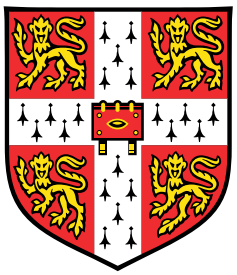
Compression without Quantization



Gergely Flamich

Department of Engineering
University of Cambridge

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

St John's College August 2019

Szüleimnek.

Declaration

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

Gergely Flamich
August 2019

Acknowledgements

And I would like to acknowledge ...

Abstract

There has been renewed interest in machine learning (ML) based lossy image compression, with recently proposed techniques beating traditional image codecs such as JPEG, WebP and BPG in perceptual quality on every compression rate. A key advantage of ML algorithms in this field are that **a)** they can adapt to the statistics of each individual image to increase compression efficiency much better than any hand-crafted method, and **b)** they can be used to very quickly develop efficient codecs for new media such as lightfield cameras, 360˚ images, Virtual Reality (VR), where classical methods would struggle, and where the develpment of efficient hand-crafted methods could take years.

In this thesis, we present an introduction to the field of neural image compression, first through the lens of image compression, then the lens of information theoretic neural compression. We examine how quantization is a fundamental block in the lossy image compression pipeline, and emphasize the difficulties it presents for gradient-based optimization techniques. We review recent influential developments in the field and see how they circumvent the issue of quantization in particular.

Our approach is different: we propose a generaly lossy compression framework that allows us to forgo quantization completely. We use this to develop a novel image compression algorithm using an extension of Variational Auto-Encoders (VAEs) called Probabilistic Ladder Networks (PLNs) and evaluate its efficiency compared to both classical and ML-based approaches on two of the currently most popular perceptual quality metrics. Surprisingly, with no fine-tuning, we achieve close to state-of-the-art performance on low bitrates while slightly underperforming on higher bitrates. Finally, we present analysis of important characteristics of our method, such as coding time and the effectiveness of our chosen model, and discuss key areas where our method could be improved.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

1.1 Motivation

There have been several exciting developments in neural image compression recently, and there now methods that consistently outperform classical methods such as JPEG, WebP and BPG (Toderici et al. (2017), Theis et al. (2017), Rippel and Bourdev (2017), Ballé et al. (2018), Johnston et al. (2018), Mentzer et al. (2018)).

The first advantage of ML-based image codecs, is that they can adapt to the statistics of each individual image much better than even the best hand-crafted methods. This allows them to compress images to much fewer bits, while retaining good perceptual quality. A second advantage is that they are generally far easier to adapt to new media formats, such as lightfield cameras, 360° images, Virtual Reality (VR), video streaming etc. The purposes of compression and the devices on which the encoding and decoding is performed varies greatly, from archiving terabytes of genetic data for later research on a supercomputer, through compressing images to be displayed on a blog or a news article to improve their loading time in a user's web browser, to streaming video on a mobile device. Classical methods are usually "one-size-fits-all", and their compression efficiency can severly degrade when attempting to compress media for which they were not designed. Designing good hand-crafted codecs is difficult, can take several years, and requires the knowledge of many experts. ML techniques on the other hand allow to create equally or better performing, and much more flexible codecs within a few months at significantly lower cost.

The chief limitation of current neural image compression methods is while most models these days are trained using gradient-based optmizers, quantization, a key step in the (lossy) image compression pipeline, is an inherently non-differentiable operation. Hence, all current methods need to resort to "tricks" and approximations so that the learning signal can still be passed through the whole model. A review of these methods will be presented in Chapter 2.

Our approach differs from virtually all previous methods in that we take inspiration from information theory (Rissanen (1986) and Harsha et al. (2007)) and neural network compression (Hinton and Van Camp (1993) and Havasi et al. (2018)) to develop a general lossy compression framework that allows us to forgo the quantization step in our compression pipeline completely. We then apply these ideas to image compression and demonstrate that our codec using Probabilisitic Ladder Networks (Sønderby et al. (2016)), an extension of Variational Auto-Encoders (Kingma and Welling (2013)), achieves close to state-of-the-art performance on the Kodak Dataset (Eastman Kodak Company (1999)) with no fine-tuning of our architecture.

1.2 Thesis Contributions

The contributions of our thesis are as follows:

1. A **comparative review** of recent influential works in the field of neural image compression.

2. The development of a **general lossy compression framework** that allows to forgo the quantization step in the compression pipeline, thus allowing end-to-end optimization of models using of gradient-based methods.

3. A **novel image compression algorithm** using our framework, that achieves close to state-of-the-art performance on the Kodak Dataset Eastman Kodak Company (1999) without any fine-tuning of model hyperparameters.

4. Three **sampling algorithms** for multivariate Gaussian distributions, that can be readily used in our compression framework.

1.3 Thesis Outline

Our thesis begins with an introduction to the field of neural image compression. We first review concepts in image compression in Section 1.4, such as lossless versus lossy compression, the rate-distortion trade-off and linear and non-linear transform coding. We emphasize the fundamental role quantization plays in virtually all previous approaches in image compression. In Section 1.5, we shift our focus to information theory, where we introduce the Minimum Description Length (MDL) Principle (Rissanen and Langdon (1981)) and the Bits-back Argument (Hinton and Van Camp (1993)). Taking inspiration from these, as well

as from Harsha et al. (2007) and Havasi et al. (2018), in Section 1.6 we develop a general framework for the lossy compression of data, and show how it is related to quantization.

In Chapter 2, we give a comparative review of recent influential developments in neural image compression. We examine their whole pipeline: the datasets used, their architectures, the "tricks" and approximations used to circumvent the non-differentiability of quantization, their coding methds, training procedures and evaluation methods.

In Chapter 3, we describe our proposed method. We explain our choice of dataset, and preprocessing steps. We give a detailed description of our model and why we chose it. We describe our training procedure, based on ideas from Sønderby et al. (2016), Higgins et al. (2017), Ballé et al. (2018) and Dai and Wipf (2019). Next, we present 3 "codable" sampling techniques, that can be used in our compression framework and point out their strengths and weaknesses.

Finally, in Chapter 4 we compare our trained models to current compression algorithms, both classical such as JPEG and BPG, and the current state-of-the-art neural methods (Ballé et al. (2018)). In particular, we compare these methods by their compression rates for a given perceptual quality as measured by the two most popular perceptual metrics, Peak Signal-to-Noise Ratio (PSNR) (Huynh-Thu and Ghanbari (2008)) and the Multi-scale Structural Similarity Index (MS-SSIM) (Wang et al. (2003)). We achieve close to state-of-the-art performance, with no fine-tuning of model hyperparameters. We also present some further analysis of our chosen models, to empirically justify their use, as well as to analyze some of the aspects that were not of primary concern of this work, such as coding speed.

1.4 Image Compression

The field of image compression is a vast topic that mainly spans over the fields of computer science and signal processing, but incorporates methods and knowledge from several other disciplines, such as mathematics, neuroscience, psychology and photography. In this section, we introduce the reader to the basics of the topic, starting with source coding, then through lossy compression we arrive at the concepts of rate and distrotion. Finally, we introduce transform coding, the category in which our work falls as well.

1.4.1 Source Coding

From a theoretical point of view, given some source S, a sender and a receiver, compression may be described as the aim of the sender communicating an arbitrary sequence $X_1, X_2, \ldots, X_n$ taken from S to the receiver in as few bits as possible, such that the receiver may recover relevant information from the message. If the receiver can always recover all the information from the message of the sender, we call the algorithm **lossless**, otherwise we call it **lossy**.

At first it might not seem intuitive to allow for lossy compression, and in some domains this is definitely true, e.g. in text compression. However, humans' audio-visual perception is neither completely aligned with the range of what can be digitally represented, nor does it always scale the same way (Eskicioglu et al. (1994), Huynh-Thu and Ghanbari (2008), Gupta et al. (2011)). Hence, there is a great opportunity for compressing media in a lossy way by discarding information with the change being imperceptible for a human observer, while making significant gains in size reduction.

1.4.2 Lossy Compression

As the medium of interest in lossy compression is generally assumed to be a real-valued vector $\mathbf{x} \in \mathbb{R}^N$, such as RGB pixel intensities in an image or frequency coefficients in an audio file, the usual pipeline consists of an encoder $C \circ \texttt{Enc}$, mapping a point $\mathbf{x} \in \mathbb{R}^N$ to a string of bits and a decoder mapping from bitstrings to some reconstruction $\hat{\mathbf{x}}$. The factor of the encoder $\texttt{Enc}$ can be understood as a map from $\mathbb{R}^N$ to a finite symbol set $\mathscr{A}$, called a **lossy encoder**, and C can be understood as a map from $\mathscr{A}$ to a string of bits called a **lossless code** (Goyal (2001)). We examine both $\texttt{Enc}$ and C in more detail in Section 1.4.5. The decoder then can be thought of as inverting the code first and then using an approximate inverse of $\texttt{Enc}$ to get the reconstruction $\hat{\mathbf{x}}$: $\texttt{Dec} \circ C^{-1}$. Given these it is of paramount importance to quantify

- The **distortion** of the compressor. On average, how closely does $\hat{\mathbf{x}}$ resemble $\mathbf{x}$?
- The **rate** of the compressor. On average, how many bits are required to communicate $\mathbf{x}$? We want this to be as low as possible of course.

1.4.3 Distortion

In order to measure "closeness" in the space of interest $\mathscr{X}$, a distance metric $d(\cdot, \cdot) : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is introduced. Then, the distortion D is is defined as

$$D = \mathbb{E}_{p(\hat{\mathbf{x}})} \left[d(\mathbf{x}, \hat{\mathbf{x}}) \right].$$

A popular choice of d, across many domains of compression is the normalized L_2 metric or Mean Squared Error (MSE), defined as

$$d(\mathbf{x}, \hat{\mathbf{x}}) = \frac{1}{N} \sum_{i}^{N} (x_i - \hat{x}_i)^2, \quad \mathcal{X} = \mathbb{R}^N.$$

It is a popular metric as it is simple, easy to implement and has nice interpretations in both the Bayesian (Bishop (2013)) and the MDL (Hinton and Van Camp (1993), to be introduced in Section 1.5.1) settings. In the image compression setting, however, the MSE is problematic, since optimizing for it does not necessarily translate to obtaining pleasant-looking reconstructions (Zhao et al. (2015)). Hence, more appropriate, so-called *perceptual metrics* were developed. The two most common ones used today are Peak Signal-to-Noise Ratio (PSNR) (Huynh-Thu and Ghanbari (2008), Gupta et al. (2011)) and the Structural Similarity Index (SSIM) (Wang et al. (2004)) and its multiscale version (MS-SSIM) (Wang et al. (2003)). Crucially, these two metrics are also differentiable, thus they lend themselves for gradient-based optimization.

1.4.4 Rate

We noted above that the code used after the lossy encoder is lossless. To further elaborate, in virtually all cases it is an **entropy code** (Goyal (2001)). This means that we assume that each symbol in the representation $\mathbf{z} = \texttt{Enc}(\mathbf{x})$ has some probability mass $P(z_i)$. A fundamental result by Shannon states that $\mathbf{z}$ may not be encoded losslessly in fewer than $H[\mathbf{z}]$ nats:

Theorem 1 *(Proven by Shannon and Weaver (1998), presented as stated in MacKay et al. (2003)) N i.i.d. random variables each with entropy $H[X]$ can be compressed into more than $N \cdot H[X]$ bits with negligible risk of information loss, as $N \to \infty$; conversely if they are compressed into fewer than $NH[X]$ bits it is virtually certain that information will be lost.*

Entropy codes, such as Huffman codes (Huffman (1952)) or Arithmetic Coding (Rissanen and Langdon (1981)) can get very close to this lower bound. We discuss coding methods further in Section 3.4.

In particular, entropy codes can compress each symbol z_i in $-\log P(z_i)$ nats.

The rate (in nats) of the compression algorithm is defined as the average number of nats required to code a single dimension of the input, i.e.

$$R = \frac{1}{N} H[\mathbf{z}].$$

1.4.5 Transform Coding

The issue with source coding is that coding $\mathbf{x}$ might have a lot of dependencies across its dimensions. For images, this manifests on multiple scales and semantic levels, e.g. a pixel being blue might indicate that most pixels around it are blue as the scene is depicting the sky or a body of water; a portrait of a face will also imply that eyes, a nose and mouth are probably present, etc. Modelling and coding this dependence structure in very high dimensions is challenging or intractable, and hence we need to make simplifying assumptions about it to proceed.

Transform coding attempts to solve the above problem by decomposing the encoder function $\texttt{Enc} = Q \circ T$ into a so-called **analysis transform** T and a **quantizer** Q. The idea is to transform the input into a domain, such that the dependencies between the dimensions are removed, and hence they can be coded individually. The decoder inverts the steps of the encoder, where the inverse operation of T is called the **synthesis transform** (Gupta et al. (2011)).

In *linear transform coding*, T is an invertible linear trasformation, such as a discrete cosine transformation (DCT), as it is in the case of JPEG (Wallace (1992)), or discrete wavelet transforms in JPEG 2000 (Rabbani and Joshi (2002)). While simple, fast and elegant, linear transform coding has the key limitation that it can only at most remove correlations (i.e. first-order dependencies), and this can severly limit its efficiency (Ballé et al. (2016a)). Instead, Ballé et al. (2016a) propose a method for *non-linear transform coding*, where T is replaced by a highly non-linear transformation, and its inverse is now replaced by an approximate inverse, which is a separate non-linear transformation. Both T and its approximate inverse are learnt, and the authors show that with a more complicated transformation they can easily surpass the performance of the much more fine-tuned JPEG codecs.

Our work also falls into this line of research, although with signifcant differences, which will be pointed out later.

1.4.6 The Significance of Quantization in Lossy Compression

The reason why quantization is required in lossy compression algorithms, is because it allows to reduce the information content of data. To study the precise meaning of this, we put

the problem in a formal setting. Let us define the quantizer as a function $[\cdot] : R \to S$, where R is the original representation space (in transform coding this would be the image $T(\mathbb{R}^N)$), and a quantized space S (usually $\mathbb{Z}^N$). $[\cdot]$ is always many-to-one mapping. Let $[s]^{-1} = \{x \in R \mid [x] = s\}$ be the preimage of s. Then, we have the further requirement on $[\cdot]$ that the fibres of S partition R, i.e.

$$\text{if } s \neq t \Rightarrow [s]^{-1} \cap [t]^{-1} = \varnothing.$$

A popular option for the quantizer is the rounding function, mapping $[\cdot] : \mathbb{R} \to \mathbb{Z}$, where for each integer $z \in \mathbb{Z}$ it is defined as $x \in \left[z - \frac{1}{2}, z + \frac{1}{2}\right) \mapsto z$. Given some probability mass $P(x)$ for some data x, we have seen that using entropy coding x can be encoded in $-\log P(x)$ nats. The way quantization enables better compression, is that it aggregates the probability mass of all elements in $[s]^{-1}$ into the mass of s. Namely, for each s, the quantizer induces a new probability mass function $\hat{Q}(s)$, such that

$$\hat{Q}(s) = \int_{[s]^{-1}} p(x)\,\mathrm{d}x,$$

where the integral is replaced by summation for discrete $[s]^{-1}$. This will allow us to code x in potentially much fewer nats. To put it precisely, assume $[x] = s$, then

$$-\log \hat{Q}(s) = -\log \int_{[s]^{-1}} p(x)\,\mathrm{d}x \leq -\log P(x).$$

This is at the cost of introducing distortion (see Section 1.4.3), as we will not be able to reconstruct x from s. In particular, quantization is vital for continuous x, as the probability mass of each individual x is 0, and hence we would require $-\log P(x) = \infty$ nats to encode them without quantization.

1.5 Theoretical Foundations

We now shift our focus from image compression to the foundations of neural compression. We begin with the Minimum Description Length (MDL) Principle (Rissanen (1986)) and the Bits-Back Argument (Hinton and Van Camp (1993)), the two core theoretical guiding principles of this work. We then see how based on these, as well as on more recent work (Harsha et al. (2007), Havasi et al. (2018)) we can develop a general ML-based compression framework that does not include quantization in its pipeline, thus allowing gradient-based optimization methods to be used in training our compression algorithms.

1.5.1 The Minimum Description Length Principle

Our approach is based on the Minimum Description Length (MDL) Principle (Rissanen (1986)). In essence, it is a formalization of Occam's Razor, i.e. the simplest model that describes the data well is the best model of the data (Grünwald et al. (2007)). Here, "simple" and "well" need to be defined, and these definitions are precisely what the MDL principle gives us. Informally, it asserts that given a class of hypotheses $\mathscr{H}$ (e.g. a certain statistical model and its parameters) and some data $\mathscr{D}$, if a particular hypothesis $H \in \mathscr{H}$ can be described with at most $L(H)$ bits and the using the hypothesis the data can be described with at most $L(\mathscr{D} \mid H)$ bits, then the minimum description length of the data is

$$L(\mathscr{D}) = \min_{H \in \mathscr{H}} \{L(H) + L(\mathscr{D} \mid H)\}, \tag{1.1}$$

and the best hypothesis is an H that minimizes the above quantity.

Crucially, the MDL principle can thus be interpreted as telling us that **the best model of the data is the one that compresses it the most**. This makes Eq 1.1 a very appealing learning objective for optimization-based compression methods, ours included. Below, we briefly review how this has been applied so far and how it translates to our case.

1.5.2 The Bits-back Argument

Here we present the bits-back argument, introduced in Hinton and Van Camp (1993). The main goal of their work was to develop a regularisation technique for neural networks, and while they talk about the compression of the model, the first method that realized bits-back efficiency came much later, developed by Havasi et al. (2018). Although the argument is essentially just the direct application of the MDL principle, it can seem quite counter-intuitive at first. Hence, we begin this section with an example to illustrate the goal of the argument, and only then move on to formulate it in more generality.

Example Let us be given a simple regression problem on the dataset $\mathscr{D} = (\mathscr{X}, \mathscr{Y})$, where $\mathscr{X} = (x_1, \ldots, x_n)$, $\mathscr{Y} = (y_1, \ldots, y_n)$ are both one dimensional input and target sets and (x_i, y_i) are a corresponding training pair. Assume we wish to fit a simple model:

$$\hat{y} = f(x) = \alpha x + \beta,$$

where we wish to learn the parameters α and β. Assuming a Gaussian likelihood with mean 0 and variance 1 on the residuals $\delta = y - \hat{y}$,

$$p(\delta \mid x, \alpha, \beta) = \mathcal{N}(\delta \mid 0, 1),$$

a popular way of fitting the model is using Maximum Likelihood Estimation (MLE), i.e. maximizing $\prod_i p(\delta_i \mid x_i, \alpha, \beta)$ which is equivalent to minimizing the negative logarithm of this quantity, $-\sum_i \log p(\delta_i \mid x_i, \alpha, \beta)$. It can be easily seen that this works out to be equivalent to minimizing the Mean Squared Error (MSE) between the predicted values and the targets:

$$L(\mathcal{D} \mid \alpha, \beta) = \frac{1}{n} \sum_i (y_i - f(x_i))^2.$$

A usual issue with MLE algorithms is that they are heavily overparameterized for the problem they are supposed to be solving, and hence can easily overfit (this is most likely not an issue with our toy model, but we shall pretend for the sake of the argument). In order to solve this issue, a standard technique is to introduce some regularisation term to the loss. Here we are intereseted in applying the MDL principle directly.

Before we discuss how it is applied, we must make precise the setting in which it *can* apply. In particular, the MDL principle assumes the form of a communications problem. Assume two parties, Alice and Bob share $\mathcal{X}$, and some other arbitrary pre-agreed information, but only Alice has access to $\mathcal{Y}$. Then, the MDL principle asks for the minimal message that Alice needs to send to Bob, such that he may recover $\mathcal{Y}$ completely. With this setup in mind, we can continue.

