DEEP IMAGE COMPRESSION WITH A UNIFIED SPATIAL AND CHANNEL CONTEXT AUTO-REGRESSIVE MODEL

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

BY

ALI SEFKAN ULUDAĞ

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

AUGUST 2022

Approval of the thesis:

DEEP IMAGE COMPRESSION WITH A UNIFIED SPATIAL AND CHANNEL CONTEXT AUTO-REGRESSIVE MODEL

submitted by **ALI SEFKAN ULUDAĞ** 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. Ahmet Oguz Akyuz
Computer Engineering, METU _____

Assist. Prof. Dr. Ahmed Hareedy
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: Ali Sefkan Uludağ

Signature :

ABSTRACT

DEEP IMAGE COMPRESSION WITH A UNIFIED SPATIAL AND CHANNEL CONTEXT AUTO-REGRESSIVE MODEL

Uludağ, Ali Sefkan
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Fatih Kamışlı

August 2022, 69 pages

Recently, variational auto-encoder-based compression models have gained much attention in learned image compression field due to their dimension reduction property. These models have three essential components which are encoder, decoder and entropy network. Encoder part of the VAE transforms a high dimensional space to a lower dimensional latent space, whereas the decoder serves as a reconstructing transformation. The entropy model is trained to generate the probability distributions of the latent representation to be used in arithmetic coding. End-to-end optimization is performed to minimize a rate-distortion loss composed of weighted sum of distortion loss of the decoder output and rate loss of the entropy model output. The VAE-based learned lossy image compression first introduced in Ballé (2016). Ballé (2018), improved this model by adapting a hyperprior network to increase the expression ability of spatial dependencies in the latent variable. Spatial auto-regressive and channel auto-regressive models are introduced in Minnen (2018) and Minnen (2020) respectively, to decrease the entropy of latent representation.

This thesis presents a lossy image compression model which builds upon the afore-

mentioned image compression architectures. It has been observed that spatial auto-regressive and channel auto-regressive models can be complementary and enhance the rate-distortion performance of the network. The channel auto-regressive model splits the latent representation along the channel dimension. It eliminates redundancies between channel slices by conditioning the slice being decoded to previously decoded slices. Spatial auto-regressive model captures the spatial correlations within a slice. Furthermore, it has been observed that the cumbersome spatial auto-regressive model can be parallelized by dividing the latent variables into smaller patches. Elements with the same index across patches are processed together, allowing parallel computation.

Keywords: image compression, deep learning, transform coding, auto-encoder

ÖZ

BİRLEŞTİRİLMİŞ UZAYSAL VE KANAL İÇERİK ÖZBAĞLANIM MODELİ İLE DERİN GÖRÜNTÜ SIKIŞTIRMA

Uludağ, Ali Sefkan
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Fatih Kamışlı

Ağustos 2022 , 69 sayfa

Yakın zamanda, boyut azaltma özelliğinden dolayı, değişimsel oto-kodlayıcılar büyük bir ilgi toplamıştır. Bu tip modellerin üç ana parçası bulunmaktadır. Bunlar kodlayıcı, kod çözücü ve entropi ağlarıdır. Kodlayıcı kısmı, görüntüyü yüksek boyutlara sahip olan girdi uzayından düşük boyutlara sahip olan örtük uzaya dönüştürür. Bunun yanında, kod çözücü ise tekrardan boyut yükselterek örtük uzaydan görüntü uzayına dönüştürme görevindedir. Entropi modeli ise aritmetik kodlamada kullanılmak üzere örtük uzaydaki elemanların olasılık dağılımlarını üretmek için eğitilmektedir. Uçtan uca optimizasyon, kod çözücünün çıktısının bozulma kaybı ve entropi ağının çıktısının hız kaybının ağırlıklı toplamı ile gerçekleştirilmektedir. Değişimsel oto-kodlayıcı temelli öğrenilmiş görüntü sıkıştırma teknikleri ilk Ballé (2016) ile tanıtılmıştır. Ballé (2018) örtük uzaydaki uzaysal koşullulukları daha iyi ifade edebilmek için bir hiper-evvel ağı uyarlayarak geliştirmiştir. Uzaysal ve kanalsal içerik modelleri Minnen (2018) ve Minnen (2020) ile eklenmiştir. Bu özbağlanımlı modeller entropiyi azaltmak için uygulanmıştır.

Bu tez bahsi geçen görüntü sıkıştırma mimarileri üzerine inşa edilen bir görüntü sıkıştırma modeli sunmaktadır. Her iki özbağlanımlı model de bu çalışmada kullanılmıştır. Uzaysal özbağlanımlı ve kanalsal özbağlanımlı modellerin birbirini tamamlayıcı özellikte oldukları ve birlikte kullanımlarının hız-bozulma performanını arttırdığı gözlenmiştir. Kanalsal özbağlanımlı model örtük uzaydaki temsili kanal boyutu boyunca daha küçük kanala sahip olan parçalara böler. Bu model, işlenmekte olan kanal parçasını daha önce işlenen kanal parçalarına koşullayarak kanallar arasındaki fazlalıkları giderir. Uzaysal özbağlanımlı model ise bir kanal parçası içerisindeki uzaysal korelasyonları yakalar. Bunlara ilaveten, örtük değişken daha küçük parçalara ayrılarak uzaysal özbağlanımlı modelin parelelleştirilmesinin mümkün olduğu gözlenmiştir.

Anahtar Kelimeler: görüntü sıkıştırma, derin öğrenme, dönüşüm kodlaması, oto-kodlayıcı

To the days that have been sacrificed for the days that are yet to come.

ACKNOWLEDGMENTS

I want to express my deepest gratitude to my advisor Assoc. Prof. Dr. Fatih Kamışlı for his guidance. He never hesitated to help me and share his knowledge whenever I needed. I greatly appreciate his belief in me.

I thank Onur Yek for giving me motivation to go on whenever I felt exhausted. Without him, these past years would be very tough. I am thankful to all my friends for being there when I needed. I know that we will continue to carry each other forward throughout our lives. I thank Lizge Gül Uludağ for being the best sister on earth. She never ceases to lend her support when I am in need. Finally, I am grateful to my mother Nadire Turgut for everything she has given to me.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

MS SSIM	Multi-Scale Structural Similarity Index Measure
JPEG	Joint Photographic Experts Group
HEVC	High Efficiency Video Coding
ANN	Artificial Neural Network
MLP	Multi-layer Perceptron
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation.
CNN	Convolutional Neural Network
GDN	Generalized Divisive Normalization
PMF	Probability Mass Function
CDF	Cumulative Density Function
VAE	Variational Auto-Encoder
SGD	Stochastic Gradient Descent
PSNR	Peak Signal-to-Noise Ratio
MSE	Mean Square Error
ICA-MG	Independent Component Analysis Marginal Gaussianization
RG	Radial Gaussianization

CHAPTER 1

INTRODUCTION

1.1 Motivation

Image data constitute a large portion of the data available today, thanks to the social media platforms that incorporate everyday lives of a large portion of the population. Thus, image compression is a crucial area of research. Image compression is the process of transforming a digital image to store it with less number of bits than the original image. Visually meaningful images have strong spatial and spectral dependencies. As correlation between neighboring pixels increases, the bits required to represent these pixels decrease, greatly reducing the demand for transmission bandwidth and storage space.

Conventional methods adapt linear transforms and predictive algorithms to benefit from natural properties of image data. For example, JPEG [3] is based on discrete cosine transform, a transform that converts two dimensional images to frequency domain features. HEVC [4], one of the most sophisticated video and still image compression algorithm, adaptively chooses the right tools such as block size, prediction mode or transform to encode and decode according to spatial and spectral properties of each block of the input. Unfortunately, it has become more and more effortful to develop new methods to increase the coding efficiency. With increasing image quality, demand for resources increases. Soon, conventional methods will be either terribly inefficient or highly costly to be used in a practical scenario. Thus, it is necessary to explore learning based image compression methods.

1.2 Image Compression Overview

There are two types of image compression methods namely lossless and lossy image compression. Lossless image compression aims to compress the image and restore it without any distortion or loss of information. By utilizing spatial and spectral dependencies, it is possible to obtain an identical image with lower size. Lossless image compression is mainly used in medical imaging, scientific imaging and for artistic purposes, where loss of quality is intolerable.

Lossy image compression methods can compress the image to less number of bits in the expense of information. Lossy image compression gained a broader application since for most practical areas limited loss of information is acceptable. There are information present in an image that cannot be perceived by the viewer. Filtering these information reduces the bits required to encode the image without loss of visual quality. Allowing more information loss enables the compression algorithm to compress the image even further in exchange for quality. The trade-off between quality and size is expressed with rate distortion criterion. Typically, as rate increases, distortion decreases and vice versa. The aim of lossy compression techniques are to reduce the bits while keeping the distortion keeping the distortion at a desired or given level. Current image compression methods use complex algorithms to increase compression factor. However, as imaging and displaying technologies advances, the need for more comprehensive algorithms grows.

1.3 Learning Based Image Compression

In recent years, success of the neural networks in computer vision has attracted the researchers to explore their abilities in low-level image processing problems such as denoising, edge extraction, super resolution and image sharpening. Image compression is not an exception. Many models have been proposed to work as an image compressor. Latest learned image compression models have been shown to exceed the performance of conventional methods.

Neural networks are universal approximators. By introducing non-linearities between

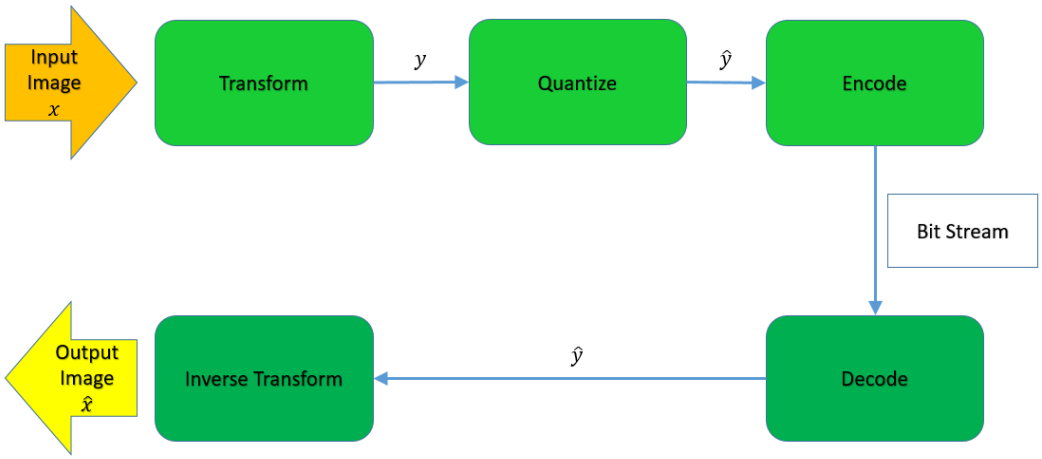


Figure 1.1: A simple block diagram explaining general image compression framework. Firstly, input image is projected onto a transform domain to obtain a compact representation. Quantization is applied to transformed image since encoder can only operate on integer values. Encoder translates the integer values into a bitstream. To restore the image, decoder translates back the integer values from input bitstream. Inverse of quantization and transformation is applied to the retrieved integer values.

linear transforms, it becomes possible to model any given function with neural networks. In the case of image compression, there are several functions to be modeled.

Analysis transform represented by 1.1 maps the input image $\boldsymbol{x}$ to a latent representation $\boldsymbol{y}$, which undergoes through quantization to be encoded. The quantized latent representation $\hat{\boldsymbol{y}}$ then transformed to output image $\hat{\boldsymbol{x}}$ by 1.3 transform.

$$g(x, \theta) = y \tag{1.1}$$

$$\hat{y} = Q(y) \tag{1.2}$$

$$\hat{x} = f(\hat{y}, \phi) \tag{1.3}$$

To approximate any function successfully, the weights of the neural network are optimized according to a designated criterion with stochastic gradient descent. In learned image compression, the optimization criterion is rate-distortion loss. Distortion is

the distance between the original image x and the reconstructed image $\hat{x}$ determined according to a distance metric as shown in 1.4. Rate is bounded from below by the entropy of the distribution of the quantized latent variable. The weighted sum is called the rate-distortion loss between the input and output image as given in Equation 1.6. The λ factor in Equation 1.6 determines the trade-off between quality and rate. Training the model with stochastic gradient descent or a variant optimization algorithm, it is possible to obtain a transformation that is able to compress the image.

$$D = d(x, \hat{x}) \tag{1.4}$$

$$R > H(\hat{y}) \tag{1.5}$$

$$L = R + \lambda D \tag{1.6}$$

Auto encoder based models such as [5] reduces the spatial dimension and obtains a distribution in latent space. The obtained probability distribution is then used to encode the latent tensor with a arithmetic coder. A factorized probability model is jointly trained with the auto-encoder in [5]. The factorized model learns the optimum sampling points to divide the distribution into equally probable regions. The factorized model assumes that there are no dependency between latent elements. [6] builds upon [5] and extends the message with extra information about spatial correlations between elements of latent variables to obtain a conditional Gaussian scale mixture model. The extra information, which is called hyperprior, incorporates much smaller space compared to the latent tensor. However, [1] and [2] introduces auto-regressive models in addition to the hyperprior for information redundancy reduction. Auto-regressive components prevents unnecessary repetition of coded information.

1.4 Contribution

The main contribution of this thesis is a lossy image compression model with an auto-regressive context model which combine spatial and channel auto-regressive models presented in [1] and [2] respectively. It has been observed that these two approaches

are complementary and can be used in combination to increase rate-distortion performance. However, the enhanced results come in the expense of computation latency. There is another approach under development which aims to decrease encoding and decoding time with minimal performance trade-off. To achieve this, the latent representation is divided into patches. Spatial auto-regressive model processes each element with same index in patches in parallel. First results show that a faster model can be obtained with little performance loss. Nonetheless, further research is needed on the parallel context model.

1.5 Outline of the Thesis

First of all, preliminary information critical to the remaining chapters is delivered in 2. The chapter starts with a brief introduction to information theory in section 2.1. In this section, concepts about information entropy, lossless and lossy data compression are explained. Then, basic knowledge about artificial neural networks is presented in 2.3. This section covers topic about neural networks and their optimization. Also, information about convolutional neural networks and early implementations are given in 2.6.

Then, a literature review composed of works supporting this thesis is presented. Ideas and contributions of these works are summarized in 3. Most significant aspects of these publications are detailed in this chapter

The methodology and implementation of proposed lossy image compression model are discussed in 4. Overview of the model, quantization of latent variables, components of entropy network and arithmetic coding are described in detail. Lastly, the compression scheme of an image is given in 4.5.

Last chapter presents training setup, compression and performance results. The designated training and validation dataset and training schedule are given in this chapter. The results includes the resulting network's performance with comparisons with previous models. Visual representations of the latent space are also given.

CHAPTER 2

BACKGROUND INFORMATION

2.1 Information Entropy

In information theory, entropy is defined as a lower bound for rate of information produced by an information source ([7]). In a sense, it numerically represents the uncertainty of a random variable. Shannon stated in his paper [7] that such a metric shall meet certain requirements. These are given below.

1. Entropy should be continuous.

2. If all the outcomes of the process have equal probabilities, then entropy should be monotonic increasing function of the total number of outcomes.

3. If an outcome of a random process splits into two successive sub-outcomes, entropy of the original outcome should be the weighted some of the entropy of the two sub-outcomes. This property can be visualized with Figure 2.1.

