-scale SSIM (MSSSIM) [Wang et al. (2004)], Frechet Inception Distance (FID) [Heusel et al. (2018)], etc.

Given the above, lossy compression is generally formalized as a multi-objective optimization problem where we attempt to minimize the average distortion $\mathbb{E}_{X\sim P(\mathrm{X})}[\Delta(X,\hat{X})]$ and the bit-rate $\mathbb{E}_{X\sim P(\mathrm{X})}[|\mathcal{M}(X)|]$ at the same time.

Rate-Distortion (R-D) theory establishes limits on the performance of any lossy compression algorithm [Berger (2003); Yang et al. (2022)]. Lossy compression implements a noisy channel that receives X and outputs $\hat{X}$, described by a conditional distribution $Q(\hat{\mathrm{X}}|X)$. The lowest achievable bit-rate, limiting the average distortion to be smaller than a threshold D, is given by the information *rate-distortion function*, defined as:

$$\inf_{Q} \; I[\mathrm{X},\hat{\mathrm{X}}], \quad \text{s.t. } \mathbb{E}_{\hat{X},X\sim P\cdot Q(\hat{\mathrm{X}},\mathrm{X})}[\Delta(X,\hat{X})] \leq D, \tag{2.5}$$

where $P\cdot Q$ represents the joint distribution of $\hat{\mathrm{X}}$ and X. Denoting $\tilde{Q}$ as the marginal distribution of $\hat{\mathrm{X}}$, and $\otimes$ as the product measure, the mutual information I is given by

$$I[\mathrm{X},\hat{\mathrm{X}}] = D_{\mathrm{KL}}[P\cdot Q||P\otimes\tilde{Q}]. \tag{2.6}$$

The R-D function is always non-increasing and convex, but is otherwise unknown analytically beyond a few cases [Yang et al. (2022)].

In practice, instead of attempting to approach the aforementioned theoretically optimal rate, we usually consider the *operational* rate-distortion function to minimize the expected coding cost $|\mathcal{M}(X)|$ (instead of the mutual information) among all the acceptable codecs (instead of all possible codecs in theory), given the distortion threshold D. Note that this function can be stated conversely as well: to minimize the distortion so that the coding cost is governed by a threshold C, which we will revisit, from the perspective of Implicit Neural Representations (INRs), when presenting our approach.

Transform Coding

A common approach to perform lossy compression is to transform data into a transformed space, where ideally the representation of data is uncorrelated [Goyal (2001)], and perform compression in this space. As its core, the sender first applies an *analysis transform* f to data, obtaining an (ideally decorrelated) representation $\boldsymbol{z} = f(\boldsymbol{x})$, and then encode $\boldsymbol{z}$ into a bit-string $\mathcal{M}(\boldsymbol{z})$, e.g., by quantization and entropy coding. The receiver, after receiving $\mathcal{M}(\boldsymbol{z})$, computes the reconstruction $\hat{\boldsymbol{x}}$ using a *synthesis transform* g.

This is closely related to neural lossy compression. If we learn the analysis transform f and the synthesis transform g by two neural networks, the entire framework of transform coding can be viewed as a variational autoencoder (VAE) [Kingma and Welling (2014)].

Compression without Quantization

In the previous sections, when encoding $\boldsymbol{z} = f(\boldsymbol{x})$, we use quantizing and entropy coding as the example. Quantization is required, since storing any continuous value deterministically requires an infinite number of bits.

However, surprisingly, there exist methods to compress a continuous value without quantization, provided we are willing to accept some level of stochasticity. To be more specific, these methods encode a continuous but stochastic sample $\boldsymbol{z} \sim q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\boldsymbol{x})$. Li and Gamal (2018) showed that assuming we use an appropriate zeta distribution for q, the coding cost is upper-bounded by

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{z}\sim p_{\mathbf{x}}\cdot q_{\mathbf{z}|\mathbf{x}}}[|\mathcal{M}(\boldsymbol{z})|] \leq I[\mathbf{x},\mathbf{z}] + \log_2(I[\mathbf{x},\mathbf{z}]+1) + 4.732, \tag{2.7}$$

where $p_{\mathbf{x}} \cdot q_{\mathbf{z}|\mathbf{x}}$ represents the joint distribution of $p_{\mathbf{x}}$ and $q_{\mathbf{z}|\mathbf{x}}$.

Algorithms encoding such a sample are commonly known as *Channel Simulation*, *Reverse Channel Coding*, or *Relative Entropy Coding* [Theis and Yosri (2022); Yang et al. (2022)]. In the following, we refer to it as Relative Entropy Coding, or REC.

Here we look at two examples of REC, namely Minimal Random Coding (MRC), and Ordered Random Coding (ORC). Both of them assume the encoder and decoder have access to the prior distribution $p_{\mathbf{z}}$.

Minimal Random Coding: MRC encodes a sample following $q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\boldsymbol{x})$ *approximately*. To achieve this, the sender first draws

$$N = 2^{\lfloor D_{\mathrm{KL}}[q_{\mathbf{z}|\mathbf{x}}||p_{\mathbf{z}}]+t \rfloor} \tag{2.8}$$

samples from the prior $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)}, \ldots, \boldsymbol{z}^{(N)} \sim p_{\mathbf{z}}$ using a pseudo random generator with a seed shared with the receiver, and then calculate the importance weight, given by

$$w_n = \frac{q_{\mathbf{z}|\mathbf{x}}(\boldsymbol{z}^{(n)}|\boldsymbol{x})}{p_{\mathbf{z}}(\boldsymbol{z}^{(n)})} \tag{2.9}$$

for each sample $\boldsymbol{z}^{(n)}$. After that, the sender randomly samples an index n^* from a discrete distribution $P(n) \propto w_n$. Note that the index is uniformly distributed, and thus can be encoded with $\log_2 N$ bits. To decode this sample, the receiver simply draws N samples from the prior with the same random seed, and directly takes the n^*-th one.

Ordered Random Coding: ORC also generates N samples in the same way as MRC. However, sample important weights are permuted in a similar fashion to the Gumbel-max trick [Papandreou and Yuille (2011)].

To be more specific, besides the samples from the prior, the sender also draws a sequence of Gumble noises $G^{(n)}$, and sorts them in decreasing order such that $\tilde{G}^{(1)} \geq \tilde{G}^{(2)} \geq \ldots \geq \tilde{G}^{(N)}$. The importance weights of each sample is given by

$$\log w_n = \log \frac{q_{\mathbf{z}|\mathbf{x}}(\boldsymbol{z}^{(n)}|\boldsymbol{x})}{p_{\mathbf{z}}(\boldsymbol{z}^{(n)})} + \tilde{G}^{(n)}. \tag{2.10}$$

Then the sender takes the index n^* with the largest weight, i.e.,

$$n^* = \underset{n=1,2,\ldots,N}{\arg\max} \log w_n. \tag{2.11}$$

Note that by permuting the weights, n^* is no longer uniformly distributed, and thus can be encoded with bits $\leq \log_2 N$. Therefore, ORC is strictly more efficient than MRC [Theis and Yosri (2022)].

Note that the sample encoded by ORC also follows $q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\boldsymbol{x})$ *approximately*. However, Theis and Yosri (2022) showed that, if $\tilde{q}$ represents the distribution of the samples encoded by ORC, the total variation distance, defined as

$$D_{\mathrm{TV}}[\tilde{q}, q_{\mathbf{z}|\mathbf{x}}] = \frac{1}{2} \int \left| \tilde{q}(\mathbf{z}) - q_{\mathbf{z}|\mathbf{x}}(\mathbf{z}|\boldsymbol{x}) \right| \mathrm{d}\mathbf{z}, \tag{2.12}$$

vanishes exponentially quickly when $N = 2^{\lfloor D_{\mathrm{KL}}[q_{\mathbf{z}|\mathbf{x}}||p_{\mathbf{z}}]+t \rfloor}, t \to \infty$. See Appendix A.2 for a more formal statement.

ORC is also known as *depth-limited, global-bound A* coding* [Flamich et al. (2022)], to which we refer as A* coding hereafter. We provide its encoding and decoding process in Algorithm 6 and Algorithm 7 in Appendix A.1 for completeness.

2.2 Deep Learning

In the following, we shift our attention to deep learning. Based on (artificial) neural networks, which has been shown to be highly efficient in capturing complex patterns in data, deep learning is undergoing an explosive rise and revolutionizing many fields. In this section, we focus our introduction on the most relevant topics, including neural networks, Bayesian Neural Networks (BNNs), and Implicit Neural Representations (INRs).

2.2.1 Neural Networks

Neural networks can be seen as a form of linear model, with stacks of learnable non-linear basis functions [Bishop (2006)]. One common choice of the neural network architecture is Multi-layer Perceptron (MLP). Precisely, an MLP models a mapping

$$f : \mathbb{R}^{d_x} \to \mathbb{R}^{d_y}, \tag{2.13}$$

where d_x and d_y represent the dimensionality of the input and the output, respectively. For an L-layer MLP, the mapping is decomposed into $L-1$ sub-mappings between successive layers, i.e.,

$$f = f^{[1]} \circ f^{[2]} \circ \ldots \circ f^{[L-1]}. \tag{2.14}$$

Each of the sub-mapping $f^{[l]} : \mathbb{R}^{d_l} \to \mathbb{R}^{d_{l+1}}$ is parameterized by a weight matrix $\boldsymbol{W}^{[l]} \in \mathbb{R}^{d_l \times d_{l+1}}$ and a bias vector $\boldsymbol{b}^{[l]} \in \mathbb{R}^{d_{l+1}}$. Formally, if $\boldsymbol{x}^{[l]}$ denotes the input of the l-th layer, and $\boldsymbol{y}^{[l]}$ denotes the output, then

$$\boldsymbol{y}^{[l]} = f^{[l]}(\boldsymbol{x}^{[l]}) = \phi(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]} + \boldsymbol{b}^{[l]}), \tag{2.15}$$

where $\phi(\cdot)$ represents an element-wise non-linear activation function. The output of one layer will then be fed into the next layer as the input, i.e., $\boldsymbol{x}^{[l+1]} = \boldsymbol{y}^{[l]}$.

The choice of ϕ is one hyperparameter. The most common choices of ϕ, including Rectified Linear Unit (ReLU) [Nair and Hinton (2010)], Sigmoid and Tanh, are illustrated

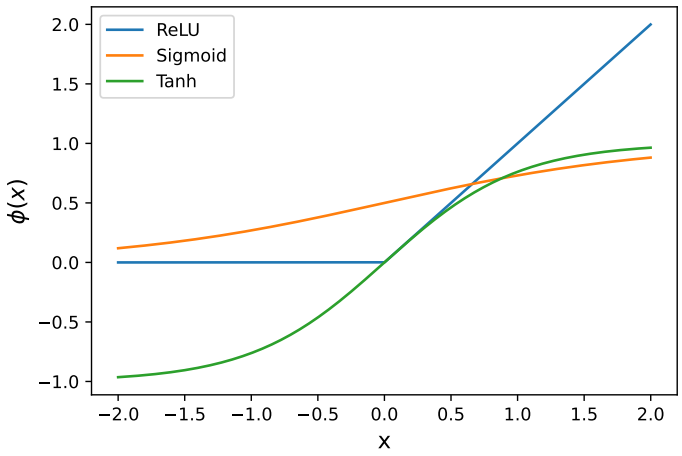


Fig. 2.1 Visualization of three common choices of the non-linear activation functions.

in Fig 2.1. We will revisit the choice in Section 2.2.3 when introducing Implicit Neural Representations.

From now on, unless stated otherwise, we use $\boldsymbol{w}$ to denote the vector concatenating all parameters in the network, i.e., $\boldsymbol{w} = \left[\text{Vec}(\boldsymbol{W}^{[1]})^\top, \boldsymbol{b}^{[1]\top}, ..., \text{Vec}(\boldsymbol{W}^{[L-1]})^\top, \boldsymbol{b}^{[L-1]\top}\right]^\top$ for brevity's sake. Besides, we use Roman letters (e.g. $\mathbf{w}, \mathrm{w}$) for symbolic representation of random variables and italicized letters (e.g. $\boldsymbol{w}, w$) for their realizations.

2.2.2 Bayesian Neural Networks

A Bayesian Neural Network (BNN) is the probabilistic variant of a standard neural network. Instead of learning a set of point estimators of the network parameters, BNNs infer their posteriors given the training data by Bayes Rule. More formally, assuming a prior distribution over the network parameters $p(\mathbf{w})$ and a set of training observations $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, the posterior distribution is given by

$$p(\mathbf{w}|\mathcal{D}) \propto p(\mathbf{w})p(\boldsymbol{y}_1, ..., \boldsymbol{y}_N|\mathbf{w}, \boldsymbol{x}_1, ..., \boldsymbol{x}_N). \tag{2.16}$$

However, unfortunately, Equation (2.16) is intractable in most case. Numerous researches have been done to address the intractability. There mainly exist two paradigms, one is to approximate the intractable $p(\mathbf{w}|\mathcal{D})$ by a tractable $q(\mathbf{w})$, including variational inference [Blundell et al. (2015); Graves (2011); Kingma and Welling (2014); Kingma et al. (2015)], Laplacian approximation [Neal (1995); Ritter et al. (2018)], etc.; the other is to simulate samples from $p(\mathbf{w}|\mathcal{D})$, such as Hybrid Monte Carlo (HMC) [Brooks et al. (2011);

Duane et al. (1987)], Stochastic Gradient Langevin Dynamics (SLGD) [Welling and Teh (2011)], etc.

Here, we look at variational inference since it is closely related to our work. Variational inference uses $q(\mathbf{w})$ to approximate the true posterior by minimizing the KL divergence from $p(\mathbf{w}|\mathcal{D})$ to $q(\mathbf{w})$, i.e.,

$$\arg\min_{q(\mathbf{w})} D_{\mathrm{KL}}[q(\mathbf{w})||p(\mathbf{w}|\mathcal{D})]. \tag{2.17}$$

However, due to the intractability of $p(\mathbf{w}|\mathcal{D})$, the KL divergence is also intractable. Therefore, VI maximizes $\Big(\log p(\mathcal{D}) - D_{\mathrm{KL}}[q(\mathbf{w})||p(\mathbf{w}|\mathcal{D})]\Big)$, since the marginal distribution of data $p(\mathcal{D})$ is a constant w.r.t. any choices of q. This objective function is known as Evidence Lower Bound (ELBO), with a better-known form, given by

$$\mathrm{ELBO} = \underbrace{\mathbb{E}_{\boldsymbol{w}\sim q(\mathbf{w})}[p(\mathcal{D}|\mathbf{w})]}_{\text{Data-fit term}} - \underbrace{D_{\mathrm{KL}}[q(\mathbf{w})||p(\mathbf{w})]}_{\text{Penalty term}}. \tag{2.18}$$

It is also common to down-weight the penalty term by β, known as β-ELBO, defined as

$$\beta\text{-ELBO} = \mathbb{E}_{\boldsymbol{w}\sim q(\mathbf{w})}[p(\mathcal{D}|\mathbf{w})] - \beta \cdot D_{\mathrm{KL}}[q(\mathbf{w})||p(\mathbf{w})]. \tag{2.19}$$

A natural question is how to optimize this objective *functional*. In practice, we first parameterize q within a tractable distribution family, and perform SGD on the parameters. For example, the most common parameterization is mean field (fully-factorized) Gaussian, i.e., $q = \mathcal{N}(\boldsymbol{\mu}_q, \mathrm{diag}(\boldsymbol{\sigma}_q^2))$.

Another difficulty lies in the calculation of the expectation $\mathbb{E}_{\boldsymbol{w}\sim q(\mathbf{w})}[p(\mathcal{D}|\boldsymbol{w})]$, which, in most cases, has no analytical form. To address this, it is typical to use a Monte Carlo estimator, employing the *reparametrization trick* to ensure that the gradient can back-propagate through the random samples, as introduced by Kingma and Welling (2014). This approach is also referred to as Bayes by Backprop (BbB) in some contexts.

However, $q(\mathbf{w})$ over the weights in BNNs can be high-dimensional. As a result, the Monte Carlo estimator will have a large variance, which slows down the convergence during training and may also negatively impact the performance. To address this, Kingma et al. (2015) proposed *local reparametrization trick*. When dealing with mean field Gaussian posteriors, this technique enables us to translate the global parameters' uncertainty into local noise, which is independent across data in one mini-batch. As a result, it effectively reduces variance.

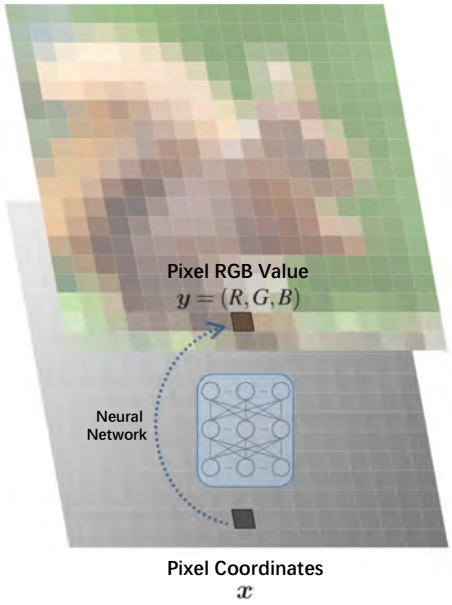


Fig. 2.2 Illustration of an INR applied to an image.

2.2.3 Implicit Neural Representations

Implicit Neural Representations (INRs) refer to continuous and differentiable signal representations that are implicitly defined by neural networks [Sitzmann et al. (2020)]. Its core idea is to view each data point as a function that maps coordinates to the signal values.

Consider the example of an image, as depicted in Fig 2.2. In the context of INRs, we can interpret the image as a function that maps each pixel's coordinates to its corresponding (R,G,B) values. The neural network fitted to this mapping is referred to as an Implicit Neural Representation of the image. It is worth noting that, from this perspective, a single *data point* is essentially viewed as a *dataset* consisting of coordinate-value pairs.

The most common architecture of INRs is the sinusoidal representation networks (SIREN), introduced by Sitzmann et al. (2020). It is a shallow MLP with sine as the activation function, i.e., $\phi(x) = \sin(\omega_0 x)$, where ω_0 is a hyperparameter typically set to $30-50$.

2.3 Related Works

In this section, we briefly review related works, including recent INR-based codecs, and MIRACLE (**Mi**nimal **Ra**ndom **C**ode **Le**arning) [Havasi et al. (2019)] which compresses Bayesian Neural Networks with Minimal Random Coding.

2.3.1 Data Compression with INRs

In this section, we first introduce recent INR-based codecs respectively, followed by a summary in Table 2.1.

COIN [Dupont et al. (2021)]

Dupont et al. (2021) proposed COIN (**Co**mpression with **I**mplicit **N**eural representations), applying INRs to image compression for the first time. When encoding an image, COIN fits an INR to an image, by minimizing the MSE between the predictions and the ground truth. Then COIN simply quantizes the INR parameters to 16 bits, without further compression such as entropy coding.

Bit-rates in COIN are controlled by Bayesian optimization among all valid network sizes. Precisely, given a bit-rate budget, COIN first lists all valid combinations of network depth and width. For instance, for images in the Kodak dataset [Kodak (1993)] (512×768 pixels) at 0.3 bpp, valid architectures include INRs with 10 layers of width 28, 7 layers of width 34 and so on. Then COIN runs a hyperparameter search over learning rates and architectures, on a single image from the dataset, by Bayesian optimization. The outcome will be used for all images in the same dataset at this bit-rate.

Without entropy coding or learning distributions over INR parameters, COIN outperforms JPEG at low bit-rates. However, the encoding is slow, because it involves thousands of gradient descent iterations to compress a single image. Besides, the framework is not yet competitive with state-of-the-art codecs at that time. Despite its limitations, COIN points out a potential direction for applying INR to compression, laying the foundations for all the following works.

COIN++ [Dupont et al. (2022)]

As the sequel to COIN, COIN++ partially addresses the problems of COIN.

By leveraging Model-Agnostic Meta-Learning (MAML) [Finn et al. (2017)], COIN++ reduces the encoding time significantly. For each test image, COIN++ only requires 10 iterations of gradient descent to converge. Besides, COIN++ learns a base network using the training set and subsequently employs modulations on this network to encode individual test data points, which enables information sharing between networks and thus increases the compression efficiency. Also, COIN++ uses entropy coding to compress quantized modulations. The quantization is applied uniformly within 3 standard deviations.

The authors also proposed their own modulation parameterization. Given a meta-learned base network, for each test datum, COIN++ only apply shifts $\boldsymbol{\beta}^{[l]}$ as the modulation, by

$$\sin(\omega_0(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]} + \boldsymbol{b}^{[l]} + \boldsymbol{\beta}^{[l]})) \tag{2.20}$$

at each layer l. To further enhance compression, instead of directly learning $\{\boldsymbol{\beta}^{[l]}\}_{l=1}^{L}$, COIN++ learns a latent vector which is linearly mapped to the modulations $\{\boldsymbol{\beta}^{[l]}\}_{l=1}^{L}$.

The authors applied COIN++ to a wide range of modalities, spanning images, audio, MRI scans, and climate data. However, due to the instability of meta-learning, when dealing with large target data, the authors cropped large data into smaller patches, and then compressed these patches individually, which negatively impacts the performance.

As the authors pointed out, the main drawback of COIN++ is the limited stability and scalability by meta-learning. Also, although COIN++ demonstrates applicability to multiple modalities, it does not outperform codecs specialized to each modality.

Furthermore, Huang and Hoefler (2023) pointed out the climate data compression in COIN++ assumes an infinite number of time slices, which does not align with the real-world climate data. Taking this into account, the actual compression ratio should be below $50\times$ instead of the reported $3000\times$ [Huang and Hoefler (2023)].

Strümpler et al. (2022)

Strümpler et al. (2022) proposed an approach using meta-learned INRs for image compression. Unlike COIN++, where meta-learning is applied in the modulation space, this method directly applies meta-learning in the network parameter space. Specifically, this approach first meta-learns a set of initial INR parameters, denoted as $\boldsymbol{w}_0$. For each test data point, the sender learns the updates to the parameters $\Delta\boldsymbol{w}$, and then quantizes $\Delta\boldsymbol{w}$ to $\Delta\tilde{\boldsymbol{w}}$, followed by entropy coding. To enhance compression, the author trained the network with L_1 regularization. During the decoding process, the image is reconstructed using the INR with the parameters $\boldsymbol{w} = \boldsymbol{w}_0 + \Delta\tilde{\boldsymbol{w}}$.

Moreover, the authors employed more sophisticated quantization in this approach. Specifically, the authors quantized each weight tensors with a different precision, and utilized AdaRound [Nagel et al. (2020)] to decide whether to round a weight up or down. Also, the authors fine-tuned the quantized weights using Quantization Aware Training (QAT), to avoid performance degradation.