In order to apply the MDL principle, we need to be able to calculate the MDLs of the data given a hypothesis and the MDLs of our hypotheses. Notice, that the former is in fact already available in the form of the MSE for a given hypothesis, and hence $L(\mathcal{D} \mid \alpha, \beta)$ is not an overload of notation. In order to code the hypothesis (the pair (α, β) in our case), we need to define two distributions over our parameters: a prior P_θ, that gives us the regularizing effect and stays fixed, and a posterior Q_ϕ, the distribution that we learn and assume that our parameters actually come from it. We use the θ and ϕ to denote the sufficient statistics of the prior and posterior, respectively. Now, learning changes, as we are no longer optimizing a single hypothesis (α, β), but a whole class of hypotheses $Q_\phi(\alpha, \beta)$, by finding the best fitting set of sufficient statistics ϕ for our dataset. Thus, our initial data description length now becomes an expectation over the possible hypotheses:

$$L(\mathcal{D} \mid \phi) = \mathbb{E}_{Q_\phi}\left[L(\mathcal{D} \mid \alpha, \beta)\right].$$

Defining the regularizing term, however, turns out to be trickier than expected, and lies at the core of the bits-back argument. We seek to find the minimum description length of a hypothesis (α, β). Using a Q_ϕ, we know we can encode a concrete hypothesis in $-\log Q_\phi(\alpha, \beta)$ nats, and thus a reasonable first guess for the MDL would be

$$\mathbb{E}_{Q_\phi}\left[-\log Q_\phi(\alpha, \beta)\right], \tag{1.2}$$

i.e. the Shannon entropy of Q_ϕ. This turns out to be wrong, however, for the reason that Bob should be able to decode Alice's message, and since he does not have access to Q_ϕ, he cannot do this. At this point, we note, that as P_θ is fixed, we may assume that Alice and Bob share it a priori. This allows us to code a pair (α, β) in $-\log P_\theta(\alpha, \beta)$ nats that Bob can definitely decode, and hence a reasonable second guess for the MDL could be

$$\mathbb{E}_{Q_\phi}\left[-\log P_\theta(\alpha, \beta)\right], \tag{1.3}$$

i.e. the cross entropy between Q_ϕ and P_θ, which is also the expected length of the actual message that gets sent to communicate the parameters. Note, that since the hypotheses are still drawn from Q_ϕ, the expectaion needs to be taken over it. This also turns out to be wrong, as the bits-back argument shows that *not all* of the bits used for the message are used to code Q_ϕ. Once the parameters are sent, Alice also sends every residual δ_i, obtained by using the parameter set sent to Bob.

Once Bob has decoded (α, β) and each δ_i, he can fully recover each y_i by calculating $x_i + \delta_i$. Now, since he has access to both $\mathcal{X}$, $\mathcal{Y}$ and P_θ, he may also fit a Q_ψ to the data, using the same learning algorithm as Alice used to fit her Q_ϕ. The key observation in (Hinton and Van Camp (1993)) is that so long as this learning algorithm is *deterministic*, after sufficient training Bob can achieve $\psi = \phi$, i.e. he recovers Alice's posterior distribution. This means that Bob is able to sample the same (α, β) pair that was sent to him (e.g. by also sharing a random seed with Alice either before their communication or during, at at most an $\mathcal{O}(1)$ cost, which is negligible).

> This must mean, that Alice not only communicated Q_ϕ itself to Bob, but also the *random bits* that were used in conjunction with Q_ϕ to draw the sample (α, β).

The fact that Alice has communicated both Q_ϕ and the random bits in a $-\log P_\theta(\alpha, \beta)$ nat long message, means that in order to get the cost of communicating Q_ϕ only, we simply need to subtract the length of the random bits. But since (α, β) were drawn from Q_ϕ, their length is going to be precisely $-\log Q_\phi(\alpha, \beta)$. Hence, the expected hypothesis description

length is the expectation of this difference, namely

$$\mathbb{E}_{Q_\phi}\left[-\log P_\theta(\alpha,\beta) - (-\log Q_\phi(\alpha,\beta))\right] = \mathbb{E}_{Q_\phi}\left[\log \frac{Q_\phi(\alpha,\beta)}{P_\theta(\alpha,\beta)}\right] = \mathrm{KL}\left[\, Q_\phi \;||\; P_\theta \,\right].$$

Above the rightmost term is called the **Kullback-Leibler Divergence** between Q_ϕ and P_ϕ. It is defined as

$$\mathrm{KL}\left[\, Q \;||\; P \,\right] = \sum_{x\in\Omega} Q(x) \log \frac{Q(x)}{P(x)}$$

for probability mass functions Q and P, where Ω denotes the sample space, and

$$\mathrm{KL}\left[\, q \;||\; p \,\right] = \int_\Omega q(x) \log \frac{q(x)}{p(x)}\,\mathrm{d}x$$

for probability density functions q and p.

The fact that Bob can "get the random bits back "used in sampling the hypothesis is the namesake of the argument.

The general argument We are now ready to state the general bits-back argument. Assume Alice has trained a model for a regression problem, on a dataset $\mathscr{D} = (\mathscr{X}, \mathscr{Y})$, with training pairs $(\mathbf{x}_i, \mathbf{y}_i)$, and shares $\mathscr{X}$ with Bob. Her model has parameters $\mathbf{w}$, with prior $p_\theta(\mathbf{w})$, and uses the likelihood function $p(\mathbf{y} \mid \mathbf{w}, \mathbf{x})$, both shared with Bob. Assume that Alice has a learned posterior $q_\phi(\mathbf{w} \mid \mathscr{D})$ over the weights, and now wishes to communicate the targets $\mathscr{Y}$ to Bob.

Then, the bits-back argument states that if Alice acts according to the MDL principle, then she can communicate q_ϕ to Bob in $\mathrm{KL}\left[\, q_\phi \;||\; p_\theta \,\right]$ nats, as follows:

1. Alice draws a random sample $\hat{\mathbf{w}} \sim q_\phi(\mathbf{w})$. This represents a message of $-\log q_\phi(\hat{\mathbf{w}})$ nats.

2. $\hat{\mathbf{w}}$ is then used to calculate the residuals $\mathbf{r}_i$ between the model's output and the targets.

3. $\hat{\mathbf{w}}$ is coded using its prior p_θ, and sent to Bob alongside the residuals $\mathbf{r}_i$. The total length of the message that contains the posterior information is hence $-\log p_\theta(\hat{\mathbf{w}})$.

4. Bob, decodes $\hat{\mathbf{w}}$ using the same prior p_θ. He then recovers all targets $\mathscr{Y}$ by adding each $\mathbf{r}_i$ to his model's output with parameters set to $\hat{\mathbf{w}}$ upon input $\mathbf{x}_i$.

5. He then trains his model using the same deterministic algorithm as Alice did, to recover Alice's posterior q_ϕ. Hence, the random bits that were used to communicate the sample must be deducted from the cost of communicating q_ϕ. The cost of these bits

is precisely $-\log q_\phi(\hat{\mathbf{w}})$. Taking the expectation of the difference w.r.t. q_ϕ, the total cost of communicating q_ϕ is

$$\mathbb{E}_{q_\phi}\left[\log q_\phi(\mathbf{w}) - \log p_\theta(\mathbf{w})\right] = \mathrm{KL}\left[\, q_\phi \;||\; p_\theta \,\right].$$

Caveats of the argument Note, that the original argument merely derives the minimum description length for the weights $\mathbf{w}$, but clearly does not achieve it (as we have to send a message whose expected length is $\mathbb{E}_{q_\phi}\left[-\log p_\theta(\mathbf{w})\right]$). The authors merely state that these bits can be "recovered", and propose that a "free" auxiliary message might be coded in them, but do not give any propositions as to how sending these bits in the first place might be avoided. Nonetheless, as the notion of bit-back efficiency has expanded in recent years, it is customary to call any method *bits-back efficient* that transmits some information in $\mathrm{KL}\,[\, q \;||\; p \,]$ nats, for some posterior q and prior p over the information.

1.6 Compression without Quantization

In this section, we present a general framework for lossy data compression, based on the arguments presented above, as well as the works of Harsha et al. (2007) and Havasi et al. (2018).

As mentioned at the end of the previous section, the bits-back argument postulates that communicating the distribution of the parameter set of a model may be achieved in $K = \mathrm{KL}\,[\, q(\mathbf{w}) \;||\; p(\mathbf{w}) \,]$ nats, where q and p are the posterior and prior over the parameters, respectively. However, they do not give a method for achieving this, rather they show that only K nats are used to communicate the posterior in a longer message. Furthermore, the original MDL setup also requires to send the residuals from the model output.

For compression, however, we are only interested the communicating a sample and not its distribution, though still at bits-back efficiency. The correct communication problem for this was formulated by Harsha et al. (2007), and it is as follows:

> Let X and Y be two correlated random variables, with sample spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively, and with joint distribution $p(X, Y) = q(Y \mid X)p(Y)$. Given a concrete $x \in \mathcal{X}$, what is the minimal message Alice needs to send to Bob, such he can generate a sample accorting to the distribution $q(Y \mid X = x)$?

We can interpret $\mathcal{X}$ as the set of all data that we might wish to compress (e.g. the set of all RGB-coded natural images, the set of all MP3 coded audio files, etc.), and $\mathcal{Y}$ as the set of latent codes of the data, from which we may obtain our lossy reconstruction.

The solution to the above problem requires essentially the same mild assumptions the bits-back argument does, namely that Alice and Bob are allowed to share a fixed prior $p(Y)$ on the latent codes, as well as the seed used for their random generators. The significance of the latter assumption is that Alice and Bob will be able to reconstruct the same sequence of random numbers. Given these assumptions, Harsha et al. (2007) propose a rejection sampling algorithm to sample from $q(Y \mid X = x)$ using $p(Y)$, depicted in Algorithm 4 in the Appendix. Alice uses this algorithm to sample q, but she also keeps track of the number of proposals made by the algorithm. Once Alice's algorihtm accepts a proposal from p, it is sufficient for Alice to communicate the sample's index K to Bob. Bob can then obtain the desired sample from q, by simply drawing K samples from p, and since he can generate the same K samples as Alice did, the Kth sampe he draws is going to be an exact sample from q. Clearly, the communication cost of K is $\log K$ nats. Harsha et al. (2007) then also prove the following result.

Theorem 2 *(Harsha et al. (2007)) Let X and Y be random variables as given above. And let the communication problem be set as above. Let $T[X : Y]$ denote the MDL (in nats) of a sample $Y = y \sim q(Y \mid X = x)$. Then,*

$$I[X : Y] \leq T[X : Y] \leq I[X : Y] + 2 \log [I[X : Y] + 1] + \mathcal{O}(1), \tag{1.4}$$

where $I[X : Y]$ is called the mutual information between X and Y, and is defined as

$$I[X : Y] = \mathbb{E}_{p(X)} [\mathrm{KL} [\, q(Y \mid X) \,||\, p(Y) \,]]$$

Furthermore, $\log K$*, given by Algorithm 4, achieves the upper bound in Eq 1.4.*

The above theorem tells us that while in the classical sense bits-back efficiency is the best that we can do, it also tells us that we can get very close to it. Hence, from now on, we shall refer to any algorithm that achieves this tight upper bound as bits-back efficient as well.

To translate this to a general ML-based compression framework, we shall switch to notation more common in statistical modelling, concretely, we shall denote our data by $\mathbf{x}$ and the latent code $\mathbf{z}$. Now, let us assume a generative model over these variables, $p(\mathbf{x}, \mathbf{z}) = p(\mathbf{x} \mid \mathbf{z}) p_\theta(\mathbf{z})$, where $p(\mathbf{x} \mid \mathbf{z})$ is the data likelihood, and $p_\theta(\mathbf{z})$ is the prior over the latent code, with sufficient statistics θ. Let us also assume an approximate posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ over the latent code, with sufficient statistics ϕ. Then our framework is as follows:

1. Given some dataset $\mathscr{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ where the training examples are distributed according to $p(\mathbf{x})$, we fit our generative model to it, by fitting θ and ϕ using the (weighted) MDL objective:

$$\begin{gathered} L(\mathscr{D}) = \mathbb{E}_{p(\mathbf{x})}\left[L(\mathbf{x})\right], \\ \text{where } L(\mathbf{x}) = \mathbb{E}_{q_\phi}\left[L(\mathbf{x} \mid \mathbf{z}) + L(\mathbf{z})\right] = -\mathbb{E}_{q_\phi}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] + \beta \mathrm{KL}\left[\, q_\phi \;||\; p_\theta \,\right]. \end{gathered} \tag{1.5}$$

   This training objective is well known in the neural generative modelling literature as the Evidence Lower Bound (ELBO) (Kingma and Welling (2013), Higgins et al. (2017)). The expectation over $p(\mathbf{x})$ is usually taken over randomly drawn mini-batches from $\mathscr{D}$ using Stochastic Gradient Descent (SGD). Here β is a hyperparameter that can be set to trade off a smaller description length at the cost of worse reconstruction, or the other way around, thus allowing the user to reach different points on the rate distortion curve. See Section 1.6.2 for the derivation of Eq 1.5 and discussion on its validity.

2. Once θ and ϕ have been learned, we fix them (equivalent to sharing them with Bob in the communication problem).

3. Now, if we wish to compress some new data $\mathbf{x}'$, use a bits-back efficient sampling algorithm (such as Algorithm 4) to sample $q(\mathbf{z} \mid \mathbf{x}')$ using $p(\mathbf{z})$, and use the code output of the sampling algorithm as the compression code, along with the random seed that was used to obtain the sample. We shall refer to such algorithms as **coded sampling algorithms**.

4. To decompress, since we always have access to the fixed prior p_θ, and we have the random seed the compressing party used, we may run the coded sampling algorithm in "decode" mode to recover the sample $\mathbf{z}'$ from q_ϕ. Finally, we may run the reconstruction transformation of our generative model to recover a lossy reconstruction $\hat{\mathbf{x}}'$.

This framework is inspired by the work of Havasi et al. (2018), where they used a very similar framework to achieve state-of-the-art weight compression in Bayesian Neural Networks.

In this thesis, we use this framework to train β-VAEs as our choice of generative models, and demonstrate the efficiency of our method compared to the state-of-the-art in neural compression. More details on this will be given in Chapter 3.

1.6.1 Relation of Quantization to Our Framework

We present a similar argument to the one given in Havasi et al. (2018). Recall the original representation space R and quantized space S of a quantizer $[\cdot]$. Recall also the Kronecker delta function on x, defined as

$$\delta_x(y) = \begin{cases} 1 & \text{if } y = x \\ 0 & \text{otherwise.} \end{cases}$$

Given a particular $x \in R$, we have seen that quantization allows us to code it in $-\log \hat{Q}([x])$ nats. If we manipulate this term slightly, we get

$$\begin{aligned} -\log \hat{Q}([x]) &= \sum_{s \in S} \left[-\delta_{[x]}(s) \log \hat{q}([x]) + \underbrace{\delta_{[x]}(s) \log \delta_{[x]}(s)}_{=0} \right] \\ &= \sum_{s \in S} \delta_{[x]}(s) \log \frac{\delta_{[x]}(s)}{\hat{q}([x])} \\ &= \mathrm{KL}\left[\delta_{[x]} \;||\; \hat{Q} \right]. \end{aligned}$$

This shows that quantization of a deterministic parameter set is also bits-back efficient, with the posterior distribution family restricted to point masses. Thus the clear advantage of our framework comes from the fact that we allow much more posteriors than point masses.

1.6.2 Derivation of the Training Objective

In this section, we present the derivation of Eq 1.5. Thus, let our likelihood $p(\mathbf{x} \mid \mathbf{z})$, our latent prior $p_\theta(\mathbf{z})$ and approximate posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ be given. Then, given a budget of C nats, we want to optimize the following constrained objective on the description lengths:

$$\mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{q_\phi(\mathbf{z})} \left[-L(\mathbf{x} \mid \mathbf{z}) \right] \right] \quad \text{subject to } \mathbb{E}_{p(\mathbf{x})} \left[L(\mathbf{z}) \right] < C.$$

As we have seen in the sections above, these quantities can be replaced by

$$\mathbb{E}_{p(\mathbf{x})} \left[\mathbb{E}_{q_\phi(\mathbf{z})} \left[\log p(\mathbf{x} \mid \mathbf{z}) \right] \right] \quad \text{subject to } \mathbb{E}_{p(\mathbf{x})} \left[\mathrm{KL}\left[q_\phi(\mathbf{z} \mid \mathbf{x}) \;||\; p_\theta(\mathbf{z}) \right] \right] < C. \tag{1.6}$$

As we want to use gradient-based optimization of our models, we need to find a continuous relaxation of Eq 1.6. To this end, we rewrite the terms inside the "outer" expectation as their Lagranagian relaxation under the KKT conditions (Karush (2014), Kuhn and Tucker

(2014), Higgins et al. (2017)) and get:

$$\mathscr{F}(\theta, \phi, \beta, \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] - \beta(\mathrm{KL}\left[\, q_\phi(\mathbf{z} \mid \mathbf{x}) \;||\; p_\theta(\mathbf{z})\,\right] - C).$$

By the KKT conditions if $C \geq 0$ then $\beta \geq 0$, hence discarding the last term in the above equation will provide a lower bound for it:

$$\mathscr{F}(\theta, \phi, \beta, \mathbf{x}) \geq \mathscr{L}(\theta, \phi, \beta, \mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right] - \beta\mathrm{KL}\left[\, q_\phi(\mathbf{z} \mid \mathbf{x}) \;||\; p_\theta(\mathbf{z})\,\right].$$

Finally, taking the expectation over this again gives

$$\begin{aligned}
\mathbb{E}_{p(\mathbf{x})}\left[\mathscr{L}(\theta, \phi, \beta, \mathbf{x})\right] &= \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right]\right] - \beta\mathbb{E}_{p(\mathbf{x})}\left[\mathrm{KL}\left[\, q_\phi(\mathbf{z} \mid \mathbf{x}) \;||\; p_\theta(\mathbf{z})\,\right]\right] \\
&= \mathbb{E}_{p(\mathbf{x})}\left[\mathbb{E}_{q_\phi(\mathbf{z})}\left[\log p(\mathbf{x} \mid \mathbf{z})\right]\right] - \beta I[\mathbf{x} : \mathbf{z}] \\
&= \mathbb{E}_{p(\mathbf{x})}\left[L(\mathbf{x})\right].
\end{aligned} \tag{1.7}$$

This is the training objective of β-VAEs first derived in Higgins et al. (2017), although we note that it is applicable any generative model where the assumed conditions are present.

An important caveat of the above formulation is that the samples from $p(\mathbf{x})$ should comparable, in the sense the initial hard optimization objective of setting an average nat budget is reasonable. In the case of image data, if all images are the same size, this is fine, as our continuous relaxation will allow for images with high information content to have slightly longer code lenghts than C nats and ones with low information content will have shorter lengths. However, if we used different sized images during training, it would be less justified to set the same average code budget for, say, a 200×300 pixel image and a 2000×2000 image, as the latter will naturally contain more information than the former. Hence in this case it would be more reasonable to make the budget a function of the number of pixels the image contains, although this might make the formulation of the training objective much harder.

An approach taken in all neural image compression methods we examined is instead to train on random, but equal-sized patches extracted from each training image. While other works do this to make training more computationally feasible. As far as we are aware, we are the first ones to argue that this practise is not only convenient, but mandatory for the training procedure to be sound.

Chapter 2

Related Works

In this chapter, we give a brief overview of the history of ML-based image compression. Then, we focus on recent advances in lossy neural image compression and describe and compare them against each other.

2.1 Machine Learning-based Image Compression

Neural image compression goes back to at least as far as the early 1980s (Mougeot et al. (1991), Jiang (1999)). These methods very closely resemble in high-level structure to contemporary methods, in that they were designed as transform coding methods. In particular, virtually all early methods used some flavour of linear auto-encoders (LAEs, no non-linearities applied on the hidden layer), with a single-layer encoder and decoder (Jiang (1999)). As convolutional layers had not yet been available back then, all architectures were fully connected, and relied on splitting up images into equal-sized blocks and feeding them block-by-block to the LAE. Most methods optimized the MSE between the reconstruction and the orginal image, although different learning objectives were also explored (Mougeot et al. (1991)). Furthermore, the issue of quantization was not formally addressed. While the pipeline was to quantize the hidden layer activations of the LAE, no explicit treatment of the distortion introduced by quantization was given (Jiang (1999)).

A notable early example of a non-neural ML-based practical compression method is DjVu (Bottou et al. (1998)), which focused on segmenting foreground and background in documents and using K-means clustering to for the analysis transforming followed by entropy coding the background.

More recently, the work of Denton et al. (2015), Gregor et al. (2015) focused on discovering compressive representations using auto-encoders on low-resolution images, using datasets such as CIFAR-10 (Krizhevsky et al. (2009)). Toderici et al. (2015) proposed an

RNN-based auto-encoder for compressing 32×32 thumbnails and outperformed classical methods such as JPEG and WebP on these sizes. Their method has been later extended in Toderici et al. (2017) for large-scale images.

2.2 Comparison of Recent Works

In this section we focus on compression methods that allow for the compression of arbitrary-sized images. The most notable recent works in this area (that we are aware of) are Ballé et al. (2016b), Toderici et al. (2017), Theis et al. (2017), Rippel and Bourdev (2017), Ballé et al. (2018), Johnston et al. (2018) and Mentzer et al. (2018). Of these, Ballé et al. (2016b), Theis et al. (2017), Rippel and Bourdev (2017) and Ballé et al. (2018) are closest to our work and thus we present a review of these below.

