Learned Compression for Images and Point Clouds

by

Mateen Ulhaq

B.A.Sc. (Hons.), Simon Fraser University, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Sciences

SIMON FRASER UNIVERSITY
Fall 2023

Declaration of Committee

Name:	**Mateen Ulhaq**
Degree:	**Master of Applied Science (Engineering)**
Thesis title:	**Learned Compression for Images and Point Clouds**
Committee:	**Chair:** **Andrew Rawicz** Professor, School of Engineering Science
	Ivan V. Bajić Supervisor Professor, School of Engineering Science
	Mirza Faisal Beg Committee Member Professor, School of Engineering Science
	Jie Liang Examiner Professor, School of Engineering Science

Abstract

Over the last decade, deep learning has shown great success at performing computer vision tasks, including classification, super-resolution, and style transfer. Now, we apply it to data compression to help build the next generation of multimedia codecs. This thesis provides three primary contributions to this new field of learned compression. First, we present an efficient low-complexity entropy model that dynamically adapts the encoding distribution to a specific input by compressing and transmitting the encoding distribution itself as side information. Secondly, we propose a novel lightweight low-complexity point cloud codec that is highly specialized for classification, attaining significant reductions in bitrate compared to non-specialized codecs. Lastly, we explore how motion within the input domain between consecutive video frames is manifested in the corresponding convolutionally-derived latent space.

Keywords: deep learning; image compression; point cloud compression; video coding for machines

Acknowledgments

I would like to thank my advisor, Dr. Ivan V. Bajić, for his guidance and support throughout my graduate studies. His feedback in reviewing and editing my papers and thesis was invaluable, as was his advice on research and career development. He first supervised my undergraduate Bachelor's thesis, during which his encouragement motivated me to develop a prototype for demonstration at the NeurIPS 2019 conference. His insight in the subject areas of compression, information theory, and deep learning helped guide me in my research.

I would also like to thank my committee members, including: Dr. Mirza Faisal Beg, who was my supervisor for an NSERC Undergraduate Student Research Awards (USRA) research project, and provided me with an opportunity to work on a challenging problem in computer vision, related to medical mobile applications; Dr. Jie Liang, who first inspired and intrigued me with his lectures on Multimedia Communications Engineering involving the subjects of compression and deep learning; and Dr. Andrew Rawicz, who taught our Capstone project course during my undergraduate studies, instilling within us the value of a practical approach to engineering, as well as skills for managing technical projects.

I also greatly enjoyed my time as an intern at Interdigital's AI Lab headed by Dr. Jaideep Chandrashekar. My discussions with Dr. Fabien Racapé and Dr. Hyomin Choi significantly improved my understanding of this field. I also enjoyed my discussions with other interns, namely Ujwal Dinesha and Ezgi Özyılkan, who imparted upon me a greater appreciation for reinforcement learning and information theory, respectively.

I would also like to show my appreciation for the community at SFU, including my friends and peers, as well as the professors of the schools of engineering science and computer science, and the departments of physics and mathematics. Within the SFU Multimedia Lab research group, I would also mention Alon Harell and Anderson de Andrade for their interest in my work as well as discussions within the subject area.

And of course, for the never-ending support from my family, including my mother and sister, I am grateful.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The storage and transmission of data is a fundamental aspect of computing. However, every bit stored or transmitted incurs a cost. To mitigate this cost, we turn to data compression. Data compression is the process of reducing the amount of bits required to represent, store, or transmit data.

The modern age of the Internet would look very different without data compression: web pages would take much longer to load, and images and video would be much lower in resolution. The transmission of an uncompressed video stream would require thousands of times more bits than its compressed counterpart. Video streaming services such as Netflix and YouTube would suffer from a much higher operating cost, and would be much less accessible to the average consumer.

Data compression algorithms, known as *codecs*, are often specialized for encoding a particular type of data. Common types of data include text, images, video, audio, and point clouds. Data from such sources often contains redundancy or patterns which can be identified and eliminated by a compression algorithm to represent the data more compactly. For instance, pixels that are near each other in an image are often similar in color, which is a phenomenon known as *spatial redundancy*.

1.1 Data compression: an example

As an example of how data compression works, consider the random variable X representing the summer weather in the city of Vancouver, Canada. Let us say that the possible weather conditions $(\texttt{Sunny}, \texttt{Rainy}, \texttt{Cloudy})$ abbreviated as $(\texttt{S}, \texttt{R}, \texttt{C})$ are predicted to occur with the probabilities $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$, respectively. To compress a sequence of weather observations $X_1, X_2, \ldots, X_n$, we can use a codebook that maps each weather condition to a binary string:

$$\texttt{S} \to \texttt{0},$$
$$\texttt{R} \to \texttt{10},$$
$$\texttt{C} \to \texttt{11}.$$

Then, a sequence of weather observations such as the 64-bit ASCII string "SRCSSRCS" can be represented more compactly as the *encoded* 12-bit binary string "010110010110".

Notably, for any given input $x \in \{\texttt{S}, \texttt{R}, \texttt{C}\}$, the length in bits of its encoded representation is equal to $-\log_2 P(X = x)$. That is, $-\log_2 \frac{1}{2} = 1$ bit for S, $-\log_2 \frac{1}{4} = 2$ bits for R, and $-\log_2 \frac{1}{4} = 2$ bits for C. Thus, if $X_1, X_2, \ldots, X_n$ are truly independently and identically distributed (i.i.d.), then this codebook is the optimal codebook.

However, many raw sources studied in compression are not i.i.d. In fact, there are often patterns or correlations between consecutive elements. For instance, in the weather example, it is more likely that the weather will be R on a given day if it was C on the previous day. Then, the probability distribution used for encoding should be reevaluated to a more realistic prediction, e.g. $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$. The codebook must then also be updated to dynamically match this new encoding distribution:

$$\texttt{S} \to \texttt{10},$$
$$\texttt{R} \to \texttt{0},$$
$$\texttt{C} \to \texttt{11}.$$

This is the optimal codebook for the new encoding distribution. It should be used instead of the general codebook whenever the previous weather observation is C.

We can compress even further by determining a more accurate encoding distribution that predicts the next observation more accurately. More sophisticated probability modeling might also take into account the weather from the past several days, or from the same day in previous years. A good model might blend in other related sources of information such as past and current humidity, temperature, and wind speed. It might also analyze such information on multiple scales: locally within the city, within the province, or within the continent. Such probability modeling is the work of a meteorologist... and also a data compression researcher! In data compression, this process of determining the encoding distribution on-the-fly based on previous information is known as *context modeling*; and more generally, for any way of determining the encoding distribution, as *entropy modeling*.

1.2 Learning-based compression: the current landscape

Deep learning based approaches have recently been applied to data compression. Learning-based approaches have demonstrated compression performance that is competitive with traditional standard codecs. For instance, Fig. 1.1 compares the Rate-Distortion (RD) performance curves for popular and state-of-the-art (SOTA) codecs in image compression, evaluated on generic non-specialized datasets.

Learned compression has been applied to various types of data including images, video, and point clouds. For learned image compression, most prominent are approaches based on Ballé *et al.*'s compressive variational autoencoder (VAE), including [2]–[4]. Other approaches based on RNNs and GANs have also been applied, including [5], [6]. Works in learned point cloud compression include [7]–[11], and works in learned video compression include [12]–[15].

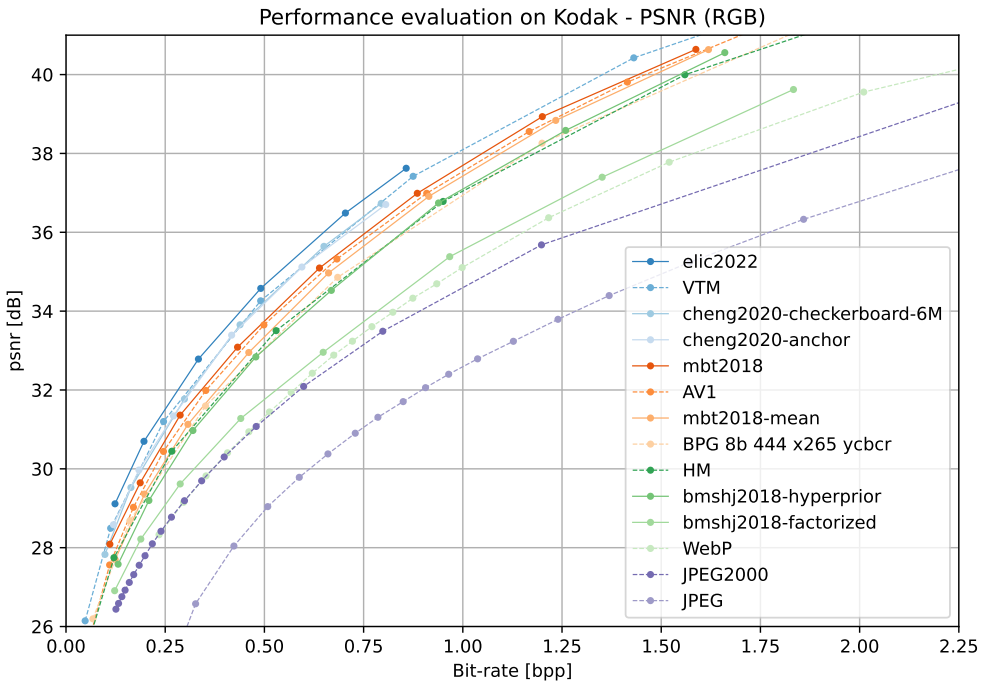


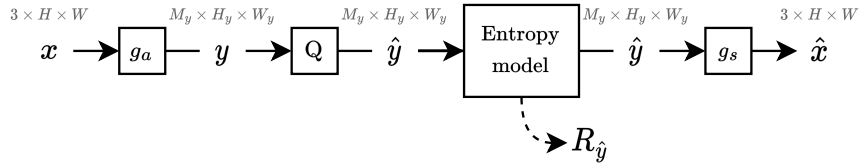
Figure 1.1: RD curves for image compression codecs on the Kodak dataset [1].

Currently, one factor inhibiting industry adoption of learning-based codecs is that they are much more computationally expensive than traditional codecs like JPEG and WebP. In fact, learned compression codecs exceed reasonable computational budgets by a factor of 100–10000×. To remedy this, there is work being done towards designing low-complexity codecs for image compression, including [16]–[19].

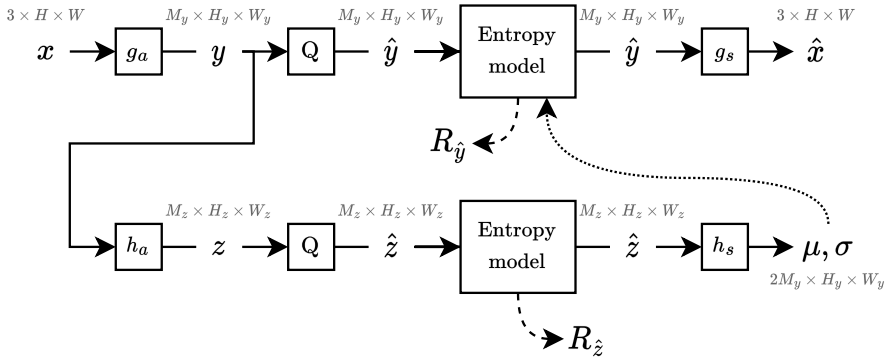
Learned compression has also shown benefits when applied to learned machine or computer vision tasks. In Coding for Machines (CfM) — also referred to as Video Coding for Machines (VCM) [20] — compression is used for machine tasks such as classification, object detection, and semantic segmentation. In this paradigm, the encoder-side device compresses the input into a compact task-specialized bitstream that is transmitted to the decoder-side device or server for further inference. This idea of partially processing the input allows for significantly lower bitrates in comparison to transmitting the entire unspecialized input for inference. Extending this technique, scalable multi-task codecs such as [21], [22] allocate a small base bitstream for machine tasks, and a larger enhancement bitstream for a higher-quality input reconstruction intended for human viewing.

1.3 Compression architecture overview

A simple compression architecture used by both traditional and learned compression methods alike is shown in Fig. 1.2a. This architecture consists of several components. Table 1.1 lists some common choices for the components in this architecture.



(a) simple



(b) hyperprior

Figure 1.2: High-level comparison of codec architectures.

Table 1.1: Components of a compression architecture for various image compression codecs.

Model	Quantizer	Entropy coding	Analysis transform (g_a)	Synthesis transform (g_s)
JPEG	non-uniform	zigzag + RLE, Huffman	8×8 block DCT	8×8 block DCT^{-1}
JPEG 2000	uniform dead-zone or trellis coded (TCQ)	arithmetic	multilevel DWT	multilevel DWT^{-1}
bmshj2018-factorized	uniform	arithmetic	(Conv, GDN) $\times 4$	(ConvT, IGDN) $\times 4$

In this architecture, the input $\boldsymbol{x}$ goes through a transform g_a to generate an intermediate representation y, which is quantized to $\hat{\boldsymbol{y}}$. Then, $\hat{\boldsymbol{y}}$ is losslessly entropy coded to generate a transmittable bitstream from which $\hat{\boldsymbol{y}}$ can be perfectly reconstructed. (at the decoder side.) Finally, $\hat{\boldsymbol{y}}$ is fed into a synthesis transform g_s which reconstructs an approximation of $\boldsymbol{x}$, which is labeled $\hat{\boldsymbol{x}}$.

Each of the components of the standard compression architecture are described in further detail below:

- **Analysis transform** (g_a): The input first is transformed by the g_a transform into a transformed representation $\boldsymbol{y}$. This transform often outputs a signal that contains less redundancy than within the input signal, and has its energy compacted into a smaller dimension. For instance, the JPEG codec transforms 8×8 blocks from the input image using a discrete cosine transform (DCT). This concentrates most of the signal energy into the low-frequency components that are often the dominating frequency component within natural images. In the case of learned compression, the analysis transform is often a nonlinear transform comprised of multiple deep layers and many parameters. For instance, the bmshj2018-factorized model's g_a transform contains 4 downsampling convolutional layers interleaved with GDN [23] nonlinear activations, totaling 1.5M to 3.5M parameters.

- **Quantization**: The analysis transform outputs coefficients contained in a rather large (potentially even continuous) support. However, much of this precision is not necessary for a reasonably accurate reconstruction. Thus, we drop most of this unneeded information by binning the transformed coefficients into a much smaller discretized support. There are many choices for the reconstructed quantization bin values and boundaries. Popular quantizers include the uniform, dead-zone, Lloyd-Max, and Trellis-coded quantizers. Ballé *et al.* [24] use a uniform quantizer during inference, which is replaced with additive unit-width uniform noise during training. More recently, the STE quantizer has also been used during training.

- **Entropy coding**: The resulting $\hat{\boldsymbol{y}}$ is losslessly compressed using an entropy coding method. The entropy coder is targeted to match a specific encoding distribution. Whenever the encoding distribution correctly predicts an encoded symbol with high probability, the relative bit cost for encoding that symbol is reduced. Thus, some entropy models are context-adaptive, and change the encoding distribution on-the-fly in order to accurately probabilistically predict the next encoded symbol value. Huffman coding is used in JPEG, though it has trouble replicating a given target encoding probability distribution and also at adapting to dynamically changing encoding distributions. Thus, more recent codecs prefer to use arithmetic coders, which can much better approximate rapidly changing target encoding distributions. The CompressAI [25] implementation uses rANS [26], [27], a popular recent innovation that is quite fast under certain conditions.

- **Synthesis transform** (g_s): Finally, the reconstructed quantized $\hat{\boldsymbol{y}}$ is fed into a synthesis transform g_s, which produces $\hat{\boldsymbol{x}}$. In JPEG, this is simply the inverse DCT. Similar to the analysis transform, in learned compression, the synthesis transform consists of several layers and many parameters. For instance, the bmshj2018-factorized model's g_s transform contains 4 upsampling transposed convolutional layers interleaved with IGDN nonlinear activations, totaling 1.5M to 3.5M parameters.

The length of the bitstream is known as the rate cost $R_{\hat{\boldsymbol{y}}}$, which we seek to minimize. We also seek to minimize the distortion $D(\boldsymbol{x}, \hat{\boldsymbol{x}})$, which is typically the mean squared error (MSE) between $\boldsymbol{x}$ and $\hat{\boldsymbol{x}}$. To balance these two competing goals, it is common to introduce a Lagrangian trade-off hyperparameter λ, so that the quantity sought to be minimized is $L = R_{\hat{\boldsymbol{y}}} + \lambda\, D(\boldsymbol{x}, \hat{\boldsymbol{x}})$.

1.4 Entropy modeling

A given element $\hat{y}_i \in \mathbb{Z}$ of the latent tensor $\hat{\boldsymbol{y}}$ is compressed using its encoding distribution $p_{\hat{y}_i} : \mathbb{Z} \to [0, 1]$, as visualized in Fig. 1.3. The rate cost for encoding $\hat{y}_i$ is the negative log-likelihood, $R_{\hat{y}_i} = -\log p_{\hat{y}_i}(\hat{y}_i)$, measured in bits. Afterward, the exact same encoding distribution is used by the decoder to reconstruct the encoded symbol.

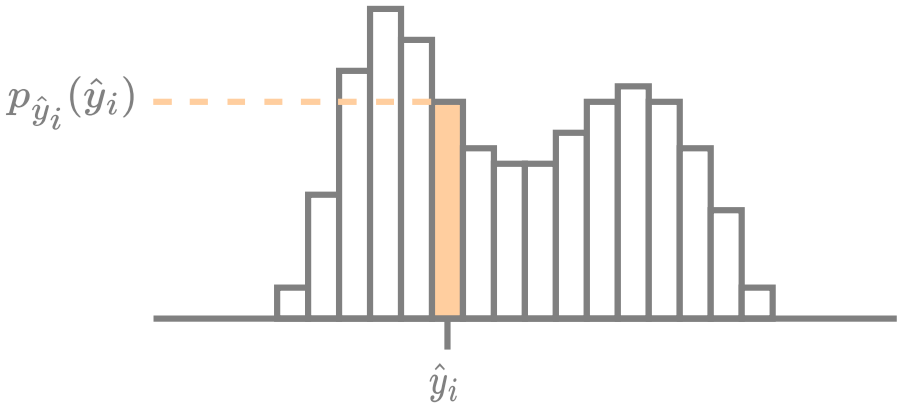


Figure 1.3: Visualization of an encoding distribution used for compressing a single element $\hat{y}_i$.

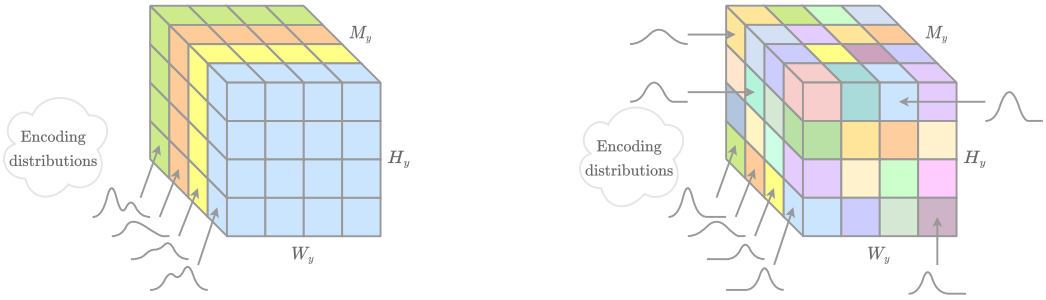
The encoding distributions are determined using an *entropy model*. Fig. 1.4 visualizes the encoding distributions generated by well-known entropy models. These are used to compress a latent tensor $\hat{\boldsymbol{y}}$ with dimensions $M_y \times H_y \times W_y$. The exact total rate cost for encoding $\hat{\boldsymbol{y}}$ using $p_{\hat{\boldsymbol{y}}}$ is simply the sum of the negative log-likelihoods of each element, $R_{\hat{\boldsymbol{y}}} = \sum_i -\log p_{\hat{y}_i}(\hat{y}_i)$.

Some popular choices for entropy models include:

- A "fully factorized" *entropy bottleneck*, as introduced by Ballé *et al.* [24]. Let $p_{\hat{y}_{c,i}} : \mathbb{Z} \to [0, 1]$ denote the probability mass distribution used to encode the i-th element $\hat{y}_{c,i}$ from the c-th channel of $\hat{\boldsymbol{y}}$. The same encoding distribution $p_{\hat{y}_c}$ is used for all elements within the c-th channel, i.e., $p_{\hat{y}_{c,i}} = p_{\hat{y}_c}, \forall i$. This entropy model works best when all such elements $\hat{y}_{c,1}, \hat{y}_{c,2}, \ldots, \hat{y}_{c,N}$ are independently and identically distributed (i.i.d.).