MSCN [Schwarz and Teh (2022)]

Schwarz and Teh (2022) proposed MSCN (**M**eta-Learning **S**parse **C**ompression **N**etworks). In this work, the authors adopted a technique proposed by Louizos et al. (2018), introducing a reparameterized L_0 objective using stochastic gates on parameters, to sparsitize the network.

The authors proposed two setups, and investigated different applications of MSCN. Here we only present the setup related to compression (Section 3.3.2 and 5.3 in [Schwarz and Teh (2022)]).

First, MSCN meta-learns a base network, parameterized by $\boldsymbol{\theta}$, and the distribution parameters $\boldsymbol{\phi}$. A specific datum is represented by a set of smoothly gated modulations $\delta\boldsymbol{\theta} = \{\boldsymbol{m}^{[l]} \odot \boldsymbol{z}^{[l]}\}_{l=1}^{L}$ on the base network, where each $\boldsymbol{z}^{[l]} = \min(1, \max(0, \boldsymbol{s}))$ and $\boldsymbol{s}$ follows the Hard concrete distribution $q(\mathbf{s}|\boldsymbol{\phi})$. The modulation in MSCN has the same form as that in COIN++, i.e., only shift is applied.

When fitted to a specific datum $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$, $\delta\boldsymbol{\theta}$ is learned by optimizing:

$$\mathcal{L} = \underbrace{\mathbb{E}_{\boldsymbol{s}\sim q(\mathbf{s}|\boldsymbol{\phi})}\left[\sum_{i=1}^{N} ||\boldsymbol{y}_i - \hat{\boldsymbol{y}}_i||_2^2\right]}_{\text{Distortion}} + \lambda \underbrace{\sum_{j=1}^{\dim(\boldsymbol{\phi})} \left(1 - Q(s_j \leq 0|\boldsymbol{\phi})\right)}_{\text{Penalty term enforcing sparsity}}. \tag{2.21}$$

Q denotes the cumulative distribution function (CDF) of q, and $\hat{\boldsymbol{y}}_i$ represents the prediction of $\boldsymbol{x}_i$ using modulations $\delta\boldsymbol{\theta}$ on the base network. We omit the dependency in the notation for simplicity. Subsequently, MSCN quantizes and entropy-codes the modulations following COIN++'s settings.

However, it is important to note that when dealing with large images in Kodak dataset with patches, the authors evaluated the PSNR of each patch independently without reconstructing them back to the entire image. As a result, the reported results on the Kodak dataset may not be directly comparable with other approaches.

VC-INR [Schwarz et al. (2023)]

Schwarz et al. (2023) proposed VC-INR (**V**ariational **C**ompression of **I**mplicit **N**eural **R**epresentations), which employs variational autoencoder to compress low-rank modulations. VC-INR has two key components:

1. *low-rank soft gated modulations*: VC-INR also begins with a base network learned by MAML, and applies modulation for each test datum. Different from COIN++ and

MSCN, the modulation is defined as

$$\sin(\omega_0(\boldsymbol{G}_{\text{low}}^{[l]} \odot \boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]} + \boldsymbol{b}^{[l]})). \tag{2.22}$$

Assuming $\boldsymbol{G}_{\text{low}}^{[l]} \in \mathbb{R}^{d\times d}$, then

$$\boldsymbol{G}_{\text{low}}^{[l]} = \text{sigmoid}(\boldsymbol{U}_{\text{low}}^{[l]}\boldsymbol{V}_{\text{low}}^{[l]\top}), \quad \text{where } \boldsymbol{U}_{\text{low}}^{[l]}, \boldsymbol{V}_{\text{low}}^{[l]} \in \mathbb{R}^{d\times m}, \tag{2.23}$$

with $m \ll d$ to enforce a lower rank explicitly.

The authors employed LayerNorm [Ba et al. (2016)], residual connections [He et al. (2015)] with large layer widths, and the sigmoid applied to $\boldsymbol{G}_{\text{low}}^{[l]}$ to stabilize the meta-learning.

2. *variational compression of modulations*: Following the deep-factorized prior introduced by Ballé et al. (2017), the authors adopted a VAE to compress the modulations. The entire setup of the VAE remains consistent with [Ballé et al. (2017)], except the encoder and decoder are defined by residual Multi-layer Perceptrons (MLPs) with SeLU activations [Klambauer et al. (2017)]. During training, the author approximated quantization with uniform noise $\mathcal{U}(-\frac{1}{2}, \frac{1}{2})$. Besides, although the VAE is applied to the modulations, the distortion is directly measured between the reconstructed image and the target image.

In the paper, the authors applied VC-INR to various data modalities, including image, audio, and video, showcasing strong performance compared to other INR-based codecs.

To summarize, we offer a comparative review of these methods in Table 2.1.

2.3.2 Probabilistic Model Compression with REC

Another work that closely related to our method is MIRACLE (**Mi**nimal **Ra**ndom **C**ode **Le**arning), proposed by Havasi et al. (2019), to compress Bayesian Neural Networks with Minimal Random Coding (MRC).

Specifically, the authors introduced a prior $p(\mathbf{w})$ on the network parameters, and inferred the variational posterior $q(\mathbf{w})$ by maximizing the β-ELBO:

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{w}\sim q(\mathbf{w})}[\log p(\mathcal{D}|\boldsymbol{w})] - \beta \cdot D_{\text{KL}}[q(\mathbf{w})||p(\mathbf{w})]. \tag{2.24}$$

	Parameterization	MAML	Compression	Modalities
COIN	a single INR for each test datum	✗	uniform quantization to 16 bits	image
COIN++	a base network with modulation $\sin(\omega_0(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]}+\boldsymbol{b}^{[l]}+\boldsymbol{\beta}^{[l]}))$	✓	uniform quantization within 3 std; entropy coding	image, audio, medical data, climate data
Y.S. et al.*	an initial network with updates $\boldsymbol{w}=\boldsymbol{w}_0+\Delta\tilde{\boldsymbol{w}}$	✓	quantization with tensor-specific precision; AdaRound; QAT; entropy coding	image, 3D shape**
MSCN	a base network with modulation $\sin(\omega_0(\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]}+\boldsymbol{b}^{[l]}+\boldsymbol{m}^{[l]}\odot\boldsymbol{z}^{[l]}))$	✓	uniform quantization within 3 std; entropy coding	image***
VC-INR	a base network with modulation $\sin(\omega_0(\boldsymbol{G}^{[l]}_{\text{low}}\odot\boldsymbol{W}^{[l]}\boldsymbol{x}^{[l]}+\boldsymbol{b}^{[l]}))$ $\boldsymbol{G}^{[l]}_{\text{low}}=\text{sigmoid}(\boldsymbol{U}^{[l]}_{\text{low}}\boldsymbol{V}^{[l]\top}_{\text{low}})$	✓	variational compression with deep-factorised prior; entropy coding	image, audio, video, climate data

Table 2.1 Comparative review of current INR-based codecs. *Strümpler et al. (2022). **The only goal of 3D shape compression in this paper is to show the transferability of the approach. ***In MSCN paper, the authors applied the methods to multiple data modalities, but only evaluated on images for compression.

The first term on RHS is the data-fitting term, reflecting the prediction accuracy. The second term is an approximation of the coding cost of REC.

After learning the variational posterior, the author adopted MRC to encode a sample $\boldsymbol{w}\sim q(\mathbf{w})$. In order to keep the MRC sample size $N=2^{\lfloor D_{\text{KL}}[q(\mathbf{w})||p(\mathbf{w})]+t\rfloor}$ manageable, the authors partitioned $\mathbf{w}$ into blocks and compress them separately. Besides, the authors also proposed to train the model after the compression of each block. These approaches are borrowed into our framework, and play crucial roles in both COMBINER and COMBINER+. We will describe them with more details when introducing our approaches.

Besides the encoded samples, in this work, the prior $p(\mathbf{w})$ also needs to be transmitted to the receiver. To avoid using excessive additional bits, the authors set the prior to be a layer-wise isotropic Gaussian, centred around $\mathbf{0}$. Therefore, the sender only needs to transmit one float number per layer, corresponding to the prior variance.

Chapter 3

COMBINER: Compression with Bayesian Implicit Neural Representations

In the last chapter, we introduced Relative Entropy Coding (REC) and Implicit Neural Representations (INRs). We also looked at how INRs are being used for data compression. In this chapter, we introduce COMBINER (**Com**pression with **B**ayesian **I**mplicit **Ne**ural **R**epresentations), which compresses INR parameters by REC, resulting in a substantial improvement of its performance.

To motivate of our approach, we restate the *operational* rate-distortion function from the perspective of INRs: assuming an image with N pixels, represented as a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\boldsymbol{x}_i$ and $\boldsymbol{y}_i$ represent the coordinate vector and the RGB value of the i-th pixel. When compressing this image, we hope to minimize the distortion between the targets $\{\boldsymbol{y}_i\}_{i=1}^N$ and the predictions $\{\hat{\boldsymbol{y}}_i\}_{i=1}^N$, and meanwhile control the code cost of the INR parameters to be smaller than some budget C.

Most current INR-based codecs treat this as two separate tasks: they first fit an INR as well as possible, and then compress the quantized parameters. As a result, these methods cannot directly optimize the coding cost, and simply quantizing the parameters to a fixed precision will impact the pre-learned fitting. Although works like MSCN [Schwarz and Teh (2022)] attempt to control the rate by enforcing sparsity during training, their penalty term is not a well-defined entropy model directly linked to the coding cost.

Is there a way to optimize the coding cost, and obviate the need for post-training quantization, thus avoiding potential performance degradation? This reminds us of Relative Entropy Coding. It, on the one hand, compresses a continuous value without quantization; on the other hand, offers an upper bound on the coding cost which we can optimize directly. More precisely, if we introduce a prior $p(\mathbf{w})$ (abbreviated as $p_{\mathbf{w}}$) and a posterior distribution $q(\mathbf{w})$

(abbreviated as $q_{\mathbf{w}}$) over network parameters $\mathbf{w}$, the coding cost by REC, as discussed in Section 2.1.2, is approximately bounded by the Kullback-Leibler (KL) divergence $D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}]$.

Therefore, given a certain distortion measure Δ and the coding budget C, the operational rate-distortion function of lossy compression can be re-formalized into the constrained optimization problem:

$$\min_{q_{\mathbf{w}}} \mathbb{E}_{\boldsymbol{w}\sim q_{\mathbf{w}}}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Delta(\boldsymbol{y},\hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{w}})\right], \quad \text{s.t. } D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}] \leq C, \tag{3.1}$$

where $\hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{w}}$ is the output of INR given $\boldsymbol{x}$ and a parameter sample $\boldsymbol{w} \sim q_{\mathbf{w}}$. After $q_{\mathbf{w}}$ is learned, we can naturally perform REC to compress a sample from it without exceeding the budget C.

In a nutshell, COMBINER can be summarized as:

learning a posterior distribution of the INR parameters, and compressing a sample from this distribution by REC.

Our approach has several advantages. As we already discussed, it enables a joint optimization of the distortion and the coding cost. Also, as we will show, our approach can adaptively activate or prune hidden units at different bitrates, avoiding tedious tuning of the network size. Besides, our approach demonstrates strong robustness during the prior learning stage since it does not rely on MAML (Model-Agnostic Meta-Learning [Finn et al. (2017)]) which involves second-order gradient and therefore can be unstable.

Sections below are organized as follows. Section 3.1 describes COMBINER in details. Section 3.2 shows its performance on two image datasets. This section also includes analysis to better understand our model, and ablation studies to verify the effectiveness of our methods. We close this chapter with a brief recap in Section 3.3. We will discuss more about the limitations in Chapter 5 at the end of this thesis.

3.1 Methods

Although Equation (3.1) serves as the objective function, there still exist three gaps to fill in before turning it into a practical compression algorithm:

- given a set of training data, how to find a prior $p_{\mathbf{w}}$ that works for the test data to be compressed within the same domain;
- after $p_{\mathbf{w}}$ is learned, given a datum represented as $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$ to be compressed, how to find the posterior $q_{\mathbf{w}}$ that satisfies Equation (3.1);

- after $q_{\mathbf{w}}$ is learned, how to encode a sample from it in practice.

This section explains how COMBINER addresses these questions. We first describe the inference of the posterior distribution, followed by the introduction of prior learning and the compression algorithm. We rearrange their order, since the prior learning stage also involves the posteriors on the training data. In the end, we summarize the entire pipeline of COMBINER.

3.1.1 Posterior Inferring by Stochastic Variational Inference

Assuming we have a prior over INR parameters $p_{\mathbf{w}}$, and want to compress a new-coming datum $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^{N}$. To solve the constraint optimization problem in Equation (3.1), we introduce a slack variable β and optimize its Lagrangian dual, which yields:

$$\mathcal{L}(q_{\mathbf{w}}, p_{\mathbf{w}}, \mathcal{D}) = \mathbb{E}_{\boldsymbol{w}\sim q_{\mathbf{w}}}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{w}})\right] + \beta \cdot D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}] + \text{const.} \tag{3.2}$$

This objective function naturally combines the distortion and the coding cost together, and balance their importance with a hyperparameter β. In the following content, we will refer to it as the *rate-distortion objective*.

Note that this has the same form as the negative β-Evidence Lower Bound (β-ELBO) as discussed in Section 2.2.2, suggesting the applicability of the standard stochastic variational inference.

In COMBINER, we parameterize the variational posterior $q_{\mathbf{w}}$ as mean field (fully-factorized) Gaussian, i.e.,

$$q_{\mathbf{w}} = \mathcal{N}(\boldsymbol{\mu}_q, \mathrm{diag}(\boldsymbol{\sigma}_q^2)). \tag{3.3}$$

While the KL divergence on the RHS of Equation (3.2) can have analytical solution, the expectation of distortion is generally intractable. Therefore, we follow the convention of Bayesian Neural Networks, using the Monte Carlo estimator with local reparametrization trick [Kingma et al. (2015)] as discussed in Section 2.2.2. We use Adam [Kingma and Ba (2014)] to optimize the parameters $\boldsymbol{\mu}_q$ and $\boldsymbol{\sigma}_q$, whose gradients of are calculated by standard back-propagation[1].

[1] In order to ensure the positivity of the standard deviation, we transform it into the entire Euclidean space by a bijection ϕ^{-1}. We set $\phi(\cdot) = \mathrm{softplus}(\cdot)/6 = \log(\exp(\cdot)+1)/6$ in practice.

Algorithm 1 Learning the model prior

Require: Training data $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M\}$; slack variable β.

Initialize: $q_{\mathbf{w}}^{(m)} = \mathcal{N}\left(\boldsymbol{\mu}_q^{(m)}, \text{diag}\left(\boldsymbol{\sigma}_q^{(m)^2}\right)\right)$ of every training instance $\mathcal{D}_m$.

Initialize: $p_{\mathbf{w}} = \mathcal{N}\left(\boldsymbol{\mu}_p, \text{diag}\left(\boldsymbol{\sigma}_p^2\right)\right)$.

repeat until convergence

  for $m \leftarrow 1$ to M **do**

    $q_{\mathbf{w}}^{(m)} \leftarrow \arg\min_{q_{\mathbf{w}}^{(m)}} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m)$ ▷ Optimizing posteriors by SVI

  end for

  $p_{\mathbf{w}} \leftarrow \arg\min_{p_{\mathbf{w}}} \left\{\frac{1}{M}\sum_{m=1}^{M} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m)\right\}$ ▷ Update prior by Equation (3.8)

end repeat

Return: $p_{\mathbf{w}} = \mathcal{N}\left(\boldsymbol{\mu}_p, \text{diag}\left(\boldsymbol{\sigma}_p^2\right)\right)$.

3.1.2 Prior Learning by Coordinates Ascent

We now consider how to find a good prior $p_{\mathbf{w}}$. A prior that matches well with the data distribution enables us to allocate bits more efficiently, and thus is critical to guarantee COMBINER performs well in practice.

First, we determine the form of the prior. Since we parameterize the variational posterior $q_{\mathbf{w}}$ by mean field Gaussian, it is natural to use mean field Gaussian as the prior to ensure an analytical KL divergence. In standard applications of Bayesian Neural Networks, it is common that all dimensions of the network parameter share the same, fixed prior, e.g., $\mathcal{N}(0,1)$. However, this form of prior will cost a huge waste in terms of compression, since different dimensions in the network can exhibit varying importance. Therefore, we consider a *parameter-wise* prior with different means and standard deviations for each parameter, i.e.,

$$p_{\mathbf{w}} = \mathcal{N}(\boldsymbol{\mu}_p, \text{diag}(\boldsymbol{\sigma}_p^2)). \tag{3.4}$$

Second, we consider what a good prior means. For a training set with M training data points $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M\}$, we want to find a prior, so that their *average* rate-distortion objective is minimized, after their posteriors are inferred using this prior, i.e.,

$$p_{\mathbf{w}}^* = \arg\min_{p_{\mathbf{w}}} \left\{\frac{1}{M}\sum_{m=1}^{M}\left[\min_{q_{\mathbf{w}}^{(m)}} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m)\right]\right\}, \tag{3.5}$$

where we use $q_{\mathbf{w}}^{(m)}$ to represent the posterior corresponding to the m-th training instance.

In order to solve this nested optimization problem where $q_{\mathbf{w}}^{(m)}$ and $p_{\mathbf{w}}$ are dependent on each other, we propose a coordinate ascent algorithm that iteratively update and the prior and the posteriors. To begin, we randomly initialize the model prior and the posteriors, and alternate the following two steps:

1. **Optimize the variational posteriors:** we fix the prior $p_{\mathbf{w}}$ and optimize the posteriors of all training instances, as discussed in Section 3.1.1.

   Note that given the fixed prior, the optimization of the posteriors can be split into M independent optimization problems, which we can perform in parallel:

   $$\text{for each } m = 1, ..., M: \quad q_{\mathbf{w}}^{(m)} \leftarrow \arg\min_{q_{\mathbf{w}}^{(m)}} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m). \tag{3.6}$$

2. **Update the prior:** we fix the posteriors $\{q_{\mathbf{w}}^{(m)}\}_{m=1}^{M}$ from the last step, and update the model prior by computing:

   $$p_{\mathbf{w}} \leftarrow \arg\min_{p_{\mathbf{w}}} \left\{ \frac{1}{M} \sum_{m=1}^{M} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m) \right\}, \tag{3.7}$$

   When $q_{\mathbf{w}}^{(m)}$ and $p_{\mathbf{w}}^{(m)}$ are Gaussians as given by Equation (3.3) and (3.4), this admits a closed-form solution:

   $$\boldsymbol{\mu}_p = \frac{1}{M} \sum_{m=1}^{M} \boldsymbol{\mu}_q^{(m)}, \quad \boldsymbol{\sigma}_p = \sqrt{\frac{1}{M} \sum_{m=1}^{M} \left[\left(\boldsymbol{\sigma}_q^{(m)} \right)^2 + \left(\boldsymbol{\mu}_q^{(m)} - \boldsymbol{\mu}_p \right)^2 \right]} \tag{3.8}$$

We provide the full derivation of this procedure in Appendix B, and formalize this iterative updating in Algorithm 1. Note that Equation (3.6) also depends on the hyperparameter β. In practice, we learn multiple priors with a range of βs, corresponding to various bit-rates.

Since the gradient optimization in Step 1, as the standard variational inference of BNNs, is stable, and the closed-form update in Step 2 guarantees the objective function to decrease, this algorithm features a stable training process and does not have convergence issue.

3.1.3 Block-wise Compression with Progressive Fine-tuning

Algorithm 1 provides an efficient solution for learning the prior $p_{\mathbf{w}}$, which will be used to train the variational posterior $q_{\mathbf{w}}$ for a new datum to be compressed. In the following content, we will refer to the new datum as *test* datum.

Once $q_{\mathbf{w}}$ is learned for a test datum, COMBINER uses Relative Entropy Coding (REC) to directly encode a single random sample $\boldsymbol{w} \sim q_{\mathbf{w}}$ instead of quantizing a point estimator and entropy coding it. This approach was first proposed by Havasi et al. (2019) who used minimal random coding (MRC) to compress Bayesian Neural Networks. In COMBINER, we use A* coding since it is more efficient than MRC as discussed in Section 2.1.2. Similar to the strategies used by Havasi et al. (2019), we also partition the parameters into blocks, compress blocks separately, and perform training between the compression of each block.

Block-wise Compression

In order to compress a sample from $q_{\mathbf{w}}$ by A* coding, we need to draw $N = \left\lfloor 2^{D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}]+t} \right\rfloor$ number of samples $\boldsymbol{w}^{(1)}, \boldsymbol{w}^{(2)}, ..., \boldsymbol{w}^{(N)}$ from the prior $p_{\mathbf{w}}$, and select the best sample according to their (permuted) importance weights. However, when $\mathbf{w}$ is the vector containing all parameters in a neural network, the required sample size is infeasible for any reasonable $D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}]$. To handle this, we partition the vector $\mathbf{w}$ into K blocks: $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K$, and perform REC on each of the block separately. To avoid transmitting extra bits, this partition is shared between the sender and the receiver, and remains fixed for any datum to be compressed.

First, we discuss the consideration when partitioning the parameter vector $\mathbf{w}$. Let $\delta_k = D_{\mathrm{KL}}[q_{\mathbf{w}_k}||p_{\mathbf{w}_k}]$ denote the KL divergence for the k-th block. The coding cost for A* coding is bounded by $\left(\delta_k + \log_2(\delta_k + 1) + \text{const.}\right)$ bits. If δ_k is too small, the second and the third term will dominate the coding cost and lead to a huge overhead. On the other hand, if δ_k is too large, the required sample size $N_k = \left\lfloor 2^{\delta_k + t} \right\rfloor$ will still be computationally intractable. Therefore, it is important to ensure that the KL divergences of all blocks stay within a reasonable range. In fact, to guarantee that COMBINER's runtime is consistent, we would like the KL divergences across all blocks to be approximately equal. To this end, we set a bit-budget of κ bits per block. When encoding each block by A* coding, we draw $2^{\kappa+t}$ samples instead of $2^{\lfloor \delta_k + t \rfloor}$ samples from the prior.

Besides, to better ensure this budget on the test datum, we partition parameters according to the average KL divergence on the training dataset. To be more specific, given a set of training data $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M\}$, we first choose β according to the desired bitrate and learn a model prior $p_{\mathbf{w}}$ by Algorithm 1. We then calculate the average KL divergence for each dimension of $\mathbf{w}$ across all M training data, and allocate dimensions into blocks, so that the average divergence $\bar{\delta}_k$ of each block matches the coding budget, i.e., $\bar{\delta}_k \approx \kappa$. Unfortunately, allocating dimensions into blocks under this constraint is the NP-hard bin packing problem. In our experiments, we use the greedy next-fit bin packing algorithm with random permutation, as presented in Algorithm 2. We found that randomly assigning dimensions into blocks with a fixed size also works well in practice.