Note on Notation: In this chapter, we denote the output of the encoders by $\mathbf{z}$, their quantized values by $\hat{\mathbf{z}}$ and their continuous relaxations by $\tilde{\mathbf{z}}$.

2.2.1 Datasets and Input Pipelines

Somewhat surprisingly, it appears that there appears to be no canonical dataset yet for general lossy neural image compression. Such a dataset should be comprised of a set of high-resolution, variable-sized losslessly encoded colour images, although the CLIC Dataset (CLIC (2018)) seems to be an emerging one. Perhaps the reason is that generally in other domains, such as image-based classification, cropping and rescaling images can effectively side-step the need to deal with variable-sized images. However, when it comes to compression, if we hope to build anything useful, side-stepping the size issue is not an option.

Ballé et al. (2016b) trained on 6507 images, selected from ImageNet Deng et al. (2009). They removed images with excessive saturation and since their method is based on dithering, they added unifrom noise to the remaining images to imitate the noise introduced by quantization. Finally, they downsampled and cropped images to be 256×256 pixels in size. They only kept images whise resampling factor was 0.75 or less, in order to avoid high frequency noise.

Theis et al. (2017) used 434 high resolution images from flickr.com under the creative commons license. As `flickr` store its images as JPEGs, they downsampled all images to be below 1536×1536 in resolution and saved them as PNGs in order to reduce the effects

of the lossy compression. Then, they extracted several 128×128 patches from each image and trained on those.

Rippel and Bourdev (2017) took images from the Yahoo Flickr Creative Commons 100 Million dataset, with 128×128 patches randomly sampled from the images. They do not state whether they used the whole dataset or just a subset, neither do they describe further preprocessing steps.

Ballé et al. (2018) scraped ≈ 1 million colour JPEG images of dimensions at most 3000×5000. They filtered out images with excessive saturation similarly to Ballé et al. (2016b). They also downsampled images by random factors such that the image's height and width stayed above 640 and 1200 pixels, respectively. Finally, they use several randomly cropped 256×256 pixel patches extracted from each image.

A clear trend is downsampling large, lossy-encoded images to get rid of the compression artifacts as an easy way of obtaining reasonable "approximations" of losslessly encoded images. Another trend is that (as we see in the next section) since all architectures are fully-convolutional, it is sufficient to train on small image patches to train the convolution kernels to speed up training, as well as to heavily "increase" the training set size.

Datasets for testing In contrast to the lack of datasets for training, all authors have used standard datasets for testing their methods, taken from classical lossy image compression research. These are the Kodak (Eastman Kodak Company (1999)) and Tecnick (Asuni and Giachetti (2014)) datasets. In order for our results to be comparable to these methods, we have also decided to test our method on the Kodak dataset.

2.2.2 Architectures

This is the most diverse aspect of recent approaches, and so we will only discuss them on a high level. We incorporated ideas from all of these papers as well as others into our work, we discuss these in Chapter 3. All architectures (including ours) realize a form of non-linear transform coding. This means all architectures will have an *analysis transform* or *encoder*, and a *synthesis transform* or *decoder* (see Section 1.4.5), both of whose parameters the methods learn using gradient descent.

All methods achieve the ability to deal with arbitrary-sized images by only utilizing convolutional and deconvolutional layers as their linear transformations. This leads to the very natural consequence that the number of latent dimensions, and thus the latent code length

increases linearly in the number of pixels of the input image. They all utilise downsampling after some convolutions. Every work that gives details on how they perform downsampling do it by using a stride larger than 1 on the convolutions, and it is reasonable to assume that the rest do it likewise. Padding and convolution mode is generally not discussed except in Theis et al. (2017), but we believe all other methods use one of the two ways they present, namely zero-padded or mirror-padded convolutions in `same` mode. The further details of each individual work is detailed below.

Ballé et al. (2016b) They build a relatively shallow autoencoder (5 layers) (Shown in the left half of Figure 2.3). They propose their own activation function, custom tailored for image compression. These non-linearities are a form of adaptive local gain control for the images, called Generalized Divisive Normalization (GDN). At the kth layer for channel i at position (m, n), for input $w_i^{(k)}(m, n)$, the GDN transform is defined as

$$u_i^{(k+1)}(m,n) = \frac{w_i^{(k)}(m,n)}{\left(\beta_{k,i} + \sum_j \gamma_{k,i,j}\left(w_j^{(k)}(m,n)\right)^2\right)}. \tag{2.1}$$

Its approximate inverse, IGDN for input $\hat{u}_i^{(k)}(m, n)$ is defined as

$$\hat{w}_i^{(k)}(m,n) = \hat{u}_i^{(k)}(m,n) \cdot \left(\hat{\beta}_{k,i} + \sum_j \hat{\gamma}_{k,i,j}\left(\hat{u}_j^{(k)}(m,n)\right)^2\right)^{\frac{1}{2}}. \tag{2.2}$$

Here, the set $\beta_{k,i}, \gamma_{i,j,k}, \hat{\beta}_{k,i}, \hat{\gamma}_{i,j,k}$ are learned during training and fixed at test time.

Theis et al. (2017) They define a Compressive Autoencoder (CAE) as a regular autoencoder with the quantization step between the encoding and decoding step. (In this sense, the architectures of Ballé et al. (2016b) and Ballé et al. (2018) are also CAEs.) They mirror pad the input first and then they follow it up by a deep, fully convolutional, residual architecture He et al. (2016). They use valid convolutions and downsample by using a stride of 2. Between convolutions they use leaky ReLUs as nonlinearities, which are defined as

$$f_\alpha(x) = \max\{x, \alpha x\}, \quad \alpha \in [0, 1].$$

The decoder mirrors the encoder. When upsampling is required, they use what they term *subpixel* convolutions, where they perform a regular convolution operation with an increased

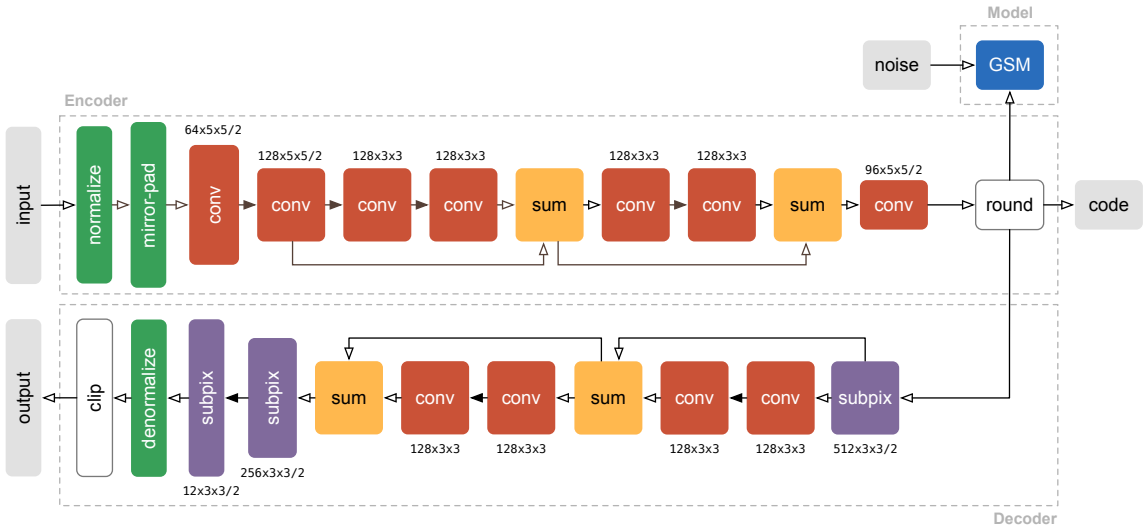


Fig. 2.1 Compressive Auto-Encoder architecture used by Theis et al. (2017). Note that for visual clarity only 2 residual blocks are displayed, in their experiments they used 3. They use a 6-component Gaussian Scale Mixture model (GSM) to model the quantization noise during the training of the architecture. The normalization layer performs batch normalization separately for each channel, denormalization is the analogous inverse operation. (Image taken from their Theis et al. (2017).)

number of filters, and then reshape the resulting tensor into one with larger spatial extent but fewer channels. Their architecture can be seen in Figure 2.1.

Rippel and Bourdev (2017) They use a reasonably shallow architecture as well, but also add in an additional residual connections from every layer to the last, summing at the end. They call this *pyramidal decomposition* and *interscale alignment*, with the rationale behind it being that the residual connections extract features at different scales, and so the latent representations can take advantage of this. Their encoder architecture is shown in Figure 2.2.

Ballé et al. (2018) They extend the architecture presented in Ballé et al. (2016b), see Figure 2.3. In particular, the encoder and decoder remain the same, and they add an additional stochastic layer on top of the architecture, resembling a probabilistic ladder network (PLN) (Sønderby et al. (2016)). The layers leading to the second level are more standard. They are still fully convlutional with downsampling after convolutions, however, instead of GDN they use ReLUs.

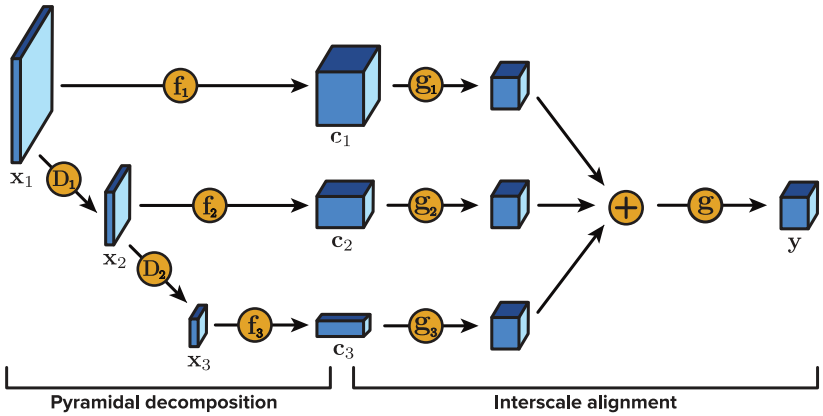


Fig. 2.2 Encoder architecture used by Rippel and Bourdev (2017). All circular blocks denote convolutions. (Image taken from Rippel and Bourdev (2017).)

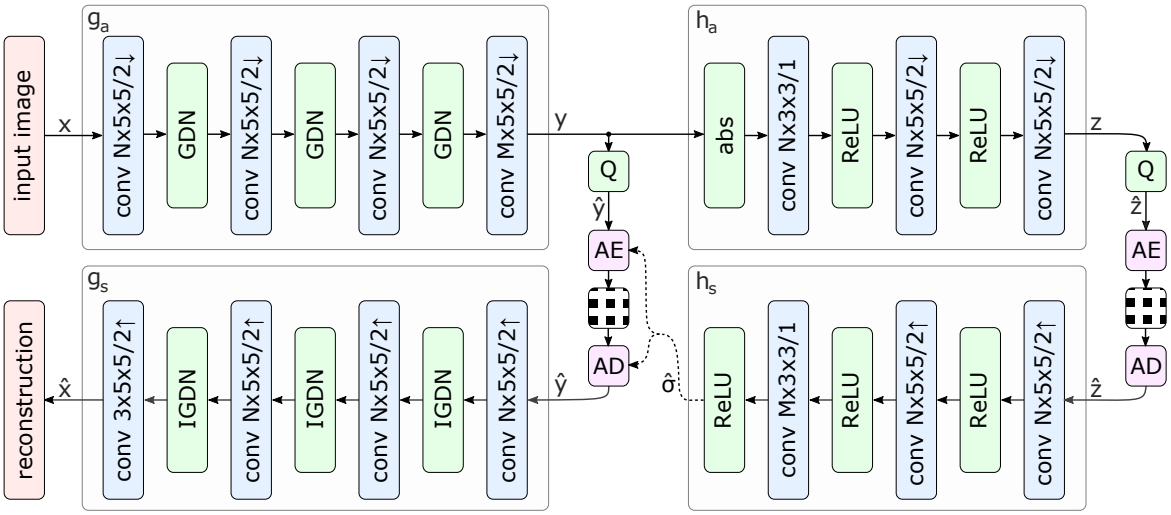


Fig. 2.3 Analysis and synthesis transforms g_a and g_s along with first level quantizer $Q(\mathbf{y})$ used in Ballé et al. (2016b). This architecutre was then extended by Ballé et al. (2018) with second level analysis and synthesis transforms h_a and h_s, along with second level quantizer $Q(\mathbf{z})$. This full architecture is also the basis of our model. A slightly strange design choice on their part is since they will wish to force the second stage activations to be positive (it will be predicting a scale parameter), instead of using an exponential or softplus ($\log(1 + \exp\{x\})$) activation at the end, they take the absolute value of the input to the first layer, and rely on the ReLUs never giving negative values. We are not sure if this was meant to be a computational saving, as taking absolute values is certainly cheaper then either of the aforementioned standard ways of forcing positive values, or it if it gave better results. (Image taken from Ballé et al. (2018))

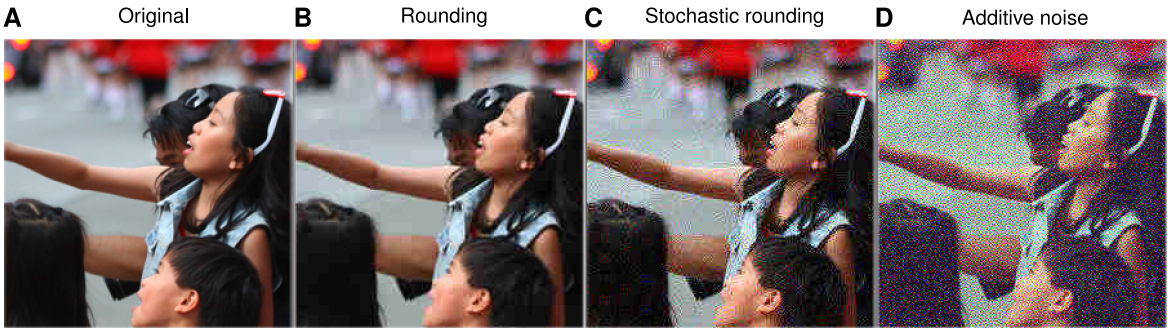


Fig. 2.4 Comparison of quantization error and its relaxations. **A)** Original image. **B)** Artifacts that result from using rounding as the quantizer. **C)** Stochastic rounding used by Toderici et al. (2017). **D)** Uniform additive noise used by Ballé et al. (2016b) and Ballé et al. (2018). (Image taken from Theis et al. (2017).)

2.2.3 Addressing Non-Differentiability

As all methods surveyed here are trained using gradient-based methods, a crucial question that needs to be answered is how they dealt with the issue of quantization. This is because end-to-end optimizing the transform coding pipeline involves back-propagating gradients through the quantization step, which yields 0 derivatives almost everywhere, stopping the learning signal. In fact, there are two issues that need to be addressed and that we examine below: first, the quantization operation itself, and second, the rate estimator $H[P(\mathbf{z})]$ (except for Rippel and Bourdev (2017) as they do not use it). As we have seen in Section 1.6, not only is there inherent error in whatever approximation is used to circumvent the non-differentiability of quantization, the use of quantization itself already fundamentally limits how effective these methods can be. A graphical representation of various continuous relaxations / approxiamtions of quantization error can be seen in Figure 2.4.

Quantization

All methods use some form of rounding $\mathbf{z}$ as

$$\hat{z}_i = \frac{1}{2^B} \left[2^B \times z_i \right], \tag{2.3}$$

for some B (usually $B = 0$). Below we see what continuous relaxations are used during training time.

Ballé et al. (2016b) and Ballé et al. (2018) They model quantization error as dither, i.e. they replace their quantized latents $\hat{z}_i$ by additive unifrom noise

$$\tilde{z}_i = z_i + \delta z_i, \quad \delta z_i \sim \mathcal{U}(0, 1).$$

Theis et al. (2017) They replace the derivative of the rounding operation in the backpropagation chain by the constant function 1:

$$\frac{d}{dy}[y] = 1.$$

This is a smooth approximation of rounding and they report that empirically it gave good results. However, as quantization itself creates an important bottleneck in the flow of information, it is key that only the derivative is replaced during the backward pass and not the operation itself during the forward pass.

Rippel and Bourdev (2017) use $B = 6$ in Eq 2.3, however, they do not reveal their relaxation of the quantization step during the learning phase. They cite Ballé et al. (2016b) and Toderici et al. (2017) though, so our best guess is that they most likely picked a method proposed in either one of those.

Rate Estimation

As all methods but Rippel and Bourdev (2017) under review aim to optimize the rate-distrotion trade-off directly, they also need to estimate the rate $H[\hat{\mathbf{z}}] = -\mathbb{E}_{P(\hat{\mathbf{z}})}\left[\log P(\hat{\mathbf{z}})\right]$. Hence to model the rate during training, they also require a distribution over the $\tilde{z}_i$s.

Ballé et al. (2016b) They assume that the latents are independent, and hence they can model the the joint as a fully factorized distribution. They use linear splines to do this, whose parameters $\psi^{(i)}$ they update separately every 10^6 iterations using SGD to maximize its log-likelihood on the latents, independently from the optimization of the rest of the model parameters. Then, they use this prior to replace the entropy term as

$$H[\tilde{\mathbf{z}}] = \mathbb{E}\left[-\sum_i \log_2 p(z_i + \delta z_i \mid \psi^{(i)})\right].$$

Theis et al. (2017) They note that

$$P(\mathbf{z}) = \int_{\left[-\frac{1}{2},\frac{1}{2}\right)^M} q(\mathbf{z}+\mathbf{u})\,\mathrm{d}\mathbf{u}$$

for some appropriate density q, where the integral is taken over the centered M dimensional hypercube. Then, they replace the the rate estimator with an upper bound using Jensen's inequality:

$$-\log_2 P(\mathbf{z}) = -\log_2 \int_{\left[-\frac{1}{2},\frac{1}{2}\right)^M} q(\mathbf{z}+\mathbf{u})\,\mathrm{d}\mathbf{u} \leq -\int_{\left[-\frac{1}{2},\frac{1}{2}\right)^M} \log_2 q(\mathbf{z}+\mathbf{u})\,\mathrm{d}\mathbf{u}.$$

This upper bound is now differentiable. They pick Gaussian Scale Mixtures for q, with $s = 6$ components, with the mixing proportions fixed across spatial dimensions, which gives the negative log likelihood

$$-\log_2 q(\mathbf{z}+\mathbf{u}) = \sum_{i,j,k} \log_2 \sum_s \pi_{k,s} \mathcal{N}\left(z_{k,i,j} + u_{k,i,j} \mid 0, \sigma_{k,s}^2\right),$$

where i, j iterate through the spatial dimensions and k indexes the filters. The integration of this in the architecture can be seen in Figure 2.1. This allows them to replace the rate estimator with

$$H[\tilde{\mathbf{z}}] = -\mathbb{E}\left[\log_2 q(\mathbf{z}+\mathbf{u})\right].$$

Ballé et al. (2018) They us a non-parametric, fully factorized prior for the second stage:

$$p(\tilde{\mathbf{z}}^{(2)} \mid \psi) = \prod_i \left(p\left(\tilde{z}_i^{(2)} \mid \psi_i\right) * \mathcal{U}\left(-\frac{1}{2},-\frac{1}{2}\right)\right).$$

Then, they model the first stage as dithered zero-mean Gaussians with variable scale depending on the second stage, thereby relaxing the initial independence assumption on the latent space to a more general *conditional indepenence* assumption Bishop (1998):

$$p(\tilde{\mathbf{z}}^{(1)} \mid \tilde{\mathbf{z}}^{(2)}) = \prod_i \left(\mathcal{N}\left(\tilde{z}_i^{(1)} \mid 0, \tilde{\sigma}_i^2\right) * \mathcal{U}\left(-\frac{1}{2},-\frac{1}{2}\right)\right).$$

They then replace the rate estimator similarly to Ballé et al. (2016b).

2.2.4 Coding

Another important part of the examined methods is the entropy coding. In particular, an interesting caveat of entropy codes is that they tend to perform slightly worse than the predicted rate, due to neglected constant factors in the algorithm Rissanen and Langdon (1981). Hence, it is always more informative to present results where the actual coding has been performed and not just the theoretical rate reported. All examined works have implemented their own coding algorithms, and we briefly review them here.

Ballé et al. (2016b) Use a context adaptive binary arithmetic coding (CABAC). They code dimensions in raster-scan order, which means they do not fully leverage the spatial dependencies between adjacent latent dimensions. As the authors note, this means that CABAC does not yield much improvement over non-adaptive artihmetic coding.