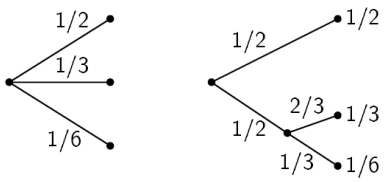


Figure 2.1: Visualization of third property

The only mathematical formula that satisfies all of the requirements is given in [7],

where n is the total number of outcomes of the random process X.

$$H(X) = -\sum_{i=1}^{n} p_i log(p_i) \tag{2.1}$$

To illustrate, entropy of a random process X with two outcomes with probabilities *p* and 1 - *p* is found with the equation in Equation 2.2, which is plotted as a function of p in Equation 2.2

$$H(X) = -plog(p) - (1-p)log(1-p) \tag{2.2}$$

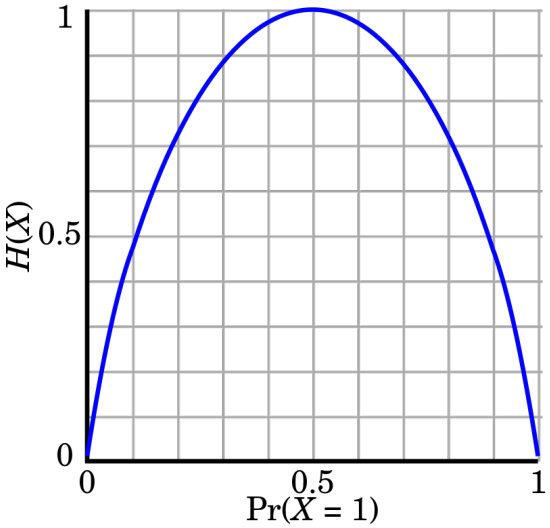


Figure 2.2: Entropy of a binary random process

This entropy definition possesses several properties which makes it a feasible measure of information. These can be listed as follows

1. H=0 implies that the outcome of the process is certain. In the example of two outcomes, if probability of one of the outcomes is 1, the entropy equals to 0.

2. Entropy of a random process is maximized when all of the outcomes have equal probabilities.

3. Entropy of a joint probability of two events is less than or equal to the sum of individual entropies of the events.

$$H(X,Y) \leq H(X) + H(Y) \tag{2.3}$$

4. It is possible to define a conditional entropy by using the conditional probability definition given in Equation 2.4.

$$p(y|x) = \frac{p(x,y)}{p(x)} \tag{2.4}$$

Applying the entropy formula, we obtain the relation given in Equation 2.5

$$H(Y|X) = H(X,Y) - H(X) \tag{2.5}$$

5. Inspection of the former properties leads to the conclusion that conditionality reduces entropy. This can be shown as Equation 2.6

$$H(X) + H(Y) \geq H(X,Y) = H(X) + H(Y|X) * H(Y) \geq H(Y|X) \tag{2.6}$$

This inequality indicates information of X can decrease the uncertainty of Y if they are dependent. Otherwise, it is unchanged.

Entropy is a key concept in data compression. It provides a lower bound for expected value of number of bits required to store or transmit information without any loss. Less bits are assigned to outcomes with higher probabilities, since these will occur more frequently. To illustrate, assume X is a random process in which a fair coin is tossed N times. It would require N bits to store the information of number of heads since

$$H(X) = N * (1/2 * log(1/2) + 1/2 * log(1/2))H(x) = N\ bits. \tag{2.7}$$

In another random process Y, the coin tossed is unfair and the probability of heads is p. Then, the required bits to express the outcome of Y would be

$$H(Y) = N * (p * log(p) + (1-p) * log(1-p))\ bits. \tag{2.8}$$

If p is 0.8, the expected value of bits required to store the outcome is bounded from below by approximately N*0.722 bits. If N=10, it would require only 8 bits on average to store the information.

2.2 Data Compression

Data compression is the reduction of the number of bits required to represent data. Data compression can decrease the occupied space, increase file transfer speed, and reduce storage hardware costs. Typically, there are two components working together to compress the data. These are encoder and decoder. Encoder transforms the information in order to represent it with less number of bits. Decoder is the inverse transform of encoder and retrieves the encoded information from the bits received. There are two types of data compression, namely lossless and lossy data compression. In lossless data compression, encoder benefit from statistical redundancies to compress the data. In this case, decoder is able to reconstruct the original data without any loss. In the case of lossy data compression, encoder eliminates some part of the data according to specific measures, resulting in an irreversible transformation. Decoder is able to approximate the data with limited loss. By using lossy compression, it is possible to reduce the size of the data even further.

2.2.1 Lossless Data Compression

There are several lossless data compression algorithms which rely on the information entropy. These types of algorithms adapt a code book in which symbols with less probability of occurrence are represented with higher bits. Huffmann coding and arithmetic coding are the two most prevalent ones.

2.2.1.1 Huffman Coding

Huffmann states in his [8] paper that it is possible to develop an optimum coding process which assigns variable-length codes to input characters correspondent to their frequency of occurrence. The most frequent character gets the smallest code and the least frequent character gets the largest code. The variable-length codes assigned to input characters are prefix codes, the code assigned to a character is not the prefix of any other code assigned to a separate character.

To produce a codebook for Huffmann coding, construction procedure of a tree is

provided in [8]. In the beginning, the characters are sorted according to their probabilities, or frequencies. Two characters with the smallest probabilities are assigned with codes with equal lengths, only differing in the last bit. These characters are the deepest leaves of the tree, which are combined to form an internal node. The probability of the new node equals to the sum of the leaves connected to it. Then, the new set of characters, including the newly formed internal node, are sorted again. The procedure is repeated until there is a single node, which is the root of the tree. To finalize the codebook generation, all of the branches are assigned with a number, 0 or 1. Branches traversed to reach a character leaf determines the code assigned to that character. The procedure is illustrated in Figure 2.3

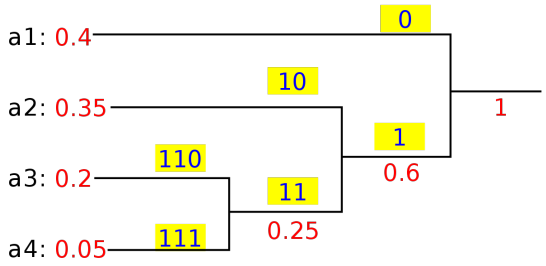


Figure 2.3: Example of Huffman Codebook generation for four symbols

2.2.1.2 Arithmetic Coding

Witten et al. defines in his [9] article arithmetic coding as a coding algorithm by which a message is represented by an interval of real numbers between 0 and 1. Longer messages are represented with a number chosen from a smaller interval, and the number of bits needed to code that interval grows. A model is needed beforehand to assign probabilities to each symbol. Efficiency of the arithmetic coding depends greatly on how close the distribution of symbols is modeled to the actual distribution. With each symbol of the message, extent of the interval is reduced according to symbol probabilities defined by the model. Symbols with higher probability reduce the range by less compared to the symbols with lower probabilities, effectively reducing the number of bits.

In [10], Langdon et al. describes the arithmetic coding algorithm as follows; primarily, the range is designated as the interval between 0 and 1, with 0 included. Each

symbol is assigned an interval, as shown in Figure 2.4. Larger intervals are assigned to symbols with larger probabilities. When first symbol arrives, the interval of the numbers are replaced from [0,1) to that character's interval. The range is divided with the previous proportions. The process is repeated until the whole message is received. A single number is obtained to represent the entire message.

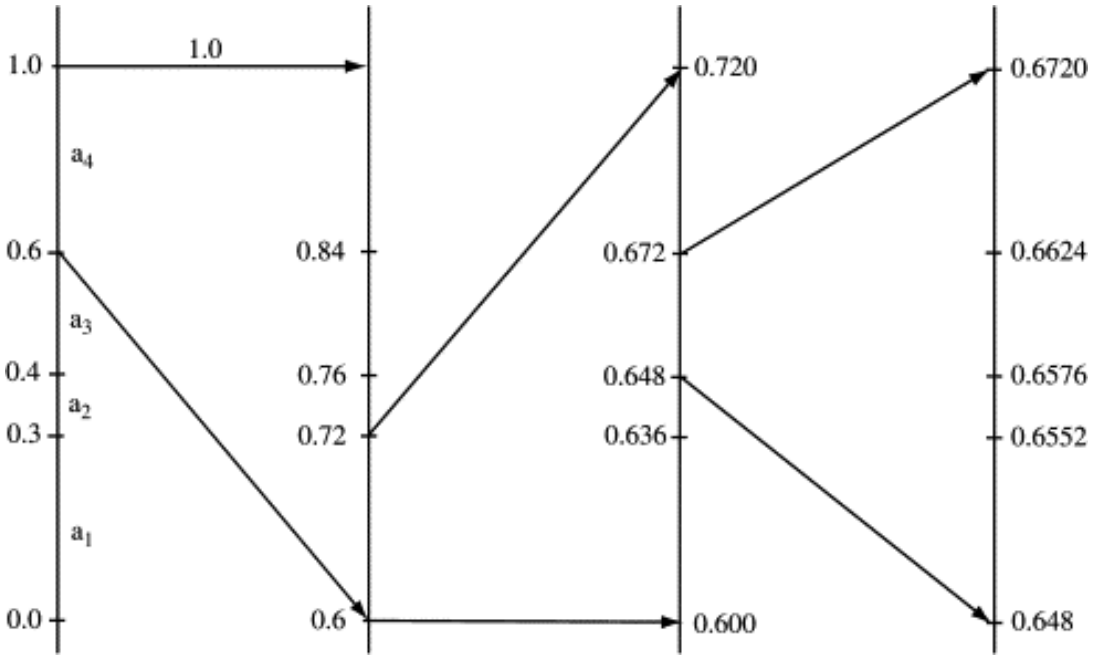


Figure 2.4: Arithmetic coding procedure

2.2.2 Lossy Data Compression

To describe the lossy data compression, it is essential to speak of rate-distortion theory. In [11] rate-distortion is described as the trade-off between source fidelity and coding rate. Either a target distortion or rate is chosen to optimize the coding. Rate-distortion theory has successfully put constrains on performance in terms of bit rate and quality for given types of sources. Most two common distortion metrics are Hamming distortion given in Equation 2.9 and squared error distortion Equation 2.10 below. These distortion metrics are favourable because they are easy to compute and widely applicable.

$$d(x, \hat{x}) = \begin{cases} 0 & x = \hat{x} \\ 1 & x \neq \hat{x} \end{cases} \tag{2.9}$$

$$d(x, \hat{x}) = (x - \hat{x})^2 \tag{2.10}$$

However, it is possible to develop more sophisticated distortion metrics considering the type of the signal and the final receiver, which is a human most of the time. For example, details of an image unperceivable by a human viewer are not needed to be transmitted, hence can be filtered without loss of perceptual quality. A similar case is exists for audio signals. It is possible to compress audio files greatly by taking into account the masking effect of human ear. It may be favourable to introduce limited perceivable distortion when there are heavier constraints on space and bandwidth. There is research going on to find a suitable distortion metric. Some examples are [12], [13] for image, [14] for video and [15] for audio signals.

2.3 Artificial Neural Networks

Artificial neural networks are a family of machine learning techniques which originated from biological neural system. The structure of ANNs resembles the neural networks, with nodes and connections mirroring the biological neurons and synapses in the brain. Similar to synapses in an animal brain, each connection can carry a signal to the attached nodes. Nodes, or artificial neurons, apply a non-linear transformation to the received signal and transmit it to other connected nodes in the next layer.

The simplest ANN structure is called perceptron, created by Rosenblatt and explained in [16]. A perceptron can be considered as a single-layer ANN, which processes one or more inputs, weighted according to their respective connections. These weighted inputs are summed and passed through a sign function, named activation function. In general, a bias term is added to the summation as well. It can be formulated as in Equation 2.11. An example of a perceptron is given in Figure 2.5. The perceptron is trained in a supervised fashion. Initially, random numbers are given to the weights of the perceptron. After output is calculated, the error is determined by subtracting

the output from the actual answer. If the output is correct, the error will be zero. Otherwise, it will be either -1 or +1. Weights are replaced by the summation of former weights and the error multiplied by a constant called learning rate and the input. The learning rate is a hyperparameter which determines the speed and sensitivity of learning procedure. Higher learning rate results in faster learning. However, the optimal point can be unreachable with a high learning rate. The learning algorithm is given in Algorithm 1.

$$\hat{y} = f(x) = \begin{cases} 1 & \sum_{i=1}^{n} \boldsymbol{W_i} * x_i + \boldsymbol{b} > 0 \\ 0 & otherwise \end{cases} \tag{2.11}$$

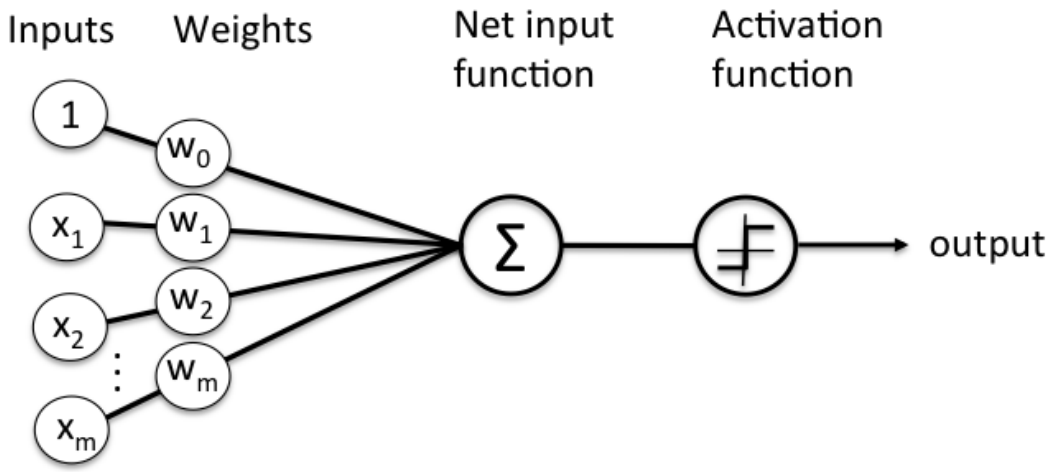


Figure 2.5: A simple perceptron

Even though perceptron provides the basic ideas of modern neural networks, its capabilities are limited. The idea of perceptron is extended to multi-layer perceptron (MLP) to obtain a higher computational capacity. Sets of perceptrons form a layer, which are inter-connected to form a feed-forward neural network. Input layer, hidden layer and output layer are three types of layers in a MLP model. The input layer's task is to take in the input signal like an image, whereas the output layer produces the output to accomplish the task, such as classification or recognition. There can be multiple hidden layers, which determines the capacity of the neural network. An

Algorithm 1 Learning algorithm of a perceptron

while True **do**

$\hat{y} = f(x)$

$error = y - \hat{y}$

for $0 < i \leq n$ **do**

$W_{i_{new}} = W_i + \lambda * error * x_i$

end for

end while

illustration of a MLP can be found in Figure 2.6. Weights of MLPs are updated via an optimization algorithm called stochastic gradient descent. The partial derivative of the error with respect to each weight is computed with an algorithm called back-propagation to apply gradient descent.