  Ballé *et al.* [24] model the encoding distribution as a static non-parametric distribution that is computed as the binned area under a probability density function $f_c : \mathbb{R} \to \mathbb{R}$, with a corresponding cumulative distribution function $F_c : \mathbb{R} \to [0, 1]$. Then,

$$p_{\hat{\boldsymbol{y}}_c}(\hat{y}_{c,i}) = \int_{-\frac{1}{2}}^{\frac{1}{2}} f(\hat{y}_{c,i} + \tau)\, d\tau = F_c(\hat{y}_{c,i} + 1/2) - F_c(\hat{y}_{c,i} - 1/2). \tag{1.1}$$



(a) fully factorized **(b)** conditional

Figure 1.4: Visualization of encoding distributions used for compressing a latent tensor $\hat{\boldsymbol{y}}$ with dimensions $M_y \times H_y \times W_y$. In (a), the encoding distributions within a given channel are all the same since the elements within a channel are assumed to be i.i.d. w.r.t. each other. Furthermore, in the case of the fully factorized entropy bottleneck used by Ballé *et al.* [24], each encoding distribution is a static non-parametric distribution. In (b), the encoding distributions for each element are uniquely determined, and conditioned on side information. Furthermore, in the case of the Gaussian conditional hyperprior used by Ballé *et al.* [24], the encoding distributions are Gaussian distributions parameterized by a mean and variance.

F_c is modelled using a small fully-connected network composed of five linear layers with channels of sizes $[1, 3, 3, 3, 3, 1]$, whose parameters are tuned during training. Note that F_c is not conditioned on any other information, and is thus static.

- A *Gaussian conditional*, as introduced by Ballé *et al.* [24]. Let $f_i(y) = \mathcal{N}(y; \mu_i, {\sigma_i}^2)$ be a Gaussian distribution with mean μ_i and variance ${\sigma_i}^2$. Then, like in (1.1), the encoding distribution $p_{\hat{\boldsymbol{y}}_i}$ is the binned area under f_i:

$$p_{\hat{\boldsymbol{y}}_i}(\hat{y}_i) = \int_{-\frac{1}{2}}^{\frac{1}{2}} f_i(\hat{y}_i + \tau)\, d\tau. \tag{1.2}$$

  In the mean-scale variant of the "hyperprior" model introduced by Ballé *et al.* [24], the parameters μ_i and ${\sigma_i}^2$ are computed by $[\mu_i, {\sigma_i}^2] = (h_s(\hat{\boldsymbol{z}}))_i$. Here, the latent representation $\hat{\boldsymbol{z}} = \text{Quantize}[h_a(\boldsymbol{y})]$ is computed by the analysis transform h_a, and then encoded using an entropy bottleneck and transmitted as *side information*; and h_s is a synthesis transform. This architecture is visualized in Fig. 1.2b. Cheng *et al.* [3] define f_i as a mixture of K Gaussians — known as a Gaussian mixture model (GMM) — with parameters $\mu_i^{(k)}$ and $\sigma_i^{(k)}$ for each Gaussian, alongside an affine combination of weights $w_i^{(1)}, \ldots, w_i^{(K)}$ that satisfy the constraint $\sum_k w_i^{(k)} = 1$. A GMM encoding distribution is thus defined as $f_i(y) = \sum_{k=1}^{K} w_i^{(k)}\, \mathcal{N}(y; \mu_i^{(k)}, (\sigma_i^{(k)})^2)$.

1.5 Thesis outline and contributions

This thesis presents three main contributions:

- In Chapter 2, "Compression of probability distributions", we present a method that can dynamically adapt the encoding distribution to the latent data distribution of a specific input. It does so by compressing and transmitting the distribution as side information. This is done using a small learned compression model that is specialized for compressing the encoding distributions. In comparison to competing methods such as a scale hyperprior, our method requires 25–130× less computation.

- In Chapter 3, "Point cloud compression for classification", we present a PointNet-based point cloud codec that is specialized for classification. Our codec attains a 94% reduction in BD-bitrate over non-specialized codecs on the ModelNet40 dataset. We also provide very lightweight model configurations of our codec that achieve similar bitrate improvements but with a very low computational cost.

- In Chapter 4, "Latent space motion analysis for collaborative intelligence", we analyze how motion between consecutive input image frames x_1 and x_2 affects the corresponding latent representations y_1 and y_2 of typical convolutional neural networks (CNNs). Specifically, given a known optical motion between x_1 and x_2, we quantify how well y_2 can be predicted by applying that same motion to y_1. Since learned video compression models encode convolutionally-derived latent representations, it is useful to know how predictably or naturally the latent space behaves under input motion.

Publications and work during MASc

Used within this thesis

1. **M. Ulhaq** and I. V. Bajić, "Learned point cloud compression for classification," in *Proc. IEEE MMSP*, 2023, arXiv: 2308.05959 [eess.IV]

2. **M. Ulhaq** and I. V. Bajić, "Latent space motion analysis for collaborative intelligence," in *Proc. IEEE ICASSP*, 2021, pp. 8498–8502, arXiv: 2102.04018 [cs.CV]

Other contributions

3. E. Özyılkan*, **M. Ulhaq***, H. Choi, and F. Racapé, "Learned disentangled latent representations for scalable image coding for humans and machines," in *Proc. IEEE DCC*, 2023, pp. 42–51, arXiv: 2301.04183 [eess.IV]

4. **M. Ulhaq** and F. Racapé, "CompressAI Trainer," 2022, [Online]. Available: https://github.com/InterDigitalInc/CompressAI-Trainer

5. H. Choi, F. Racapé, S. Hamidi-Rad, **M. Ulhaq**, and S. Feltman, "Frequency-aware learned image compression for quality scalability," in *Proc. IEEE VCIP*, 2022, pp. 1–5, arXiv: 2301.01290 [eess.IV]

6. S. R. Alvar, **M. Ulhaq**, H. Choi, and I. V. Bajić, "Joint image compression and denoising via latent-space scalability," *Frontiers in Signal Processing*, vol. 2, 2022, arXiv: 2205.01874 [eess.IV]

7. **M. Ulhaq** and I. V. Bajić, "ColliFlow: A library for executing collaborative intelligence graphs," demoed at NeurIPS, 2020, [Online]. Available: https://yodaembedding.github.io/neurips-2020-demo/

Chapter 2

Compression of probability distributions

The entropy bottleneck introduced by Ballé *et al.* [24] is a common component used in many learned compression models. It encodes a transformed latent representation using a static distribution whose parameters are learned during training. However, the actual distribution of the latent data may vary wildly across different inputs. The static distribution attempts to encompass all possible input distributions, thus fitting none of them particularly well. This unfortunate phenomenon, sometimes known as the amortization gap, results in suboptimal compression rates. Moreover, the transform also suffers difficulties from being constrained into targeting an inflexible static distribution, leading to further inefficiencies. To address these issues, we propose a method that dynamically adapts the encoding distribution to match the latent data distribution for a specific input. First, our model estimates a better encoding distribution for a given input. This distribution is then compressed and transmitted as an additional side-information bitstream. Finally, the decoder reconstructs the encoding distribution and uses it to decompress the corresponding latent data. Our method achieves a BD-rate gain of -6.95% on the Kodak test dataset when applied to the standard fully factorized architecture. Furthermore, the transform used by our method costs only as few as 10 MACs/pixel, in contrast to related side-information methods such as the scale hyperprior, which costs at least 1300 MACs/pixel.

2.1 Introduction

At the heart of compression is the probability distribution. Typically, we model the data to be compressed using a probability distribution. It is less common to model the probability distribution itself. Nonetheless, as will soon see, there is potential value in doing so.

In this chapter, we explore the compression of probability distributions that are used in compression. Specifically, we will focus on probability distributions that are used to model the latent representations produced by an image compression model. The entropy models of learned image compression models utilize probability distributions to compress transformed latent representations. For example, Ballé *et al.* [24] model a latent representation $\boldsymbol{y}$ using the following entropy models:

1. A "fully factorized" *entropy bottleneck*. Each element $y_{c,i}$ within the c-th channel is modelled using a single channel-specific non-parametric probability distribution $p_c(y)$, which is fixed once training completes. That is, the elements $y_{c,1}, y_{c,2}, \ldots, y_{c,HW}$ within the c-th channel are assumed to be drawn from independent and identically distributed (i.i.d.) random variables with the probability distribution $p_c(y)$.

2. A *Gaussian conditional*. For each element y_i, the parameters μ_i and σ_i of its encoding distribution are computed, conditioned on the latent representation $\hat{\boldsymbol{z}} = \text{Quantize}[h_a(\boldsymbol{y})]$. In some cases, the modeling distribution is a mixture of K Gaussians — known as a Gaussian mixture model (GMM) — with parameters $\mu_i^{(k)}$ and $\sigma_i^{(k)}$ for each Gaussian, alongside an affine combination of weights $w_i^{(K)}$ that satisfy the constraint $\sum_k w_i^{(k)} = 1$.

Both of these have their advantages and disadvantages. Of these two, the entropy bottleneck is the most flexible at modeling arbitrary distributions. Indeed, it often does, modeling many channels via Laplacian-like distributions, which the latent representations often tend towards. However, because all elements within a given channel are modelled using the same distribution, it is quite poor at specializing and adapting to specific input distributions. Instead, it models a "conservative" distribution that tries to encompass all possible inputs, which is often suboptimal, as visualized in Fig. 2.1. This is sometimes known as the *amortization gap* [35], [36]. In contrast, the Gaussian conditional is better at adapting to a specific input, though it is limited to modeling Gaussian distributions. Furthermore, it must also pay an additional cost for transmitting the distribution parameters, though this is often worth the trade-off, e.g., a savings of roughly 40% for a 10% increase in rate, as shown by [24].

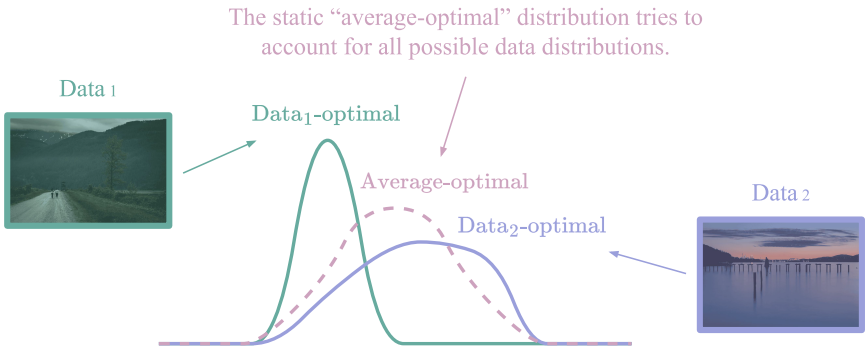


Figure 2.1: Visualization of the suboptimality of using a single static encoding distribution. This distribution tries to "average" (in an amortized information-theoretic sense) all the best possible data-specific distributions. However, the resulting distribution is less optimal than data-specific distributions.

In their "scale hyperprior" architecture, Ballé *et al.* [24] utilize an entropy bottleneck to compress the latent representation $\boldsymbol{z} = h_a(\boldsymbol{y})$. This is known as "side-information", and its compressed bitstream is transmitted before the bitstream for $\boldsymbol{y}$. The reconstructed latent $\hat{\boldsymbol{z}}$ is then used to determine

the parameters[1] $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of a Gaussian conditional distribution which is used to reconstruct $\hat{\boldsymbol{y}}$. More sophisticated entropy models, such as ELIC [4] use multiple successive applications of the Gaussian conditional entropy model, alongside an entropy bottleneck functioning as a "scale hyperprior". In many such models that utilize the scale hyperprior, the rate usage of the entropy bottleneck often comprises roughly 10% of the total rate. Evidently, the entropy bottleneck is a key component of many state-of-the-art (SOTA) image compression models, and so it is important to address the amortization gap suffered by the entropy bottleneck.

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

2.2 Related works

In Campos *et al.* [37], the latent y is refined by performing gradient descent to optimize it, while holding the trained model parameters fixed. This is effectively a form of Rate-Distortion Optimized Quantization (RDOQ), though the optimized loss $L = R_{\tilde{\boldsymbol{y}}} + \lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}})$ is not precisely aware of the exact effect of quantization.

Balcilar *et al.* [35] propose non-learned methods for the compression and transmission of the encoding distributions. For the factorized entropy model, they first measure a discrete histogram of the c-th latent channel. Then, they construct an approximation of the histogram using a K-Gaussian mixture model parameterized by $w^{(k)}$, $\mu^{(k)}$, and $\sigma^{(k)}$. The aforementioned parameters are compactly encoded into 8 bits each, and transmitted. That is, for a model using C activated channels, $(3K-1)\cdot C \cdot 8$ bits are required to transmit the parameters. (Typically, $K \in [1,3]$ is used, and $C \in [16, 192]$ for low-rate models.) The approximation model is then used as the encoding distribution $p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}}_c) = \sum_{k=1}^{K} w_k \, \hat{\mathcal{N}}(\hat{\boldsymbol{y}}_c;\, \mu_k,\, {\sigma_k}^2)$. For the Gaussian conditional entropy model, they instead transmit a factor $\beta^{(c)}$ that corrects the error in the center (and most likely) bin's probability. A key difference between our work and this prior work is that we formulate a learnable method for the compression of the encoding distribution.

Galpin *et al.* [16] introduce a lightweight non-learned context model for raster-scan ordered encoding, similar to Minnen *et al.* [2]. They also implement a simple raster-scan ordered greedy RDOQ method which repeatedly reruns the decoder on the quantized y hat to discretely optimize the loss. To reduce the computational cost of the RDOQ process, the loss estimation is restricted within the receptive field (a 9×9 latent window) and only delta y hat in $\{-1, 0, +1\}$ is explored. They then compare their proposed methods in conjunction with [35], [37] in various configurations on both the standard factorized model and its GDN to ReLU replacement model variant.

[1] In their initial publication, $\boldsymbol{\mu}$ is fixed to 0, though later results [2] show that it is better to predict $\boldsymbol{\mu}$ as well.

2.3 Proposed method

2.3.1 Compression of probability distributions

Let $\boldsymbol{x}$ be an input image, and $\boldsymbol{y} = g_a(\boldsymbol{x})$ be its transformed latent representation. Consider a fully-factorized entropy model [24], which models $\hat{\boldsymbol{y}} = \mathrm{Quantize}[\boldsymbol{y}]$ using a single channel-specific non-parametric probability distribution $\hat{\boldsymbol{p}}_j$ for each channel j. Since all elements within a given channel are modelled using the same distribution, in this setting, the "true" probability distribution $\boldsymbol{p}_j$ for the j-th channel of $\boldsymbol{y}$ can be determined exactly by computing the normalized histogram of $\hat{\boldsymbol{y}}_j$. The true distribution for the j-th channel is also the encoding distribution $\hat{\boldsymbol{p}}_j$ with the minimum rate cost for encoding a specific latent representation $\hat{\boldsymbol{y}}$ using the entropy model. That is, the ideal encoding distribution $\hat{\boldsymbol{p}}_j^*$ is given by

$$\hat{\boldsymbol{p}}_j^* = \arg\min_{\hat{\boldsymbol{p}}_j} \Big[H(\boldsymbol{p}_j) + D_{\mathrm{KL}}(\boldsymbol{p}_j \parallel \hat{\boldsymbol{p}}_j) \Big] \implies \hat{\boldsymbol{p}}_j^* = \boldsymbol{p}_j.$$

Unfortunately, using the true distribution $\boldsymbol{p}$ for encoding is infeasible, since the decoder does not have access to it. Instead, we propose a method that generates a reconstructed approximation $\hat{\boldsymbol{p}}$ that targets $\boldsymbol{p}$, subject to a rate trade-off. We discuss the specific loss function that is optimized in Section 2.3.4.

Fig. 2.2 visualizes the true ($\boldsymbol{p}$) and reconstructed ($\hat{\boldsymbol{p}}$) (via our proposed method) probability distributions for a fully-factorized entropy model. The left column shows various input images taken from the Kodak test dataset [1] — except for the "(Default)" image in the first row, which is an abstract "amortized" representation of all possible input images that may be randomly drawn from the training dataset distribution. The middle column and right columns visualize the negative log-likelihoods of the true and reconstructed probability distributions, respectively. Each probability distribution is visualized as a 2D color plot, where

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

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

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

Evidently, the default amortized dataset-optimized probability distribution is conservative and encompasses each of the image probability distributions. However, trying to cover all possible distributions with a single distribution leads to suboptimal compression performance since that distribution does not match each individual input distribution sufficiently well. In contrast, with our probability distribution compression method, the encoding distribution has adapted to the probability distributions for each image. Indeed, the true and reconstructed probability distributions look visually quite similar. This is particularly true for high-probability regions of the true distribution, which is also where accurate reconstruction is most important. For less probable regions, there is sometimes a larger mismatch between $\boldsymbol{p}$ and $\hat{\boldsymbol{p}}$, though this is less important, since incorrect prediction of low-probability regions has a smaller impact on the overall rate. This indicates that our method

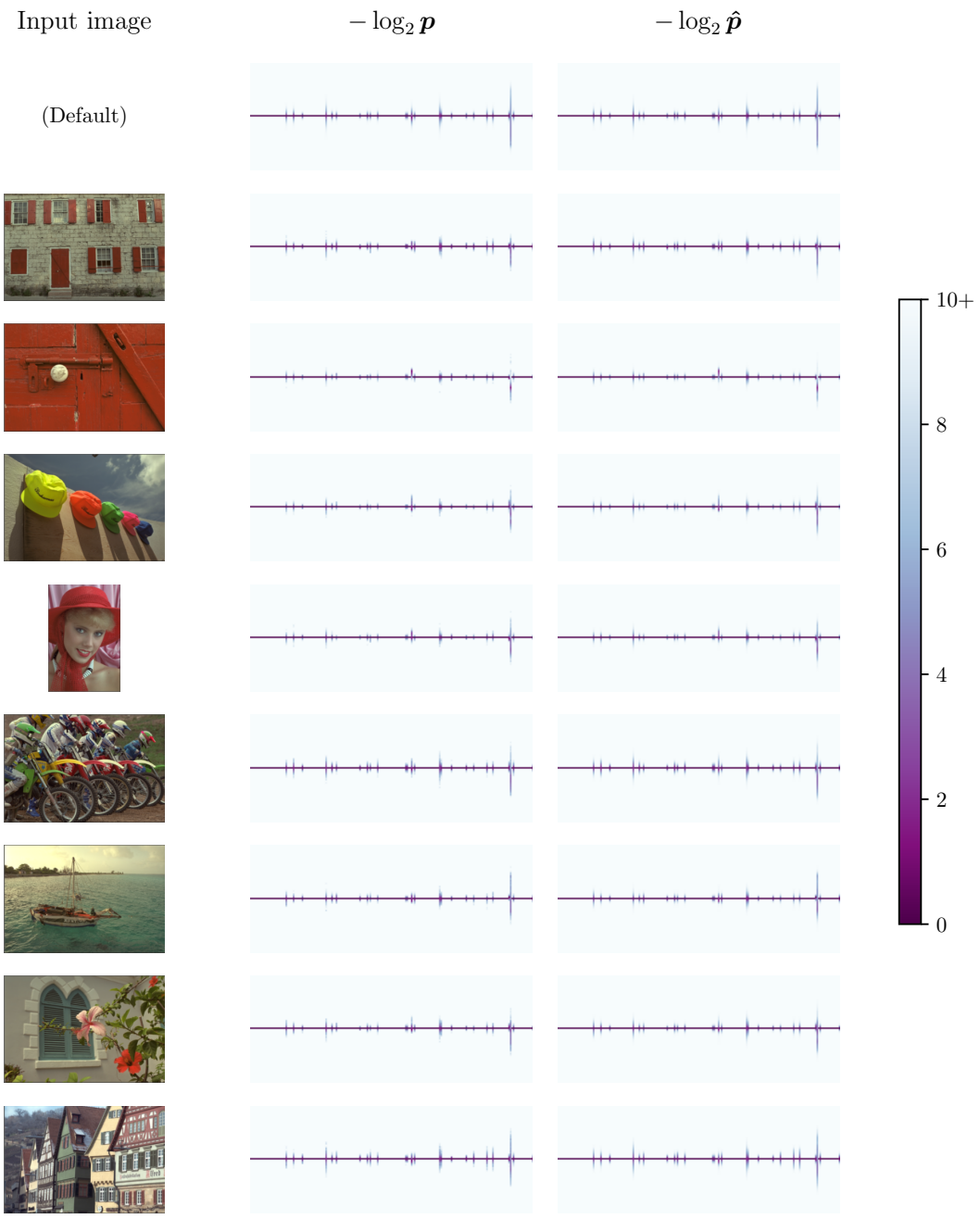


Figure 2.2: Different images from the Kodak dataset, and their corresponding negative log-likelihoods (in bits) of the true and reconstructed probability distributions.