Theis et al. (2017) used their estimated probabilities $q(\mathbf{z})$ and used an off-the-shelf publicly available range coder to compress their latents.

Rippel and Bourdev (2017) treat each bit of their B-bit precision quantized representations individually, because they want to utilize the sparsity of more significant bits. They train a separate binary classifier to predict probabilities for each individual bit based on a set of features (they call it a *context*) to use in an adaptive arithmetic coder. They further add a regularizing term during training based on the codelength of a batch to match a length target. This is to encourage sparsity for high-resolution, but low entropy images and a longer codelength for low resolution but high entropy images.

Ballé et al. (2018) use a non-adaptive arithmetic coder as their entropy code. As they have two stochastic levels, with the first depending on the second, they have to code them sequentially. For the second level, they get their frequency estimates for $\hat{\mathbf{z}}^{(2)}$ from the non-parametric prior:

$$p(\hat{z}_i^{(2)}) = \int_{\hat{z}_i^{(2)}-\frac{1}{2}}^{\hat{z}_i^{(2)}+\frac{1}{2}} p(\tilde{z}_i \mid \psi_i)\,\mathrm{d}\tilde{z}_i.$$

Then, on the first level, their probaibilities are given by:

$$p(\hat{z}^{(1)} \mid \tilde{z}^{(2)}) = p(\hat{z}^{(1)} \mid \tilde{\sigma}_i^2) = \int_{\hat{z}_i^{(1)}-\frac{1}{2}}^{\hat{z}_i^{(1)}+\frac{1}{2}} \mathcal{N}\left(\tilde{z}_i \mid 0, \tilde{\sigma}_i^2\right)\,\mathrm{d}\tilde{z}_i.$$

2.2.5 Training

Ballé et al. (2016b), Theis et al. (2017) and Ballé et al. (2018) optimize the rate-distrotion trade-off directly,

$$L(\mathscr{D}) = H[\hat{\mathbf{z}}] + \beta \mathbb{E}\left[d(\mathbf{x}, \hat{\mathbf{x}})\right],$$

where the expectation is taken over training batches. This turns out to be equivalent to maximizing the ELBO for a certain VAE, whereas Rippel and Bourdev (2017) use a more complex loss function, see below for details. All methods train their model using Adam Kingma and Ba (2014), but most do not state the number of training epoch.

Ballé et al. (2016b) Since they use MSE as the distance metric, they note that their architecture could be considered as a (somewhat unconventional), VAE with Gaussian likelihood

$$p(\mathbf{x} \mid \tilde{\mathbf{z}}, \beta) = \mathscr{N}\left(\mathbf{x} \mid \hat{\mathbf{x}}, (2\beta)^{-1}\mathbf{1}\right),$$

mean-field prior

$$p(\tilde{\mathbf{z}} \mid \psi^1, \ldots, \psi^N) = \prod_i p(\tilde{z}_i \mid \psi^{(i)})$$

and mean-field posterior

$$q(\tilde{\mathbf{z}} \mid \mathbf{x}) = \prod_i \mathscr{U}\left(\tilde{z}_i \mid z_i, 1\right),$$

where $\mathscr{U}\left(\tilde{z}_i \mid z_i, 1\right)$, is the uniform distribution centered on z_i of width 1. They used learning rate decay during training.

Theis et al. (2017) They performed the training incrementally, in the sense that they masked most latents at the start, such that their contribution to the loss was 0. Then, as the training performance saturated, they unmasked them incrementally. They also used learning rate decay during training.

Rippel and Bourdev (2017) They use a complex loss function, with an MS-SSIM distrotion cost, a code length regularization term (not equivalent to the rate term) as well as an adversarial loss term. In their adversarial setup they feed ground truth, reconstruction pairs to the discriminator, where they shuffle the images in the pair with probability $\frac{1}{2}$ and their discriminator was trained to predict which image was the reconstruction. Their setup can be seen in Figure 2.5. To stabilize the adversarial training procedure, they also introduce an adaptive learning signal scheduler, preventing any of the signals from dominating the total.

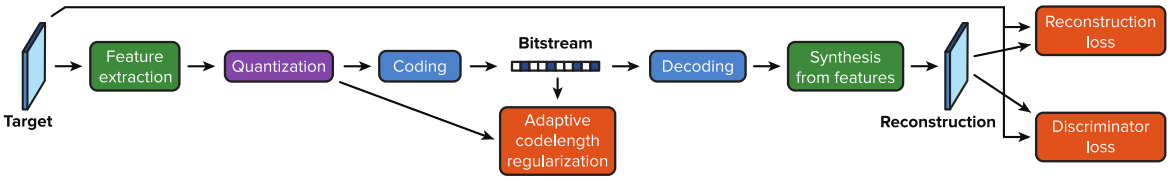


Fig. 2.5 Compression pipeline used by Rippel and Bourdev (2017). The red boxes show the terms used in their loss function. (Image taken from Rippel and Bourdev (2017).)

Ballé et al. (2018) In the same vein as they laid out their VAE-based training objective in Ballé et al. (2016b), the data log-likelihood term stays, but now the regularizing term is the KL divergence between the joint posterior $q(\tilde{\mathbf{z}}^{(1)}, \tilde{\mathbf{z}}^{(2)} \mid \mathbf{x})$ and the joint prior $q(\tilde{\mathbf{z}}^{(1)}, \tilde{\mathbf{z}}^{(2)})$. Here, as due to the dithering assumption, the joint posterior works out to be

$$q\left(\tilde{\mathbf{z}}^{(1)}, \tilde{\mathbf{z}}^{(2)} \mid \mathbf{x}\right) = \prod_i \mathcal{U}\left(\tilde{z}_i^{(1)} \mid \hat{z}^{(1)}, 1\right) \cdot \prod_i \mathcal{U}\left(\tilde{z}_i^{(2)} \mid \hat{z}^{(2)}, 1\right).$$

Then, taking the KL between these, the full traning objective works out to be

$$L = \mathbb{E}\left[-\sum_i \log_2 p(\tilde{z}_i^{(2)} \mid \psi^{(i)}) - \sum_i \log_2 p(\tilde{z}_i^{(1)} \mid \tilde{\sigma}_i^2) + \beta d(\mathbf{x}, \hat{\mathbf{x}})\right]. \tag{2.4}$$

Eq 2.4 is important from our prespective, as it will be directly translated to our learning objective.

They train 32 models, half using the architecture from Ballé et al. (2016b) and half using the current one, half optimized for MSE and half of MS-SSIM, with 8 different βs. They report that neither batch normalization nor learning rate decay gave better results, which they attribute to GDN.

2.2.6 Evaluation

As mentioned at the end of Section 2.2.1, all methods were tested on the Kodak dataset (Eastman Kodak Company (1999)). All authors report the (interpolated) rate-distrotion curves achieved by their models. All methods report the curves using PSNR (Huynh-Thu and Ghanbari (2008)) as the distrotion metric as well as MS-SSIM (Wang et al. (2003)), except for Rippel and Bourdev (2017), who only report MS-SSIM. Based on these reported curves, the current state-of-the-art in neural compression is set by Ballé et al. (2018).

An important issue raised in Ballé et al. (2016b) and Ballé et al. (2018) is how aggregate results over should be reported, or whether reporting such figures is meaninful in the first place. This is because since models were trained on different datasets, with different

model design philosophies, there might be significant fluctuation in the comparative model performance between individual images. Furthermore, averaging results achieved by classical methods that were not directly optimized for the rate-distrotion trade-off (virtually all of them) might lead to inconsistent results even on the same image, depending on what settings were used to achieve a given bitrate. Hence, they argue that the efficiency of the methods should be examined on individual images instead. This is also the philosophy we follow, and hence we will be reporting model performances on individual images.

Chapter 3

Method

In this chapter, we describe the models we use to demonstarte the efficiency of the general lossy compression framework developed in Section 1.6. We begin by describing the input pipeline, followed by our model architectures and their training procedure. We then present 3 tractable, coded sampling algorithms that can be used within our framework.

Based on our framework, at a high level our image compression algorithm is as follows:

1. Pick an appropriate VAE-based architecture (Probabilistic Ladder Networks (PLNs) in our case) for image reconstruction. (Section 3.2)
2. Train the model on a reasonably selected dataset for this task. (Sections 3.1 and 3.2)
3. Once the VAE is trained, given a new image $\mathbf{x}$, we can use the latent posterior $q(\mathbf{z} \mid \mathbf{x})$ and prior $p(\mathbf{z})$ for our coded sampling algorithm. (Section 3.4)
4. We may consider to use entropy codes to further increase the efficiency of our coding, if appropriate.

A rather pleasing aspect of this is the modularity that is allowed by the removal of quantization from the training pipeline: our method is reusable with virtually any regular VAE architecture, which opens up the possibility of creating efficient compression algorithms for any domain where a VAE can be used to reconstruct the objects of interest.

3.1 Dataset and Preprocessing

We trained our models on the CLIC 2018 dataset (CLIC (2018)), as it seemed sufficiently extensive for our project. It was also curated for an image compression challenge, and thus "bad" images have been filtered out, which reduced the amount of preprocessing required on our side.

The dataset contains high-resolution PNG encoded photographs, 585 in the training set and 41 photos in the validation set. The test set is not publicly available, as it was reserved for the competition. To make training tractable, similarly to previous works, we randomly extracted 256×256 pixel patches from each image. The number of patches P was based on their size of the image, according to the formula

$$P(W, H) = C \times \left\lfloor \frac{W}{256} \right\rfloor \times \left\lfloor \frac{H}{256} \right\rfloor,$$

where W, H are the width and height of the current image, respectively and C is an integer constant we set. We used $C = 15$, which yielded us a training set of 93085 patches.

We note that all image data we used for learning was in RGB format. It is possible to achieve better compression rates using the YCbCr format (Ballé et al. (2016b), Rippel and Bourdev (2017)), however, for simplicity's sake as well as due to time constraints we leave investigating this for later work.

3.2 Architectures

In this section we describe the various architectures that we experimented with. The basis of all our architectures were inspired by the ones used in Ballé et al. (2016b) and Ballé et al. (2018). In particular, we use the General Divisive Normalization (GDN) layer for encoding and its approximate inverse, the IGDN layer for decoding (Ballé et al. (2015), Ballé et al. (2016b)).

3.2.1 VAEs

As a baseline, we started by replicating the exact architecture presented in Ballé et al. (2016b), but using a Gaussian prior and posterior instead. We chose mirror padding with our convolutions, as it is standard for in this setting (Theis et al. (2017)). Luckily, the most error-prone part, the implementation of the GDN and IGDN layers was already available in `Tensorflow`[1].

While VAEs are by now fairly standard and we assume that the reader is at least somewhat familiar with them, our later models build on them and are non-standard, hence it is useful to briefly go over them and introduce notation that we extend in the following sections.

Note: In the following, we will assume that all multivariate distributions have diagonal covariance structure, and hence we parameterize their scales using vectors, to be understood as the diagonal of the covariance matrix. Furthermore, In the case of Gaussians, in this section only we parameterize them using their standard deviations instead of their variances. All arithmetc operations on vectors are also to be understood elementwise.

In a regular VAE, we have a first level encoder, that given some input $\mathbf{x}$ predicts the posterior

$$q^{(1)}(\mathbf{z}^{(1)} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{z}^{(1)} \mid \boldsymbol{\mu}^{e,(1)}(\mathbf{x}), \boldsymbol{\sigma}^{e,(1)}(\mathbf{x})\right),$$

where $\boldsymbol{\mu}^{e,(1)}(\mathbf{x}) = (m \circ f)(\mathbf{x})$ predicts the mean and $\boldsymbol{\sigma}^{e,(1)}(\mathbf{x}) = (\exp \circ s \circ f)(\mathbf{x})$ predicts the standard deviation of the posterior. Here f is a highly nonlinear mapping of the input; in reality corresponding to several layers of neural network layers. Notice that f is shared for the two statistics. Then, m and s are custom linear transformations, and finally we take the exponential of $s \circ f$ to force the standard deviation to be positive. We sample $\tilde{\mathbf{z}}^{(1)} \sim q^{(1)}$. We show a typical posterior distribuion given an image in Figure 3.1. The first level prior is usually assumed to be a diagonal Gaussian

$$p^{(1)}(\mathbf{z}^{(1)}) = \mathcal{N}\left(\mathbf{z}^{(1)} \mid \mathbf{0}, I\right).$$

Finally, the first level decoder predicts the statistics of the data likelihood,

$$p(\mathbf{x} \mid \tilde{\mathbf{z}}^{(1)}).$$

A note on the latent distributions We have chosen to use Gaussian latent distributions due to their simplicity, as well as their extenesibility to PLNs (see Section 3.2.3). On the other hand, we note that Gaussians are inappropriate, as it has been shown that the filter responses of natural images usually follow a heavy-tailed distribution, usually assumed to be a Laplacian (Jain (1989)), as used directly in (Zhou et al. (2018)), but can also be approximated reasonably well by Gaussian Scale Mixtures (Portilla et al. (2003)), as adopted by Theis et al. (2017). While it would be interesting to investigate incorporating these into our model, as they do not extend trivially to our more complex model settings (in particular PLNs, as we

[1]https://www.tensorflow.org/api_docs/python/tf/contrib/layers/gdn

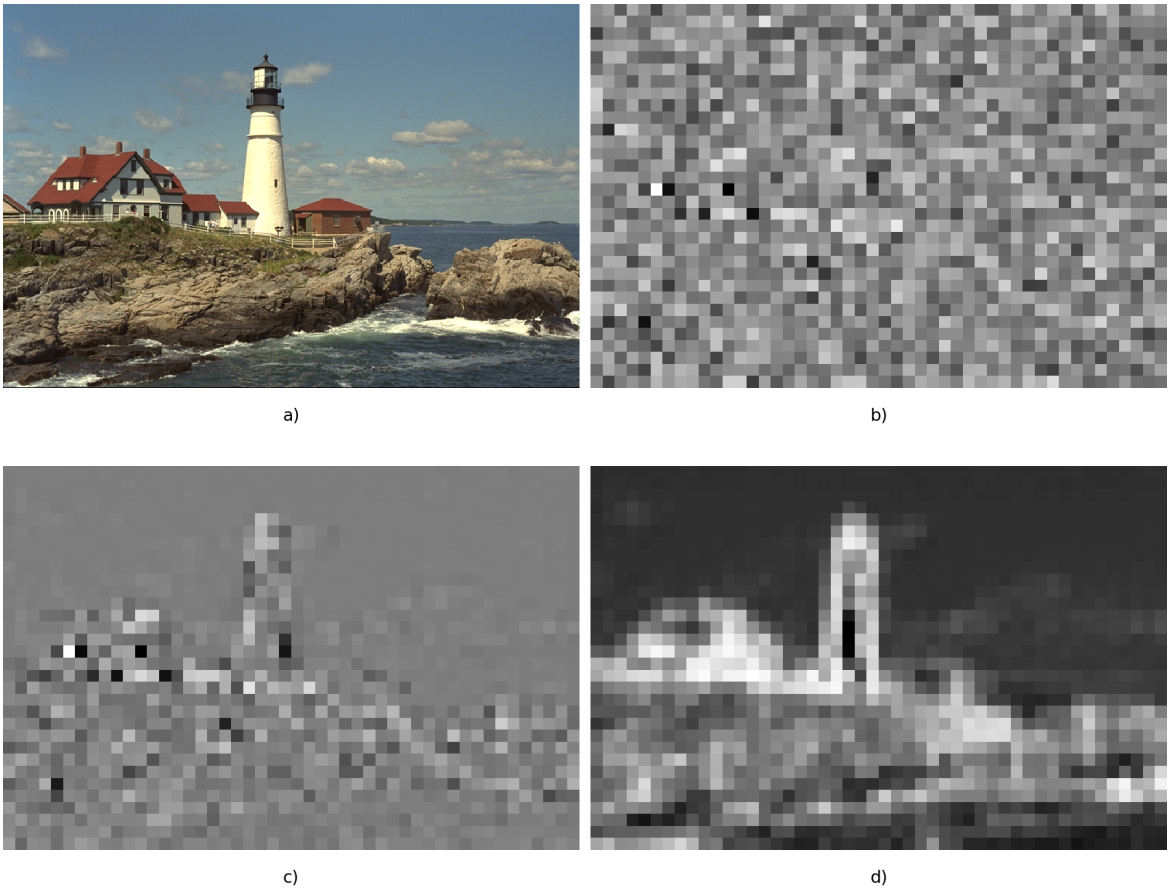


Fig. 3.1 **a)** `kodim21.png` from the Kodak Dataset. **b)** A random sample from the VAE posterior. **c)** Posterior means in a randomly selected channel. **d)** Posterior standard deviations in the same randomly selected channel. We can see that there is a lot of structure in the latent space, on which the full indepenence assumption will have a detrimental effect. (We have examined several random channels and observed the similarly high structure. We present the above cross-section without preference.)

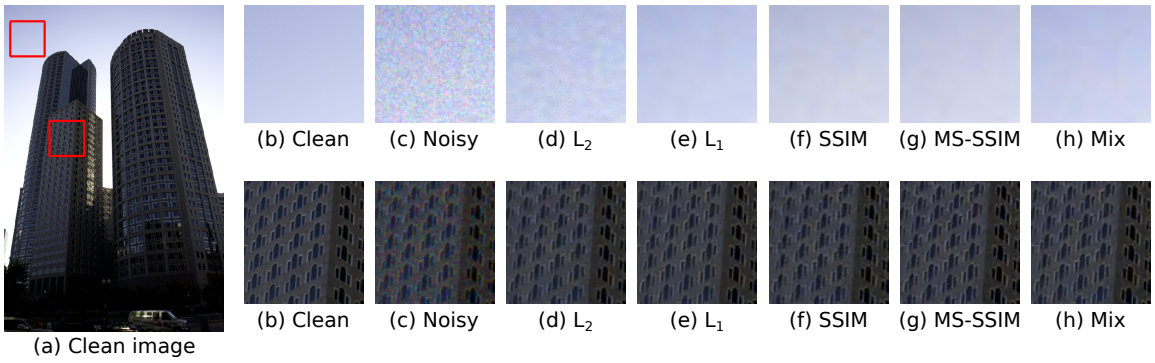


Fig. 3.2 Image reconstruction quality comparison on the task of joint image denoising and demosaicing, for the same architecture optimized using different distortion metrics. **a)-b)** Show the original image. **c)** Show the input to the networks. **d) - h)** Show reconstructions using various distrotion metrics. Mix is (approximately) defined as $(1-\lambda)L_1 + \lambda$MS-SSIM for $\lambda = 0.84$. The differences are best seen on the electronic version, zoomed in. We can clearly see the patchy artifacts introduced by Mean Squared Error (**d**)), and how much better Mean Absolute Error (**e**)) performs compared to it. (Image taken from Zhao et al. (2015). We changed the fonts of their captions to a sans-serif font for better readabitity.)

formulated them here requre the latent posterior distribution's family to be self-conjugate), we leave this for future work.

3.2.2 Data Likelihood and Training Objective

Based on the framework presented in Section 1.6, the training objective used to train the VAE is the (weighted) ELBO:

$$\mathbb{E}_{q^{(1)}(\mathbf{z}^{(1)})}\left[\log p(\mathbf{x} \mid \mathbf{z}^{(1)})\right] - \beta \mathrm{KL}\left[q^{(1)}(\mathbf{z}^{(1)} \mid \mathbf{x}) \mid\mid p^{(1)}(\mathbf{z}^{(1)})\right]. \tag{3.1}$$

As the latent posterior and prior are both Gaussians, the KL can be computed analytically, and there is no need for a Monte Carlo estimation. A popular and simple choice for the likelihood to be chosen a Gaussian, in which case the expectectation of the log-likelihood corresponds to the mean squared error between the original image and its reconstruction. This also corresponds to optimizing for the PSNR as the perceptual metric. However, PSNR correlates badly with the HVS's perception of image quality (Girod (1993) Eskicioglu et al. (1994)). This is mainly because an MSE training objective is tolerant to small deviations regardless of the structure in the image, and hence this leads to blurry colour patch artifacts in low-textured regions, which the HVS quickly picks up as unpleasant. A thorough survey of different training losses for image reconstruction was performed by Zhao et al. (2015), see Figure 3.2. Optimizing the MS-SSIM distortion for the same architecture, they show

that the artifacts are greatly reduced. They also show that, somewhat surprisingly, Mean Absolute Error (MAE) also significantly reduces and in some cases completely removes the unpleasant artifacts introduced by MSE. This is because MAE no longer underestimates small deviations, at the cost of somewhat blurrier edges, which MSE penalized more. The MAE corresponds to a diagonal Laplacian log-likelihood with unit scale, which is what we decided to use in our experiments. This results in efficient training (an MS-SSIM training loss, though differentiable, is very expensive to compute) as well as it will enable us to use a further enchancement, see Section 3.3.1.