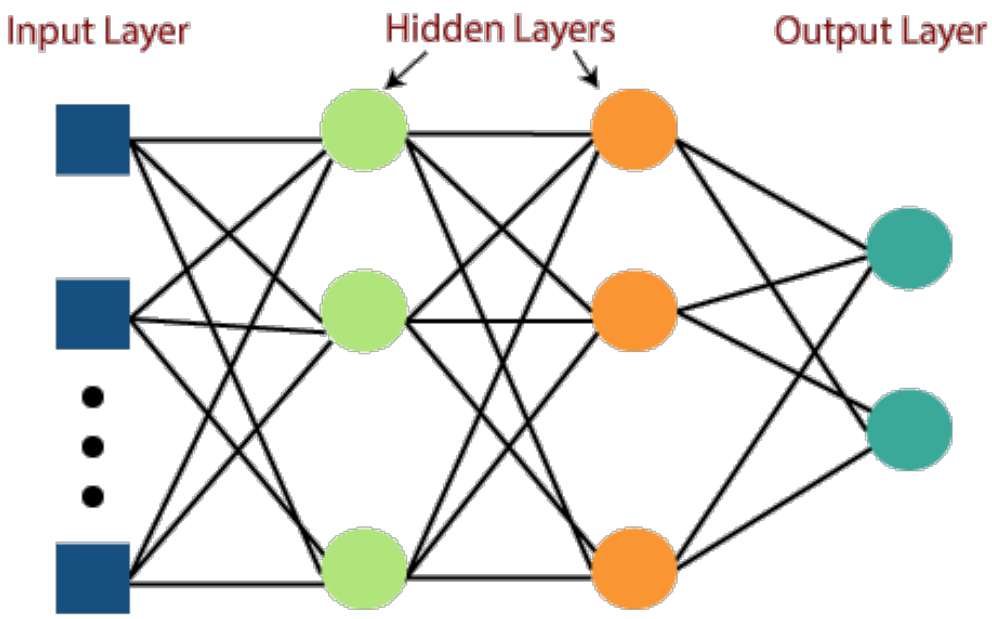


Figure 2.6: A multi-layer perceptron

2.4 Back-propagation

Back-propagation is an algorithm that is extensively adopted in training neural networks. Back-propagation was first applied to neural networks in [17]. Given an input-

output pair, back-propagation algorithm calculates derivatives at each layer with respect to the distance between the desired output and produced output, which is called loss of the model. Chain rule is exploited to accomplish this task. Partial gradient computed at a layer is reused in the computation of the gradient for the previous layer. This reverse flow of error gradient allows for efficient computation in contrast with computing the gradient of each layer separately. To minimize the loss, the gradients obtained from back-propagation are used by an optimization algorithm, such as stochastic gradient descent or a variant of it.

In the beginning, the weights of the neural network to be optimized are initialized randomly with small numbers. Hence, the first output of the network is completely random. Then, error between the actual result and the output is found according to a criterion related to the task. To make the model learn to recognize certain patterns to accomplish the task, the randomly initialized weights have to be updated in such a way that the error is minimized. To this end, gradients of the output layer's weights are calculated first. Next, these gradients are passed as inputs to the previous layer to take advantage of chain rule. By doing so, unnecessary duplicate computations are avoided and each gradient on the chain is reused. Forward and backward propagation are illustrated in Figure 2.7.

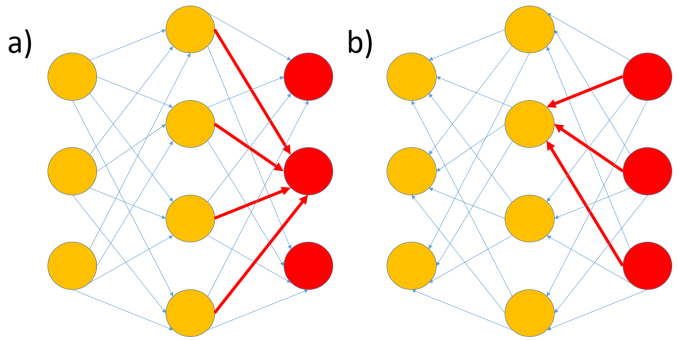


Figure 2.7: Illustrations of a) forward propagation b) backward propagation. In forward propagation, the result of a node in a layer is given as input to each node in the next layer. Multiple inputs are accumulated to produce an output. In backward propagation, the gradient is propagated from output to the input. The gradients received from nodes in the next layer are accumulated and passed on to the previous layer.

2.5 Optimization and Gradient Descent

Optimization is the process of solving quantitative mathematical expressions. Optimization includes finding most feasible values of some objective function given a defined input space. In the case of neural networks, it is the set of actions taken to minimize the loss between input and output. Optimization algorithms or methods are used to find the parameters θ of the neural network that minimize the loss $L(y, f(x, \theta))$. Optimizers are used to solve optimization problems by minimizing the loss function.

Gradient descent was first proposed by Cauchy (1847). Lemaréchal (2012) describes the Cauchy's idea as a means to minimize a non-negative continuous function of several variables as in Equation 2.12.

$$L = f(x, y, z, ...) \tag{2.12}$$

In machine learning, gradient descent is used to minimize the cost function which can be represented by

$$Q(w) = \frac{1}{n} \sum_{i=1}^{n} Q_i(w) \tag{2.13}$$

Each Q_i is the loss function associated with ith sample in the training dataset. Gradient descent can be applied to the cost function as given in Equation 2.14. ∇Q is the gradient of cost function, w_{old} and w_{new} are the parameters used in calculating the cost function and the new set of parameters after optimization respectively.

$$w_{new} = w_{old} - \lambda \nabla Q(w_{old}) = w_{old} - \frac{\lambda}{n} \sum_{i=1}^{n} \nabla Q_i(w_{old}) \tag{2.14}$$

In the case of neural networks, it not feasible to calculate the gradients of each summand function. Neural networks require a vast amount of samples to reach acceptable performance. Furthermore, neural network models are typically used to tackle non-convex problems, which cannot be represented in a closed form. Therefore, a

computationally efficient approach to gradient descent named stochastic gradient descent is utilized. Instead of finding the cost and its derivative of the whole dataset, a single sample or a subset of samples are computed at each step, which serves as a approximation to ∇Q. w_{new} is determined by the approximated gradient of $Q(w)$. For a single sample approximation, the stochastic gradient descent can be formulated as shown in 2.15

$$w_{new} = w_{old} - \lambda \nabla Q_i(w_{old}) \tag{2.15}$$

It is possible to find a more accurate approximation by taking into account a subset of samples, which are called "mini-batches". This approach to gradient descent optimization reduces the complexity greatly in the expanse of convergence rate. Together with back-propagation, stochastic gradient descent is a core algorithm in training artificial neural networks.

Another limitation of stochastic gradient descent is choosing a suitable value for learning rate. Larger values may cause the model to diverge. Setting the learning rate small leads to slower convergence. A simple approach to avoid this problem is to set a learning rate schedule. Learning rate schedule decays the learning rate at a certain rate in accordance to certain rules such as reducing the learning rate when a plateau is reached. There are certain modifications to stochastic gradient descent applied in address to this issue.

RMSProp is a variant of stochastic gradient descent which adapts a learning rate for each of the parameters. It was introduced in [18] A running average of gradients for each weight is kept to divide the learning rate to obtain an adaptive learning rate.

First, running average of means square is calculated as in 2.16. In this algorithm, another hyper-parameter is introduced which is named as decay rate. Decay rate γ determines how strongly will the past values of average means square will affect the current learning step which is denoted as t. Then, the weights are updated as shown in 2.17. The δ term that is in the denominator of 2.17 prevents the algorithm from becoming unstable.

$$r(w,t) = \gamma r(w,t-1) + (1-\gamma)(\nabla Q_i(w))^2 \tag{2.16}$$

$$w = w - \frac{\lambda}{\sqrt{r(w,t)} + \delta} \nabla Q_i(w) \tag{2.17}$$

Another optimization algorithm that is widely used in training artificial neural networks is Adam. It was first proposed in Kingma (2015). In Adam optimization, running average of second moments of the gradients are taken into account in addition to the gradients. Adam optimization is especially used in networks with large number of parameters because of its improved convergence speed.

2.6 Convolutional Neural Networks

Convolutional neural networks are a type of neural networks that have proven to be extremely successful in learning spatial hierarchies. Therefore, it is commonly used in tasks involving visual data. Apparent from its name, convolutional neural network layers use convolution operation to produce an output, as shown in 2.18. It should be noted that there is no reversal of signals as in conventional convolution operation since only the position of the weights learned will change. MLPs fall short in capturing the complex patterns of images. CNNs, on the other hand, successfully capture the spatial and temporal dependencies in visual data by applying learnable filters to an input.

$$x[m,n] * f[m,n] = \sum_{k=0}^{M} \sum_{l=0}^{N} x[k,l] f[k-m, l-n] \tag{2.18}$$

CNNs are in fact regularized MLPs, where number of parameters are reduced by reusing the weights. These weights are called in the context of convolution operation. Filters can be optimized to learn hierarchical features present in the data, progressively assembling low level features to form more complex patterns related to the task as shown in Figure 2.8. An illustration of a fully convolutional neural network is given in Figure 2.9.

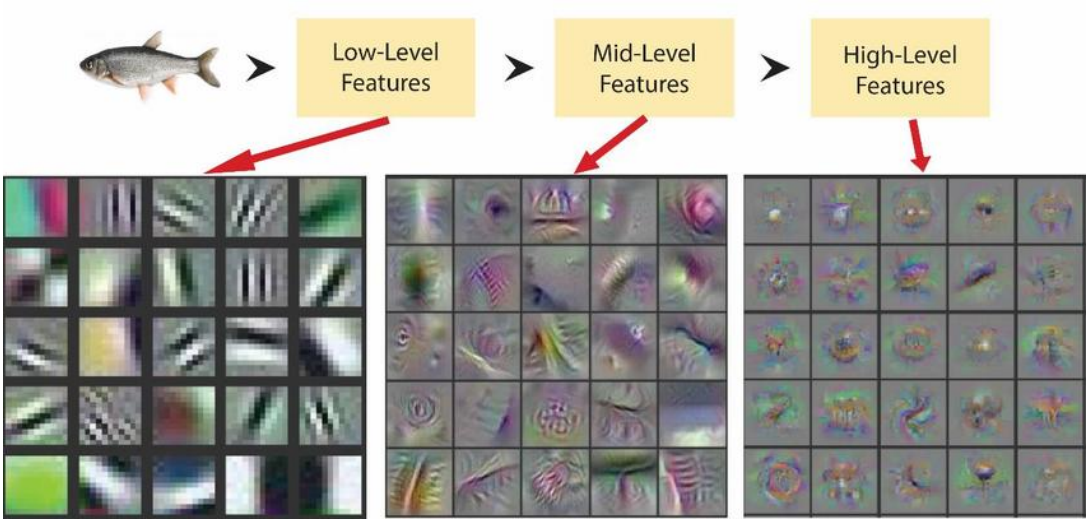


Figure 2.8: Hiearchy of features extracted from input image. Earlier layers extract low level information which has no meaningful structure when examined alone. These features are passed on to following layers to form higher level features. The image is taken from (Yosinski et al. 2015)

CNN is a biologically inspired algorithm. It is first devised to resemble the activity of visual cortex of animals. For example, research on cat's visual cortex in [19] and [20] explains the receptive field of a neuron in a cat's visual cortex. Each neighboring cell possess similar overlapping receptive fields.[21] presents research conducted on macaque visual cortex. Hubel and Wiesel demonstrate the organization of special neurons in the presence of visual stimuli. It was found that there were two groups of neurons one of which capture low level information such as vertical and horizontal lines and their orientation, while the other produces output if there is a signal in its receptive field.

First model to incorporate convolutions into neural networks is neocognitron proposed in Fukushima (1980). The structure of the network was inspired by the visual nervous system of vertebrates. The network involves two types of cells called S-cells, which are simple cells or lower order hypercomplex cells, and C-cells, complex cells or higher order hypercomplex cells. With repetition of feature-extraction by S-cells followed by positional activation by C-cells, local features extracted in lower stages are combined into more global features. Figure 2.10 shows the working principle of neocognitron. Convolution and downsampling operations, two basic operations that

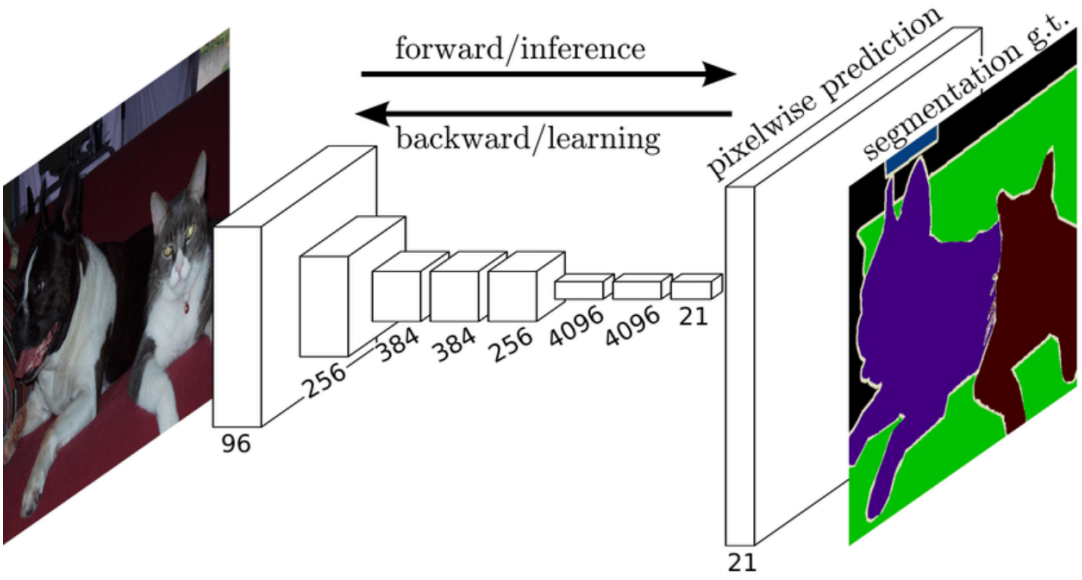


Figure 2.9: A fully convolutional neural network for semantic segmentation. The pixel predictions are compared with the ground truth to optimize the weights of filters in each layer.

are cornerstone of CNNs, have been first introduced with neocognitron.

In the context of CNN, receptive field is the area on the input image which determine the value of an element in a feature map. It is possible to expand the receptive field of subsequent layers with downsampling operation to increase efficiency of CNNs. There are two approaches for downsampling, the first one is using a pooling function such as max pooling or average pooling to decrease the spatial dimensions. Max pooling operations propagate the maximum value inside the kernel to the succeeding layer. On the other hand, average pooling produces the mean value inside the kernel as output. Both operations map values inside a kernel to a single value in a non-learnable manner. Hence, these operations limit the learning potential of CNNs. An explanotary visual is given in Figure 2.11.

