END-TO-END LEARNED IMAGE COMPRESSION WITH CONDITIONAL LATENT SPACE MODELLING FOR ENTROPY CODING

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

AZİZ BERKAY YEŞİLYURT

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2019

Approval of the thesis:

END-TO-END LEARNED IMAGE COMPRESSION WITH CONDITIONAL LATENT SPACE MODELLING FOR ENTROPY CODING

submitted by **AZİZ BERKAY YEŞİLYURT** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronics Engineering Department, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of Department, **Electrical and Electronics Engineering** _____

Assoc. Prof. Dr. Fatih Kamışlı
Supervisor, **Electrical and Electronics Engineering, METU** _____

Examining Committee Members:

Prof. Dr. A. Aydın Alatan
Electrical and Electronics Engineering, METU _____

Assoc. Prof. Dr. Fatih Kamışlı
Electrical and Electronics Engineering, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Elif Vural
Electrical and Electronics Engineering, METU _____

Assist. Prof. Dr. Gökhan Koray Gültekin
Electrical and Electronics Engineering, AYBU _____

Date:

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Aziz Berkay Yeşilyurt

Signature :

ABSTRACT

END-TO-END LEARNED IMAGE COMPRESSION WITH CONDITIONAL LATENT SPACE MODELLING FOR ENTROPY CODING

Yeşilyurt, Aziz Berkay
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Fatih Kamışlı

September 2019, 94 pages

This thesis presents a lossy image compression system based on an end-to-end trainable neural network. Traditional compression algorithms use linear transformation, quantization and entropy coding steps that are designed based on simple models of the data and are aimed to be low complexity. In neural network based image compression methods, the processing steps, such as transformation and entropy coding, are performed using neural networks. The use of neural networks enables transforms or probability models for entropy coding that can optimally process or represent data with much more complex dependencies instead of simple models, all at the expense of higher computational complexity than traditional methods.

One major line of work on neural network based lossy image compression uses an autoencoder-type neural network for the transform and inverse transform of the compression system. The quantization of the latent variables, i.e. transform coefficients, and the arithmetic coding of the quantized latent variables are done with traditional methods. However, the probability distribution of the latent variables, which the arithmetic encoder works with, is represented also with a neural network. Parameters of

all neural networks in the system are learned jointly from a training set of real images by minimizing the rate-distortion cost.

One major work assumes the latent variables in a single channel (i.e. feature map or signal band) are independent and learns a single distribution model for each channel. The same authors then extend their work by incorporating a hyperprior neural network to capture the dependencies in the latent representation and improve the compression performance significantly. This thesis uses an alternative method to exploit the dependencies of the latent representation. The joint density of the latent representation is modeled as a product of conditional densities, which are learned using neural networks. However, each latent variable is not conditioned on all previous latent variables as in the Chain rule of factoring joint distributions, but only on a few previous variables, in particular the left, upper and upper-left spatial neighbors of that latent variable based on Markov property assumption. The compression performance is on par with the hyperprior based work, but the conditional densities require a much simpler network than the hyperprior network in the literature. While the conditional densities require much less training time due to their simplicity and less number of parameters than the hyperprior based neural network, their inference time is longer.

Keywords: image compression, transform coding, deep learning, conditional modeling

ÖZ

UÇTAN UCA ÖĞRENİLMİŞ GÖRÜNTÜ SIKIŞTIRMA VE ENTROPİ KODLAMA İÇİN GİZLİ UZAYIN KOŞULLU MODELLENMESİ

Yeşilyurt, Aziz Berkay
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Fatih Kamışlı

Eylül 2019 , 94 sayfa

Bu tezde, uçtan uca eğitilebilen sinir ağı temelli kayıplı görüntü sıkıştırma sistemi sunulmaktadır. Geleneksel sıkıştırma algoritmaları basit veri modellerine dayalı ve basit yapılı doğrusal dönüşüm, nicemleme ve entropi kodlama adımlarından oluşur. Sinir ağı temelli görüntü sıkıştırma yöntemlerinde ise doğrusal dönüşüm ve entropi kodlama adımları sinir ağları aracılığıyla gerçekleştirilir. Sinir ağları sayesinde, sayısal işlem karmaşıklığı artmakla birlikte, dönüşümler ve entropi kodlama için gerekli olasılık modelleri görüntüler eniyilenmiş bir şekilde işlenebilir ve karmaşık bağlantılar modellenebilir.

Sinir ağı tabanlı başlıca görüntü sıkıştırma çalışmalarında, dönüşüm ve ters-dönüşüm adımları için otokodlayıcı temelli bir ağ yapısı kullanılır. Saklı değişkenlerin (dönüşüm kat sayıların) nicemlenmesi ve aritmetik kodlanmasında geleneksel yöntemler kullanılırken, aritmetik kodlayıcı tarafından kullanılan bu değişkenlerin olasılık dağılımları ise bir sinir ağı ile temsil edilir. Sistemde bulunan sinir ağlarının parametreleri, resimlerden oluşan bir veri kümesi yardımıyla, hız-bozulma bedelini küçülterek öğrenilir.

Sinir ağı tabanlı sıkıştırma konusundaki önemli bir çalışmada, tek bir kanaldaki (özellik haritasınındaki) saklı değişkenlerin kendi içinde bağımsız olduğu kabul edilip, her bir kanal için tek bir dağılım modeli öğrenilmiştir. Daha sonra aynı yazarlar bu çalışmayı, üstönsel yardımıyla, bir kanal içerisindeki bağımlılıkları modelleyerek sıkıştırma başarımını önemli derecede artırmıştır. Bu tezde ise saklı gösterimdeki bağımlılıkları modellemek için alternatif bir yöntem kullanılmıştır. Saklı değişkenlerin dağılımı, koşullu dağılımların çarpımı olarak modellenmiştir. Ancak, her bir değişkeni önceki tüm değişkenlere koşullamak yerine, Markov özelliği varsayılarak sadece bir kaç yakın komşusuna (üst, sol ve sol-üst) koşullanmıştır. Sunulan koşullu dağılım tabanlı algoritma, üstönsel tabanlı algoritmaya göre daha basit bir sinir ağı içermesi ve daha az sayıda parametre içermesine rağmen, sıkıştırma performansı üstönsel tabanlı çalışma ile başa baş başarım elde edilirken, kodlama süresi açısından daha uzun sürmektedir.

Anahtar Kelimeler: görüntü sıkıştırma, dönüşüm kodlaması, derin öğrenme, koşullu modelleme

To Türk Eğitim Vakfı

for supporting me since high school

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to Assoc. Prof. Dr. Fatih Kamışlı for being an excellent supervisor for the past two years. Apart from guiding through the obstacles and hassles of graduate education, he treated me as a fellow during our discussions on this research and shared his experiences with me whenever I needed.

I thank Aybüke Erol for being a caring partner and an exceptional colleague to me. I cannot count how many times my problem is solved just by discussing it with her. It was comforting to know she was there to help me out. It was a remarkable experience to be a part of METU Center for Image Analysis during my thesis studies. I would like to thank every member of OGAM for the great memories. I thank Oğul Can for motivating me to go on and helping me when I feel confused. I am grateful to Yeti Ziya Gürbüz for paying attention to my problems and making suggestions on the critical decisions that I had to make. Special thanks to Ece Selin Böncü for always being helpful and cheering me up. I appreciate Dr. Alper Koz for his insightful advices on the graduate education and the academia. I would like to thank Esat Kalfaoğlu, İhsan Emre Üstün, Ayberk Aydın, İzlen Geneci and İlker Gürcan for their valuable time and friendship.

I would like to express my genuine appreciation towards my family for motivating me to make the important decisions on my own and supporting me no matter what.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

2D	2 Dimensional
3D	3 Dimensional
BPG	Better Portable Graphics
bpp	bits per pixel
CDF	Cumulative Distribution Function
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DN	Divisive Normalization
FFT	Fast Fourier Transform
GAN	Generative Adversarial Network
GDN	Generalized Divisive Normalization
IGDN	Inverse Generalized Divisive Normalization
PSNR	Peak Signal-to-Noise Ratio
PDF	Probability Density Function
RNN	Recurrent Neural Network
MONDE	Monotonic Neural Density Estimator
MSE	Mean squared error
SGD	Stochastic Gradient Descent
VAE	Variational Autoencoder

CHAPTER 1

INTRODUCTION

1.1 Image Compression Basics

Image compression or coding aims to represent an image signal using as few bits as possible for a given reconstruction quality. The compression of a signal is possible because of a few reasons. Firstly, it is due to spatial and spectral redundancy in the image signal provided by the image-capturing sensor. The adjacent samples of the image signal are highly correlated. This is because the image signal samples are obtained using a sensor which collects the data from the environment at spatially or temporally closer intervals. Light sensors on a camera will pick up temporally close light signals which are likely to be correlated if they are from the same source. Secondly, the human visual system is more sensitive to low frequency content than the high frequency content. As a result, it is possible to throw away some of the high frequency content in the image signal, and store fewer bits that are significant to the viewer.

The demand for the transmission of higher resolution image and video is increasing constantly. Increasing the communication bandwidth is not a feasible solution, an efficient transmission of the images and videos is also necessary. Throughput of the sensors are increasing with the advances in the manufacturing processes and computational capabilities of processors. Higher sensor data-rate requires better compression, faster transmission and also larger storage space.

Structure of a typical image compression scheme is shown in Fig. 1.1 and the blocks are explained further in this section. To reduce the number of bits to be transmitted/stored, the signal is generally represented in a different domain than the original

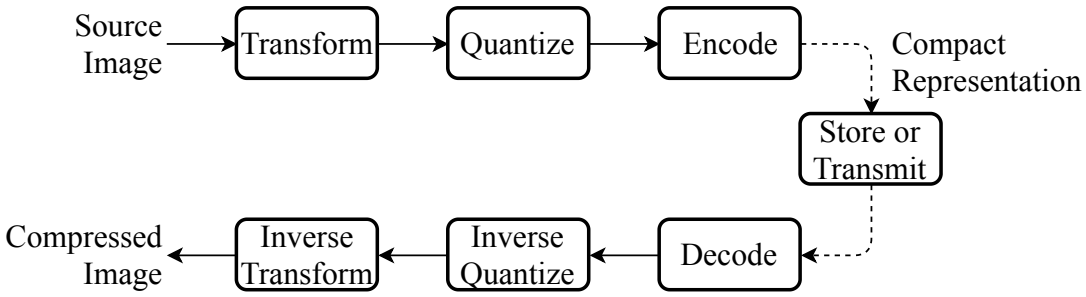


Figure 1.1: Block scheme for a generic image compression framework. A source image is transformed to represent the pixel in a transform domain with good energy compaction property. After that, the quantization takes place to reduce the representation accuracy at the cost of introducing distortion. In the end, integer coefficient values are encoded with an entropy coder using their statistical properties to represent them with as few bits as possible. The decoding process is just the inverse of the encoding process, the bits are decoded, inverse quantized and inverse transformed to reconstruct the compressed image.

domain where the signal lies [7]. By applying transformation, it is aimed to decorrelate the signal samples, hence make the implementation of the quantizer easier, i.e. use scalar quantizer [8].

The type of compression discussed so far is called lossy compression, where the signal is transmitted with negligible error or distortion to achieve low bit-rates. In lossy compression, distortion in the source signal is compensated with the reduced bit rate. The trade-off between rate and distortion gives rise to a rate-distortion optimization problem in compression, where the algorithms try to decrease the marginal cost in the distortion for lower bit-rate.

Lossy compression schemes make use of another type of redundancy that is dependent on the final receiver/observer [9]. The observer is not interested in the exact representation of the original signal. This is a common situation in multimedia applications, because the human visual system is less sensitive to high-frequency information. The most popular image/video compression algorithms use that knowledge and suppress the high-frequency information content in the image, to reduce the number of bits required to store an image. It is possible to achieve very high compression ratios even with a small distortion by quantization of the transform coefficients. The quantization step is chosen to be small for low-frequency information and large for the high-frequency information.

It is apparent that the performance of the lossy image coders are highly dependent on the type of transform and quantization strategy. There is a vast literature [7, 10, 11] on the effectiveness of linear transforms for images that discuss their rate-distortion performance. Usage of linear transforms, such as discrete cosine transform, discrete sine transform, wavelet transform, etc., for the decorrelation of the source signals, are currently the primary approach, due to their reasonable performance and relatively low complexity on hardware.

In recent years, deep learning is applied to many problems in image processing [12] such as denoising[13, 14, 15], inpainting[16, 17], super-resolution[18, 19, 20]. Image compression is also among these problems [5, 4, 6, 21, 22, 23, 24, 21], where a neural network is trained with the aim of representing an image with less number of bits under different distortion levels. The use of neural networks have a few advantages compared to the classical image compression algorithms.

1.2 Current Standards

In the classical image compression algorithms, the transformation, quantization and entropy coding steps are designed based on the simple models of the data (e.g. Gaussian-Markov model of the image pixels [11]) and low complexity algorithms (e.g. linear transforms with fast computation algorithms such as DCT [10]). Recent classical image or video compression algorithms use multiple transforms [25], probability models for entropy coding [26, 25] and prediction modes [25, 2] to model image better.

Each of the processing steps with multiple modes require iterative tweaking to adjust the processing depending on the previous steps and the used modes to obtain decent performance. This type of approach requires many iterations and an immense amount of engineering work. The advantage of using a system based on neural network is that the processing steps such as transformation and probability models for entropy coding can be performed with neural networks whose parameters are learned joint from the dataset of images, hence no iterative tweaking is required and the processing parameters are optimized from real images instead of simpler models.

1.3 Problem Definition for Neural Network based Image Compression

Training of a neural network for image compression is formulated in Eq. 1.3 as a weighted sum of rate and distortion costs [4], where rate is measured with the entropy, $H(P_{\hat{Y}})$, of the quantized latent variables/transform coefficients $\hat{\mathbf{y}}$, and the distortion is measured with the mean squared error, $\mathbb{E}\|\mathbf{x}-\hat{\mathbf{x}}\|$, between the original image, $\mathbf{x}$ and the compressed/reconstructed image $\hat{\mathbf{x}}$.

$$\min_{\phi,\theta} J = H(P_{\hat{Y}}) + \lambda \mathbb{E}\|\mathbf{x}-\hat{\mathbf{x}}\| \tag{1.1}$$

$$= H(P_{\hat{Y}}) + \lambda \mathbb{E}\|\mathbf{x}-g(f(\mathbf{x},\phi),\theta)\| \tag{1.2}$$

$$= \mathbb{E}[-\log_2 p_{\hat{Y}}(\hat{y})] + \lambda \mathbb{E}\|\mathbf{x}-g(f(\mathbf{x},\phi),\theta)\| \tag{1.3}$$

The neural network for transform/analysis is $f(\mathbf{x},\phi)$ and inverse transform/synthesis is $g(\hat{\mathbf{y}},\theta)$, where ϕ and θ represent the parameters of these neural networks.

There are some problems regarding the differentiability for the image compression using neural nets. First of all, quantization of transform coefficients is necessary for the lossy compression, but the quantization operation is not differentiable, which is explicitly required by the popular nonlinear optimization algorithms using gradients. The second problem is the quantization of probability densities that is required by entropy coders such as arithmetic coder. The last problem is the training of a single neural network to work at different rate-distortion performance, i.e. allow changing the compression amount in a wide range of possibilities. There are different methods applied as a solution to these problems. Some of the important steps in the learned image compression is summarized below.

To overcome the derivative problem of the quantization operation, Ballé et. al. [5] uses additive uniform noise as a proxy for quantization during training. The latent variables at the output of the analysis transform are mixed additively with the uniform noise to imitate the effect of quantization. During the training, continuous univariate density model for each channel of latent space is learned, assuming the independence of adjacent pixels in the latent space. The density model and the latent variables are only quantized during the testing phase.

Rippel and Bourdev [27] trained a pyramidal structured Generative Adversarial Network (GAN) to train a scale-independent network. The latent space is decomposed into bit-planes such that each bit level for a given channel is modeled jointly for the entropy coding.

Theis et. al. [21] proposes an alternative way to deal with the quantization. Instead of distorting the latent variable with the additive noise, they apply the quantization in the forward pass, but the derivative of the quantizer is set to unity for the backward pass. This way the decoder always trains on quantized variables and the quantization operation only affects the training of the analysis transform (encoder of autoencoder). They also propose the use of parameter scaling for the analysis transform after the training, to tune the rate-distortion performance of the network. Otherwise, retraining of the network is necessary for each time a new point of the rate-distortion curve is needed.

Toderici et. al. [23, 24] trains an Recurrent Neural Network (RNN) that outputs a binary sequence of bits each time the network is run. This way it is possible to adress the problem of variable rate coding using a single network. Each time the network is run for a single image, more bits are generated, which can be used to decrease the distortion. The generated bitstream is then modeled using an architecture similar to PixelRNN [28] for conditional coding.

Li et. al. [29] trains an importance map enabling the network to tune the quantization steps so that it is possible to control the bit-rate for regions in an image. The resulting network sets quantization steps smaller for the edges and corners, and larger for the smooth regions. As a result, better perceptual quality is achieved for the same bit rate. In addition to that, Trimmed Convolutional Networks are used to model the latent space coefficients and achieve higher bit rates

Mentzer et. al. [22] proposes an alternative quantization method such that the quantization centers are learned during training. The quantization is applied as it is during the forward pass and the relaxation of the quantization is performed by a smooth distance kernel to each quantization center in the backward pass.

1.4 Motivation and Contribution

This thesis focuses on the density modeling of the latent variables to achieve higher coding efficiency. While a univariate density model is trained in [5] with the assumption that the latent variables are independent, it is apparent that the latent variables are still correlated after the training. Motivated by this observation, a conditional density model, that makes use of the neighboring coefficients, is trained for the joint probability density modeling of latent variables using an architecture similar to Monotonic Neural Density Estimator (MONDE) proposed in [30].

Different from the publications that performs context modeling in the latent space [6, 22, 29], the proposed network directly learns the distributions, rather than making nonlinear predictions of the values of the coefficients. In addition to that, learning the distributions directly requires a very simple network structure, so that the network complexity is reduced compared to the similar architectures.

In summary, the contribution of this thesis is to train a conditional density model that outputs the likelihoods of the latent variables conditioned on the values of the neighborhood coefficients, such that the arithmetic coder can achieve lower bit rates by coding each coefficient under its density model.

1.5 Outline of the Thesis

Chapter 2 presents some preliminaries which contain essential information about the following chapters. Topics from information theory related to the lossless and lossy compression such as rate, entropy, distortion, coding, etc. are discussed. The trade-off between the rate and distortion and the entropy coding using arithmetic encoder is presented. The classical methods in image compression are reviewed using image codecs. In addition, a short background on deep learning including topics such as mathematical formulation, popular optimization methods and non-linearities is presented.

Chapter 3 contains a literature review on learned image compression methods. The key points of major publications in recent years on this topic are discussed, while a

short summary of their methodology is presented.

Some of the topics from the literature that are closely relevant to this thesis are discussed in detail in Chapter 4. In addition, the proposed method is formulated and compared to the univariate density model used in the literature using toy examples and visualizations. In addition to that, the details of the network architecture and the training procedure are given. The network architecture is explained in detail and supported with the block diagrams. The key differences between the training and testing phases are described. The use of conditional density model in arithmetic encoder is explained.

Chapter 5 presents compression results and discusses the practicalities of the training and testing. The used dataset during the training and testing phase, and the network's performance on these data is shown with comparisons from the literature. Visualizations of the latent space as images and histograms, the correlations between the pixels are also presented.

CHAPTER 2

BACKGROUND

2.1 Source Coding

Source coding or data compression aims to represent data with as few bits as possible. There are two types of compression methods [31]: the first one is called lossless compression, where the information can be exactly recovered using the compressed bits. The other is called lossy compression where the recovered information is allowed to have some distortion/loss with respect to the original information in order to achieve higher compression.

A signal to be compressed is represented with $\mathbf{x} \in \mathbb{R}^N$. The encoder typically consists of a transform, quantizer and entropy coder. A transform $f(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathbb{R}^M$ is used to represent the signal $\mathbf{x}$ in a different domain $\mathbb{R}^M$. Quantization, $Q(\mathbf{x}) : \mathbb{R}^N \rightarrow \mathcal{I}$, is a mapping from the signal space $\mathbb{R}^N$ to a set $\mathcal{P}$. Entropy coding, $\gamma(x) : \mathcal{P} \rightarrow \mathcal{C}$, is an invertible mapping to a sequence of bits.

Similarly, the decoder is composed of an inverse mapping, $\gamma^{-1}(x) : \mathcal{C} \rightarrow \mathcal{I}$, and an inverse quantization, $Q^{-1}(x) : \mathcal{I} \rightarrow \mathbb{R}^N$. The encoding operation can be represented as $T_e = Q \circ \gamma$ and the decoding operation can be represented as $T_d = \gamma^{-1} \circ Q^{-1}$.

2.2 Quantization

Quantization function $Q(.)$ is a mapping [32, 33] of a signal $\mathbf{x} \in \mathbb{R}^N$ to a discrete set, i.e. codebook $\mathcal{P} = \{\hat{x}_k\}_{k \in \mathcal{I}}$. The quantizer is called scalar quantizer if $N = 1$, otherwise, it is called the vector quantizer. Fig. 2.1 shows a representation for a uniform

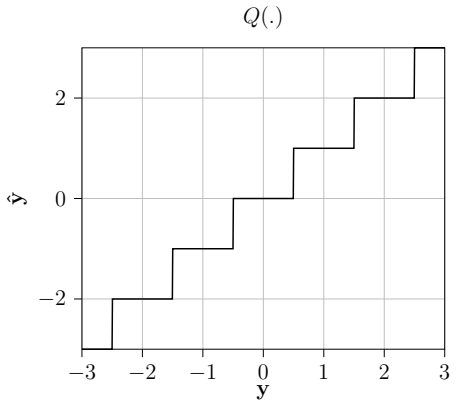


Figure 2.1: A uniform quantizer mapping the values of y to $\hat{y}$.

quantization function. The assignment to a codebook element (i.e. reconstruction level) is done by finding the decision region in which the signal resides and mapping to the reconstruction level in that region. Quantization is a lossy operation and thus not exactly recoverable. A quantizer with M distinct quantization / reconstruction value denoted by $\{\hat{x}_k\}_{k=1}^{M}$ is expressed as

$$Q(x) = \hat{x}_k \iff x \in [b_{k-1}, b_k), \tag{2.1}$$

where $\{b_k\}_{k=1}^{M-1}$ denotes the boundaries/decision regions for quantization / reconstruction with $b_0 = -\infty$ and $b_M = \infty$. A quantizer aims to minimize the distortion while satisfying other constraint, such as using limited number of codebook size. Because of that, the decision boundaries and the reconstruction values for each bin should be set carefully. Quantization noise, σ_Q^2, is typically used to quantify the amount of distortion introduced by quantization process, defined as the MSE between the source signal and the quantized values as:

$$\sigma_Q^2 = \mathbb{E}[(x - Q(x))^2] = \int_{-\infty}^{\infty} (x - Q(x))^2 f(x) dx = \sum_{k=1}^{M} \int_{b_{k-1}}^{b_k} (x - \hat{x}_k)^2 f(x) dx \tag{2.2}$$

The details of the distortion is discussed in Sec. 2.2.1.