Concretely, our likelihood is going to be

$$p(\mathbf{x} \mid \mathbf{z}^{(1)}) = \mathscr{L}\left(\hat{\mathbf{x}} \mid \boldsymbol{\mu}^{d,(1)}(\tilde{\mathbf{z}}^{(1)}), I\right), \tag{3.2}$$

where $\boldsymbol{\mu}^{d,(1)}$ is the reverse operation of $\boldsymbol{\mu}^{e,(1)}$.

3.2.3 Probabilistic Ladder Network

We now introduce two extensions of VAEs to accomodate more complex latent dependency structures: hierarchical VAEs (H-VAEs) and Probabilistic Ladder Networks (PLNs) (Sønderby et al. (2016)). For simplicity's sake, we only consider two-level H-VAEs and PLNs, the these can be easily extended to more stochastic levels. In both cases, we essentially stack VAEs on top of each other and train them together, though the way this is done is crucial for our use case.

To extend the VAE architecture from Section 3.2.1 to get a 2-level H-VAE, once $\tilde{\mathbf{z}}^{(1)}$ is sampled, we use it to predict the statistics of the second level posterior

$$q^{(2)}(\mathbf{z}^{(2)} \mid \tilde{\mathbf{z}}^{(1)}) = \mathcal{N}\left(\mathbf{z}^{(2)} \mid \boldsymbol{\mu}^{e,(2)}(\tilde{\mathbf{z}}^{(1)}), \boldsymbol{\sigma}^{e,(2)}(\tilde{\mathbf{z}}^{(1)})\right),$$

where $\boldsymbol{\mu}^{e,(2)}(\tilde{\mathbf{z}}^{(1)})$ and $\boldsymbol{\sigma}^{e,(2)}(\tilde{\mathbf{z}}^{(1)})$ are analogous to their first level counterparts. Next the second level is sampled $\tilde{\mathbf{z}}^{(2)} \sim q^{(2)}$. The second level prior $p^{(2)}(\mathbf{z}^{(2)})$ is now the diagonal unit-variance Gaussian, and the first level priors' statistics are predicted using $\tilde{\mathbf{z}}^{(2)}$:

$$p^{(1)}(\mathbf{z}^{(1)} \mid \tilde{\mathbf{z}}^{(2)}) = \mathcal{N}\left(\mathbf{z}^{(1)} \mid \boldsymbol{\mu}^{d,(2)}(\tilde{\mathbf{z}}^{(2)}), \boldsymbol{\sigma}^{e,(2)}(\tilde{\mathbf{z}}^{(2)})\right).$$

The data likelihood's mean is predicted using $\tilde{\mathbf{z}}^{(1)}$ as before (Sønderby et al. (2016)).

The issue with H-VAEs is that the flow of information is limited by the bottleneck of the final stochastic layer. PLNs resolve this issue by allowing the flow of information between lower levels as well. To arrive at them, we make the following modification to our H-VAE: first, once $q^{(1)}$ is known, instead of sampling it immediately, we instead use its mean to

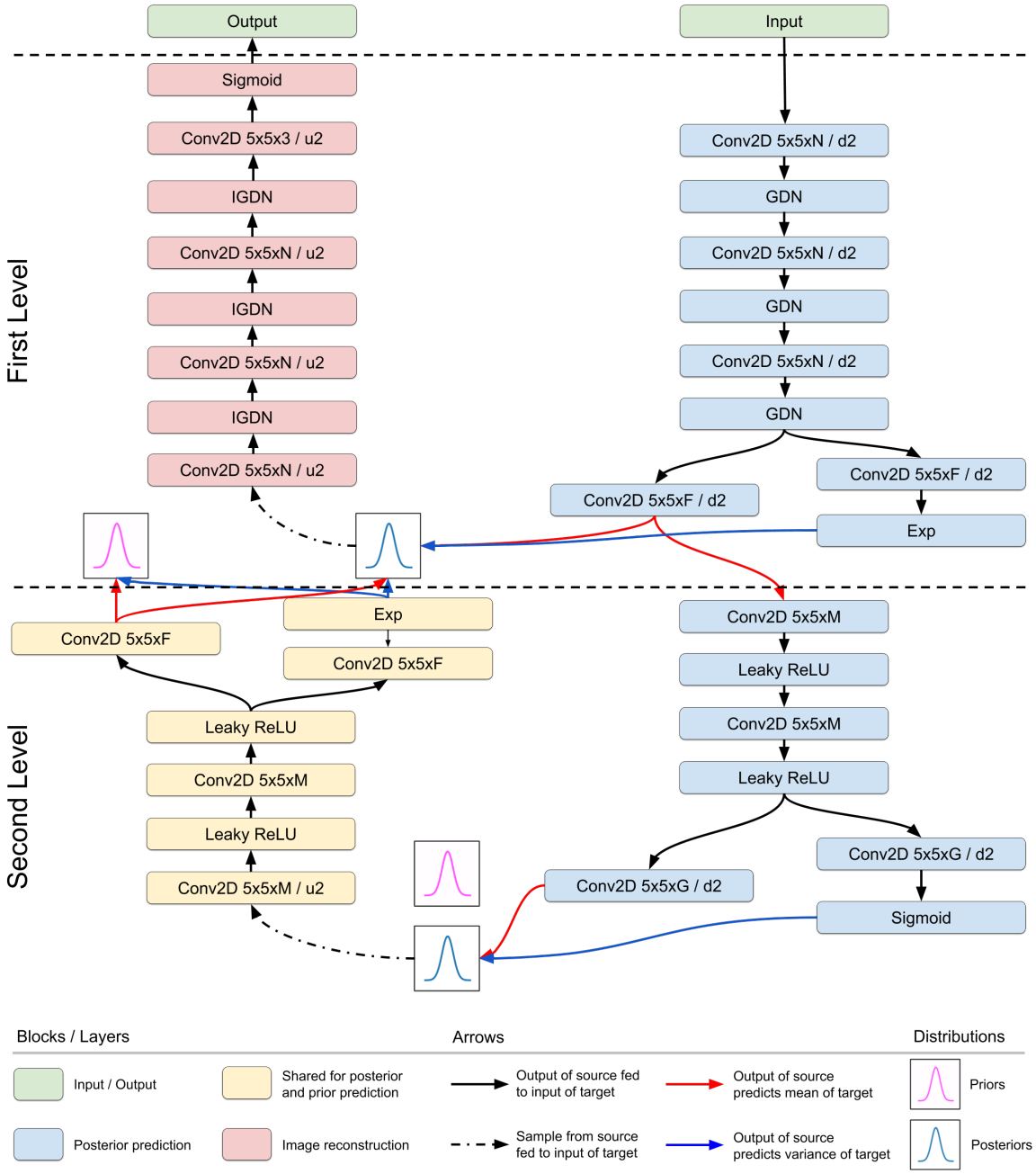


Fig. 3.3 PLN network architecture. The blocks signal data transformations, the arrows signal the flow of information. **Block descriptions:** *Conv2D:* 2D convolutions along the spatial dimensions, where the $W \times H \times C/S$ implies a $W \times H$ convolution kernel, with C target channels and S gives the downsampling rate (given a preceding letter "d") or the upsampling rate (given a preceding letter "u"). If the slash is missing, it means that there is no up/downsampling. All convolutions operate in `same` mode with mirror padding. *GDN / IGDN:* these are the non-linearities described in Ballé et al. (2016b). *Leaky ReLU:* elementwise non-linearity defined as $\max\{x, \alpha x\}$, where we set $\alpha = 0.2$. *Sigmoid:* Elementwise non-linearity defined as $\frac{1}{1+\exp\{-x\}}$. We ran all experiments presented here with $N = 196, M = 128, F = 128, G = 24$.

predict the statistics of the second level posterior:

$$q^{(2)}(\mathbf{z}^{(2)} \mid \tilde{\mathbf{z}}^{(1)}) = \mathcal{N}\left(\mathbf{z}^{(2)} \mid \boldsymbol{\mu}^{e,(2)}(\boldsymbol{\mu}_{\mathbf{x}}), \boldsymbol{\sigma}^{e,(2)}(\boldsymbol{\mu}_{\mathbf{x}})\right),$$

where $\boldsymbol{\mu}_{\mathbf{x}} = \boldsymbol{\mu}^{e,(1)}(\mathbf{x})$. Now, $\tilde{\mathbf{z}}^{(2)} \sim q^{(2)}$ is sampled. The first level prior $p^{(1)}$ is calculated as before. Finally, we allow the flow information on the first level by setting the posterior $q^{(1)}$ as the combination of the statistics predicted on the first level from the data and the statistics of $p^{(1)}$, inspired by the self-conjugacy of the Normal distribution in Bayesian inference[2]:

$$q^{(1)}(\tilde{\mathbf{z}}^{(1)} \mid \tilde{\mathbf{z}}^{2}, \mathbf{x}) = \mathcal{N}\left(\tilde{\mathbf{z}}^{(1)} \,\middle|\, \frac{\boldsymbol{\sigma}_{\mathbf{z}^{(1)}}^{-2}\boldsymbol{\mu}_{\mathbf{x}} + \boldsymbol{\sigma}_{\mathbf{x}}^{-2}\boldsymbol{\mu}_{\mathbf{z}^{(1)}}}{\boldsymbol{\sigma}_{\mathbf{x}}^{-2} + \boldsymbol{\sigma}_{\mathbf{z}^{(1)}}^{-2}}, \frac{1}{\sqrt{\boldsymbol{\sigma}_{\mathbf{x}}^{-2} + \boldsymbol{\sigma}_{\mathbf{z}^{(1)}}^{-2}}}\right),$$

where $\boldsymbol{\mu}_{\mathbf{x}} = \boldsymbol{\mu}^{e,(1)}(\mathbf{x}), \boldsymbol{\mu}_{\mathbf{z}^{(1)}} = \boldsymbol{\mu}^{d,(2)}(\tilde{\mathbf{z}}^{(2)})$ and $\boldsymbol{\sigma}_{\mathbf{x}} = \boldsymbol{\sigma}^{e,(1)}(\mathbf{x}), \boldsymbol{\sigma}_{\mathbf{z}^{(1)}} = \boldsymbol{\sigma}^{d,(2)}(\tilde{\mathbf{z}}^{(2)})$. We sample $\tilde{\mathbf{z}}^{(1)} \sim q^{(1)}(\tilde{\mathbf{z}}^{(1)} \mid \tilde{\mathbf{z}}^{(2)}, \mathbf{x})$, and predict the mean of the likelihood using it.

The reason why H-VAEs and PLNs are more powerful models than regular VAEs, is because regular VAEs make an independence assumption between the latents to make the model tractable to compute, while H-VAEs and PLNs relax this to a *conditional indpendence* assumption. In this sense, the architecture of Ballé et al. (2018) also defines a PLN. We present the PLN architecture we used in our experiments in Figure 3.3. We demonstarte the power of the conditional independence in PLNs compared to the full independence assumption in Figure 3.4.

Finally we need to update the regularizing term of the ELBO to incorporate the joint posterior and priors over the latents. This works out to be

$$\begin{aligned}\mathrm{KL}\left[\, q(\mathbf{z}^{(1)}, \mathbf{z}^{(2)} \mid \mathbf{x}) \;||\; p(\mathbf{z}^{(1)}, \mathbf{z}^{(2)}) \,\right] = \mathrm{KL}\left[\, q(\mathbf{z}^{(2)} \mid \mathbf{x}) \;||\; p(\mathbf{z}^{(2)}) \,\right] + \\ \mathrm{KL}\left[\, q(\mathbf{z}^{(1)} \mid \mathbf{z}^{(2)}, \mathbf{x}) \;||\; p(\mathbf{z}^{(1)} \mid \mathbf{z}^{(2)}) \,\right],\end{aligned}$$

which we can also compute analytically.

3.3 Training

Sønderby et al. (2016) give two key advices to train PLNs:

[2]We note that the formula we used in our definition is the actual combination rule for a Gaussian likelihood and Gaussian prior. The formula given in Sønderby et al. (2016) is slightly different. We are not sure if it is a typo or it is what they actually used. We found our combination rule worked quite well in practice.

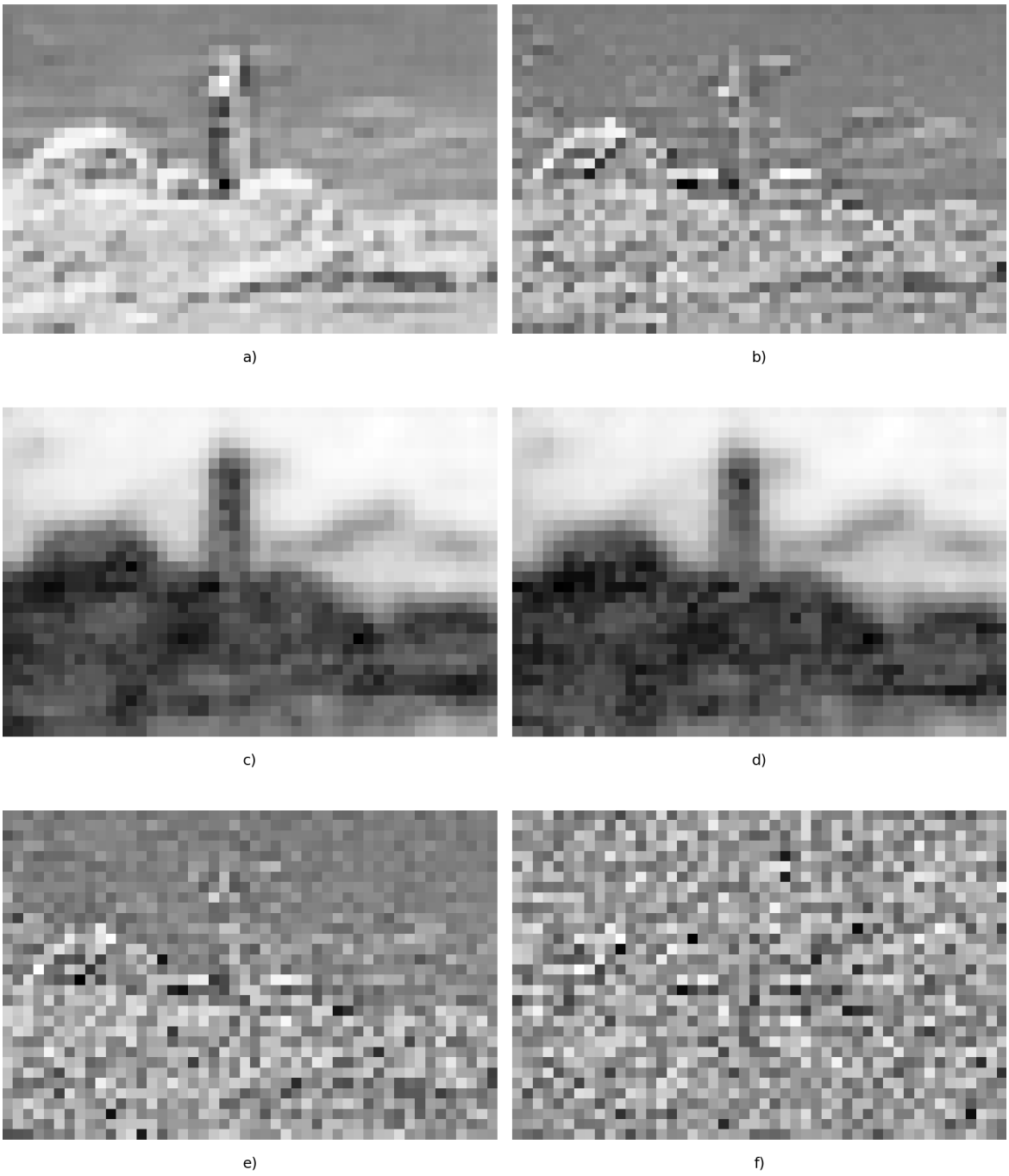


Fig. 3.4 We continue the analysis of the latent spaces induced by `kodim21` from the Kodak Dataset. Akin to Figure 3.1, we have selected a random channel for both the first and second levels each and present the spatial cross-sections along these channels. **a)** Level 1 prior means. **b)** Level 1 posterior means. **c)** Level 1 prior standard deviations. **d)** Level 1 posterior standard deviations. **e)** Random sample from the Level 1 posterior. **f)** The sample from **e)** standardized according to the level 1 prior. Most structure from the sample is removed, hence we see that the second level has successfully learned a lot of the dependencies between the latents. We have checked cross-sections along several randomly selected channels and observed the same phenomenon. We present the above with no preference.

- Use batch normalization (BN) (Ioffe and Szegedy (2015)).
- Use a warmup on the coefficient of the KL term in the loss. Concretely, given a target coefficient β_0, the actual coefficient they recommend should be

$$\beta(t) = \min\left\{\frac{t}{W}, 1\right\} \times \beta_0,$$

  where t is the current iteration and W is the *warmup period*.

We ended up not utilizing point 1, due to an argument of Ballé et al. (2018), namely that GDN already performs a similar kind of normalization as BN, and it did not improve their results.

3.3.1 Learning the Variance of the Likelihood

As we have noted, the reconstructions are blurry. A solution offered by Dai and Wipf (2019) is to introduce a new parameter γ to the model, that will be the scale of the data likelihood. In our case, since we are using a Laplace likelihood, we will have

$$p(\hat{\mathbf{x}} \mid \mathbf{z}^{(1)}) = \mathscr{L}\left(\hat{\mathbf{x}} \mid \boldsymbol{\mu}^{d,(1)}(\tilde{\mathbf{z}}^{(1)}), \gamma\right).$$

In the case of a Gaussian, γ would be the variance of the distribution. Then, it is suggested that instead of predicting gamma (i.e. using a heteroscedastic model), or setting it as a hyperparameter, *we learn it*. In Dai and Wipf (2019) this is used in conjunction with another novel technique to achieve generative results with VAEs that are competitive with state-of-the-art GANs (Goodfellow et al. (2014)). In this work, however, as the second technique is irrelevant to us, we focus on learning γ only.

Let us examine this concept a bit more: let D be the (original) log-likelihood term with unit scale, and R the (original) regularizing term, already multiplied by our target coefficient β. Then, our new loss is going to be

$$L = \frac{1}{\gamma} D + R.$$

Multiplying this through by γ does not change the minimizer of the expression, but we get the new loss

$$L' = D + \gamma R.$$

Dai and Wipf (2019) show that if γ is learned, then under some mild assumptions $\gamma \to 0$ as $t \to \infty$ (where t is the number of iterations in the training procedure). This means that if we

set some target γ_∞, and use

$$\gamma' = \max\{\gamma, \gamma_\infty\},$$

as the scale of the data likelihood, we actually get a dual effect to the warmup recommended by Sønderby et al. (2016), but with automatic scaling.

In parctice, γ does not converge to 0, but to a number very close to zero, which in experiments was around ≈ 0.02, and hence instead of using γ', we set different βs to achieve different rate-distrotion results. From now on, we will refer to PLNs where we also learned γ as γ-PLNs. In Figure 3.5 we show the posterior of one of our γ-PLNs.

3.4 Sampling and Coding

Once the appropriate network has been trained, this means that for any image $\mathbf{x}$ we are able to produce a latent posterior $q(\mathbf{z} \mid \mathbf{x})$ and prior $p(\mathbf{z})$. The next step in our framework is to use a bits-back efficient coded sampling algorithm to code a sample from the posterior. The first practical coded sampling algorithm to was described in Havasi et al. (2018), but there are several key differences between our setting and theirs:

- **Variable sized latent space:** The original method was developed for compressing the weight distribution of a BNN, whose dimension is fixed. In our case, due to our fully convolutional architecture, our rendition of the algorithm must be able to adapt gracefully to a range of latent space dimensions.

- **Different posterior effects:** As our latent spaces will carry much more information about the topological nature of the coded image, the distribution of informative versus non-informative posteriors and their properties will be different from the original setting, and we will need to adapt to these effects. These effects can be clearly seen in Figures 3.1, 3.4 and 3.5.

- **Practicality / Resource efficiency:** Since the original method has been proposed to compress neural networks after they have been trained, the algorithm was proposed with people who have sufficient resources to train them in mind. In particular, the original method incorporates several rounds of incremental retraining during the compression process to maintain low distortion, which might require several GPU hours to complete. As our aim in this work to present a practical, universal compression algorithm, we must also design our method with a much broader audience in mind. Though we still assume the presence of a GPU, our requirements and coding times are much lighter than that of the original work.