Other method is to adapt strided convolution to perform downsampling via learnable filters. The filter will learn to transmit and enhance the important features, effectively increasing accuracy. Similarly, it is possible to apply transposed convolution for upsampling with learnable interpolation, as opposed to non-learnable interpolation methods such as bilinear interpolation. A toy example is given in Figure 2.12

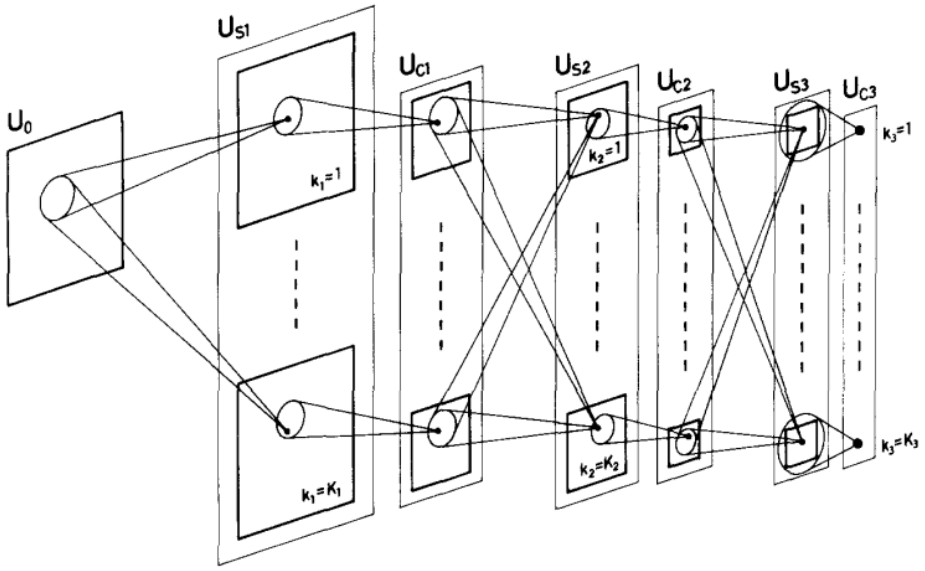
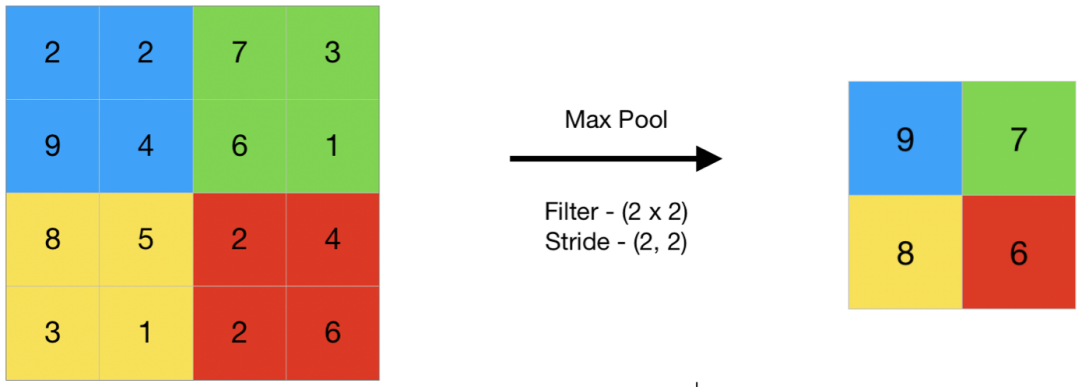
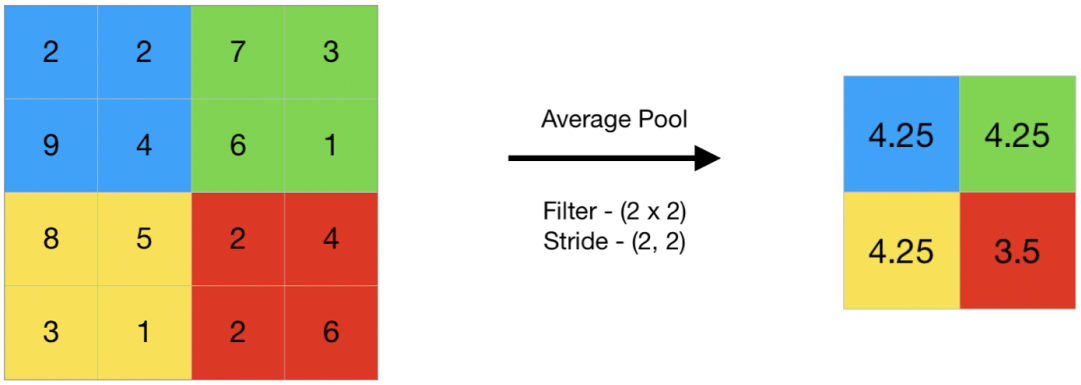


Figure 2.10: Neocognitron model developed by Fukushima(1980) et al.

with 1-D signals.



(a)



(b)

Figure 2.11: Examples of max pooling (a) and average pooling (b) operations.

$$\vec{x} * \vec{a} = X\vec{a} \qquad \vec{x} *^T \vec{a} = X^T\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix} \qquad \begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Figure 2.12: 1-D example of downsampling with convolution on left, upsampling with transposed convolution on right. On left, a filter $\vec{x}$ with length 3 is convolved with a zero-padded signal $\vec{a}$ with 4 elements. The transformation denoted with X is the matrix representation of convolution operation with a stride of 2. The dimension of output is the length of the input divided by stride. The next example shows a transposed convolution operation with the same kernel and a signal with length 2. Notice that transpose of X is applied to the input to obtain the result. Hence, this operation is named transposed convolution, and it increases the dimension of input. However, different padding rules should be applied to obtain proper upsampling.

CHAPTER 3

LITERATURE REVIEW

Image compression with nonlinear transform coding is possible by taking advantage of deep neural networks. There have been many works contributing to the development of learned image compression. Below, a number of publications in this area that have been used in this thesis are explained.

3.1 Generalized Divisive Normalization

Density estimation is crucial to obtain an optimum coding for latent representations. Ballé proposes a novel probability model for images in his 2016 paper [22] in the form of a non-linear function. Named generalized divisive normalization, this non-linear function is shown to successfully decorrelate and Gaussianize natural images. The proposed function is a generalization of divisive normalization, as the name suggests. This family of normalization functions are first used to model activation of cortical neurons ([23]), in which linear responses are divided by summed responses of neighboring neurons.

The generalized divisive normalization is formulated as a linear transformation followed by a divisive normalization operation. GDN is expressed as shown in Equation 3.1. In the equation, vectors $\boldsymbol{\beta}, \boldsymbol{\epsilon}$ and matrices $\boldsymbol{H}, \boldsymbol{\alpha}, \boldsymbol{\gamma}$ are parameters to be set or learned, depending on the desired divisive normalization operation. By choosing appropriate set of parameters, it is possible to express other transforms such as standard divisive normalization ([24]), a form of ICA-MG which is the first iteration of

Gaussianization algorithm described in [25] or RG [26].

$$\begin{aligned} \boldsymbol{z} &= \boldsymbol{H}\boldsymbol{x} \\ y_i &= \frac{z_i}{(\beta_i + \sum_j \gamma_{ij} |z_j|^{\alpha_{ij}})^{\epsilon_i}} \end{aligned} \tag{3.1}$$

An approximate inverse transform of generalized divisive normalization has also been proposed in [22]. Since finding a closed form of the inverse of a non-linear function is not feasible, the approximate inverse GDN transformation is obtained by using a fixed point iteration given in Equation 3.2. As initial point $z_i^{(0)}$ is chosen. In [27], it has been stated that using the first iteration is sufficient for image compression.

$$\begin{aligned} z_i^{(0)} &= sgn(y_i)(\gamma_{ii}^{\epsilon_i} |y_i|)^{\frac{1}{1-\alpha_{ii}\epsilon_i}} \\ z_i^{(n+1)} &= (\beta_i + \sum_j \gamma_{ij} |z_j^{(n)}|^{\alpha_{ij}})^{\epsilon_i} y_i \end{aligned} \tag{3.2}$$

The results from the paper [22] are given in Figure 3.1. It is clear that transformed image can be modeled with a Gaussian distribution. Since to define a Gaussian model, first and second moments are sufficient, GDN provides much flexibility and ease of implementation.

3.2 Nonlinear Transform Coding for Image Compression

A framework for nonlinear transform coding training has been proposed in [27], which generalizes the transform coding paradigm. To compress an image x, it is transformed to a latent representation y by using a differentiable function $y = g_a(x; \phi)$, where ϕ is the collection of learnable parameters of the analysis transform g_a. Scalar quantization is applied to the latent vector y, yielding $\hat{y}$. The quantized latent $\hat{y}$ is then transformed to the image space with synthesis transform g_s. Restoration of the image can be shown with $\hat{x} = g_s(\hat{y}, \theta)$, where $\hat{x}$ is the reconstructed image and θ represents the parameters of synthesis transform. Figure 3.2 summarizes the transform coding framework.

The neural transform coding model is optimized according to rate-distortion loss given in Equation 3.3. The rate is assessed by measuring the entropy H of the discrete probability distribution $P_{\hat{y}}$ of the quantized latents over a collection of images. To

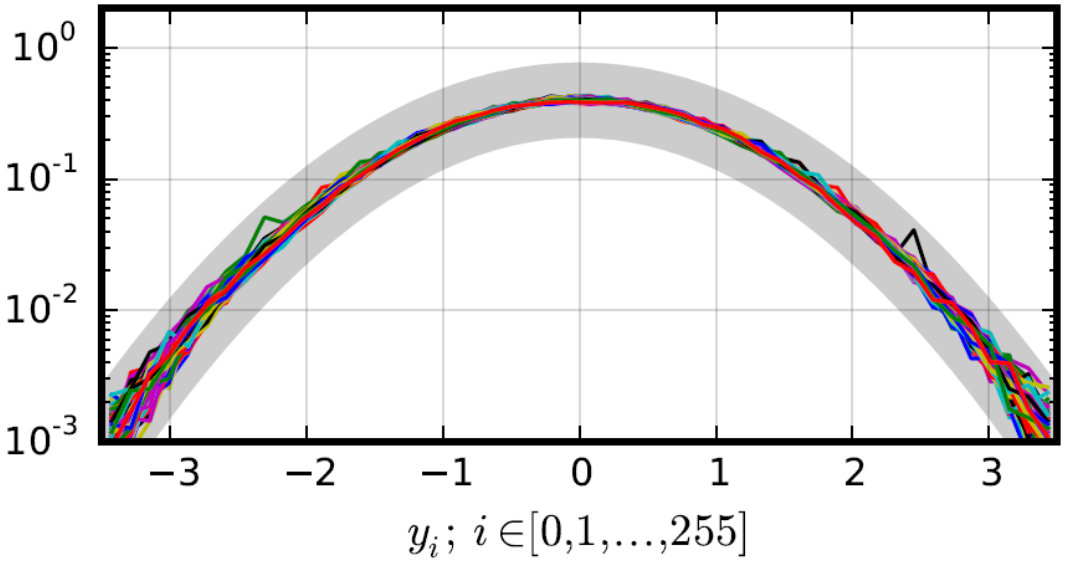


Figure 3.1: Histogram of GDN transformed image and Gaussian distribution. The grey area shows a Gaussian distribution. The y-axis holds the number of elements, x-axis shows the values. The lines are the dimensions of output of GDN transform y. The histogram proves that GDN successfully Gaussianize the signal.

find the probability mass function $P_{\hat{y}}$, the integral of probability density function p_y is taken over the quantization bin. In the case of uniform quantization where each bin has size 1, the integral can be computed with Equation 3.4. Distortion is denoted by $d(x, \hat{x})$ in Equation 3.3, where $d(.,.)$ is a suitable distortion metric. Even though a perceptually accurate metric would be more successful in optimization for visually appealing results, Euclidean distance is widely used because of its universality and ease of use. Both terms of the summation are expected values determined over a set of images, namely training dataset.

$$L = H[P_{\hat{y}}] + \lambda\, \mathbb{E}[d(x, \hat{x})] \tag{3.3}$$

$$P_{\hat{y}}(n) = \int_{n-1/2}^{n+1/2} p_y(t)dt \text{ , for all } n \in \mathbb{Z} \tag{3.4}$$

End-to-end optimization of non-linear transform coding framework suffers from zero gradient of quantization function, which prohibits gradient descent optimization. In

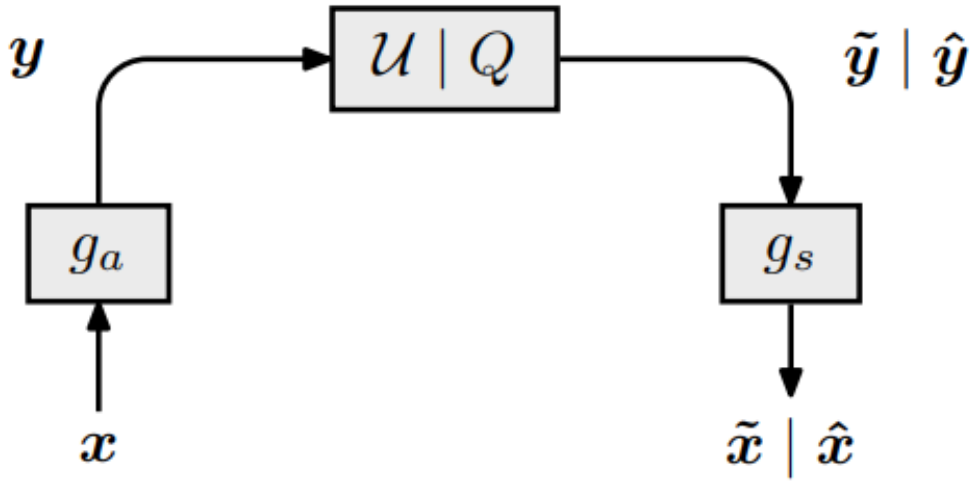


Figure 3.2: Diagram of nonlinear transform coding framework.

[5], this issue has been resolved by replacing the quantization with a additive uniform noise during training. The uniform noise provides a continuous approximation of quantization function, allowing optimization via stochastic gradient descent. Moreover, the property given in Equation 3.6 shows that differential entropy of $\tilde{y}$ can be used to approximate entropy of y.

$$\tilde{y} = y + \delta y \tag{3.5}$$

$$p_{\tilde{y}}(n) = P_{\hat{y}} \qquad \text{for all } n \in \mathbb{Z}^M \tag{3.6}$$

Combining all of the definitions mentioned above, the loss function can be expressed in form given in Equation 3.7.

$$L = \mathop{\mathbb{E}}_{x,\delta y}[-\sum_i \log_2 p_{\tilde{y}_i}(g_a(x;\phi) + \delta y; \psi^{(i)}) + \lambda d(g_s(g_a(x;\phi) + \delta y; \theta), x)] \tag{3.7}$$

In Equation 3.7, each $\psi^{(i)}$ is the collection of parameters of the probability model with i showing the channel number of latent variable. Every $\psi^{(i)}$ is jointly optimized

with θ and ϕ to find an optimum estimation to p_{y_i}. In [5], p_{y_i} is modeled with a factorized prior to reduce model error. Each marginal p_{y_i} is represented as a piecewise linear function sampled with a certain number points per unit interval. The parameter vector $\psi^{(i)}$ represents the value of p_{y_i} at these sampling points. The model is trained to minimize the negative expected log likelihood function given in Equation 3.8.

$$L_\psi = -\mathop{\mathbb{E}}_{\tilde{y}} \sum_i \log(p_{y_i}(\tilde{y}_i; \psi^{(i)})) \tag{3.8}$$

Ballé (2017) provides architectures for analysis and synthesis transform in the form of learnable filters. Analysis transform g_a consists of three layers. Convolution, downsampling and divisive normalization operations are applied in order at each layer. To reduce dimension of the input image with consequent convolution layers, strided convolution with stride of 2 is utilized. Strided convolution can be formulated as shown in Equation 3.9 where T is stride value, $w_i^{(r)}$ is the ith channel of output of convolution at layer r, $u_j^{(r)}$ is the jth channel of input of layer r and $h_{ij}^{(r)}$ is the jth channel of ith filter of layer r. Strided convolution allows the downsampling operation to be optimized in a way that the loss of information is minimized.

$$w_i^{(r)}[m,n] = \sum_j \sum_{k=0}^{M} \sum_{l=0}^{N} h_{ij}^{(r)}[k,l] u_j^{(r)}[mT-k, nT-l] + c_i^{(r)} \tag{3.9}$$

The generalized divisive normalization described in Equation 3.1, a Gaussainizing non-linear function, is used as activation function in the analysis transform with $\alpha = 2$ and $\epsilon = 1/2$, the function is in the form given in Equation 3.10, where γ and β are learnable parameters, and $x_i^{(r+1)}[m,n]$ is the output of the current layer and input of the next layer.

$$u_i^{(r+1)}[m,n] = \frac{w_i^{(r)}[m,n]}{(\beta_i^{(r)} + \sum_j \gamma_{ij}^{(r)} (w_j^{(r)}[m,n])^2)^{1/2}} \tag{3.10}$$

The collection of parameters h, c, β and γ are represented with ϕ in the loss function and are learned during training.