For a given number of quantization values, the locally optimum values for boundaries and reconstruction levels that minimize Eq. 2.2 can be determined using Lloyd-Max algorithm [34, 35]. The Lloyd-Max algorithm iteratively calculates the reconstruction

levels and boundaries by the following equations:

$$\hat{x}_k = \frac{\int_{b_{k-1}}^{b_k} x f(x) dx}{\int_{b_{k-1}}^{b_k} f(x) dx} \tag{2.3}$$

$$b_k = \frac{\hat{x}_k + \hat{x}_{k+1}}{2} \tag{2.4}$$

Intuitively, the Lloyd-Max algorithm places the reconstruction levels at the probability centroids between the boundaries and places the boundaries at the middle of these reconstruction levels. Although the Lloyd-Max algorithm provides an optimal quantizer in the MSE sense for a given number of reconstruction levels, achieving the optimum rate for the same MSE distortion requires the utilization of entropy.

2.2.1 Distortion

The average distortion for a given distribution can be expressed in terms of the joint distribution of $\mathbf{x}$ and $\hat{\mathbf{x}}$ denoted as $p_{X\hat{X}}(\mathbf{x}, \hat{\mathbf{x}})$ and the distortion metric $d(.,.)$ usually selected as mean-squared error (MSE). Mathematically, the expected distortion can be defined as:

$$D = \mathbb{E}[d(\mathbf{x}, \hat{\mathbf{x}})] = \sum_{\mathbf{x} \in X, \hat{\mathbf{x}} \in \hat{X}} p_{X\hat{X}}(\mathbf{x}, \hat{\mathbf{x}}) d(\mathbf{x}, \hat{\mathbf{x}}) \tag{2.5}$$

where the summation is evaluated over the whole sample space of X and $\hat{X}$. In a typical compression scheme, the data is generally transformed into another domain and quantized and entropy coded in that domain [33, 36]. If we denote the analysis transform and quantization with $f(.)$ and synthesis transform and dequantization with $g(.)$, we can express the recovered signal as $\hat{\mathbf{x}} = g(f(\mathbf{x}))$ where the entropy coding and decoding steps are omitted since they are lossless operations and have no effect on the distortion. After representing $\hat{\mathbf{x}}$ in terms of $\mathbf{x}$, it is possible to rewrite Eq. 2.5 only in terms of $\mathbf{x}$,

$$D = \mathbb{E}[d(\mathbf{x}, g(f(\mathbf{x})))] = \sum_{\mathbf{x} \in X} p_X(\mathbf{x}) d(\mathbf{x}, g(f(\mathbf{x}))) \tag{2.6}$$

Eq. 2.6 suggests that it is possible to minimize loss for a given quantizer function by designing the analysis and synthesis transforms such that they introduce most of the error when $p_X(\mathbf{x})$ is low. Hence, it is possible to enhance the encoder and the decoder by the addition of these transforms. In classical compression approaches [8], the transforms are constrained to be linear to limit the complexity of the compressor.

In general, it is not possible to know or estimate the distribution $p_X(\mathbf{x})$ because it is a very high dimensional. Hence a statistical approximate is used to measure the distortion:

$$D = \mathbb{E}[d(\mathbf{x}, g(f(\mathbf{x})))] = \frac{1}{N}\sum_{\mathbf{x}\in\mathcal{D}} d(\mathbf{x}, g(f(\mathbf{x}))) \tag{2.7}$$

where $\mathcal{D}$ is the dataset consisting of N realizations of x.

2.3 Entropy

The concept of entropy is proposed by Shannon in [37] as a tool to measure amount of the information that a discrete signal contains and its typically measured in bits. The entropy is defined as:

$$H(X) = -\sum_i p_X(x_i)\log_2 p_X(x_i) \tag{2.8}$$

Intuitively, the entropy measures the amount of surprise such that the sources with unpredictable outputs have higher entropy while the sources with predictable outputs have smaller entropy. For example, a fair coin with outcomes heads, H, and tails, T has the entropy of 1 bit, while an unfair coin with $P(X = \mathrm{T}) = 0.99$ has the entropy of 0.08 bits. A plot showing the entropy of a binary source for varying probability is shown in Fig. 2.2. The equation used to calculate the entropy for this example is given in Eq. 2.9

$$H(X) = -P(X = \mathrm{T})\log_2 P(X = \mathrm{T}) - P(X = \mathrm{H})\log_2 P(X = \mathrm{H}) \tag{2.9}$$

Entropy can used to estimate the number of bits to send a message from a source to receiver [31]. In other words, it gives the expected number of bits to use to commu-

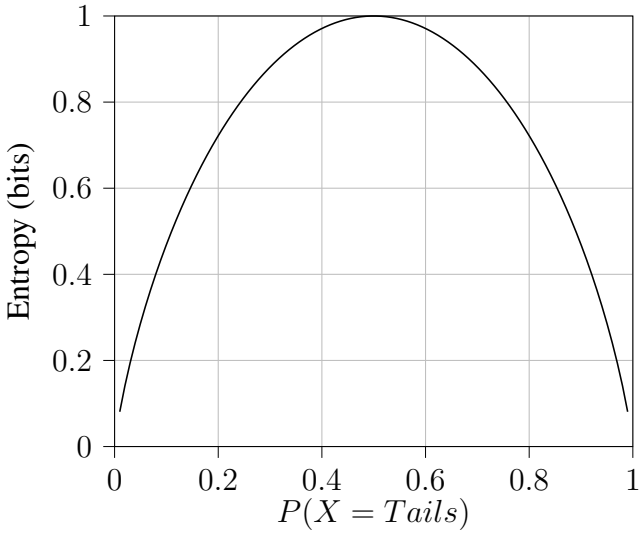
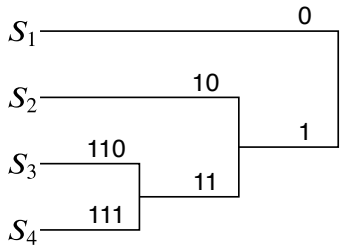


Figure 2.2: Entropy of a binary source: the entropy peaks when the outcome is less predictable.

nicate the information. Following the coin case, let us assume that we want to send the result of the coin toss to a receiver, under a lossless communication medium. For the fair coin case, it would be required to send $N * H(X) = N$ bits without loss of information. That means that each outcome of the fair coin toss requires 1 bit. For the unfair case, $P(X = \mathrm{T}) = 0.99$, expected number of bits is $N * H(X) = N * 0.08$, which means that 8 bits is sufficient to send the outcome of 100 unfair coin tosses on average. Sending 8 bits instead of 100 bits is the subject of data compression or source coding, where the most popular method called arithmetic encoding which will be the main topic of the next section.

2.4 Huffman Coding

This section presents Huffman coding [38], which is the most well known method for lossless coding of symbols with known symbol probabilities. Huffman coding is preferred in many applications due to its simplicity. To code the symbols with Huffman coding, the symbols are associated with bi- nary code-words. The most probable symbols are assigned shorter binary codes and less probable symbols are assigned longer binary codes. Codes are chosen such that they are uniquely decodable [38]. The following explains how to generate the Huffman code-words. Given a set of



Symbols	Codes	Probabilities
S_1	0	0.4
S_2	10	0.3
S_3	110	0.2
S_4	111	0.1

Figure 2.3: Huffman coding diagram

symbols with their probabilities, the two symbols with the smallest probabilities are combined (graphically shown by branches combining them) to generate a new virtual symbol with a probability equaling the sum of the probabilities of the combined symbols and a 0 is assigned to every lower branch and a 1 to every upper branch . This procedure is repeated until only one virtual symbol is left. Then the binary code-words are obtained by combining the assigned 1's and 0's from the leaf of the tree to the root of the tree. Fig. 2.3 shows an example. To code a source with the generated binary code-words, i.e. Huffman code-words, the code-word for every symbol to be coded is concatenated after each other forming a binary stream of bits.

2.5 Arithmetic Coding

In this section, an entropy coding algorithm, named arithmetic coding, to code a signal close to rate estimated by entropy is presented [33].

The arithmetic coding algorithm [39, 40] given in Alg. 1 works as follows: The probability distribution of the codewords to be coded should be defined. The probability distribution is the design parameter for an arithmetic coder, which has a very big impact on the average number of bits produced by the algorithm. The arithmetic coder uses range values to output a bit stream. The initial range is set to $[0, 1)$. To code the each codeword, the range is updated according to the probability distribution. At the end, any value between the final range values can be transmitted. The procedure of

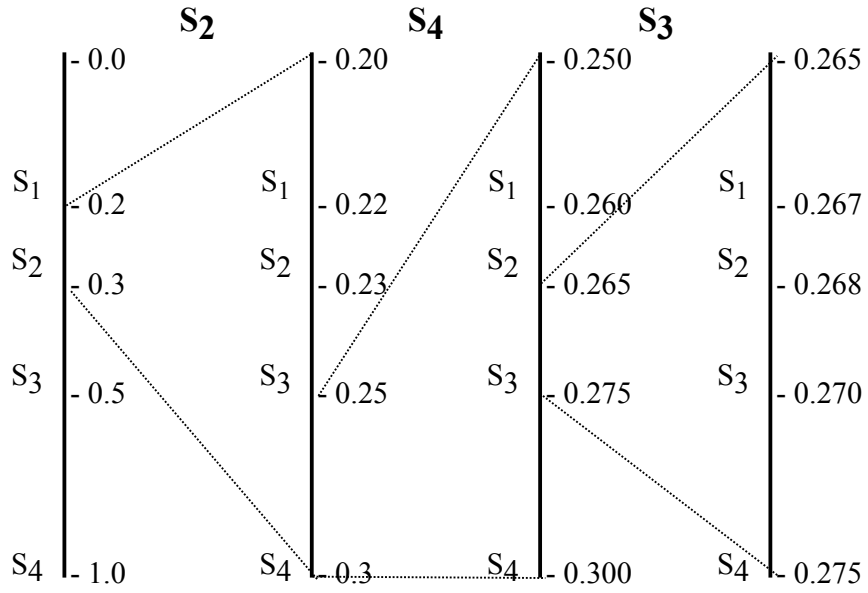


Figure 2.4: Arithmetic coding diagram for 4-symbol $\{S_1, S_2, S_3, S_4\}$ source. The low and high ranges are set to $[0.0, 1.0)$ at the first step and the sequence to be coded is S_2, S_4, S_3. In the first step, S_2 is coded and the ranges become $[0.2, 0.3)$. New steps are scaled into these ranges and S_4 is coded. At the last step, S_3 is coded and the ranges are finally between $[0.265, 0.275)$. At this point, if the coding stops, usually the lower range value (0.265) is sent to the decoder, however, any value in this range can be used and they will be decoded the same. If the coding continues, the procedure can be resumed as it is.

updating the range values is depicted in Fig. 2.4.

One known version of arithmetic coding is the context-adaptive binary arithmetic coding [41] (CABAC), which is used in H.264 and HEVC video coding standards. CABAC converts symbols first to binary code-words (hence, binary arithmetic coding) and then applies arithmetic coding on the bits in the codeword. CABAC also updates the probabilities of each bit of the binary code-words during encoding and decoding (hence, context-adaptive) to achieve lower rate.

2.6 Rate-Distortion Theory

Rate-distortion theory provides a mathematical basis for lossy compression. It is possible to analyze the performance of the source coder or compression scheme using rate-distortion theorem initially proposed by Shannon in [37]. Bit-rate of the source

Algorithm 1 Arithmetic coding algorithm

function ENCODE(data, $\texttt{cdf}[]$):
    $[L, H] = [0, 1]$
    for each $s \in$ data **do**
        if s == EOF **then**
            break
        $[L, H] = L + (H - L)[\texttt{cdf}[s-1], \texttt{cdf}[s]]$
    return L

function DECODE(V, $\texttt{cdf}[]$, N): ▷ V: bitstream, N: number of codewords
    $[L, H] = [0, 1]$
    data = []
    for 1 **to** N **do**
        find i such that $\texttt{cdf}[i-1] \leq \frac{V-L}{H-L} < \texttt{cdf}[i]$
        data.push(i)
        $[L, H] = L + (H - L)[\texttt{cdf}[i-1], \texttt{cdf}[i]]$
    return $data$

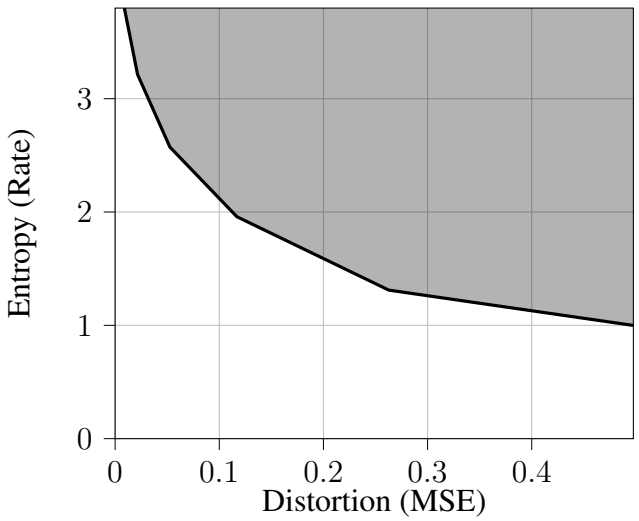


Figure 2.5: Rate-Distortion plot for the different number of quantization levels is plotted by applying the Lloyd-Max algorithm on Laplacian distribution. As the rate gets higher, the distortion is reduced. However, the amount of decrease is getting less and less as the rate is increased.

could be tuned using variable-sized quantization bins, however, it comes with the cost of distortion in the transmitted data inversely proportional to the bit-rate. This trade-off is visualized in Fig. 2.5, for a Laplacian source with scale parameter equals to one. The data is sampled from the laplacian distribution and quantized using the Lloyd-Max quantizer to yield minimum distortion for a given codebook size.

As shown in the graph, using higher bit rates has diminishing returns on the distortion. Optimal lossy encoding algorithms in general can be seen as a system operating at a point on the rate-distortion curve. The location of the point depends on the performance of the quantizer, which is discussed in Sec. 2.2. Sub-optimal quantizers may yield points in the shaded region called the feasible region. Any point on the distortion curve is optimal, hence a Lagrangian term is used to move on this curve to set the trade-off between the rate and the distortion. The curve is defined by $D + \lambda R$ where D represents the distortion and R represents the rate. The rate-distortion theory is the information-theoretic way to analyze lossy compression.

2.7 JPEG

JPEG compression is one of the most popular compression methods, utilizing YCbCr color space transformation, 8×8 block Discrete Cosine Transform (DCT), quantization, run-length and Huffman coding.

2.7.1 YCbCr Color Conversion and Chroma Subsampling

The image in classical compression systems are often converted into YCbCr color space so that the intensity levels and color information (chroma) can be processed independently. In YCbCr color space, Y channel holds the intensity values for the images while Cb and Cr channels carry the color information. YCbCr color space was initially developed for the color TV [7], to use the bandwidth effectively for the transmission of color signal and make the new colored video broadcast compatible with the old black and white TV receivers. That was possible because the chromatic components (color information) can be transmitted in lower resolution, with unnoticeable visual distortion. The reason for that is the sensitivity of the human eye to the light intensity rather than the color information. YCbCr color format carried its popularity in the digital systems as well, due to the same reason. In compression systems, the chromatic components are often vertically and horizontally decimated by 2. Such image formats are marked as YCbCr420. Other alternatives are only horizontal subsampling, YCbCr422 or no subsampling at all, YCbCr444.

Figure 2.6: YCbCr420 representation of an image from Kodak Image Dataset. The first image is in the RGB domain. The second image is the intensity channel (Y) of the YCbCr representation. The top image in the third column is the Cb channel and finally, the bottom image is the Cr channel. In 420 image format, the chroma components of the image are downsampled by 2 in both direction, assuming that they do not have high-frequency components. If observed, one can see that the doorknob and the lock is not visible in the Cb channel, and it is barely visible in the Cr channel.

$$\begin{bmatrix} \mathrm{Y} \\ \mathrm{Cb} \\ \mathrm{Cr} \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} \mathrm{R} \\ \mathrm{G} \\ \mathrm{B} \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \tag{2.10}$$

Eq. 2.10 shows the transformation matrix from RGB color space to YCbCr color space. While the intensity channel (Y) is the weighted sum of the R,G,B channels, the chroma channels (Cb, Cr) are zero mean before the addition of a bias (128). The effect of the transformation is given in Fig. 2.6. The intensity channel contains most of information about the image and chroma channels contain slowly changing/low bandwidth color information.

2.7.2 Discrete Cosine Transform

The natural images contain most of their energy in the low-frequency component. This empirical observation leads to usage of Discrete Fourier Transform (DFT) or its variants in the compression of images. Although, DFT provides a good transformation for the image compression, the blocking artifacts due to quantization are not desirable [7]. An alternative transformation is the Discrete Cosine Transform (DCT) [10] which is often used in lossy image and video compression algorithms such as JPEG[1], MPEG[42], H264[26], H265[25] due to its energy compaction property and low blocking artifacts. Unlike DFT, DCT contains only real coefficients. DCT of

a signal is the DFT of a signal with mirrored padding, so that the DFT input becomes even and the results are real. Due to this mirroring, abrupt changes in high-frequency terms of consecutive blocks are prevented. The transform of a signal $x[n]$ of length N is formulated as:

$$X[k] = \sum_{n=0}^{N-1} x[n] cos\left[\frac{k\pi}{N}\left(n + \frac{1}{2}\right)\right] \tag{2.11}$$

For image processing applications, 2D-DCT formulation is a simple extention of Eq. 2.11, assuming that 2D image signal $x[n_1, n_2]$ of support $N_1 \times N_2$ is given by:

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] cos\left[\frac{k_1\pi}{N_1}\left(n_1 + \frac{1}{2}\right)\right] cos\left[\frac{k_2\pi}{N_2}\left(n_2 + \frac{1}{2}\right)\right] \tag{2.12}$$

Fig. 2.7 shows the DCT basis vectors for an 8×8 image, meaning that an image patch could be fully represented by scaled combinations of these patches. JPEG performs compression by dividing the image into 8×8 sub-blocks and applying DCT, quantization, and coding respectively.

2.8 Better Portable Graphics

Better Portable Graphics (BPG) is a wrapper around High Efficiency Video Coding (HEVC) providing much better compression efficiency compared to JPEG. HEVC [25], also known as H265, is the latest video coding standard by ITU-T Video Coding Experts Group, providing up to 50% better compression rate at the same quality compared to its earlier version H264 [26]. BPG uses the intra-coding [2] of HEVC, which is mainly designed to compress a video frame using only the information content in the current frame of interest. Hence, the performance of BPG is directly linked to the HEVC.

Intra coding consists of quadtree-based coding of a frame, using angular and planar predictions and transform coding. Neighboring pixels in a coding block of interest are used for the predictions. The quadtree structure consists of blocks having varying sizes: $4\times4, 8\times8, \ldots 64\times64$ as shown in Fig. 2.9. Left and top neighbor pixels of the block are used to make an angular prediction out of 33 possible angular directions and 2 planar prediction methods, called modes, as shown in Fig. 2.8. The choice of intra-

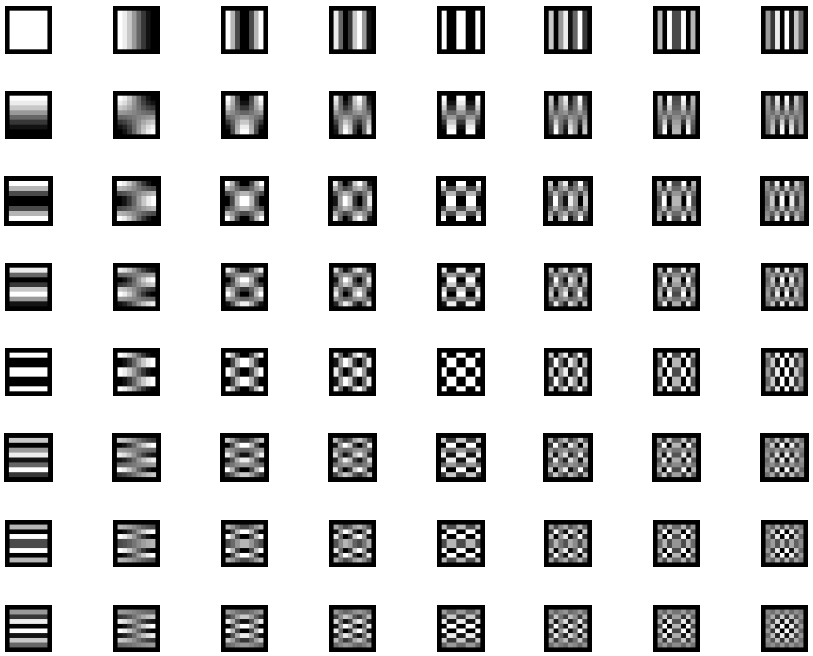


Figure 2.7: 8×8 DCT base images used by JPEG is shown in spatial domain. Top left corner represents the lowest frequency (DC) and bottom right represents the highest frequency in the discrete domain. As moved from top left to bottom right, the checkerboard-like patterns start to repeat more frequently. JPEG [1] uses the assumption that the human eye is most sensitive to the low-frequency content.

prediction modes and the block sizes are determined by rate-distortion optimization during the encoding. Since this procedure is repeated for many times for each block in a frame, it becomes very costly to do intra-coding for large frame sizes. As a result, it is an area of research to reduce the complexity of the intra-prediction algorithm. Following the prediction, transform coding of residuals using DCT and post-processing to suppress blocking artifacts at the edges of coding blocks is performed. The transformed and quantized residuals are encoded using CABAC algorithm.