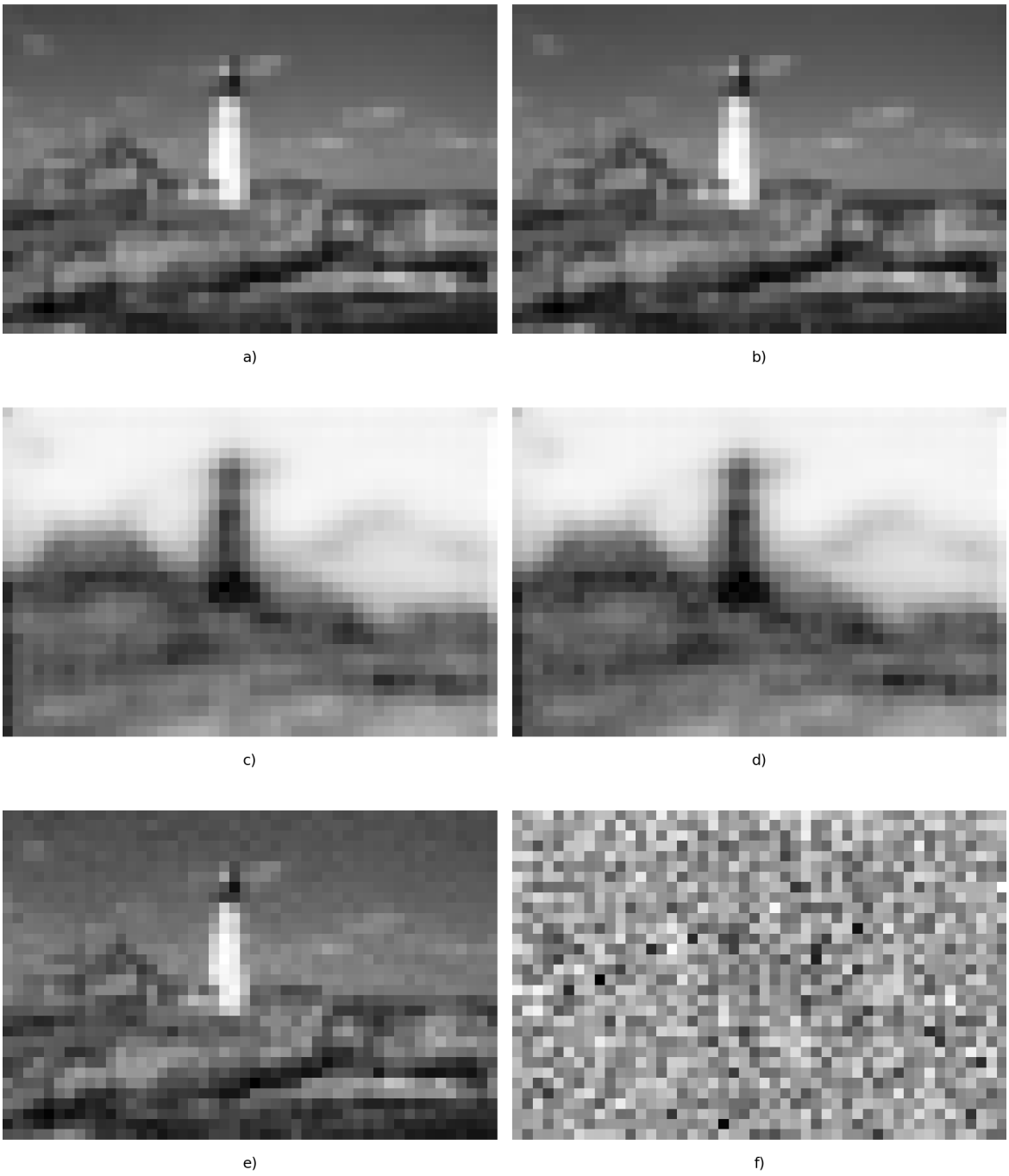


Fig. 3.5 We continue the analysis of the latent spaces induced by `kodim21` from the Kodak Dataset. Akin to Figures 3.1 and 3.4, we have selected a random channel for both the first and second levels each and present the spatial cross-sections along these channels. **a)** Level 1 prior means. **b)** Level 1 posterior means. **c)** Level 1 prior standard deviations. **d)** Level 1 posterior standard deviations. **e)** Random sample from the Level 1 posterior. **f)** The sample from **e)** standardized according to the level 1 prior. We observe the same phenomenon, with no significant difference, as in Figure 3.4. We note that while the posterior sample may seem like it has more significant structure than the one in the previous Figure. This is only coincidence; some of the regular PLN's channels contain similar structure, and some of the γ-PLN's channels contain more noisy elements.

In the rest of this section we present the ways we addressed the above points.

3.4.1 Parallelized Rejection Sampling and Arithmetic Coding

Sampling As a baseline, we designed a parallelized version of Algorithm 4, the original bits-back efficient sampling algorithm presented by Harsha et al. (2007). As pointed out by Havasi et al. (2018), this algorithm quickly becomes intractable once we leave the univariate setting. Fortunately, as we are working with (conditional) independence assumptions between the latents, sampling each dimension individually and then concatenating them will also give an exact multivariate sample.

We modify Algorithm 4 in two ways to make it more efficient. While their algorithm is Las Vegas, i.e. it always gives the right answer, but in random time, this can get very inefficient if we have a few dimensions with very high KL divergence. We circumvent this issue and fix the runtime of the algorithm by allocating a bit budget B to each dimension, and only allowing 2^B samples to be examined. If the sample is accepted within these draws, then we have a code for them. The dimensions where all 2^B samples were rejected, we sample from the target q, and quantize the samples to 16 bits. This gave about a 1% increase in size at a much imporved runtime. In particular, for dimensions with KL larger than 16 bits this is actually more efficient than coding them with Algorithm 4. We then use the quantized samples as code. The concrete details can be seen in Algorithm 1. Instead of using the density of the posterior and the prior to sample, we finely quantized them and used the probability masses in the algorithm.

Coding Simply writing the codes of the individual dimensions given by the rejection sampler would be very inefficient, because without additional assumptions the way to uniquely decode them would be to block code them (i.e. code all indices in 8 bits, say). This would however would add an $\mathcal{O}(1)$ cost per dimension on the coding length, which is very undesirable. Hence, we implemented a simple non-adaptive arithmetic coder (Rissanen and Langdon (1981)) to mitigate this issue. Probabilities for the symbol table have been estimated by encoding the entire CLIC training set and using the empirical probability distribution on the indices. We used Laplace/additive smoothing for unseen indices (Chen and Goodman (1999)). In particular, given the empirical distribution of the sample indices P, the probability distribution used is

$$\tilde{P}(i) = \begin{cases} (1-\alpha)P(i) & \text{if } i \in I \\ \frac{\alpha}{N} & \text{otherwise} \end{cases}$$

Algorithm 1 Parallelized, bit-budgeted rejection sampling based on Algorithm 4.

Inputs:
B - Bit budget for coding the rejection sample indices
P - Prior probability mass function
Q - Posterior probability mass function
$\langle x_i \sim Q \mid i \in \mathbb{N} \rangle$ - Sequence of random draws from Q

procedure Rej-Sampler($B, P, Q, \langle x_i \sim Q \mid i \in \mathbb{N} \rangle$)
  $D \leftarrow \dim(P)$
  $p_{d,0}(x) \leftarrow 0 \quad \forall x \in \mathcal{X}, \forall d = 1, \dots D.$
  $p^*_{d,0} \leftarrow 0, \forall d = 1, \dots D.$
  $A = \mathbf{0} \in \{0, 1\}^D$ ▷ Keeps track of whether a dimension has been accepted or not
  $S = \mathbf{0} \in \mathbb{R}^D$ ▷ Sample we are "building"
  $I = -\mathbf{1} \in \mathbb{N}^D$ ▷ Index vector for each dimension
  for $i \leftarrow 1, \dots 2^B$ **do**
    for $d \leftarrow 1, \dots D$ **do**
      if $A_d = 1$ **then**
        Skip
      end if
      $\alpha_{d,i}(x) \leftarrow \min P_d(x) - p_{d,i-1}(x), (1 - p^*_{d,i-1})Q_d(x) \quad \forall x \in \mathcal{X}$
      $p_{d,i}(x) \leftarrow p_{d,i-1}(x) + \alpha_{d,i}(x)$
      $p^*_{d,i} \leftarrow \sum_{x \in \mathcal{X}} p_{d,i}(x)$
      $\beta_{d,i}(x_i) \leftarrow \frac{\alpha_{d,i}(x)}{(1-p^*_{d,i})Q_d(x)}$
      Draw $u \sim \mathcal{U}(0, 1)$
      if $u < \beta_{d,i}(x_i)$ **then**
        $A_d \leftarrow 1$ ▷ Indicate we accepted the sample
        $S_d \leftarrow x_i$
        $I_d \leftarrow i$
      end if
    end for
  end for
  Draw $S' \sim Q_{\text{where } I=-1}$ ▷ Sample dimensions where we have not accepted.
  $S_{\text{where } I=-1} \leftarrow S'$ ▷ Set the "built" sample's missing dimensions to S'.
  $I_{\text{where } I=-1} \leftarrow \text{Quantize}(S', 16)$
  ▷ Set the missing dims. of I to S' quantized to 16 bits.
  return I, S
end procedure

where $I = \{i \in \mathbb{N} \mid P(i) > 0\}$. In our case we allocated $B = 8$ bits for each individual dimension, $I = \{0, \dots 255\}$ as all codable indices appeared. Since we quantized the outliers to 16 bits, $N = 2^{16} - 2^8$. We found that choosing $\alpha \approx 0.01$ gave the best results.

A note on the Arithmetic Coder While for small alphabets the naive implementation of the arithmetic coder is very fast, the decoding time actually grows as $\mathcal{O}(|\mathscr{A}|)$ in the size of the alphabet. In particular, decoding the arithmetic code of a reasonably large image would take up to 30 minutes using the naive implementation on a CPU. The inefficiency is due to a bottleneck where we need to find in a partition of $[0, 1]$ into which the currently coded interval fits. The naive uses a linear search over the partitions. This can be made more efficient by using height-balanced binary search trees (BSTs). In particular, we need to find the least upper bounding item in the BST for the given point, which can be done in $\mathcal{O}(\log_2 |\mathscr{A}|)$ time. Using this additional trick, we can decode large images' code in a few seconds. In particular we implemented an AVL-tree to serve as the BST, which is always guaranteed to be height balanced (Adel'son-Vel'skii and Landis (1962)).

Issues Theorem 2 guarantees that the MDL of a realization of a random variable Y given X by using Algorithm 4 is

$$L(X) \leq \mathrm{KL}\left[\, q(Y \mid X) \;||\; p(Y) \,\right] + 2 \log \left(\mathrm{KL}\left[\, q(Y \mid X) \;||\; p(Y) \,\right] + 1\right) + c,$$

for some small constant c. However, to achieve this for the joint latents $\mathbf{z}$, we would need to sample them jointly, which is intractable. Instead, we sample the dimensions individually, which by the same theorem introduces a $D \cdot c$ nat extra length to the code, where we assumed $\mathbf{z} \in \mathbb{R}^D$. This is because the constant cost c is now incurred in every dimension. Since in our case D is usually on the magintudes of $10^5 - 10^6$ even for small c this cost becomes non-negligible. In our experiments (not show here) this lead to coding efficiencies $2.5 - 3.5$ times worse than the optimal efficiency predicted by Theorem 2 for the whole multivariate sample. This lead us to abandon rejection sampling early on in the project for more efficient approximate methods, which we describe below.

3.4.2 Greedy Coded Sampling

To solve the issue of dimensionality, we need a way to code a sample drawn from the multivariate distribution. The strategy of the greedy sampler is to progressively "build" a reasonable sample from the posterior. A well known fact about Gaussian distributed random variables is that they are closed under addition. Concretely, if $X \sim \mathcal{N}\left(\mu_X, \sigma_X^2\right), Y \sim$

Algorithm 2 Greedy Coded Sampler

Inputs:
K - Number of proposal shards
B - Bit budget for coding the seed of individual shards.
Will result in 2^B samples to be examined per shard.
$\boldsymbol{\mu}_p$ - Mean of proposal distribution
$\boldsymbol{\sigma}_p^2$ - Variance of proposal distribution
q - Posterior distribution

procedure Greedy-Sampler($K, B, \boldsymbol{\mu}_p, \boldsymbol{\sigma}_p^2, q$)
  $\tilde{\mathbf{z}}_0 \leftarrow \mathbf{0}$ ▷ Initialize the sample
  $I = ()$ ▷ Initialize the index set to an empty list
  for $k = 1, \ldots, K$ **do**
    Set random seed of generator to k.
    Draw $\mathbf{s}_{k,b} \sim \mathcal{N}\left(\frac{\boldsymbol{\mu}_p}{K}, \frac{\boldsymbol{\sigma}_p}{K}\right)$ for $b = 1, \ldots, B$
    $\mathbf{c}_{k,b} = \tilde{\mathbf{z}}_{k-1} + \mathbf{s}_{k,b}$
    $\tilde{\mathbf{z}}_k \leftarrow \arg\max_{\mathbf{c}_{k,b}} \left\{\log q(\mathbf{c}_{k,b})\right\}$ ▷ Create new sample
    $i_k \leftarrow \arg\max_b \left\{\log q(\mathbf{c}_{k,b})\right\}$ ▷ Store the index of the shard sample that we used
    Append i_k to I.
  end for
  return $\tilde{\mathbf{z}}_K, I$
end procedure

$\mathcal{N}\left(\mu_Y, \sigma_Y^2\right)$, then

$$X + Y \sim \mathcal{N}\left(\mu_X + \mu_Y, \sigma_X^2 + \sigma_Y^2\right).$$

In the multivariate case assuming they are diagonal, the extension of the above is straight forward. Using this, we may now do the following: pick an integer K, that we will call the *number of shards*. Then, given the prior $p(\mathbf{z} \mid \boldsymbol{\mu}, \boldsymbol{\sigma}^2)$, we can break it up into K equal shards $p_k\left(\mathbf{z} \mid \frac{\boldsymbol{\mu}}{K}, \frac{\boldsymbol{\sigma}^2}{K}\right)$. Note, if we assign a different, but preagreed sequence of random seeds to each shard, then each shard sample can be coded in B bits that we can set. Start with an initial sample $\tilde{\mathbf{z}}_0 = \mathbf{0}$ and draw 2^B samples $\mathbf{s}_{1,b}$ from the first shard p_1 and create "candidate samples" $\mathbf{c}_{1,b} = \tilde{\mathbf{z}}_0 + \mathbf{s}_{1,b}$ and calculate their log-likelihoods under the target. Finally, set $\tilde{\mathbf{z}}_1 = \arg\max_{\mathbf{c}_{1,b}} \log q(\mathbf{c}_{1,b})$, and repeat until we reach $\tilde{\mathbf{z}}_K$, at which point we return it. Then the returned vector will be approximately from the target. More precisely, this is a "guided" random walk, where we bias the trajectory towards the median of the distribution. The code of the distribution is then the codes of the selected shard samples, concatenated. The algorithm is described in more detail in Algorithm 2.

A note on implementation We note that empirically the greedy sampler underperforms when the variances of the some priors is small. To ameliorate this, we standardize the prior, and scale the posterior according to the standardization, i.e. we set

$$q'(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}\left(\mathbf{z} \middle| \frac{\mu_q - \mu_p}{\sigma_p}, \frac{\sigma_q^2}{\sigma_p^2}\right),$$

where μ_p, σ_p^2 are the statistics of the original prior and μ_q, σ_q^2 are the statistics of the original posterior. We communicate the approximate sample $\mathbf{z}'$ from q' instead of q. This is not problematic, as Gaussian distributed random variables are closed under linear transformations, i.e. given $X \sim \mathcal{N}\left(m, s^2\right)$, we have

$$\alpha X + \beta = Y \sim \mathcal{N}\left(\alpha m + \beta, \alpha^2 s^2\right).$$

Hence, the decoder may recover an approximate sample from q, by calculating $\mathbf{z} = \sigma_p \mathbf{z}' + \mu_p$.

Issues While the greedy sampler makes sampling efficient and tractable from the posterior, it comes at the cost of reduced sample quality. In particular, it gives blurrier images. This also means that if we use a PLN to compress an image and we use the greedy technique to code the latents on the second level, the first level priors' statistics derived from the biased sample will be off, and $\mathrm{KL}\left[q(\mathbf{z}^{(1)} \mid \tilde{\mathbf{z}}^{(2)}, \mathbf{x}) \;||\; p(\mathbf{z}^{(1)} \mid \tilde{\mathbf{z}}^{(2)})\right]$ will be higher. We have verified empirically, that while using a biased sample on the second level does not degrade image quality (possibly

due to the noise tolerance of PLNs), it does significantly increase the compression size (by a factor of $1.2 - 1.5$) of the first level, which is very significant. This motivated the final sampling algorithm presented here, only used on the second level of our PLNs.

3.4.3 Adaptive Importance Sampling

The adaptive importance sampler uses the importance sampler described in Algorithm 5, introduced by Havasi et al. (2018). The idea is to use similar block-based importance sampling as proposed in Havasi et al. (2018). However, unlike them, we allocate the block sizes dynamically. In particular, we set a bit budget B per group, a maximum group size G and an individual limit K. We begin by discarding individual dimensions where the KL is larger than K. Then, we flatten the remaining dimensions in raster-scan order and iteratively add dimensions into the current group so long as the total KL of the group reaches either the bit budget B or the number of dimensions added to the group reaches G. At this point we start with a new group. Once the whole vector has been partitioned, we importance sample each group using Algorithm 5. The removed dimensions are sampled directly from the posterior and then the samples are quantized to 16 bits. The complete algorithm can be seen in Algorithm 3.

For ease of referral, since we would perform Adaptive Importance Sampling on the second level, followed by the Greedy Sampling on the first, We will refer to this way of sample coding as IS-GS.

For the importance sampler, the $\mathcal{O}(1)$ sampling cost impeding the rejection sampler is negligible, as it will be now shared across approximately G dimensions. We can also forgo the arithmetic coder, as the indices output by the sampler are already going to be quite efficient. On the other hand, we also have to communicate the groups as well. Instead of communicating group indices, we can instead order the latents and communicate consecutive group sizes, from which the indices can be easily reconstructed. Each group's size takes up at most $\lceil \log_2 G \rceil$ bits. In total, we usually get slightly more than $\left\lceil \frac{N_2}{G} \right\rceil$ groups, (where N_2 is the dimensionality of the second level). This is still very inefficient, but so long as N_2 is sufficiently small compared to the codelength for the indices, it will be manageable. We also note that the group size is likely to be biased towards higher values, and hence building an empyrical distribution them and arithmetic coding the sequence could lead to a big reduction in the inefficiency, however, we found this not to be too important to focus on.

Greedy Sampler It is easy to see that the greedy sampler is already as efficient as it could be. The sample indices for each shard are as compactly represented, as we expect the max-

Algorithm 3 Adaptive Importance Sampler based on Algorithm 5, introduced by Havasi et al. (2018)

Inputs:
K - Maximum individual KL allowed
G - Maximum group size
B - Bit budget per group
P - Proposal distribution
Q - Target Distribution
$\langle x_i \sim Q \mid i \in \mathbb{N} \rangle$ - Shared sequence of random draws from Q

procedure Adaptive-Importance-Sampler($K, G, B, P, Q, \langle x_i \sim Q \mid i \in \mathbb{N} \rangle$)
  $\Gamma \leftarrow ()$ ▷ Group sizes
  $kl_i \leftarrow \mathrm{KL}\left[Q_i \mid\mid P_i \right] \forall i = 1, \ldots N$ ▷ Get KLs for each dimension
  $OI \leftarrow \mathrm{Where}(kl_i > K)$ ▷ Outlier indices in the vector
  Sample $O \sim Q_{OI}$
  $\hat{O} \leftarrow \mathrm{Quantize(O)}$
  $Q' \leftarrow Q_{\setminus OI}$ ▷ Target distribution restricted to the dimensions defined by OI.
  $P' \leftarrow P_{\setminus OI}$ ▷ Remove outlier dimensions
  $\gamma \leftarrow 0$ ▷ Current group size
  $k \leftarrow 0$ ▷ Current group KL
  for $i \leftarrow 1, \ldots \dim(Q')$ **do**
    if $k + kl_i > B$ or $\gamma + 1 > G$ **then**
      Append γ to Γ
      $k \leftarrow kl_i$
      $\gamma \leftarrow 1$
    else
      $k \leftarrow k + kl_i$
      $\gamma \leftarrow \gamma + 1$
    end if
  end for
  Append γ to Γ ▷ Append the last group size
  $S = ()$ ▷ Importance samples for the groups
  $I = ()$ ▷ MIRACLE sample indices
  $g \leftarrow 0$ ▷ Current group index
  for γ in Γ **do** ▷ Now importance sample each group
    $idx, s \leftarrow \text{Importance-Sample}(P_{g:g+\gamma}, Q_{g:g+\gamma}, \langle x_i \sim Q \mid i \in \mathbb{N} \rangle)$
    Append idx to I
    Append s to S
  end for
  return I, S
end procedure

imum of the samples to be uniformly distributed. Hence, the only other things that needs to be coded is the number of shards and the number of samples for each shard for the cumulative sample to be decodable. Hence, for the greedy sampler we just wrote the indices straight to the binary file.