Algorithm 2 Partition the network parameters by Next-fit Bin Packing Algorithm

Require: Model prior $p_{\mathbf{w}}$, and the variational posteriors of the training data $\{q_{\mathbf{w}}^{(m)}\}_{m=1}^{M}$ learned by Algorithm 1; bit-budget κ for each block.

Initialize: $k \leftarrow 1$; ▷ Initialize number of blocks
Initialize: Empty list $\mathcal{B}_k = \{\}$; ▷ Initialize list of parameters in the current block
Initialize: $\bar{\delta} \leftarrow 0$. ▷ Initialize KL of current block to 0

repeat until all parameters are assigned
  Randomly pick one unassigned parameter w_j; ▷ Ensure the KL spreads evenly
  Calculate $\bar{d}_j \leftarrow \frac{1}{M}\sum_{m=1}^{M} D_{\mathrm{KL}}[q_{\mathrm{w}_j}||p_{\mathrm{w}_j}]$;
  if $\bar{\delta} + \bar{d}_j > \kappa$ **then** ▷ Check if current block is full
    $k \leftarrow k+1$;
    $\mathcal{B}_k \leftarrow \{\}$; ▷ Initialize a new block
    $\bar{\delta} \leftarrow 0$; ▷ Reset KL of the new block
  end if
  Append w_j to $\mathcal{B}_k$; ▷ Add current parameter to current block
  $\bar{\delta} \leftarrow \bar{\delta} + \bar{d}_j$; ▷ Update KL of current block
end repeat

Return: $\mathcal{B}_1, \mathcal{B}_2, \ldots$.

KL-aware Training

However, despite the effort we take to partition the parameters carefully, we can only ensure $\bar{\delta}_k \approx \kappa$ for each block *on average* even when the test data and the training data matches well. For a certain datum, it is unlikely that the actual KL divergence δ_k matches κ. Therefore, we impose block-wise β_k, and optimize a modified version of Equation (3.2), given by

$$\mathcal{L}(\{q_{\mathbf{w}_k}\}_{k=1}^{K}, \{p_{\mathbf{w}_k}\}_{k=1}^{K}, \mathcal{D}) = \mathbb{E}_{q_{\mathbf{w}_1} q_{\mathbf{w}_2} \cdots q_{\mathbf{w}_K}} \left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}}) \right] + \sum_{k=1}^{K} \beta_k \cdot \delta_k + \text{const.} \tag{3.9}$$

Here we omit the dependency between $\hat{\boldsymbol{y}}$ and $\boldsymbol{x}, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_K$ for simplicity.

To enforce the budget κ, each β_k is dynamically adjusted during optimization: every I iterations, we calculate the KL divergence δ_k for each block. If the KL for block k exceeds the coding budget, i.e., $\delta_k > \kappa$, we increase β_k to $\beta_k \times (1+\tau)$; if $\delta_k < \kappa - \varepsilon_\kappa$, we decrease β_k to $\beta_k/(1+\tau)$. This procedure is modified from Havasi et al. (2019) by introducing a buffer area ε_κ where we do not change β_k to stabilize the training. We present the detailed algorithm in Algorithm 3. In our experiments, unless stated, we set $t = 0$, $I = 15$, $\kappa = 16$ bits, $\varepsilon_\kappa = 0.4$ bits, and $\tau = 0.05$.

Algorithm 3 KL-aware Training

Require: Test datum $\mathcal{D}$; model prior $p_{\mathbf{w}}$ learned by Algorithm 1, and the β used in this prior learning process; parameter blocks $\mathcal{B}_1, ..., \mathcal{B}_K$ by Algorithm 2.

Initialize: $q_{\mathbf{w}_k} = \mathcal{N}(\boldsymbol{\mu}_{q_k}, \text{diag}(\boldsymbol{\sigma}^2_{q_k})), \forall k = 1, ..., K$; ▷ Initialize posterior for each block
Initialize: $\beta_k \leftarrow \beta, \forall k = 1, ..., K$. ▷ Initialize β_k for each block

for u in $1, 2, ...,$ NumIter **do**
  $\{q_{\mathbf{w}_k}\}_{k=1}^K \leftarrow$VariationalUpdates$\Big(\mathcal{L}(\{q_{\mathbf{w}_k}\}_{k=1}^K, \{p_{\mathbf{w}_k}\}_{k=1}^K, \mathcal{D})\Big)$;
  ▷ Update posteriors by SVI; $\mathcal{L}$ is defined in Equation (3.9)
  if $u \equiv 0 \pmod I$ **then**
    $\{\beta_k\}_{k=1}^K \leftarrow$ AdjustBeta$\Big(\{\beta_k\}_{k=1}^K, \{q_{\mathbf{w}_k}\}_{k=1}^K, \{p_{\mathbf{w}_k}\}_{k=1}^K\Big)$
    ▷ Update β_k by Algorithm 4
  end if
end for

Return: adjusted $\{\beta_k\}_{k=1}^K$; variational posteriors $\{q_{\mathbf{w}_k}\}_{k=1}^K$ for each block.

Algorithm 4 AdjustBeta($\{\beta_k\}, \{q_{\mathbf{w}_k}\}, \{p_{\mathbf{w}_k}\}$): adjust β_k according to δ_k.

Require: current $\{\beta_k\}_{k=1}^K$ values; model priors $\{p_{\mathbf{w}_k}\}_{k=1}^K$; variational posteriors $\{q_{\mathbf{w}_k}\}_{k=1}^K$.

for k in $1, 2, ..., K$ **do**
  Calculate $\delta_k \leftarrow D_{\text{KL}}[q_{\mathbf{w}_k} || p_{\mathbf{w}_k}]$;
  if $\delta_k > \kappa$ **then**
    $\beta_k \leftarrow \beta_k \times (1 + \tau)$; ▷ Increase β_k if budget is exceeded
  end if
  if $\delta_k < \kappa - \varepsilon_\kappa$ **then**
    $\beta_k \leftarrow \beta_k / (1 + \tau)$; ▷ Decrease β_k if budget is not fully occupied
  end if
end for

Return: updated $\{\beta_k\}_{k=1}^K$ values.

Fine-tuning between Compression of Blocks

To further improve the performance, after the compression of each block, we follow the approach proposed by Havasi (2021) to train the posteriors of not-yet encoded blocks. We will refer to this training process as *fine-tuning* in our content.

This fine-tuning not only corrects the error caused by A* coding, which only generate samples following the posterior *approximately*, but also allows us to obtain a better posterior sample from a richer variational family.

To provide an intuition how this process enrich variational family, we first consider the optimal posterior distribution $q^*_{\mathbf{w}}$ that minimizes the *rate-distortion objective* given in Equation (3.2). Ideally, we want to search among all valid distributions and select the best one as $q^*_{\mathbf{w}}$, which is clearly intractable. In practice, we restrict ourselves within a much smaller set, i.e., the set of all fully-factorized Gaussian distributions, which yields a rather crude approximation, where all the dependencies between dimensions are lost.

However, fine-tuning allows us to partially retrieve the dependency between blocks. Assuming the first blocks is A* encoded by the sample $\boldsymbol{w}_1$, the objective function of the following fine-tuning process becomes

$$\begin{aligned}&\mathcal{L}(\{q_{\mathbf{w}_k}\}_{k=2}^K, \{p_{\mathbf{w}_k}\}_{k=2}^K, \mathcal{D}|\boldsymbol{w}_1)\\ &\quad= \mathbb{E}_{q_{\mathbf{w}_2|\boldsymbol{w}_1} q_{\mathbf{w}_3|\boldsymbol{w}_1} \cdots q_{\mathbf{w}_K|\boldsymbol{w}_1}} \left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Delta(\boldsymbol{y},\hat{\boldsymbol{y}})\right] + \sum_{k=2}^K \beta_k \cdot D_{\mathrm{KL}}[q_{\mathbf{w}_k|\boldsymbol{w}_1} || p_{\mathbf{w}_k|\boldsymbol{w}_1}] + \text{const.}\end{aligned} \tag{3.10}$$

we use "$\cdot|\boldsymbol{w}_1$" to explicitly indicate that the fine-tuning of the posteriors for block $2,3,\ldots,K$ are conditional on the first sample $\boldsymbol{w}_1$. In total, we iterate the fine-tuning procedure K times, progressively conditioning on more blocks, until we obtain samples for all blocks $\{\boldsymbol{w}_k\}_{k=1}^K$. Due to the dependency introduced by fine-tuning, the final posterior is *autoregressive*, and thus approximates $q^*_{\mathbf{w}}$ much better[2]. In Section 3.2.3, we will provide examples to illustrate the effectiveness of fine-tuning.

We also adaptively adjust δ_k for not-yet compressed blocks every I iterations to ensure the bit-budget will not be exceeded after fine-tuning. We describe the entire compression procedure in Algorithm 3.

3.1.4 Entire Pipeline of COMBINER

In this section, we recap the entire pipeline of COMBINER. We also provide a visualization in Fig 3.1. In total, there are 4 steps:

1. Given a training dataset $\{\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_M\}$, we select an appropriate INR architecture, and run Algorithm 1 with different β values to obtain model priors for a range of rate-distortion trade-offs.

[2]Note that, although each conditional posterior distribution $q_{\mathbf{w}_k|\boldsymbol{w}_1}$ is parameterized by Gaussian, its dependency on the current sample $\boldsymbol{w}_1$ introduced by the fine-tuning is highly non-linear. Therefore, the final autoregressive posterior is not Gaussian any more. In fact, it is even more complex than fully-joint Gaussian.

Algorithm 5 Relative Entropy Coding with Fine-tuning

Require: Test datum $\mathcal{D}$; model prior $p_{\mathbf{w}}$; variational posteriors $\{q_{\mathbf{w}_k}\}_{k=1}^K$ and adjusted $\{\beta_k\}_{k=1}^K$ by Algorithm 3; the bit-budget κ.

for $\tilde{k}$ in $1, 2, ..., K$ **do**
  $n_{\tilde{k}}^*, \boldsymbol{w}_{\tilde{k}} \leftarrow$ A* Coding$(q_{\mathbf{w}_{\tilde{k}}}, p_{\mathbf{w}_{\tilde{k}}}, \kappa)$ ▷ Encode $\tilde{k}$-th block by A* coding
  for u in $1, 2, ...,$ FineTuneIter **do**
    $\{q_{\mathbf{w}_k}\}_{k=\tilde{k}+1}^K \leftarrow$ VariationalUpdates$\Big(\mathcal{L}(\{q_{\mathbf{w}_k}\}_{k=\tilde{k}+1}^K, \{p_{\mathbf{w}_k}\}_{k=\tilde{k}+1}^K, \mathcal{D} | \{\boldsymbol{w}_k\}_{k=1}^{\tilde{k}})\Big)$;
    ▷ Fine-tune uncompressed blocks by SVI
    if $u \equiv 0 \pmod I$ **then**
      $\{\beta_k\}_{k=\tilde{k}+1}^K \leftarrow$ AdjustBeta$\Big(\{\beta_k\}_{k=\tilde{k}+1}^K, \{q_{\mathbf{w}_k}\}_{k=\tilde{k}+1}^K, \{p_{\mathbf{w}_k}\}_{k=\tilde{k}+1}^K\Big)$;
      ▷ Update β_k for uncompressed blocks by Algorithm 4
    end if
  end for
end for

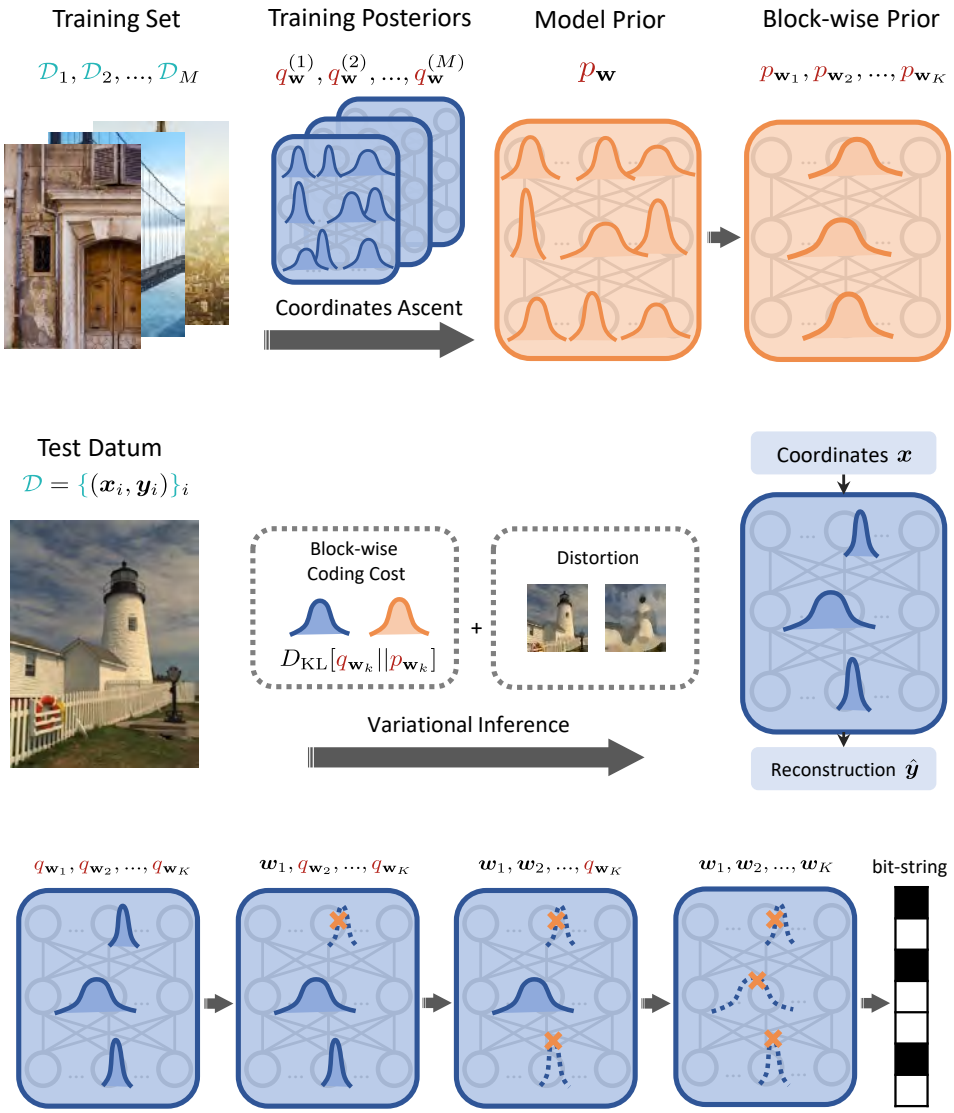
Return: $n_1^*, n_2^*, ..., n_K^*$ ▷ Return sample indices for each block

2. Given a coding budget κ, we partition the parameter vector $\mathbf{w}$ into K blocks $\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K$ by Algorithm 2. Note that for different model priors learned with different β will end up with different partitions.

3. For a new test datum $\mathcal{D}$ to be compressed, according to the desired rate-distortion trade-off, we select a prior and its corresponding partition. Then we run Algorithm 3 to infer the variational posterior $q_{\mathbf{w}_k}$ for each block. To ensure the coding budget κ, we run Algorithm 4 every I iterations to adjust β_k for each block.

4. Finally, we encode a single random parameter sample $\boldsymbol{w}_k \sim q_{\mathbf{w}_k}$ for each block k by A* Coding, and perform fine-tuning after the compression of each block, as described in Algorithm 5. Since the model prior, the parameter partition, and the pseudo random generator are shared with the receiver, we only need to transmit one index for each block, corresponding to the sample.

Note that steps 1, 2 are shared with the receiver, while steps 3, 4 only happen on the sender's side. After compression, the receiver can easily retrieve the sample $\boldsymbol{w}_k$ by its index n_k^* and the shared pseudo random generator.



(c) Step 4: Relative Entropy Coding with progressive fine-tuning.

Fig. 3.1 Overview of COMBINER's pipeline. (a) For a certain data modality (e.g., image in our case), COMBINER learns the model prior $p_{\mathbf{w}}$ using a set of training data $\mathcal{D}_1, ..., \mathcal{D}_M$. Then, COMBINER partitions parameters into blocks by Algorithm 2; the crossing arrows indicate the random permutation when partitioning blocks. (b) for a test datum $\mathcal{D}$, COMBINER run Algorithm 3 to learn the posterior $q_{\mathbf{w}_k}$ for each block; (c) COMBINER performs REC with fine-tuning to compress a sample for each block. In this figure, we colour priors in orange and posteriors in blue. The orange crosses in (c) indicate the fact that the sample by REC is actually drawn from the prior.

3.2 Experiments and Results

In this section, we test the performance of COMBINER on both the low-resolution Cifar-10 images [Krizhevsky (2009)] and the high-resolution Kodak dataset [Kodak (1993)]. Also, we perform a comprehensive study of our approach in Section 3.2.3. We illustrate that our model can adaptively activate or prune the INR hidden units to represent a specific signal. We also investigate the effectiveness of fine-tuning quantitively and qualitatively. Moreover, we conduct an ablation study of the model prior, and demonstrate the strong robustness of our approach against the number of training instances.

3.2.1 Experiment Settings

We evaluate COMBINER on two image datasets: Cifar-10 and Kodak. On both datasets, we adopt SIREN [Sitzmann et al. (2020)] as the INR architecture. Before feeding the coordinates $\boldsymbol{x}$ into INRs, we apply Fourier embeddings [Tancik et al. (2020)] to the coordinates. Specifically, for the coordinate $\boldsymbol{x} = [x_1, x_2]^\top$, we define its Fourier embeddings $\gamma(\boldsymbol{x}) \in \mathbb{R}^D$ as

$$\gamma(\boldsymbol{x}) = \Big[\cos(a^{0/d}\pi x_1), \cos(a^{0/d}\pi x_2), ..., \cos(a^{d/d}\pi x_1), \cos(a^{d/d}\pi x_2), \\ \sin(a^{0/d}\pi x_1), \sin(a^{0/d}\pi x_2), ..., \sin(a^{d/d}\pi x_1), \sin(a^{d/d}\pi x_2)\Big]^\top, \quad (3.11)$$

where $d = \lfloor \frac{D}{4} \rfloor - 1$ and $a = 1024$ in our experiments. Moving forward, we present the specific experiment settings for each of these two datasets:

Cifar-10: CIFAR-10 consists of 50,000 train and 10,000 test images with a resolution of 32×32. We randomly select 2,048 images from the training set to learn the model prior, and evaluated our model on all 10,000 test images. The model we use for Cifar has 4-layers, each with 16 hidden units. In total, there are 1,123 parameters in each INR.

Kodak: Kodak is a standard dataset used to evaluate compression models, comprising 24 high-resolution images with sizes of either 512×768 or 768×512. Since there was no specific training dataset available for Kodak, we randomly crop 512 patches, each having the same size as Kodak images, from the CLIC training set [Toderici et al. (2020)]. We train our model priors on these patches and then evaluate on 24 Kodak images. To cover a broader range of bit rates, we employ two distinct model sizes. The smaller model contains 12,675 parameters, while the larger model consists of 21,563 parameters.

For more information on the detailed model structures and additional experimental details, please refer to Table C.1 in Appendix.

3.2.2 Compression Performance

Rate-Distortion Curve

We present the Rate-Distortion (R-D) curve of our approach in Fig 3.2, showing the PSNR averaged across all test images at different bitrate. Specifically, we use the Mean Square Error (MSE) as the distortion measure, and calculate PSNR by

$$\text{PSNR} = 10 \cdot \log_{10} \left[\frac{255^2}{\text{MSE}} \right]. \tag{3.12}$$

Besides the baseline model COIN [Dupont et al. (2021)], we also compare our method with other INR-based codecs, including COIN++ [Dupont et al. (2022)], MSCN [Schwarz and Teh (2022)]. Additionally, we include results from VAE-based codecs such as [Ballé et al. (2018)] and [Cheng et al. (2020)], and traditional codecs including JPEG, JPEG2000, BPG for reference. We also present results of VC-INR [Schwarz et al. (2023)], a hybrid codec of VAE and INR. We can see our proposed method presents a significant gain compared to its baseline approach, COIN. Besides, when compared to other INR-based codecs, our model also exhibits competitive performance. However, there is still a gap between COMBINER and methods utilizing VAEs.

Also, as a qualitative evaluation of our method, we decode and visualize some sample images from both datasets. We show 5 images from Cifar-10 test set at 0.906 bpp and 4.453 bpp in Fig 3.3, and Kodak-03, Kodak-05 and Kodak-23 at 0.070 bpp and 0.293 bpp in Fig 3.4, Fig 3.5, Fig 3.6.

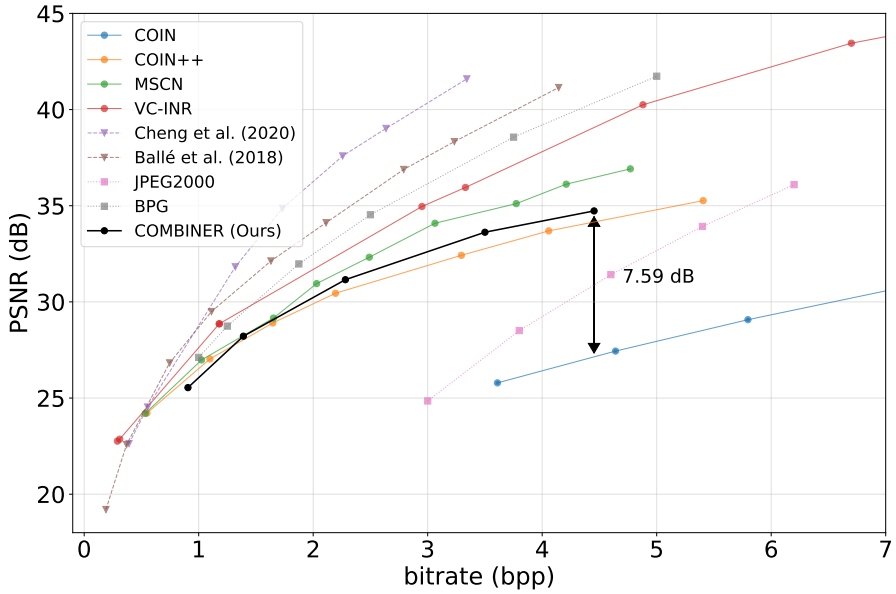
Encoding and Decoding Speed

We provide the compression speed of our method on Cifar and Kodak in Table D.1 and Table D.2 in Appendix D.1.