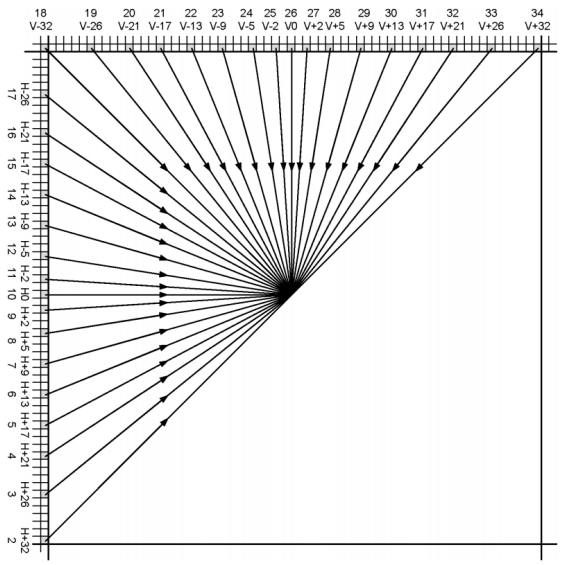


Figure 2.8: 33 angualar intra prediction directions of HEVC (redrawn from [2])

2.9 Feed Forward Neural Networks

A fully connected neural network could be expressed as series of chained nonlinear functions:

$$T = f_K \circ f_{K-1} \dots f_1 \tag{2.13}$$

The domain and range of the functions are $f_k : \mathbb{R}^{N_k} \rightarrow \mathbb{R}^{N_{k+1}}$, $k \in \{1 \dots K\}$. Each function denotes a layer in the network, e.g. f_1 is the first layer, f_2 is the second layer and so on. Also, f_1 denotes the input layer, f_K denotes the output layer, and $f_i, i \in \{2, \dots, K-1\}$ are the hidden layers of the network. A single function can be expressed as a linear matrix multiplication, followed by a non-linear function called activation:

$$f_k(\mathbf{x}_k) = g_k(\mathbf{W}_k \mathbf{x}_k + \mathbf{b}_k) \tag{2.14}$$

where $\mathbf{W}_k$ and $\mathbf{b}_k$ is the weight matrix and the bias vector to be learned, $g_k(.)$ is the nonlinear activation function, $\mathbf{x}_k$ is the input vector at l-th layer. Fig. 2.10 shows a sample representation of Eq. 2.14

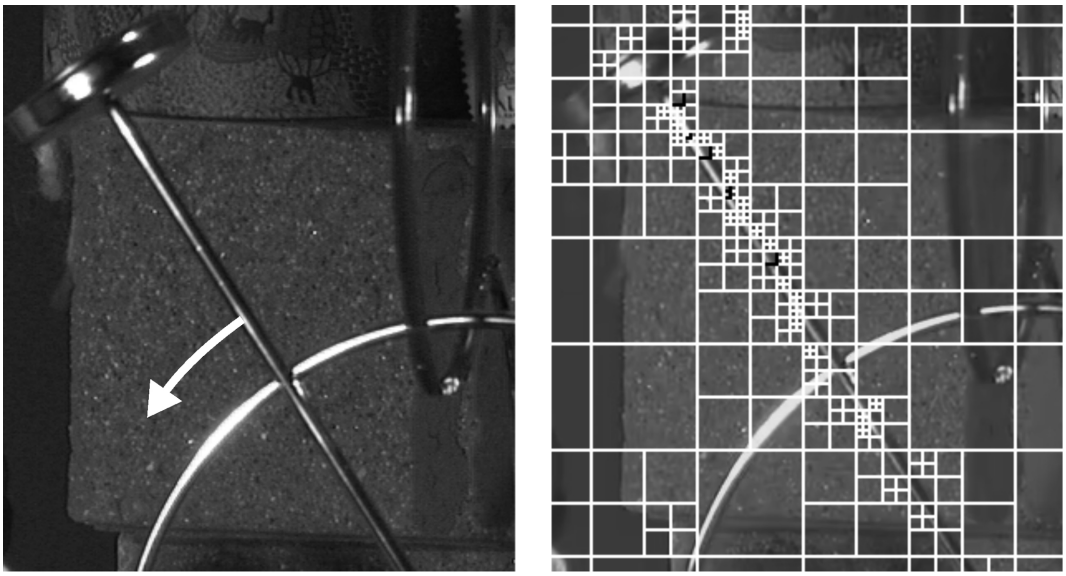


Figure 2.9: An example of quadtree partitioning (redrawn from [3]).

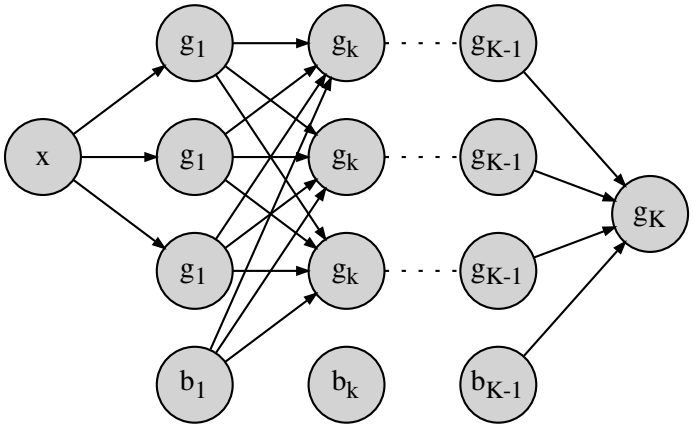


Figure 2.10: A fully connected neural network with $K-1$ hidden layers and 3 nodes in each layer with a bias term. Arrows represent the weights, nodes labelled with g_ik represent the nonlinearities at each layer and nodes labelled with b_i represent the bias terms.

2.10 Back-propagation Algorithm

Determining the weights $\mathbf{W}_k$ of the neural network requires an optimization algorithm to produce the expected results. This procedure of optimization of the weights is called learning, which is achieved by making use of an algorithm called back-propagation.

The back-propagation algorithm [43] is an efficient way to calculate the derivatives at each layer. The gradients calculated by the back-propagation algorithm is used by an optimization algorithm, such as the stochastic gradient descent, to optimize the network's weights so that the network produces expected results. The back-propagation algorithm calculates the gradients for a given function $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y})$. Most optimization algorithms try to optimize a scalar cost function $J(\theta)$, hence its gradient $\nabla_\theta J(\theta)$ is required.

The main idea behind the back-propagation algorithm is the chain rule of calculus. The chain rule of calculus can be explained by a simple example. Suppose two functions $f(x) = y$ and $g(x) = y$ are combined, such that $z = f \circ g(x)$, then the derivative of the combined function is equal to:

$$(f \circ g)'(x) = (f' \circ g)(x) g'(x) \tag{2.15}$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dz} \tag{2.16}$$

For high dimensional functions $f(\mathbf{x}) = \mathbf{y}$ and $g(\mathbf{x}) = \mathbf{y}$, the chain rule can be generalized as:

$$\nabla_{\mathbf{x}} \mathbf{z} = \frac{\partial \mathbf{z}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial \mathbf{x}} \tag{2.17}$$

where $\frac{\partial \mathbf{z}}{\partial \mathbf{y}}$ and $\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$ are Jacobian matrices. Similarly, a network of L layers consists of L Jacobian matrices and the back-propagation algorithm computes them starting from the output. For high dimensional tensor functions, a similar approach for vectors is used [44, p. 203], where the tensors are rearranged to flatten into a vector form to calculate the gradients, then reshaped back to the original tensor form. The importance of the properties of the Jacobian matrix for this work is discussed in Sec. 4.3.

2.11 Optimization of Neural Networks

The optimization of machine learning models is different from the classical optimization algorithms [44, Ch. 8]. In the latter case, the function to be optimized $J(\theta)$ is the goal itself. However, the optimization algorithms of machine learning aims to optimize a performance metric, P, which cannot be optimized directly, hence a cost function $J(\theta)$ is optimized using a dataset, so that P could be optimized indirectly. For example, the final metric for this thesis is the rate-distortion cost which is not differentiable, hence a differentiable cost function is optimized instead.

$$J(\theta) = \mathbb{E}\left[L(f(x;\theta), y)\right] \tag{2.18}$$

The expectation is approximated emprically using a dataset

$$\mathbb{E}\left[L(f(x;\theta), y)\right] = \frac{1}{N}\sum_{i=1}^{N} L(f(x_i;\theta), y_i) \tag{2.19}$$

where N is the size of the dataset. Another difference is the evaluation of the loss. As the machine learning algorithms become computationally demanding, evaluation of the loss over the entire dataset becomes unfeasible. Hence, most of the optimization algorithms work on a loss function approximated by a randomly sampled subset of the training data called batches. Since the gradient computations become much faster, compared to the gradient computation over the whole dataset, the convergence time is highly reduced, even though more updates are required.

This work uses the Adam algorithm for the optimization of the parameters. Although there are many variants of the optimization algorithms in deep learning, only Stochastic Gradient Descent, RMSProp, and Adam algorithms will be discussed in detail. A thorough review of the optimization algorithms is presented in [45].

2.11.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is a popular optimization algorithm in deep learning. Similar to the gradient descent algorithms used in classical optimization prob-

lems, SGD tries to reach the minima using approximate gradients using batches that are randomly sampled from the dataset. The algorithm is summarized in Alg. 2

The main difference from the classical descent algorithms is the approximation of the gradients using a small subset of the dataset called batches, instead of using the whole dataset per gradient calculation. Using smaller dataset has several benefits. First of all, the neural network training requires a lot of memory, while consumer-level GPUs can provide somewhat limited memory, usage of small dataset makes training easy on these memory limited GPUs. Secondly, smaller data size requires fewer computations, decreasing the time required per weight update during the training of the neural networks. Thirdly, research shows that large batch sizes causes the optimization algorithm to reach a bad local minima [44], due to the high nonlinearity of the deep learning problems.

While providing a basis for advanced optimization algorithms in deep learning, SGD suffers from vanishing gradients towards the end of the training, because the loss is decreasing and gradients are getting proportionally smaller. To prevent vanishing of the gradients near the local minima, the learning rate parameter should be adjusted so that the step size is large enough to continue the optimization.

Algorithm 2 Stochastic Gradient Descent Algorithm

1: θ, Parameter Set
2: ϵ, Learning Rate
3: $\mathcal{D}$, Dataset
4: m, Batch size
5: **while** stop criterion not met **do**
6: $\quad \{x_i, y_i\} = \text{sampleBatchPair}(\mathcal{D})$
7: $\quad d = \frac{1}{m} \nabla_\theta \sum_i L(f(x_i; \theta), y_i)$ ▷ Compute gradient
8: $\quad \theta = \theta - \epsilon d$ ▷ Update weights

2.11.2 RMSProp

RMSProp [46], introduced informally on a lecture note is an algotithm that adapts the learning rate using the history of the gradients, so that it is possible to reach a good minima while optimizing the weights. The values of gradients are weighted exponentially, hence the terms in the past affect the current learning rate less. As shown in Alg. 3, the main difference from the SGD is line 11, which is used to scale

the learning rate. This approach can be viewed as IIR filtering of the gradients, with a single-pole filter.

Algorithm 3 RMSProp Algorithm

θ, Parameter Set
ϵ, Learning Rate
$\mathcal{D}$, Dataset
m, Batch size
ρ, Decay rate
δ, Small constant used for numerical stability
$r = 0$, Second moment
while stop criterion not met **do**
    $\{x_i, y_i\} = \text{sampleBatchPairs}(\mathcal{D})$
    $d = \frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$ ▷ Compute gradient
    $r = \rho r + (1-\rho) d \odot d$ ▷ Update second moment
    $\theta = \theta - \epsilon \frac{1}{\sqrt{r}+\delta} \odot d$ ▷ Update weights

2.11.3 Adam

The Adam algorithm [47] improves the RMSProp algorithm by addition of the momentum, and proposes a solution to correct bias introduced by the exponential weighting. Currently, Adam is one of the most popular optmization algorithms, providing fast convergence and robustness to hyperparameters. The Adam algorithm is summarized in Alg. 4

2.12 Convolution

The use of convolution operation in neural networks is motivated by the local correlations in natural images. It is possible to optimize much fewer parameters using convolutional layers and parameter sharing is possible among the image data. Convolution of two functions is defined as:

$$f(t) * g(t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau \tag{2.20}$$

Algorithm 4 Adam

θ, Parameter Set
ϵ, Learning Rate
$\mathcal{D}$, Dataset
m, Batch size
ρ_1, ρ_2, Decay rates
δ, Small constant used for numerical stability
$s = 0, r = 0$, First and second moments
$t = 0$, Step
while stop criterion not met **do**
    $\{x_i, y_i\} = \text{sampleBatchPairs}(\mathcal{D})$
    $d = \frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$ ▷ Compute gradient
    $s = \frac{\rho_1 s + (1-\rho_1)d}{1-\rho_1^t}$ ▷ Update first moment
    $r = \frac{\rho_2 r + (1-\rho_2)d \odot d}{1-\rho_2^t}$ ▷ Update second moment
    $\theta = \theta - \epsilon \frac{s}{\sqrt{r}+\delta}$ ▷ Update weights
    $t = t + 1$

Since image data is discrete and 2-dimensional (2D), 2D discrete convolutions are utilized in image processing defined as follows:

$$f[m,n] * g[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] * g[m-k, n-l] \tag{2.21}$$

There are some differences between the convolution operation in the image processing area and deep learning area. First of all, since the convolution kernels are learned rather than engineered, the kernels are not flipped before the convolution operation, and this would only affect the position of the learned weights. Hence, the right terminology would be correlation:

$$f[m,n] * g[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] * g[k-m, l-n] \tag{2.22}$$

Secondly, amount of padding is usually optional and the output size of the convolution depends on the padding of the input image. In addition to that, the images or latent representations have a third dimension, which the convolution kernel also operates on to yield a single-dimensional representation. Also, there are usually more than one

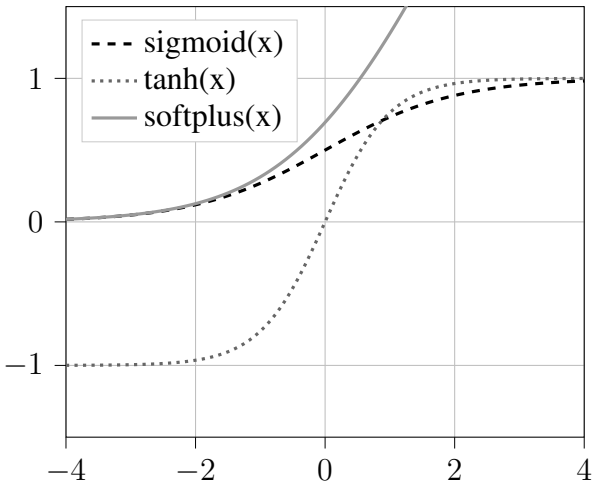


Figure 2.11: Three activation functions used in this work.

filter. Hence, the commonly used convolution equation becomes:

$$f[m,n] * g_j[m,n] = \sum_{i=1}^{N} \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l,i] * g_j[k-m, l-n, i] \tag{2.23}$$

In Eq. 2.23, $f[k,l,i]$ denotes the input data and i is the index at the third dimension, $g_j[k,l,i]$ denotes the j-th kernel.

2.13 Activation Functions

Activation functions are the source of nonlinearity in neural networks. The structure of the layers, weight initialization is dependent on the selected nonlinearity. Important non-linearities for this work, namely sigmoid, softplus, tanh, is drawn in Fig. 2.11. The equations for the activation functions and the derivatives are as follows

$$\text{sigmoid}(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.24}$$

$$\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x)) \tag{2.25}$$

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.26}$$

$$\frac{d}{dx}\tanh(x) = (1 - \tanh^{2}(x)) \tag{2.27}$$

$$\text{softplus}(x) = \sigma^{+}(x) = \ln(1 + e^{-1}) \tag{2.28}$$

$$\frac{d}{dx}\sigma^{+}(x) = \sigma(x) \tag{2.29}$$

The use of these nonlinearities and importance of their derivatives are discussed in Chapter 4.

CHAPTER 3

RELATED WORK

3.1 Overview

This chapter presents core ideas published in the learned image compression literature. The chapter starts with the generalized idea of the nonlinear transform coding [4] and how it compares to the classical transform coding approach. A general formulation for optimizing a nonlinear coding framework is presented. After that, the key publication [5] which is also the baseline for this work is presented in details. The proposed non-linearity functions to use in the analysis and synthesis transforms, solving the quantization problem for both latent variables and the probability distributions and finally the training of the end-to-end learned model is explained in detail. Following this, advances in the image compression methodology and different approaches from the key publications are presented with their core ideas and formulations while giving a short background where necessary.

3.2 Nonlinear Transform Coding

The linear transforms are preferred for their simplicity in the analysis and design of compression systems. For a similar reason, the modular design approach for the design of transformation, quantization and entropy coding steps are preferred. However, this kind of approach hinders the performance of compression systems, because the design of the modules are based on simple models of the data (e.g. Gauss-Markov model for image pixels to derive transforms [11] or spatial prediction methods [48, 49]) and low-complexity algorithms, hence their joint performance is sub-

optimal. To overcome these issues, general nonlinear transform coding framework is proposed [4].

In the nonlinear transform coding framework, the whole compression system is represented in three signal domains as shown in Fig. 3.1. The original signal is represented as x, and its domain is called signal domain. The other two domains are the perceptual and code domains.

First domain is the code domain, where the signal is transformed using a nonlinear differentiable function, $\mathbf{y} = f(\mathbf{x}; \phi)$, and ϕ is the parameter vector to be optimized. Scalar quantization is applied to the transformed signal $\hat{\mathbf{y}} = Q(\mathbf{y})$, and the rate is measured using the entropy of the resulting integer stream. To measure the distortion in the perceptual domain, signal is nonlinearly transformed back to signal domain, $\hat{\mathbf{x}} = g(\hat{\mathbf{y}}, \theta)$. Again, θ is the parameter vector of the synthesis transform to be optimized.

It is known that measuring PSNR or MSE of the raw signal does not reflect the human visual system well [50]. Hence, a general perceptual domain approach is proposed, that the signal is first transformed into another domain using a transform h, where measuring the distortion reflects the target perceptual quality better. One commonly used perceptual domain metric is the SSIM [51] or MS-SSIM metric [52] The distortion is measured in the perceptual domain by making use of the priors about the observer.

In this framework, the analysis and the synthesis transforms should be optimized to minimize the rate distortion function:

$$\min_{\phi,\theta} J = H(p_Y) + \lambda \mathbb{E}\|\mathbf{z} - \hat{\mathbf{z}}\| \tag{3.1}$$

where $\mathbf{z} = h(\mathbf{x})$ is the original and $\hat{\mathbf{z}} = h(\hat{\mathbf{x}})$ is the reconstructed signal in the perceptual domain. In this equation, the second term measures the total perceptual distortion while the first term measures the total rate to store the signal. Noting that both of the terms require explicit probability distribution for the source signal which is generally not available, they are estimated using the ensemble averages.

There is one thing to note in Eq. 3.1 and Fig. 3.1 that the quantization operation is

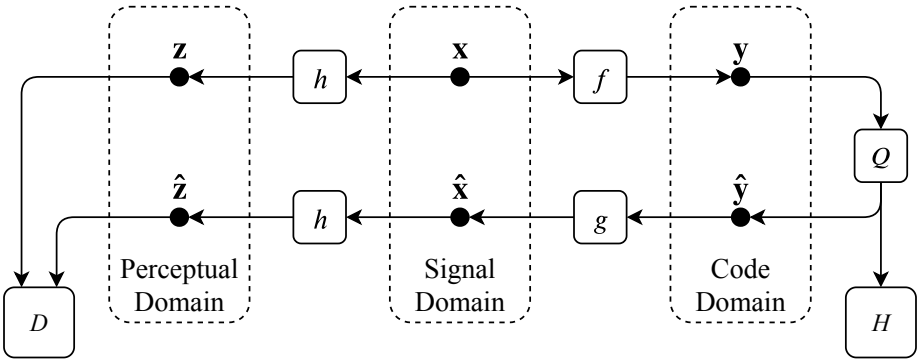


Figure 3.1: Block diagram for nonlinear transform coding (redrawn from [4])

not differentiable, while most optimization methods require differentiable functions. A relaxation of quantization function is utilized [4], to overcome this challenge.

As an improvement to [4], Balle et. al. [5] proposed to optimize a convolutional autoencoder for the rate-distortion cost. The analysis transform, $\mathbf{y} = f(\mathbf{x}; \phi)$, and synthesis transform, $\hat{\mathbf{x}} = g(\hat{\mathbf{y}}, \theta)$, are the encoder and decoder parts of the convolutional autoencoder. The encoder network generates the latent variables to be coded by the arithmetic encoder and the decoder network recovers the image from the latent variables. During the joint optimization of this network, the decoder learns to minimize the distortion while the encoder learns to minimize the rate.

There are a few challenges to overcome to train a compression network. First one is the quantization operation, where its derivative is zero almost everywhere. That is very problematic since currently the state-of-the-art for the training of the neural networks is performed by back-propagation algorithm used in variants of gradient descent algorithms, which requires differentiable functions to optimize. There are different approaches to the quantization problem. Balle et. al. [5] uses an approach similar to the one used in the quantization literature, where the quantization is modeled as additive noise. During training, uniform noise is added to the output of the encoder network, and during testing, rounding is performed. This way the decoder network learns to recover the image under quantization noise during training, as a result, it will be robust to the rounding operation during the testing phase.

Second challenge is the calculation of the rate. Although calculation of distortion between the input and recovered images to measure distortion, calculation of the rate

is rather difficult since the discrete probability distribution of the latent variables is required for the rate estimation. The representation of the probability distributions of the latent variables should be differentiable, and possibly be implemented with a neural network, so that it is easily integrated into the neural network based non-linear transform coding system and allows learning its parameters and the estimation of the rate term in Eq. 3.1 with the standard optimization methods.