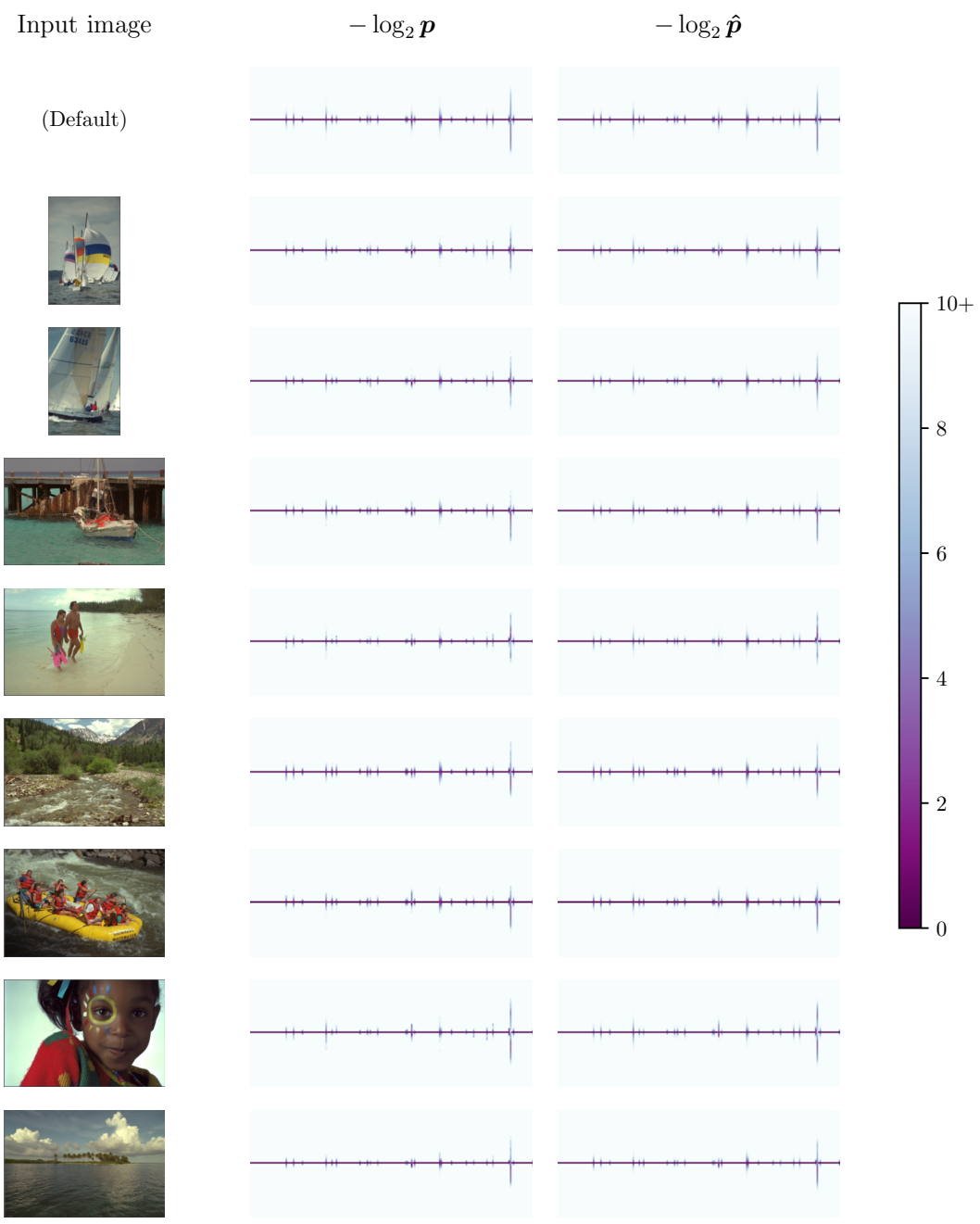


Figure 2.2: (continued)

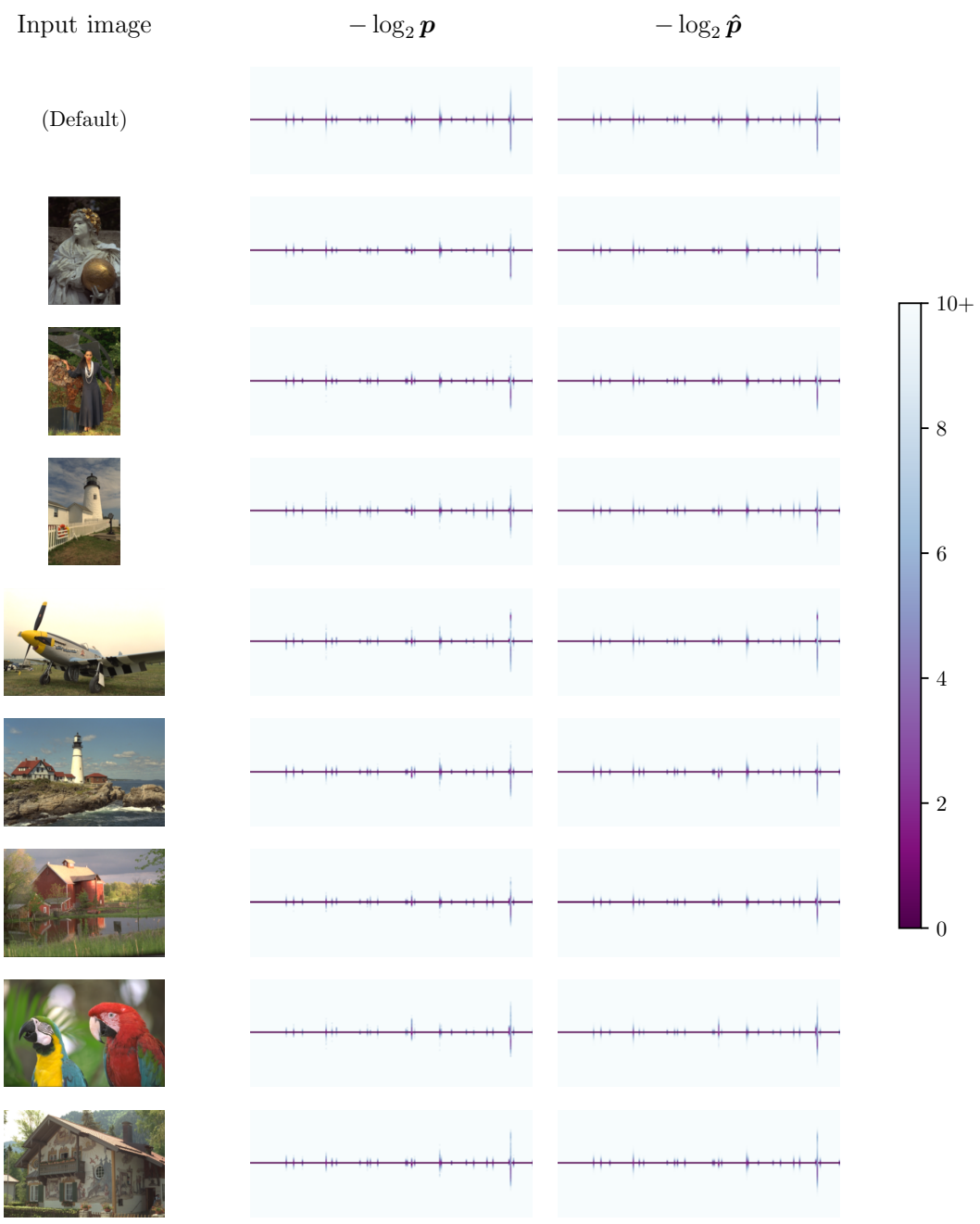


Figure 2.2: (continued)

is capable of adapting to the input distribution well, and in a way that is cognizant of the tradeoff between rates.

2.3.2 Architecture overview

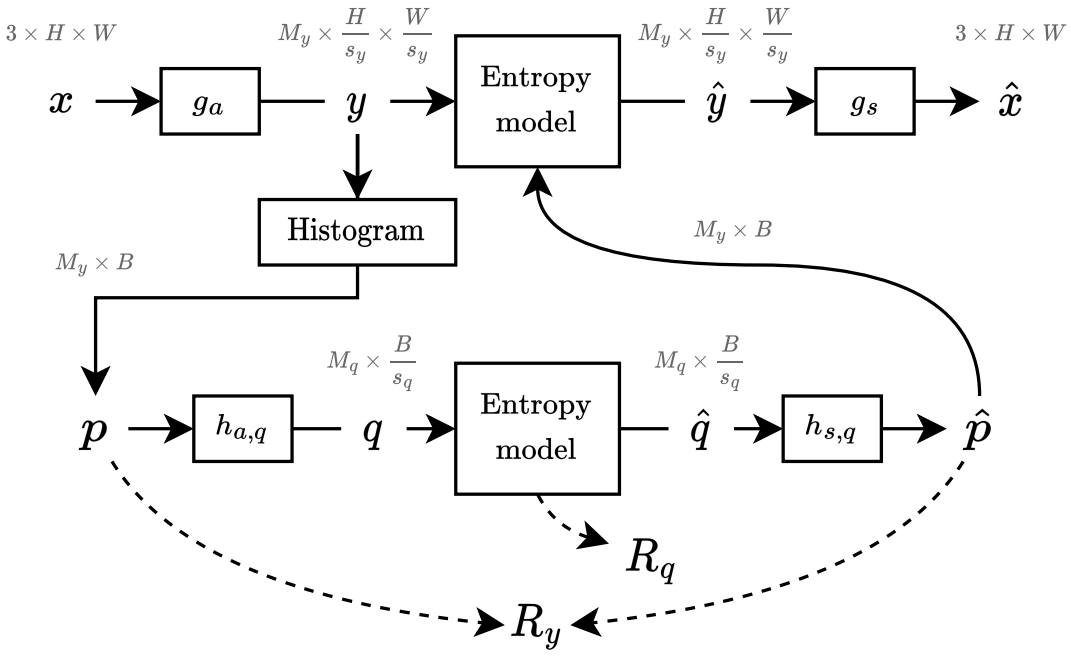


Figure 2.3: Adaptive probability distribution compression architecture.

Our proposed method applied to the traditional "fully-factorized" architecture [24] is visualized in Fig. 2.3. In this configuration, the latent representation $\boldsymbol{y}$ is first passed through the histogram layer described in Section 2.3.3, which estimates the target distribution $\boldsymbol{p}$. Then, $\boldsymbol{p}$ is passed through a distribution compression architecture consisting of the analysis transform $h_{a,q}$ which generates a latent representation $\boldsymbol{q}$, followed by an entropy model (e.g. entropy bottleneck), and then finally a synthesis transform $h_{s,q}$ which reconstructs the distribution $\hat{\boldsymbol{p}}$. The rate cost of encoding $\boldsymbol{q}$ is labelled R_q, and the rate cost of encoding $\boldsymbol{y}$ using $\hat{\boldsymbol{p}}$ is labelled R_y.

2.3.3 Histogram estimation

In the entropy bottleneck, the probability distribution for a given channel is fixed, regardless of the input. In order to adapt this distribution to the input, we must first estimate the input distribution more accurately. We do so by estimating the input distribution using a histogram. In order to ensure differentiability backwards through the histogram module, we use the method of kernel density estimation (KDE).

Let $[y_{\min}, y_{\max}]$ be a supporting domain that encompasses all values $y_1, y_2, \ldots, y_N$ in a specific channel. Let $\{b_1, b_2, \ldots, b_B\}$ denote the bin centers of the B-bin histogram, where $b_i = y_{\min} + i - 1$.

In the method of kernel density estimation, which is used to estimate the probability density function from a set of observations, a kernel function is placed centered at each observed sample, and those functions are summed to create the overall density function. To construct the histogram, we evaluate the kernel density estimate at each b_i. Or rather, we use an equivalent formulation where we instead place a kernel function at each bin center, as visualized in Fig. 2.4. Then, the probability mass in the i-th bin can be estimated as

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

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

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

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

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

$$K_{\text{hard}}(u) = \text{rect}(u) = \begin{cases} 1 & \text{if } |u| \leq \frac{1}{2} \\ 0 & \text{otherwise.} \end{cases}$$

We can then use the "straight-through" estimator (STE) [38] trick to get the benefits of both:

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

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

2.3.4 Loss function

Our model follows the variational formulation based on [24]; further details on the construction may be found in earlier works such as [39]–[41]. Briefly, for a variable $\boldsymbol{y}$, its counterparts $\tilde{\boldsymbol{y}}$ and $\hat{\boldsymbol{y}}$ denote the noise-modelled (for training) and quantized (for inference) versions of $\boldsymbol{y}$, respectively. Furthermore, the variational encoder $q_{\boldsymbol{\phi}}(\tilde{\boldsymbol{y}}, \tilde{\boldsymbol{q}} \mid \boldsymbol{x})$ with trainable parameters ϕ is a probabilistic[2] model of the joint distribution for $\tilde{\boldsymbol{y}}$ and $\tilde{\boldsymbol{q}}$, for a given input $\boldsymbol{x}$. The loss function that we seek to

[2]During training, the encoder is probabilistic in order to model quantization via uniform noise.

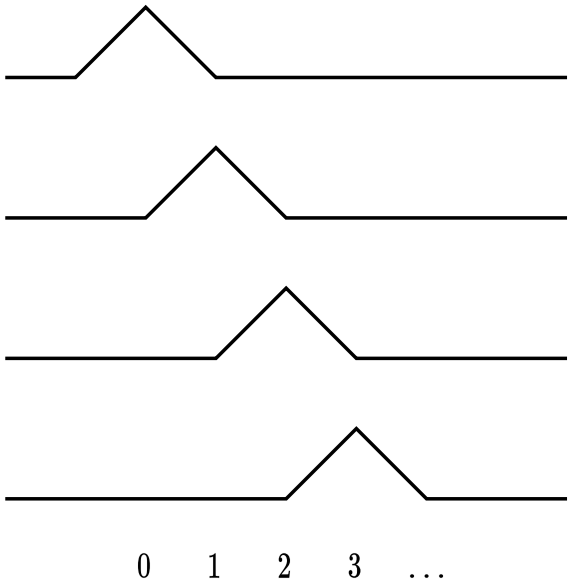


Figure 2.4: Visualization of triangular kernel functions for estimating the discrete probability mass histogram.

minimize over the input data distribution $p_{\boldsymbol{x}}(\boldsymbol{x})$ can then be expressed as

$$\begin{aligned}\mathcal{L} &= \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}(\boldsymbol{x})}\, \mathbb{E}_{\tilde{\boldsymbol{y}},\tilde{\boldsymbol{q}}\sim q_{\boldsymbol{\phi}}(\tilde{\boldsymbol{y}},\tilde{\boldsymbol{q}}|\boldsymbol{x})} \left[-\log p_{\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{q}}}(\tilde{\boldsymbol{y}} \mid \tilde{\boldsymbol{q}}) - \lambda_q \log p_{\tilde{\boldsymbol{q}}}(\tilde{\boldsymbol{q}}) + \lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \right] \\ &= \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}(\boldsymbol{x})} \left[R_{\tilde{\boldsymbol{y}}}(\boldsymbol{x}) + \lambda_q R_{\tilde{\boldsymbol{q}}}(\boldsymbol{x}) + \lambda_x D(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \right].\end{aligned}$$

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

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

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

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

Let $\boldsymbol{p}_j = (p_{j1}, \ldots, p_{jB})$ refer to the discrete B-bin probability distribution used to encode the j-th channel of $\tilde{\boldsymbol{y}}$. Then concretely, the rate cost of $\tilde{\boldsymbol{y}}$ may be measured by the cross-entropy between

the true and reconstructed[3] discretized distributions,

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

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

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

2.3.5 Optimization

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

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

where s is the downscale factor (e.g., $s = 2^4 = 16$ for a model with four downscaling strides of length 2), and H and W are the height and width of the input image $\boldsymbol{x}$, respectively. And so,

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

since

$$\frac{\partial}{\partial u} \max\{0, 1 - |u|\} = -H(1 - |u|) \cdot [2 \cdot H(u) - 1].$$

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

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

$$\begin{aligned}\frac{\partial \hat{p}_i}{\partial y_k} &= \sum_l \frac{\partial \hat{p}_i}{\partial p_l}\frac{\partial p_l}{\partial y_k} \\ &\approx \sum_l \delta_{il}\frac{\partial p_l}{\partial y_k} \quad \text{assuming } \frac{\partial \hat{p}_i}{\partial p_l} \approx \delta_{il} \\ &= \frac{\partial p_i}{\partial y_k},\end{aligned}$$

where we have assumed that the dominant term in the sum is the term where $l = i$.

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

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

Then, we may compute the derivative as follows:

$$\begin{aligned}\frac{\partial R_y}{\partial y_k} &= \frac{-HW}{s^2}\sum_i \frac{\partial}{\partial y_k}\left[p_i \log \hat{p}_i\right] \\ &= \frac{-HW}{s^2}\sum_i \left[p_i \frac{\partial}{\partial y_k}\left[\log \hat{p}_i\right] + \frac{\partial p_i}{\partial y_k}\log \hat{p}_i\right] \\ &= \frac{-HW}{s^2}\sum_i \left[\frac{p_i}{\hat{p}_i}\frac{\partial \hat{p}_i}{\partial y_k} + \frac{\partial p_i}{\partial y_k}\log \hat{p}_i\right] \\ &\approx \frac{-HW}{s^2}\sum_i \left[\frac{p_i}{\hat{p}_i}\frac{\partial p_i}{\partial y_k} + \frac{\partial p_i}{\partial y_k}\log \hat{p}_i\right] \quad \text{assuming } \frac{\partial \hat{p}_i}{\partial y_k} \approx \frac{\partial p_i}{\partial y_k} \\ &= \frac{-HW}{s^2}\sum_i \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i\right]\frac{\partial p_i}{\partial y_k} \\ &= \frac{-1}{\Delta b}\sum_i \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i\right] \cdot H\left(1 - \left|\frac{y_k - b_i}{\Delta b}\right|\right) \cdot \left(1 - 2 \cdot H\left(\frac{y_k - b_i}{\Delta b}\right)\right) \\ &= \frac{-1}{\Delta b}\left(\left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i\right]_{i=\lceil y_k - y_{\min}\rceil + 1} - \left[\frac{p_i}{\hat{p}_i} + \log \hat{p}_i\right]_{i=\lfloor y_k - y_{\min}\rfloor + 1}\right) \\ &\approx \frac{-1}{\Delta b}\left(\log \hat{p}_{\lceil y_k - y_{\min}\rceil + 1} - \log \hat{p}_{\lfloor y_k - y_{\min}\rfloor + 1}\right) \quad \text{assuming } \hat{p}_i \approx p_i.\end{aligned} \tag{2.1}$$

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

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

where $\alpha = (y_k - b_{\lfloor y_k - y_{\min}\rfloor + 1})/\Delta b = 1 - (b_{\lceil y_k - y_{\min}\rceil + 1} - y_k)/\Delta b$. The linear interpolation may be justified by the fact that $\hat{y}_k$ inhabits only a single bin at a time. The probability of $\hat{y}_k$ inhabiting

the left bin is α, and the probability of inhabiting the right bin is $1-\alpha$. This aligns perfectly with Shannon's measure of entropy, which is the expected value of the code length. (Isn't math elegant?)

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

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

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

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

We may then compute its derivative as follows:

$$\begin{aligned}
\frac{\partial R_y}{\partial y_k} &= -\sum_i \frac{\partial}{\partial y_k}\left[\log p(y_i)\right] \\
&= -\frac{\partial}{\partial y_k}\left[\log p(y_k)\right] \\
&= -\frac{1}{p(y_k)} \cdot \frac{\partial}{\partial y_k}\left[p(y_k)\right] \\
&= -\frac{1}{p(y_k)} \cdot \left[f\left(y_k + \frac{1}{2}\right) - f\left(y_k - \frac{1}{2}\right)\right].
\end{aligned} \tag{2.2}$$

Interestingly, whereas the derivative for the proposed method (under the assumption that $\hat{p}_i \approx p_i$) computed in (2.1) contains a difference between the "right" and "left" log-likelihoods, the derivative for the standard method computed in (2.2) contains a difference between the "right" and "left" evaluations of the probability density function.