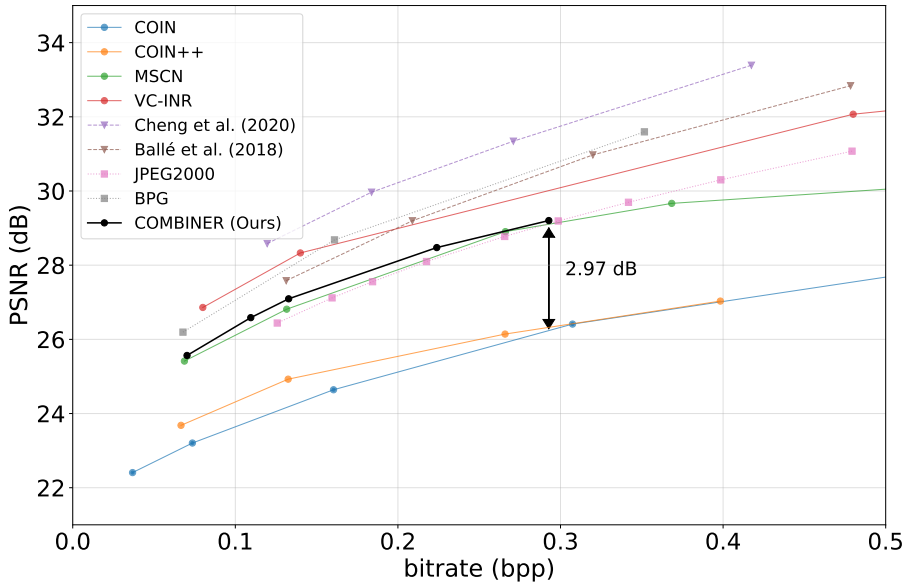
The encoding speed is measured on a single NVIDIA A100 (80GB) GPU. On Cifar, we compress images in batch, with a batch size of 500 images[3]. On Kodak, we compress each image separately. The decoding speed is measured per image on CPU.

Similar to COIN, our approach features a high encoding time complexity: the encoding period in our method takes thousands of iterations to converge, and REC with fine-tuning procedure also requires extra thousands of iterations. However, the decoding process is remarkably fast, even on CPU.

[3]We should note that the batch size does not influence the results. Individual INRs for images within a batch are optimized in parallel, and their gradients are not crossed.



(a) Cifar-10



(b) Kodak

Fig. 3.2 The Rate-Distortion Curve of COMBINER on Cifar-10 and Kodak datasets. The arrows illustrate the gains in comparison to the baseline model COIN. We use solid lines to denote INR-based methods, dotted lines for VAE-based methods, and dashed lines for classical methods.

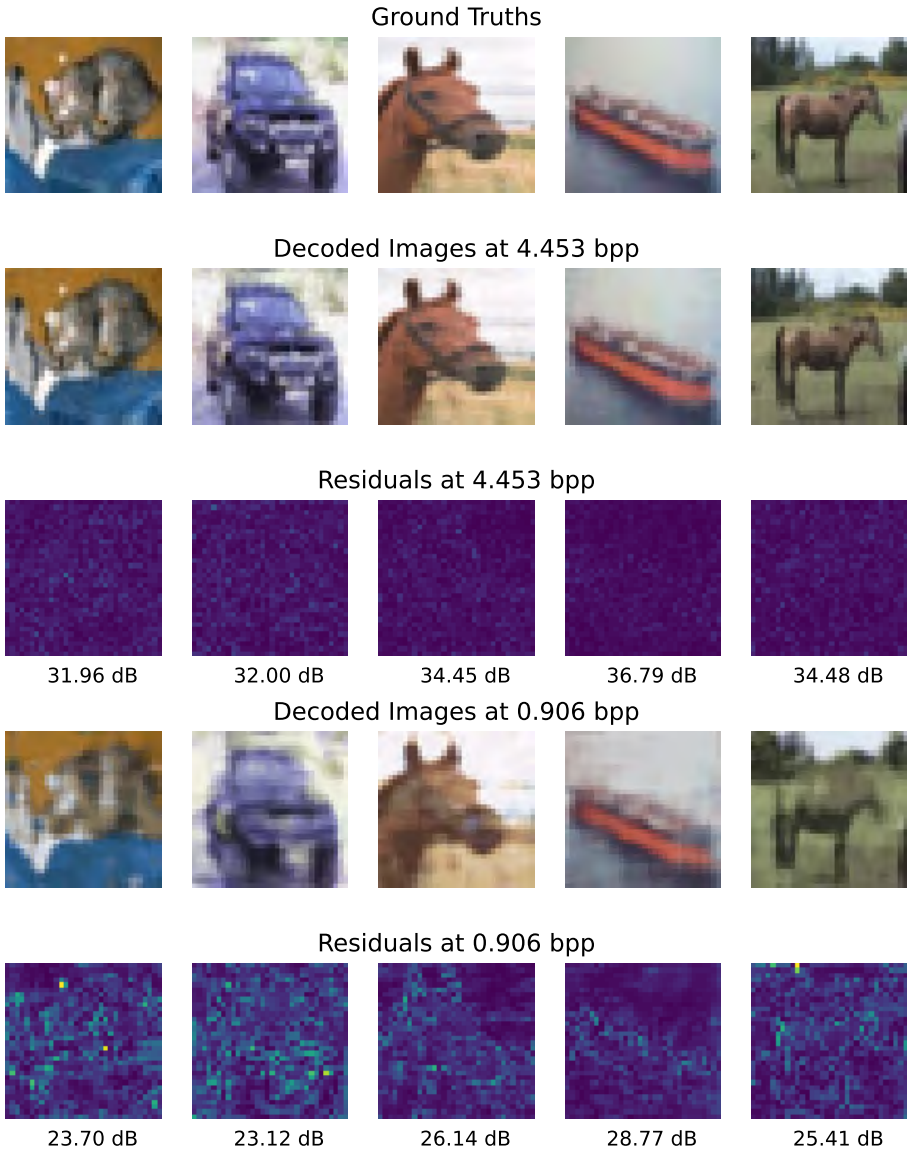


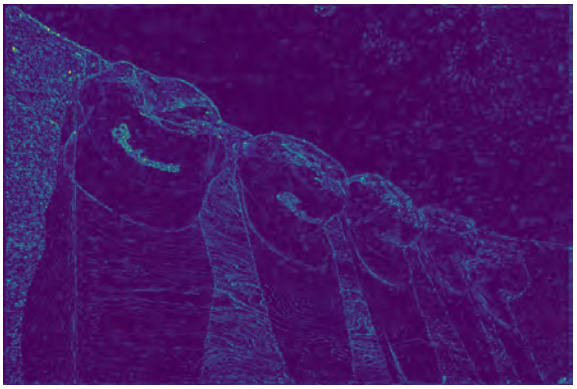
Fig. 3.3 Decoded Cifar-10 Images and their residuals at 4.453 bpp and 0.906 bpp. These 5 example images are randomly selected from the Cifar-10 test set.

(a) Ground Truth

(b) Decoded Image (PSNR 33.59 dB)

(c) Decoded Image (PSNR 29.81 dB)

(d) Residuals



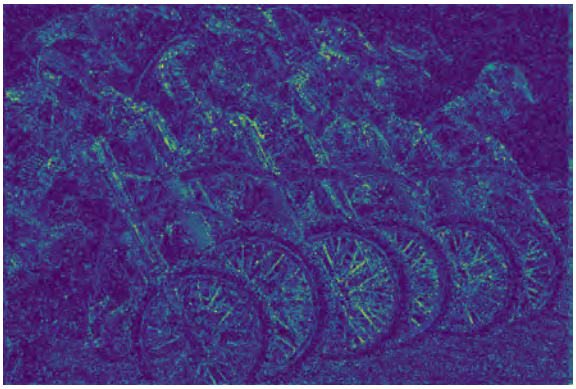
(e) Residuals

Fig. 3.4 Decoded Kodak-03 and its residuals at 0.293 bpp (left) and 0.070 bpp (right).

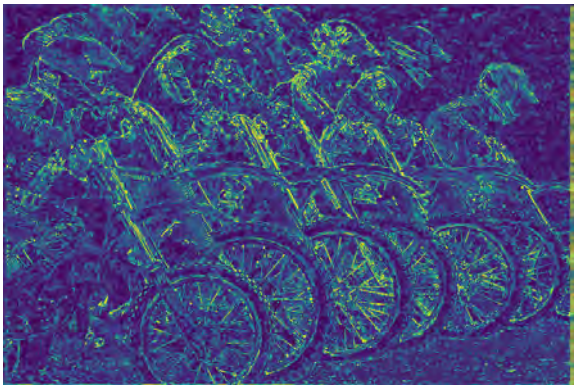
(a) Ground Truth

(b) Decoded Image (PSNR 23.54 dB)

(c) Decoded Image (PSNR 20.28 dB)



(d) Residuals



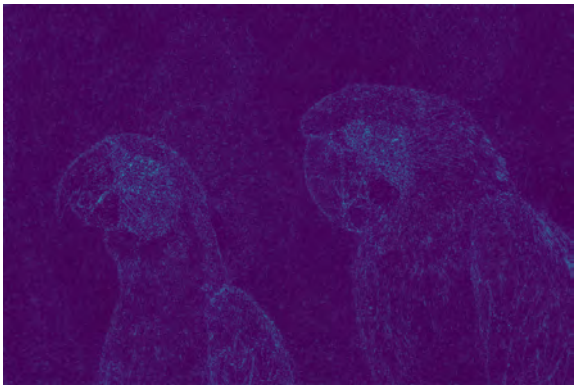
(e) Residuals

Fig. 3.5 Decoded Kodak-05 and its residuals at 0.293 bpp (left) and 0.070 bpp (right).

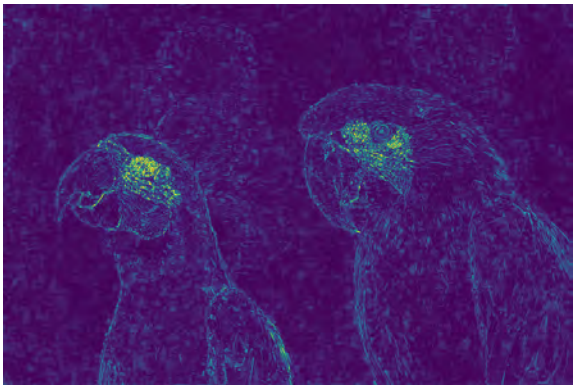
(a) Ground Truth

(b) Decoded Image (PSNR 33.97 dB)

(c) Decoded Image (PSNR 28.46 dB)



(d) Residuals



(e) Residuals

Fig. 3.6 Decoded Kodak-23 and its residuals at 0.293 bpp (left) and 0.070 bpp (right).

3.2.3 Analysis

Visualization of Adaptive Activation and Pruning

In this section, we visually illustrate how our model adaptively activates or prunes its hidden units to represent a specific signal, and how the priors are adjusted for different bit-rates, eliminating the necessity for manual tuning of the network size.

Specifically, we investigate a model trained on Cifar-10 dataset at 3.50 bpp. This MLP is built with 4 layers and 16 units in each hidden layer. Here we take the second layer for visualization. As shown in Fig 3.7, we visualize the model prior (both $\boldsymbol{\mu}_p$ and $\boldsymbol{\sigma}_p$) learned by Algorithm 1 in the left column. We display the variational posteriors of two distinct images (randomly selected from the Cifar-10 test set) in the second and third columns, and KL divergence $D_{\mathrm{KL}}[q_{\mathbf{w}}^{(1)}||p_{\mathbf{w}}]$, $D_{\mathrm{KL}}[q_{\mathbf{w}}^{(2)}||p_{\mathbf{w}}]$ in the rightmost column. Note that both the prior and the posteriors are fully-factorized, enabling us to visualize the KL at each position separately.

Interestingly, there are seven darker columns in $\boldsymbol{\sigma}_p$, indicating that only seven hidden units of this layer will be activated when fitting to a test datum at this specific bit-rate. For example, when representing image 1, four columns are activated, which is evidenced by the KL divergence. Similarly, only three columns are activated when representing image 2.

This observation clearly demonstrates the role of the variational posterior as model selection: it adaptively selects different subsets of units to activate, tailoring the representation for distinct test signals. Additionally, from the visualization of the prior, we can see the model prior learned by our proposed approach (Algorithm 1) is automatically adjusted according to a specific bit-rate. Certain columns (corresponding to certain units) in the prior exhibit extremely small standard deviations, effectively preventing these units from being activated by any test datum at this particular bit-rate. As a result, the network with such a prior is equivalent to a smaller network, but does not require manual tuning of the network size.

Effectiveness of Fine-tuning

When presenting our methods, we motivate the fine-tuning as an approach to not only rectify A* coding errors, but also attain a better posterior approximation from a theoretical perspective. Here, we provide experimental evidence, using Cifar-10 dataset at 3.50 bpp as an example.

Fine-tuning rectifying A coding errors:* first, we demonstrate how fine-tuning prevents the accumulation of A* coding errors. We randomly select 500 Cifar images as examples, encode them at 3.50 bpp, and visualize their average PSNR during REC in Fig 3.8. As depicted in the graph, there is a slight decline in PSNR after encoding each block. However,

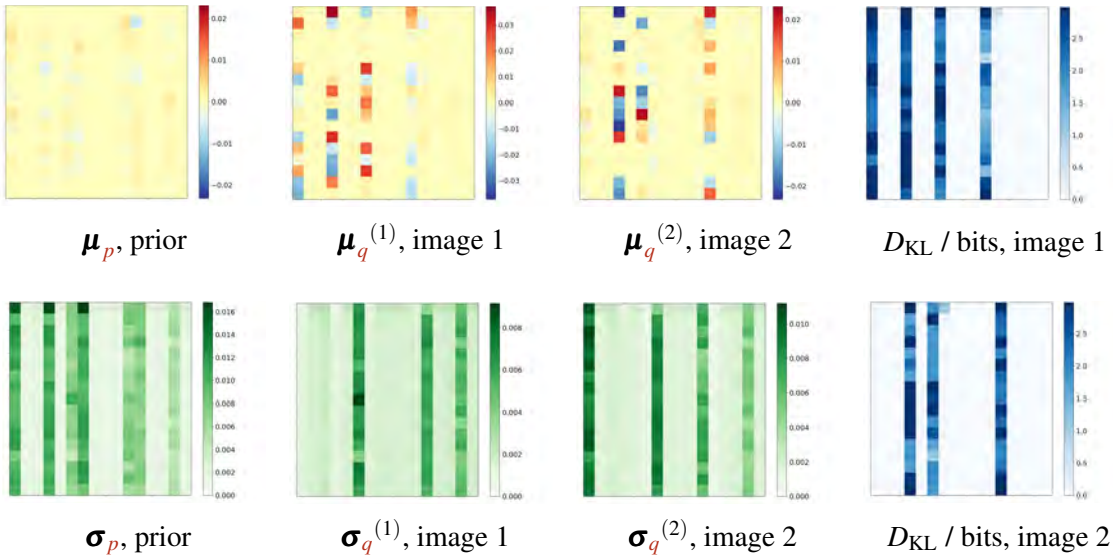


Fig. 3.7 Visualizations of the prior, and the posteriors for two test images. We focus on the 2nd layer, with 16 hidden units. Each column represents parameters for one hidden unit. The parameter matrix has a shape of 17×16 (the first 16 rows is the weight matrix, and the last is the bias vector).

the subsequent fine-tuning effectively compensates for this decrease. We also plot the results without fine-tuning for comparison. We can see the performance consistently degrades due to the error introduced by REC.

Fine-tuning attaining a better posterior approximation: in the following, we demonstrate how fine-tuning can yield a better posterior approximation qualitatively. We randomly select 1 Cifar image and encode it at 3.50 bpp as an example.

We first run Algorithm 3 to learn the posterior distribution $q_{\mathbf{w}}$. Then, starting from the *same* $q_{\mathbf{w}}$, we perform REC with fine-tuning (Algorithm 5) 500 times with different random seeds, and collect samples from each run. For comparison, we also perform another 500 runs, starting from the *same* $q_{\mathbf{w}}$, without fine-tuning. Visualizing the density of these samples offers insights into how the fine-tuning yields a more complicated posterior.

We curate two dimensions in $\mathbf{w}$ and visualize their prior, the posterior, and samples in Fig 3.9. The results clearly show that, through fine-tuning, it is possible to retrieve the correlation between parameters to a certain extent and obtain a more complex posterior than the mean-field Gaussian approximation $q_{\mathbf{w}}$.

We further provide an ablation study on Cifar-10 to quantitively demonstrate the effectiveness of fine-tuning. As depicted in Fig 3.10, fine-tuning provides an overall improvement of 2-4 dB, with greater gain observed at higher bit-rates.

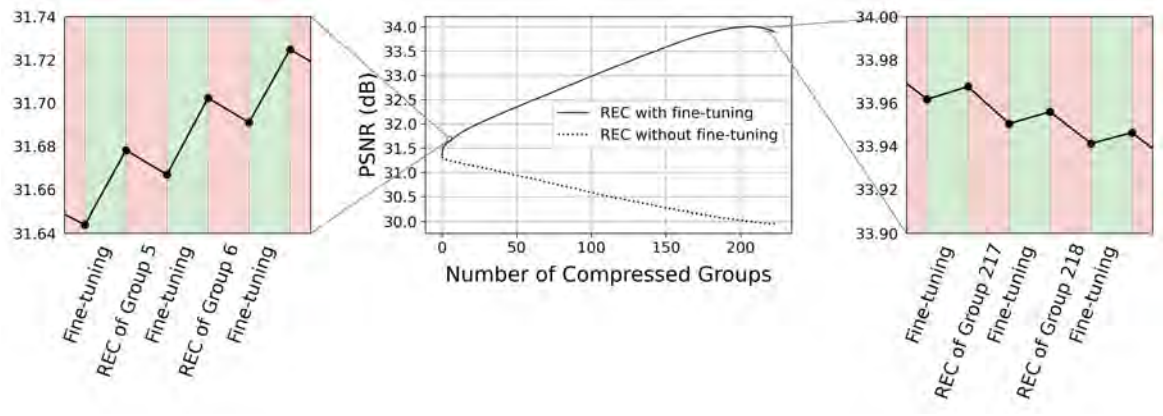


Fig. 3.8 Average PSNR when performing REC with fine-tuning. Two zoom-in plots provide a closer look at what happens after the REC of each block, and after subsequent fine-tuning.

Ablation Study of Model Prior

We also conduct an ablation study to verify the effectiveness of the model prior on Cifar-10. To be more specific, instead of learning model priors, we learn and transmit one univariate Gaussian prior for each layer, as proposed by Havasi et al. (2019). As shown in Fig 3.10, learning model priors increase the performance consistently by 4 dB.

Besides, we investigate the influence of the training size in the prior learning stage. We train model priors with different numbers of training data, and evaluate their R-D curve on 500 randomly selected Cifar-10 test images, as shown in Fig 3.11. Surprisingly, the performance levels off when the training size is larger than 16, indicating our approach learns a good prior even with merely 16 training data. This demonstrates the strong generalizability of the model prior, and the robustness of the prior learning process.

3.3 Summary

In this chapter, we propose COMBINER, a new neural compression framework representing data as variational INRs, and then encoding an approximate posterior sample by REC. Unlike previous INR-based codecs, it naturally supports joint rate-distortion optimization and is able to adaptively activate and prune the network units according to certain bit-rates and certain data signal. Within the confines of this framework, we propose an iterative algorithm for learning prior, and introduce fine-tuning with REC, substantially enhancing the performance.

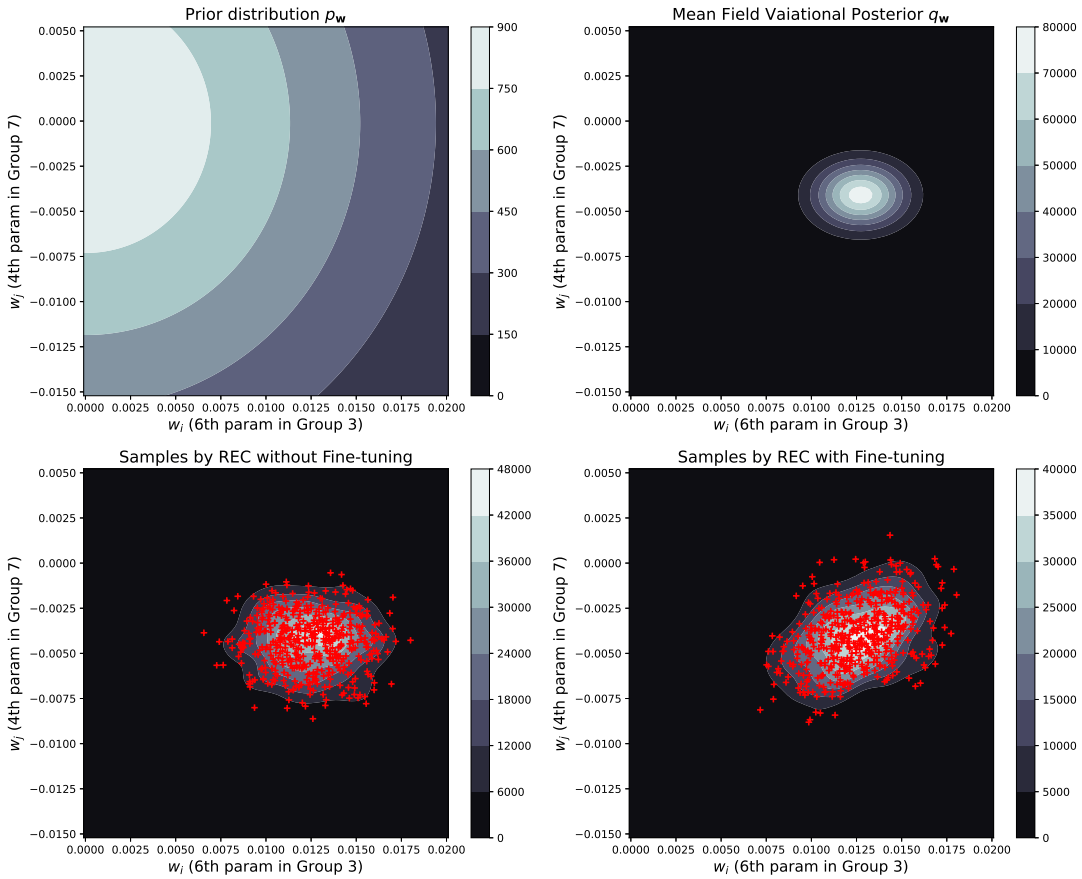


Fig. 3.9 Illustration of how REC with fine-tuning yields a more complicated posterior. w_i is the 6th parameter in block 3; w_j is the 4th parameter in block 7. The top left plot shows the prior $p_{\mathbf{w}}$; the top right plot shows the variational posterior $q_{\mathbf{w}}$; the bottom left plot shows the samples by REC without fine-tuning; the bottom right plot shows the samples by REC with fine-tuning. We can see the correlation between w_i and w_j is retrieved after fine-tuning.