Chapter 4

Results

In this section we detail how we setup and empiricall show the correctness and the efficiency of our model. We compare our results against JPEG, the most widely used lossy compression method Bull (2014), and the current state-of-the-art, the results of Ballé et al. (2018)[1]. Zhao et al. (2015)

Note: All experiments were run on a GeForce GTX 1080 GPU.

4.1 Experimental Setup

As we based our models on that of Ballé et al. (2016b) and Ballé et al. (2018), we mirror a lot of their training setup as well (See Section 3.1 for the dataset and preprocessing). We trained all our models with Adam with a starting learning rate of $\alpha_0 = 3 \times 10^{-5}$ and trained all of our models for 20 epochs or equivalently, approximately 200,000 iterations.. We used a smooth exponential learning rate decay schedule except in the case of the γ-VAEs, according to the formula

$$\alpha(t) = \alpha_0 \times r^{\frac{t}{D}}.$$

Where r is the decay rate, D is the decay step size and t is the current batch number. We found $r = 0.96$ and $D = 1500$ worked well for our experiments. We note, however, that we did not notice signifcant performance gains by using this schedule compared to just using a fixed one.

A surprising result is that even though we have not trained our models for nearly as long, or on nearly as much data as Ballé et al. (2018), our method still gets reasonably close to

[1] We thank the authors of the paper for making their data available to us.

their results. We compare our results to theirs, which as far as we are aware are the current state-of-the-art on both the MS-SSIM and PSNR perceptual metrics.

4.2 Comparison of our method with other algorithms

We present the rate-distorsion curves for the following:

- JPEG, with quality settings from 1 to 92, with increments of 7 between settings. As this is the most widely used lossy image compression codec, it is crucial to demonstrate that our method is at least competitive with it, and ideally beats it.
- BPG[2]with 4:4:4 chroma sampling, as we are comparing against RGB-based compression techniques. We used quantization settings between 51 to 33 wiht decrements of 3 between settings.
- Two models with the same architecture from Ballé et al. (2018), one optimized for a MSE training objective, and one optimized for the MS-SSIM perceptual metric.
- Two of our models, all of which were optimized with Laplacian likelihoods, one PLN and one γ-PLN. We plot both their theoretically optimal performance as well as their actual performance, with the differences explained below.

For our method, for each model we present two results: the *theoretically optimal* performance, and the *actual* performance. The theoretically optimal BPP was calculated using the theoretically achievable upper bound for the compression size in bits as given by Harsha et al. (2007), without the constant term:

$$\mathbb{I}[\mathbf{x} : \mathbf{z}] + 2 \log \left(\mathbb{I}[\mathbf{x} : \mathbf{z}] + 1 \right).$$

The optimal reconstruction error was calculated by passing an image through the VAE regularly, instead of using the coded approximate sample. Thus, any actual method's performance using the same setup must apper to the right of (less efficient compression) or below (worse reconstruction quality) the theoretical position.

We trained the PLNs using $\beta = \{0.01, 0.03, 0.1, 0.3, 1\}$ and the γ-PLNs using $\beta = \{10, 3, 1, 0.3, 0.1\}$.

Our results can be seen in Figure 4.1. We observe a similar phenomenon as Ballé et al. (2018): there is a mismatch in the comparison of models according to different perceptual

[2]We used the implementation available at http://bellard.org/bpg

metrics, depending on what objective they have been optimized for. In particular, JPEG and BPG have both been optimized so that they give high PSNR (thus, low MSE), whereas they underperform on the newer MS-SSIM metric. We note that as MS-SSIM correlates better with what the HVS perceives, we find it more important to do well on that comparison.

In interesting note, that also justifies our choice of the MAE as the training objective, is the fact that our model optimized for it does well on this metric.

4.3 Analysis of the contribution of the second level

An important part of verifying the validity of using PLNs is to analyze the contribution of the second level. Here we look at

- its contribution to the codelength
- its efficiency in capturing dependencies between the first level latents

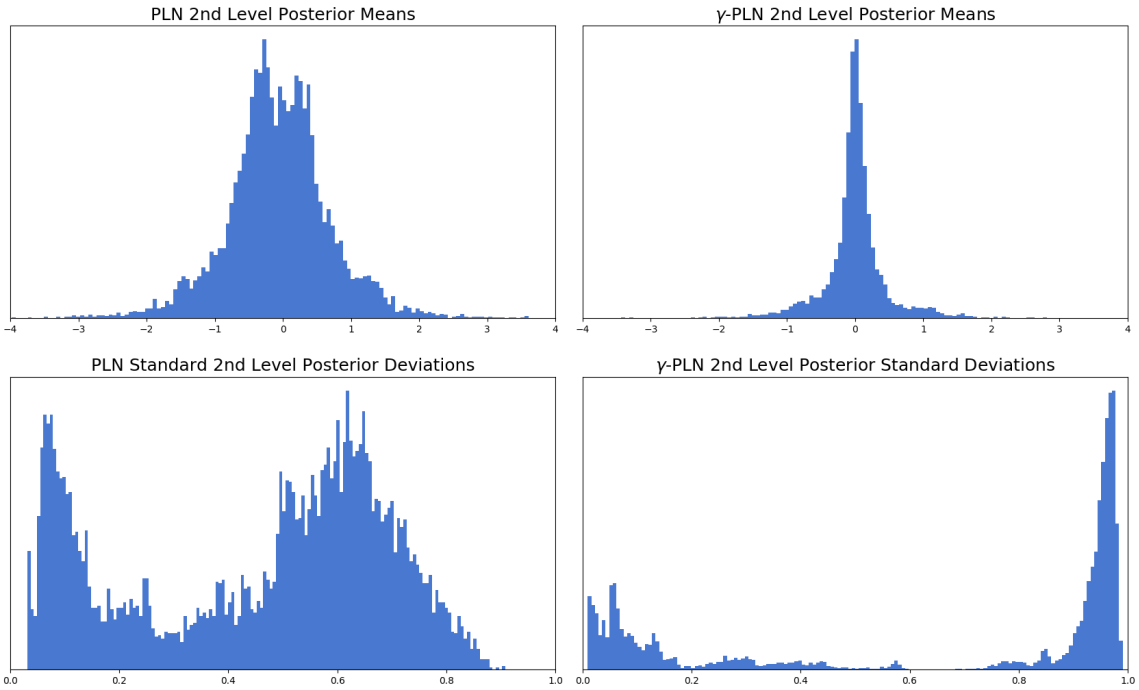


Fig. 4.3 ladder on kodim21

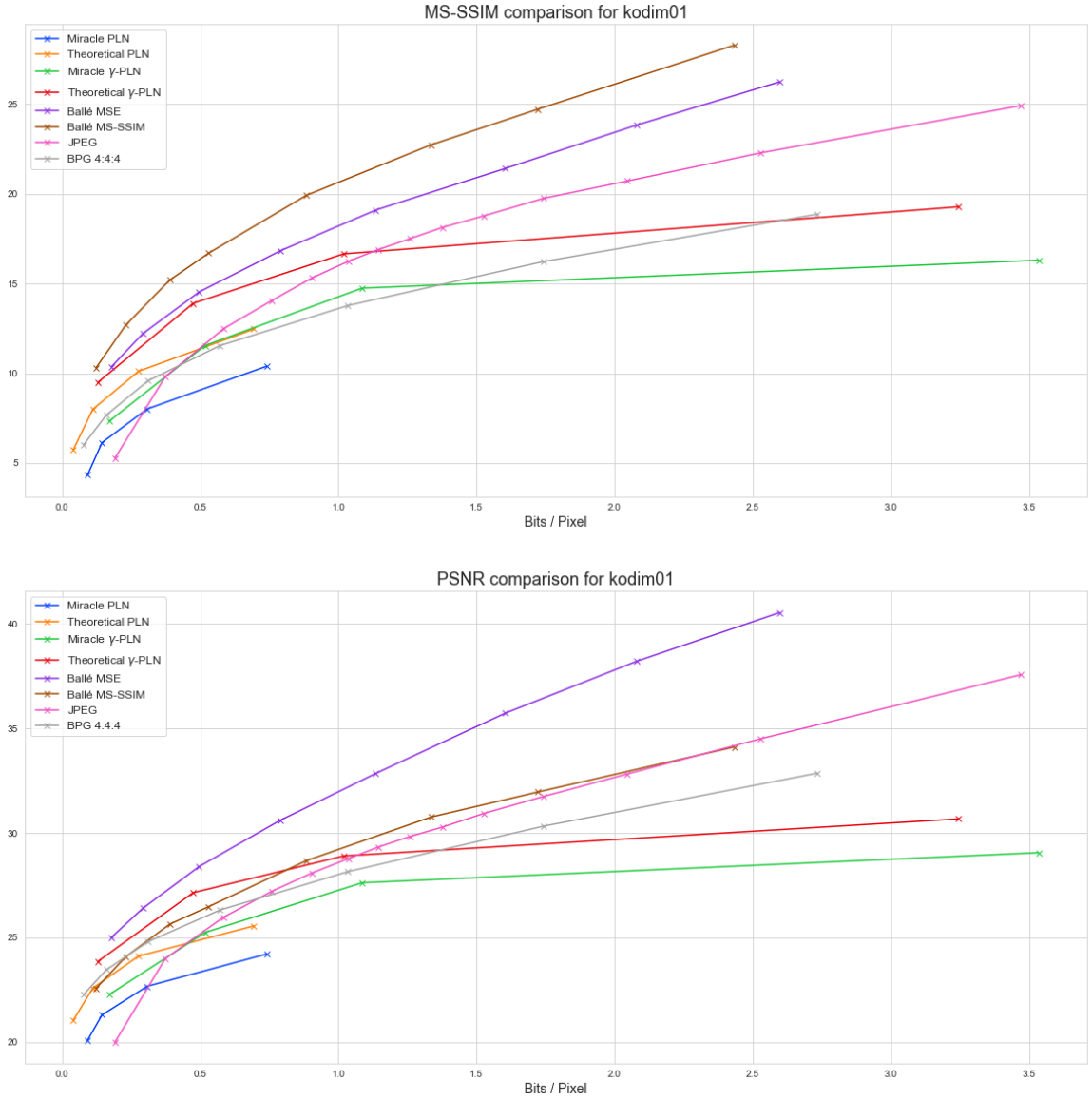


Fig. 4.1 Rate-Distorsion curves of several relevant methods. Please see Section 4 for the description of how we obtained each curve. We note that the MS-SSIM results are presented in decibels, where the conversion is done using the formula $-10 \cdot \log_{10}(1 - \text{MS-SSIM}(\mathbf{x}, \hat{\mathbf{x}}))$. The PSNR is computed from the mean squared error, using the formula $-10 \cdot \log_{10} \text{MSE}(\mathbf{x}, \hat{\mathbf{x}})$.

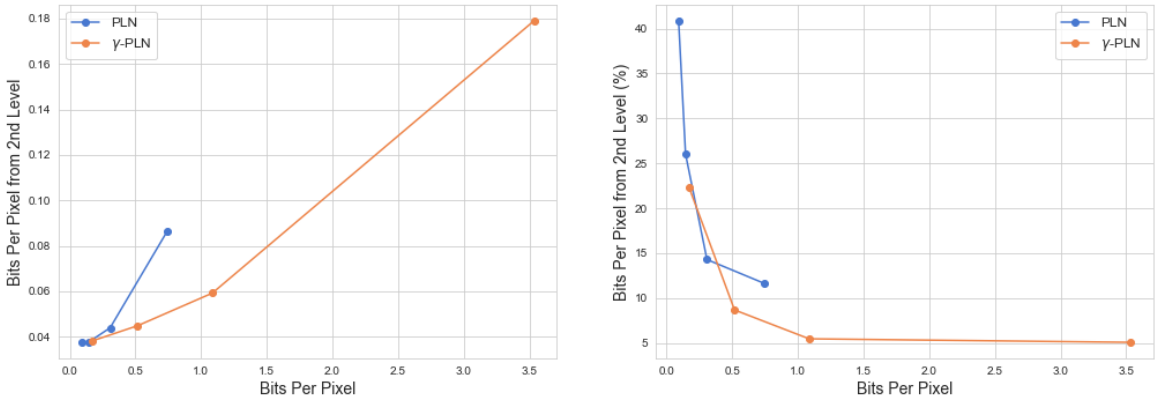


Fig. 4.2 Contribution of the second level to the rate, plotted agains the actual rate. **Left:** Contribution in BPP, **Right:** Contribution in percentages. We see that for lower bitrates there is more contribution from the second level and it quickly decreases for higher rates. It is also clear that on the same bitrates, the γ-PLN requires less contribution from the second level than regular PLN.