2.4 Experimental setup

2.4.1 Architecture details

As shown in Fig. 2.5, our $h_{a,q}$ and $h_{s,q}$ are implemented using a simple five-layer convolutional neural network. In between each of the convolutional layers shown is a ReLU activation function, as well as a channel shuffle operation, as is done in ShuffleNet [42]. There are two strides of length 2, resulting in a total downscaling factor of $s_q = 4$. For all our models, we set $K = 15$ to control the kernel sizes, and $G = 8$ to control the number of channel groups. Furthermore, we set $(N_q, M_q) = (32, 16)$ for low-resolution models, and $(N_q, M_q) = (64, 32)$ for high-resolution models. The number of bins B is set between 128 and 1024 across different models. We have elected to use an entropy bottleneck design for simplicity, though one can likely further improve the compression performance of the probability distribution compression architecture by using more powerful entropy

modeling techniques (e.g., a scale hyperprior). Since λ_q is a parameter that depends on the ratio between the target and trained-upon input dimensions, it may also be advisable to train a single model which that supports λ-rate control strategies (e.g., G-VAE [43], [44] and QVRF [45]) for both latent representations y, q. However, we have not yet explored this possibility.

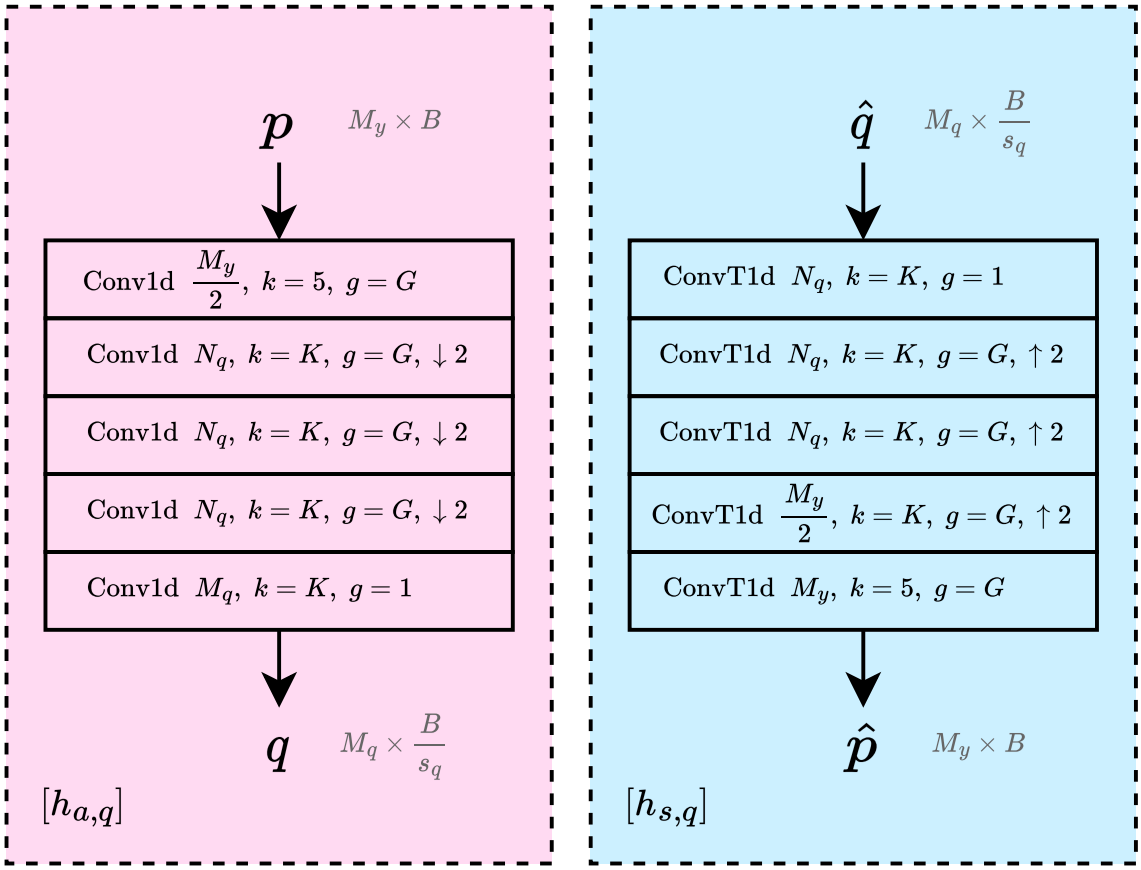


Figure 2.5: Architecture layer diagram for $h_{a,q}$ and $h_{s,q}$ transforms. k denotes kernel size, g denotes number of channel groups, and $\downarrow, \uparrow$ denote stride.

2.4.2 Training details

Our models are trained on 256×256 image patches from the Vimeo-90K triplet dataset [46]. A training batch size of 16 was used, along with the Adam optimizer [47] with an initial learning rate of 10^{-4} that was decayed by a factor of 0.1 whenever the validation loss plateaued. Specifically, we loaded the weights of the pretrained models from CompressAI [25], which were also trained using the same setup as above. We replaced the static distributions of the `EntropyBottleneck` module with the dynamically generated adaptive distributions from our proposed probability distribution compression module. Then, we froze the weights for g_a and g_s, and trained only the weights for our probability distribution compression model (i.e., for $h_{a,q}$ and $h_{s,q}$, and the entropy model for q).

Finally, we evaluated our models on the standard Kodak test dataset [1] containing 24 images of size 768×512. (Thus, we set $\lambda_q = \frac{1}{6}$ during training.)

2.5 Experimental results

Fig. 2.6 shows the rate-distortion (RD) curves comparing a given base model against the same model enhanced with our proposed probability distribution compression module.

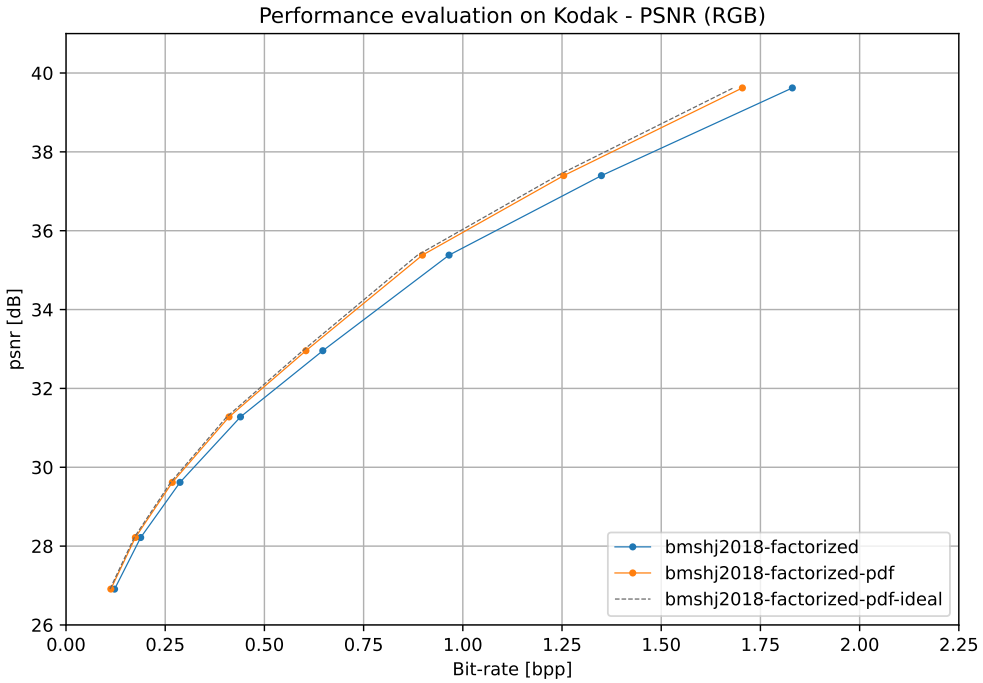


Figure 2.6: RD curves for the Kodak dataset.

In Table 2.1, we compare the rate savings of our proposed method against the base model at various quality levels. Additionally, we also list the maximum rate savings that can be theoretically achieved by perfectly eliminating the amortization gap between the true and reconstructed distributions, i.e., by using the true distribution directly as the encoding distribution with zero additional transmission cost. As shown, our approach achieves a 6.95% BD-rate reduction over the base model, in comparison to a maximum possible reduction of 8.33% BD-rate reduction achievable by perfectly eliminating the amortization gap.

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

Table 2.1: Potential and achieved rate savings for the bmshj2018-factorized model [24] when equipped with our proposed distribution compression method.

Quality	Original (bpp)	Potential gain (bpp)	Potential BD-rate	Our (bpp)	Our gain (bpp)	Our BD-rate
1	0.122	-0.012	-9.45	0.113	-0.009	-7.66
2	0.188	-0.016	-8.66	0.175	-0.013	-7.13
3	0.287	-0.024	-8.20	0.268	-0.019	-6.76
4	0.440	-0.035	-7.90	0.411	-0.029	-6.57
5	0.647	-0.051	-7.83	0.605	-0.043	-6.57
6	0.965	-0.080	-8.25	0.898	-0.067	-6.95
7	1.349	-0.111	-8.24	1.254	-0.095	-7.04
8	1.830	-0.149	-8.14	1.705	-0.126	-6.88
mean			-8.33			-6.95

In Table 2.2, we compare the rate savings for various base models equipped with a distribution compression method. "Ratio" is the fraction of the total rate occupied by the original unmodified model. "Gap" is the maximum potential rate gain with perfect distribution reconstruction and zero additional transmission cost. "Gain" is the actual rate gain achieved by the given model. All models are evaluated on the Kodak dataset [1].

Table 2.2: Comparison of rate savings for various models.

Model	Quality	Factorized			Total
		Ratio (%)	Gap (%)	Gain (%)	Gain (%)
bmshj2018-factorized [24] + Balcilar2022 [35]	1	100	-9.45	-6.79	-6.79
bmshj2018-factorized [24] + ours	1	100	-9.45	-7.66	-7.66
bmshj2018-factorized [24] + ours	*	100	-8.33	-6.95	-6.95

Table 2.3 reports the number of trainable parameters and the number of multiply-accumulate operations (MACs) per pixel for various model configurations. The results are calculated assuming an input image size of 768×512, and a latent representation size of 48×32. As shown, our model requires far fewer parameters and MACs/pixel than the comparable scale hyperprior [24] model. In particular, for comparable configurations, our method's $h_{a,q}$ and $h_{s,q}$ transforms require 96–97% fewer parameters and 96–99% fewer MACs/pixel than the scale hyperprior's h_a and h_s transforms.

Table 2.3: Trainable parameter counts and number of multiply-accumulate operations (MACs) per pixel.

Model configuration	Params	MACs/pixel	Params	MACs/pixel
(M_y, N_q, M_q, K, G, B)	$h_{a,q}$		$h_{s,q}$	
Ours $(192, 32, 16, 15, 8, 256)$	0.029M	10	0.029M	10
Ours $(320, 64, 32, 15, 8, 1024)$	0.097M	126	0.097M	126
(N, M)	h_a		h_s	
bmshj2018-hyperprior [24] $(128, 192)$	1.040M	1364	1.040M	1364
bmshj2018-hyperprior [24] $(192, 320)$	2.396M	3285	2.396M	3285

2.6 Conclusion

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

2.6.1 Future work

A few steps remain towards making probability distribution corrective methods viable parts of more advanced entropy models such as [3], [4], [24]. These include the following:

- The proposed adaptive entropy bottleneck needs to be formulated in such a way that it can be trained fully end-to-end. When training a pretrained base model equipped with our adaptive distribution compression module, we found that unfreezing the pretrained transform weights led to a degradation in RD performance. In contrast, keeping the transform frozen led to improvements in RD performance. This suggests that the rate-minimizing gradients flowing backwards into the transform through our adaptive distribution compression module are not necessarily well-formulated.

- Application of our adaptive distribution method to the Gaussian conditional component of entropy models. This component (along with the entropy bottleneck) is used to construct entropy modeling methods such as those used in [3], [4], [24]. Most SOTA learned image compression models predict the parameters of the Gaussian or Gaussian mixture encoding distributions using various sophisticated methods. However, in all such models, the focus has been on optimizing the location and scale of the fixed Gaussian-shaped distributions. Thus, there are potential rate improvements to be made by adapting the shapes of the encoding

distributions to shapes that more closely match the data distribution. (One such effort is Gaussian-Laplacian-Logistic Mixture Model (GLLMM) proposed in [48].) One way to directly apply our adaptive distribution method to Gaussian distributions would be to discretize them and then run the result through a distribution compression model.

Chapter 3

Point cloud compression for classification

Deep learning is increasingly being used to perform machine vision tasks such as classification, object detection, and segmentation on 3D point cloud data. However, deep learning inference is computationally expensive. The limited computational capabilities of end devices thus necessitate a codec for transmitting point cloud data over the network for server-side processing. Such a codec must be lightweight and capable of achieving high compression ratios without sacrificing accuracy. Motivated by this, we present a novel point cloud codec that is highly specialized for the machine task of classification. Our codec, based on PointNet, achieves a significantly better rate-accuracy trade-off in comparison to alternative methods. In particular, it achieves a 94% reduction in BD-bitrate over non-specialized codecs on the ModelNet40 dataset. For low-resource end devices, we also propose two lightweight configurations of our encoder that achieve similar BD-bitrate reductions of 93% and 92% with 3% and 5% drops in top-1 accuracy, while consuming only 0.470 and 0.048 encoder-side kMACs/point, respectively. Our codec demonstrates the potential of specialized codecs for machine analysis of point clouds, and provides a basis for extension to more complex tasks and datasets in the future. This chapter has been presented as [28].

3.1 Introduction

Point clouds are used to represent 3D visual data in many applications, including autonomous driving, robotics, and augmented reality. Recent advances in deep learning have led to the development of deep learning-based methods for machine vision tasks on point cloud data. Common tasks include classification, segmentation, object detection, and object tracking. However, current deep learning-based methods often require significant computational resources, which impose hardware requirements. Such significant requirements may not be physically or economically feasible for end devices.

One approach to address the issue of insufficient end-device computational resources is to transmit the point cloud and other sensor data to a server for processing. However, this introduces its own challenges, including the effects of network availability, latency, and bandwidth. In order to reduce network requirements, the end device may compress the point cloud data before transmission.

However, network capabilities vary depending on various factors, including end-device location and network congestion. This means that sufficient bandwidth may still not be available to transmit the point cloud data. A hybrid strategy is to perform part of the machine task on the end device itself. This can reduce the amount of data that needs to be transmitted to the server for further processing, without exceeding the computational budget of the end device [49]. This enhances the robustness of the end device to varying network conditions, while potentially improving overall system latency and adaptability [50].

We propose a novel learned point cloud codec for classification. Our learned codec takes a point cloud as input, and outputs a highly compressed representation that is intended solely for machine analysis. To our knowledge, this is the first point cloud codec specialized for machine analysis. Existing codecs for point clouds are designed to reconstruct point clouds intended for human viewing. This means that a significant amount of bits is wasted on encoding information that is not strictly relevant to machine analysis. By partially processing the point cloud before compression, our codec is able to achieve significantly better compression performance, without compromising task accuracy.

In this chapter, we present our task-specialized codec architecture in full and lightweight configurations. We evaluate its rate-accuracy (RA) performance on the ModelNet40 dataset [51]. We also investigate how the number of input points (and thus reduced computation) affects the RA performance of our proposed codec. Furthermore, we compare our proposed codec's performance against alternative non-specialized methods. Our code for training and evaluation is available online[1].

3.2 Related work

Point cloud classification models can be organized into groups based on the type of input data they accept. Models such as VoxNet [52] take as input point clouds that have been preprocessed into a voxel grid. Unfortunately, these methods often use 3D convolutions, which require a significant amount of computational resources. Additionally, since most voxels are usually empty, these methods arguably waste a significant amount of computation on empty space. Furthermore, the voxel grid representation is not very compact, and thus requires a significant amount of memory for higher spatial resolutions (e.g., a 32-bit tensor of shape $1024 \times 1024 \times 1024$ occupies 32 GB). Models such as OctNet [53] take octrees as input. Octrees offer a more compact representation of the voxelized point cloud by encoding the node occupancy in bitstrings. Large unoccupied regions of space may be represented via a single "0" node in an octree. Point-based models such as PointNet [54] and PointNet++ [55] directly accept raw point lists $(x_1, x_2, \ldots, x_P)$, where $x_i \in \mathbb{R}^3$ represents a point in a 3D space and P is the number of points. Some challenges faced with this input format include designing order-invariant models (due to the lack of a worthwhile canonical ordering of points), as well as in devising operations capable of using the metric structure induced by point locality. Despite

[1] https://github.com/multimedialabsfu/learned-point-cloud-compression-for-classification

the challenges, point-based models are able to achieve surprisingly competitive accuracy, and offer the most promise in terms of minimizing computational requirements.

PointNet [54], which our proposed architecture is based on, can be represented as an input permutation-invariant function:

$$f(x_1, \ldots, x_n) = (\gamma \circ \pi)(h(x_1), \ldots, h(x_n)),$$

where h is applied to each point x_i individually, π is a simple permutation-invariant function, and γ may be any function. In the original PointNet architecture, h is a weight-shared MLP, π is a max pooling function applied across the point dimension, and γ is an MLP.

In the related field of learned image compression, Ballé *et al.* [24] proposed a variational autoencoder (VAE) architecture for image compression. Here, the model transforms the input $\boldsymbol{x}$ into a latent representation $\boldsymbol{y}$, which is then quantized into $\hat{\boldsymbol{y}}$ and losslessly compressed using a learned entropy model. The codec is trained end-to-end using the loss function

$$\mathcal{L} = R + \lambda \cdot D(\boldsymbol{x}, \hat{\boldsymbol{x}}), \tag{3.1}$$

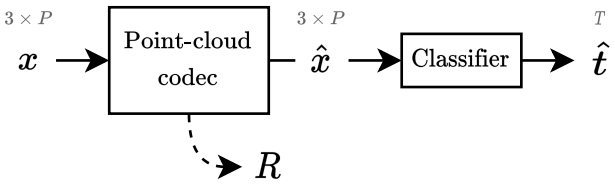
where $D(\boldsymbol{x}, \hat{\boldsymbol{x}})$ is the distortion measure between input $\boldsymbol{x}$ and decoded $\hat{\boldsymbol{x}}$, and R is the estimate of the entropy of $\hat{\boldsymbol{y}}$. One simple entropy model, known in the literature as an *entropy bottleneck*, makes a "factorized" prior assumption — that each element within a latent channel is independently and identically distributed. It models a monotonically increasing non-parametric cumulative distribution function using a differentiable MLP. This mechanism has also shown effectiveness in learned codecs in other fields, including learned point cloud compression (PCC), and has been incorporated by a variety of works including [7]–[11].

We have also based our work on ideas introduced for machine tasks on images. Early works demonstrated the use of standard codecs in compressing the latent features [56]. More recently, approaches such as Video Coding for Machines (VCM) [20] and Coding for Machines (CfM) have gained traction in the research community. For instance, works such as [22], [57] demonstrate the potential bitrate savings of scalable image compression in a multi-task scenario for a machine vision task (e.g., facial landmark detection, or object detection) and human vision (e.g., image reconstruction). In this work, we focus solely on a single machine vision task applied to point cloud data.

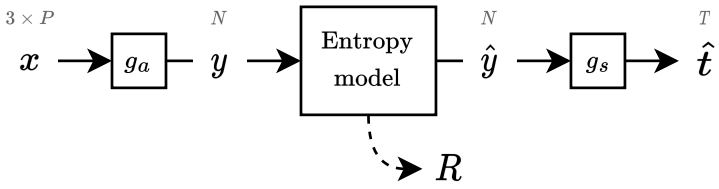
3.3 Proposed codec

3.3.1 Input compression

In Fig. 3.1a, we show an abstract representation of an input codec, similar to the "chain" configuration explored by [58] for end-to-end image compression for machines. In this codec, the input point cloud $\boldsymbol{x}$ is encoded directly, without any intermediate feature extraction. On the decoder side, the point



(a) input compression



(b) proposed

Figure 3.1: High-level comparison of codec architectures.

cloud is then reconstructed as $\hat{\boldsymbol{x}}$. Any point cloud compression codec can be used for this purpose, including standard non-learned codecs such as G-PCC [59]. Finally, the reconstructed point cloud $\hat{\boldsymbol{x}}$ is fed into a classification model (e.g., PointNet) in order to obtain the class prediction $\hat{\boldsymbol{t}}$. This approach provides a baseline for comparison with our proposed codec.