3.3 End-to-end optimized image compression

The work of Ballé et. al. [5] is explained in detail in the following subsections and also in Sec. 4.3, because their work on has a central role for this thesis. The use of the univariate density model in [5] is replaced with the conditional density model in this thesis to improve the rate-distortion performance.

The nonlinear compression scheme is divided into two parts, namely the analysis and synthesis transforms which are a series of linear convolutions followed by nonlinear activation functions. The analysis transform is explained in Sec. 3.3.3 and the synthesis transform is explained in Sec. 3.3.4. Nonlinear functions are choosen as GDN [53], which is better at Gaussianizing the data [5]. The details of GDN is explained in Sec. 3.3.1 and 3.3.2.

3.3.1 Divisive Normalization

Image compression systems aim to minimize rate and distortion and it has been shown that [54] the problem is much more tractable in a transform domain. The first reason is that the elements of the representations in the transformed domain tend to be less correlated. The second reason is that it is easier to control the source of distortion in the transform domain. The transforms were preferred to be linear because of their low complexity and tractability. However, it is known that linear transforms are insufficient to achieve the aforementioned goals since the images or image blocks are not only linear combinations of independent patches but they are mostly composed of different objects in the scene.

To overcome these limitations of linear transforms, several nonlinear transforms are proposed. Inspired by the gain control and response normalization of neurons [55, 56] in the visual cortex, a nonlinear transform called Divisive Normalization (DN) [57] is proposed.

$$y_i = \frac{\text{sgn}(z_i)|z_i|^\gamma}{\beta_i + \sum_j h_{ij}|z_i|^\gamma} \tag{3.2}$$

$$\mathbf{z} = \mathbf{H}\mathbf{x} \tag{3.3}$$

In Eq. 3.3, the image, represented with input vector $\mathbf{x}$, is transformed using a linear transform $\mathbf{H}$, then in Eq. 3.2 each element z_i of the transformed vector $\mathbf{z}$ is independently normalized using the neighbouring transform coefficients to introduce non-linearity. Hence, the divisive normalization weights the energy of the neighbor coefficients in the transform domain, such that each coefficient z_i is rectified and exponentiated. The resulting coefficients are divided by the weighted sum of the neighbor coefficients to normalize the transform. Unlike the current trend in the optimization of parameters, authors of the DN proposed to use block-DCT for the linear transform for $\mathbf{H}$ and a Gaussian kernel for the weights h_{ij} in Eq. 3.2:

$$h_{ij} = \text{exp}(-\frac{(f_i - f_j)^2}{\sigma_{f_i}^2}) \tag{3.4}$$

$$\sigma_{f_i}^2 = \frac{1}{6}|f_i| + 0.05 \tag{3.5}$$

where f_i and f_j are 2-D frequency vectors of the j-th and i-th basis functions. Such a choice of transforms enhances the compression performance of JPEG [1] by more than 50% when the distortion metric is chosen as PSNR.

3.3.2 Generalized Divisive Normalization

A generalized version of DN, named Generalized Divisive Normalization(GDN) [53], is shown to perform well at modeling local probability models for natural images. In

GDN, parameters of the DN are left for optimization such that Eq. 3.2 becomes:

$$y_i = \frac{z_i}{\sqrt{\beta_i + \sum_j \gamma_{ij} z_j^2}} \tag{3.6}$$

$$\mathbf{z} = \mathbf{H}\mathbf{x} \tag{3.7}$$

In Eq. 3.6 the parameter set $\{\beta, \gamma, \mathbf{H}\}$ is to be optimized, hence it is possible achieve better independence using a very nonlinear transform.

It should be noted that there is an important difference between a popular normalization method, Batch Normalization (BN) [58], for the training of deep networks and GDN. In BN, every dimension, i, of an input vector x is independently normalized, scaled and shifted to prevent internal covariance shift such that:

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \mathbb{E}[\mathbf{x}_i]}{\sqrt{\text{Var}[\mathbf{x}_i]}} \tag{3.8}$$

$$\mathbf{y}_i = \gamma_i \hat{\mathbf{x}}_i + \beta_i \tag{3.9}$$

First and second order statistics, $\mu_i = \mathbb{E}[\mathbf{x}_i]$, $\sigma_i^2 = \text{Var}[\mathbf{x}_i]$ and the parameter set $\{\gamma, \beta\}$ is learned while training. As seen in Eq. 3.8, the data in each dimension is separately normalized with the ensemble mean and variance learned over the dataset. Although, there is a shifting and scaling operation in Eq. 3.9, it is not adaptive to the data and learned over the dataset, as a result, they are fixed for the inference. On the other hand, the normalization factor in the denominator in Eq. 3.6 is dependent on the input data as well, hence it is said to be signal adaptive even after the training phase.

3.3.3 Analysis Transform

In the analysis transform, each channel j of input image $u_j^l[m, n]$ at layer l is convolved with a series of 2D-filters $h_{ij}^l[m, n]$ and a bias term b_i^l is added to produce an output image $v_i^l[m, n]$ at channel i.

$$v_i^l[m, n] = \sum_j h_{ij}^l[m, n] * u_j^l[m, n] + b_i^l \tag{3.10}$$

According to this notation a color image in the first layer can be expressed as $\mathbf{x} = u_c^0[m,n]$ where $c \in \{1,2,3\}$. Each convolutional layer is followed by a downsampling operation, a common operation used in deep learning community [59], to reduce the number of computations. Downsampling of $v_i^l[m,n]$ by a downsampling factor of s^l can be expressed as:

$$w_i^l[m,n] = v_i^l[s^l m, s^l n] \tag{3.11}$$

It is a common procedure in signal processing to combine convolution and decimation operations to avoid the computational burden. The two operations are combined in this work as well and the convolution expression in Eq. 2.21 can be rewritten as:

$$f[m,n] * g[m,n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k,l] * g[Mm-k, Mn-l] \tag{3.12}$$

where M is the downsampling factor to be applied. This operation is commonly used in generative models in deep learning to reduce the network complexity and make training faster and also interpreted as learning downsampling filters or nonlinear downsampling [60]. The power of the neural networks comes from their nonlinearity, hence the convolutional layers are followed by a nonlinear layer proposed by Balle et. al. [53].

$$u_i^{l+1}[m,n] = \frac{w_i^l[m,n]}{\sqrt{\beta_i^l + \sum_j \gamma_{ij}^l (w_j^l[m,n])^2}} \tag{3.13}$$

The parameter set to be optimized for the analysis transform is $\phi = \{h_{ij}^l, b_i^l, \beta_i^l, \gamma_{ij}^l\}$, which are kernel parameters and biases and scaling terms of GDN. The analysis transform will be denoted as $\mathbf{y} = f(\mathbf{x}, \phi)$, where $\mathbf{x}$ is the input image and $\mathbf{y}$ is the coefficients in the transform domain.

3.3.4 Synthesis Transform

The synthesis transform follows the inverse structure of the analysis transform. The downsampling operation is replaced with the upsampling operation, where the input to the synthesis transform is $\hat{u}_i^l[m/\hat{s}^l]$ at layer l and upscaling factor $\hat{s}^l$. Hence the

upsampling can be expressed as:

$$\hat{w}_i^l[m,n] = \begin{cases} \hat{u}_i^l\left[\frac{m}{\hat{s}^l}, \frac{n}{\hat{s}^l}\right] & \frac{m}{s^l}, \frac{n}{s^l} \in \mathbb{Z}, \\ 0 & \text{otherwise} \end{cases} \tag{3.14}$$

After the upsampling, another convolution operation is applied to the input similar to the linear interpolation filter in signal processing domain. Unlike the interpolation filter, the filter coefficients are learned rather than designed, which is shown to produce high-quality results compared to linear interpolators such as bilateral and bicubic interpolator [18, 19, 20]. The convolution operation for the synthesis transform can be written as

$$\hat{v}_i^{l+1}[m,n] = \sum_j \hat{h}_{ij}^l[m,n] * \hat{w}_j^l[m,n] + \hat{b}_i^l \tag{3.15}$$

Similar to the downsampling filters, the upsampling and convolution can be applied at once to reduce the number of computations. Hence Eq. 2.21 for 2D convolution can be rewritten as:

$$y[i+mM, i+nM] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[kM, lM] * g[m-k, n-l], i = 0, 1, ..., M-1 \tag{3.16}$$

Folllowing this operation, approximate IGDN is applied.

$$\hat{u}_i^l[m,n] = \hat{v}_i^l[m,n] \sqrt{\hat{\beta}_i^l + \sum_j \hat{\gamma}_{ij}^l (\hat{v}^l[m,n])^2} \tag{3.17}$$

Following notation, output image is expressed as $\mathbf{y} = u_c^3[m,n]$, where $c \in \{1,2,3\}$. The parameter set to be optimized for the synthesis transform is $\theta = \{\hat{h}_{ij}^l, \hat{b}_i^l, \hat{\beta}_i^l, \hat{\gamma}_{ij}^l\}$ which are kernel parameters and biases and scaling terms of IGDN.

The synthesis transform will be denoted as $\hat{\mathbf{x}} = g(\hat{\mathbf{y}}, \theta)$, where $\hat{\mathbf{y}}$ is the quantized transform coefficients and $\hat{\mathbf{x}}$ is the reconstructed/compressed image.

3.3.5 Rate Estimation

The rate estimation is an important topic for learned image compression and the work of Ballé et. al. [6] on this problem is closely related to this thesis work, hence it is

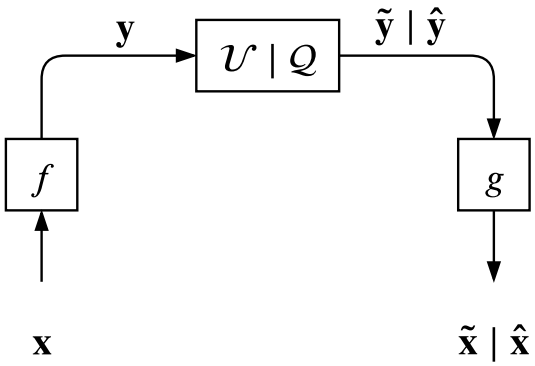


Figure 3.2: Operation diagram for the compression model in [5]. Arrows indicate the flow of data and boxes represent the transformation of the data. $\mathcal{U}|\mathcal{Q}$ represent the additive noise during the training or the quantization and arithmetic coding during testing. The vectors with a tilde is produced during training and the vectors with a hat is produced during testing. (redrawn from [6])

discussed in detail in Chapter 4 and summarized here. The latent variables are subject to noise during the training instead of the quantization, which can be used as a proxy for the quantization as discussed in Sec. 4.2. Fully connected neural networks for each channel in the latent space are trained to estimate the probability densities of each channel as explained in Sec. 4.3. The rate is estimated as the continuous entropy of the probability densities during training. For the density modeling, Ballé et. al. [5] assumes independence between the latent variables of a channel. In this thesis, the densities of latent variables are estimated using conditional density models as explained in Sec. 4.4.

3.3.6 Training

Since we aim to optimize for both rate and distortion, a Lagrangian function is to be optimized

$$\min_{\phi,\theta} J = H(P_q) + \lambda \mathbb{E}\|\mathbf{x} - \hat{\mathbf{x}}\| \tag{3.18}$$

$$= H(\hat{\mathbf{Y}}) + \lambda \mathbb{E}\|\mathbf{x} - g(f(\mathbf{x}, \phi), \theta)\| \tag{3.19}$$

$$= \mathbb{E}[-\log_2 p_{\hat{Y}}(\hat{y})] + \lambda \mathbb{E}\|\mathbf{x} - g(f(\mathbf{x}, \phi), \theta)\| \tag{3.20}$$

Since $\hat{\mathbf{Y}}$ and $\hat{\mathbf{x}}$ are both discrete, it is not straightforward to directly optimize the network weights over the loss function with gradient descent. The solution for these terms is using a differentiable proxy as discussed in detail in Sec. 4.2. The overall operation of the network is shown in Fig. 3.2.

3.4 Variational Autoencoders

Variational Autoencoders (VAE) [61] are one of the most popular tools for unsupervised learning of complex distributions. The variational Bayesian methods [62] aim to model a complicated distribution $p_X(x)$ of high dimensional random vector X, e.g. pdf of images, assuming an underlying latent space exists. The assumption that variational models make is that a latent distribution $p_Z(z)$ defined in a high dimensional space $\mathcal{Z}$ exists such that it is possible to find a model $f(z, \theta) : \mathcal{Y} \times \Theta \rightarrow \mathcal{X}$ where the set of parameters θ in the parameter space Θ is to be optimized. Using the variational model $f(z, \theta)$, it is possible to generate samples by randomly sampling $\mathcal{Z}$. The likelihood of the resulting model should be maximized to optimize the parameter set, θ, of the model. The maximum likelihood optimization is performed over the expression:

$$p_X(x) = \int p_X(x|z;\theta)p_Z(z)dz \tag{3.21}$$

For the VAEs, the output space is assumed to be a Gaussian distribution.

$$p_X(x|z;\theta) = \mathcal{N}(f(z;\theta), \sigma^2 I) \tag{3.22}$$

having mean $f(z;\theta)$ and covariance matrix $\sigma^2 I$. The preference for Gaussian distribution is due to its differentiability. The maximum likelihood expression can be written as

$$\mathcal{L}(\theta; x, z) = \arg\max_{\theta} \prod_{x_i \in \mathcal{D}} p_X(x_i|z;\theta) \tag{3.23}$$

Current deep learning frameworks require differentiable functions such that they can be optimized using the stochastic gradient methods. However, the evaluation of the integration Eq. 3.21 is not straightforward since the high dimensionality of the latent space $\mathcal{Z}$ taking the integral is not straightforward. This kind of integrations are

generally evaluated using Monte Carlo methods, such that the samples $\{z_i\} \in \mathcal{Z}$ is sampled randomly and the integral is approximated by

$$p_X(x) \approx \frac{1}{n} \sum_i p_X(x|z_i;\theta) \tag{3.24}$$

In practice, the posterior probability $p_X(x|z;\theta)$ is nearly zero for most of z, hence their contribution is very low for the maximum likelihood estimation. As a result, sampling meaningful results of x requires a lot of samples of z which may not be feasible in a high dimensional spaces in terms of memory and time. Variational Autoencoder (VAE) provides a solution for the sampling problem of z, it generates values of z that will have higher posterior probability, so that it is possible to maximize the likelihood. In other words, it provides a function $q(x) : \mathcal{X} \to \mathcal{Z}$ that generates z which are more likely to produce x. The distribution for the new samples of z values is shown with $p_Q(z|x)$.

To measure the distance between the distributions of the approximate generator $p_Q(z|x)$ and true prior $p_Z(z|x)$ KL divergence can be utilized,

$$\begin{aligned}
\mathcal{D}[p_Q(z|x)||p_Z(z|x)] &= \mathbb{E}_z \left[\log \frac{p_Q(z|x)}{p_Z(z|x)}\right] && (3.25)\\
&= \mathbb{E}_z \left[\log p_Q(z|x) - \log p_Z(z|x)\right] && (3.26)\\
&= \mathbb{E}_z \left[\log p_Q(z|x) - \log \frac{p_X(x|z)p_Z(z)}{p_X(x)}\right] && (3.27)\\
&= \mathbb{E}_z \left[\log p_Q(z|x) - \log p_X(x|z) - \log p_Z(z) - \log p_X(x)\right] && (3.28)\\
&= \mathbb{E}_z \left[\log p_Q(z|x) - \log p_X(x|z) - \log p_Z(z)\right] - \log p_X(x) && (3.29)
\end{aligned}$$

The definition of KL divergence is used in Eq. 3.25, Bayes' theorem is used in Eq. 3.27 and $\mathbb{E}_z$ is the expectation over the random variable z. The last term in Eq. 3.29 goes out of the expectation since it does not depend on z. Rearranging the terms yield the following equation:

$$\log p_X(x) - \mathbb{D}\left[p_Q(z|x)||p_Z(z|x)\right] = \mathbb{E}_z \left[\log p_X(x|z)\right] - \mathbb{D}\left[p_Q(z|x)||p_Z(z)\right] \tag{3.30}$$

Eq.3.30 is the main equation that is used for the training of VAE. Since the left-hand side is not easily calculated, it is called Evidence Lower Bound and the optimization procedure minimizes the right-hand side. To perform stochastic gradient descent optimization, the form of $p_Q(z|x)$ should be differentiable, which is choosen to be a Gaussian

$$p_Q(z|x) = \mathcal{N}(\mu(x;\phi_\mu), \Sigma(x;\phi_\Sigma)) \tag{3.31}$$

μ and Σ are deterministic functions to generate mean and diagonal covariance parameters for the multivariate Gaussian distribution with parameters ϕ_μ and ϕ_Σ. The selection of this function makes the computation of the second term in the right hand side of Eq. 3.30 easy:

$$\mathbb{D}\left[\mathcal{N}(\mu_1, \Sigma_1)||\mathcal{N}(\mu_2, \Sigma_2)\right] = \frac{1}{2}\text{tr}\ \Sigma_2^{-1}\Sigma_1 + (\mu_2 - \mu_1)^T\Sigma_1(\mu_2 - \mu_1) - k + \log\frac{\det\Sigma_1}{\det\Sigma_2} \tag{3.32}$$

Since z is assumed to be normally distributed with zero mean and identity covariance the equation simplifies to:

$$\mathbb{D}\left[\mathcal{N}(\mu_1, \Sigma_1)||\mathcal{N}(0, I)\right] = \frac{1}{2}\text{tr}\ \Sigma_1 + \mu_1^T\mu_1 - k + \log\det\Sigma_1 \tag{3.33}$$

where $\mu_1 = \mu(x;\phi_\mu)$ and $\Sigma_1 = \Sigma(x;\phi_\Sigma)$. The first term on the right-hand side is not differentiable since it is sampling x using a random variable z. The solution for that is the "reparametrization trick" that enables the training of VAE. To overcome the differentiability problem, the mapping function $q(x)$ is modified to generate the parameters of z instead of generating z directly.

$$\mathbb{E}_z\left[\log p_X(x|z = \mu(x) + \Sigma^{1/2}\epsilon)\right] - \mathbb{D}\left[p_Q(z|x)||p_Z(z)\right] \tag{3.34}$$

3.5 Variational Image Compression with a Scale Hyperprior

Balle et. al. [6] utilizes a scale hyperprior approach which makes the encoder send adaptive side information concerning the image that is being coded. The network learns to send side information related to the latent representation of the image. By utilization of the scale hyperprior, the fully factorized models of latent variables are

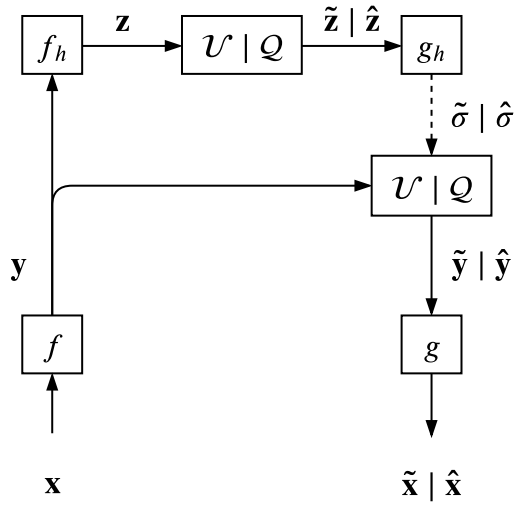


Figure 3.3: Operation diagram for the compression model in [6], same as in Fig. 3.2, but extended with the hyperprior model. (redrawn from [6])

modeled as independent Gaussians, hence the entropy is further reduced. Since the side information content is smaller than the reduction of the code length of the latent representation, the overall rate is reduced.

This procedure is analogous to sending the side information in HEVC [25] such as quadtree sizes. In contrast, the side information in [6] is learned, while the side information in HEVC is engineered, hence the scale hyperprior yields efficient compression.

To reduce the spatial dependencies among the latent variables y_i, they are further conditioned on independent variables z_i. Each y_i is modelled as zero-mean Gaussian with independent standard deviations σ_i. The encoder uses this information to scale the latent variables and codes them in as normalized variables. The scale information is sent independently to the decoder. At the decoder side, the scale information is decoded first and the recovered latent variables are denormalized to reconstruct the final latent variables to feed to the "decoder" part of the autoencoder. Overall operation diagram for the hyperprior model is given in Fig. 3.3.

3.6 Real-Time Adaptive Image Compression

Rippel and Bourdev [27] utilizes discriminative loss to introduce GANs into the learned compression literature and uses pyramidal structured convolutional networks in the feature extraction for scale-dependent structures.

GANs are known to be image generators that can synthesize high-resolution sharp images [63, 19]. Hence, the use of GANs is also yielding sharp images in low bitrates. On the other hand, training of GANs are highly unstable due to their training loss function and the authors propose a training scheme to improve stability.

For the entropy coding part, bitplane decomposition followed by adaptive arithmetic coding is used. After the feature extraction, The bitplane decomposition is the procedure of splitting the latent variables of size $C \times W \times H$ into binary planes of size $B \times C \times W \times H$, where C is the number of channels, W and H is the width and height in the latent domain, B is the number of bits after the quantization of the latent variables. The motivation behind this decomposition is to exploit the spatial correlations in the bit level, such that it is possible to achieve further compression.

A classifier is trained to predict the bits in the bitplane using the context features, and the probabilities at the output layer of the classifier is fed to arithmetic encoder. In the decoder step, the same context is used to estimate the probabilities and decode the bitstream.

3.7 Lossy Image Compression with Compressive Autoencoders

Theis et. al. [21] proposes an alternative way to deal with the quantization. Instead of relaxing the quantization function, which distorts the derivatives in the decoder, they only replace the derivatives such that the relaxation only affects the back-propagation.

$$\frac{d}{dy}\text{round}(y) := \frac{d}{dy}y \tag{3.35}$$

As a result of Eq. 3.35, the derivatives for the decoder is calculated with respect to the quantized values $\hat{y} = \text{round}(y)$ and the decoder's derivates directly propagate to

the encoder bypassing the quantizer.