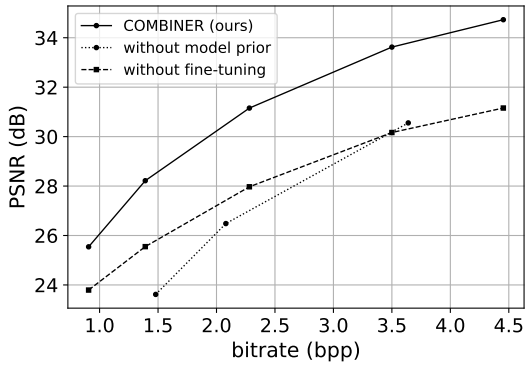


Fig. 3.10 Ablation Study of fine-tuning and the model prior.

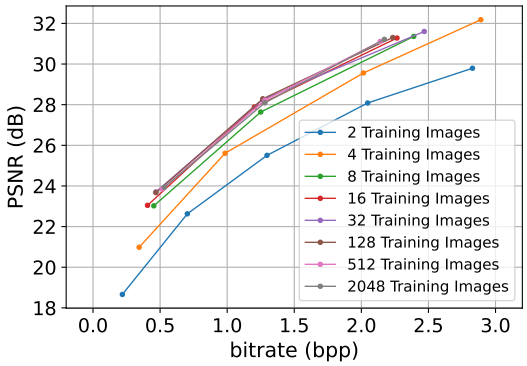


Fig. 3.11 R-D performance with different training sizes.

Chapter 4

COMBINER+: Improving COMBINER with Linear Transformation and Learnable Positional Encoding

In the previous chapter, we introduced COMBINER, a framework for data compression based on Bayesian Implicit Neural Representations. While the performance does not reach the state-of-the-art level, its simplicity in principle allows ample room for further improvements.

In this chapter, we present advancements in enhancing the COMBINER framework, named as COMBINER+ , which remarkably outperforms its predecessor without loss of its simplicity. On Cifar-10 dataset, it achieves the state-of-the-art rate-distortion performance at low bit-rates, outperforming even VAE-based codecs. On Kodak images, although falling short of matching the performance of VAEs, it significantly narrows the gap.

This chapter is organized as follows. In Section 4.1, we describe the methods. Rather than introducing COMBINER+ from scratch, we focus on the modifications to the COMBINER pipeline. In Section 4.2, we present the experimental results, and analyze our approaches in depth. We also evaluate COMBINER+ on audio to illustrate its modality transferability. This chapter is ended with a summary in Section 4.3, while more discussion on the limitations will be presented in Chapter 5 at the end of this thesis.

4.1 Methods

Built on top of COMBINER, COMBINER+ shares the same pipeline as described in Section 3.1.4. Beyond this, it improves its performance through two key modifications:

1. it employs linear transformation on the network parameters to eliminate redundancy;

2. it incorporates learnable positional encoding, assisting the fitting of INRs.

We first describe these two approaches in Section 4.1.1 and Section 4.1.2 respectively, and discuss their practical implementation together in Section 4.1.3.

Besides, to increase the scalability of COMBINER+, we choose to use patches for high-resolution images. To mitigate the influence of patching, we take a simple yet efficient trick to allocate bits across patches adaptively, which is described in Section 4.1.4.

4.1.1 Linear Transformation on Network Parameters

To motivate this approach, we consider the big picture of transform coding. As we discussed in Section 2.1.2, when compressing data $\boldsymbol{x}$, we apply an analysis transformation f into a transformed space, where the transformed data $f(\boldsymbol{x})$ is ideally decorrelated. In COMBINER, this transformation is conducted implicitly by gradient descent, and the transformed data is simply the network parameters, i.e., $\boldsymbol{w} = f(\boldsymbol{x})$.

However, the parameters of a neural network are known to have inherent redundancy. Denil et al. (2013) showcased that a small subset of parameters is adequate to reconstruct the entire network; Chen et al. (2015) demonstrated that, by randomly sharing parameters in the network, the model size can be reduced drastically, with only minor impact on the accuracy. This indicates that our data representation in the transformed space (i.e, the space of network parameters) is not fully decorrelated.

Therefore, we adopt linear transformation on the network parameters. Specifically, for an L-layer INR, we introduce *latent* network parameters

$$\boldsymbol{h} = \left[\left(\boldsymbol{h}^{[1]}\right)^\top, \left(\boldsymbol{h}^{[2]}\right)^\top, ..., \left(\boldsymbol{h}^{[L]}\right)^\top \right]^\top, \tag{4.1}$$

and a set of linear transformations $\left\{\boldsymbol{A}^{[l]}\right\}_{l=1}^{L}$. The parameters of each INR layer is mapped from the *latent* parameters by the linear transformation corresponding to this layer, i.e.,

$$\boldsymbol{w}^{[l]} = \boldsymbol{A}^{[l]}\boldsymbol{h}^{[l]}. \tag{4.2}$$

$\boldsymbol{w}^{[l]}$ denotes the vector concatenating the weight and bias of the l-th layer, i.e., $\boldsymbol{w}^{[l]} = [\mathrm{Vec}(\boldsymbol{W}^{[l]})^\top, \boldsymbol{b}^{[l]\top}]^\top$. The dimensionality of $\boldsymbol{h}^{[l]}$ is set to be the same as $\boldsymbol{w}^{[l]}$ in our experiments. We find, empirically, applying a single linear transformation to the entire network provide no improvement comparing with the layer-wise transformations, indicating the redundancy between layers is less than that within each layer.

To simply notations, we hereafter denote

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{A}^{[1]} & & & \\ & \boldsymbol{A}^{[2]} & & \\ & & \ddots & \\ & & & \boldsymbol{A}^{[L]} \end{pmatrix}, \quad \boldsymbol{w} = \boldsymbol{A}\boldsymbol{h}. \tag{4.3}$$

In the following, we describe the learning of $\boldsymbol{h}$ and $\boldsymbol{A}$ respectively. We focus on a conceptual introduction here, and will provide more practical details in Section 4.1.3, after introducing the learnable positional encoding.

Learning $\boldsymbol{h}$ by Stochastic Variational Inference

The *latent* parameter vector $\mathbf{h}$ in COMBINER+ is treated in the same way as the network parameter vector $\mathbf{w}$ in COMBINER. Specifically, we put a prior $p_{\mathbf{h}}$ which is learned by coordinate ascent on the training set, and a posterior $q_{\mathbf{h}}$ which will be learned by stochastic variational inference for each test datum. Then we perform REC with fine-tuning to encode a sample $\boldsymbol{h} \sim q_{\mathbf{h}}$. Similar to COMBINER, the prior and the posterior are parameterized by mean field Gaussian.

Learning $\boldsymbol{A}$ with Prior by Coordinate Ascent

Before delving into the learning of $\boldsymbol{A}$, we first consider the role that it should fulfil in our setting. Ideally, INR fitted to each datum exhibits its unique redundancy pattern. For instance, an INR for a simple image is likely to present higher redundancy. Therefore, it appears to be the most effective to learn a different $\boldsymbol{A}$ for each datum. However, encoding $\boldsymbol{A}$ for each datum is not favored since it will require a large amount of bits.

Consequently, rather than learning and compressing an individual $\boldsymbol{A}$ for each datum, we seek to find an $\boldsymbol{A}$ that captures the *common redundancy* pattern across the whole dataset. By sharing it between the sender and receiver, we eliminate the need for additional bits.

To achieve this, we incorporate the learning of $\boldsymbol{A}$ into the prior learning stage. Similar to COMBINER (Algorithm 1), the prior learning process involves a 2-stage coordinate ascent, where the posteriors of the training data and the prior are updated iteratively. In COMBINER+, when optimizing the posteriors of the training data using stochastic variational inference, we also back-propagate the gradient through $\boldsymbol{A}$ and perform gradient descent on it.

Recall in COMBINER, we learn multiple priors for a range of bit-rates. Here, we also learn multiple matrices $\boldsymbol{A}$, each corresponding to one bit-rate, as INRs at different bit-rates can display diverse patterns of redundancy.

4.1.2 Learnable Positional Encoding

A major dilemma associated with INR-based codecs is that the network represent the entire signal *globally*. While this characteristic enables an efficient removal of global redundancy, it simultaneously poses challenges in the fitting of INR. For instance, a minor change in a single pixel may result in the changes of all network parameters.

There exist studies on signal representations using *hybrid* coordinate-based neural representations (hybrid CNRs) which incorporates learnable latent code, or positional features [Chen et al. (2021b); Jiang et al. (2020); Kim et al. (2022); Mehta et al. (2021)]. Inspired by these works, we introduce a learnable positional encoding into our compression framework. The learnable positional encoding can be viewed as an *explicit* representation, complementing the *implicit* neural representation. To further enhance compression, this positional encoding is mapped from a down-sized code in the latent space.

In the following, we focus on image as the example. Data compression in other modality follows the same principle, and we only need to match the positional encoding's dimensionality with the data's dimensionality.

First, we introduce a *latent* positional encoding tensor $\mathbf{Z} \in \mathbb{R}^{C_\mathbf{Z} \times W_\mathbf{Z} \times H_\mathbf{Z}}$ for each image, and a convolution-based up-sampling network $\phi_{\text{CNN}}(\cdot)$. When reconstructing an image $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$, we first map its $\boldsymbol{Z}$ through ϕ_{CNN}, resulting in a pixel-wise positional encoding:

$$\tilde{\boldsymbol{Z}} = \phi_{\text{CNN}}(\boldsymbol{Z}) \in \mathbb{R}^{D_0 \times W \times H}, \tag{4.4}$$

where W and H are the width and height of the image. Then, for each pixel at position i, we concatenate the corresponding positional encoding $\tilde{\boldsymbol{z}}_i \in \mathbb{R}^{D_0}$, with the Fourier embedding $\gamma(\boldsymbol{x}_i) \in \mathbb{R}^D$ of its coordinate $\boldsymbol{x}_i$, i.e.,

$$\tilde{\boldsymbol{x}}_i = \left[\tilde{\boldsymbol{z}}_i^\top, \gamma^\top(\boldsymbol{x}_i)\right]^\top \in \mathbb{R}^{D_0 + D}. \tag{4.5}$$

The Fourier embedding is defined in Equation (3.11)[1] in Section 3.2.1. Finally, we map $\tilde{\boldsymbol{x}}_i$ through the INR, resulting in the reconstructed pixel value $\hat{\boldsymbol{y}}_i$. Unless stated otherwise, we

[1]For easier reference, we restate the Fourier embedding here:

$$\gamma(\boldsymbol{x}) = \Big[\cos(a^{0/d}\pi x_1), \cos(a^{0/d}\pi x_2), ..., \cos(a^{d/d}\pi x_1), \cos(a^{d/d}\pi x_2), \\ \sin(a^{0/d}\pi x_1), \sin(a^{0/d}\pi x_2), ..., \sin(a^{d/d}\pi x_1), \sin(a^{d/d}\pi x_2)\Big]^\top,$$

where $d = \lfloor \frac{D}{4} \rfloor - 1$ and $a = 1024$.

set $D = D_0 = 16$, $C_{\mathbf{z}} = 128$, $W_{\mathbf{z}} = W/4$, $H_{\mathbf{z}} = H/4$ in our experiments. The up-sampling network architecture is presented in Fig C.1 in Appendix.

We treat the $\mathbf{Z}$ in the same way as $\mathbf{h}$, and learn the up-sampling network during the prior learning process in the same way as $\boldsymbol{A}$. To be more specific, denoting $\boldsymbol{z} = \mathrm{Vec}(\boldsymbol{Z}) \in \mathbb{R}^{C_{\mathbf{z}} \cdot W_{\mathbf{z}} \cdot H_{\mathbf{z}}}$, we learn prior $p_{\mathbf{z}}$ and $\phi_{\mathrm{CNN}}(\cdot)$ by coordinate ascent. Then $p_{\mathbf{z}}$ and $\phi_{\mathrm{CNN}}(\cdot)$ will be fixed for all test datum, and shared between the sender and receiver. For a test datum at a specific bit-rate, we learn a posterior $q_{\mathbf{z}}$ by stochastic variational inference, and compress a random sample $\boldsymbol{z} \sim q_{\mathbf{z}}$ by REC with fine-tuning. Same as $\mathbf{h}$, $p_{\mathbf{z}}$ and $q_{\mathbf{z}}$ are parameterized by mean field Gaussian.

Connection with VAEs

Essentially, the up-sampling network can be viewed as the decoder of a VAE. This leads to our method being positioned as an interpolation between VAE-based and INR-based codecs, albeit without an explicit encoder. Importantly, the balance between VAE and INR components can be adaptively learned without manual tuning. In Section 4.2.3 (Fig 4.10), we demonstrate that our method dynamically adjusts the emphasis between these two components, rather than consistently favoring one.

Therefore, our approach holds the potential to inherit advantages of VAEs, particularly in terms of better scalability. Furthermore, owing to the compact size of our up-sampling network in comparison to fully VAE-based techniques like [Ballé et al. (2018)], our method preserves INRs' fast decoding.

Moreover, unlike many VAE-based codecs elaborately designed for each specific data modality, our up-sampling network ϕ_{CNN}, composed solely of up-sampling layers, convolution layers, and LeakyReLU activations, can be adapted to various modalities by simply adjusting the dimensionality of the convolution kernels. We demonstrate its strong transferability by 1D audio data in Section 4.2.4.

4.1.3 Implementation in Practice

Since we treat $\mathbf{h}$ and $\mathbf{z}$ in the same way, we concatenate them to form a single vector $\mathbf{v}$ in practice, i.e.,

$$\boldsymbol{v} = [\boldsymbol{h}^\top, \boldsymbol{z}^\top]^\top. \tag{4.6}$$

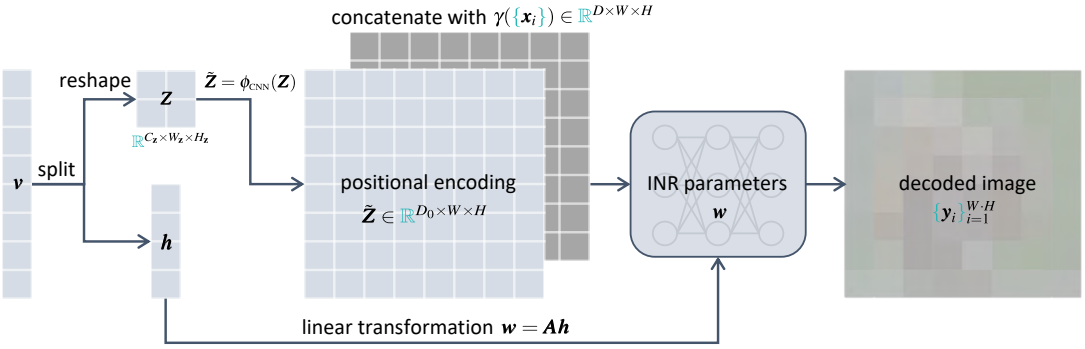


Fig. 4.1 Visualization of the reconstruction process of COMBINER+.

We then directly operate on this unified vector $\mathbf{v}$, with the prior $p_{\mathbf{v}}$ and the posterior $q_{\mathbf{v}}$, both parameterized by mean field Gaussian, i.e.,

$$p_{\mathbf{v}} = \mathcal{N}\left(\boldsymbol{\mu}_{p_{\mathbf{v}}}, \text{diag}\left(\boldsymbol{\sigma}^2_{p_{\mathbf{v}}}\right)\right), \quad q_{\mathbf{v}} = \mathcal{N}\left(\boldsymbol{\mu}_{q_{\mathbf{v}}}, \text{diag}\left(\boldsymbol{\sigma}^2_{q_{\mathbf{v}}}\right)\right). \tag{4.7}$$

When decoding an image, we simply split $\boldsymbol{v}$ into $\boldsymbol{h}$ and $\boldsymbol{z}$, and predict the pixel values, by the INR with parameters $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{h}$ and the up-sampled positional encoding $\tilde{\boldsymbol{Z}} = \phi_{\text{CNN}}(\boldsymbol{Z})$. This process is depicted in Figure 4.1.

In the following, to avoid ambiguity, we call $\mathbf{h}$ as the *latent* network parameter, $\mathbf{w}$ as the network parameter, $\mathbf{z}$ & $\mathbf{Z}$ as the *latent* positional encoding, $\tilde{\mathbf{z}}$ & $\tilde{\mathbf{Z}}$ as the positional encoding, $\mathbf{v}$ as the *latent* vector.

Having presented all the key components, we state the entire pipeline of COMBINER+ below. Similar to COMBINER, there are also 4 steps:

1. Given a training dataset $\{\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_M\}$, we select a β corresponding to the desired bit-rate, and learn $p_{\mathbf{v}}$, $\boldsymbol{A}$ and $\phi_{\text{CNN}}(\cdot)$ by coordinate ascent, where we iteratively alternate the following steps **(a)** and **(b)**:

(a) optimize posteriors, linear transformation and up-sampling network: fixing the prior $p_{\mathbf{v}}$, we minimize the average rate-distortion objective across all M training instances:

$$\mathcal{L}(\{q^{(m)}_{\mathbf{v}}\}, p_{\mathbf{v}}, \boldsymbol{A}, \phi_{\text{CNN}}, \{\mathcal{D}_m\}) \propto \sum_{m=1}^{M} \mathcal{L}(q^{(m)}_{\mathbf{v}}, p_{\mathbf{v}}, \boldsymbol{A}, \phi_{\text{CNN}}, \mathcal{D}_m) \tag{4.8}$$

$$= \sum_{m=1}^{M} \left\{ \mathbb{E}_{\boldsymbol{v}\sim q^{(m)}_{\mathbf{v}}} \left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_m} \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{v},\boldsymbol{A},\phi_{\text{CNN}}}) \right] + \beta \cdot D_{\text{KL}}[q^{(m)}_{\mathbf{v}} || p_{\mathbf{v}}] \right\}, \tag{4.9}$$

where $\hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{v},\boldsymbol{A},\phi_{\text{CNN}}}$ denotes the reconstructed pixel value at coordinate $\boldsymbol{x}$. The minimization is performed by gradient descent on $\boldsymbol{A}$, ϕ_{CNN} , and the variational parameters of training data $\{\boldsymbol{\mu}_{q_{\mathbf{v}}}^{(m)}, \boldsymbol{\sigma}_{q_{\mathbf{v}}}^{(m)}\}$, jointly.

(b) update the prior in closed-form: fixing $\{\boldsymbol{\mu}_{q_{\mathbf{v}}}^{(m)}, \boldsymbol{\sigma}_{q_{\mathbf{v}}}^{(m)}\}$, $\boldsymbol{A}$ and ϕ_{CNN}, we update prior by:

$$\boldsymbol{\mu}_{p_{\mathbf{v}}} = \frac{1}{M}\sum_{m=1}^{M} \boldsymbol{\mu}_{q_{\mathbf{v}}}^{(m)}, \quad \boldsymbol{\sigma}_{p_{\mathbf{v}}} = \sqrt{\frac{1}{M}\sum_{m=1}^{M}\left[\left(\boldsymbol{\sigma}_{q_{\mathbf{v}}}^{(m)}\right)^2 + \left(\boldsymbol{\mu}_{q_{\mathbf{v}}}^{(m)} - \boldsymbol{\mu}_{p_{\mathbf{v}}}\right)^2\right]} \tag{4.10}$$

The derivation of Equation (4.10) is the same as Equation (3.8) in COMBINER.

2. Given a coding budget κ, we partition the parameter vector $\mathbf{v}$ into K blocks $\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_K$ by the next-fit bin-packing algorithm. Different priors learned with different β will end up with different partitions.

3. For a new test datum $\mathcal{D}$, according to the desired rate-distortion trade-off, we select a prior $p_{\mathbf{v}}$, the matrix $\boldsymbol{A}$, the up-sampling network ϕ_{CNN}, and the corresponding partition. Fixing $\boldsymbol{A}$ and ϕ_{CNN}, we learn the variational posterior $q_{\mathbf{v}_k}$ for each block by minimizing:

$$\mathcal{L}(\{q_{\mathbf{v}_k}\}_{k=1}^K, \{p_{\mathbf{v}_k}\}_{k=1}^K, \mathcal{D}) = \mathbb{E}_{q_{\mathbf{v}_1}...q_{\mathbf{v}_K}}\left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}} \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}})\right] + \sum_{k=1}^{K} \beta_k \cdot D_{\text{KL}}[q_{\mathbf{v}_k} || p_{\mathbf{v}_k}]. \tag{4.11}$$

Here we omit the dependency between $\hat{\boldsymbol{y}}$ and $\boldsymbol{x}, \boldsymbol{v}_1, ..., \boldsymbol{v}_K, \boldsymbol{A}, \phi_{\text{CNN}}$ for simplicity. Besides, to ensure the coding budget κ, we run Algorithm 4, i.e.,

$$\{\beta_k\} \leftarrow \text{AdjustBeta}(\{\beta_k\}, \{q_{\mathbf{v}_k}\}, \{p_{\mathbf{v}_k}\}) \tag{4.12}$$

every I iterations to adjust β_k, so that the KL divergence for each block is roughly κ.

4. Finally, we encode a single random parameter sample $\boldsymbol{v}_k \sim q_{\mathbf{v}_k}$ for each block by A* Coding, and perform fine-tuning after the compression of each block.

One subtlety of this approach is that, by the linear transformation $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{h}$, the network parameter $\boldsymbol{w}$ is not fully-factorized, and thus local reparametrization trick [Kingma et al. (2015)] is not compatible. To mitigate the influence of the consequent higher variance, when performing stochastic variational inference on test datum by Equation (4.11), we use 5 samples to estimate the expectation[2]. We present experimental results showing the influence of sample size in Appendix D.2.

[2]While during the prior learning stage, we still use 1 sample to optimize Equation (4.9).