3.3.2 Motivation for the proposed codec

An efficient task-specific codec can be developed using the concept of Information Bottleneck (IB) [60]:

$$\min_{p(\hat{\boldsymbol{y}}|\boldsymbol{x})} \quad I(\boldsymbol{x};\hat{\boldsymbol{y}}) - \beta \cdot I(\hat{\boldsymbol{y}};\hat{\boldsymbol{t}}), \tag{3.2}$$

where $I(\cdot;\cdot)$ is the mutual information [61], $p(\hat{\boldsymbol{y}} \mid \boldsymbol{x})$ is the mapping from the input point cloud $\boldsymbol{x}$ to the latent representation $\hat{\boldsymbol{y}}$, and $\beta > 0$ is the IB Lagrange multiplier [60]. We can think of $p(\hat{\boldsymbol{y}} \mid \boldsymbol{x})$ as feature extraction followed by quantization. Hence, $\hat{\boldsymbol{y}}$ is fully determined whenever $\boldsymbol{x}$ is given, so $H(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) = 0$, where $H(\cdot \mid \cdot)$ is the conditional entropy [61]. Therefore, $I(\boldsymbol{x};\hat{\boldsymbol{y}}) = H(\hat{\boldsymbol{y}}) - H(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) = H(\hat{\boldsymbol{y}})$.

Furthermore, since decreasing $-\beta \cdot I(\hat{\boldsymbol{y}};\hat{\boldsymbol{t}})$ would improve the accuracy of the task, we can use $\lambda \cdot D(\boldsymbol{t},\hat{\boldsymbol{t}})$ as a proxy for $-\beta \cdot I(\hat{\boldsymbol{y}};\hat{\boldsymbol{t}})$, where $\boldsymbol{t}$ is the ground-truth label, $\hat{\boldsymbol{t}}$ is the label produced using the compressed latent representation, and $D(\boldsymbol{t},\hat{\boldsymbol{t}})$ is a distortion measure. Therefore, in our case, the IB (3.2) becomes:

$$\min_{p(\hat{\boldsymbol{y}}|\boldsymbol{x})} \quad H(\hat{\boldsymbol{y}}) + \lambda \cdot D(\boldsymbol{t},\hat{\boldsymbol{t}}). \tag{3.3}$$

It is clear that the form of IB in (3.3) is analogous to the loss function (3.1). We make use of this analogy to develop the proposed codec, which is described next.

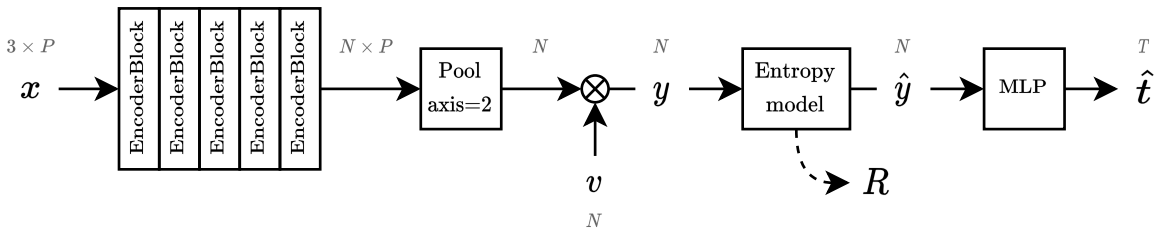


Figure 3.2: Proposed codec architecture.

3.3.3 Proposed architecture

In Fig. 3.1b, we show a high-level representation of our proposed codec architecture. Following the terminology of [24], we refer to g_a as the *analysis transform*, and g_s as the *synthesis transform*. In this architecture, the input point cloud $\boldsymbol{x}$ is first encoded into a latent representation $\boldsymbol{y} = g_a(\boldsymbol{x})$, which is then quantized as $\hat{\boldsymbol{y}} = Q(\boldsymbol{y})$, and then losslessly compressed using a learned entropy model. Therefore, $p(\hat{\boldsymbol{y}} \mid \boldsymbol{x})$ from the IB is $Q \circ g_a$. Then, the reconstructed latent representation $\hat{\boldsymbol{y}}$ is used to predict the classes $\hat{\boldsymbol{t}} = g_s(\hat{\boldsymbol{y}})$.

Our proposed architecture, visualized in Fig. 3.2, is based on the PointNet [54] classification model. The input 3D point cloud containing P points is represented as a matrix $\boldsymbol{x} \in \mathbb{R}^{3\times P}$. The input $\boldsymbol{x}$ is fed into a sequence of encoder blocks. Each encoder block consists of a convolutional layer, a batch normalization layer, and a ReLU activation. As described in [54], the early encoder blocks must be applied to each input point independently and identically, in order to avoid learning input permutation-specific dependencies. Therefore, we use pointwise convolutional layers with kernel size 1, which are exactly equivalent to the "shared MLP" described in [54].

Following the sequence of encoder blocks, a max pooling operation is applied along the point dimension to generate an input permutation-invariant feature vector of length N. The resulting feature vector is then multiplied element-wise by a trainable gain vector $v \in \mathbb{R}^N$, which is initialized to $[1, 1, \ldots, 1]$ before training. Due to the batch normalization layers, the resulting feature vector has small values. Thus, in order to improve training stability and the rate of convergence, the feature vector is multiplied by a constant scalar value of 10. The resulting vector is the output of the encoder-side analysis transform, which we label as $\boldsymbol{y}$.

Then, we quantize $\boldsymbol{y}$ via uniform quantization (specifically, integer rounding) to obtain $\hat{\boldsymbol{y}}$. During training, uniform quantization is simulated using additive uniform noise $\mathcal{U}(-0.5, 0.5)$. The quantized vector $\hat{\boldsymbol{y}}$ is then losslessly encoded using a fully-factorized learned entropy model introduced in [24].

On the decoder side, the decoded vector $\hat{\boldsymbol{y}}$ is fed into an MLP consisting of fully-connected layers interleaved with ReLU activations and batch normalizations. Before the last fully-connected layer, we use a dropout layer that randomly sets 30% of its inputs to zero. The output of the MLP is a vector of logits $\hat{\boldsymbol{t}} \in \mathbb{R}^T$, where T is the number of classes.

We provide "full", "lite", and "micro" configurations of the proposed codec. For each configuration, Table 3.1 lists the number of layer output channel sizes along with the estimated

Table 3.1: Layer sizes and MAC counts for various proposed codec types.

Proposed codec	Encoder layer sizes	Decoder layer sizes	Encoder MAC/pt	Decoder MAC
full	64 64 64 128 1024	512 256 40	150k	670k
lite	8 8 16 16/2 32/4	512 256 40	0.47k	160k
micro	16	512 256 40	0.048k	150k

MAC (multiply-accumulate) counts[2]. Group convolutional layers are specified in the format "output_size/group". In contrast to [54], we do not use any input or feature transformations in order to simplify the architectures for clearer analysis, as well as to reduce the computational requirements.

3.3.4 Lightweight and micro architectures

In addition to our "full" proposed codec, we also provide a lightweight configuration, which we denote as "lite". In this architecture, the encoder-side layers contain fewer output channels. To further reduce encoder-side computational costs, they also use group convolutions with channel shuffles in between, as is done in ShuffleNet [42]. After training, the gain and batch normalization layers may be fused into the preceding convolutional layer. The "lite" architecture strikes a balance between RA performance and encoder complexity. In fact, the encoder-side transform requires only 0.47k MACs/point, which is significantly less than the 150k MACs/point required by the "full" architecture encoder. For input point clouds consisting of $P = 256$ points, the total MAC count for the "lite" codec is 120k, which is below the corresponding decoder-side MAC count of 160k.

Additionally, we examine a "micro" architecture, whose encoder-side transform consists of only a single encoder block with 16 output channels, and a max pooling operation. This codec is useful for analysis and comparison — and yet, it is also capable of surprisingly competitive RA performance.

3.4 Experiments

Our models were trained on the ModelNet40 [51] dataset, which consists of 12311 different 3D object models organized into 40 classes. We used an Adam optimizer with a learning rate of 0.001. Our code was written using the PyTorch, CompressAI [25], and CompressAI Trainer [31] libraries.

The loss function that is minimized during training is:

$$\mathcal{L} = R + \lambda \cdot D(\boldsymbol{t}, \hat{\boldsymbol{t}}),$$

where the rate $R = -\log p_{\hat{\boldsymbol{y}}}(\hat{\boldsymbol{y}})$ is the log of the likelihoods outputted by the entropy model, and the distortion $D(\boldsymbol{t}, \hat{\boldsymbol{t}})$ is the cross-entropy between the one-hot encoded labels $\boldsymbol{t}$ and the softmax of the

[2]One MAC operation may be considered equivalent to a FLOP (floating-point operation) on most hardware.

model's prediction $\hat{\boldsymbol{t}}$. We trained different models to operate at different rate points by varying the hyperparameter $\lambda \in [10, 16000]$.

3.4.1 Proposed codec

For each of the proposed codec architectures, we trained a variation of each codec to accept an input point cloud containing $P \in \mathcal{P}$ points. We trained eight such variations for each of the values in the set $\mathcal{P} = \{8, 16, 32, 64, 128, 256, 512, 1024\}$. Although each codec is capable of handling a variable number of points, training a separate model for each P guarantees that each codec is well-optimized for each rate-accuracy trade-off.

3.4.2 Input compression codec

We compare our proposed codec against an "input compression" codec architecture. For this codec, the encoder may be taken from any point cloud codec. We have tested multiple codecs, including TMC13 [62] v14 (an implementation of the G-PCC v2 [59] standard), OctAttention [10], and Draco [63]. On the decoder side is the corresponding point cloud decoder, followed by a PointNet classification model. We trained a PointNet model (without the input and feature transforms) for each $P \in \mathcal{P}$.

We generated eight separate datasets of P-point point clouds, where each point cloud was uniformly subsampled from the test dataset. Then, we compressed and decompressed each point cloud from each P-point dataset at various compression ratios. The compression ratio can be effectively controlled by varying the amount of input scaling, which we denote by S. (The input scaling parameter is directly proportional to the number of bins used during uniform quantization of the input points.) We varied S over the set $\mathcal{S} = \{1, 2, 4, \ldots, 256\}$ and P over $\mathcal{P}$ to produce $|\mathcal{P}| \cdot |\mathcal{S}|$ distinct datasets. We evaluated each dataset associated with the pair $(P, S) \in \mathcal{P} \times \mathcal{S}$ on the correspondingly trained PointNet models to obtain a set of rate-accuracy points. Finally, we took the Pareto front of this set to obtain the best rate-accuracy curve achieved by the tested input compression codec.

3.4.3 Reconstruction

In order to visually assess the contents of the machine task-specialized bitstream, we trained a point cloud reconstruction network on top of our trained models. This auxiliary network was trained to minimize the loss function $\mathcal{L} = D(\boldsymbol{x}, \hat{\boldsymbol{x}})$, where we used Chamfer distance for D.

We also identify a critical point set for a fixed point cloud. A critical point set is a minimal set of points which generate the exact same latent $\boldsymbol{y}$, and correspondingly, the same bitstream. Formally, for any given point cloud $\boldsymbol{x}$, let $\boldsymbol{x}_C \subseteq \boldsymbol{x}$ denote a (not necessarily unique) critical point set. Then, $g_a(\boldsymbol{x}_C) = g_a(\boldsymbol{x}) = \boldsymbol{y}$, and there is uniquely one valid critical point set $(\boldsymbol{x}_C)_C$ for $\boldsymbol{x}_C$, and it is itself. Since g_a contains a max pooling operation, the critical point set is not theoretically unique; however, in practice, it is rare for there to be more than one critical point set. A critical point set may

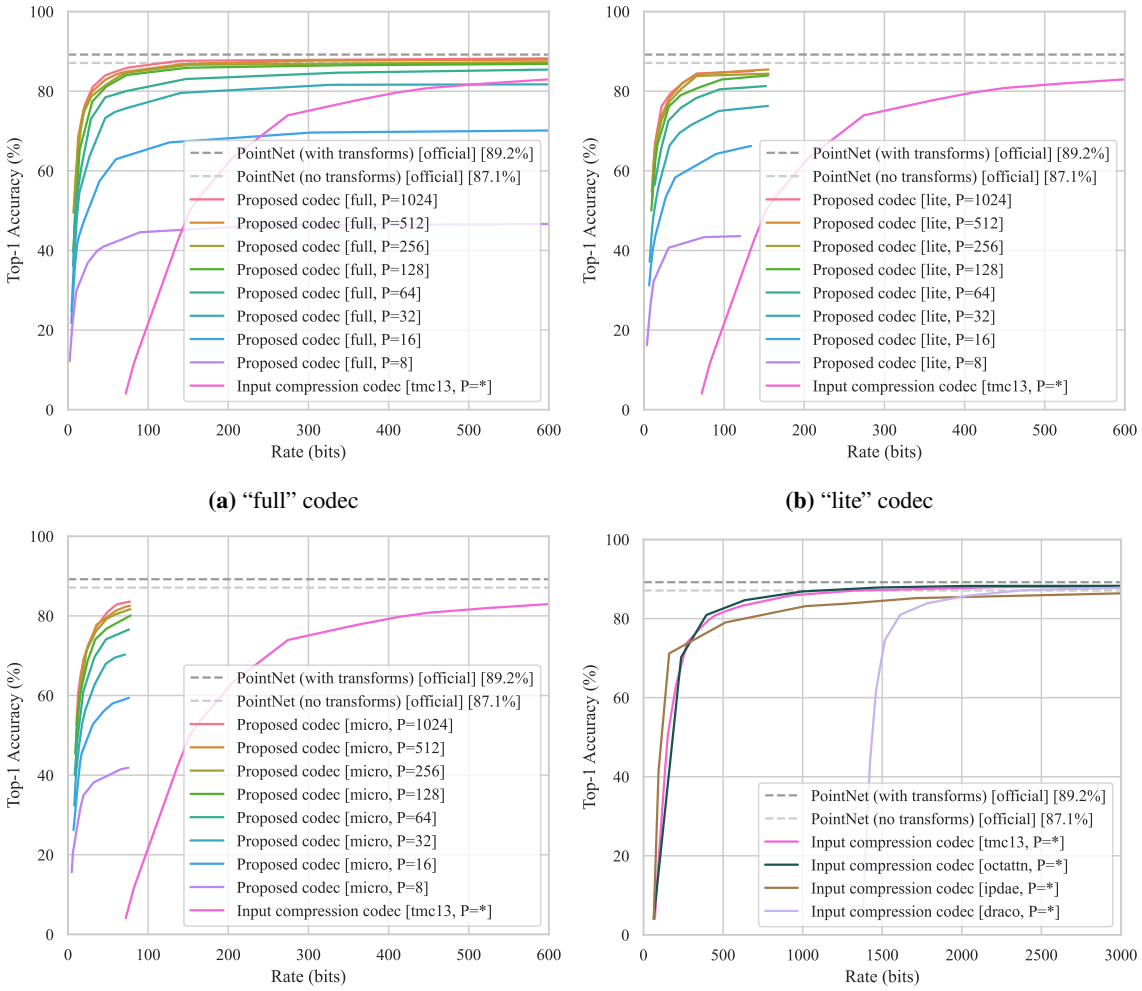
be computed by $\boldsymbol{x}_C = \bigcup_{1 \leq j \leq N} \arg\max_{\boldsymbol{x}_i \in \boldsymbol{x}} (h(\boldsymbol{x}_i))_j$, where $\{h(\boldsymbol{x}_i) : 1 \leq i \leq P\}$ represents the entire set of generated latent vectors immediately preceding max pooling.

3.5 Results

Fig. 3.3 shows the rate-accuracy (RA) curves for the proposed "full", "lite", and "micro" codecs in comparison with the input compression codec. Also included are two baseline accuracies taken from the official PointNet paper [54], for the model with (89.2%) and without (87.1%) the input/feature transforms. Since our compression models were all trained *without* the input/feature transforms, the lower baseline offers a more direct comparison. In Table 3.2, we list the peak accuracies attained by each codec, as well as the Bjøntegaard-Delta (BD) [64] improvements in rates and accuracies relative to the reference input compression codec.

Our "full" proposed codec, which is an extension of PointNet (no transforms), achieves the lower baseline accuracy at 120 bits, and an 80% accuracy at 30 bits. Our "lite" proposed codec saturates in RA performance at around $P = 512$ input points. At around $P = 256$, the total MAC count of the proposed "lite" encoder is roughly equal to the decoder. As shown by the rate-accuracy curves, the $P = 256$ model does not suffer too significant a drop in rate-accuracy performance. This suggests that our method is capable of achieving a good trade-off between rate, accuracy, and runtime performance. Similarly, our "micro" codec suffers a further slight drop in rate-accuracy performance, but achieves another significant improvement in runtime performance. The input compression codec is the worst performing codec, and attains the lower baseline accuracy at roughly 2000 bits.

In Fig. 3.4, we show various point clouds that have been reconstructed from the bitstreams generated by our proposed codecs. For each codec, we include samples reconstructed from bitstreams compressed at different rates. Above each reconstructed point cloud, we show the corresponding reference point cloud, with critical points marked in red.



(a) "full" codec

(b) "lite" codec

(c) "micro" codec

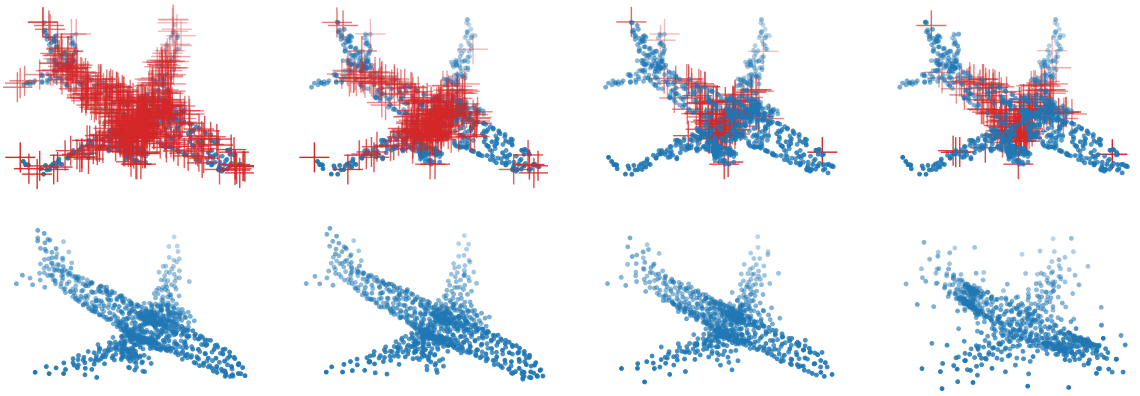
(d) input compression codecs

Figure 3.3: Rate-accuracy curves evaluated on the ModelNet40 test set.

Table 3.2: BD metrics and max attainable accuracies per codec.