Another advantage of this compression model is that, it is possible to tune the rate-distortion parameter after the training such that training of a single network is sufficient for a wide range of rate-distortion performances, i.e. compression factors. On the other hand, most of the compression networks [6, 5, 29], require retraining for a given rate-distortion parameter, i.e. compression factor.

The idea can be expressed as defining a scaling function:

$$S_\omega(f(\mathbf{x})) \coloneqq \frac{f(\omega \odot \mathbf{x})}{\omega} \tag{3.36}$$

where ω is the scale parameter set to be optimized for the desired rate distortion performance. Note that, x and w are vectors of the same size and $\odot$ is the element-wise multiplication and the division is also element-wise. Rewriting the rate-distortion cost in Eq. 3.20 using the scaling function,

$$\min_{\phi,\theta} J = \mathbb{E}[-\log_2 S_\omega(p_{\hat{Y}}(\hat{y}))] + \lambda \mathbb{E}\|\mathbf{x} - g(S_\omega(f(\mathbf{x},\phi)),\theta)\| \tag{3.37}$$

such that the latents are scaled before the quantization, hence the distortion induced by the quantization is reduced. The latents are modeled using Gaussian scale mixtures [64] with 6 scales for the bit rate estimation during the training. For the entropy coding, Laplacian-smoothed histograms are prepared using the training data. The neural network structure includes input normalization and output denormalization, residual blocks [65] and subpixel convolutions [66].

3.8 PixelRNN and PixelCNN

PixelRNN [28] and PixelCNN [67] architectures are a way to conditionally model images in the following form:

$$p(\mathbf{x}) = \prod_{i=1}^{N} p(x_i | x_{i-1}, \ldots, x_1, \mathbf{h}) \tag{3.38}$$

where $\mathbf{x}$ is the image signal and x_i are the image pixels and $\mathbf{h}$ is the optional high level description parameters, such as latent representations or labels. To condition the current pixel on only the previous pixels in raster scan order, masked convolutions are utilized which masks out the coefficients that are on the bottom or right, so that the network only processes the previous data. The conditional model generates a multinomial distribution of 256 possible values for each pixel to be generated. For an $N \times N \times 3$ image, $N \times N \times 3 \times 256$ predictions are produced. Although the training is parallel during the training, the network should run sequentially for each pixel prediction during the inference to condition the previous pixels which are already predicted.

PixelRNN models the pixel distributions using 2D LSTM [68, 69], which are good at modeling long term dependencies, while PixelCNN models the pixel distributions using convolutional neural networks, and much faster during training and inference than PixelRNN. To take the best of two worlds, ReLU activation functions between the masked convolutional layers in the PixelCNN are replaced with gated activation units:

$$\mathbf{y} = \tanh(W_f * x) \odot \sigma(W_g * x) \tag{3.39}$$

where σ is the sigmoid activation, $*$ is the convolution operator, W_f and W_g are the convolution kernels to be learned and $\odot$ is the element wise product.

3.9 Full Resolution Image Compression with Recurrent Neural Networks

Toderici et. al. [23] proposed an RNN based image compression pipeline for the variable rate compression on images. Encoder, decoder and binarizer parts of the networks are based on RNN and the entropy coding is performed by a fully connected NN. An image is divided into image blocks of 32×32 and compressed independently. Later, Toderici et. al. [24] also proposed another RNN model for increased independent compression performance and an entropy coder to capture spatial dependencies between image patches.

While CNN based image compression models require retraining for each point on the rate-distortion curve, usage of RNN enables compressing an image progressively,

which yields a variable rate compression. At each run, RNN produces residual bits that will be added to the previous ones, to increase the transmitted details. Since RNNs store their state until they are reset, each bit from different residual states are encoded and decoded under different contexts. The compression rate and the distortion is determined by the number of residual bits, i.e. number of iterations of the network.

Although the binarizer network is also RNN, it is chosen to be stateless, producing binary stream, $b \in \{-1, 1\}^m$ of length m. At each iteration, the same number of bits are produced, i.e. $k \times m$ bits are produced for k iterations.

In particular, the encoder network together with the binarizer works on image blocks of size 32×32, producing a binary stream of length $2 \times 2 \times 32$ for a colored image patch of size $32 \times 32 \times 3$. Hence, the first iteration yields $192 : 1$ compression without entropy coding.

The generated bitstream is progressively entropy-coded in raster scan order to exploit spatial dependencies among the generated bitstream. The bitstream is encoded using an architecture similar to PixelRNN [28], for conditional compression. This way, the encoded bits depends on the short term and long term information. Similar to [28], masked convolutions are utilized to enforce causality, making sure the bitstream can be decoded using previously decoded bits. At the output of the BinaryRNN, Bernoulli-distribution parameter is estimated using sigmoid activation, while the network is optimized for the cross-entropy loss.

3.10 Learning Content-Weighted Deep Image Compression

The image codes aim for the decorrelation of the spatial dependencies, which is very high for natural images. The lossy decorrelation processes usually compromise the high-frequency details like edges and corners, which generally carry significant information for the human observer. In addition to that, the spatial correlations still exist so that the resulting code lengths are still sub-optimal.

To overcome the drawbacks of the previous approaches, [29] proposes to make us of

an importance mask such that the bit rate allocation along the image is locally adaptive to the information content rather than uniform. The importance map is trained using the intermediate features extracted from the encoder to produce a single channel map having the same spatial size as the encoder output.

The importance map learns to adaptively allocate the number of channels that is used to code the codeword at a given location. As a result, it is possible to code the salient features using more channels, while smooth regions require fewer channels, which helps to preserve fine details without any increase in the code length. Since the code length corresponds to the number of channels utilized, and it is controlled by the importance map, it is possible to estimate the bit rate using the importance map without any assumptions on the distribution of quantized codes.

It is indicated that the spatial correlations still exist, hence a new method, named trimmed convolutional networks (TCNN), is proposed to make use of spatial dependencies for arithmetic coding to achieve higher bit rates. Trimmed CNN predicts the current symbols using the previous symbols which are available to the decoder, so that fewer bits are required. Furthermore, an inclined TCNN is proposed to partition the 3D code map to inclined planes such that they can be decoded in parallel.

3.11 Conditional Probability Models for Deep Image Compression

Mentzer et. al. [22] trains a 3D CNN to model the conditional distribution's latent space following a similar approach to [29]. A context model is trained to output an importance map to control the bit allocation along with the image, while the autoencoder learns to map the image into latent space.

Similar to [28], a conditional distribution of the quantized latent space representation $\hat{\mathbf{z}}$ is learned:

$$p(\hat{\mathbf{z}}) = \prod_{i=1}^{N} p(\hat{z}_i | \hat{z}_{i-1}, \ldots, \hat{z}_1) \tag{3.40}$$

The quantization is represented as:

$$\hat{z}_i = \arg\min_j ||z_i - c_j|| \tag{3.41}$$

and the relaxation is provided using the following formulation:

$$\tilde{z} = \sum_{j=1}^{L} \frac{\exp(-\sigma||z_i - c_j||)}{\sum_{k=1}^{L} \exp(-\sigma||z_i - c_k||)} c_j \tag{3.42}$$

While Eq. 3.41 is used in the forward pass, the gradients are calculated using Eq. 3.42, hence the differentiability is assured. This way, the measured loss during the training is very close to the inference stage because the forward and backward passes are isolated and it is possible to learn the quantization centers instead of the fixed points.

CHAPTER 4

PROPOSED METHOD

4.1 Overview

If the latent variables in a channel of the latent representation (i.e. transform coefficients of a neural network based non-linear transform) are assumed to be independent and their distribution does not change with the position of that variable inside the channel, then the entire channel's joint probability distribution can be represented using a single univariate probability distribution. In addition, arithmetic coding of the entire channel can be performed in a single manner using this single univariate distribution. This approach was used in [5]. However, our empirical observations indicate that latent variables in a channel are not independent. Indeed, the authors of [6] improve their work by accounting for the dependencies using a hyperprior neural network that transforms the latent variables to another set of variables, conditioned on which the latent variables become independent.

This thesis proposes an alternative way to model the dependencies of latent variables to reduce the modeling error. The joint density of the latent representation is modeled as a product of conditional densities. However, only a few of the neighbors are taken into account for the conditioning based on Markov property assumption. Hence, the latent variables are only conditioned on the adjacent values, namely upper, left and upper-left pixels. The conditional densities are no longer univariate functions due to the conditions and are learned with neural networks. The analysis for the conditional density model using neural networks follows a similar analysis to [30].

In this chapter, we first discuss how the zero-derivative problem of the quantizer is handled in neural network based compression systems in the literature in Sec. 4.2.

Next, starting with the detailed explanation of the univariate model solution from [6], the chapter continues with the details of the conditional density model in Sec. 4.4. The two approaches are compared using a toy example on an artificial source distribution in Sec. 4.5. The use of arithmetic coding with the univariate and conditional density model are discussed in Sec. 4.6 and 4.7. Finally, the overall compression system is outlined and supported with diagrams in Sec. 4.8.

4.2 Differentiable Quantization

A major problem with end-to-end learned image compression systems is that the quantization operation has zero derivative, hampering its use in differentiation based learning frameworks. Fig. 2.1 shows the plot of a basic quantizer that is implemented with rounding:

$$\hat{y} = Q(y) = \texttt{round}(y) \tag{4.1}$$

The optimization frameworks generally require differentiable functions, however the derivative of the quantizer given in Eq. 4.1 is zero at non-integer points and is undefined at integer points. To overcome this challenge, different methods are proposed. One of them is to approximate the quantization using additive noise as it is used in the signal processing literature [70]. To understand how this method works, the effect of quantization should be investigated. Let us start by assuming a laplacian distribution with mean, μ, and scale, b, for the source values to be compressed.

$$f_Y(y;\mu,b) = \frac{1}{2b}\text{exp}\left(-\frac{|y-\mu|}{b}\right) \tag{4.2}$$

The PDF of y for $\mu = 0$ and $b = 1$, as well as the PMF of $\hat{y}$ given in Eq. 4.1, is shown in Fig. 4.1. The problem with the quantization in this case is that the back-propagation algorithm will encounter zeros for the gradients almost everywhere, hence the optimization algorithms will not be able to minimize the cost function. To have an estimate entropy for the intermittent values, [4] proposes to add continuous

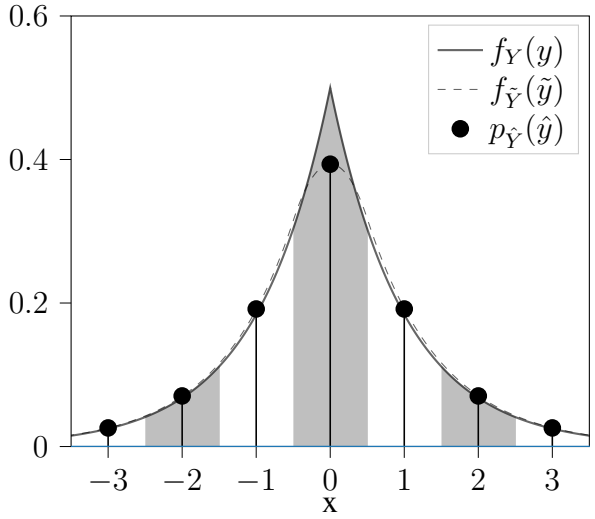


Figure 4.1: Continuous Laplacian distribution with zero mean and unity scale parameter is shown with $f_Y(y)$. Addition of a uniform noise to the samples from Laplacian distribution corresponds to the convolution of their densities, which is shown as $f_{\tilde{Y}}(\tilde{y})$. The uniform quantization of the samples from the Laplacian distribution corresponds to the integration of probability density function at the integer numbers within a unity window, which is shown with the probability mass function $p_{\hat{Y}}(\hat{y})$.

uniform noise, $w \sim \mathcal{U}[-0.5, 0.5]$, such that $\tilde{y} = y + w$:

$$f_W(w) = \begin{cases} 1 & \text{if } w \in [-0.5, 0.5], \\ 0 & \text{otherwise.} \end{cases} \tag{4.3}$$

Hence, the PDF of $\tilde{y}$ can be expressed as:

$$\begin{aligned} f_{\tilde{Y}}(\tilde{y}) =& f_Y(\tilde{y}) * f_W(\tilde{y}) && (4.4)\\ =& \int_{-\infty}^{+\infty} f_W(\theta) f_Y(\tilde{y} - \theta) d\theta && (4.5)\\ =& \int_{-0.5}^{0.5} f_Y(\tilde{y} - \theta) d\theta && (4.6)\\ =& F_Y(\tilde{y} - \theta)\Big|_{\theta=0.5}^{\theta=-0.5} && (4.7)\\ =& F_Y(\tilde{y} + 0.5) - F_Y(\tilde{y} - 0.5) && (4.8) \end{aligned}$$

where Eq. 4.3 in step 4.5 represent the PDF of the noise, and $F_Y(y) \coloneqq \int_{-\infty}^{x} f_Y(y)$

is defined as the cumulative distribution function (CDF) of $f_Y(y)$. Interestingly, the distributions match at integer locations, such that the resulting PDF $f_{\tilde{Y}}(\tilde{y})$ can be used as a proxy for the quantized distribution $p_{\hat{Y}}(\hat{y})$ as shown in Fig. 4.1:

$$p_{\hat{Y}}(\hat{y}) = f_{\tilde{Y}}(\tilde{y}) \iff \hat{y} = \tilde{y} \in \mathbb{Z} \tag{4.9}$$

This way the quantizer is relaxed and now a differentiable function can be used in rate/entropy estimation in a gradient descent optimization framework. Since the distribution is continuous now, the rate estimation should be calculated using differential entropy rather than discrete entropy.

4.3 Univariate Non-parametric Density Modelling

A univariate probability density function[1] $p_X(x) : \mathbb{R} \to \mathbb{R}^+$ and its CDF $F_X(x) : \mathbb{R} \to [0, 1]$ should satisfy the following constraints:

$$F_X(-\infty) = 0 \tag{4.10}$$

$$F_X(\infty) = 1 \tag{4.11}$$

$$\frac{\partial F_X(x)}{\partial x} = p_X(x) \geq 0 \tag{4.12}$$

These conditions imply that the CDF should be a monotonically increasing function, starting from 0 raising up to 1. If we want to express the CDF by series of K functions such as:

$$F_X(x) = f_K \circ f_{K-1} \circ ... \circ f_1 \tag{4.13}$$

then the PDF can be expressed using the chain rule of calculus as the derivative of the functions according to Eq. 4.12:

$$p_X(x) = \frac{\partial F_X(x)}{\partial x} \tag{4.14}$$

$$= \frac{\partial}{\partial x}(f_K \circ f_{K-1} \dots f_1) \tag{4.15}$$

$$= f_K^{'} \cdot f_{K-1}^{'} \dots f_1^{'} \tag{4.16}$$

[1] This section uses the notation $p_X(x)$ for the probability densities and $f_k(x)$ for the neural network layers to be coherent with the notation in the literature.

where $f_k^{'} = \dfrac{\partial f_k}{\partial x}$ are the partial derivatives. Eq. 4.16 can also be used when f_k are vector functions such that $f_k : \mathbb{R}^{N_k} \rightarrow \mathbb{R}^{N_{k+1}}$, where $\mathbb{R}^{N_k}$ is the domain of f_k and the range of f_{k-1}. If f_k is a vector function, $f_k^{'}$ is a Jacobian matrix with entries are derivatives with respect to each dimention:

$$f_k^{'} = \frac{\partial f_k}{\partial x} = \begin{bmatrix} \dfrac{\partial f_k}{\partial x_1} \\ \dfrac{\partial f_k}{\partial x_2} \\ \vdots \\ \dfrac{\partial f_k}{\partial x_{N_{k-1}}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f_{k_1}}{\partial x_1} \cdots \dfrac{\partial f_{k_1}}{\partial x_{N_{k-1}}} \\ \dfrac{\partial f_{k_2}}{\partial x_1} \cdots \dfrac{\partial f_{k_2}}{\partial x_{N_{k-1}}} \\ \vdots \\ \dfrac{\partial f_{k_{N_k}}}{\partial x_1} \cdots \dfrac{\partial f_{k_{N_k}}}{\partial x_{N_{k-1}}} \end{bmatrix} \tag{4.17}$$

Hence the PDF can be expressed as series of matrix multiplications, which suggests the use of fully connected neural networks explained in Sec. 2.9. To ensure that $p_X(x)$ is univariate, input and output dimensions of $p_X(x)$ are $N_0 = N_K = 1$.

According to the Eq. 4.12, $F_X(x)$ should be monotonically increasing since $p_X(x)$ is nonnegative. Although $F_X(x)$ can be represented by neural networks with several layers, the constraints in Eq. 4.10, 4.11 and 4.12 should be imposed such that the resulting function is a proper probability distribution function. Although imposing the constraints in Eq. 4.10 and 4.11 can be as simple as applying the sigmoid activation function at the output of the last layer f_K, the procedure to satisfy Eq. 4.12 is not straightforward.

To assure that the derivative of $F_X(x)$ is always positive, the analysis of derivatives should be conducted. Let f_k be fully connected layers as defined in Sec. 2.9,

$$f_k(\mathbf{x}_k) = g_k(\mathbf{W}_k \mathbf{x}_k + \mathbf{b}_k) \tag{4.18}$$

$$f_K(\mathbf{x}_K) = \sigma(\mathbf{W}_K \mathbf{x}_K + \mathbf{b}_K) \tag{4.19}$$

where $k \in \{1 \dots K-1\}$ are the intermediate layers and K denotes the output layer, σ is the sigmoid nonlinearity, used to constrain the output in the range $[0, 1]$. The

derivatives of the layers are

$$f'_k(\mathbf{x}_k) = \mathbf{W}_k \,\text{diag}\, g'_k(\mathbf{W}_k\mathbf{x}_k + \mathbf{b}_k) \tag{4.20}$$

$$f'_K(\mathbf{x}_K) = \mathbf{W}_K \,\sigma'(\mathbf{W}_K\mathbf{x}_K + \mathbf{b}_K) \tag{4.21}$$

To ensure that $p_X(x) \geq 0$, all the elements of $f'_k \in \{1 \dots K\}$, i.e. the Jacobian matrices given in Eq. 4.12 must be nonnegative. Starting with Eq. 4.20, it is assumed that the nonlinearity g_k is a pointwise nonlinearity, hence its Jacobian is a diagonal matrix, since $\dfrac{\partial g_{k_i}(\mathbf{x})}{\partial x_j} = 0, \forall i \neq j$. The nonlinearity g_k is chosen as

$$g_k(\mathbf{x}_k) = \mathbf{x}_k + \mathbf{a}_k \odot \tanh(\mathbf{x}_k) \tag{4.22}$$

where $\mathbf{a}_k$ is a parameter vector and $\odot$ is the element-wise multiplication. This particular choice for the nonlinearity enables the function to model troughs as easy as the peaks of $p_X(x)$ as indicated by Balle et. al. [6].

The derivative of the nonlinearity given in Eq. 4.22 is

$$g'_k(\mathbf{x}_k) = 1 + \mathbf{a}_k \odot \tanh'(\mathbf{x}_k) \tag{4.23}$$

To ensure that Eq. 4.23 is nonnegative, all of $\mathbf{a}$'s elements should be greater than -1, since the derivative of tanh : $\mathbb{R} \to [-1, 1]$ is already nonnegative:

$$\tanh'(x) = 1 - \tanh^2(x) \tag{4.24}$$

To assure nonnegativity of g'_k, $\mathbf{a}$ could be reparametrized to yield values greater than -1,

$$\mathbf{a}_k = \tanh(\hat{\mathbf{a}}_\mathbf{k}) \tag{4.25}$$

For Eq. 4.20, the only thing that remains is to assure that $\mathbf{W}_k$ is a nonnegative matrix. For this purpose, a similar reparametrization is applied here such that

$$\mathbf{W}_k = \sigma^+(\hat{\mathbf{W}}_k) \tag{4.26}$$

where $\sigma^+ : \mathbb{R} \to \mathbb{R}^+$ is the softplus function. Overall, all elements of $\mathbf{W}_k$ and the

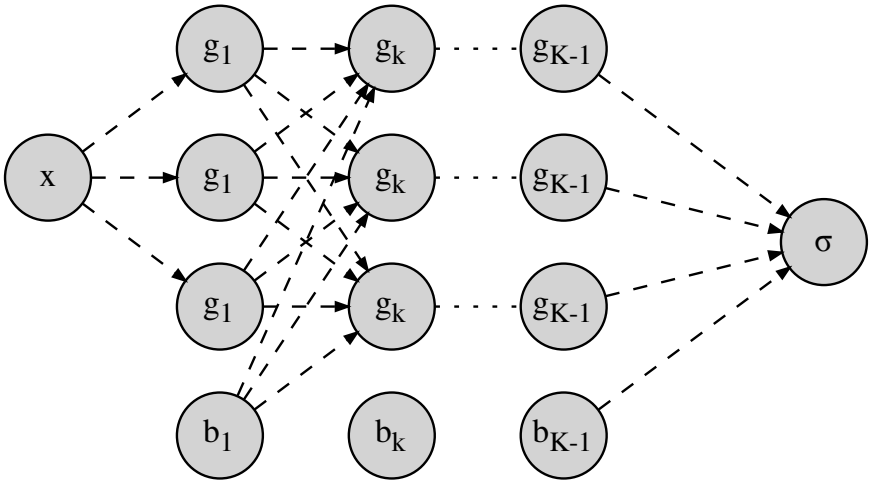


Figure 4.2: Network diagram for the univariate density model. The dashed lines correspond to softplus reparametrization, nodes with g_j are the nonlinearity function given in Eq. 4.22, b_j are the bias terms, and the σ corresponds to the sigmoid nonlinearity at the output.

diagonal matrix in Eq. 4.20 are positive. To ensure positiveness of the Jacobian matrix in Eq. 4.21, a similar parametrization for $\mathbf{W}_K$ can be used and note the derivative of a sigmoid is always positive.