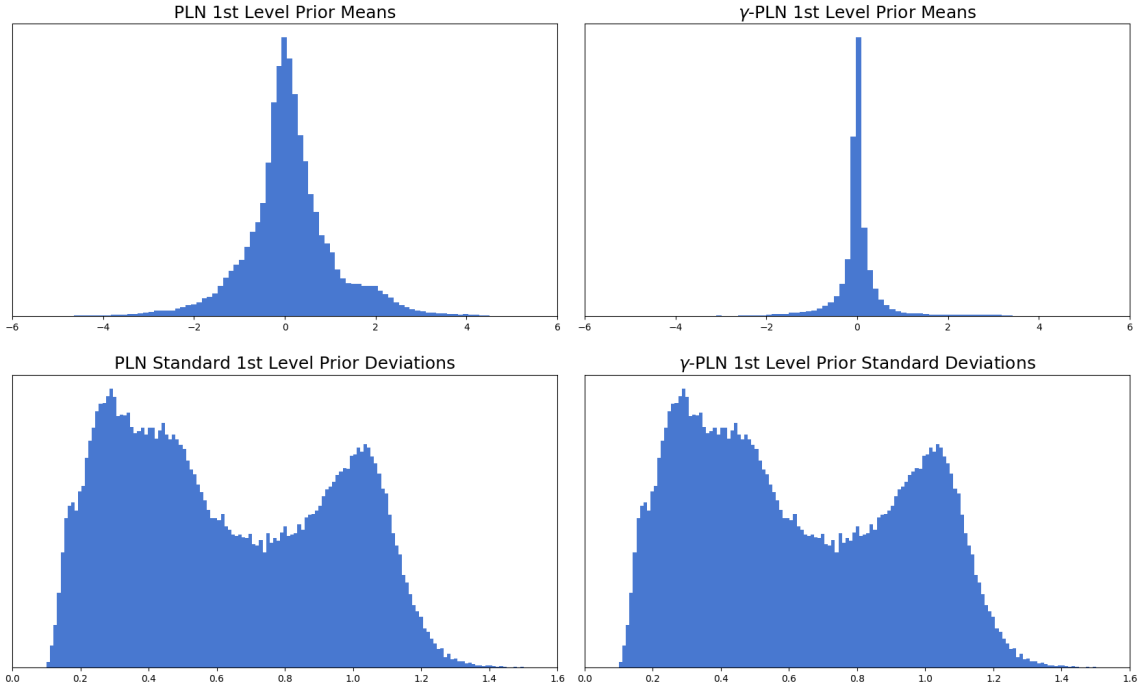


Fig. 4.4 ladder on kodim21

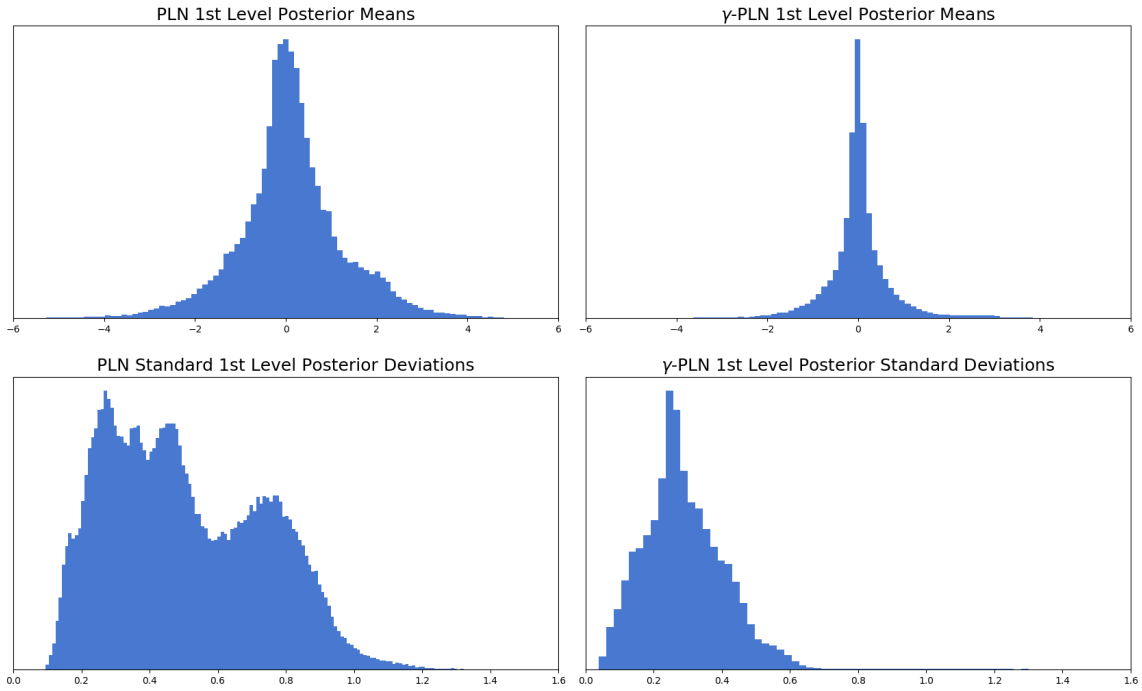


Fig. 4.5 ladder on kodim21

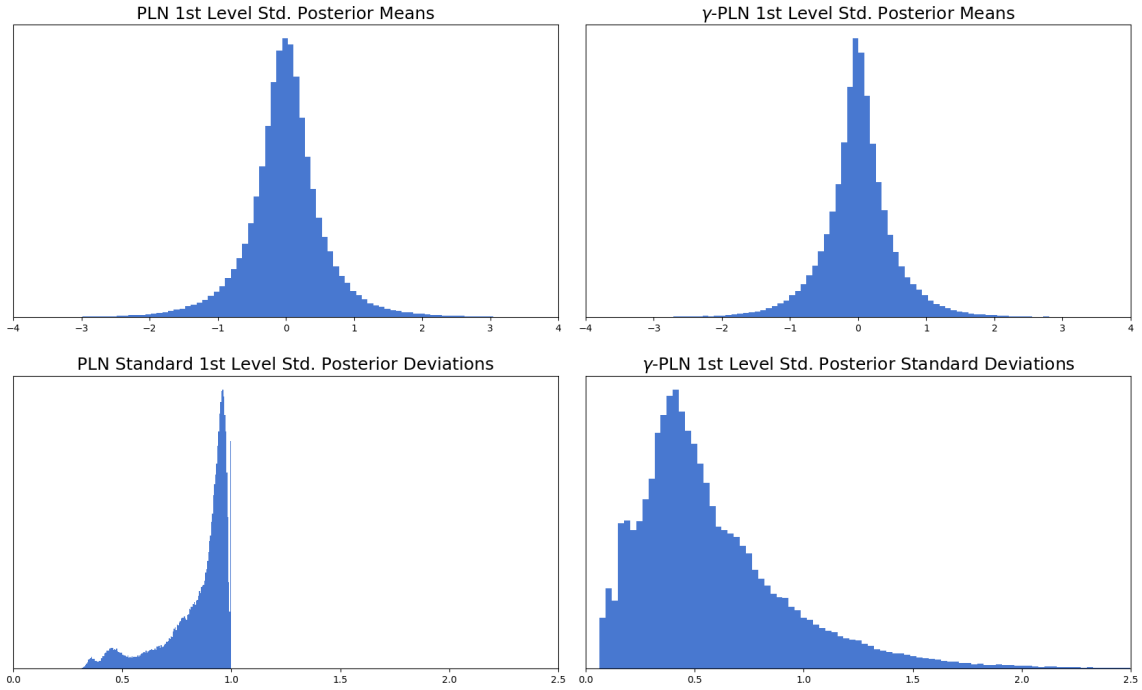


Fig. 4.6 ladder on kodim21

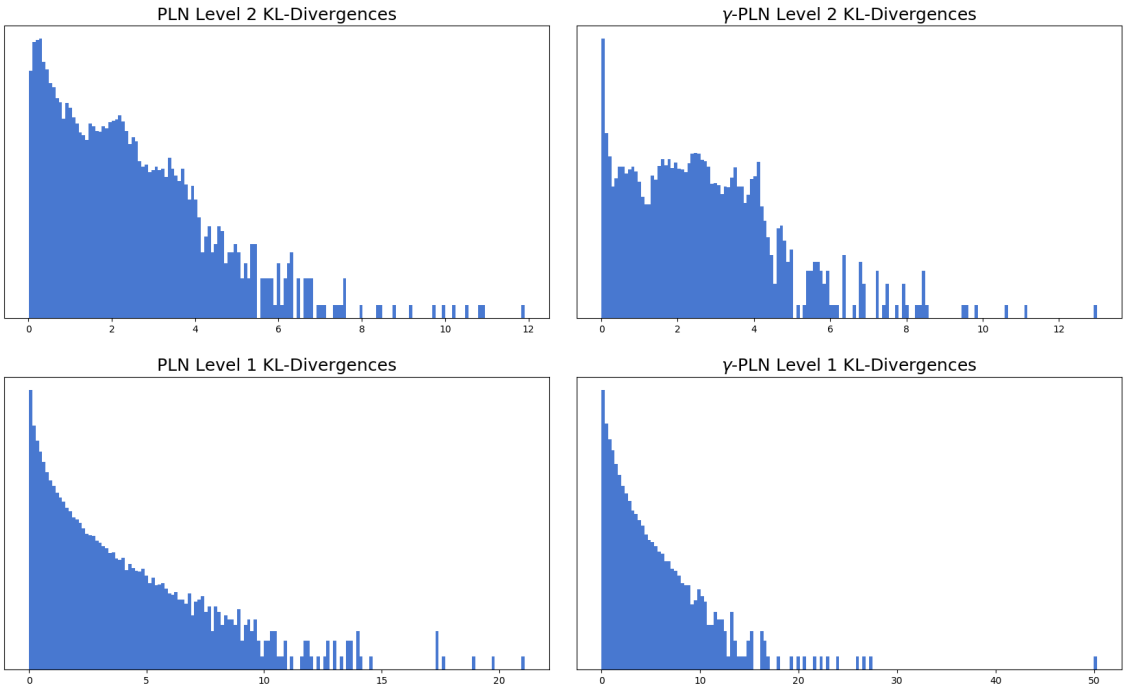


Fig. 4.7 ladder on kodim21

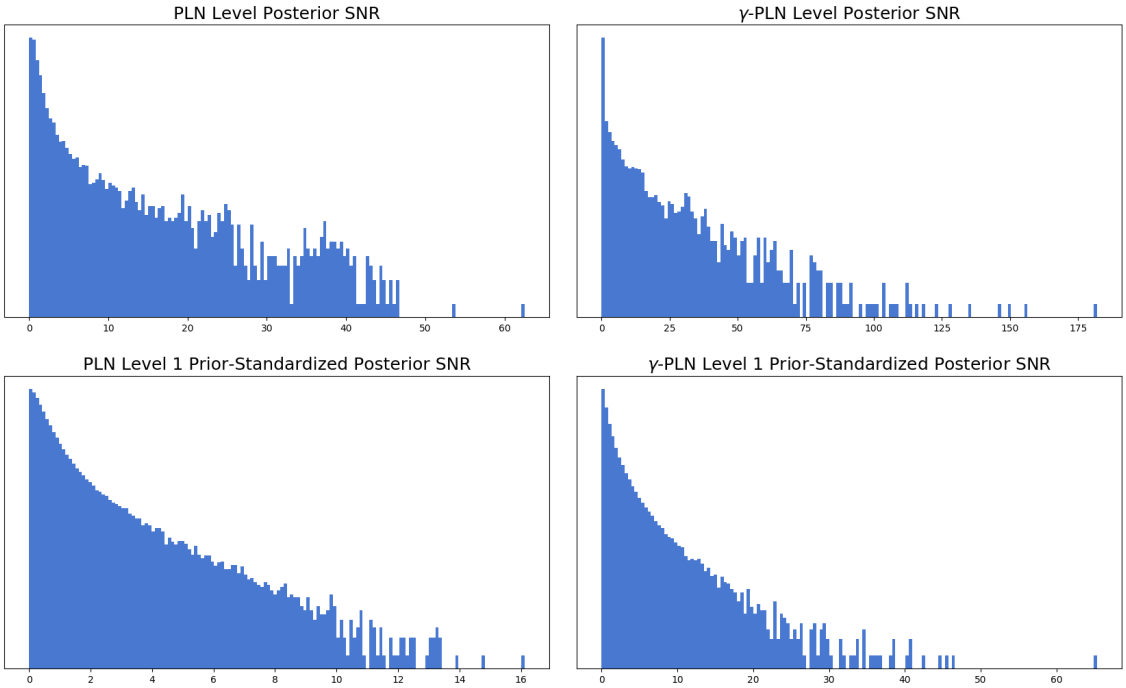


Fig. 4.8 ladder on kodim21

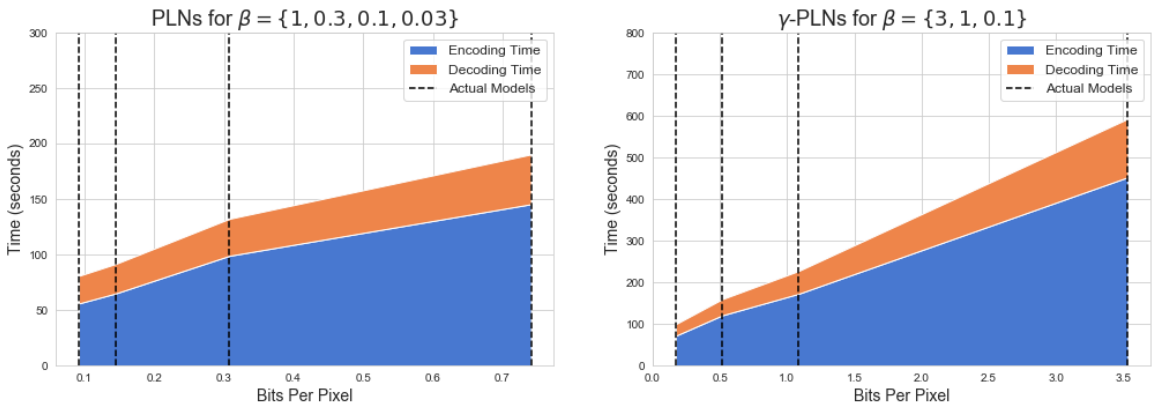


Fig. 4.9 Coding times of models plotted agains their rates. **Left:** Regular PLNs. **Right:** γ-PLNs. The striped lines indicate the concrete positions of our models in the rate line. While it seems that there is a linear relationship between rate and coding time, we do not have enough datapoints to conclude this.

	PLNs				**γ-PLNs**			
β	1	0.3	0.1	0.03	10	3	1	0.1
Encoding Time (s)	55.91	64.95	98.85	145.38	71.40	120.54	172.34	452.49
Decoding Time (s)	24.85	26.61	33.34	44.85	27.81	38.87	54.86	140.52

Table 4.1 haha

4.4 Compression Speed

Although not a focus of our project, we now briefly examine the the encoding and decoding speed of our method. We have plotted the compression ratios of our models against the time it took them to encode / decode them using IS-GS in Figure 4.9. As increasing the reconstruction quality leads to higher KL divergences between the latent posteriors and priors, both the importance sampler and the greedy sampler will need to split up a higher total KL, and thus we expect the coding to become slower. This is precisely what we observe, with a seemingly approximately linear growth, although we do not have data to conclude this. We also see that encoding consistently takes around 3 times as long as decoding. It is clear that our method is not yet practical: even the fastest case takes around a minute to encode and about 20 seconds to decode, which very far away for real-time applications for now. The precise values are reported in Table 4.1.

Chapter 5

Conclusion

5.1 Discussion

In this work, we gave an introduction to image compression and machine learning based compression. Based on the MDL principle (Rissanen (1986)) and the Bits-back argument (Hinton and Van Camp (1993)), as well as more recent work in information theory (Harsha et al. (2007))and neural network compression (Havasi et al. (2018)), we developed a general lossy compression framework, and described how previous quantization based approaches fit into it. We gave a comparative review of recent influential works in the field of neural image compression. We demonstrated the efficiency of the framework we developed by applying it to image compression. We trained several Probabilistic Ladder Networks, optimized for different rate-distrotion trade-offs, and achieved results close to the current state of the art. We also presented 3 coded sampling algorithms with different advantages and disadvantages that may be used to compress data using our framework. We present detailed analysis supporting our model choices.

5.2 Future Work

There are many aspects that should be considered for a practical compression algorithm. Some of these are:

- Quality,
- Rate,
- Speed,

- Memory footprint,
- Power consumption of compression and decompression,
- Robustness of the compressor (i.e. resistance to errors or adversarial attacks),
- Security / privacy of compressed representation,
- Scalability e.g. in terms of image size.

In this work we focused only on the first two items and also propose future directions along these lines, although some improvements could help the other items as well.

5.2.1 Data Related Improvements

We have trained our models on the training dataset of the CLIC 2018 dataset (CLIC (2018)), which consists of 585 images. This is quite meagre, and thus increasing the dataset size could lead to large improvements in performance. In particular, using γ-PLNs might be prone to overfitting to some degree, especially in conjunction with small βs. A larger dataset could be gathered from flickr.com similarly to Theis et al. (2017), and reduce the risk of overfitting.

5.2.2 Model Related Improvements

Architecture In this thesis, we selected Probabilistic Ladder Networks (PLNs) and γ-PLNs and showed that with standard training techniques and no fine-tuning we may achieve rate-distrotion results close to the current state-of-the-art. Thus, finding a better fitting architecture and fine-tuning our models, e.g. in terms of number of layers, convolution filters per layer, latent dimension size could greatly increase the efficiency of the network. Exploring the contributions of further stochastic layers, or residual connections might also be a fruitful direction.

Loss As shown by Zhao et al. (2015), the training loss used is crucial for the perceptual quality of the reconstructed image. In this work, we trained using an L_1 loss which is equivalent to a Laplacian data likelihood given the latents. An interesting line of research could be investigating more complex losses, e.g. the mixture loss proposed by Zhao et al. (2015), or using an extended transform coding pipeline, where the distrotion between the original and reconstructed images is measured in a transformed space, as proposed by Ballé et al. (2016b). A simple example would be to use the *VGG-19 loss*, where both images would be passed through the VGG-19 classifier, and an L_2 loss is measured between the activations

of certain convolutional layers (Johnson et al. (2016)). Adversarial losses, like the one used in Rippel and Bourdev (2017) might also be interesting to try.

Latent Representations We used Gaussians to represent the distribution of the latents space, mainly because this allowed a simple extension from VAEs to PLNs in our case. However, as discussed earlier, filter responses of natural images are much better represented as Laplacians or Gaussian Scale Mixtures (Portilla et al. (2003)). Hence, it may be worthwhile to investigate if PLNs or similar models could be extended to allow for these distribution in a mathematically sound fashion, still relying on conjugacy, but perhaps not requiring a self-conjugate distribution.

5.2.3 Coding Related Improvements

As we have seen, currently both the output quality of and sampling speed of our current algorithms is suboptimal. In particular it is currently unclear how the postulated bits-back efficiency of the joint latent posterior could be achieved in a tractable manner. An interesting candidate could be A^* sampling (Maddison et al. (2014)) for coding, however, this method also suffers from the curse of dimensionality.

Another open question is whether index-based codes using variants of rejection sampling are the only bits-back efficient codes.

References

Adel'son-Vel'skii, G. M. and Landis, E. M. (1962). An algorithm for organization of information. In *Doklady Akademii Nauk*, volume 146, pages 263–266. Russian Academy of Sciences.

Asuni, N. and Giachetti, A. (2014). Testimages: a large-scale archive for testing visual devices and basic image processing algorithms. In *Eurographics Italian Chapter Conference*, volume 1, page 3.

Ballé, J., Laparra, V., and Simoncelli, E. P. (2015). Density modeling of images using a generalized normalization transformation. *arXiv preprint arXiv:1511.06281*.

Ballé, J., Laparra, V., and Simoncelli, E. P. (2016a). End-to-end optimization of nonlinear transform codes for perceptual quality. In *2016 Picture Coding Symposium (PCS)*, pages 1–5. IEEE.

Ballé, J., Laparra, V., and Simoncelli, E. P. (2016b). End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*.

Ballé, J., Minnen, D., Singh, S., Hwang, S. J., and Johnston, N. (2018). Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*.

Bishop, C. (2013). *Pattern Recognition and Machine Learning*. Information science and statistics. Springer (India) Private Limited.

Bishop, C. M. (1998). Latent variable models. In *Learning in graphical models*, pages 371–403. Springer.

Bottou, L., Haffner, P., Howard, P. G., Simard, P., Bengio, Y., and LeCun, Y. (1998). High quality document image compression with djvu.

Bull, D. (2014). *Communicating Pictures: A Course in Image and Video Coding*. Elsevier Science.

Chen, S. F. and Goodman, J. (1999). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13(4):359–394.

CLIC (2018). Workshop and challenge on learned image compression. https://www.compression.cc. Accessed: 2019-03-25.

Dai, B. and Wipf, D. (2019). Diagnosing and enhancing vae models. *arXiv preprint arXiv:1903.05789*.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Denton, E. L., Chintala, S., Fergus, R., et al. (2015). Deep generative image models using a□ laplacian pyramid of adversarial networks. In *Advances in neural information processing systems*, pages 1486–1494.

Eastman Kodak Company (1999). Kodak lossless true color image suite. http://r0k.us/graphics/kodak/.

Eskicioglu, A. M., Fisher, P. S., and Chen, S.-Y. (1994). Image quality measures and their performance.

Girod, B. (1993). What's wrong with mean-squared error? *Digital images and human vision*, pages 207–220.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

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

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. (2015). Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*.

Grünwald, P., Grunwald, A., and Rissanen, J. (2007). *The Minimum Description Length Principle*. Adaptive computation and machine learning. MIT Press.

Gupta, P., Srivastava, P., Bhardwaj, S., and Bhateja, V. (2011). A modified psnr metric based on hvs for quality assessment of color images. In *2011 International Conference on Communication and Industrial Application*, pages 1–4. IEEE.

Harsha, P., Jain, R., McAllester, D., and Radhakrishnan, J. (2007). The communication complexity of correlation. In *Twenty-Second Annual IEEE Conference on Computational Complexity (CCC'07)*, pages 10–23. IEEE.

Havasi, M., Peharz, R., and Hernández-Lobato, J. M. (2018). Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-vae: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.

Hinton, G. and Van Camp, D. (1993). Keeping neural networks simple by minimizing the description length of the weights. In *in Proc. of the 6th Ann. ACM Conf. on Computational Learning Theory*. Citeseer.

Huffman, D. A. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

Huynh-Thu, Q. and Ghanbari, M. (2008). Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*.

Jain, A. K. (1989). *Fundamentals of digital image processing*. Englewood Cliffs, NJ: Prentice Hall,.

Jiang, J. (1999). Image compression with neural networks–a survey. *Signal processing: image Communication*, 14(9):737–760.

Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *European conference on computer vision*, pages 694–711. Springer.

Johnston, N., Vincent, D., Minnen, D., Covell, M., Singh, S., Chinen, T., Jin Hwang, S., Shor, J., and Toderici, G. (2018). Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Karush, W. (2014). Minima of functions of several variables with inequalities as side conditions. In *Traces and Emergence of Nonlinear Programming*, pages 217–245. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

Kuhn, H. W. and Tucker, A. W. (2014). Nonlinear programming. In *Traces and emergence of nonlinear programming*, pages 247–258. Springer.

MacKay, D., Kay, D., and Press, C. U. (2003). *Information Theory, Inference and Learning Algorithms*. Cambridge University Press.

Maddison, C. J., Tarlow, D., and Minka, T. (2014). A* sampling. In *Advances in Neural Information Processing Systems*, pages 3086–3094.

Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Van Gool, L. (2018). Conditional probability models for deep image compression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Mougeot, M., Azencott, R., and Angeniol, B. (1991). Image compression with back propagation: improvement of the visual restoration using different cost functions. *Neural networks*, 4(4):467–476.

Portilla, J., Strela, V., Wainwright, M. J., and Simoncelli, E. P. (2003). Image denoising using scale mixtures of gaussians in the wavelet domain. *IEEE Trans Image Processing*, 12(11).

Rabbani, M. and Joshi, R. (2002). An overview of the jpeg 2000 still image compression standard. *Signal processing: Image communication*, 17(1):3–48.

Rippel, O. and Bourdev, L. (2017). Real-time adaptive image compression. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2922–2930. JMLR. org.

Rissanen, J. (1986). Stochastic complexity and modeling. *The annals of statistics*, pages 1080–1100.

Rissanen, J. and Langdon, G. (1981). Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1):12–23.

Shannon, C. E. and Weaver, W. (1998). *The mathematical theory of communication*. University of Illinois press.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). How to train deep variational autoencoders and probabilistic ladder networks. In *33rd International Conference on Machine Learning (ICML 2016)*.

Theis, L., Shi, W., Cunningham, A., and Huszár, F. (2017). Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*.

Toderici, G., O'Malley, S. M., Hwang, S. J., Vincent, D., Minnen, D., Baluja, S., Covell, M., and Sukthankar, R. (2015). Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*.

Toderici, G., Vincent, D., Johnston, N., Jin Hwang, S., Minnen, D., Shor, J., and Covell, M. (2017). Full resolution image compression with recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5306–5314.

Wallace, G. K. (1992). The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv.

Wang, Z., Bovik, A. C., Sheikh, H. R., Simoncelli, E. P., et al. (2004). Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612.

Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. In *The Thirty-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee.

Zhao, H., Gallo, O., Frosio, I., and Kautz, J. (2015). Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861*.

Zhou, L., Cai, C., Gao, Y., Su, S., and Wu, J. (2018). Variational autoencoder for low bit-rate image compression. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

Appendix A

Appendix A

Rejection Sampling

The rejection sampling algorithm presented here is due to Harsha et al. (2007).

Algorithm 4 Rejection sampling presented in Harsha et al. (2007).

procedure Rej-Sampler($P, Q, \langle x_i \sim Q \mid i \in \mathbb{N} \rangle$) ▷ P is the prior ▷ Q is the posterior ▷ x_i are i.i.d. samples from Q

  $p_0(x) \leftarrow 0 \quad \forall x \in \mathcal{X}$.
  $p_0^* \leftarrow 0$.
  for $i \leftarrow 1, \ldots \infty$ **do**
    $\alpha_i(x) \leftarrow \min P(x) - p_{i-1}(x), (1 - p_{i-1}^*)Q(x) \quad \forall x \in \mathcal{X}$
    $p_i(x) \leftarrow p_{i-1}(x) + \alpha_i(x)$
    $p_i^* \leftarrow \sum_{x \in \mathcal{X}} p_i(x)$
    $\beta_i(x_i) \leftarrow \frac{\alpha_i(x)}{(1-p_i^*)Q(x)}$
    Draw $u \sim \mathcal{U}(0, 1)$

    if $u < \beta_i(x_i)$ **then**
      return i, x_i
    end if
  end for
end procedure

Algorithm 5 Importance sampling algorithm proposed by Havasi et al. (2018)

procedure Importance-Sampler($P, Q, \langle x_i \sim Q \mid i \in \mathbb{N} \rangle$) ▷ P is the prior
▷ Q is the posterior
▷ x_i are i.i.d. samples from Q

$K \leftarrow \exp\{\mathrm{KL}\,[\,Q \;||\; P\,]\}$
$\tilde{w}_i \leftarrow \frac{Q(x_i)}{P(x_i)} \quad \forall i = 1, \ldots K$
Sample $j \sim p(\tilde{w})$
return j, x_j

end procedure

Appendix B

Appendix B: Images