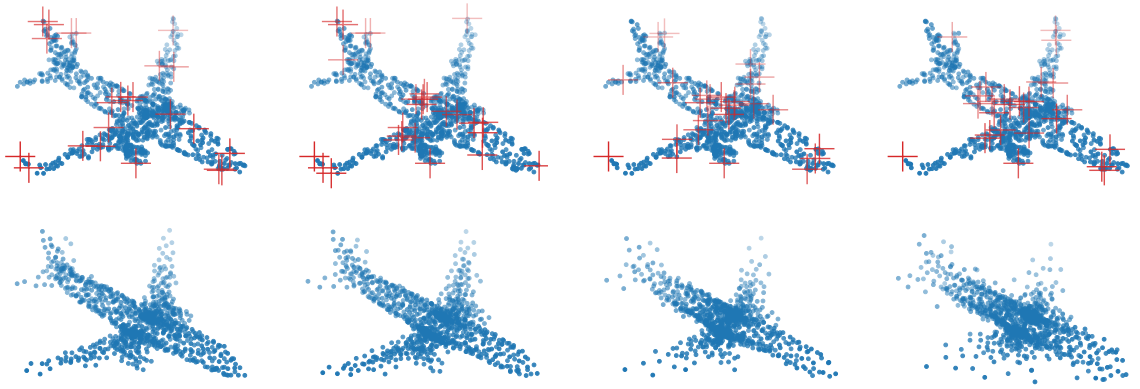
Codec	Max acc (%)	BD rate (rel %)	BD acc (%)
Input compression			
TMC13 [62]	88.5	0.0	0.0
OctAttention [10]	88.4	-13.2	+2.1
IPDAE [11]	87.0	-23.0	+3.6
Draco [63]	88.3	+780.7	-4.2
Proposed (full)			
$P = 1024$	88.5	-93.8	+16.4
$P = 512$	88.0	-93.7	+15.9
$P = 256$	87.6	-93.3	+15.4
$P = 128$	87.1	-92.7	+14.9
$P = 64$	86.1	-91.1	+13.2
$P = 32$	81.8	-90.6	+9.3
$P = 16$	70.4	-86.8	-2.3
$P = 8$	46.8	-88.5	-25.3
Proposed (lite)			
$P = 1024$	85.0	-93.0	+13.5
$P = 512$	85.5	-92.8	+14.2
$P = 256$	84.4	-92.4	+12.8
$P = 128$	84.0	-91.6	+12.5
$P = 64$	81.3	-88.5	+9.8
$P = 32$	76.3	-88.7	+4.9
$P = 16$	66.2	-86.1	-4.1
$P = 8$	43.6	-90.2	-28.0
Proposed (micro)			
$P = 1024$	83.6	-91.8	+12.7
$P = 512$	82.5	-91.6	+11.6
$P = 256$	81.6	-91.1	+11.0
$P = 128$	80.1	-90.9	+9.9
$P = 64$	76.6	-89.9	+6.5
$P = 32$	70.3	-89.0	+0.1
$P = 16$	59.4	-87.6	-10.8
$P = 8$	41.9	-88.3	-28.8

P is the number of points in the input $\boldsymbol{x}$. The BD metrics were computed using the TMC13 input compression codec as the reference anchor.



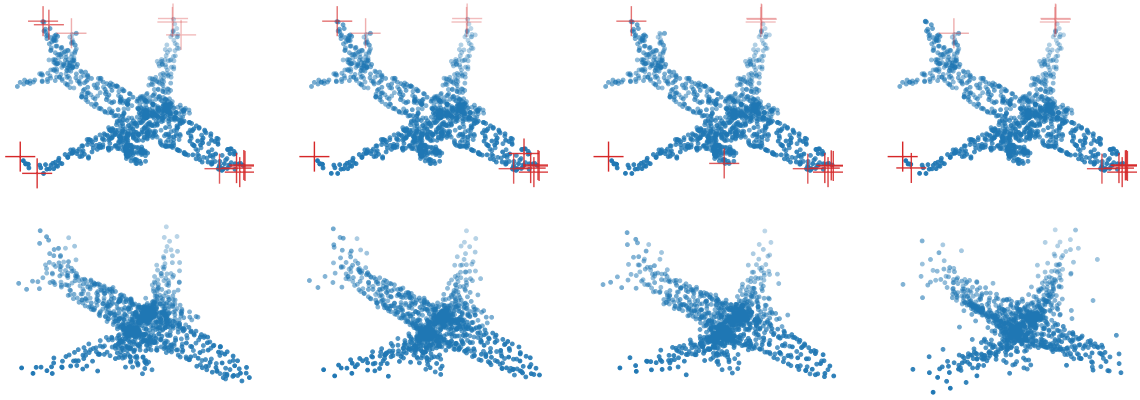
(i) 284 bits **(ii)** 43 bits **(iii)** 18 bits **(iv)** 12 bits

(a) "full" codec



(i) 124 bits **(ii)** 36 bits **(iii)** 19 bits **(iv)** 13 bits

(b) "lite" codec



(i) 76 bits **(ii)** 27 bits **(iii)** 23 bits **(iv)** 14 bits

(c) "micro" codec

Figure 3.4: Reconstructions of a sample airplane 3D model from the ModelNet40 test set for various codecs and bitrates. For each reconstruction, its corresponding reference point cloud is marked with *critical points* in red.

3.6 Discussion

For input point clouds containing $P = 1024$ points, our "full", "lite", and "micro" codec configurations achieve an accuracy of 80% with as few as 30, 40, and 50 bits. For comparison, $\log_2(40) \approx 5.3$ bits are required to losslessly encode uniformly distributed class labels of the 40 classes from ModelNet40. Our codec comes surprisingly close to this theoretical lower bound, despite the fact that our architecture design omits the traditional MLP "classifier" within the encoder. The same pointwise function is applied to all points, and the only operation that "mixes" information between the points is a pooling operation. This suggests that our encoder should possess limited classification abilities, and yet it consumes only a few times more bits than the theoretical lower bound.

To put this into perspective, consider coding for image classification, which is one of the best developed areas in the field of coding for machines. Current state-of-the-art (SOTA) approaches [65]–[67] for coding for image classification on ImageNet [68] require upwards of 0.01 bits per pixel (bpp) to maintain a reasonable top-1 accuracy. With the typical input image resolution of 224×224, this works out to be around 500 bits. However, the maximum classifier output entropy with 1000 classes is only $\log_2(1000) \approx 10$ bits, which is several orders of magnitude lower. Hence, the gap between the current SOTA and theoretical limits on coding for image classification is much higher than what is achieved by our proposed codec for point cloud classification.

To explore why our codec comes so close to the theoretical lower bound, we propose the following arguments. Let $\boldsymbol{x}$ represent a possible input point cloud, and let $\hat{\boldsymbol{y}} = (Q \circ g_a)(\boldsymbol{x})$ be its quantized transformed latent representation. Applying the data processing inequality to the Markov chain $\boldsymbol{x} \to \boldsymbol{x} \to \hat{\boldsymbol{y}}$, we determine that $I(\boldsymbol{x}; \boldsymbol{x}) \geq I(\boldsymbol{x}; \hat{\boldsymbol{y}})$. Furthermore, since $Q \circ g_a$ is deterministic, $H(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) = 0$, and so

$$H(\boldsymbol{x}) = I(\boldsymbol{x}; \boldsymbol{x}) \geq I(\boldsymbol{x}; \hat{\boldsymbol{y}}) = H(\hat{\boldsymbol{y}}) - H(\hat{\boldsymbol{y}} \mid \boldsymbol{x}) = H(\hat{\boldsymbol{y}}).$$

This indicates theoretically that the quantized latent representation $\hat{\boldsymbol{y}}$ must on average be at least as compressible as the input point cloud $\boldsymbol{x}$ that it was derived from.

Since the critical point set $\boldsymbol{x}_C \subseteq \boldsymbol{x}$ produces the exact same $\hat{\boldsymbol{y}}$ as the original input point cloud $\boldsymbol{x}$, we may use the same arguments as above to argue that

$$H(\boldsymbol{x}) \geq H(\boldsymbol{x}_C) \geq H(\hat{\boldsymbol{y}}).$$

This provides us with a potentially tighter bound. In fact, as shown in Fig. 3.4iii, much of the general shape of the shown sample point cloud can be reconstructed from only 23 bits of information. Furthermore, since $|\boldsymbol{x}_C| \leq N$, there are only at most 32 and 16 distinct critical points for the "lite" and "micro" codecs, respectively. This suggests part of the reason for why our proposed codec achieves such big gains in comparison to input compression.

3.7 Conclusion

In this chapter, we proposed a new codec for point cloud classification. Our experiments demonstrated that the "full" configuration of the codec achieves stellar rate-accuracy performance, far exceeding the performance of alternative methods. We also presented "lite" and "micro" configurations of the codec whose encoders consume minimal computational resources, and yet achieve comparable gains in rate-accuracy performance.

Our work may be extended to other point cloud tasks, such as segmentation and object detection, or to more complex tasks involving larger models and larger point clouds from real-world datasets. Our work also sets a good starting point for further research into approaches for scalable and multi-task point cloud compression. We hope that our work will help towards achieving the end goal of more capable end devices.

Chapter 4

Latent space motion analysis for collaborative intelligence

When the input to a deep neural network (DNN) is a video signal, a sequence of feature tensors is produced at the intermediate layers of the model. If neighboring frames of the input video are related through motion, a natural question is, "what is the relationship between the corresponding feature tensors?" By analyzing the effect of common DNN operations on optical flow, we show that the motion present in each channel of a feature tensor is approximately equal to the scaled version of the input motion. The analysis is validated through experiments utilizing common motion models. This chapter has been presented as [29].

4.1 Introduction

Collaborative intelligence (CI) [69] has emerged as a promising strategy to bring AI "to the edge." In a typical CI system (Fig. 4.1), a deep neural network (DNN) is split into two parts: the edge sub-model, deployed on the edge device near the sensor, and the cloud sub-model deployed in the cloud. Intermediate features produced by the edge sub-model are transferred from the edge device to the cloud. It has been shown that such a strategy may provide better energy efficiency [49], [70], lower latency [49], [70], [71], and lower bitrates over the communication channel [56], [72], compared to more traditional cloud-based analytics where the input signal is directly sent to the cloud. These potential benefits will find a number of applications in areas such as intelligent sensing [73] and video coding for machines [20], [74]. In particular, compression of intermediate features has become an important research problem, with a number of recent developments [75]–[79] for the case when the input to the edge sub-model is a still image.

When the input to the edge sub-model is video, its output is a sequence of feature tensors produced from successive frames in the input video. This sequence of feature tensors needs to be compressed prior to transmission and then decoded in the cloud for further processing. Since motion plays such an important role in video processing and compression, we are motivated to examine whether any similar relationship exists in the latent space among the feature tensors. Our theoretical and experimental

results show that, indeed, motion from the input video is approximately preserved in the channels of the feature tensor. An illustration of this is presented in Fig. 4.2, where the estimated input-space motion field is shown on the left, and the estimated motion fields in several feature tensor channels are shown on the right. These findings suggest that methods for motion estimation, compensation, and analysis that have been developed for conventional video processing and compression may provide a solid starting point for equivalent operations in the latent space.

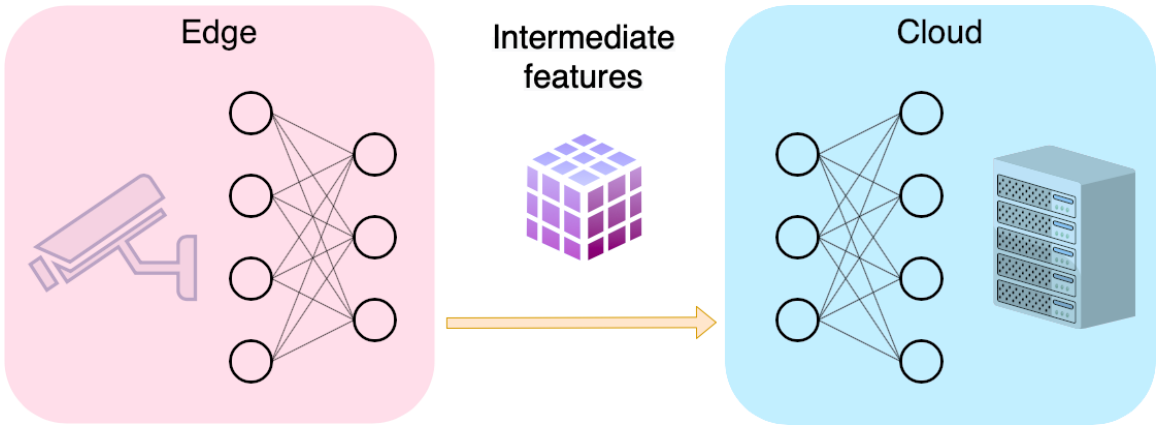


Figure 4.1: Basic collaborative intelligence system.

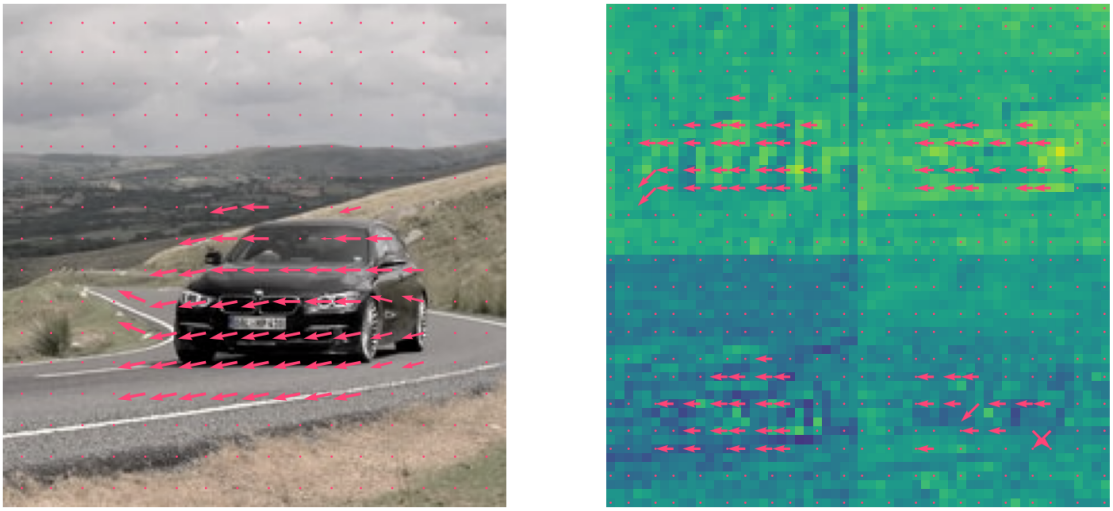


Figure 4.2: Motion estimates for input frames (left) and select channels from the output of ResNet-34's add_3 layer (right).

This chapter is organized as follows. In Section 4.2, we analyze the actions of typical operations found in deep convolutional neural networks on optical flow in the input signal, and show that these operations tend to preserve the optical flow, at least approximately, with an appropriate scale. In Section 4.3 we provide empirical support for the theoretical analysis from Section 4.2.

4.2 Latent space motion analysis

The basic problem studied in this chapter is illustrated in Fig. 4.3. Consider two images (video frames) input to the edge sub-model of a CI system. It is common to represent their relationship via a motion model. The question we seek to answer here is, "what is the relationship between the corresponding feature tensors produced by the edge sub-model?" To answer this question, we will look at the processing pipeline between the input image and a given channel of a feature tensor. In most deep models for computer vision applications, this processing pipeline consists of a sequence of basic operations: convolutions, pointwise nonlinearities, and pooling. We will show that each of these operations tends to preserve motion, at least approximately, in a certain sense, and from this we will conclude that (approximate) input motion may be observed in individual channels of a feature tensor.

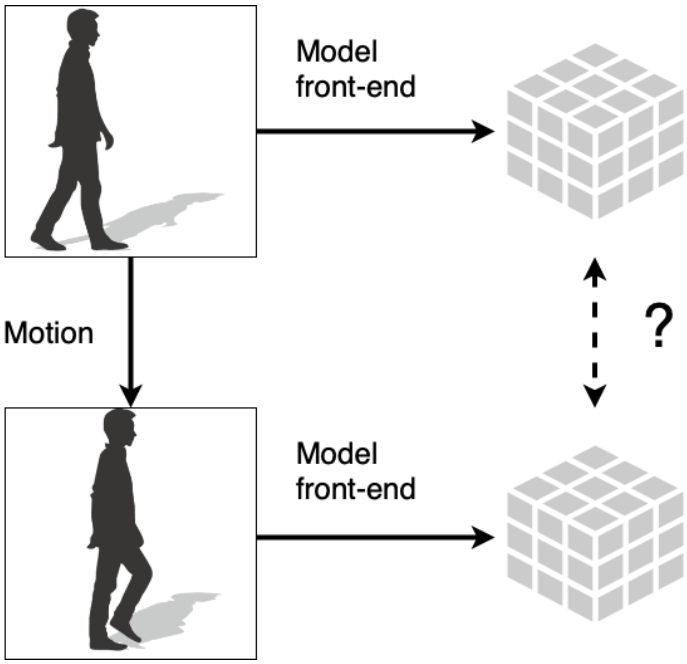


Figure 4.3: The problem studied in this chapter: if input images are related via motion, what is the relationship between the corresponding intermediate feature tensors?

Motion model. Optical flow is a frequently used motion model in computer vision and video processing. In a "2D+t" model, $I(x, y, t)$ denotes pixel intensity at time t, at spatial position (x, y). Under a constant-intensity assumption, optical flow satisfies the following partial differential equation [80]:

$$\frac{\partial I}{\partial x} v_x + \frac{\partial I}{\partial y} v_y + \frac{\partial I}{\partial t} = 0, \tag{4.1}$$

where (v_x, v_y) represents the motion vector. For notational simplicity, in the analysis below we will use a "1D+t" model, which captures all the main ideas but keeps the equations shorter. In a "1D+t"

model, $I(x,t)$ denotes intensity at position x at time t, and the optical flow equation is

$$\frac{\partial I}{\partial x}v + \frac{\partial I}{\partial t} = 0, \tag{4.2}$$

with v representing the motion. We will analyze the effect of basic operations — convolutions, pointwise nonlinearities, and pooling — on (4.2), to gain insight into the relationship between input space motion and latent space motion.

Convolution. Let f be a (spatial) filter kernel, then the optical flow after convolution is a solution to the following equation

$$\frac{\partial}{\partial x}(f * I)v' + \frac{\partial}{\partial t}(f * I) = 0, \tag{4.3}$$

where v' is the motion after the convolution. Since the convolution and differentiation are linear operations, we have

$$f * \left(\frac{\partial I}{\partial x}v' + \frac{\partial I}{\partial t}\right) = 0. \tag{4.4}$$

Hence, solution v from (4.2) is also a solution of (4.4), but (4.4) could also have other solutions, besides those that satisfy (4.2).

Pointwise nonlinearity. Nonlinear activations such as `sigmoid`, `ReLU`, etc., are commonly applied in a pointwise fashion on the output of convolutions in deep models. Let $\sigma(\cdot)$ denote such a pointwise nonlinearity, then the optical flow after this nonlinearity is a solution to the following equation

$$\frac{\partial}{\partial x}\left[\sigma(I)\right]v' + \frac{\partial}{\partial t}\left[\sigma(I)\right] = 0, \tag{4.5}$$

where v' is the motion after the pointwise nonlinearity. By using the chain rule of differentiation, the above equation can be rewritten as

$$\sigma'(I) \cdot \left(\frac{\partial I}{\partial x}v' + \frac{\partial I}{\partial t}\right) = 0. \tag{4.6}$$

Hence, again, solution v from (4.2) is also a solution of (4.6). It should be noted that (4.6) may have solutions other than those from (4.2). For example, in the region where inputs to `ReLU` are negative, the corresponding outputs will be zero, so $\sigma'(I) = 0$. Hence, in those regions, (4.6) will be satisfied for arbitrary v'. Nonetheless, the solution from (4.2) is still one of those arbitrary solutions.

Pooling. There are various forms of pooling, such as max-pooling, mean-pooling, learnt pooling (via strided convolutions), etc. All these can be decomposed into a sequence of two operations: a spatial operation (local maximum or convolution) followed by scale change (downsampling). Spatial convolution operations can be analyzed as above, and the conclusion is that motion before such an operation is also a solution to the optical flow equation after such an operation. Hence, we will focus here on the local maximum operation and the scale change.

Local maximum. Consider the maximum of function $I(x,t)$ over a local spatial region $[x_0 - h, x_0 + h]$, at a given time t. We can approximate $I(x,t)$ as a locally-linear function, whose slope is the spatial derivative of I at x_0, $\frac{\partial}{\partial x}I(x_0,t)$. If the derivative is positive, the maximum is $I(x_0+h,t)$,

and if it is negative, it is $I(x_0 - h, t)$. In the special case when the derivative is zero, any point in $[x_0 - h, x_0 + h]$, including the endpoints, is a maximum. From Taylor series expansion of $I(x, t)$ around x_0 up to and including the first-order term,

$$I(x_0 \pm \epsilon, t) \approx I(x_0, t) \pm \frac{\partial}{\partial x} I(x_0, t) \cdot \epsilon, \tag{4.7}$$

for $|\epsilon| \leq h$. With such linear approximation, the local maximum of $I(x, t)$ over $[x_0 - h, x_0 + h]$ occurs either at $x_0 + h$ or at $x_0 - h$, depending on the sign of $\frac{\partial}{\partial x} I(x_0, t)$; if the derivative is zero, every point in the interval is a local maximum. Hence, the local maximum of $I(x, t)$ can be approximated as

$$\begin{aligned} &\max_{x \in [x_0 - h, x_0 + h]} I(x, t) \\ &\quad \approx I(x_0, t) + \text{sign}\left(\frac{\partial}{\partial x} I(x_0, t)\right) \cdot \frac{\partial}{\partial x} I(x_0, t) \cdot h. \end{aligned} \tag{4.8}$$

Let (4.8) be the definition of $M(x_0, t)$, the function that takes on local spatial maximum values of $I(x, t)$ over windows of size $2h$. The optical flow after such a local maximum operation is described by

$$\frac{\partial M}{\partial x} v' + \frac{\partial M}{\partial t} = 0, \tag{4.9}$$

where v' represents the motion after local spatial maximum operation. Using (4.8) in (4.9), after some manipulation we obtain the following equation

$$\frac{\partial I}{\partial x} v' + \frac{\partial I}{\partial t} + \text{sign}\left(\frac{\partial I}{\partial x}\right) \cdot \frac{\partial}{\partial x}\left(\frac{\partial I}{\partial x} v' + \frac{\partial I}{\partial t}\right) \cdot h = 0. \tag{4.10}$$

Note that if v' satisfies the original optical flow equation (4.3), it will also satisfy (4.10), hence pre-max motion v is also one possible solution to post-max motion v'.