In summary, the univariate non-parametric density model can be implemented using a neural network with the following explicit form:

$$f_k(\mathbf{x}_k) = \sigma^+(\hat{\mathbf{W}}_k)\mathbf{x}_k + \mathbf{b}_k + \tanh(\hat{\mathbf{a}}) \odot \tanh(\sigma^+(\hat{\mathbf{W}}_k)\mathbf{x}_k + \mathbf{b}_k) \tag{4.27}$$

$$f_K(\mathbf{x}_K) = \sigma(\sigma^+(\hat{\mathbf{W}}_K)\mathbf{x}_K + \mathbf{b}_K) \tag{4.28}$$

$\Phi \in \{\hat{\mathbf{W}}_\mathbf{k}, \mathbf{b}_k, \hat{\mathbf{a}}_\mathbf{k}\}$, $k \in \{1 \dots K\}$ is the parameter set to be optimized using back-propagation algorithm. Fig. 4.2 provides a graphical representation of the neural network.

4.4 Conditional Density Model

To guarantee the validity of a *univariate* CDF[2] generated by a neural network, the conditions that must be satisfied are

$$\lim_{x\to-\infty} F_X(x) = 0 \tag{4.29}$$

$$\lim_{x\to\infty} F_X(x) = 1 \tag{4.30}$$

$$\frac{\partial F_X(x)}{\partial x} = p_X(x) \geq 0 \tag{4.31}$$

as explained in detail in Sec. 4.3. The validity of *conditional* CDF generated by a neural network are analyzed in [30] and similar conditions should be satisfied:

$$\lim_{x\to-\infty} F_X(x|\mathbf{y}) = 0 \tag{4.32}$$

$$\lim_{x\to\infty} F_X(x|\mathbf{y}) = 1 \tag{4.33}$$

$$\frac{\partial F_X(x|\mathbf{y})}{\partial x} = p_X(x|\mathbf{y}) \geq 0 \tag{4.34}$$

Eq. 4.32 and 4.33 is satisfied using sigmoid nonlinearity at the output layer of the neural network. An example network is shown in Fig. 4.3. To achieve Eq. 4.34, every layer following x should be constrained such that the weight matrix should be nonnegative $\mathbf{W_j} \geq 0$. This can be shown using a similar procedure discussed in Sec. 4.3. Let $F_X(x)$ denote a K layer neural network and let f_j denote the j-th layer of it.

$$F_X(x) = f_K \circ f_{K-1} \circ \cdots \circ f_1(x) \tag{4.35}$$

After the introduction of the conditional terms, Eq. 4.35 becomes

$$F_X(x|\mathbf{y}) = f_K \circ f_{K-1} \circ \cdots \circ f_1(x|\mathbf{y}) \tag{4.36}$$

It is clear that the analysis to show how to achieve $\frac{\partial F_X(x|\mathbf{y})}{\partial x} \geq 0$ does not change for the layers following x as long as $\frac{\partial f_i(x)}{\partial dx} \geq 0$, which is achieved by reparametrization of the weights using softmax as in Sec. 4.3. If the conditional variables are designed

[2] This section uses the notation $p_X(x)$ for the probability densities and $f_k(x)$ and $h_k(x)$ for the neural network layers to be coherent with the notation in the literature.

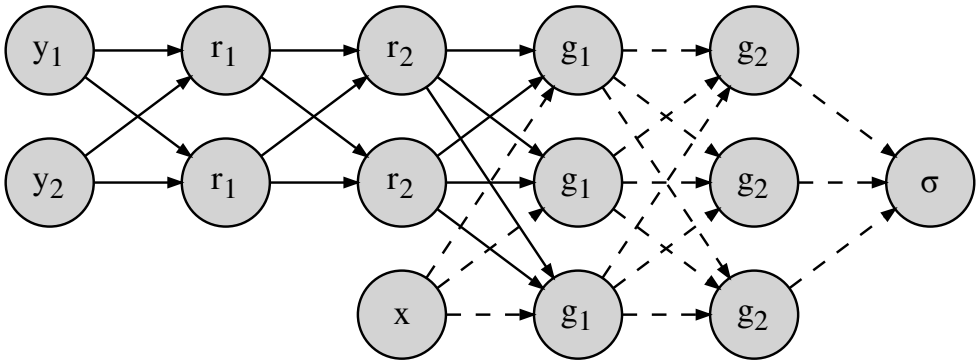


Figure 4.3: Network diagram for the conditional density model. The dashed lines correspond to softplus reparametrization, while the solid lines correspond to unconstrained parameters. The nodes with g_j are the nonlinearity function given in Eq. 4.22, the nodes with r_j are the unconstrained nonlinearity functions. The input nodes labeled with y_j correspond the condition variables. Finally, the σ corresponds to the sigmoid non-linearity and the bias terms are omitted in the diagram.

to be transformed using extra layers to model the condition space better, Eq. 4.36 becomes

$$F_X(x|\mathbf{y}) = f_K \circ f_{K-1} \circ ... \circ f_1(x|H(\mathbf{y})) \tag{4.37}$$

$$H(\mathbf{y}) = h_L \circ h_{L-1} \circ \cdots \circ h_1(\mathbf{y}) \tag{4.38}$$

where $H()$ denotes the composition of L layers to process only y. The weights forming the conditional terms, namely the weights of h_j, are not constrained to be positive since there are no constraints for the conditional terms. A network satisfying the conditional CDF requirements is given in Fig. 4.3, where $\mathbf{y} = [y_1, y_2]$, $K = 3$ and $L = 2$. The layers with positive weights are indicated with dashed lines, while straight lines indicate the layers with unconstrained weights.

4.5 Modelling Simple Distributions using the Density Models

A simple example is demonstrated to show how the density models perform for a dependent signal. A stochastic signal is generated making use of three Laplacian distributions and a Markov chain to model the dependencies between consecutive samples. The three laplacian distributions have the same scale parameter and are differentiated

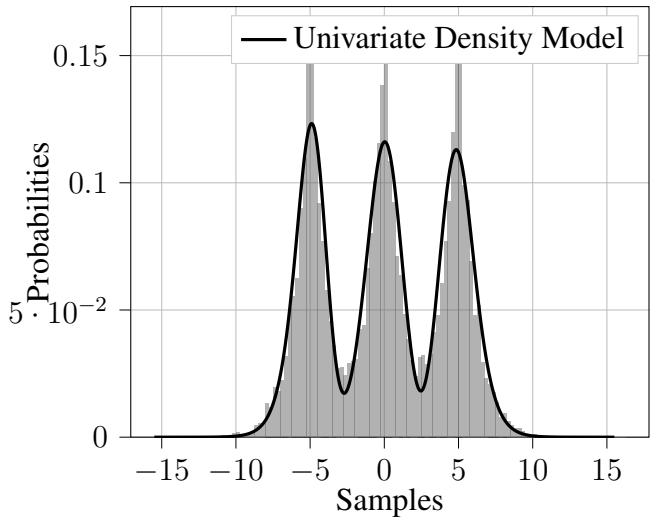


Figure 4.4: A dependent source is modeled using the transition matrix given in Eq. 4.39. Each sample is sampled from one of the three Laplacian distributions with different means and same scale parameters. The probability of sampling from the same distribution as the previous sample is 0.6, while the probability of sampling from other two distributions is 0.2. The univariate density model learns to represent this density model, ignoring any dependencies among adjacent samples.

by their mean μ that can have three different values $\mu_i \in \{-5, 0, 5\}$. Each sample of the sequence $\{X_n\}$ is sampled from one of three Laplacian distributions $f_{X_i}(x; \mu_i, 1)$ $i \in \{1, 2, 3\}$.

The distribution to be sampled is determined with a state machine. The transition probability between 3 different states $s_i, i \in \{1, 2, 3\}$, is given by the state transition matrix,

$$P_{s_i|s_{i-1}} = \begin{bmatrix} 0.6 & 0.2 & 0.2 \\ 0.2 & 0.6 & 0.2 \\ 0.2 & 0.2 & 0.6 \end{bmatrix} \tag{4.39}$$

According to Eq. 4.39, there is a 0.6 probability that the consecutive samples are from the same distribution and 0.4 probability of sampled from a different distribution. 10000 samples are generated to create a dataset using this formulation and their histogram is as shown in Fig. 4.4. The generated dataset is then used for training of the univariate model and the output of the univariate model is shown in Fig 4.4 with solid lines. The univariate network can model the frequencies of occurrence for the given distribution, however, the resulting model does not incorporate the dependen-

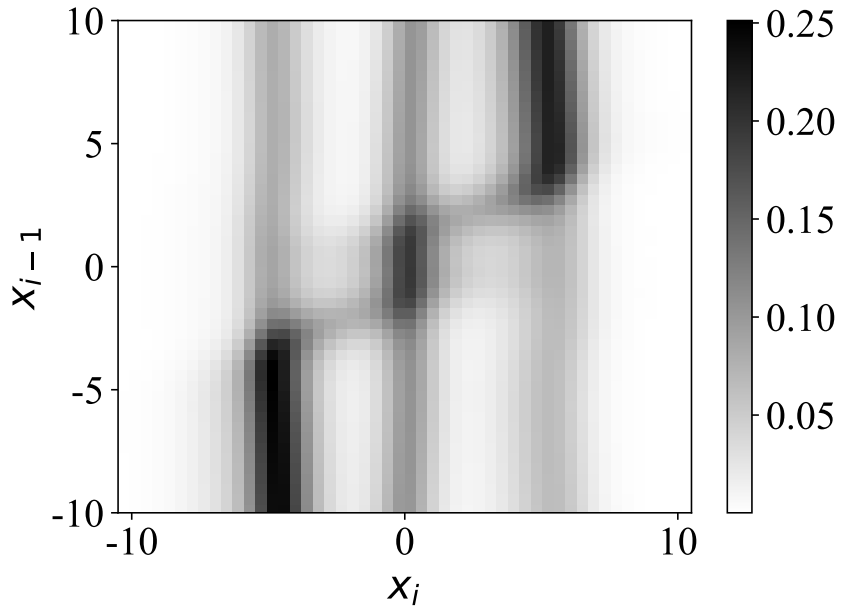


Figure 4.5: The plot represents the output of the conditional density model trained on the samples data similar to 4.4. While *x-axis* shows the sampled values x_i, *y-axis* represent 1 adjacent/previous sample values x_{i-1}. The colors represent the conditional probability density function $f_{X_i}(x_i|x_{i-1})$. In contrast to uniform density model, the conditional density model is able to extract the dependence information and able to represent the density closer.

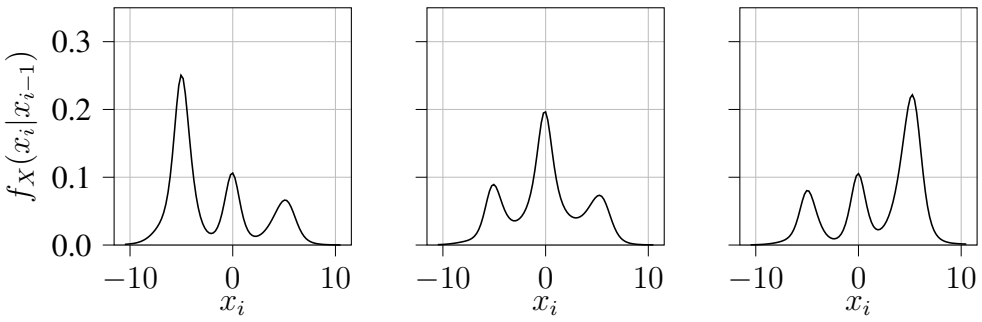


Figure 4.6: The output of the conditional density model is plotted for three different conditions $x_{i-1} = -5, x_{i-1} = 0, x_{i-1} = 5$ respectively, which correspond to 3 horizontal slices of the plot in Fig. 4.6. It is clear that the density model assigns higher likelihoods around the mean if the distributions from which the condition x_{i-1} is likely to come while assigning lower but peaky likelihoods to the means of the other two distributions.

cies between the consecutive samples. This causes inefficiency in the bitrate of the compression system, since the extra information about the signal is not modeled and its price is paid as extra bits.

The conditional density modeling is useful for modeling this type of distributions, where close pairs are dependent. To model dependencies of neighboring latent variables, the conditional density model is trained using the same dataset, but this time, previous samples are fed as a conditional variable to the network in Sec. 4.4. In other words, the network is trying to model $p_X(x_i|x_{i-1})$. After the training, the function that the conditional density model represents is drawn in Fig. 4.5. In this figure, the current signal values, x_i are represented on the *x-axis* and the previous values of the signal values x_{i-1} are represented on the *y-axis*. For example, the network estimates the probability of the next symbol to be around $x_i = 0$ much higher, if the value of $x_{i-1} = 0$. In addition to that, the peak probabilities shifts to 5 when the previous value is increased and shifts to -5 when the previous value is decreased. This behaviour is clearer in Fig 4.6, which includes 3 sub-figures, each drawn for a given previous sample value $X_{i-1} \in \{-5, 0, 5\}$ respectively. For example, $X_{i-1} = 0$ for the center figure, and probability of sampling from the distribution $f_X(x; \mu = 0, 1)$ is more likely than probability of sampling from the distribution with $\mu = -5$ or $\mu = 5$ and therefore the PDF has higher likelihoods around the mean of $f_X(x; \mu = 0, 1)$ than around the means of the other two distributions.

4.6 Arithmetic Coding Using Univariate Density Model

After training of the compression network, all that remains is coding of transform coefficients using the learned probability models. A practical implementation of the arithmetic encoder works with integer values for CDF and the finite number of symbols due to varying implementations of the floating-point arithmetic across different hardware. In addition, a discrete PMF is required rather than a continuous one, due to the discrete nature of computers. Hence, the quantization of the continuous CDF and the limitation of the possible number of symbols is necessary.

Conversion of the continuous density model $f_{\tilde{Y}}(\tilde{y})$ to discrete model $p_{\hat{Y}}(\hat{y})$ is already discussed in Sec. 4.2 and handled during the training by introducing uniform noise to the transform coefficients. After the training, sampling the CDF at integer points is sufficient.

The quantization of the discrete CDF is necessary for arithmetic encoding, due to implementation differences of floating-point hardware. Firstly, the floating points have finite precision and coding the arbitrary number of symbols in the range $[0, 1]$ is not possible. Secondly, the implementation of the floating-point arithmetic may differ across the software and hardware platforms. As a result, the quantization of the CDF is required by arithmetic encoders. The step size for the quantization controls the trade-off between distortion and the complexity of the arithmetic encoder. Smaller steps size causes lower distortion, hence yields higher compression efficiency, while increasing the complexity of the encoder and the decoder so that the encoding and decoding times are increased. The steps size is generally powers of 2 and controlled by the bit depth parameter of B in the expression 2^{-B}. Practical values of B for the arithmetic coding algorithm used in this work is in the range of $[10, 16]$.

The last step is limiting of a discrete quantized CDF to represent the finite number of symbols. The univariate model represents a continuous monotonic function, and its input is unbounded during the training, In other words, the network yields a probability value for symbol in the range $[-\infty, \infty]$. For the arithmetic encoder to support this kind of input, infinite precision is required, hence the number of possible symbols should be handled correctly. The CDF model is clipped at a predefined value, and only the symbols inside the clipped range are encoded using arithmetic coding. The arithmetic encoder assigns zero probability to the symbols outside this range, which means that if a symbol outside this range is encountered, (even though the occurrence of this event has a very low probability, it is still possible) it cannot be encoded using arithmetic coding.

The problem of coding the symbols falling outside of the range is solved by creating a density model for these symbols. After the training, a normalized histogram for the latent variables which are outside of the density range is calculated. After that, the histogram is used as a density model in the arithmetic coder when a latent variable which cannot be encoded using the conditional density model.

4.7 Arithmetic Coding Using Conditional Density Model

Arithmetic coding using the conditional density model is mostly the same as using the univariate model. While the discretization and quantization parts are the same, the limitation (i.e. clipping to a finite range) part requires some more processing.

When performing arithmetic coding using the conditional density model, each symbol to be coded is using a different CDF, determined from the conditional density network using the previous coded symbols' value as the condition.

Finding the range of the univariate model is rather easy, since the neural network is approximating a monotonically increasing function in 1D. In the conditional model, space becomes ND, and the complexity of searching the space to find the tails for every symbol combination grows exponentially. Applying the binary search algorithm while the function $F_X(x)$ is univariate has complexity $\mathcal{O}(\log n)$. By addition of a single conditional variable $F_X(x|y)$, even though the complexity for a given $Y = y$ is still $\mathcal{O}(\log n)$, the tails need to be found for each y, which increases the complexity to $\mathcal{O}(N \log n)$, assuming that y takes N distinct values. By addition of another conditional variable $F_X(x|y_1, y_2)$, the complexity becomes $\mathcal{O}(N^2 \log n)$. For arbitrary conditional vector $\mathbf{y} \in \mathbb{R}^M$ for a given the CDF $F_X(x|\mathbf{y})$, the complexity of finding ranges is $\mathcal{O}(N^M \log n)$.

4.8 Overall System

The overall system for the training of the network is given in Fig. 4.7. An input image is fed to the convolutional encoder for the analysis transform to obtain the latent space coefficients. The output of the transform is subject to additive uniform noise of unit length and zero mean during training. The distorted latent variables are fed to convolutional decoder for the reconstruction of the input image. Mean squared error (MSE) is used for the evaluation of distortion by comparing the reconstructed image from the distorted latent variables and the input image. In the meantime, the distorted latent variables are used in the density model for the conditional distribution estimation for each channel in the latent space. Each channel in the latent space is fed to a

separate fully connected neural network to model the dependencies in that channel. The conditional density model is conditioned on three neighboring coefficients. In this thesis, only three neighbors (top, left, top left) are used for conditional density modeling. The coefficients that are on the boundaries of the channels and do not have the necessary neighbors, are assumed to have neighbors with zero value, such as the first latent variable in a channel which is missing all three neighbors. The likelihoods from the output of the density model is used to estimate the entropy. Finally, the rate and distortion losses are combined into a single loss using a scalar multiplier.

After the training phase, the network structure changes such that the additive noise is replaced with quantization for compression of the latent variables. In addition, the arithmetic coder is inserted to the graph instead of the rate estimation. The overall block diagram for the testing is shown in Fig. 4.8. Similar to the training phase, an image is transformed using convolutional encoder. The latent variables are quantized to be used by the arithmetic encoder and density estimation. The arithmetic coder uses the density model for the range estimation for a given latent variable. The density model requires 3 neighbors variables (top, left and top left) of each coefficient and outputs a density or range of values for each integer latent value. For the variables that do not have any neighboring pixels, neighbors with the zero value are assumed similar to the training phase. The ranges are also quantized to prevent floating-point errors in the arithmetic coder. Range values are used in the arithmetic encoder to code the corresponding latent variable and each channel in the latent space is coded independently from other channels. In other words, each channel uses its own conditional density model for compression.

The coded latent variables are sent to the decoder. To decode the bitstream, the arithmetic decoder requires the same density model, so that each code can be recovered without any error. Since the densities or the range values of the coded variables depends on the neighbors, the value of the neighbors should always be available to the density model. However, this is not the case for the edge values, because they do not have any neighbors. To solve this issue, the missing neighbors are assumed to be zero as it is the case in the encoding process. For example, for the decoding of the first latent variable without any neighbors, the density model assumes the neighbors to be zero. The next variable will have a left neighbor, but others are assumed to be zero

again. This way, the decoding of the edge values of each channel is possible.

The hyperparameters of the overall network are shown in Fig. 4.9. For example, the analysis and synthesis transforms each have 3 layers, use filters with support of 3×3 or 5×5 and have 128 channels/feature maps.

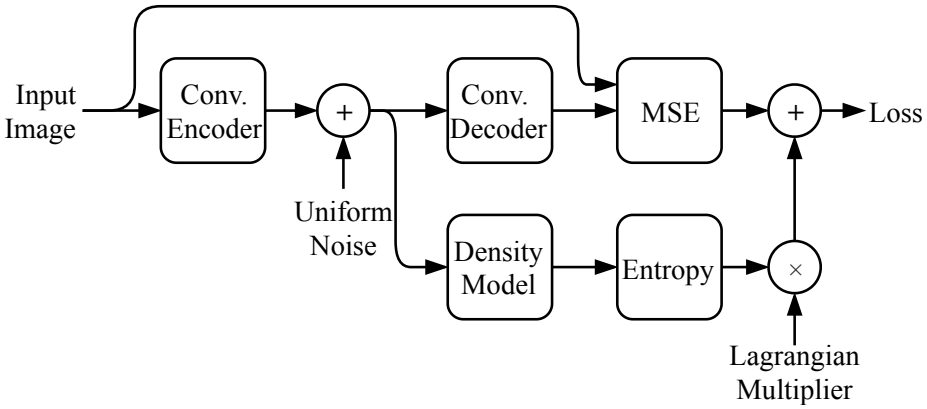


Figure 4.7: Graph used in the training of the proposed network. An input image is transformed using convolutional encoder. The transform coefficients are subject to noise during training as a proxy for quantization. The distorted latent variables are inverse transformed by convolutional decoder and image are reconstructed. The quality of the reconstruction is evaluated using mean squared error. In the meantime, each channel of the latent variables are sent to a separate density model for the likelihood estimation. Every coefficients likelihood is evaluated using their adjacent coefficients with a conditional density model. The likelihoods are used for the rate estimation by evaluating their average entropy. Finally, the rate and distortion values are combined into a single loss using a scalar multiplier, which controls the amount of distortion and the bit-rate.