On the other hand, the synthesis transform g_s consist of reverse operations in reverse

order, approximation of inverse of generalized divisive normalization followed by upsampling transposed convolution. The inverse of generalized divisive normalization is implemented as in Equation 3.11.

$$\hat{w}_i^{(r)}[m,n] = \hat{u}_i^{(r)}[m,n](\hat{\beta}_i^{(r)} + \sum_j \hat{\gamma}_{ij}^{(r)}(\hat{u}_j^{(r)}[m,n])^2)^{1/2} \tag{3.11}$$

Like in the analysis transform, the collection of parameters $\hat{h}, \hat{c}, \hat{\beta}$ and $\hat{\gamma}$ are represented with θ in the loss function and are learned during training. For such a large set of parameters, convergence of SGD will be very slow. For this reason, Adam optimizer is used in [5] to minimize the cost function expressed as Equation 3.7.

3.3 Variational Generative Image Models

The loss function given in Equation 3.7 is close to the loss definition of generative image models, more specifically variatonal autoencoders as in [28]. Variational Bayesian inference aims to find a posterior $p_{y|x}(y|x)$ from a set of observations of a random variable x and a likelihood model $p_{x|y}(x|y)$. The method described in [28] consists of approximating this posterior with a density $q(y|x)$, by minimizing the Kullback–Leibler divergence between the true posterior $p_{y|x}(y|x)$ and approximated posterior $q(y|x)$. This can be formulated as follows

$$\begin{aligned} D_{KL}[q||p_{y|x}] &= \mathop{\mathbb{E}}_{q} \log q(y|x) - \mathop{\mathbb{E}}_{q} \log p_{y|x}(y|x) \\ &= \mathop{\mathbb{E}}_{q} \log q(y|x) - \mathop{\mathbb{E}}_{q} \log p_{x|y}(x|y) - \mathop{\mathbb{E}}_{q} \log p_y(y) \end{aligned} \tag{3.12}$$

which is equivalent to the relaxed rate-distortion loss given in Equation 3.7 when mean-square error is used as distortion metric and the likelihood model is defined as a Gaussian distribution. Thus, the elements in Equation 3.12 becomes

$$p_{x|\tilde{y}}(x|\tilde{y};\lambda,\theta) = N(x, g_s(\tilde{y};\theta), (2\lambda)^{-1}\mathbf{1}) \tag{3.13}$$

$$p_{\tilde{y}}(\tilde{y};\psi^{(0)},\psi^{(1)},...) = \prod_i p_{\tilde{y}_i}(\tilde{y}_i;\psi^{(i)}) \tag{3.14}$$

$$q_{\tilde{y}|x}(\tilde{y}|x;\phi) = \prod_i U(\tilde{y}_i; y_i, 1) \tag{3.15}$$

where $U(\tilde{y}_i; y_i, 1)$ is the uniform probability density on the unit interval around the center y_i, which is found by analysis transform. Inspecting Equation 3.12 in this context, the first element $\mathbb{E}_q \log q(y|x)$ is constant and can be eliminated from loss function to be optimized. The second element $-\mathbb{E}_q \log p_{x|y}(x|y)$ expresses the distortion, and the final element $-\mathbb{E}_q \log p_y(y)$ is the rate. Even though there are differences, the training process of the network can be inspected under the hood of variational autoencoders. It should be noted that the probability model of $\tilde{y}$, $p_{\tilde{y}}(\tilde{y}; \psi^{(0)}, \psi^{(1)}, ...)$ is also called a fully factorized model.

3.4 Variational Image Compression with a Hyperprior

The work presented in [5] present promising results, yet it is far from optimal. The distribution obtained by a fully factorized model as in Equation 3.14 proves to be a coarse approximation of the true distribution. The network proposed by [5] is extended in [6]. Balle et al. incorporates a side information to increase the performance of the network. Conventional compression methods append a side information to the encoded image to increase the efficiency. Typically, this side information has a very small size compared to the gain it provides. In HEVC [4], adaptive block partitioning is used to increase the coding efficiency. The sizes of these blocks are sent as side information to the decoder, so that correct decoding can be achieved. In the method described in [6], the side information to be sent is extracted from latent representation through a side network called hyperprior network, as illustrated in Figure 3.3. The model incorporates a hyperprior to effectively capture spatial dependencies in the latent representation. The output of the hyper-analysis transform serve as a side information to be sent with the encoded latent representation.

The fully factorized model can be inefficient in capturing spatial dependencies present in the latent representation. To eliminate these dependencies, a hyperprior latent $\tilde{z}$ is introduced. $\tilde{z}$ is a random variable obtained by adding uniform noise to hyper-latent z, the output of hyper analysis transform given in Equation 3.16. The latent with

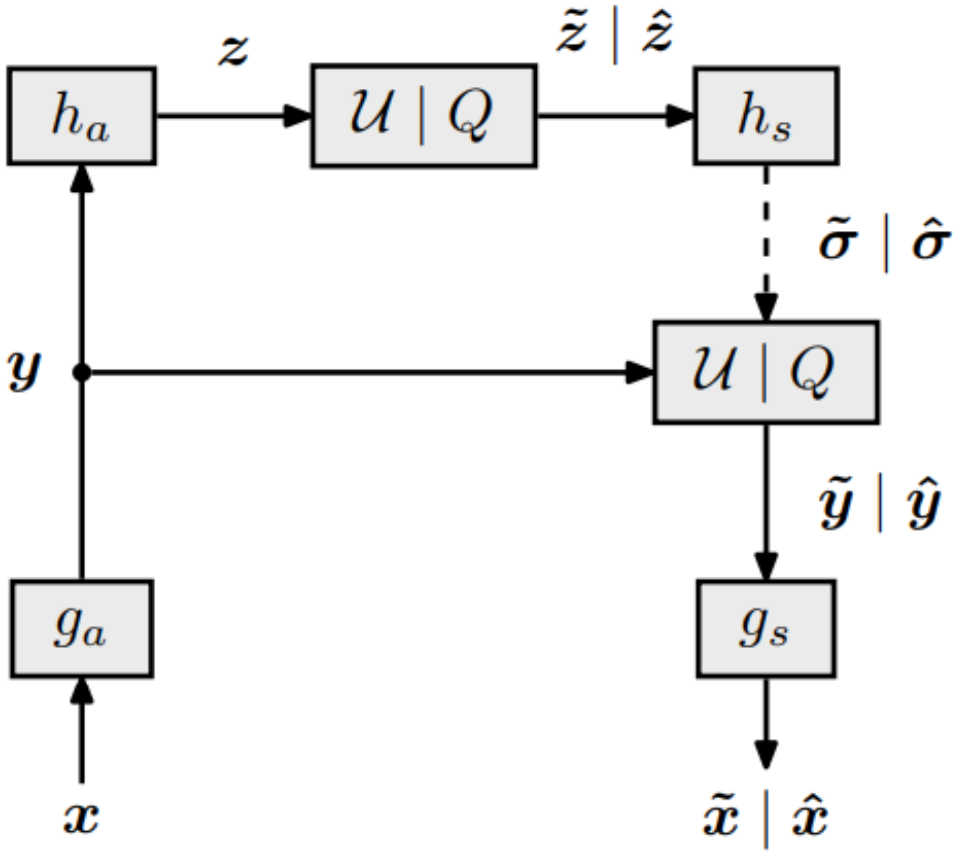


Figure 3.3: Diagram of variational image compression with a hyperprior model.

additive uniform noise $\tilde{y}_i$ is modeled as a zero mean Gaussian with scale σ_i^2. The scale is predicted by applying hyper-synthesis transform as shown in Equation 3.17. ϕ_h and θ_h represents the parameters of hyper-analysis and hyper-synthesis transforms respectively. The probability of latent $\tilde{y}$ can be modeled as in Equation 3.18.

$$z = h_a(y, \phi_h) \tag{3.16}$$

$$\sigma_i^2 = h_s(\tilde{z}; \theta_h) \tag{3.17}$$

$$p_{\tilde{y}|\tilde{z}}(\tilde{y}|\tilde{z}, \theta_h) = \prod_i (N(0, \sigma_i^2) * U(-1/2, 1/2))(\tilde{y}_i) \tag{3.18}$$

A joint approximated posterior can be written as given in Equation 3.19. Ballé et al. suggested hyperprior method to determine the scales with the intuition that latent variable possesses information about its probability distribution. With zero-mean Gaussian distribution assumption, the scales determine the conditional distribution of the latent variable. In this model, $\tilde{z}$ is modeled with a fully factorized density model, which was previously used for $\tilde{y}$ in Equation 3.14. The KL divergence previously formulated as Equation 3.12, is now in the form in Equation 3.20. In Equation 3.20 the last two elements represent entropy or rate, the second element is the distortion. The first element equals to zero since $q(\tilde{y}, \tilde{z}|x)$ is the product of two uniform densities with unit width as shown in Equation 3.19.

$$q(\tilde{y}, \tilde{z}|x, \phi_g, \phi_h) = \prod_i U(\tilde{y}_i|y_i - 1/2, y_i + 1/2). \prod_j U(\tilde{z}_j|z_j - 1/2, z_j + 1/2) \quad (3.19)$$

$$D_{KL}[q||p_{\tilde{y},\tilde{z}|x}] = \mathop{\mathbb{E}}_{q}[\log q(\tilde{y}, \tilde{z}|x) - \log p_{x|\tilde{y}}(x|\tilde{y}) - \log p_{\tilde{y}|\tilde{z}}(\tilde{y}|\tilde{z}) - \log p_{\tilde{z}}(\tilde{z})] \quad (3.20)$$

The fundamental observation of this work is that $\tilde{z}$ can be appended to the encoded latent as side information, which allows the decoder to adapt a conditional probability model. Hence, the probability model becomes adaptive to the image, getting closer to the true distribution and increasing coding efficiency.

3.5 Autoregressive Learned Image Compression

Minnen et al. provides an inspection of autoregressive and hyperprior models and their impact on image compression in [1]. Even though autoregressive models impose heavy computational burden on the model during the decoding process, it is found that they can greatly increase the compression performance. [1] suggests that autoregressive entropy model and hyperprior network are compatible and their combined usage can predict the spatial dependencies present in the image more successfully than the model described in [6]. Furthermore, the model adapts a Gaussian model with non-zero mean values which is determined together with scale by hyper-synthesis transform, as observed in Equation 3.21. The zero mean Gaussian scale mixture model

given in Equation 3.18 is replaced with non-zero mean Gaussian model shown in Equation 3.22.

$$\mu_i, \sigma_i^2 = h_s(\tilde{z}; \theta_h) \tag{3.21}$$

$$p_{\tilde{y}|\tilde{z}}(\tilde{y}|\tilde{z}, \theta_h) = \prod_i (N(\mu_i, \sigma_i^2) * U(-1/2, 1/2))(\tilde{y}_i) \tag{3.22}$$

A model is called autoregressive when its current output depends on the previous outputs. In the context of image compression, this means no pixel or block of pixels, in the case of block-based compression, can be decoded before the prior pixels are decoded first. Autoregressive models provides great predictive coding performance since there are strong spatial dependencies in an image, in the expense of decoding time. The latency of decoding procedure results from sequential processing of each element. In other words, operations conducted to find mean and scale of the latent feature map cannot be done in parallel since each element is conditioned on previous elements.

Even though autoregressive models can be conditioned on the whole set of previous outputs theoretically, in practice this is not feasible. Hence, autoregressive model is implemented as a masked convolution in [1], as in PixelCNN [29]. The kernel size of masked convolution is chosen as five. An overview of whole model can be seen in Figure 3.4.

3.6 Channel Autoregressive Model

The work shown in [1] take advantage of a spatially autoregressive entropy model, while Minnen et al. [2] takes a different approach at defining backward conditioning. To increase the parallel processing capabilities of the model, the entropy model is defined as a channel-wise autoregressive model, where the latent channels are divided into slices. These slices are decoded sequentially and each slice is conditioned on the previously decoded slices. This method greatly minimizes the need for serial processing compared to the spatial autoregressive model where each pixel has to be

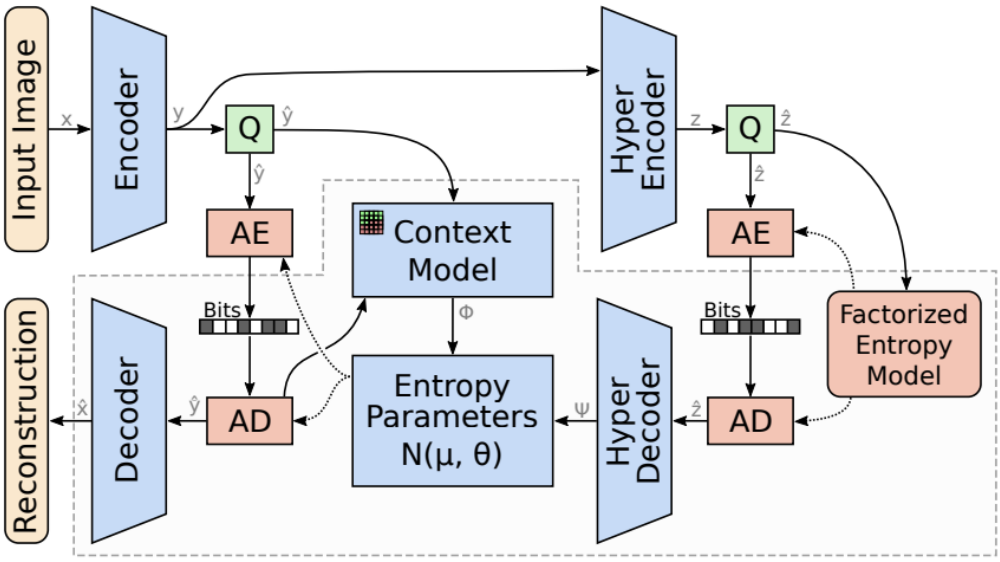


Figure 3.4: Block schema of the model presented in [1].

decoded sequentially. The work presented in [2] is built upon [6], the hyperprior network. Also, the hyper-synthesis transform is duplicated to predict mean and scale separately.

There is one more improvement to the architecture in [2] which is called latent residual prediction. It is designed to predict the information loss during the quantization process, and augment the decoded latent with this information. There are two purposes of latent residual prediction. The first one is to reduce the distortion of the synthesis transform g_s, the other is to increase the efficiency of channel conditioning as more channels augmented with latent residual prediction are taken into account while decoding the next slice.

As an alternative to regular training with noise regimen used in earlier models, Minnen et al. [2] propose to replace the additive noise with rounding whenever the latent is passed onto a synthesis transform. The training of entropy model is carried out in the same manner. This has been shown to improve the rate-distortion performance of the network. Visualization of the model can be found in Figure 3.5.

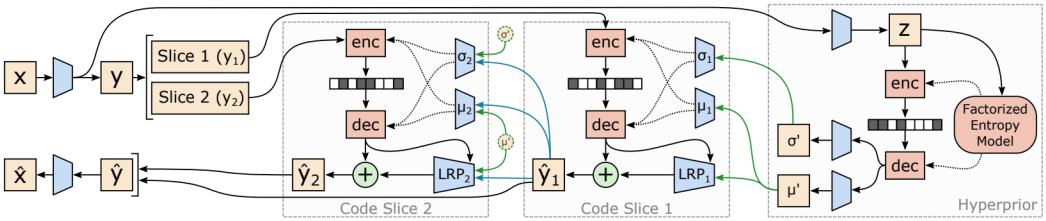


Figure 3.5: Block schema of model presented in [2].