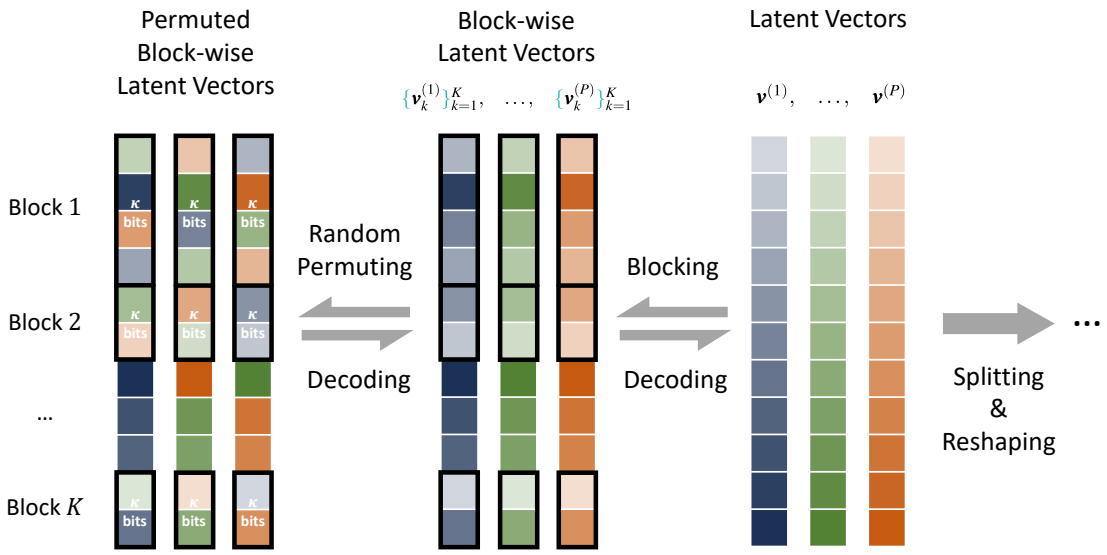


Fig. 4.2 Illustration of the random permutation across patches.

4.1.4 Towards Scalability: Compression with Patches

While introducing the linear transformation eliminates the redundancy in INRs, it limits the scalability since the number of parameters in $\boldsymbol{A}$ grows *quartically* w.r.t. the number of hidden units in each layer. Therefore, when applying COMBINER+ to high-resolution images, we divide the image into non-overlapped 64×64 patches, and encode each patch separately.

Allocating Bits across Patches by Random Permutation

However, in our scenario, patching significantly impairs performance. Apart from the unavoidable spatial redundancy introduced by patching, another factor is that the number of bits allocated to each patch only depends on the number of blocks, and thus remains constant for all patches at a specific bit-rate. Consequently, when compressing an uneven image, we may waste too many bits on simple patches, leaving insufficient bits for more intricate patches.

To address this, we apply random permutations across the *latent* vectors of all patches in one image. Concretely, considering P patches and their corresponding latent vectors $\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, ..., \boldsymbol{v}^{(P)}$, for each dimension d within the latent vector, we apply a random permutation on $v_d^{(1)}, v_d^{(2)}, ..., v_d^{(P)}$. This permutation is governed by a shared seed with the receiver, and different seeds are used for different dimensions. We illustrate the random permutation in Fig 4.2.

As a result of this permutation, each block comprises parameters from different patches. When learning the posteriors and enforcing the KL budget κ to each block, parameters corresponding to more complex patches can dynamically acquire more bits. Experimentally, we find the random permutation can bring around 1 dB gain on Kodak images.

Up-sampling on Entire Image

Additionally, unlike the linear transformation, the convolution-based up-sampling network $\phi_{\text{CNN}}(\cdot)$ is not subject to the size of input. Therefore, instead of passing the latent positional encoding of each patch separately, we first assemble them together, forming one latent positional encoding for the entire image, and pass it through $\phi_{\text{CNN}}(\cdot)$ as a whole. After this, we re-split the up-sampled positional encoding, according to the previous patches' position, and pass them through individual INRs. However, we find up-sampling on each individual patch also works well, with a negligible performance degradation of less than 0.05 dB.

4.2 Experiments and Results

In this section, we evaluate the performance of COMBINER+ on Cifar-10 and Kodak dataset. We also provide comprehensive analysis of linear transformation, positional encoding and random permutation. We close this section with ablation studies, providing quantitative evidence of our methods.

4.2.1 Experiment Settings

On both of the Cifar-10 and Kodak datasets, in the prior training period, we initialize elements in $\boldsymbol{A}$ by:

$$A \sim \mathcal{U}(-0.001, 0.001). \tag{4.13}$$

We find initializing $\boldsymbol{A}$ to small values can accelerate the convergence. Besides, we initialize the latent positional encodings by $Z \sim \mathcal{N}(0, 0.1)$ and the latent network parameters $\mathbf{h}$ by SIREN initialization [Sitzmann et al. (2020)], i.e.,

$$h^{[l]} \sim \begin{cases} \mathcal{U}\left(-1/d_l, 1/d_l\right), & \text{if } l = 1; \\ \mathcal{U}\left(-\sqrt{6/d_l}, \sqrt{6/d_l}\right), & \text{otherwise,} \end{cases} \tag{4.14}$$

where d_l is the input dimension for the layer l. It is worth noting that the initialization of $\boldsymbol{A}$ and $\mathbf{h}$ is crucial for a good performance in our case.

Below, we present the experiment settings specific to each dataset.

Cifar-10: We randomly select 15,000 training images from the training set, and evaluate our model on all 10,000 test images. The INR has 4 layers, and each hidden layer has 32 hidden units. The size of linear transformation for each layer is $\boldsymbol{A}^{[1]}, \boldsymbol{A}^{[2]}, \boldsymbol{A}^{[3]} \in \mathbb{R}^{1056\times1056}$, and $\boldsymbol{A}^{[4]} \in \mathbb{R}^{99\times99}$. The shape of latent positional encoding for each image is $\mathbf{Z} \in \mathbb{R}^{128\times2\times2}$.

Kodak: We randomly crop 8,000 patches from Div2k dataset [Agustsson and Timofte (2017)] as the training set, and evaluate our approach on all the 24 Kodak images. For each Kodak image, we first rotate it to 512×768, and then divide it into 96 patches, each of size 64×64. We use the same INR architecture as Cifar for each patch, and the latent positional encoding has a size of $\mathbf{Z} \in \mathbb{R}^{128\times4\times4}$.

For additional experimental details, please refer to Table C.2 in Appendix.

4.2.2 Compression Performance

Rate-Distortion Curve

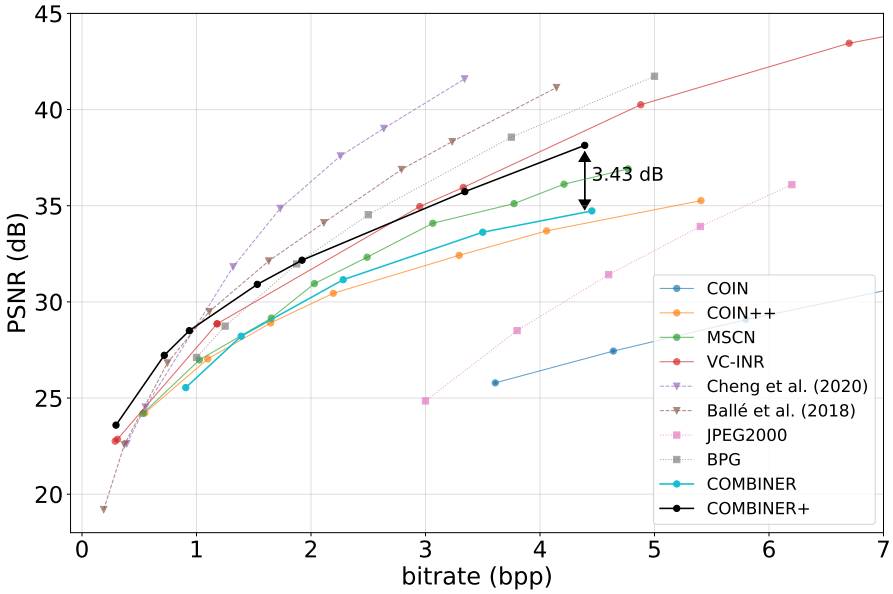
We show the R-D curves of COMBINER+ in Fig 4.3. We can see COMBINER+ exhibits a significant gain compared with COMBINER. Moreover, on Cifar-10, the method displays remarkable performance, especially at low bit-rates, surpassing even VAE-based codecs. On Kodak, it also exhibits clear superiority over JPEG2000. While not reaching the level of VAE-based codecs, it significantly reduces the gap.

We also visualize some sample images as a qualitative evaluation of our method. We show 5 images from Cifar-10 test set at 0.297 bpp and 4.391 bpp in Fig 4.4, and Kodak-03, Kodak-05 and Kodak-23 at 0.078 bpp and 0.484 bpp in Fig 4.5, 4.5, 4.7.

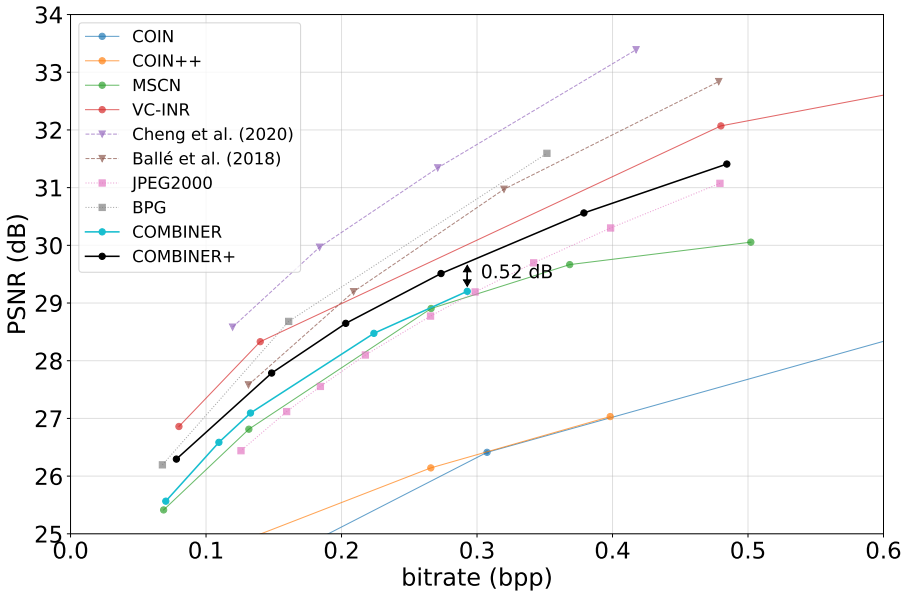
Encoding and Decoding Speed

We provide the encoding and decoding time of COMBINER+ on Cifar-10 and Kodak in Table D.3 and Table D.4 in Appendix, following the same setup as described for COMBINER in Section 3.2.2.

As we have not specifically focused on reducing time complexity, COMBINER+ also features a long encoding time. Using 5 samples to optimize Equation (4.11) also contributes to the encoding time. However, the decoding time is still remarkably fast due to the compact nature of the INR, the linear transformation and the up-sampling network.



(a) Cifar-10



(b) Kodak

Fig. 4.3 The Rate-Distortion Curve of COMBINER+ on Cifar-10 and Kodak datasets. The arrows illustrate the gains in comparison to COMBINER. We use solid lines to denote INR-based methods, dotted lines for VAE-based methods, and dashed lines for classical methods.

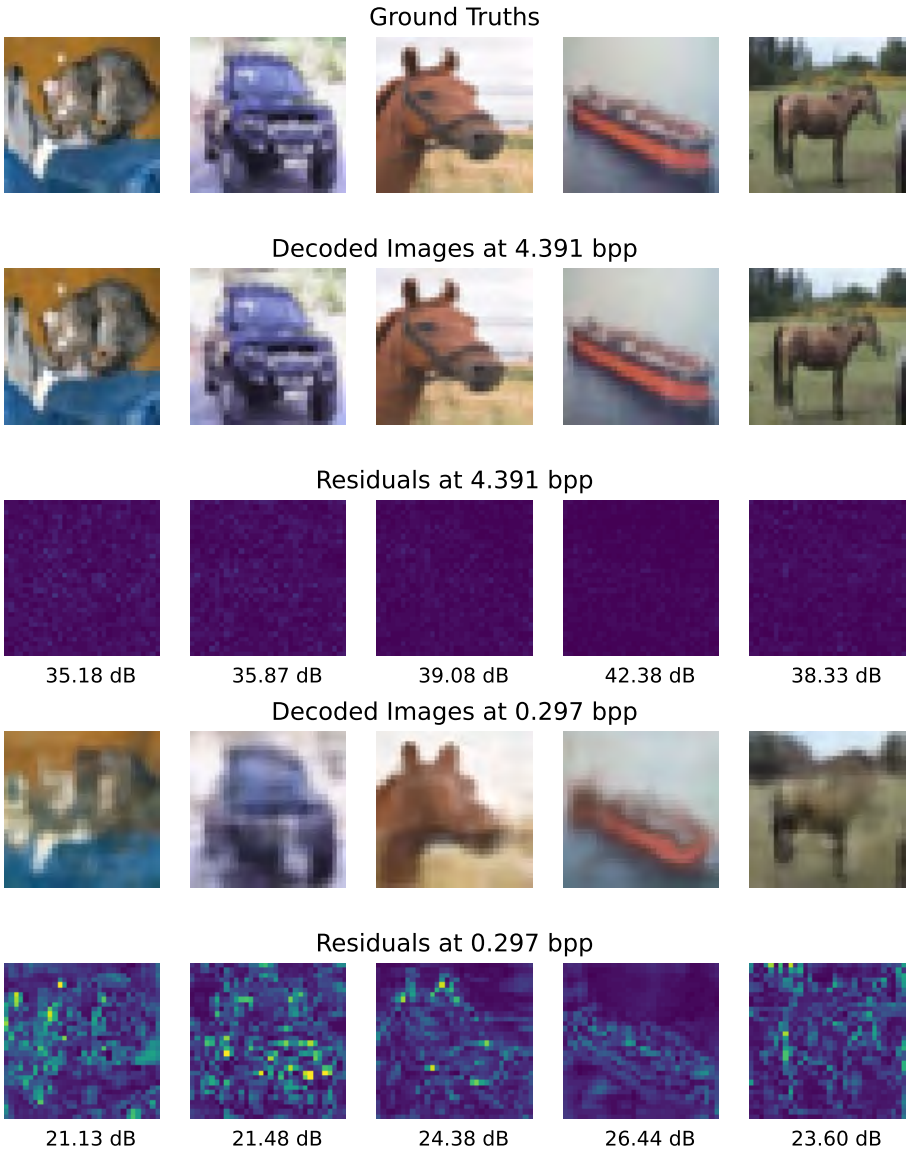
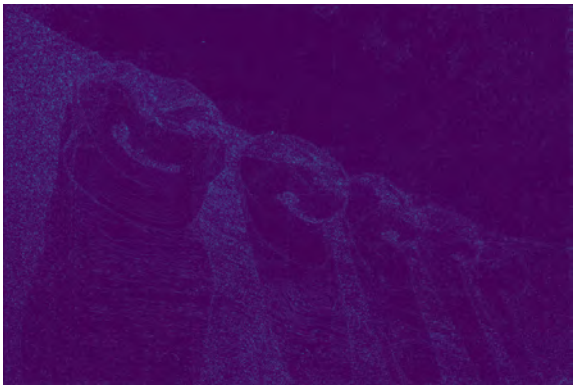


Fig. 4.4 Decoded Cifar-10 Images and their residuals at 4.391 bpp and 0.297 bpp. These 5 example images are randomly selected from the Cifar-10 test set.

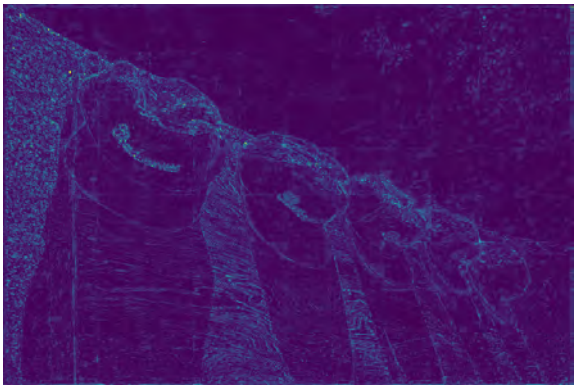
(a) Ground Truth

(b) Decoded Image (PSNR 35.89 dB)

(c) Decoded Image (PSNR 30.39 dB)



(d) Residuals



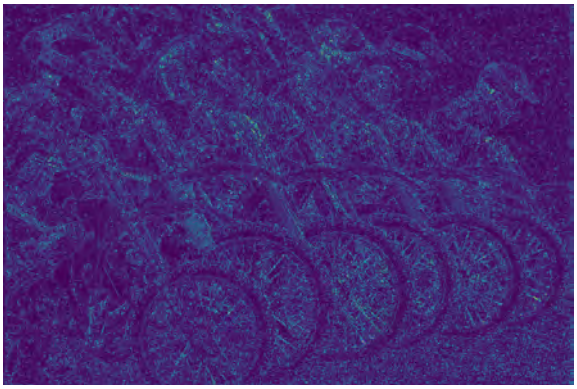
(e) Residuals

Fig. 4.5 Decoded Kodak-03 and its residuals at 0.484 bpp (left) and 0.078 bpp (right).

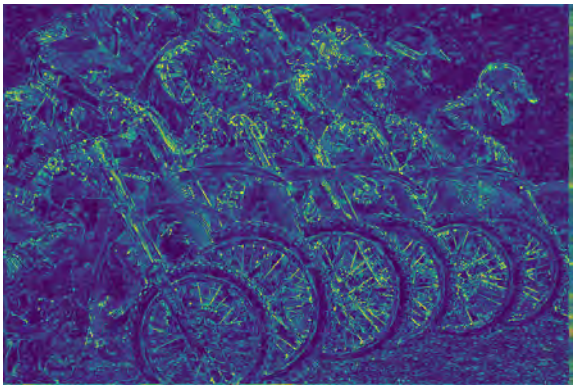
(a) Ground Truth

(b) Decoded Image (PSNR 26.10 dB)

(c) Decoded Image (PSNR 21.28 dB)



(d) Residuals



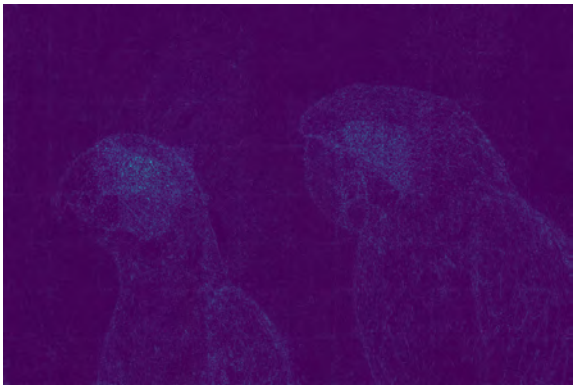
(e) Residuals

Fig. 4.6 Decoded Kodak-05 and its residuals at 0.484 bpp (left) and 0.078 bpp (right).

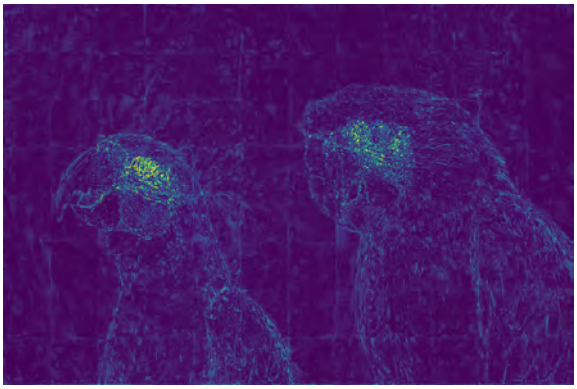
(a) Ground Truth

(b) Decoded Image (PSNR 36.16 dB)

(c) Decoded Image (PSNR 29.48 dB)



(d) Residuals



(e) Residuals

Fig. 4.7 Decoded Kodak-23 and its residuals at 0.484 bpp (left) and 0.078 bpp (right).

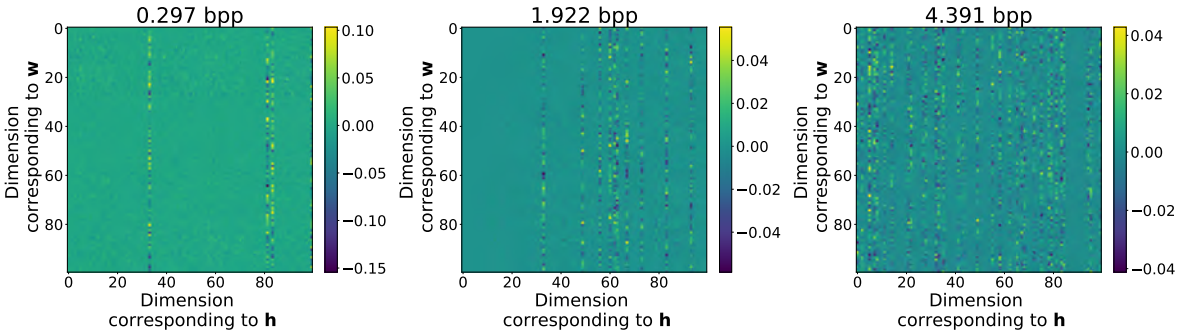


Fig. 4.8 Visualization of $\boldsymbol{A}$ at different bit-rates. Recall $\boldsymbol{w} = \boldsymbol{A}\boldsymbol{h}$, thus each column corresponding to each dimension of $\mathbf{h}$, and rows corresponding to $\mathbf{w}$. For a clearer visualization, this plot only shows 100 randomly selected dimensions from $\boldsymbol{A}^{[2]}$.

4.2.3 Analysis

Interpretation of Linear Transformation $\boldsymbol{A}$

Here, we visualize the linear transformation $\boldsymbol{A}$ at different bit-rates, which brings us insight of how our method controls the bit-rate while representing the signal effectively.

We take Cifar-10 dataset as an example, and visualize the linear transformation for the 2nd layer, i.e., $\boldsymbol{A}^{[2]}$, at 3 different bit-rates in Fig 4.8. When bit-rate is high, most elements in $\boldsymbol{A}$ are active, enabling a flexible model. Conversely, at lower bit-rates, many columns in $\boldsymbol{A}$ become zero, effectively pruning out corresponding dimensions in $\mathbf{h}$. This clarifies the reason for initializing elements in $\boldsymbol{A}$ to be small, which accelerates the pruning of unnecessary columns in $\boldsymbol{A}$.