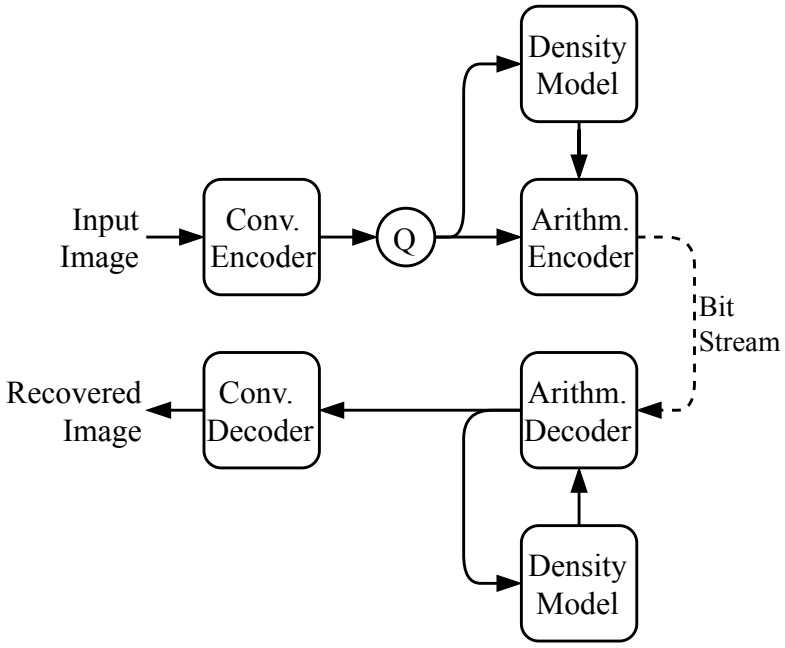


Figure 4.8: Graph used in the testing of the proposed network. An image is transformed using convolutional encoder. The transform coefficients are quantized and sent to the density model and the arithmetic encoder. The density model outputs range values for quantized codes and arithmetic encoder uses these ranges to encode the corresponding coefficients. The arithmetic decoder uses the same density model to estimate the ranges and decode the latent variables one-by-one. The decoded latent variables are inverse transformed and the image is reconstructed.

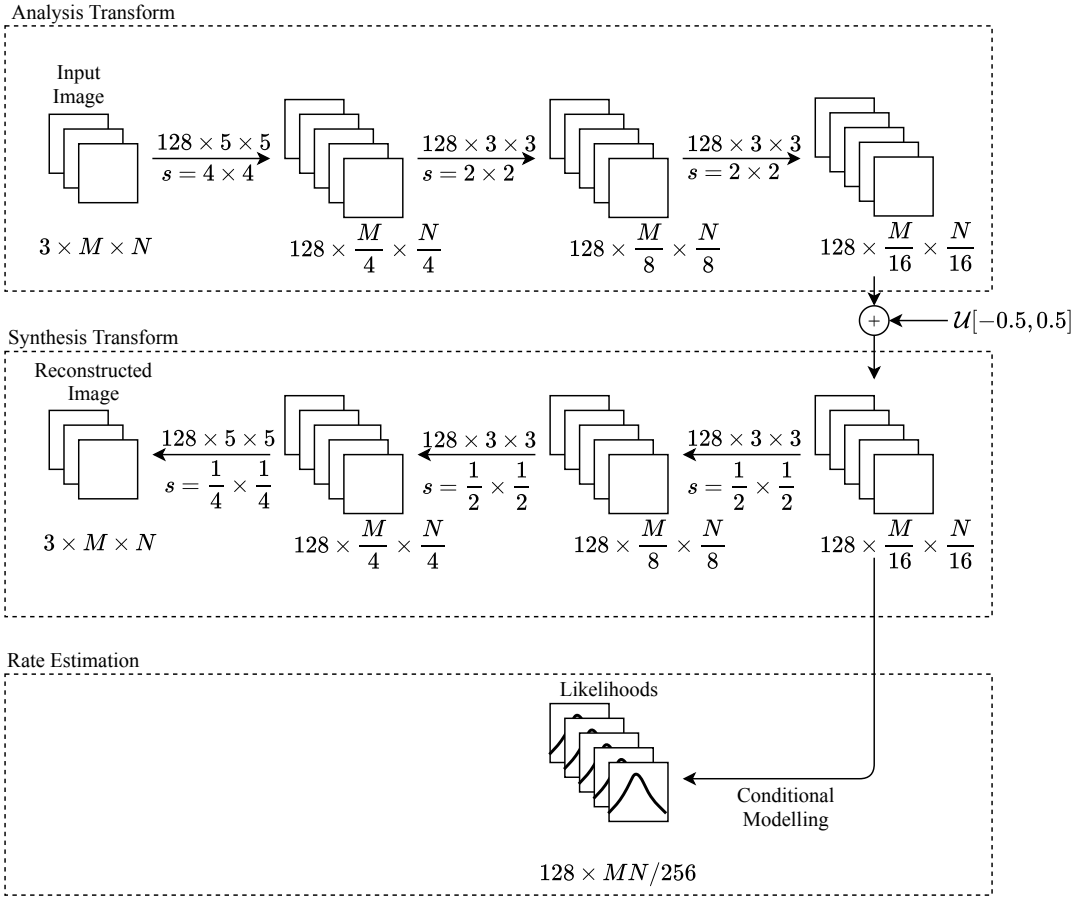


Figure 4.9: Detailed visualization of the network layers. Each convolutional layer has 128 channels and followed by GDN non-linearity. The stride values of each channel are shown with s. The output of the encoder is fed to rate estimation block where each channel has a separate fully connected NN.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Overview

This chapter presents the image compression performance of the proposed network. The details of training and testing are also discussed. Visualizations of the latent space and the correlations of the latent variables are also presented to show the improvements with the conditional density models.

5.2 Training and Evaluation

A combination of two uncompressed public image datasets are used for the training of the neural network. A set of approximately 2000 uncompressed natural images of size 2040×1356 from DIV2K dataset [71, 72] and the dataset from CVPR 2019 - Challenge on Learned Image Compression (CLIC) is used for training. PyTorch [73], a deep learning library, is used for implementation of the convolutional and fully connected layers. *rangecoder*[1] is used for arithmetic coding of quantized latent variables. Batch size is set to 8 images and a batch is created by randomly cropping 256×256 pathces from the images in the dataset. Learning rate is set to 10^{-5} for the convolutional layers and GDN layers and 10^{-4} for the conditional density model. The networks are trained for 10^6 iterations or approximately 5000 epochs.

The comparisons to other algorithms are done by measuring the rate and distortion performance of the algorithm on the Kodak Image Dataset[2]. Kodak Dataset consists

[1] github.com/kazuho/rangecoder
[2] r0k.us/graphics/kodak/

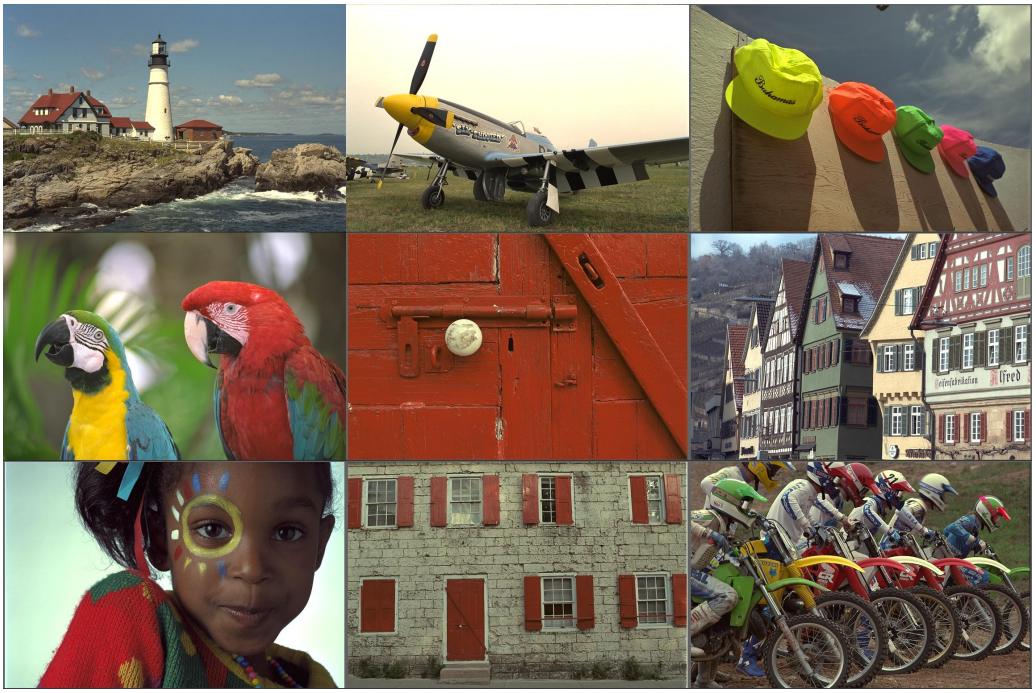


Figure 5.1: Nine images from the Kodak Image Dataset of 24 images.

of 24 RGB images of size 768×512, where a subset of the dataset is shown in Fig. 5.1. The rate is calculated by the size of the file written by the arithmetic coder and reported as bit per pixels (bpp) using the formula $\frac{\text{total number of bits}}{\text{total number of pixels}}$. The distortion is calculated using the Peak Signal-to-Noise Ratio (PSNR) defined as $10 \log_{10} \frac{255^2}{\text{MSE}}$.

5.3 Compression Performance

To evaluate the compression performance of the proposed neural network based system in the thesis, we compare its rate-distortion curve over the Kodak Image Dataset with 4 other compression systems:

- JPEG[1]
- BPG [2]
- Univariate density model based neural network [5]
- Hyperprior incorporating neural network [6].

JPEG, proposed in 1992, is the most popular compression standard due to its simplicity and decent performance. Compression in JPEG is performed by applying 8×8 block-DCT, quantization, run-length and Huffman coding steps. BPG, based on the intra-frame coding of HEVC [25, 2], is the current state-of-the-art in the traditional image compression. Intra-frame coding is performed by intra-prediction of the quadtree blocks and followed by the transform coding of residuals. The univariate density model based neural network [5] is the pioneer work in the literature which this thesis is based on. An image is transformed using a convolutional autoencoder and the latent variables are coded using the independent density estimation of the latent variables. The hyperprior incorporating neural network [6], written by the same authors, captures the spatial dependencies in the latent representation using a hyperprior network which estimates the mean and the scale parameter of the distribution assuming Gaussian density.

Rate and distortion are the competing parameters for image compression systems. The classical compression algorithms provide an option to tune the trade-off between the rate and distortion. It is possible to compress an image with lower distortion and higher rate or vice versa. A rate-distortion curve is drawn for each system by changing their operation point on the rate-distortion curve to compare their performance. To achieve this, JPEG provides a *Quality* parameter from 0 to 100, BPG also provides a *Quality* parameter from 0 to 51 which is known as *QP* in HEVC. Unfortunately,

the neural networks require retraining for each operation point on the rate-distortion curve.

Fig. 5.2 shows the rate-distortion curves for each compression system as well as the proposed network in this thesis. The worst compression is performed by JPEG, because it is aimed to be very low complexity. The next is the neural network with univariate density model where the indepence assumption hinders its performance. Followed by the hyperprior model and the conditional model which are built upon the previous network, perform on par. The hyperprior models the density for each latent variable as a Gaussian density and estimates its mean and scale parameter. Our density model assumes Markov property for the latent representation and models the densities conditionally using adjacent latent variables. Finally, the best performing algorithm is the BPG, which is a heavily engineered state-of-the-art compression algorithm used mainly for the coding of intra-frames in HEVC.

The encoding and decoding times of the compared algorithms are given in Table 5.1. JPEG is by far the fastest system, followed by BPG. The neural network based systems require more time due to their complexity. The conditional model proposed in this work require much more time compared to the other neural network based systems, because the encoding and decoding is performed sequentially. In addition to that, the tails of the density should be searched each time a latent variable needs to be coded/decoded and this operation has very high complexity depending on the number of conditioned variables as explained in Sec. 4.7. The duration of the encoding/decoding times are measured on a system with Ubuntu 18.04 with 16GB of RAM and Intel i7-6700HQ. JPEG is working on MATLAB, BPG is working on C, neural networks are working on Python on CPU. The default configuration values are used for JPEG and BPG.

	Encoding Time (sec.)	Decoding Time (sec.)
JPEG [1]	0.01	0.01
BPG [2]	0.38	0.17
Univariate Mdl. [5]	7.80	7.60
Hyperprior Mdl. [6]	9.10	8.50
Cond. Mdl. (this work)	53.62	50.12

Table 5.1: Encoding and decoding times of compression algorithms.

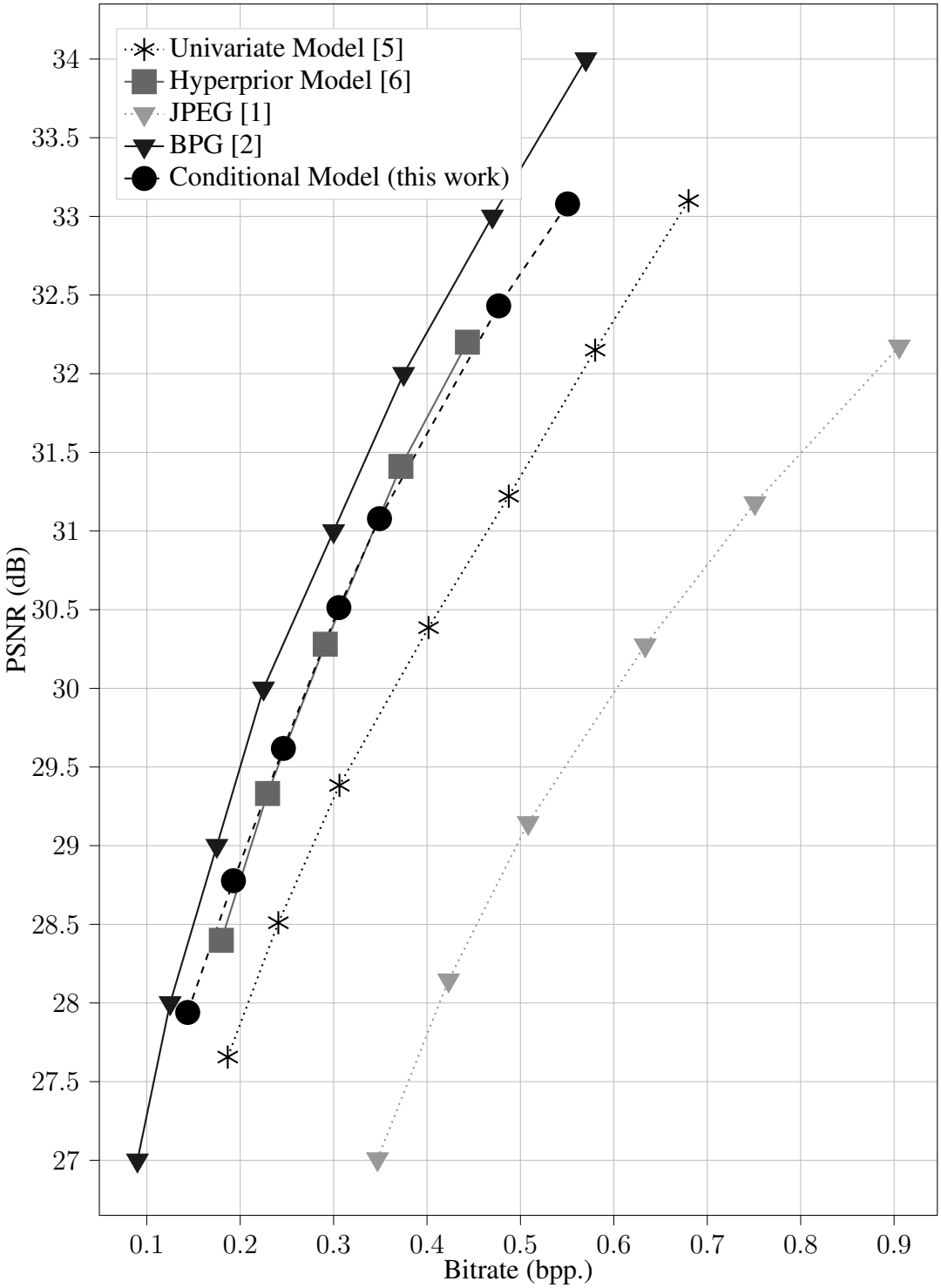


Figure 5.2: Comparison of the rate distortion performance of the proposed network with other compression algorithms. The conditional model (this work) improves the univariate model [5] by 30% and have almost the same performance with the hyperprior model [6].

5.4 Visualization and Histograms of the Latent Space

After the training is finished, quantized latent variables from our neural network based systems for a single image in the dataset is shown in Fig. 5.3. 128 channels in the latent space are drawn in raster order, to create 16-by-8 grid of latent images. The input image, *kodim23.png*, is the fourth image in Fig. 5.1 and its size is 768×512, the encoder/analysis transform of the network downsamples the input by 16, hence each channels are represented by 48×32 image patches.

As seen in Fig. 5.3, most of the channels are very sparse. In fact, most of the channels have coefficients y are either $y \in \{0\}$ or $y \in \{-1, 0, 1\}$. In other words, only a few of the channels contains most of the information. This property is called energy compaction and studied extensively for linear transforms [11, 10]. This is expected since the network is trained to minimize the rate while maintaining low distortion. Bit rate for this image is 0.10 bpp and PSNR is 30.91 dB.

The channels with higher entropies are drawn in Fig. 5.5 seperately and upsampled by 8 using nearest neighbour upsampling to emphasize the details. The histograms of the coefficients are shown in Fig. 5.6. It should be noted that the *y-axis* of the histograms are in *log-scale* to emphasize the sparsity of the channels. Even after the 5th channel with the highest entropy, the counts are quickly decaying to zero. This is expected since lower entropy implies impulse-like probability distributions.

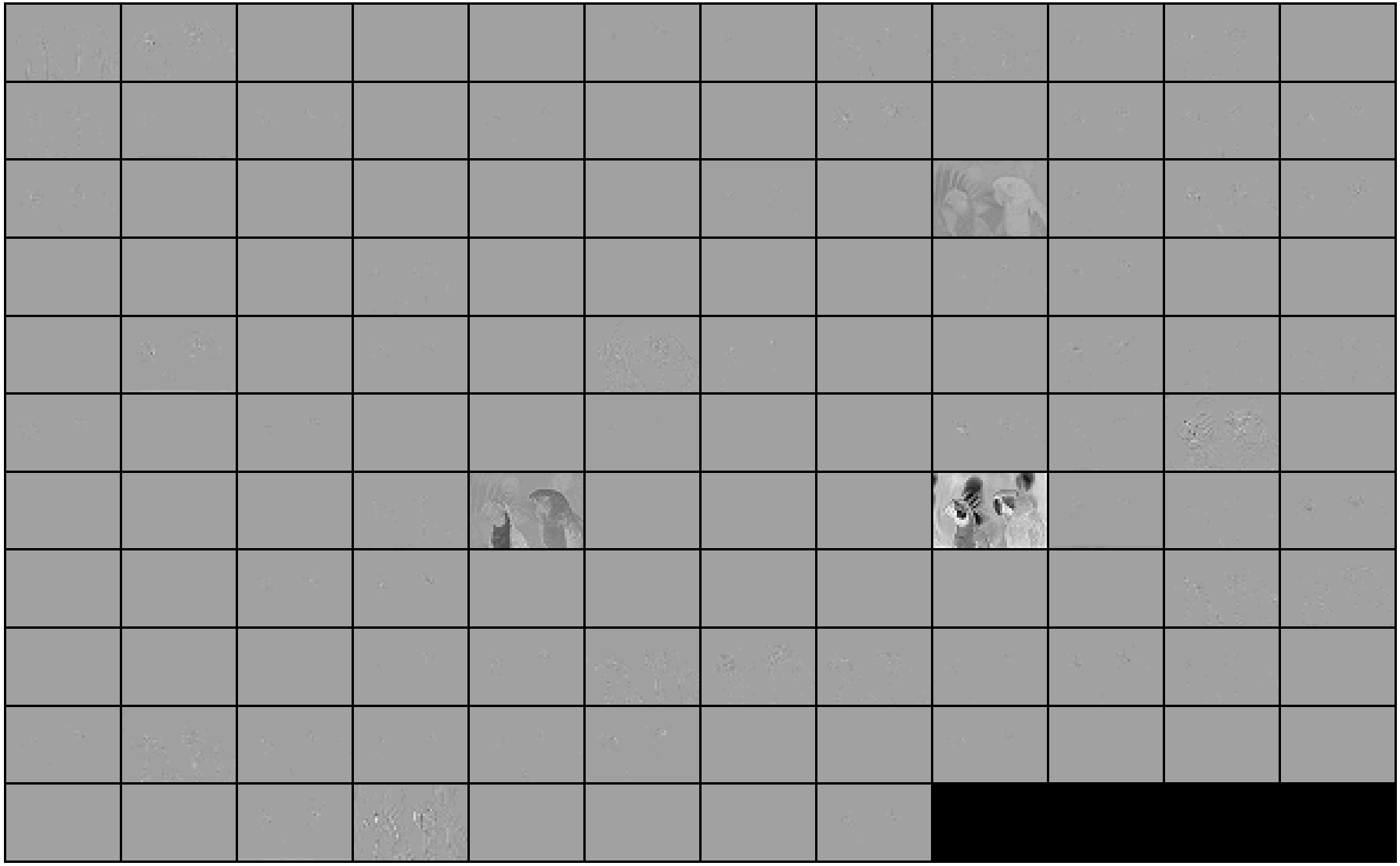


Figure 5.3: The visualization of the quantized latent space after training of the neural network. Each channel in the latent space is drawn separately in raster order. All of the channels are sparse except three of them. The black blocks at the end appear due to padding and can be ignored.

5.5 Conditional Densities of the Latent Space

While the histograms in Fig. 5.6 provide information about the marginal distributions of the latent variables, the pictures of the channels in Fig. 5.5 indicate that the neighboring latent variables are dependent. A simple evidence of the dependency is given in Fig. 5.4, which provides the joint 2D histogram of two adjacent latent variables. Hence, our neural network that models the dependencies using conditional densities performs better at compression. An example of the learned conditional densities $f_Y(y|y_left, y_top)$ is provided in Fig. 5.7. As can be seen from Fig. 5.7, there is a strong dependency between the latent variable and its left and upper neighbors. The mean and variance of the latent variable can change significantly depending on the value of the left and upper neighbors.