CHAPTER 4

PROPOSED METHOD

In learned image compression with variational auto-encoder architecture, the input image is projected into a latent space which has lower spatial dimension as shown in [5]. The latent representations are quantized and encoded losslessly into a bitstream with arithmetic encoder. To be able to encode with the least number of bits, the joint distribution of the latent elements should be modeled as accurately as possible. Balle et al. models each element in the latent representation as independent random variables and uses a factorized entropy model. However, [6], [1] and [2] show that there are strong dependencies present in the latent representation which are ignored by a factorized model. The method in [6] shows that by introducing a side-information, it is possible to achieve a better rate-distortion performance. Moreover, [1] utilizes a context model to obtain the conditional distribution of latent elements conditioned on previously decoded elements. [2] takes a different approach and exploits the inter-channel dependencies in the latent space.

This thesis proposes a lossy image compression model with an entropy model that process both spatial and channel context information. The latent representation is divided along the channel dimension into a number of equal-sized slices. The entropy parameters are conditioned on previously decoded channel slices and previously decoded elements in the same slice, more effectively decreasing the information redundancy. Additionally, a latent residual prediction network is employed to reduce the error caused by quantization.

In this chapter, the overall model is explained in detail. First, the structure of the model is presented. Then, quantization method during training and inference is discussed. Entropy network and arithmetic coder are explained in the following sections.

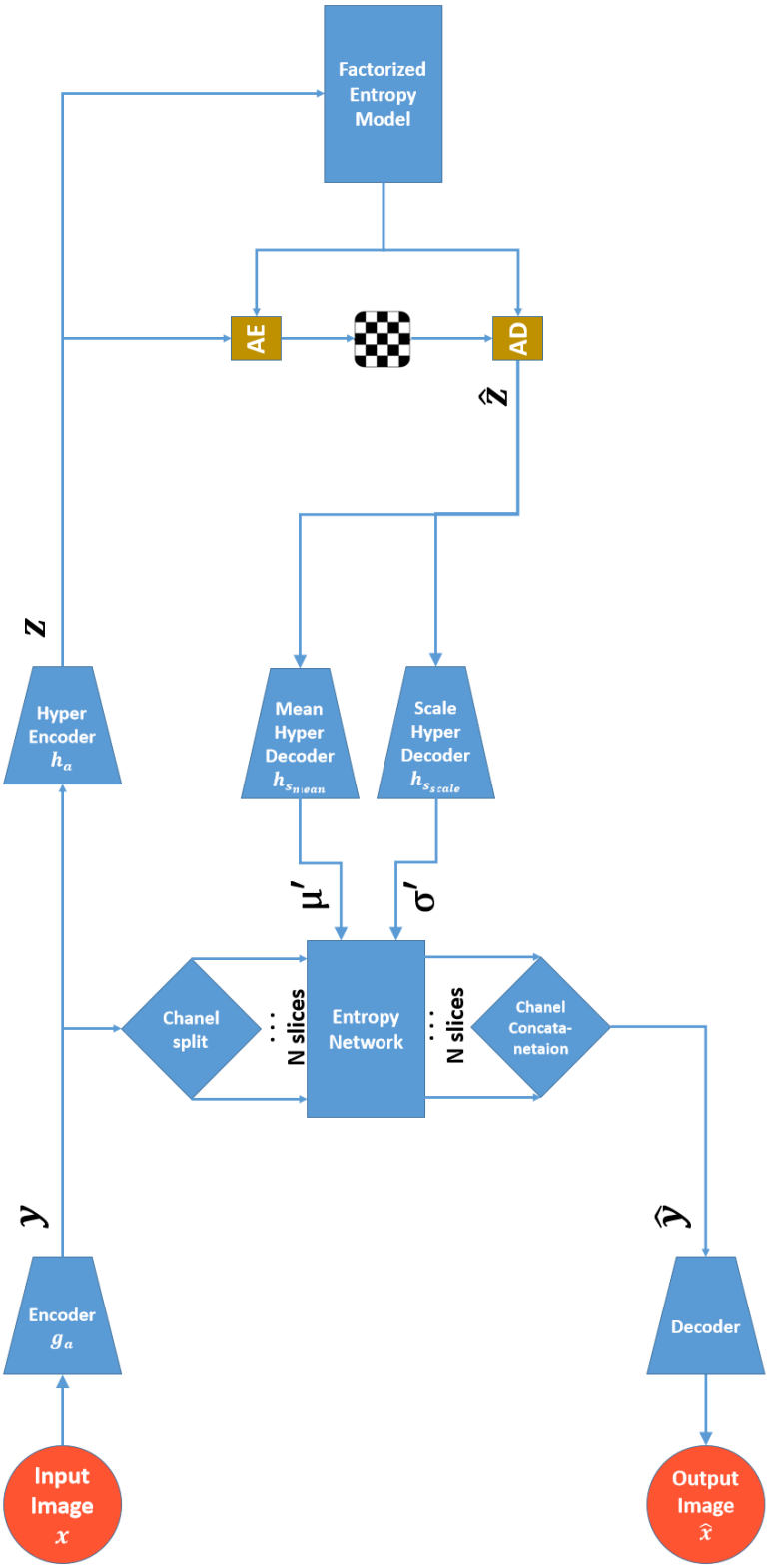


Figure 4.1: Diagram of proposed image compression network. The input image is fed to the encoder to be projected onto the latent space. Gaussian parameters of latent variable are found with entropy network. The decoder restores the image from the quantized latent variable.

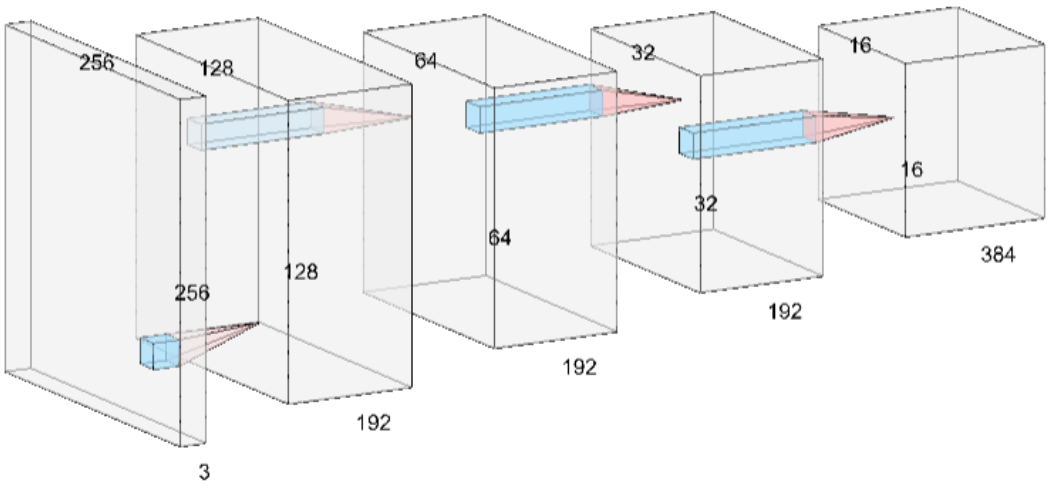


Figure 4.2: Encoder network

4.1 Model Structure

The compression model presented in this thesis builds upon the network in [6] and combines the ideas presented in [1] and [2]. The model includes an encoder, a decoder, a hyper encoder, a hyper decoder and entropy network. These parts are described further below. The network structure can be visualized with Figure 4.1.

The encoder g_a transforms the input image to latent variable y. There are 4 convolutional layers in the encoder network, each layer downsampling the image by 2. The channel number in the middle stages is 192. The output of the last layer is the latent variable, and has 1/16 times smaller spatial dimensions than input image while the channel number is increased to 384. In encoder, GDN non-linearity is applied to each intermediate output except the last one, which is the latent tensor. Each element of the latent tensor is modeled with a Gaussian distribution whose parameters are determined by the entropy network. The probabilities of the quantized values are found from the Gaussian distribution. The encoder network is shown in Figure 4.2.

The hyper encoder network h_a used in this thesis is the one given in [1]. There are 3 convolutional layers in this hyper encoder network, with the first one having stride of one and others two. As a result, y is downsampled further to hyper latent variable z, which has dimensions equal to input image's size divided by 64. The first and second

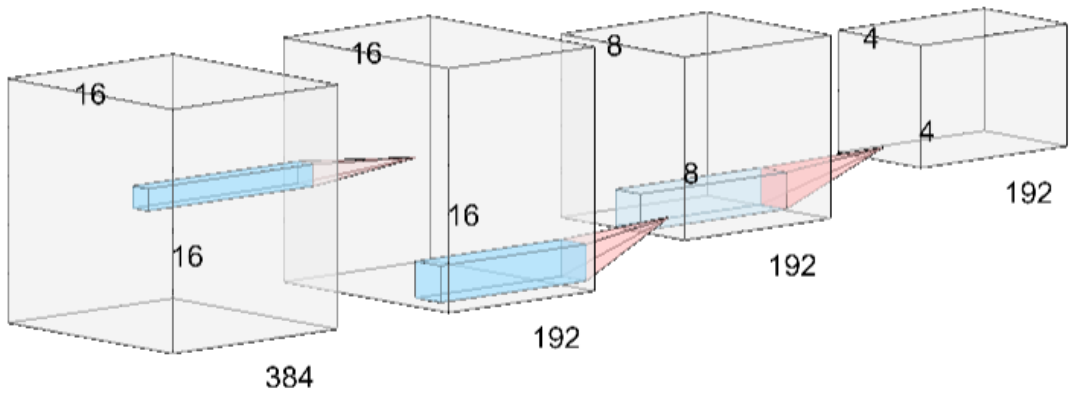


Figure 4.3: Hyper encoder network

layers produce feature maps with 192 channels. z, the output of final layer, has 384 layers. As y, z is also quantized to $\hat{z}$ before being entropy coded. $\hat{z}$ is modeled with a fully factorized probability model since the bit allocated for $\hat{z}$ are expected to be very small. The hyper encoder network is shown in Figure 4.3.

There are two identical hyper decoders in this network. The first one is dedicated to determination of mean of $\hat{y}$. This network is denoted by $h_{s_{mean}}$. The other network determines the scale of latent variable and is shown as $h_{s_{scale}}$. Both of these networks act on $\hat{z}$ to extract statistical information, and their outputs are denoted as μ' and σ' respectively. Hyper decoder network has 2 transposed convolution layers with stride 2 to upsample the image by 4 factors. The last layer is a regular convolutional layer with unit stride. The structure of these networks is displayed in Figure 4.4.

Entropy network approximates the parameters of Gaussian probability model of $\hat{y}$ from hyperprior and spatial and channel context information. As the hyper decoder network, there two entropy networks which are dedicated to mean and scale extraction. Entropy network is detailed in Section 4.3.

The decoder reconstructs the image from $\hat{y}$. With 4 transposed convolution layers, the latent representation is upsampled to the original image's dimensions. As nonlinearity, approximate inverse GDN function is applied. Like the encoder, the intermediate results have 192 channels. The output image has three channels which corresponds to RGB channels. The decoder network can be seen in Figure 4.5.

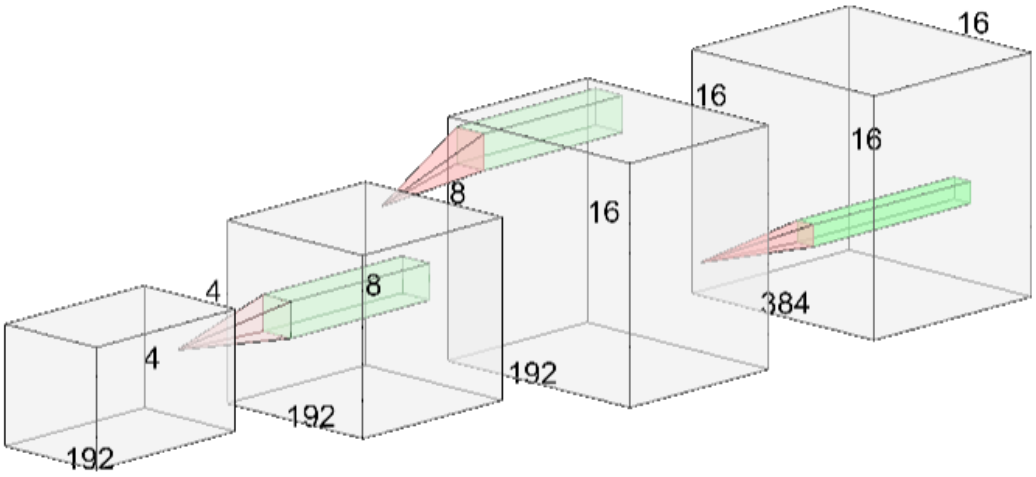


Figure 4.4: Hyper decoder network

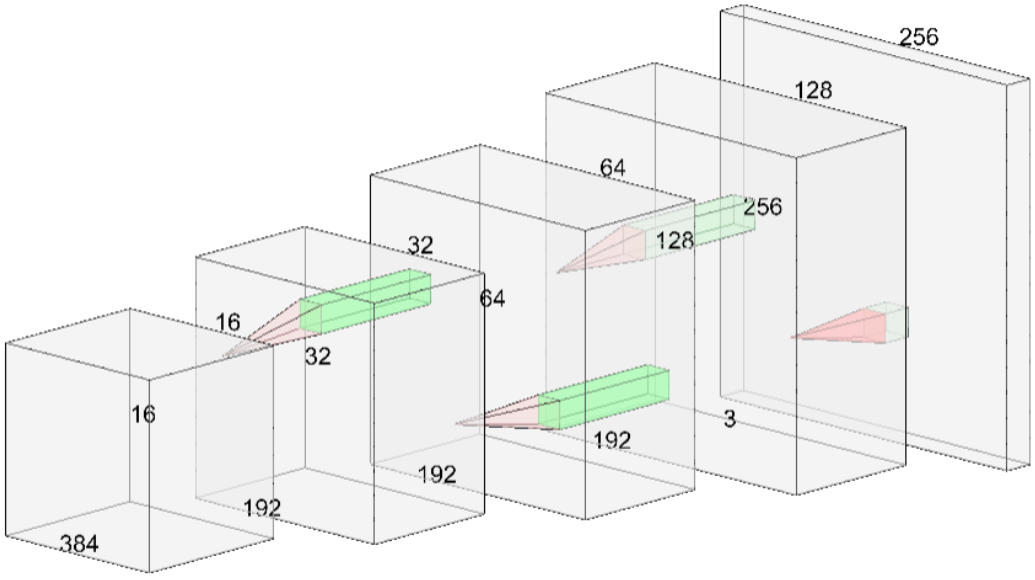


Figure 4.5: Decoder network

4.2 Quantization

Quantization is essential since only discrete variables can be encoded losslessly with arithmetic coder. However, quantization function has zero derivative almost everywhere, inhibiting gradient-based learning. To overcome this issue, [5] introduces additive uniform noise to imitate quantization error during training. Rounding is applied to the latent vector to obtain integer values during inference and compression. It should be noted that scalar quantization is adopted in this approach. Another approach is to apply rounding during training as well. Zero gradient issue is solved by substituting the gradient of round function with identity function during back propagation when training with rounded values [30].

In this thesis, a hybrid quantization approach during training is adopted, which was first proposed in [2]. In this method, uniform noise is exclusively added to the input of probability model. On the other hand, rounding with gradient substitution is applied to the input of synthesis and hyper synthesis transforms. The probability model learns a more accurate probability model with uniform noise. However, it has been observed in [2] that training synthesis transform with integer values increases reconstruction performance.