Scale change. Finally, consider the change of spatial scale by a factor s, such that the new signal is $I'(x, t) = I(s \cdot x, t)$. The optical flow equation is now

$$\frac{\partial I'}{\partial x} v' + \frac{\partial I'}{\partial t} = 0. \tag{4.11}$$

Since $\frac{\partial I'}{\partial x} = s \cdot \frac{\partial I}{\partial x}$ and $\frac{\partial I'}{\partial t} = \frac{\partial I}{\partial t}$, we conclude that $v' = v/s$, where v is the solution to pre-scaling motion (4.2). Hence, as expected, down-scaling the signal spatially by a factor of 2 ($s = 2$) would reduce the motion by a factor of 2.

Combining the results of the above analyses, we conclude that convolutions, pointwise non-linearities, and local maximum operations tend to be motion-preserving operations, in the sense that pre-operation motion is also a solution to post-operation optical flow, at least approximately. The operation with the most obvious impact on motion is scale change. Hence, when looking at latent-space motion at some layer in a deep model, we should expect to find motion similar to the input motion, but scaled down by a factor of n^k, where k is the number of pooling operations (over

$n \times n$ windows) between the input and the layer of interest. Specifically, if $\mathbf{v}$ is the motion vector at some position in the input frame, then at the corresponding spatial location in all the channels of the feature tensor we can expect to find the vector

$$\mathbf{v}' \approx \mathbf{v}/n^k. \tag{4.12}$$

In Section 4.3, we will verify these conclusions experimentally.

4.3 Experiments

An illustration of the correspondence between the input-space motion and latent-space motion was shown in Fig. 4.2. This example was produced using a pair of frames from a video of a moving car. The motion vectors were estimated using an exhaustive block-matching search at each pixel, which sought to minimize the sum of squared differences (SSD). In the input frames, whose resolution was 224×224, the block size of 31×31 around each pixel and the search range of ± 11 were used. In the corresponding feature tensor channels, whose resolution was 28×28, the block size of 3×3 and a search range of ± 5 were used. Although the estimated motion vector fields are somewhat noisy, the similarity between the input-space motion and latent-space motion is evident.

To examine the relationship between input-space and latent-space motion more closely, we performed several experiments with synthetic input-space motion. In this case, exact input-space motion is known, so relationship (4.12) can be tested more reliably. Fig. 4.4 shows examples of various transformations (translation, rotation, stretching, shearing) applied to an input image of a dog. The second column displays several channels from the actual tensor produced by the transformed image, and the third column shows the corresponding channels produced by motion compensating the tensor of the original image via (4.12). The last column shows the difference between the actual and predicted tensor channels. Note that regions that cannot be predicted, such as regions "entering the frame," were excluded from difference computation. As seen in Fig. 4.4, the model (4.12) works reasonably well, and the differences between the actual and predicted tensors are low.

For quantitative evaluation, experiments were conducted on several layers of ResNet-34 [81] and DenseNet-121 [82]. Normalized Root Mean Square Error (NRMSE) [83] was used for this purpose:

$$\text{NRMSE} = \frac{1}{R}\sqrt{\frac{1}{N}\sum_{i=1}^{N}(p_i - a_i)^2}, \tag{4.13}$$

where a_i is the actual tensor value produced from the transformed input, p_i is the tensor value predicted using our motion model (4.12), N is the number of elements in the feature tensor, and R is the dynamic range. Again, regions that cannot be predicted were excluded from NRMSE computation. Fig. 4.5 shows NRMSE computed across a range of parameters for several transformations, at various layers of the two DNNs.

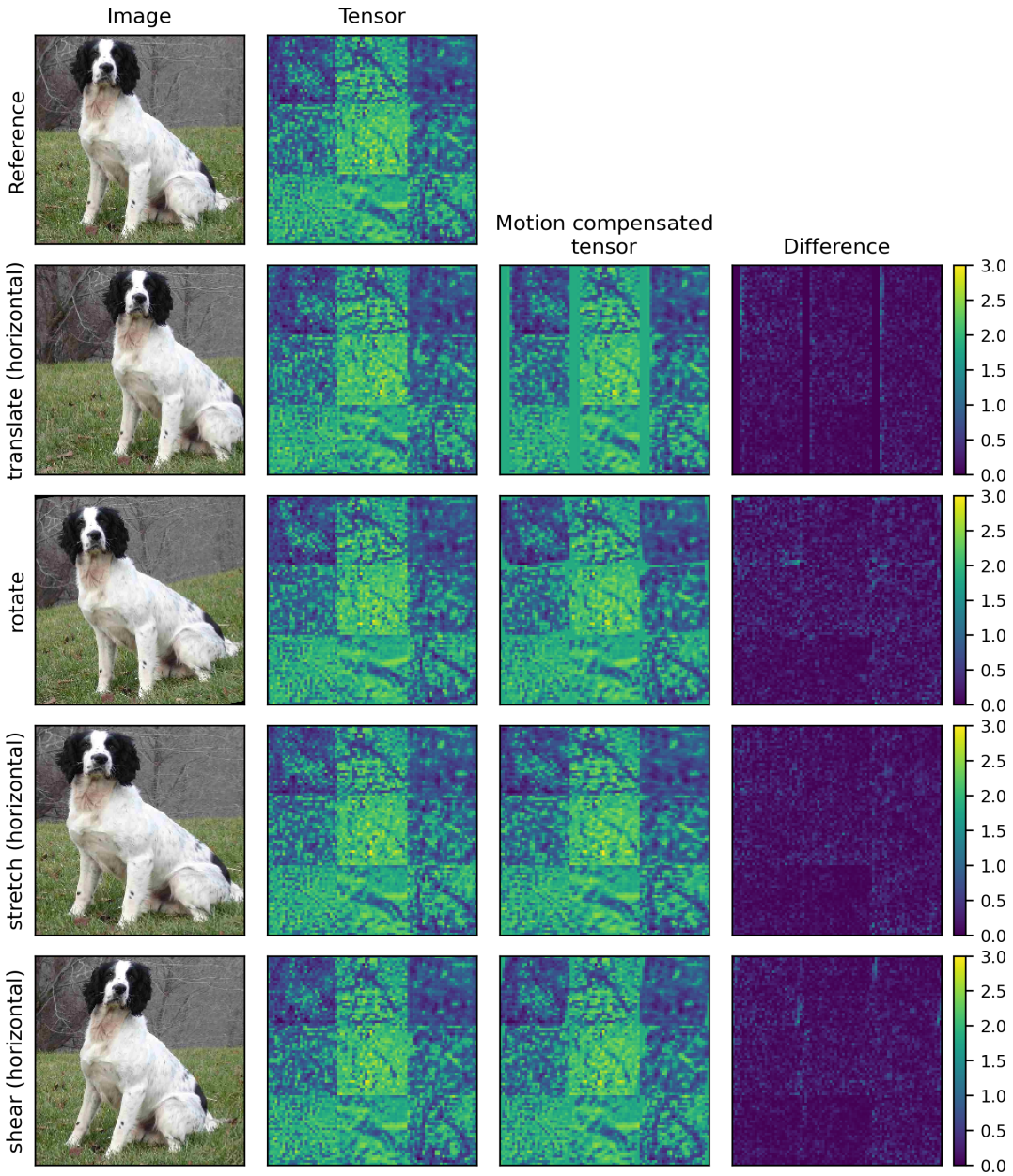


Figure 4.4: Examples of motion transformations applied to reference image. The output tensors of ResNet-34's add_3 layer are reliably predicted from only the reference tensor and known input-space motion.

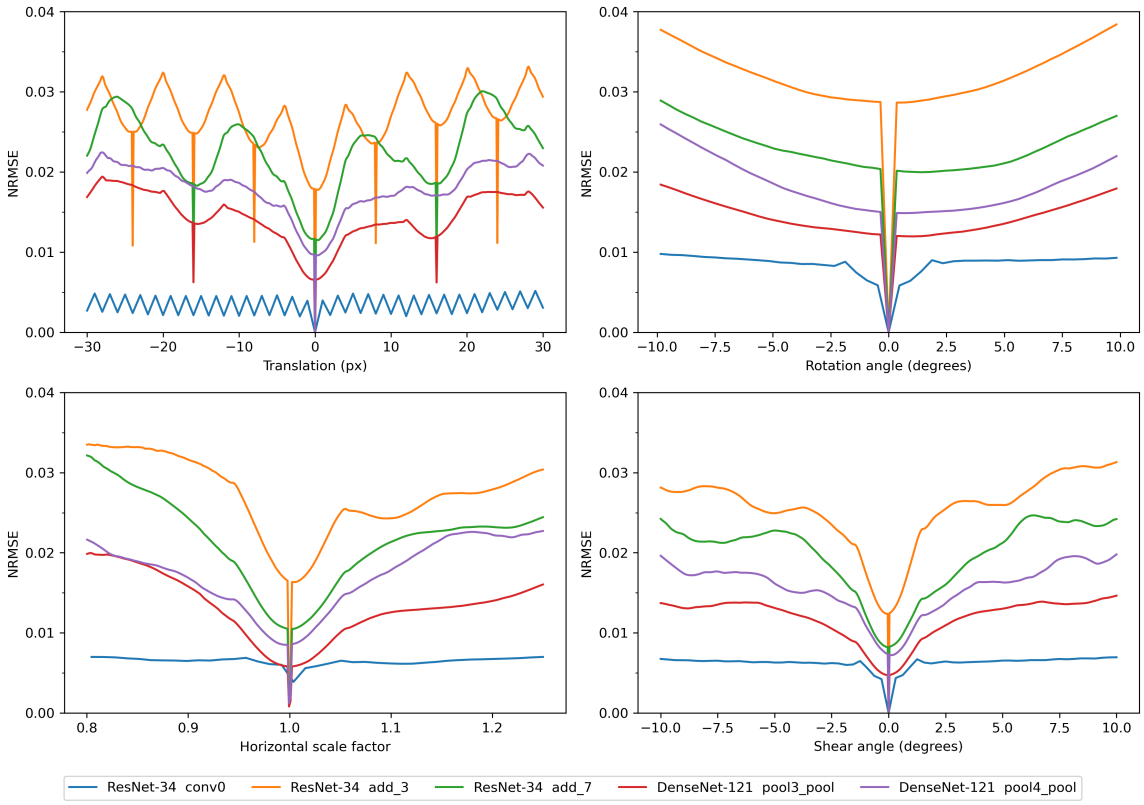


Figure 4.5: NRMSE across parameter ranges for translation (top-left), rotation (top-right), scaling (bottom-left), and shear (bottom-right). For translation, NRMSE local minima occur when the input-space shifts correspond to integer latent-space shifts in (4.12).

As seen in Fig. 4.5, NRMSE goes up to about 0.04 for reasonable ranges of transformation parameters. How good is this? To answer this question, we set out to find the typical values of NRMSE found in conventional motion-compensated frame prediction. In a recent study [84], the quality of frames predicted by conventional motion estimation and motion compensation (MEMC) in High Efficiency Video Coding (HEVC) [85] was compared against a DNN developed for frame prediction. From Table III in [84], the luminance Peak Signal to Noise Ratio (PSNR) of frames predicted uni-directionally by the DNN and conventional HEVC MEMC was in the range 27–41 dB over several HEVC test sequences. NRMSE can be computed from PSNR as

$$\text{NRMSE} = \frac{1}{256}\sqrt{\frac{255^2}{10^{\text{PSNR}/10}}}, \tag{4.14}$$

so the PSNR range of 27–41 dB corresponds to the NRMSE range of 0.009–0.044. These levels of NRMSE are indicative of how much motion models used in video coding deviate from the true motion. As seen in Fig. 4.5, the model (4.12) produces NRMSE in the same range, so the accuracy of (4.12) is comparable to the accuracy of common motion models used in video coding. Another illustration of this is presented in Fig. 4.6, which shows the histogram of NRMSE computed across a range of

affine transformation parameters. Hence, (4.12) represents a good starting point for development of latent-space motion compensation.

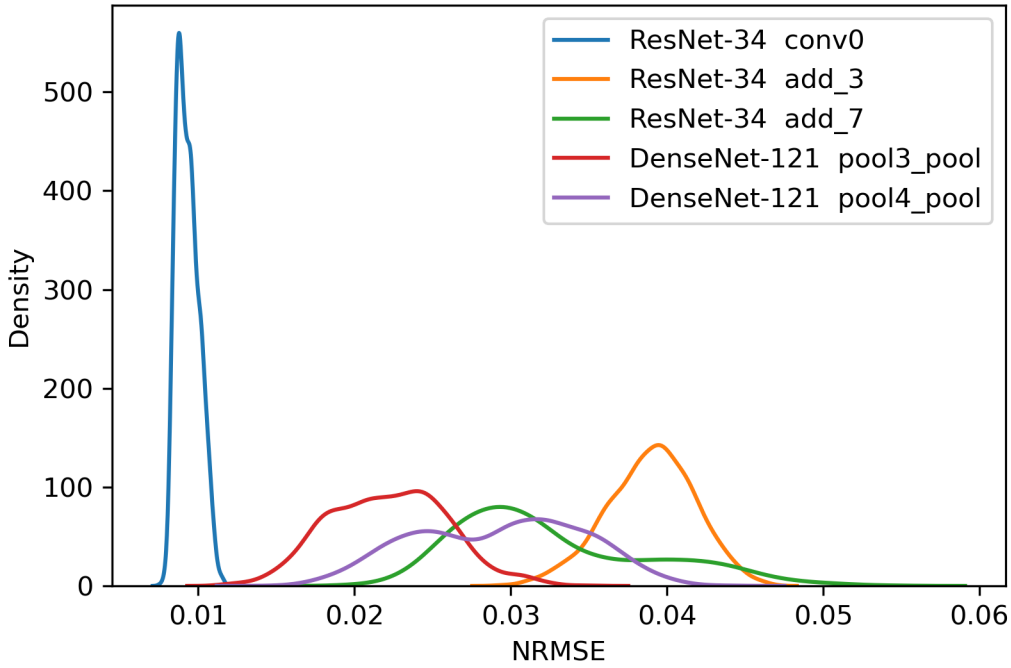


Figure 4.6: NRMSE histogram computed for an affine motion model [86] over the combination of the following seven independent uniformly distributed parameters: x- and y-translation (± 32 px), x- and y-scaling (0.95–1.05), x- and y-shearing ($\pm 5°$), and rotation ($\pm 10°$).

4.4 Conclusions

Using the concept of optical flow, in this chapter we analyzed motion in the latent space of a deep model induced by the motion in the input space, and showed that motion tends to be approximately preserved in the channels of intermediate feature tensors. These findings suggest that motion estimation, compensation, and analysis methods developed for conventional video signals should be able to provide a good starting point for latent-space motion processing, such as motion-compensated prediction and compression, tracking, action recognition, and other applications.

Chapter 5

Conclusion

In this thesis, we have explored various aspects of learned compression for images, point clouds, and video.

Our first contribution made in Chapter 2, "Compression of probability distributions", proposes a simple low-complexity and yet effective method for dynamically adapting the entropy bottleneck to the input distribution. The static entropy bottleneck is a common component in many SOTA learned compression models. However, its proportional bitrate usage is minimal in image compression models, in comparison to the Gaussian conditional component — though, this is likely because of its non-adaptive nature! Thus, replacing the static entropy bottleneck with an alternative adaptive entropy bottleneck may lead to a small but consistent boost in performance. It should be noted that a correct plug-and-play end-to-end training implementation of our dynamic entropy model is still under development.

Nonetheless, when trained on a pretrained model with frozen transform weights, our dynamically adaptive entropy bottleneck showed gains of -6.95% in BD-rate over a static entropy bottleneck. Thus, our method provides a low-complexity mechanism that can be used to aid in building practical, efficient models for the real world. Currently, many SOTA models have been largely focused on RD performance. While the exceptional results on the frontier of RD performance achievement are exciting, it is important to step back and think about how to make learned compression a feasible option for practical use. Our proposed method takes a step in that direction. It is quite possible that a practical standardized learned codec might someday make use of our highly efficient low-cost solution in lieu of a heavier alternative.

Our work presented in Chapter 3, "Point cloud compression for classification", opens the way for point cloud-oriented machine-task oriented codecs. We showed how a simple Point Net-based codec focused on the machine task can achieve significantly better results than a more conventional "compress, transmit, decompress, machine task model" approach. Furthermore, our "micro" encoder operates at an ultra-low complexity of 48 MACs/pt, and yet achieves a far better accuracy for any given bitrate than the conventional approach.

Interestingly, our codec comes close to the theoretical limits of compression for 40 evenly-distributed class labels. This can in part be attributed to the inherently low entropy of small point

clouds. (e.g. SOTA codecs compress to $\tilde{1}$ bit/pt with acceptable amounts of distortion.) But another dominant factor is that arguably, the ModelNet40 dataset contains fairly easy-to-classify CAD object point clouds that are noise-free, isolated, and well-defined from all angles. In comparison, real world point clouds are much noisier, contain a surrounding environment, and often only have measurements visible from a particular angle. This is evident in the case of LIDAR-generated point clouds. It is harder to isolate the relevant information in such a setting. Thus, real-world scenarios present a greater challenge for machine-task oriented codecs. Nonetheless, we believe that with some effort, machine-task codecs can demonstrate their utility in more complex settings.

In Chapter 4, "Latent space motion analysis for collaborative intelligence", we showed that in convolutional-based models, motion within the input domain leads to predictable motion within the latent domain. Furthermore, we quantified how predictable the next "latent frame" is in terms of the amount of residual error that is produced by warping the "reference latent frame". This is akin to video coding via p-frames, except in the latent domain. We found that the residual NRMSE is kept under 0.04 for randomly tested "reasonably expected" affine transformations/warps. (For reference, a NRMSE of 0.04 very roughly corresponds to 27 dB PSNR for conventional image coding.)

Our results suggest that learned video compression approaches that rely upon warping of the latent space are feasible. Methods such as Scale-Space Flow [13] rely upon transforming the predicted frame back into the input domain before applying motion-based warping, and then computing the residual frame, and then transforming the residual back into a latent domain for encoding. Potentially, this costly and possibly even suboptimal step of converting back-and-forth between domains can be eliminated by simply warping the latent space directly, as our study indicates may be viable.

All in all, we believe that our work has contributed to the new and promising field of learned compression.

References

[1] E. Kodak, "Kodak lossless true color image suite (PhotoCD PCD0992)," [Online]. Available: http://r0k.us/graphics/kodak (cit. on pp. 3, 13, 24, 25).

[2] D. Minnen, J. Ballé, and G. D. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," *Advances in neural information processing systems*, vol. 31, 2018, arXiv: 1809.02736 [cs.CV] (cit. on pp. 2, 12).

[3] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Proc. IEEE/CVF CVPR*, 2020, pp. 7939–7948, arXiv: 2001.01568 [eess.IV] (cit. on pp. 2, 7, 26).

[4] D. He, Z. Yang, W. Peng, R. Ma, H. Qin, and Y. Wang, "ELIC: Efficient learned image compression with unevenly grouped space-channel contextual adaptive coding," in *Proc. IEEE/CVF CVPR*, 2022, pp. 5718–5727, arXiv: 2203.10886 [cs.CV] (cit. on pp. 2, 12, 26).

[5] G. Toderici, D. Vincent, N. Johnston, *et al.*, "Full resolution image compression with recurrent neural networks," in *Proc. IEEE/CVF CVPR*, IEEE, 2017, pp. 5306–5314, arXiv: 1608.05148 [cs.CV], [Online]. Available: http://dx.doi.org/10.1109/CVPR.2017.577 (cit. on p. 2).