Moreover, this visualization indicates how $\boldsymbol{A}$ contributes to improve the performance: first, it is evident that $\boldsymbol{A}$ greatly promotes parameter sharing. For instance, at low bit-rates, merely 5 percent of the parameters get involved in constructing the entire network. Second, the pruning in $\mathbf{h}$ is more efficient than that in $\mathbf{w}$. Recall how COMBINER controls its bit-rate in Fig 3.7. Without the linear transformation, COMBINER prunes or activates the *hidden units* instead of the network *parameters*. When a unit is pruned, the entire column in the weight matrix will be pruned out. In other words, the pruning in $\mathbf{w}$ is always conducted *in chunks*, which highly limits the flexibility of the network. On the contrary, the linear transformation $\boldsymbol{A}$ enables COMBINER+ to directly prune or activate each parameter in $\mathbf{h}$ *individually*, ensuring the flexibility of INR while effectively managing the rate.

Interpretation of Positional Encoding

To understand how positional encoding functions, we first visualize the positional encoding $\tilde{\boldsymbol{Z}}$, using Kodak-23 as an example, in Fig 4.9. We can see that before fed to the INR, $\tilde{\boldsymbol{Z}}$ already contain the spatial information of the image, therefore significantly simplifying the fitting of INRs. On the contrary, in the absence of positional encoding, INRs are required to learn complex mappings from the coordinates to the signal values, which is especially challenging when bit-rate is constrained. This insight also aligns with the ablation study (Fig 4.12), showing that positional encoding exhibits more benefits at low bit-rates and for larger images.

Another question is how much our model relies on the positional encoding. To answer this, we illustrate the percentage of bits allocated to the *latent* network parameter $\mathbf{h}$ and the *latent* positional encoding $\mathbf{z}$ on the Kodak dataset at different bit-rates in Fig 4.10. We use KL as a proxy for the actual coding bits. More precisely, we compute the average KL divergence $\delta_{\mathbf{h}} = D_{\mathrm{KL}}[q_{\mathbf{h}}||p_{\mathbf{h}}]$ and $\delta_{\mathbf{z}} = D_{\mathrm{KL}}[q_{\mathbf{z}}||p_{\mathbf{z}}]$ across all test data points, and depict $\frac{\delta_{\mathbf{h}}}{\delta_{\mathbf{h}}+\delta_{\mathbf{z}}}\%$ and $\frac{\delta_{\mathbf{z}}}{\delta_{\mathbf{h}}+\delta_{\mathbf{z}}}\%$ at varying bit-rates. We can see our method dynamically adjusts the emphasis between these two components based on the bit-rate, rather than favoring one consistently. When the INR capacity is constrained, the model relies more on the positional encoding. While INR takes the dominance at higher bit-rates.

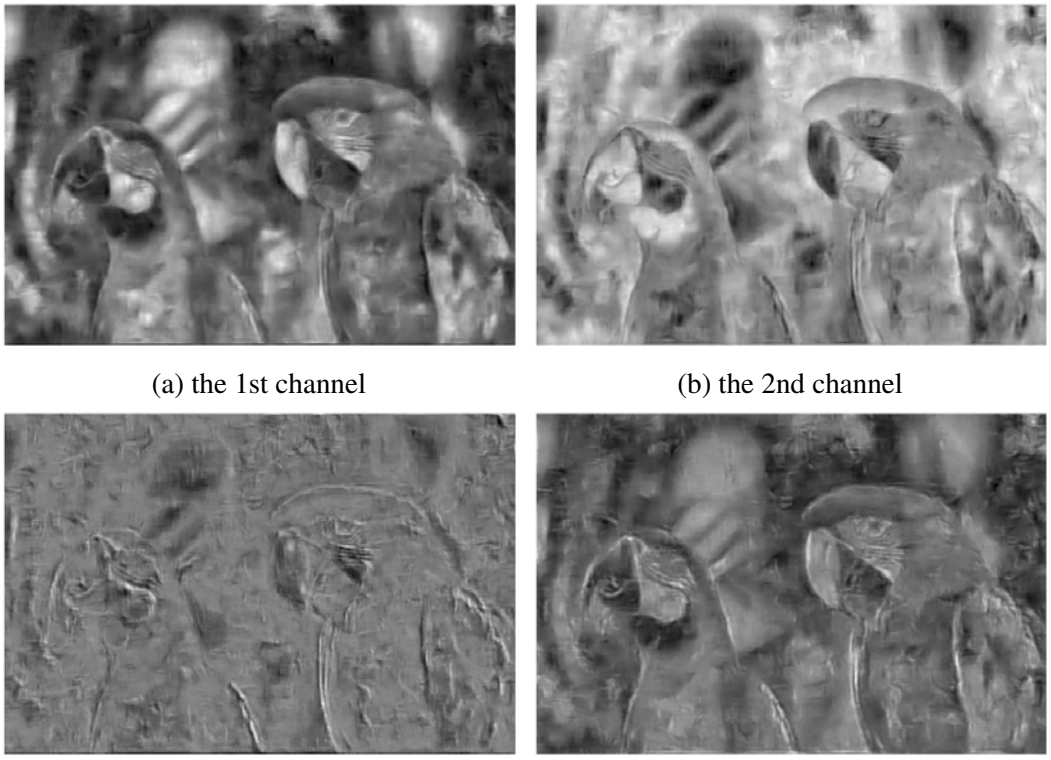
Influence of Random Permutation

To intuitively illustrate the effectiveness of random permutation across patches, we compare the residual of decoded Kodak-23 with and without permutation in Fig 4.13. We can see the random permutation results in a more balanced distribution of residuals across patches: complex patches achieve improved reconstructions, whereas simpler patches exhibit only marginal performance degradation. This observation implies that, through random permutation, the allocation of bits to each patch occurs in an adaptive manner. Overall, random permutation yields a 1.28 dB gain on this image.

Ablation Study

We conduct ablation studies to quantitively show the effectiveness of linear transformation, learnable positional encoding, and the random permutation across patches. The results on Cifar-10 and Kodak are depicted in Fig 4.12.

The ablation study reveals that the linear transformation yields substantial improvements on both datasets, particularly at higher bit-rates. For Cifar-10, the improvement can be up to 4dB, and on Kodak, it also delivers over 0.5 dB gain consistently.



(a) the 1st channel (b) the 2nd channel

(c) the 3rd channel (d) the 4th channel

Fig. 4.9 Different channels of the positional encoding $\tilde{\boldsymbol{Z}} = \phi_{\text{CNN}}(\boldsymbol{Z})$ for Kodak-23. Positional encodings of patches are stitched together to form a feature map with the same width and height as the original image.

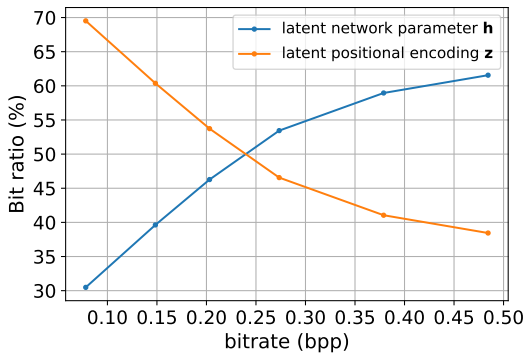


Fig. 4.10 Percentage of bits (estimated by KL) for the *latent* network parameter **h** and the *latent* positional encoding **z** on the Kodak dataset at various bit-rates.

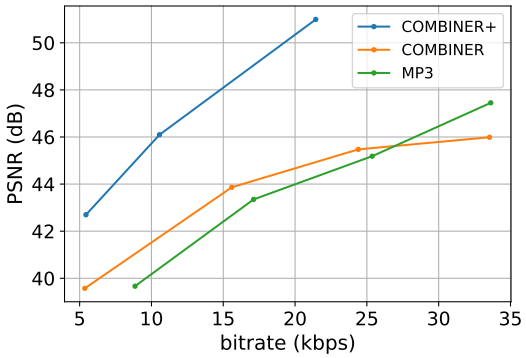
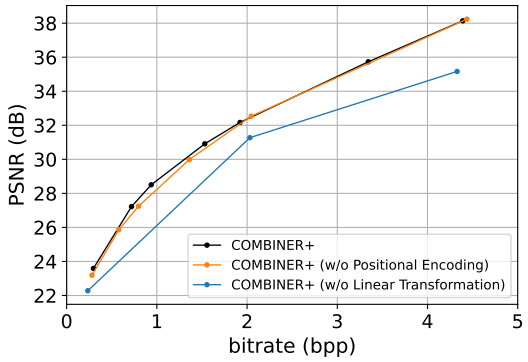
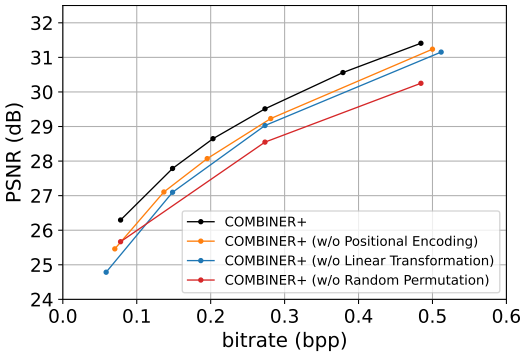


Fig. 4.11 Rate-distortion curve of COMBINER+, COMBINER and MP3 on audio compression. Kbps stands for kilobits per second.



(a) on Cifar-10

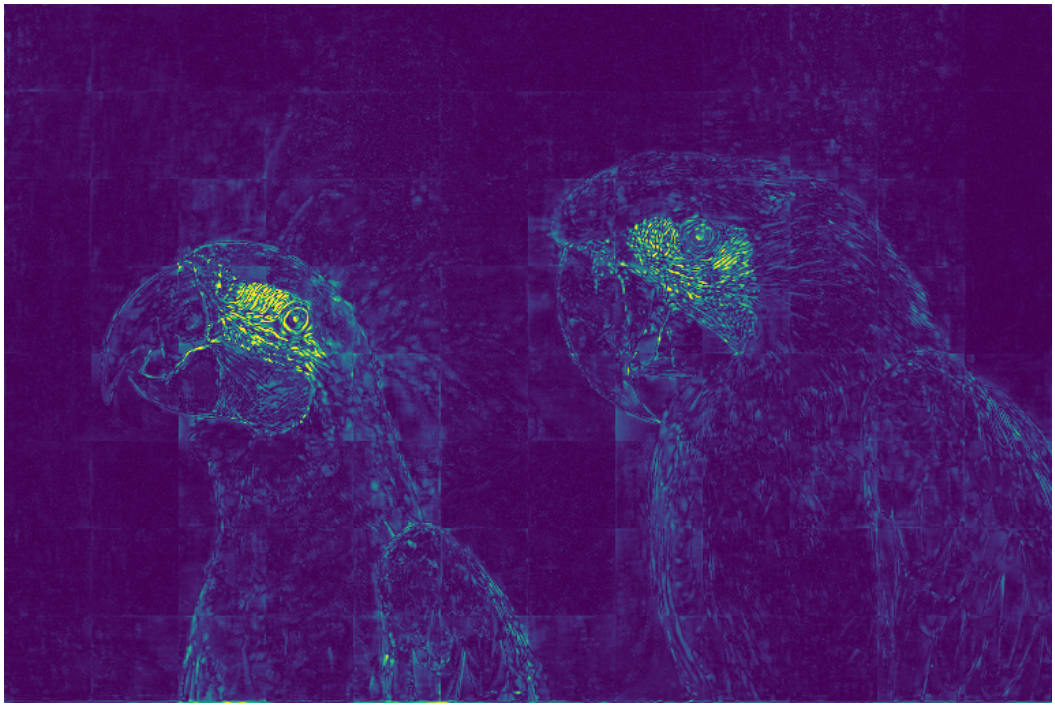


(b) on Kodak

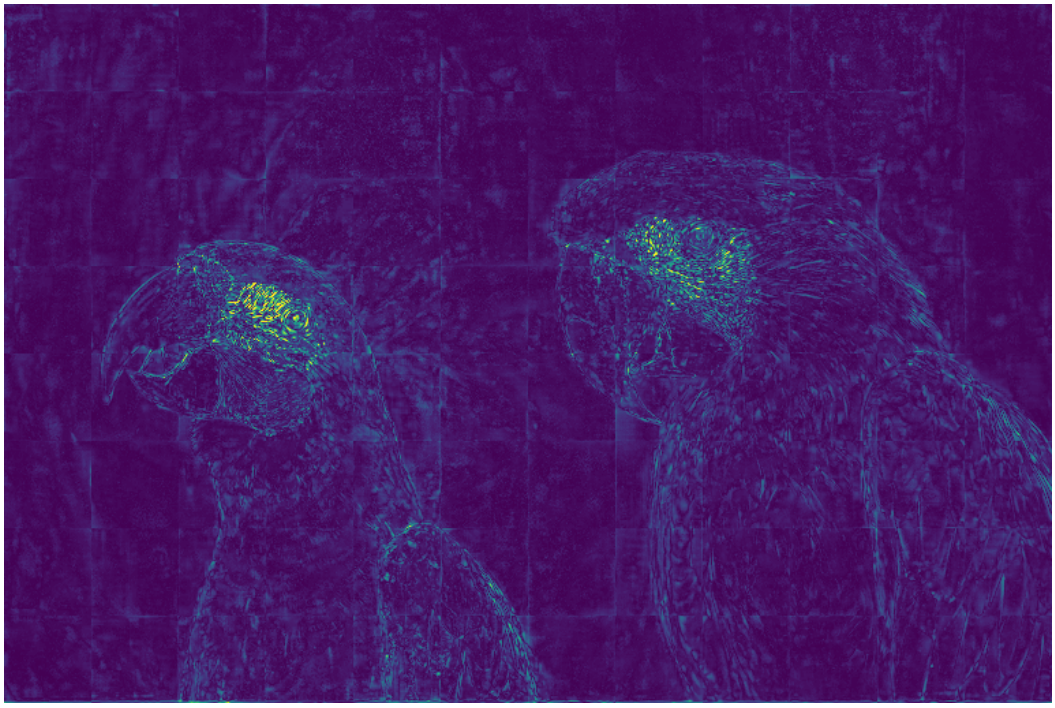
Fig. 4.12 Ablation study of learnable positional encoding and linear transformation on Cifar-10 and Kodak, and random permutation on Kodak. In the experiments without learnable positional encoding, the input of the INR only consists of the Fourier embeddings of the coordinates.

On the other hand, the learnable positional encoding is more effective at lower bit-rate and on larger images, aligning with our previous discussion. It provides a 0.5 dB improvement on both Kodak and Cifar-10 at low rates. However, for Cifar-10, no improvement is observed when the bit-rate surpasses 2 bpp.

Besides, the ablation study also demonstrates the effectiveness of random permutation, which provides a notable improvement of around 1 dB on Kodak.



(a) Residuals without random permutation (PSNR 28.20 dB)



(b) Residuals with random permutation (PSNR 29.48 dB)

Fig. 4.13 Residuals of decoded Kodak-23 at 0.078 bpp with and without random permutation.

4.2.4 Transferability across Modalities

In this section, we demonstrate the modality transferability of COMBINER+ using audio data. We evaluate our method on LibriSpeech [Panayotov et al. (2015)], a speech dataset recorded at a 16kHz sampling rate. Following Dupont et al. (2022), we take the first 3 seconds of every recording, corresponding to 48,000 audio samples. For each audio recording, we crop it into 60 non-overlapped "patches", with each containing 800 audio samples. We train COMBIENR+ on 11,820 training "patches" (corresponding to 197 training instances), and evaluate on 24 randomly selected test instances. During testing, we also apply random permutation across "patches". We describe the detailed experimental setup in Table C.2 in Appendix.

The Rate-distortion curve is depicted in Fig 4.11. We also provide results of COMBINER and MP3 on these 24 test instances for reference[3]. We can see COMBINER+ significantly outperforms both of them, which strongly evidences the transferability and effectiveness of COMBINER+. We also provide links to the decoded examples in Appendix D.3.

Note that the main aim of this experiment is to show of the transferability of COMBINER+ across modalities. We hence do not tune any hyperparameters apart from replacing 2D convolutional layers in ϕ_{CNN} with 1D convolutional layers. We may expect an improved performance with more careful tuning.

4.3 Summary

In this chapter, we propose COMBINER+. Built on top of COMBINER, it enhances the performance greatly via simple modifications. While a performance gap exists in comparison to state-of-the-art codecs and scalability considerations arise, COMBINER+ demonstrates the significant benefits of promoting parameter sharing within the network and the potential of integrating *implicit* representations with *explicit* representations (e.g., positional encodings), without loss of modality transferability.

[3]We acknowledge Zongyu Guo for providing the experimental results of COMBINER and MP3 on these 24 instances.

Chapter 5

Conclusions

In this thesis, we propose COMBINER and COMBINER+. The former provides a general framework of data compression, demonstrating strong performance within the realm of INR-based codes. While many current studies in INR-based compression strive to develop increasingly complicated parameterizations to enforce sparsity or low-rankness, our approach forges a novel path, naturally supporting joint rate-distortion optimization. On top of this framework, we introduce COMBINER+, significantly improving the performance without loss of simplicity or transferability. Despite the gap in comparison to VAE-based codecs on Kodak, COMBINER+ showcases remarkable benefits of sharing parameters within the network, and the potential of integrating *implicit* representations with *explicit* representations, which, we believe, indicates a valuable direction for enhancing INR-based approaches.

5.1 Limitations

A major limitation of our work is the encoding time complexity. Both COMBINER and COMBINER+ require thousands of gradient iterations to acquire the variational posterior, and also need extra thousands of iterations to perform fine-tuning. While a VAE encoder could be constructed to amortize the learning of the latent positional encoding in COMBINER+, training and fine-tuning the INR parameters still demand extensive iterations. A potential remedy involves Probabilistic MAML [Finn et al. (2018)], but it will impact the stability of the prior learning stage.

Another challenge arises in terms of scalability. This is a common issue for most current INR-based codecs, including [Dupont et al. (2022); Schwarz and Teh (2022); Schwarz et al. (2023)]. In COMBINER+, we employ patching to mitigate this problem, albeit with the drawback of inducing block artifacts and introducing spatial redundancy. Given the successes observed with positional encoding, which does not suffer from scalability, a possible strategy

is to emphasis positional encoding components and utilize a smaller INR for the entire image. However, this might entail a larger convolutional network, which potentially offsets the benefits of INRs.

5.2 Future Works

Besides addressing the aforementioned limitations, future works involves different distributions, different losses, and different data modalities.

Different distributions: Introducing more complicated priors have demonstrated strong benefits in VAE-based codecs. Ballé et al. (2018) introduced a scale hyper-prior to eliminate the spatial redundancy in the latent space; following that, Minnen et al. (2018) proposed an autoregressive context model, and hyper-priors on both mean and scale. Cheng et al. (2020) further demonstrated the effectiveness of Gaussian Mixture models. Although their motivation is to obtain a more accurate entropy model, these methods might be inspiring for the prior in our setting as well.

Different losses: Perceptual loss, which prioritizes generating realistic images over minimizing MSE, is gaining attention in neural lossy compression. Current efforts involve introducing adversarial loss to VAEs [Mentzer et al. (2020)] or directly applying diffusion models [Theis et al. (2022)]. However, the role of INR in this context remains unclear. One potential avenue is integrating INR with sinGAN [Shaham et al. (2019)], a model learning statistics from a single image. Alternatively, INR-based generative models like Poly-GAN [Singh et al. (2023)] are emerging. Investigating these architectures presents an intriguing direction.

Different Modalities: In this work, we only demonstrate the transferability of COMBINER+ to audio. Future work can center on applying our methods to other modalities like MRI scans, or 3D shapes. Designing suitable 3D convolution networks for positional encodings will be a key consideration in these modalities.

Video compression is also a challenging task. Treating videos directly as 3D tensors, similar to how MRI scans or 3D shapes are handled, is not practical due to the considerable size along the temporal axis. NeRV [Chen et al. (2021a)] tackled this by representing videos as convolutional neural networks, which take the frame index as the input and output the corresponding frame. There also exist recent works [Chen et al. (2023); Gomes et al. (2023); Kim et al. (2022)] investigating hybrid INRs for video representation or compression. Extending our approaches to these architectures is also a direction worth exploring.

References

Eirikur Agustsson and Radu Timofte. Ntire 2017 challenge on single image super-resolution: Dataset and study. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016.

Johannes Ballé, Valero Laparra, and Eero P. Simoncelli. End-to-end optimized image compression, 2017.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior, 2018.

Toby Berger. *Rate-Distortion Theory*. John Wiley & Sons, Ltd, 2003. ISBN 9780471219286. doi: https://doi.org/10.1002/0471219282.eot142. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/0471219282.eot142.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 0387310738.

Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks, 2015.

Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng, editors. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC, may 2011. doi: 10.1201/b10905. URL https://doi.org/10.1201%2Fb10905.

Hao Chen, Bo He, Hanyu Wang, Yixuan Ren, Ser-Nam Lim, and Abhinav Shrivastava. Nerv: Neural representations for videos, 2021a.

Hao Chen, Matt Gwilliam, Ser-Nam Lim, and Abhinav Shrivastava. Hnerv: A hybrid neural representation for videos, 2023.

Wenlin Chen, James T. Wilson, Stephen Tyree, Kilian Q. Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick, 2015.

Yinbo Chen, Sifei Liu, and Xiaolong Wang. Learning continuous image representation with local implicit image function, 2021b.

Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules, 2020.

Misha Denil, Babak Shakibi, Laurent Dinh, Marc' Aurelio Ranzato, and Nando de Freitas. Predicting parameters in deep learning. In C.J. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013. URL https://proceedings.neurips.cc/paper_files/paper/2013/file/7fec306d1e665bc9c748b5d2b99a6e97-Paper.pdf.

Simon Duane, A.D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216–222, 1987. ISSN 0370-2693. doi: https://doi.org/10.1016/0370-2693(87)91197-X. URL https://www.sciencedirect.com/science/article/pii/037026938791197X.

Emilien Dupont, Adam Goliński, Milad Alizadeh, Yee Whye Teh, and Arnaud Doucet. Coin: Compression with implicit neural representations, 2021.

Emilien Dupont, Hrushikesh Loya, Milad Alizadeh, Adam Goliński, Yee Whye Teh, and Arnaud Doucet. Coin++: Neural compression across modalities, 2022.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper_files/paper/2018/file/8e2c381d4dd04f1c55093f22c59c3a08-Paper.pdf.

Gergely Flamich, Stratis Markou, and Jose Miguel Hernandez-Lobato. Fast relative entropy coding with a* coding. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 6548–6577. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/flamich22a.html.

Carlos Gomes, Roberto Azevedo, and Christopher Schroers. Video compression with entropy-constrained neural representations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 18497–18506, June 2023.

V.K. Goyal. Theoretical foundations of transform coding. *IEEE Signal Processing Magazine*, 18(5):9–21, 2001. doi: 10.1109/79.952802.