4.3 Entropy Network

The density of latent variables are defined as a Gaussian density [22] in order to be efficiently encoded with an arithmetic encoder after quantization. The mean and scale of this distribution are extracted by the entropy network. To capture the statistical dependencies among elements, each element is conditioned on multiple parameters. These parameters are hyperprior, channel auto-regressive and spatial auto-regressive components. The hyperprior network learns to express useful information to be compressed with the latent representation as side information. The size of the hyperprior is greatly smaller than the rate saving it provides. Moreover, since the information is available during decoding, there is no increase in computation latency. However, auto-regressive model works sequentially, dramatically increasing decoding time. There are two such components in this model, channel-wise auto-regressive

Figure 4.6: Channel prediction network

model and spatial auto-regressive model. Together, these two auto-regressive models are called the context model.

To establish a channel-wise auto-regressive model, the latent variables are split into slices along the channel dimension. The first slice uses no channel context information. Subsequent slices are conditioned on all of the previous channel slices. To extract channel context information, channel prediction network seen in Figure 4.6 is used. Each slice possesses its own channel prediction network. As seen in the figure, input size grows with slice index.

Spatial auto-regressive model aims to take advantage of spatial dependencies within a channel slice. Two methods have been explored to build spatial conditioning. The first one is using a masked convolution layer, which has been proposed in [1]. Masked convolution operation enforces the causal processing of the auto-regression. This method disables parallel processing since each element has to be encoded and decoded sequentially. Another method is to divide the splits further into patches and process each patch in parallel. The latter approach allows highly parallel processing in GPU, speeding up the encoding and decoding processes. An illustration of parallel spatial context prediction for a single channel tensor is given in Figure 4.7.

Outputs of hyperprior decoders, μ' and σ', are concatenated with the spatial and channel context model output. As the hyperprior decoder, there are two context models. μ' and σ' are bundled together with the appropriate tensors. These collection of tensors are given to their respective entropy parameter network. Entropy parameter network is a three-layered convolutional network, while each layer has a kernel size of 1. Again, there are two entropy parameter networks, and their outputs μ_i and σ_i are used

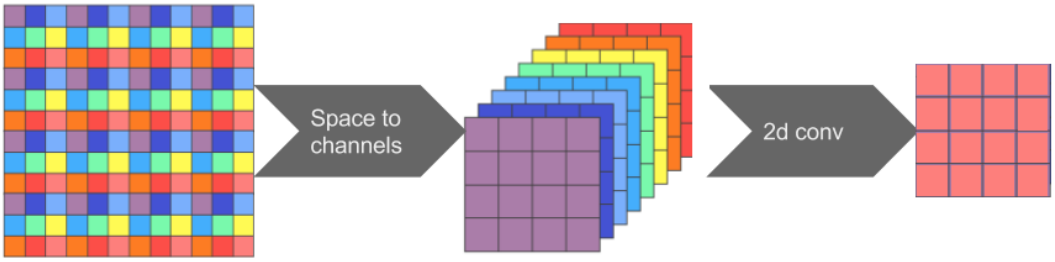


Figure 4.7: Parallel spatial context prediction. To parallelize the spatial context model, the latent is divided into patches. The colors represent indexes. In the image, spatial context prediction for the last index is done by concatenating the previous indexes in channel dimension and processing them with a convolution layer.

Figure 4.8: Entropy parameter network.

in the Gaussian probability model. The i-th entropy parameter network is given in Figure 4.8.

After the slice of latent variable is decoded, latent residual prediction is applied to reduce the information loss caused by quantization. The i-th latent residual prediction network is visualized in Figure 4.9. The tensor used to predict the mean μ_i is also used in latent residual prediction, along with the decoded slice.

The entropy network is the collection of auto-regressive models and entropy parameter networks. It collects and process statistical deep features to represent the distribution of latent variables with a multi-variate Gaussian model. The entropy parameter network dedicated to the i-th channel slice is shown in Figure 4.10.

Figure 4.9: Latent residual prediction network

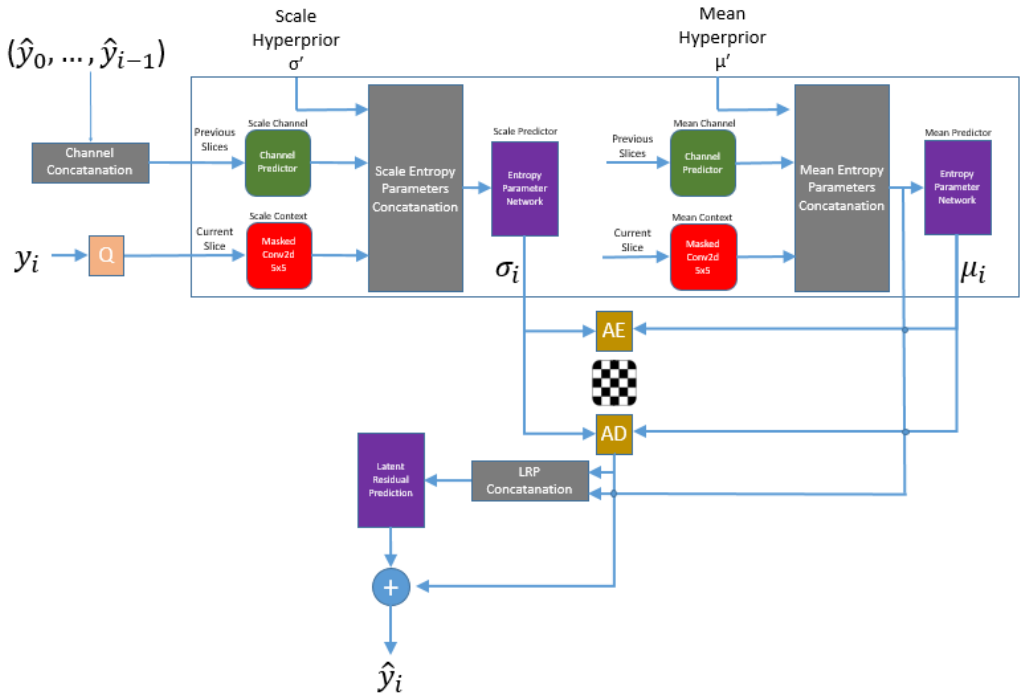


Figure 4.10: i-th entropy network. Firstly, quantization is applied to the i-th channel slice which is input of this network. The previously decoded channels are concatenated and given to the channel prediction network. Masked convolution establishes pixel-wise conditioning. The outputs of auto-regressive models are concatenated with hyperprior. The parameters of Gaussian model are estimated with entropy parameter networks. After the slice is decoded, the latent residual prediction network attempts to reduce the information loss caused by quantization. The output slice is passed on to the next entropy network, along with the previous slices.

4.4 Arithmetic Coding

Training of the compression network requires entropy calculation, since entropy provides a bound for minimum achievable bit rate. Hence, minimization of entropy corresponds to reducing the bits required to encode the latent representation. Nonetheless, for practical usage the latent variables have to be encoded into a bitstream. Arithmetic coding, which is a lossless compression algorithm, has been used widely in image compression literature ([5], [6], [1]). Arithmetic coding requires the probability model to be available during encoding and decoding, which is obtained with the pre-trained entropy network. Arithmetic coder operates on finite number of symbols with a discretized PMF. Consequently, the quantization of the continuous Gaussian CDF and determining the number of symbols are necessary. During training, the discretization of CDF is imitated by uniform noise as discussed in Section 4.2. During practical application, rounding is used to sample the CDF on integer points. However, this CDF covers the whole real line $(-\infty, \infty)$. To apply arithmetic coding, the number of symbols have to be finite. The arithmetic coder handles this issue by considering probability of any symbol outside a pre-defined range as zero. Afterwards, the arithmetic coding algorithm described in Section 2.2.1.2 is applied.

4.5 Compression Procedure

From input to bit-stream, the steps can be listed as follows;

1. The input image x is transformed to latent variable y with encoder g_a.
2. y is passed to the hyperprior encoder h_a as input to obtain z.
3. Both y and z are quantized to obtain $\hat{y}$ and $\hat{z}$.
4. $\hat{z}$ is transmitted as side information and encoded with a factorized prior entropy model.
5. From $\hat{z}$, p_{mean} and p_{scale} are obtained with hyperprior decoders $h_{s_{mean}}$ and $h_{s_{scale}}$.

6. $\hat{y}$ is divided into S slices, each slice having M/S channels where M = total # of channels of y, S = total number of slices.

7. From first slice $\hat{y_0}$, spatial context predictions $cp_{0_{mean}}$ and $cp_{0_{scale}}$ are extracted by spatial context model.

8. Concatenation of p_{mean} and $cp_{0_{mean}}$ is fed into Gaussian mean prediction network to determine mean values. A similar operation is done to obtain scale values.

9. The obtained Gaussian parameters are used to encode $\hat{y_0}$.

10. With concatenation of mean parameters and quantized slice, latent residual prediction is applied to reduce quantization errors

11. For subsequent slices, channel context predictions $ch_{i_{mean}}$ and $ch_{i_{scale}}$ are computed by giving concatenation of previously decoded slices $\hat{y_{<i}}$ to channel context model.

12. From slice $\hat{y_i}$, spatial context predictions $cp_{i_{mean}}$ and $cp_{i_{scale}}$ are extracted by spatial context model.

13. In addition to hyperprior and spatial context prediction, $ch_{i_{mean}}$ and $ch_{i_{scale}}$ are concatenated with their relevant parameter tensor to be fed into Gaussian mean and scale prediction networks.

14. The Gaussian parameters are used to encode $\hat{y_i}$ with arithmetic encoder.

15. With concatenation of mean parameters and quantized slice, latent residual prediction is applied to reduce quantization errors

16. Return to step 10 until all of the slices are encoded.

4.6 Decompression Procedure

From bit-stream to reconstructed image, the steps can be listed as follows;

1. $\hat{z}$ is decoded with factorized prior entropy model at receiver side

2. From $\hat{z}$, p_{mean} and p_{scale} are obtained with hyperprior decoders $h_{s_{mean}}$ and $h_{s_{scale}}$.

3. $p_{mean}(0,0)$ and $p_{scale}(0,0)$ are used to obtain mean and scale values of the first element of the first slice $y_0(\hat{0},0)$.

4. From bit-stream, first element $y_0(\hat{0},0)$ is decoded with computed mean and scale values.

5. From decoded element(s) of $\hat{y_0}$, $cp_{0_{mean}(m,n)}$ and $cp_{0_{scale}(m,n)}$ are found with spatial context module. (0<m<height of latent variable, 0<n<width of latent variable)

6. Concatenation of $p_{mean}(m,n)$ and $cp_{0_{mean}(m,n)}$ is fed into Gaussian mean prediction network to determine the mean value. A similar operation is done to obtain the scale value.

7. The obtained Gaussian parameters are used to decode $y_0(\hat{m},n)$

8. Return to Step 4 until all elements are decoded in the first slice $\hat{y_0}$

9. With concatenation of mean parameters and decoded slices, latent residual prediction is applied to reduce quantization errors

10. For subsequent slices, channel context predictions $ch_{i_{mean}}$ and $ch_{i_{scale}}$ are computed by giving concatenation of previously decoded slices $\hat{y_{<i}}$ to channel context model.

11. From bit-stream, first element $y_i(\hat{0},0)$ of i-th slice is decoded with hyperprior parameters $p_{mean}(0,0)$ and $p_{scale}(0,0)$ and channel context prediction $ch_{i_{mean}}(0,0)$ and $ch_{i_{scale}}(0,0)$.

12. From decoded element(s) $cp_{i_{mean}(m,n)}$ and $cp_{i_{scale}(m,n)}$ are computed. 0<m<height, 0<n<width

13. Concatenation of $p_{mean}(m,n)$, $ch_{i_{mean}}(m,n)$ and $cp_{i_{mean}(m,n)}$ is fed into Gaussian mean prediction network to determine the mean value. A similar operation is done to obtain the scale value.

14. The obtained Gaussian parameters are used to decode $y_i(\hat{m},n)$

15. Return to Step 12 until all elements are decoded
16. With concatenation of mean parameters and decoded slices, latent residual prediction is applied to reduce quantization errors
17. Return to Step 10 until all slices are decoded
18. Concatenate all decoded slices in channel dimension to obtain $\hat{y}$
19. $\hat{y}$ passes through decoder g_s to obtain reconstructed image $\hat{x}$

CHAPTER 5

EXPERIMENTAL RESULTS

This chapter is dedicated to present performance of the proposed image compression network. In addition, training setup and testing details are discussed as well. To show the effects of spatial and channel context models, visualizations of tensors involved in entropy model are given.

5.1 Training and Evaluation

The training of sequential model are done on two environments. The first one is a personal computer which has NVIDIA RTX 3070 GPU, Intel i7 10700KF CPU and 16 GB RAM. The other environment is Colab virtual environment provided by Google. For developing the model, CompressAI [31], which is a framework for deep learning-based image compression development is used. CompressAI library is based on PyTorch [32] deep learning library. During training, the datasets released on 2020 and 2019 for Challenge on Learned Image Compression (CLIC) have been used. There are 835 high quality natural images in total. Batch size is set to 8 images. Randomly cropped images with 256x256 size constitute each batch. The model with highest lambda is trained first, then each network is obtained by reducing the lambda when the model reaches a flat region. Learning rate is set to 10^{-4} in the beginning of each training phase, then reduced to 10^{-5} when a plateau is reached. It should be noted that this training strategy is not optimal and was chosen because of time constraints. The results can be improved by choosing a different training strategy.

Validation and performance evaluation are done using the Kodak Image Dataset. Kodak Image Dataset contains 24 RGB images of size 768x512. Rate vs. distortion

graph is obtained by finding the bit-rate (in bits-per-pixel) of the compressed file and distortion according to PSNR or MS-SSIM. The most common distortion metric is PSNR. The formula for PSNR is given in Equation 5.1 where MSE is the mean-square error between the input image and reconstructed image.

$$PSNR = 10log_{10}\frac{255^2}{MSE} \tag{5.1}$$

5.2 Number of Parameters

The number of parameters is a numerical indicator for the complexity of the model. Table 5.1 shows total number of parameters of each network. It should be noted that the number of channels of the proposed models have not been optimized and can be reduced without significant performance loss.

	Hyperprior	Spatial Auto-regressive	Channel Auto-regressive	Proposed	Proposed (Parallel)
Number of parameters (M)	5	14	122	65	131

Table 5.1: Number of parameters of various models

As expected, the hyperprior model has the least amount of parameters. Spatial auto-regressive model extends the hyperprior model with an context prediction and entropy parameter network. Channel auto-regressive model is considerably larger since there is a separate entropy network for each slice. The proposed model is smaller than channel auto-regressive model because of the chosen channel sizes. The proposed model is much larger because of the paralellized spatial context predictor.

5.3 Compression Performance

Evaluation of compression performance is done by comparing the rate-distortion curves obtained from previous models. These models are proposed in [5], [6], [1], [33] and [2]. [5] uses a factorized entropy model to learn the distribution of pixels. This model assumes the pixels are independent. On the other hand, the model proposed [6] adopts a zero mean Gaussian model to be used by an arithmetic coder. Furthermore, a hyperprior network to extract dependencies between neighboring pixels is introduced. There are two models presented in [1]. The first one extends the hyperprior model with a non-zero Gaussian assumption and predicts mean values from hyperprior along with scale values. The second one adds a spatial auto-regressive component to the first model to capture spatial correlations more successfully. [33] proposes to use attention modules and residual connections with the model presented in [1]. [33] also uses a deeper network with smaller filter sizes. [2] suggests to use a channel-wise auto-regressive model to reduce information redundancy in the latent space. [34] adds window based attention modules to enhance the performance of channel-wise auto-regressive model. The results of these models, together with the proposed model and conventional methods such as BPG and JPEG are given in Figure 5.1. It should be noted that models optimized for MS-SSIM always have less PSNR then thier MSE optimized counterparts since MS-SSIM optimization tends to over-smooth edges and details. Rate-distortion curves with respect to MS-SSIM metric are shown in Figure 5.2. Since performance of MS-SSIM optimized channel auto-regressive model is not published, it is not included in Figure 5.2.