[6] F. Mentzer, G. D. Toderici, M. Tschannen, and E. Agustsson, "High-fidelity generative image compression," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 913–11 924, 2020, [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/8a50bae297807da9e97722a0b3fd8f27-Paper.pdf (cit. on p. 2).

[7] W. Yan, Y. shao, S. Liu, T. H. Li, Z. Li, and G. Li, "Deep autoencoder-based lossy geometry compression for point clouds," 2019, arXiv: 1905.03691 [cs.CV] (cit. on pp. 2, 30).

[8] Y. He, X. Ren, D. Tang, Y. Zhang, X. Xue, and Y. Fu, "Density-preserving deep point cloud compression," in *Proc. IEEE/CVF CVPR*, 2022, pp. 2323–2332, arXiv: 2204.12684 [cs.CV] (cit. on pp. 2, 30).

[9] J. Pang, M. A. Lodhi, and D. Tian, "GRASP-Net: Geometric residual analysis and synthesis for point cloud compression," in *Proc. 1st Int. Workshop on Advances in Point Cloud Compression, Processing and Analysis*, 2022, arXiv: 2209.04401 [cs.CV] (cit. on pp. 2, 30).

[10] C. Fu, G. Li, R. Song, W. Gao, and S. Liu, "OctAttention: Octree-based large-scale contexts model for point cloud compression," in *Proc. AAAI*, vol. 36, Jun. 2022, pp. 625–633, DOI: 10.1609/aaai.v36i1.19942 (cit. on pp. 2, 30, 34, 37).

[11] K.-S. You, P. Gao, and Q. T. Li, "IPDAE: Improved patch-based deep autoencoder for lossy point cloud geometry compression," in *Proc. 1st Int. Workshop on Advances in Point Cloud Compression, Processing and Analysis*, 2022, arXiv: 2208.02519 [cs.CV] (cit. on pp. 2, 30, 37).

[12] O. Rippel, S. Nair, C. Lew, S. Branson, A. G. Anderson, and L. Bourdev, "Learned video compression," in *Proc. IEEE/CVF ICCV*, 2019, pp. 3454–3463, arXiv: 1811.06981 [eess.IV] (cit. on p. 2).

[13] E. Agustsson, D. Minnen, N. Johnston, J. Ballé, S. J. Hwang, and G. Toderici, "Scale-Space Flow for end-to-end optimized video compression," in *Proc. IEEE/CVF CVPR*, 2020, pp. 8500–8509, [Online]. Available: https://openaccess.thecvf.com/content_CVPR_2020/papers/Agustsson_Scale-Space_Flow_for_End-to-End_Optimized_Video_Compression_CVPR_2020_paper.pdf (cit. on pp. 2, 51).

[14] Z. Hu, G. Lu, and D. Xu, "FVC: A new framework towards deep video compression in feature space," in *Proc. IEEE/CVF CVPR*, 2021, pp. 1502–1511, arXiv: 2105.09600 [eess.IV], [Online]. Available: http://dx.doi.org/10.1109/CVPR46437.2021.00155 (cit. on p. 2).

[15] Y.-H. Ho, C.-P. Chang, P.-Y. Chen, A. Gnutti, and W.-H. Peng, "CANF-VC: Conditional augmented normalizing flows for video compression," in *Proc. ECCV*, Springer, 2022, pp. 207–223, arXiv: 2207.05315 [cs.CV] (cit. on p. 2).

[16] F. Galpin, M. Balcilar, F. Lefebvre, F. Racapé, and P. Hellier, "Entropy coding improvement for low-complexity compressive auto-encoders," in *Proc. IEEE DCC*, 2023, pp. 338–338, arXiv: 2303.05962 [eess.IV] (cit. on pp. 3, 12).

[17] T. Ladune, P. Philippe, F. Henry, G. Clare, and T. Leguay, "COOL-CHIC: Coordinate-based low complexity hierarchical image codec," in *Proc. IEEE/CVF ICCV*, 2023, pp. 13 515–13 522, arXiv: 2212.05458 [eess.IV] (cit. on p. 3).

[18] T. Leguay, T. Ladune, P. Philippe, G. Clare, and F. Henry, "Low-complexity overfitted neural image codec," 2023, arXiv: 2307.12706 [eess.IV] (cit. on p. 3).

[19] F. Kamisli, "Learned lossless image compression through interpolation with low complexity," *IEEE Trans. Circuits Syst. Video Technol.*, pp. 1–1, 2023, arXiv: 2212.13243 [eess.IV] (cit. on p. 3).

[20] L.-Y. Duan, J. Liu, W. Yang, T. Huang, and W. Gao, "Video Coding for Machines: A paradigm of collaborative compression and intelligent analytics," *IEEE Trans. Image Process.*, vol. 29, pp. 8680–8695, 2020, arXiv: 2001.03569 [cs.CV] (cit. on pp. 3, 30, 41).

[21] H. Choi and I. V. Bajić, "Latent-space scalability for multi-task collaborative intelligence," in *Proc. IEEE ICIP*, 2021, pp. 3562–3566, arXiv: 2105.10089 [eess.IV] (cit. on p. 3).

[22] H. Choi and I. V. Bajić, "Scalable image coding for humans and machines," *IEEE Trans. Image Process.*, vol. 31, pp. 2739–2754, Mar. 2022, arXiv: 2107.08373 [eess.IV] (cit. on pp. 3, 30).

[23] J. Ballé, V. Laparra, and E. P. Simoncelli, "Density modeling of images using a generalized normalization transformation," in *Proc. ICLR*, 2016, arXiv: 1511.06281 [cs.LG] (cit. on p. 5).

[24] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *Proc. ICLR*, 2018, arXiv: 1802.01436 [eess.IV] (cit. on pp. 5, 6, 7, 10, 11, 13, 17, 18, 25, 26, 30, 32).

[25] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "CompressAI: A PyTorch library and evaluation platform for end-to-end compression research," 2020, arXiv: 2011.03029 [cs.CV] (cit. on pp. 5, 23, 33).

[26] J. Duda, "Asymmetric numeral systems: Entropy coding combining speed of huffman coding with compression rate of arithmetic coding," 2013, arXiv: 1311.2540 [cs.IT] (cit. on p. 5).

[27] F. Giesen, "Ryg_rans," 2014, [Online]. Available: https://github.com/rygorous/ryg_rans (cit. on p. 5).

[28] **M. Ulhaq** and I. V. Bajić, "Learned point cloud compression for classification," in *Proc. IEEE MMSP*, 2023, arXiv: 2308.05959 [eess.IV] (cit. on pp. 8, 28).

[29] **M. Ulhaq** and I. V. Bajić, "Latent space motion analysis for collaborative intelligence," in *Proc. IEEE ICASSP*, 2021, pp. 8498–8502, arXiv: 2102.04018 [cs.CV] (cit. on pp. 8, 41).

[30] E. Özyılkan*, **M. Ulhaq***, H. Choi, and F. Racapé, "Learned disentangled latent representations for scalable image coding for humans and machines," in *Proc. IEEE DCC*, 2023, pp. 42–51, arXiv: 2301.04183 [eess.IV] (cit. on p. 8).

[31] **M. Ulhaq** and F. Racapé, "CompressAI Trainer," 2022, [Online]. Available: https://github.com/InterDigitalInc/CompressAI-Trainer (cit. on pp. 8, 33).

[32] H. Choi, F. Racapé, S. Hamidi-Rad, **M. Ulhaq**, and S. Feltman, "Frequency-aware learned image compression for quality scalability," in *Proc. IEEE VCIP*, 2022, pp. 1–5, arXiv: 2301.01290 [eess.IV] (cit. on p. 9).

[33] S. R. Alvar, **M. Ulhaq**, H. Choi, and I. V. Bajić, "Joint image compression and denoising via latent-space scalability," *Frontiers in Signal Processing*, vol. 2, 2022, arXiv: 2205.01874 [eess.IV] (cit. on p. 9).

[34] **M. Ulhaq** and I. V. Bajić, "ColliFlow: A library for executing collaborative intelligence graphs," demoed at NeurIPS, 2020, [Online]. Available: https://yodaembedding.github.io/neurips-2020-demo/ (cit. on p. 9).

[35] M. Balcilar, B. Damodaran, and P. Hellier, "Reducing the amortization gap of entropy bottleneck in end-to-end image compression," in *Proc. Picture Coding Symposium (PCS)*, IEEE, 2022, pp. 115–119, arXiv: 2209.00964 [eess.IV] (cit. on pp. 11, 12, 25).

[36] C. Cremer, X. Li, and D. Duvenaud, "Inference suboptimality in variational autoencoders," in *Int. Conf. on Machine Learning (ICML)*, PMLR, 2018, pp. 1078–1086, arXiv: 1801.03558 [cs.LG] (cit. on p. 11).

[37] J. Campos, S. Meierhans, A. Djelouah, and C. Schroers, "Content adaptive optimization for neural image compression," in *Proc. IEEE/CVF CVPR Workshops*, 2019, arXiv: 1906.01223 [cs.CV] (cit. on p. 12).

[38] Y. Bengio, N. Léonard, and A. Courville, "Estimating or propagating gradients through stochastic neurons for conditional computation," 2013, arXiv: 1308.3432 [cs.LG] (cit. on p. 18).

[39] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013, arXiv: 1312.6114 [stat.ML] (cit. on p. 18).

[40] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," in *Proc. ICLR*, 2017, arXiv: 1611.01704 [cs.CV] (cit. on p. 18).

[41] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," in *Proc. ICLR*, 2017, arXiv: 1703.00395 [stat.ML] (cit. on p. 18).

[42] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE/CVF CVPR*, 2018, pp. 6848–6856, arXiv: 1707.01083 [cs.CV] (cit. on pp. 22, 33).

[43] Z. Cui, J. Wang, S. Gao, B. Bai, T. Guo, and Y. Feng, "G-VAE: A continuously variable rate deep image compression framework," 2020, arXiv: 2003.02012v2 [eess.IV] (cit. on p. 23).

[44] Z. Cui, J. Wang, S. Gao, T. Guo, Y. Feng, and B. Bai, "Asymmetric gained deep image compression with continuous rate adaptation," in *Proc. IEEE/CVF CVPR*, 2022, pp. 10 532–10 541, arXiv: 2003.02012 [eess.IV] (cit. on p. 23).

[45] K. Tong, Y. Wu, Y. Li, K. Zhang, L. Zhang, and X. Jin, "QVRF: A quantization-error-aware variable rate framework for learned image compression," in *Proc. IEEE ICIP*, 2023, arXiv: 2303.05744 [eess.IV], pre-published (cit. on p. 23).

[46] T. Xue, B. Chen, J. Wu, D. Wei, and W. T. Freeman, "Video enhancement with task-oriented flow," *Int. J. Comput. Vis. (IJCV)*, vol. 127, pp. 1106–1125, 2019, arXiv: 1711.09078 [cs.CV] (cit. on p. 23).

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2015, arXiv: 1412.6980 [cs.LG] (cit. on p. 23).

[48] H. Fu, F. Liang, J. Lin, *et al.*, "Learned image compression with discretized Gaussian-Laplacian-Logistic mixture model and concatenated residual modules," *IEEE Trans. Image Process.*, vol. 32, pp. 2063–2076, 2023, arXiv: 2107.06463 [eess.IV] (cit. on p. 27).

[49] Y. Kang, J. Hauswald, C. Gao, *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, 2017, DOI: 10.1145/3037697.3037698 (cit. on pp. 29, 41).

[50] N. Shlezinger and I. V. Bajić, "Collaborative inference for AI-empowered IoT devices," *IEEE Internet of Things Magazine*, vol. 5, no. 4, pp. 92–98, Dec. 2022, arXiv: 2207.11664 [eess.SP] (cit. on p. 29).

[51] Z. Wu, S. Song, A. Khosla, *et al.*, "3D ShapeNets: A deep representation for volumetric shapes," in *Proc. IEEE/CVF CVPR*, 2015, pp. 1912–1920, arXiv: 1406.5670 [cs.CV] (cit. on pp. 29, 33).

[52] D. Maturana and S. A. Scherer, "VoxNet: A 3D convolutional neural network for real-time object recognition," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928, DOI: 10.1109/IROS.2015.7353481 (cit. on p. 29).

[53] G. Riegler, A. O. Ulusoy, and A. Geiger, "OctNet: Learning deep 3D representations at high resolutions," in *Proc. IEEE/CVF CVPR*, 2017, pp. 6620–6629, arXiv: 1611.05009 [cs.CV] (cit. on p. 29).

[54] C. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proc. IEEE/CVF CVPR*, 2017, pp. 77–85, arXiv: 1612.00593 [cs.CV] (cit. on pp. 29, 30, 32, 33, 35).

[55] C. Qi, L. Yi, H. Su, and L. J. Guibas, "PointNet++: Deep hierarchical feature learning on point sets in a metric space," in *Neural Information Processing Systems (NIPS)*, 2017, arXiv: 1706.02413 [cs.CV] (cit. on p. 29).

[56] H. Choi and I. V. Bajić, "Near-lossless deep feature compression for collaborative intelligence," in *Proc. IEEE MMSP*, 2018, pp. 1–6, arXiv: 1804.09963 [eess.IV] (cit. on pp. 30, 41).

[57] Y. Hu, S. Yang, W. Yang, L.-Y. Duan, and J. Liu, "Towards coding for human and machine vision: A scalable image coding approach," in *Proc. IEEE ICME*, 2020, pp. 1–6, arXiv: 2001.02915 [cs.CV] (cit. on p. 30).

[58] L. D. Chamain, F. Racapé, J. Bégaint, A. Pushparaja, and S. Feltman, "End-to-end optimized image compression for machines, a study," in *Proc. IEEE DCC*, 2020, pp. 163–172, arXiv: 2011.06409 [eess.IV] (cit. on p. 30).

[59] K. Mammou, P. A. Chou, D. Flynn, M. Krivokuća, O. Nakagami, and T. Sugio, "G-PCC codec description v2," ISO/IEC JTC1/SC29/WG11 N18189, 2019, [Online]. Available: https://mpeg.chiariglione.org/standards/mpeg-i/geometry-based-point-cloud-compression/g-pcc-codec-description-v2 (cit. on pp. 31, 34).

[60] N. Tishby, F. C. Pereira, and W. Bialek, "The information bottleneck method," in *Proc. 37th annual Allerton Conf. on Communication, Control, and Computing*, Sep. 1999, pp. 368–377, arXiv: physics/0004057 [physics.data-an] (cit. on p. 31).

[61] T. M. Cover and J. A. Thomas, "Elements of Information Theory," 2nd. Wiley, 2006, DOI: 10.1002/047174882X (cit. on p. 31).

[62] D. Flynn and K. Mammou, "MPEG-PCC-TMC13," 2021, [Online]. Available: https://github.com/MPEGGroup/mpeg-pcc-tmc13 (cit. on pp. 34, 37).

[63] Google, "Draco: 3D data compression," 2017, [Online]. Available: https://google.github.io/draco/ (cit. on pp. 34, 37).

[64] G. Bjøntegaard, "Calculation of average PSNR differences between RD-curves," in *ITU-T SC16/Q6 VCEG-M33*, Apr. 2001, pp. 2–4, [Online]. Available: https://www.itu.int/wftp3/av-arch/video-site/0104_Aus/VCEG-M33.doc (cit. on p. 35).

[65] Y. Matsubara, R. Yang, M. Levorato, and S. Mandt, "Supervised compression for resource-constrained edge computing systems," in *Proc. IEEE/CVF Winter Conf. on Applications of Computer Vision (WACV)*, 2022, pp. 923–933, arXiv: 2108.11898 [cs.CV] (cit. on p. 39).

[66] Z. Duan and F. Zhu, "Efficient feature compression for edge-cloud systems," in *Proc. Picture Coding Symposium (PCS)*, Dec. 2022, pp. 187–191, arXiv: 2211.09897 [eess.IV] (cit. on p. 39).

[67] N. Ahuja, P. Datta, B. Kanzariya, V. S. Somayazulu, and O. Tickoo, "Neural rate estimator and unsupervised learning for efficient distributed image analytics in split-DNN models," in *Proc. IEEE/CVF CVPR*, Jun. 2023, pp. 2022–2030, DOI: 10.1109/CVPR52729.2023.00201 (cit. on p. 39).

[68] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and F.-F. Li, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE/CVF CVPR*, 2009, pp. 248–255, DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 39).

[69] I. V. Bajić, W. Lin, and Y. Tian, "Collaborative intelligence: Challenges and opportunities," in *Proc. IEEE ICASSP*, to appear, 2021, arXiv: 2102.06841 [eess.IV] (cit. on p. 41).

[70] A. E. Eshratifar, M. S. Abrishami, and M. Pedram, "JointDNN: An efficient training and inference engine for intelligent mobile cloud computing services," *IEEE Trans. Mobile Computing*, 2019, Early Access, arXiv: 1801.08618 [cs.DC] (cit. on p. 41).

[71] **M. Ulhaq** and I. V. Bajić, "Shared mobile-cloud inference for collaborative intelligence," NeurIPS'19 demonstration, 2019, arXiv: 2002.00157 [cs.AI] (cit. on p. 41).

[72] H. Choi and I. V. Bajić, "Deep feature compression for collaborative object detection," in *Proc. IEEE ICIP*, Oct. 2018, pp. 3743–3747, arXiv: 1802.03931 [cs.CV] (cit. on p. 41).

[73] Z. Chen, K. Fan, S. Wang, L. Duan, W. Lin, and A. C. Kot, "Toward intelligent sensing: Intermediate deep feature compression," *IEEE Trans. Image Processing*, vol. 29, pp. 2230–2243, 2019, DOI: 10.1109/TIP.2019.2941660 (cit. on p. 41).

[74] ISO/IEC, "Draft call for evidence for video coding for machines," ISO/IEC JTC 1/SC 29/WG 11 W19508, Jul. 2020 (cit. on p. 41).

[75] S. R. Alvar and I. V. Bajić, "Multi-task learning with compressible features for collaborative intelligence," in *Proc. IEEE ICIP*, Sep. 2019, pp. 1705–1709, arXiv: 1902.05179 [cs.MM] (cit. on p. 41).

[76] H. Choi, R. A. Cohen, and I. V. Bajić, "Back-and-forth prediction for deep tensor compression," in *Proc. IEEE ICASSP*, 2020, pp. 4467–4471, arXiv: 2002.07036 [cs.LG] (cit. on p. 41).

[77] S. R. Alvar and I. V. Bajić, "Bit allocation for multi-task collaborative intelligence," in *Proc. IEEE ICASSP*, May 2020, pp. 4342–4346, arXiv: 2002.07048 [cs.LG] (cit. on p. 41).

[78] R. A. Cohen, H. Choi, and I. V. Bajić, "Lightweight compression of neural network feature tensors for collaborative intelligence," in *Proc. IEEE ICME*, Jul. 2020, pp. 1–6, arXiv: 2105.06002 [cs.LG] (cit. on p. 41).

[79] S. R. Alvar and I. V. Bajić, "Pareto-optimal bit allocation for collaborative intelligence," *arXiv:2009.12430*, Sep. 2020, arXiv: 2009.12430 [eess.IV] (cit. on p. 41).

[80] B. K. P. Horn and B. G. Schunck, "Determining optical flow," *Artificial Intelligence*, vol. 17, no. 1, pp. 185–203, 1981, DOI: https://doi.org/10.1016/0004-3702(81)90024-2 (cit. on p. 43).

[81] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE/CVF CVPR*, 2016, pp. 770–778, arXiv: 1512.03385 [cs.CV] (cit. on p. 46).

[82] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE/CVF CVPR*, 2017, pp. 2261–2269, arXiv: 1608.06993 [cs.CV] (cit. on p. 46).

[83] Wikipedia contributors, "Root-mean-square deviation — Wikipedia, the free encyclopedia," 2020, [Online]. Available: https://en.wikipedia.org/wiki/Root-mean-square_deviation (cit. on p. 46).

[84] H. Choi and I. V. Bajić, "Deep frame prediction for video coding," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 7, pp. 1843–1855, Jul. 2020, arXiv: 1901.00062 [eess.IV] (cit. on p. 48).

[85] ITU, "High efficiency video coding," Recommendation ITU-T H.265, Nov. 2019., [Online]. Available: https://handle.itu.int/11.1002/1000/14107 (cit. on p. 48).

[86] Y. Wang, J. Ostermann, and Y.-Q. Zhang, "Video Processing and Communications." Prentice-Hall, 2002, ISBN: 0130175471 (cit. on p. 49).