Alex Graves. Practical variational inference for neural networks. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011. URL https://proceedings.neurips.cc/paper_files/paper/2011/file/7eb3c8be3d411e8ebfab08eba5f49632-Paper.pdf.

Zongyu Guo, Zhizheng Zhang, Runsen Feng, and Zhibo Chen. Causal contextual prediction for learned image compression. *IEEE Transactions on Circuits and Systems for Video Technology*, 32(4):2329–2341, 2022. doi: 10.1109/TCSVT.2021.3089491.

Marton Havasi. Advances in compression using probabilistic models. 2021. doi: 10.17863/CAM.79008. URL https://www.repository.cam.ac.uk/handle/1810/331555.

Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=r1f0YiCctm.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

Langwen Huang and Torsten Hoefler. Compressing multidimensional weather and climate data into neural networks, 2023.

David A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101, 1952. doi: 10.1109/JRPROC.1952.273898.

Chiyu Max Jiang, Avneesh Sud, Ameesh Makadia, Jingwei Huang, Matthias Nießner, and Thomas Funkhouser. Local implicit grid representations for 3d scenes, 2020.

J. Jiang. Image compression with neural networks – a survey. *Signal Processing: Image Communication*, 14(9):737–760, 1999. ISSN 0923-5965. doi: https://doi.org/10.1016/S0923-5965(98)00041-1. URL https://www.sciencedirect.com/science/article/pii/S0923596598000411.

Subin Kim, Sihyun Yu, Jaeho Lee, and Jinwoo Shin. Scalable neural video representations with learnable positional features, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

Diederik P. Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks, 2017.

Eastman Kodak. Kodak lossless true color image suite (photocd pcd0992). http://r0k.us/graphics/kodak/, 1993.

Alex Krizhevsky. Learning multiple layers of features from tiny images. pages 32–33, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Cheuk Ting Li and Abbas El Gamal. Strong functional representation lemma and applications to coding theorems, 2018.

Christos Louizos, Max Welling, and Diederik P. Kingma. Learning sparse neural networks through l_0 regularization, 2018.

Ishit Mehta, Michaël Gharbi, Connelly Barnes, Eli Shechtman, Ravi Ramamoorthi, and Manmohan Chandraker. Modulated periodic activations for generalizable local functional representations, 2021.

Fabian Mentzer, George D Toderici, Michael Tschannen, and Eirikur Agustsson. High-fidelity generative image compression. *Advances in Neural Information Processing Systems*, 33, 2020.

David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression, 2018.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization, 2020.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

Radford M. Neal. Bayesian learning for neural networks. 1995. URL https://api.semanticscholar.org/CorpusID:60809283.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: An asr corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015. URL https://api.semanticscholar.org/CorpusID:2191379.

George Papandreou and Alan Loddon Yuille. Perturb-and-map random fields: Using discrete optimization to learn and sample from energy models. *2011 International Conference on Computer Vision*, pages 193–200, 2011.

Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *International Conference on Learning Representations*, 2018. URL https://api.semanticscholar.org/CorpusID:52366640.

Jonathan Richard Schwarz and Yee Whye Teh. Meta-learning sparse compression networks, 2022.

Jonathan Richard Schwarz, Jihoon Tack, Yee Whye Teh, Jaeho Lee, and Jinwoo Shin. Modality-agnostic variational compression of implicit neural representations, 2023.

Tamar Rott Shaham, Tali Dekel, and Tomer Michaeli. Singan: Learning a generative model from a single natural image, 2019.

C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948. doi: 10.1002/j.1538-7305.1948.tb01338.x.

Rajhans Singh, Ankita Shukla, and Pavan Turaga. Polynomial implicit neural representations for large diverse datasets, 2023.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 7462–7473. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/53c04118df112c13a8c34b38343b9c10-Paper.pdf.

Yannick Strümpler, Janis Postels, Ren Yang, Luc van Gool, and Federico Tombari. Implicit neural representations for image compression, 2022.

Matthew Tancik, Pratul P. Srinivasan, Ben Mildenhall, Sara Fridovich-Keil, Nithin Raghavan, Utkarsh Singhal, Ravi Ramamoorthi, Jonathan T. Barron, and Ren Ng. Fourier features let networks learn high frequency functions in low dimensional domains, 2020.

Lucas Theis and Noureldin Yosri. Algorithms for the communication of samples, 2022.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders, 2017.

Lucas Theis, Tim Salimans, Matthew D. Hoffman, and Fabian Mentzer. Lossy compression with gaussian diffusion, 2022.

George Toderici, Wenzhe Shi, Radu Timofte, Lucas Theis, Johannes Balle, Eirikur Agustsson, Nick Johnston, and Fabian Mentzer. Workshop and challenge on learned image compression (clic2020), 2020. URL http://www.compression.cc.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004. doi: 10.1109/TIP.2003.819861.

Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

Ian H. Witten, Radford M. Neal, and John G. Cleary. Arithmetic coding for data compression. *Commun. ACM*, 30(6):520–540, jun 1987. ISSN 0001-0782. doi: 10.1145/214762.214771. URL https://doi.org/10.1145/214762.214771.

Yibo Yang, Stephan Mandt, and Lucas Theis. An introduction to neural data compression, 2022.

Appendix A

Relative Entropy Coding with A* Coding Algorithm

A.1 Pseudocodes

We provide A* encoding and decoding pseudocodes in this section. TruncGumbel$(\cdot)$ in Algorithm 6 represent a standard Gumbel random variable truncated by an upper bound[1].

Algorithm 6 A* encoding

Require: Proposal (prior) distribution $p_{\mathbf{w}}$ and target (posterior) distribution $q_{\mathbf{w}}$.

Initialize : $N \leftarrow 2^{\lfloor D_{\mathrm{KL}}[q_{\mathbf{w}}||p_{\mathbf{w}}]+t \rfloor}$ ▷ Initialize sample size
Initialize : $G^{(0)} \leftarrow \infty$ ▷ Set the upper bound of truncated Gumbel distribution to ∞
Initialize : $\boldsymbol{w}^*, n^* \leftarrow \perp, \perp$ ▷ $\perp$ is a placeholder
Initialize : $L \leftarrow -\infty$ ▷ Set the largest important weight to be $-\infty$

for $n = 1, ..., N$ **do** ▷ N samples from proposal distribution
  $\boldsymbol{w}^{(n)} \sim p_{\mathbf{w}}$
  $G^{(n)} \sim \text{TruncGumbel}(G^{(i-1)})$ ▷ Truncated Gumbel noise
  $L_n \leftarrow G^{(n)} + \log\left(q_{\mathbf{w}}(\boldsymbol{w}^{(n)}) / p_{\mathbf{w}}(\boldsymbol{w}^{(n)})\right)$ ▷ Perturbed importance weight
  if $L_n \geq L$ **then**
    $L \leftarrow L_n$
    $\boldsymbol{w}^*, n^* \leftarrow \boldsymbol{w}^{(n)}, n$ ▷ Keep the sample with largest importance weight
  end if
end for
return $\boldsymbol{w}^*, n^*$ ▷ $\boldsymbol{w}^*$ is encoded by its index n^*

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

Algorithm 7 A* decoding

Require: Proposal (prior) distribution $p_{\mathbf{w}}$, sample size N

Simulate $\{\boldsymbol{w}^{(n)}\}_{n=1}^{N}$ from $p_{\mathbf{w}}$ ▷ Simulate N samples from $p_{\mathbf{w}}$ with the shared seed
Receive n^* ▷ Receive the code

return $\boldsymbol{w}^* \leftarrow \boldsymbol{w}^{(n^*)}$ ▷ Retrieve the approximate posterior sample

A.2 Bound on the Bias of A* Coding

Here, we present the bound of the bias of A* coding samples. Assuming we use $p(\mathbf{z})$ to encode a sample $\boldsymbol{z} \sim q(\mathbf{z})$ by A* coding. The real underlying distribution of A* samples is $\tilde{q}(\mathbf{z})$. The bound on the total variation between $\tilde{q}(\mathbf{z})$ and $q(\mathbf{z})$ is given by Lemma A.2.1 [(Theis and Yosri, 2022)].

Lemma A.2.1 (Bound on the total variation between $\tilde{q}(\mathbf{z})$ and $q(\mathbf{z})$)**.** *Assuming we run A* encoding (Algorithm 6) with $N = 2^{D_{KL}[q||p]+t}$ samples for some parameter $t \geq 0$, then*

$$D_{TV}[\tilde{q}, q] \leq 4\varepsilon, \tag{A.1}$$

where

$$\varepsilon = \left(2^{-t/4} + 2\sqrt{\mathbb{P}_{\boldsymbol{z}\sim q}\left\{\log_2 \frac{q(\boldsymbol{z})}{p(\boldsymbol{z})} \geq D_{KL}[q||p] + t/2\right\}}\right)^{1/2}. \tag{A.2}$$

$\mathbb{P}$ is the probability of an event. This result indicates that at least $N \approx 2^{D_{\mathrm{KL}}[q||p]}$ samples need to be drawn in A* coding to ensure low sample bias, and the bias decreases exponentially as t increases towards infinity. However, it is essential to note that the sample size also grows exponentially with t. In practice, we observed that when $D_{\mathrm{KL}}[q||p]$ exceeds 16 bits, even for $t = 0$, we obtain samples of sufficient quality in our application.

Appendix B

Closed-Form Update for the Model Prior

We derive the closed-form update for the model prior in Eqaution 3.8. Recall that, we want to minimize the *rate-distortion* objective across all training data:

$$\mathcal{L} = \frac{1}{M}\sum_{m=1}^{M} \mathcal{L}(q_{\mathbf{w}}^{(m)}, p_{\mathbf{w}}, \mathcal{D}_m) \tag{B.1}$$

$$= \frac{1}{M}\sum_{m=1}^{M} \left\{ \mathbb{E}_{\boldsymbol{w}\sim q_{\mathbf{w}}^{(m)}} \left[\sum_{(\boldsymbol{x},\boldsymbol{y})\in\mathcal{D}_m} \Delta(\boldsymbol{y}, \hat{\boldsymbol{y}}_{\boldsymbol{x},\boldsymbol{w}}) \right] + \beta \cdot D_{\mathrm{KL}}[q_{\mathbf{w}}^{(m)} || p_{\mathbf{w}}] \right\}. \tag{B.2}$$

Now calculate the derivative w.r.t. the prior's parameters $\boldsymbol{\mu}_p$, $\boldsymbol{\sigma}_p^2$, yielding

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_p} = \frac{\beta}{M}\sum_m \frac{\partial D_{\mathrm{KL}}[q_{\mathbf{w}}^{(m)} || p_{\mathbf{w}}]}{\partial \boldsymbol{\mu}_p}; \tag{B.3}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\sigma}_p^2} = \frac{\beta}{M}\sum_m \frac{\partial D_{\mathrm{KL}}[q_{\mathbf{w}}^{(m)} || p_{\mathbf{w}}]}{\partial \boldsymbol{\sigma}_p^2}. \tag{B.4}$$

Note that the KL between two fully-factorized Gaussian is given by

$$D_{\mathrm{KL}}[q_{\mathbf{w}}^{(m)} || p_{\mathbf{w}}] = \frac{1}{2}\log\frac{\boldsymbol{\sigma}_p^2}{\left(\boldsymbol{\sigma}_q{}^{(m)}\right)^2} + \frac{\left(\boldsymbol{\sigma}_q{}^{(m)}\right)^2 + \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q{}^{(m)}\right)^2}{2\boldsymbol{\sigma}_p^2} - \frac{1}{2}. \tag{B.5}$$

The logarithm and fractions are element-wise operation.

Therefore, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_p} \propto \sum_m \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q^{(m)} \right) \tag{B.6}$$

$$= M \cdot \boldsymbol{\mu}_p - \sum_m \boldsymbol{\mu}_q^{(m)}; \tag{B.7}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\sigma}_p^2} \propto \sum_m \left(\frac{1}{2\boldsymbol{\sigma}_p^2} - \frac{\left(\boldsymbol{\sigma}_q^{(m)}\right)^2 + \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q^{(m)}\right)^2}{2\left(\boldsymbol{\sigma}_p^2\right)^2} \right). \tag{B.8}$$

Now, setting both of the derivatives to be $\mathbf{0}$, we obtain the closed-form solutions, i.e.,

$$\boldsymbol{\mu}_p = \frac{1}{M} \sum_m \boldsymbol{\mu}_q^{(m)}; \tag{B.9}$$

$$\boldsymbol{\sigma}_p^2 = \frac{1}{M} \sum_m \left(\boldsymbol{\sigma}_q^{(m)}\right)^2 + \left(\boldsymbol{\mu}_p - \boldsymbol{\mu}_q^{(m)}\right)^2 \tag{B.10}$$

Appendix C

Supplementary Experiments Details

	Cifar-10	Kodak	
		small model	**large model**
INR Architecture			
layers	4	6	7
hidden units	16	48	56
Fourier embedding dimension	32	64	96
number of parameters	1123	12675	21563
Prior Learning			
training size	2048	512	
epochs	128	96	
optimizer	Adam (lr=0.0002)	Adam (lr=0.0001)	
SGD iterations between updating prior	100	200	
SGD iterations before updating prior	250	500	
initial posterior variance	9×10^{-6}	4×10^{-6}	$\{4 \times 10^{-6}, 4 \times 10^{-10}\}$
initial posterior mean	SIREN initialization		
β	$\{2 \times 10^{-5}, 5 \times 10^{-6}, 2 \times 10^{-6}$ $1 \times 10^{-6}, 5 \times 10^{-7}\}$	$\{10^{-7}, 10^{-8}, 4 \times 10^{-7}\}$	$\{4 \times 10^{-6}, 4 \times 10^{-6}\}$
Posterior Learning			
gradient descent iterations	25000	25000	
optimizer	Adam (lr=0.0002)	Adam (lr=0.0001)	
number of blocks	$\{58, 89, 146, 224, 285\}$	$\{1729, 2962, 3264\}$	$\{5503, 7176\}$

Table C.1 Experimental setting of our COMBINER experiments.

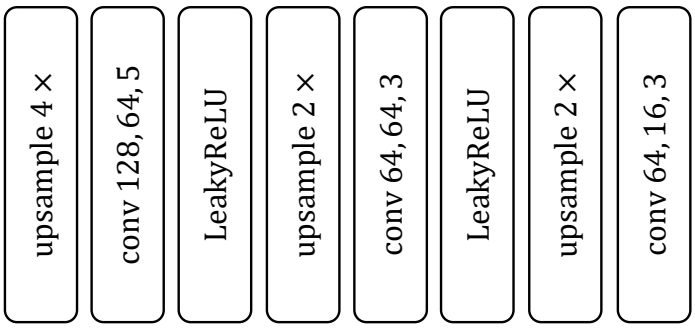


Fig. C.1 Architecture of the up-sampling network $\phi_{\text{CNN}}(\cdot)$ in COMBINER+. The parameters in the convolution layer represent the number of input channels, the number of output channels, and kernel size respectively. Same padding is used in all convolution layers.

Appendix D

Supplementary Experiments Results

D.1 Coding Time

We provide encoding and decoding time in details in this section.

bit-rate	Encoding (500 images, GPU)			Decoding (1 image, CPU)
	Learning Posterior	REC w. Fine-tuning	Total	
0.91 bpp	∼7 min	∼6 min	∼13 min	0.00206 s
1.39 bpp		∼9 min	∼16 min	0.00209 s
2.28 bpp		∼14 min 30 s	∼21 min 30 s	0.00286 s
3.50 bpp		∼21 min 30 s	∼28 min 30 s	0.00382 s
4.45 bpp		∼27 min	∼33 min	0.00388 s

Table D.1 Encoding time and decoding time of COMBINER on Cifar-10 dataset.

bit-rate	Encoding (1 images, GPU)			Decoding (1 image, CPU)
	Learning Posterior	REC w. Fine-tuning	Total	
0.070 bpp	∼9 min	∼12 min 30 s	∼21 min 30 s	0.34842 s
0.110 bpp		∼18 mins	∼27 min	0.38153 s
0.132 bpp		∼22 min	∼31 min	0.40538 s
0.224 bpp	∼11 min	∼50 min	∼61 min	0.59739 s
0.293 bpp		∼68 min	∼79 min	0.60232 s

Table D.2 Encoding time and decoding time of COMBINER on Kodak dataset.

bit-rate	Encoding (500 images, GPU)			Decoding (1 image, CPU)
	Learning Posterior	REC w. Fine-tuning	Total	
0.297 bpp	∼30 min	∼33 min	∼63 min	0.00386 s
0.719 bpp		∼35 min	∼65 min	0.00429 s
0.938 bpp		∼37 min 30 s	∼67 min 30 s	0.00461 s
1.531 bpp		∼42 min	∼72 min	0.00514 s
1.922 bpp		∼45 min	∼75 min	0.00581 s
3.344 bpp		∼56 min 30 s	∼86 min 30 s	0.00776 s
4.391 bpp		∼63 min	∼93 min	0.01050 s

Table D.3 Encoding time and decoding time of COMBINER+ on Cifar-10 dataset.

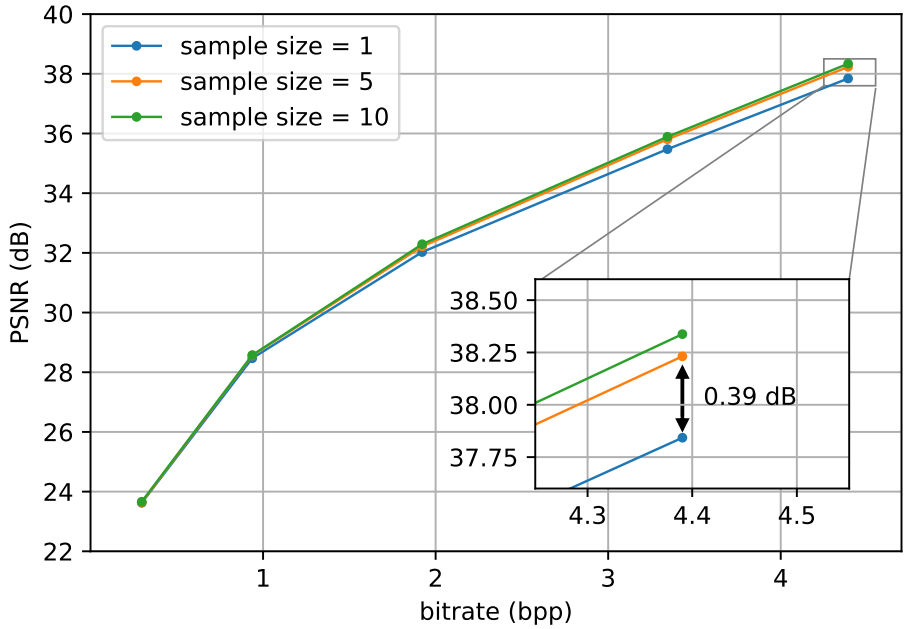
bit-rate	Encoding (1 images, GPU)			Decoding (1 image, CPU)
	Learning Posterior	REC w. Fine-tuning	Total	
0.078 bpp	∼24 min	∼24 min	∼48 min	0.17682 s
0.148 bpp		∼24 min 30 s	∼48 min 30 s	0.20884 s
0.203 bpp		∼25 min	∼49 min	0.25013 s
0.273 bpp		∼25 min	∼49 min	0.25380 s
0.379 bpp		∼25 min 30 s	∼49 min 30 s	0.28652 s
0.484 bpp		∼25 min 30 s	∼49 min 30 s	0.32980 s

Table D.4 Encoding time and decoding time of COMBINER+ on Kodak dataset.

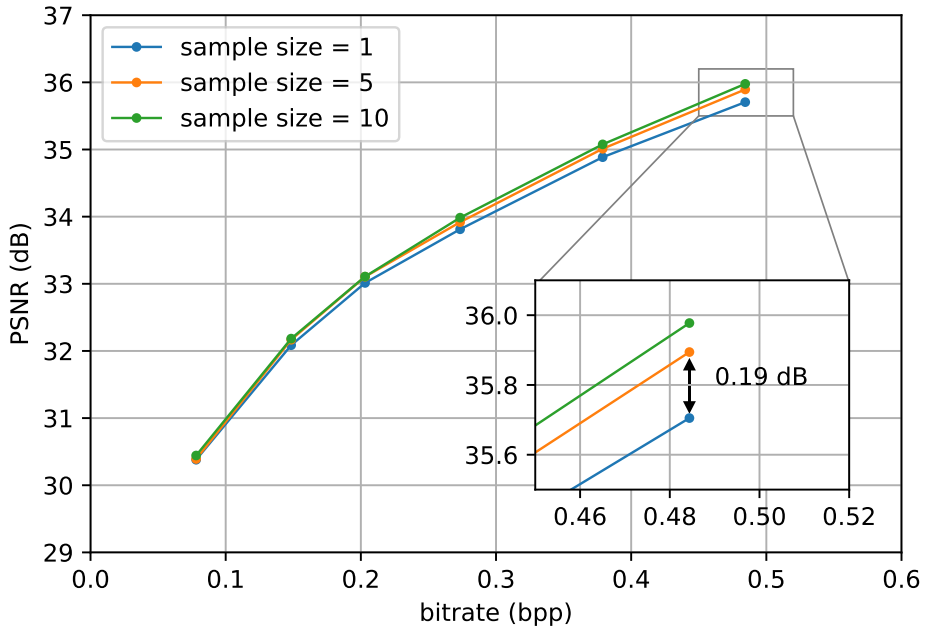
D.2 Influence of Sample Size in COMBINER+

In COMBINER+, we use 5 samples to optimize Equation (4.11) when learning the posterior for a test datum. Here, we provide the R-D curve using 1, 5 and 10 samples, on 500 randomly selected Cifar-10 test images and Kodak-03 as examples. The curves are shown in Fig D.1. We can see the sample size mainly impacts the performance at high bit-rates. Besides, further increasing the sample size to 10 only brings a minor improvement. Therefore, we choose 5 samples in our experiments to balance between encoding time and performance.

It is also notable that using 1 sample does not greatly degrade the performance. Thus, we can choose to reduce the sample size when emphasizing encoding time, with minimal impact on performance.



(a) on 500 Randomly selected Cifar-10 test images



(b) on Kodak-03

Fig. D.1 Rate-distortion curve of COMBINER+ with different sample sizes.

D.3 Decoded Audio Examples

Here, we provide links to the decoded audio of one test example[1]: [Ground Truth]; [Decoded audio at 5.44 kbps]; [Decoded audio at 10.56 kbps]; [Decoded audio at 21.44 kbps].

[1] In case the hyperlink does not work, the folder containing the examples is available at https://drive.google.com/drive/folders/1CLkxWMsjytxD3KhIjssClnHTHMO_HIh2?usp=drive_link