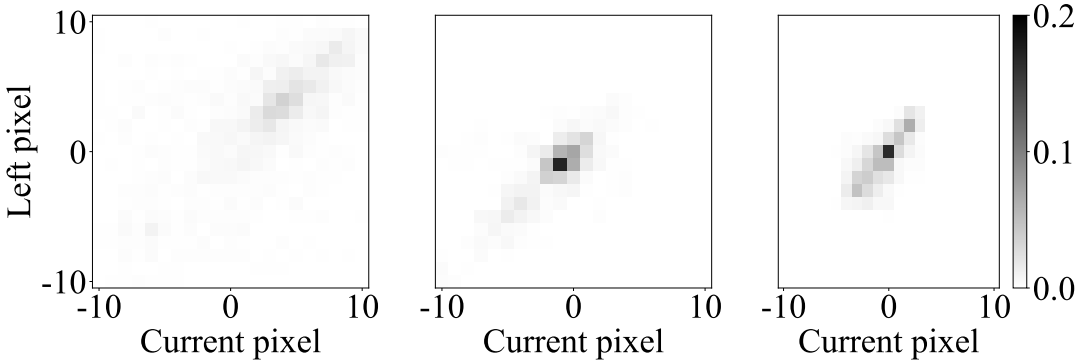


Figure 5.4: 2D histograms of the adjacent latent value pairs is shown for three channels with highest entropy. To draw the histograms, quantized pixels are paired with their left neighbors to create a 2D pair, then each unique pair is counted and represented on the figure.

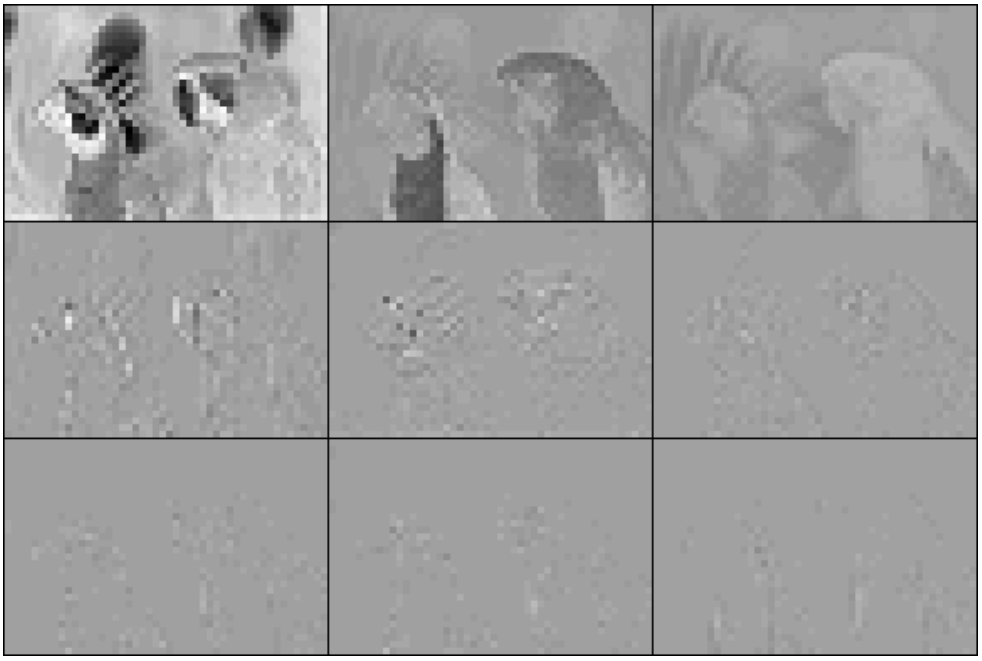


Figure 5.5: Channels with the highest entropy from Fig. 5.3 are redrawn to emphasize the details. While images in the first row resembles the original image, the remaining ones have less and less details about the image. In fact, the remaining channels are used mostly for coding of the edges. There is a one-to-one correspondence between the latent variable images, i.e. channels, and their histograms in Fig. 5.6. Meaning that the image and histogram at the same position are related.

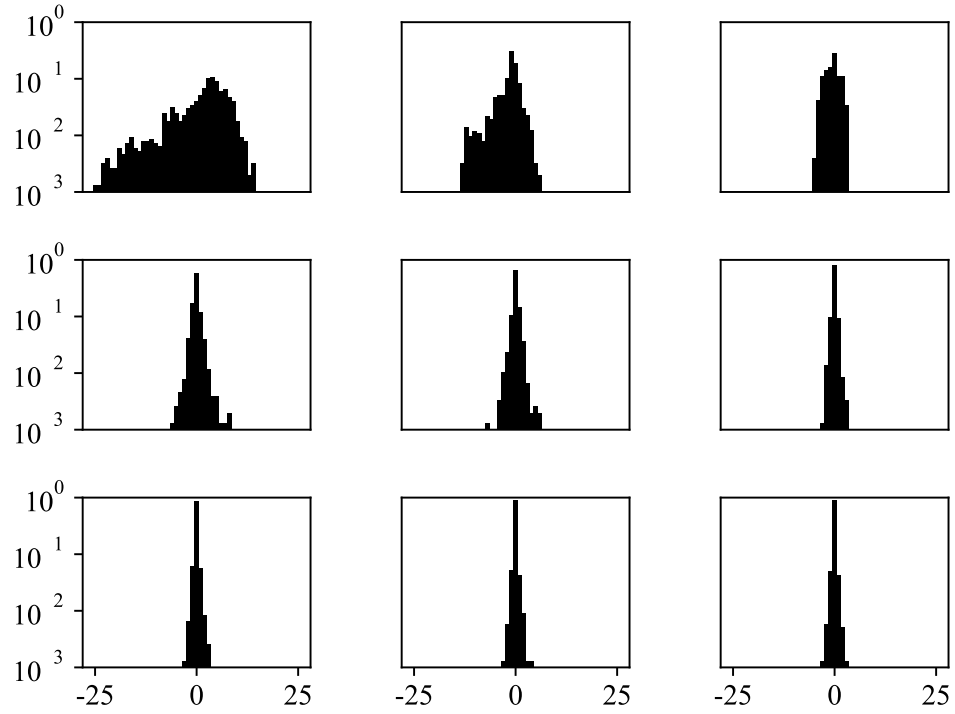


Figure 5.6: The number of occurrences of a latent variable for each channel in Fig. 5.5 are plotted as a histogram and normalized. Note that the *y-axis* is in log-scale. The channels distributions quickly becomes narrow and more than 90% of a channel is just zeros. The histograms are normalized so that, their sum is 1.0.

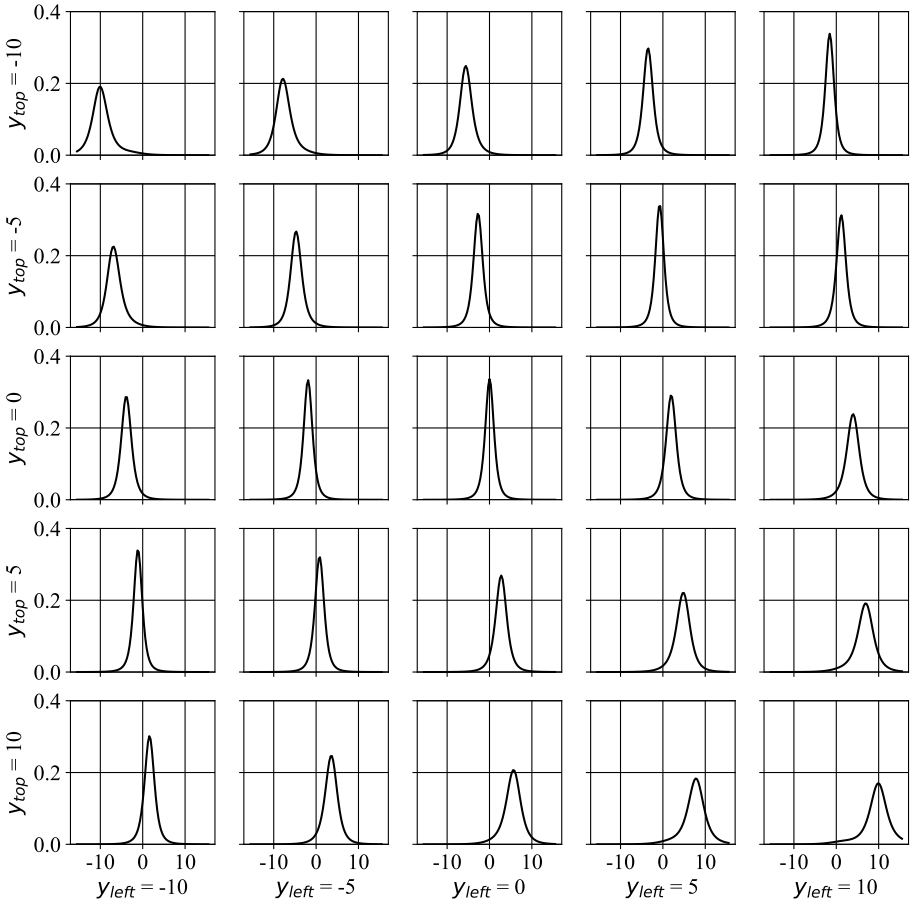


Figure 5.7: A grid of conditional density plots, $f_Y(y|y_{left}, y_{top})$, for a single channel in the latent space is given. 5-by-5 grid is composed of 5 different values for adjacent pixels, i.e. mesh of $y_{left}, y_{top} \in \{-10, -5, 0, 5, 10\}$. The network learns the positive correlations shown in Fig. 5.4. For example, when both of the neighbours $y_{left} = y_{top} = 10$, the likelihood peaks at $y = 10$. To make the visualization easier, a network is trained with the densities conditioned only on two neighbours (only for this plot), namely left and top.

5.6 Visual Results

Two images from the Kodak Image Dataset are coded with all image compression systems that we use in comparisons. Fig. 5.8 shows an image compressed at the same rate (0.21 bpp.) using 5 different systems. Fig. 5.9 shows a small patch from the same image. The BPG system introduces the lowest distortion in terms of MSE, followed by the network proposed in this thesis. The distortion is visible around the text and the faces.

Fig. 5.10 shows another image encoded at the same PSNR (~ 26.5 dB) and Fig. 5.11 shows a smaller area from the same image. The PSNR is set as close as possible across different algorithms. The BPG system achieves lowest bpp., followed by hyperprior model [6] and the conditional model(this thesis).

Original Image

[4] Univariate Model (26.00 dB, 0.21 bpp)

[5] Hyperprior Model (26.75 dB, 0.21 bpp)

[1] JPEG (21.85 dB, 0.21 bpp)

[2] BPG (27.17 dB, 0.21 bpp)

This work (PSNR: 26.86, Rate: 0.21 bpp)

Figure 5.8: Comparison of the images coded with different algorithms.

Original Image

[4] Univariate Model (26.00 dB, 0.21 bpp)

[5] Hyperprior Model (26.75 dB, 0.21 bpp)

[1] JPEG (21.85 dB, 0.21 bpp)

[2] BPG (27.17 dB, 0.21 bpp)

This work (PSNR: 26.86, Rate: 0.21 bpp)

Figure 5.9: Comparison of the images coded with different algorithms.

Original Image

[4] Univariate Model (26.84 dB, 0.67 bpp)

[5] Hyperprior Model (26.51 dB, 0.44 bpp)

[1] JPEG (26.50 dB, 0.88 bpp)

[2] BPG (26.66 dB, 0.39 bpp)

This work (PSNR: 26.81, Rate: 0.45 bpp)

Figure 5.10: Comparison of the images coded with different algorithms.

Original Image

[4] Univariate Model (26.84 dB, 0.67 bpp)

[5] Hyperprior Model (26.51 dB, 0.44 bpp)

[1] JPEG (26.50 dB, 0.88 bpp)

[2] BPG (26.66 dB, 0.39 bpp)

This work (PSNR: 26.81, Rate: 0.45 bpp)

Figure 5.11: Comparison of the images coded with different algorithms.

CHAPTER 6

CONCLUSIONS

In this thesis work, a neural network architecture for end-to-end learning of image compression is presented. The proposed network jointly learns the transform and entropy coding operations using neural networks, while minimizing the rate-distortion cost estimated from a dataset of images.

An end-to-end learning framework is presented in [5] by coding the latent variables with an independent density model. The same authors then extend their work by incorporating a hyperprior to capture the spatial dependencies in the latent representation [6].

This thesis uses an alternative method to exploit the dependencies of the latent representation. The joint density of the latent representation is modeled as a product of conditional densities based on Markov property. In other words, each latent variable is not conditioned on all previous latent variables but only on a few previous variables, in particular the left, upper and upper-left spatial neighbors of that latent variable. The analysis for conditional density model is conducted similar to the monotonic neural density estimator proposed in [30]. Use of conditional density model increases the performance of the arithmetic coder, because each latent variable is coded using a density estimated from the adjacent variables and has better density estimation.

The compression performance of the proposed network is compared to other classical compression algorithms, as well as learning-based compression algorithms on Kodak Image Dataset for different rate-distortion performance. The proposed network improves the coding efficiency of the univariate density model in [5] by 30% and is on par with the hyperprior model in [6].

6.1 Future Work

In future, the proposed method could be extended in two ways. Firstly, the encoding/decoding duration could be decreased. Secondly, the dependencies between the latent variables between the channels could be explored to achieve higher compression efficiency.

In the current approach, the tails of the densities are required to be searched for each latent variable using the conditional variables, which increases the time required to encode/decode an image. The encoding/decoding duration could be decreased by minimizing the number of tail searches. This is possible by storing/caching the data for the frequently used conditional variables in a map. In addition to that, the same map could be used as an initial condition for a search with close values of conditional variables to decrease the convergence time for the search of the tails of the densities.

The visualizations of the latent space in Fig. 5.3 indicates that there is also dependency between the channels of the latent space for the same spatial location. This dependency could be explored in a future work to increase the coding efficiency.

REFERENCES

[1] G. K. Wallace, "The jpeg still picture compression standard," *IEEE transactions on consumer electronics*, vol. 38, no. 1, pp. xviii–xxxiv, 1992.

[2] J. Lainema, F. Bossen, W. Han, J. Min, and K. Ugur, "Intra coding of the hevc standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1792–1801, Dec 2012.

[3] P. Helle, S. Oudin, B. Bross, D. Marpe, M. O. Bici, K. Ugur, J. Jung, G. Clare, and T. Wiegand, "Block merging for quadtree-based partitioning in hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1720–1731, Dec 2012.

[4] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimization of nonlinear transform codes for perceptual quality," *CoRR*, vol. abs/1607.05006, 2016.

[5] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," *CoRR*, vol. abs/1611.01704, 2016.

[6] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," in *International Conference on Learning Representations*, 2018.

[7] K. Sayood, "13 - transform coding," in *Introduction to Data Compression (Third Edition)* (K. Sayood, ed.), The Morgan Kaufmann Series in Multimedia Information and Systems, pp. 391 – 422, Burlington: Morgan Kaufmann, third edition ed., 2006.

[8] V. K. Goyal, "Theoretical foundations of transform coding," *IEEE Signal Processing Magazine*, vol. 18, pp. 9–21, Sep. 2001.

[9] K. Sayood, "8 - mathematical preliminaries for lossy coding," in *Introduction to Data Compression (Third Edition)* (K. Sayood, ed.), The Morgan Kaufmann Se-

ries in Multimedia Information and Systems, pp. 195 – 225, Burlington: Morgan Kaufmann, third edition ed., 2006.

[10] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transform," *IEEE transactions on Computers*, vol. 100, no. 1, pp. 90–93, 1974.

[11] H. Kekre, S. D. Thepade, A. Athawale, A. Shah, P. Verlekar, and S. Shirke, "Performance evaluation of image retrieval using energy compaction and imagetiling over dct row mean and dct column mean," in *Thinkquest˜ 2010*, pp. 158–167, Springer, 2011.

[12] L. Deng, D. Yu, *et al.*, "Deep learning: methods and applications," *Foundations and Trends® in Signal Processing*, vol. 7, no. 3–4, pp. 197–387, 2014.

[13] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep cnn denoiser prior for image restoration," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3929–3938, 2017.

[14] M. Gharbi, G. Chaurasia, S. Paris, and F. Durand, "Deep joint demosaicking and denoising," *ACM Transactions on Graphics (TOG)*, vol. 35, no. 6, p. 191, 2016.

[15] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[16] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in neural information processing systems*, pp. 341–349, 2012.

[17] R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, "Semantic image inpainting with deep generative models," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5485–5493, 2017.

[18] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1646–1654, 2016.

[19] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

[20] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[21] L. Theis, W. Shi, A. Cunningham, and F. Huszár, "Lossy image compression with compressive autoencoders," 03 2017.

[22] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Conditional probability models for deep image compression," *CoRR*, vol. abs/1801.04260, 2018.

[23] G. Toderici, S. M. O'Malley, S. Jin Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar, "Variable rate image compression with recurrent neural networks," 11 2015.

[24] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell, "Full resolution image compression with recurrent neural networks," *CoRR*, vol. abs/1608.05148, 2016.

[25] G. J. Sullivan, J. Ohm, W. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, pp. 1649–1668, Dec 2012.

[26] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the h.264/avc video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, pp. 560–576, July 2003.

[27] O. Rippel and L. Bourdev, "Real-time adaptive image compression," in *International Conference on Machine Learning*, 2017.

[28] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," *CoRR*, vol. abs/1601.06759, 2016.

[29] M. Li, W. Zuo, S. Gu, J. You, and D. Zhang, "Learning content-weighted deep image compression," *CoRR*, vol. abs/1904.00664, 2019.

[30] P. Chilinski and R. Silva, "Neural likelihoods via cumulative distribution functions," 11 2018.

[31] K. Sayood, "2 - mathematical preliminaries for lossless compression," in *Introduction to Data Compression (Third Edition)* (K. Sayood, ed.), The Morgan Kaufmann Series in Multimedia Information and Systems, pp. 13 – 39, Burlington: Morgan Kaufmann, third edition ed., 2006.

[32] K. Sayood, "9 - scalar quantization," in *Introduction to Data Compression (Third Edition)* (K. Sayood, ed.), The Morgan Kaufmann Series in Multimedia Information and Systems, pp. 227 – 271, Burlington: Morgan Kaufmann, third edition ed., 2006.

[33] T. Wiegand and H. Schwarz, "Source coding: Part i of fundamentals of source and video coding," *Found. Trends Signal Process.*, vol. 4, pp. 1–222, Jan. 2011.

[34] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[35] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, pp. 7–12, March 1960.

[36] T. Wiegand and H. Schwarz, *Video Coding: Part II of Fundamentals of Source and Video Coding*. now, 2016.

[37] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[38] A. Moffat, "Huffman coding," *ACM Comput. Surv.*, vol. 52, pp. 85:1–85:35, Aug. 2019.

[39] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, 1987.

[40] K. Sayood, "4 - arithmetic coding," in *Introduction to Data Compression (Third Edition)* (K. Sayood, ed.), The Morgan Kaufmann Series in Multimedia Infor-

mation and Systems, pp. 81 – 115, Burlington: Morgan Kaufmann, third edition ed., 2006.

[41] V. Sze and M. Budagavi, "High throughput cabac entropy coding in hevc," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1778–1791, 2012.

[42] D. Le Gall, "Mpeg: A video compression standard for multimedia applications," *Communications of the ACM*, vol. 34, no. 4, pp. 46–59, 1991.

[43] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[45] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[46] G. Hinton, N. Srivastava, and K. Swersky, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent,"

[47] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[48] F. Kamisli, "Block-based spatial prediction and transforms based on 2d markov processes for image and video compression," *IEEE Transactions on Image Processing*, vol. 24, pp. 1247–1260, April 2015.

[49] F. Kamisli, "Intra prediction based on markov process modeling of images," *IEEE Transactions on Image Processing*, vol. 22, pp. 3916–3925, Oct 2013.

[50] B. Girod, "Digital images and human vision," ch. What's Wrong with Mean-squared Error?, pp. 207–220, Cambridge, MA, USA: MIT Press, 1993.

[51] Z. Wang, A. C. Bovik, H. R. Sheikh, E. P. Simoncelli, *et al.*, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[52] Z. Wang, E. P. Simoncelli, and A. C. Bovik, "Multiscale structural similarity for image quality assessment," in *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers, 2003*, vol. 2, pp. 1398–1402 Vol.2, Nov 2003.

[53] J. Ballé, V. Laparra, and E. Simoncelli, "Density modeling of images using a generalized normalization transformation," 11 2015.

[54] J. Malo, I. Epifanio, R. Navarro, and E. P. Simoncelli, "Nonlinear image representation for efficient perceptual coding," *IEEE Transactions on Image Processing*, vol. 15, pp. 68–80, Jan 2006.

[55] J. R Cavanaugh, W. Bair, and J. Anthony Movshon, "Selectivity and spatial distribution of signals from the receptive field surround in macaque v1 neurons," *Journal of neurophysiology*, vol. 88, pp. 2547–56, 12 2002.

[56] M. Carandini and D. J. Heeger, "Normalization as a canonical neural computation," *Nature Reviews Neuroscience*, vol. 13, pp. 51–62, 2011.

[57] J. Malo, I. Epifanio, R. Navarro, and E. P. Simoncelli, "Nonlinear image representation for efficient perceptual coding," *IEEE Transactions on Image Processing*, vol. 15, pp. 68–80, Jan 2006.

[58] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015.

[59] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015.

[60] Y. Li, D. Liu, H. Li, L. Li, Z. Li, and F. Wu, "Learning a convolutional neural network for image compact-resolution," *IEEE Transactions on Image Processing*, vol. 28, pp. 1092–1107, March 2019.

[61] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[62] C. Doersch, "Tutorial on variational autoencoders," *ArXiv*, vol. abs/1606.05908, 2016.

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

[64] J. Portilla, V. Strela, M. J. Wainwright, and E. P. Simoncelli, "Image denoising using scale mixtures of gaussians in the wavelet domain," *IEEE Trans Image Processing*, vol. 12, no. 11, 2003.

[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[66] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

[67] A. van den Oord, N. Kalchbrenner, L. Espeholt, k. kavukcuoglu, O. Vinyals, and A. Graves, "Conditional image generation with pixelcnn decoders," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 4790–4798, Curran Associates, Inc., 2016.

[68] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks," in *Advances in Neural Information Processing Systems 21* (D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds.), pp. 545–552, Curran Associates, Inc., 2009.

[69] L. Theis and M. Bethge, "Generative image modeling using spatial lstms," in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 1927–1935, Curran Associates, Inc., 2015.

[70] S. Haykin, *Communication Systems*. Wiley Publishing, 5th ed., 2009.

[71] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[72] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, B. Lim, *et al.*, "Ntire 2017 challenge on single image super-resolution: Methods and results,"

in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, July 2017.

[73] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS Autodiff Workshop*, 2017.