According to the Figure 5.1, JPEG has the worst compression performance. This is an expected result since JPEG has the lowest complexity and fastest encoding and decoding times. Even though JPEG 2000 provides a considerable performance boost, it still cannot reach the performance of lowest performing learning based model, which is the factorized model. BPG, which is a image compression algorithm based on HEVC compression algorithm, is almost on par with the hyperprior model presented. BPG performs slightly better at lower bit-rates. However, as bit-rate increases, the hyperprior model surpass BPG. Non-zero Gaussian assumption increases the performance of hyperprior model. Remaining learning based models leverage non-zero

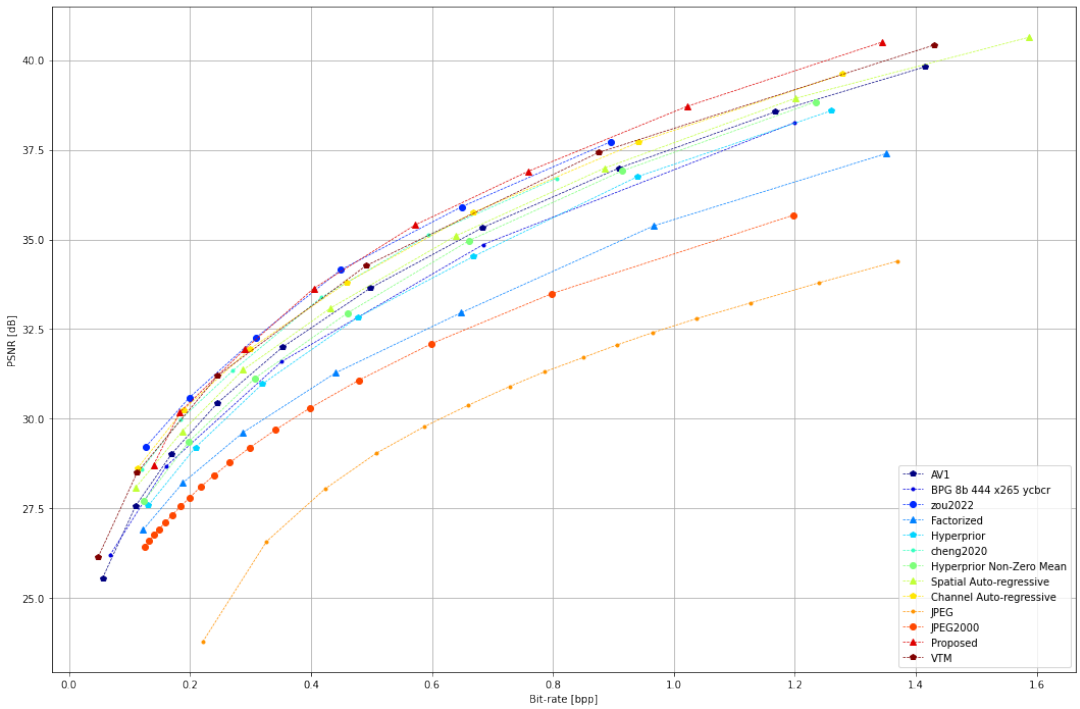


Figure 5.1: PSNR vs Bit-Rate Curves

Gaussian probability model as well. Introduction of auto-regressive models yield significantly better results. Comparison of spatial and channel auto-regressive models shows that there are stronger dependencies between channels and modeling these dependencies can improve the performance more. Furthermore, comparing the model in [33] with the spatial auto-regressive model proves that residual connections and deeper network can improve the performance. Furthermore, [34] shows that window based attention module can increase the rate-distortion performance when compared with channel auto-regressive model. The proposed method performs the best at high bit-rates, beating both [2] and [34]. However, it does not perform well at low bit-rates. The reason for this outcome is that features contained in each channel does not differentiate much at low bit-rates and channel auto-regressive model is sufficient to express the dependencies present in the latent variable. As bit-rate increases, the features get more distinct and channel auto-regressive model is not enough to capture all of the correlations among elements. Thus, the proposed model performs better as the bit-rate increases. It should be noted that ideally the proposed model's performance should be equal to the model in [2] at the very least. Because of the non-optimal training strategy, a worse performance is observed at low bit-rates.

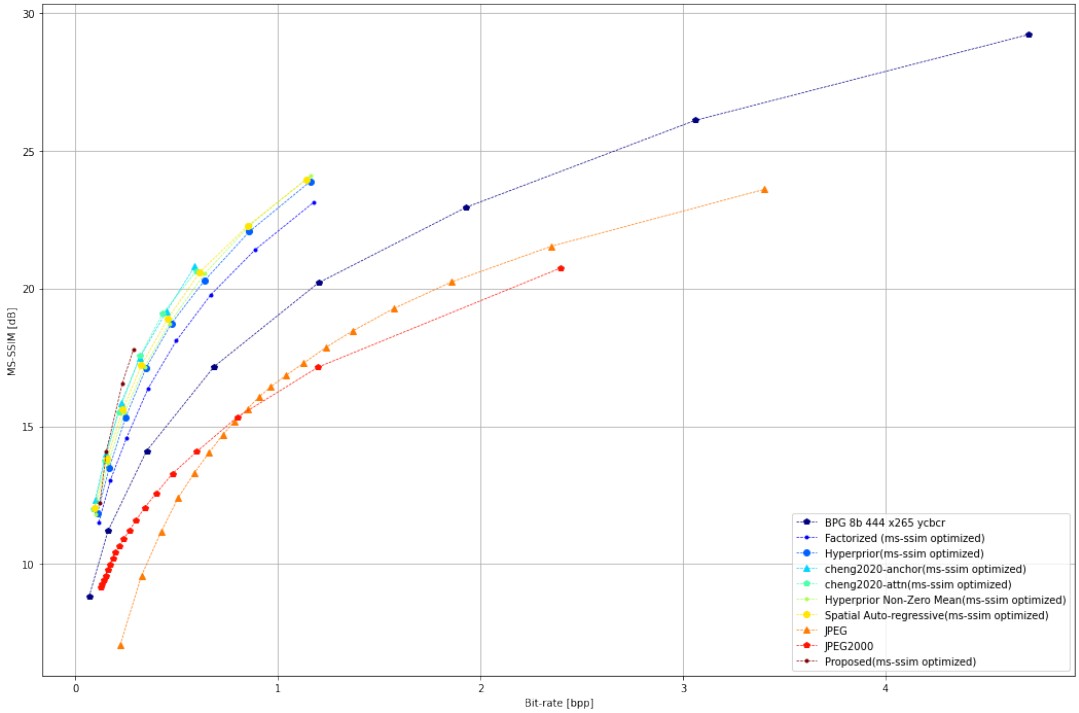


Figure 5.2: MS-SSIM vs Bit-Rate Curves

5.4 Inspection of Latent Space

To asses the effectiveness of the proposed method, tensors in the latent space are inspected in this section. A visualization of the latent variable of the image in Figure 5.3 can be seen in Figure 5.4. PSNR is 26.0832 dB and bit-per-pixel is 0.2123 for this image. It can be observed that most of the energy is compacted into the channel shown in left-upper-most patch. This channel also has the highest bit-per-pixel allocation with 0.0056.

Figure 5.3: Input Image

After the mean is computed, it is subtracted from the latent before rounding operation. The mean-subtracted latent tensors can be seen in Figure 5.5. Most of the channels are sparse except a small number of channels. It can easily be concluded that most of the bits are accumulated in a small portion of the channels. Presence of structure in the latent space decreases the efficiency of the coder. In the Figure 5.6 errors normalized with scale values are shown. It can be observed that structures are mostly eliminated in the normalized tensor.

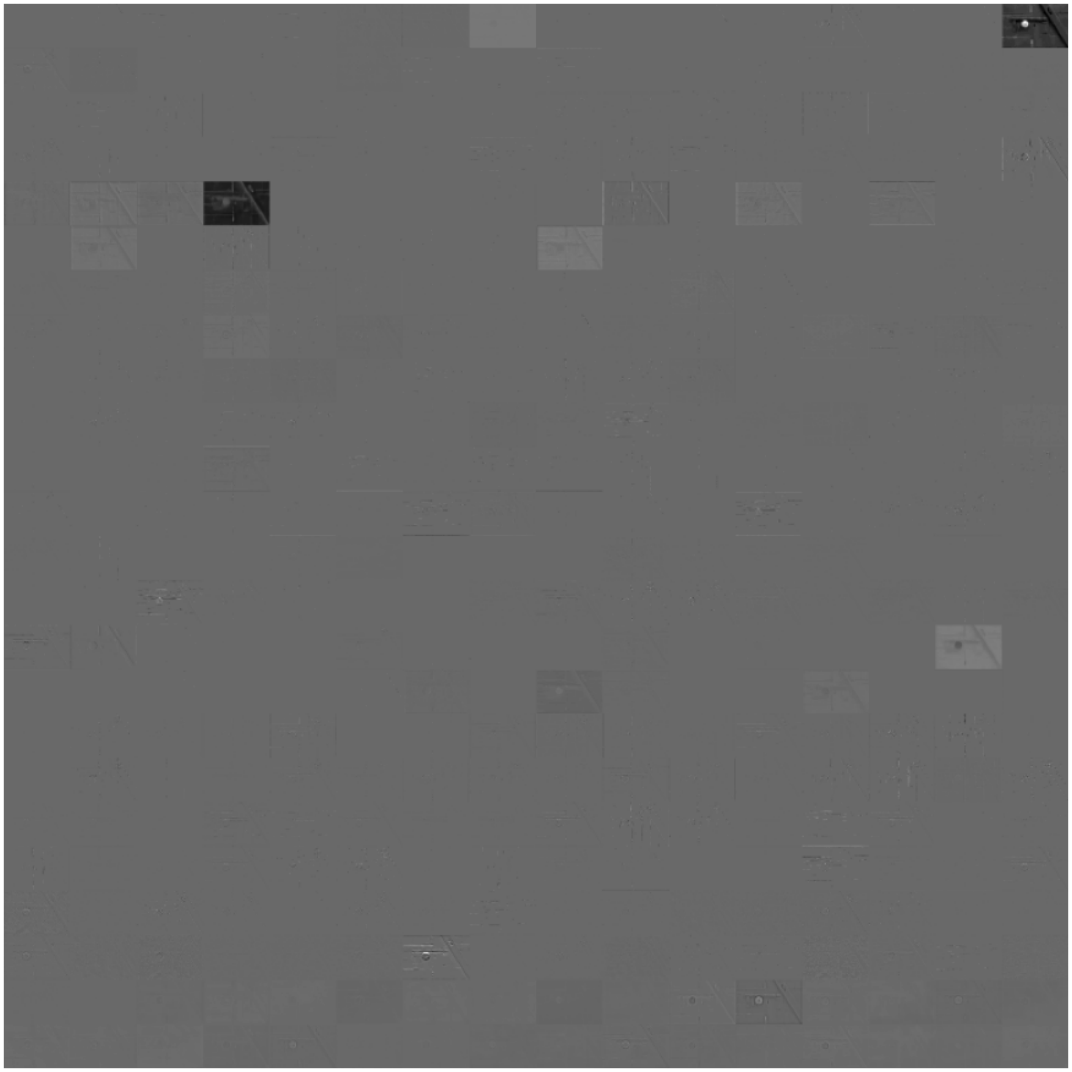


Figure 5.4: Latent Representation of the Input Image

With increasing bit-rate, more details are allowed to be represented in the latent space. By comparing the latent variables produced with different lambda values, this phenomena can be observed. An illustration can be found in Figure 5.7 and 5.8. As expected, more complex structures are observed in the latent representation as bit-rate increases.

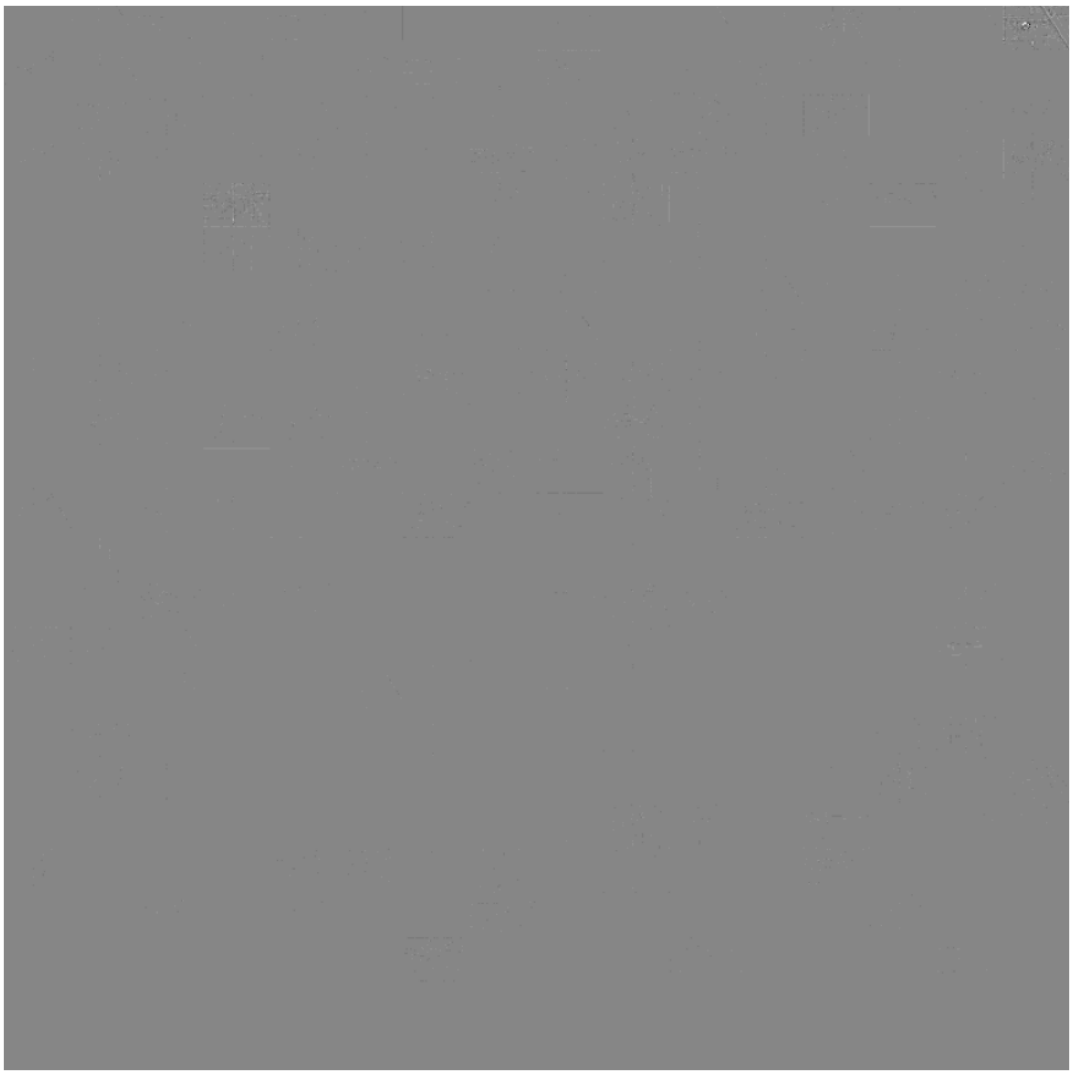


Figure 5.5: Mean Subtracted Latent

Histograms of the channels which require the most bit-per-pixel for multiple lambda values are shown in Figure 5.9. Even though at low bit-rates Gaussian model performs well, as more details are allowed, the histogram deviates from a normal distribution. A different probability model can be utilized in the future to represent the distribution of the latent more successfully.

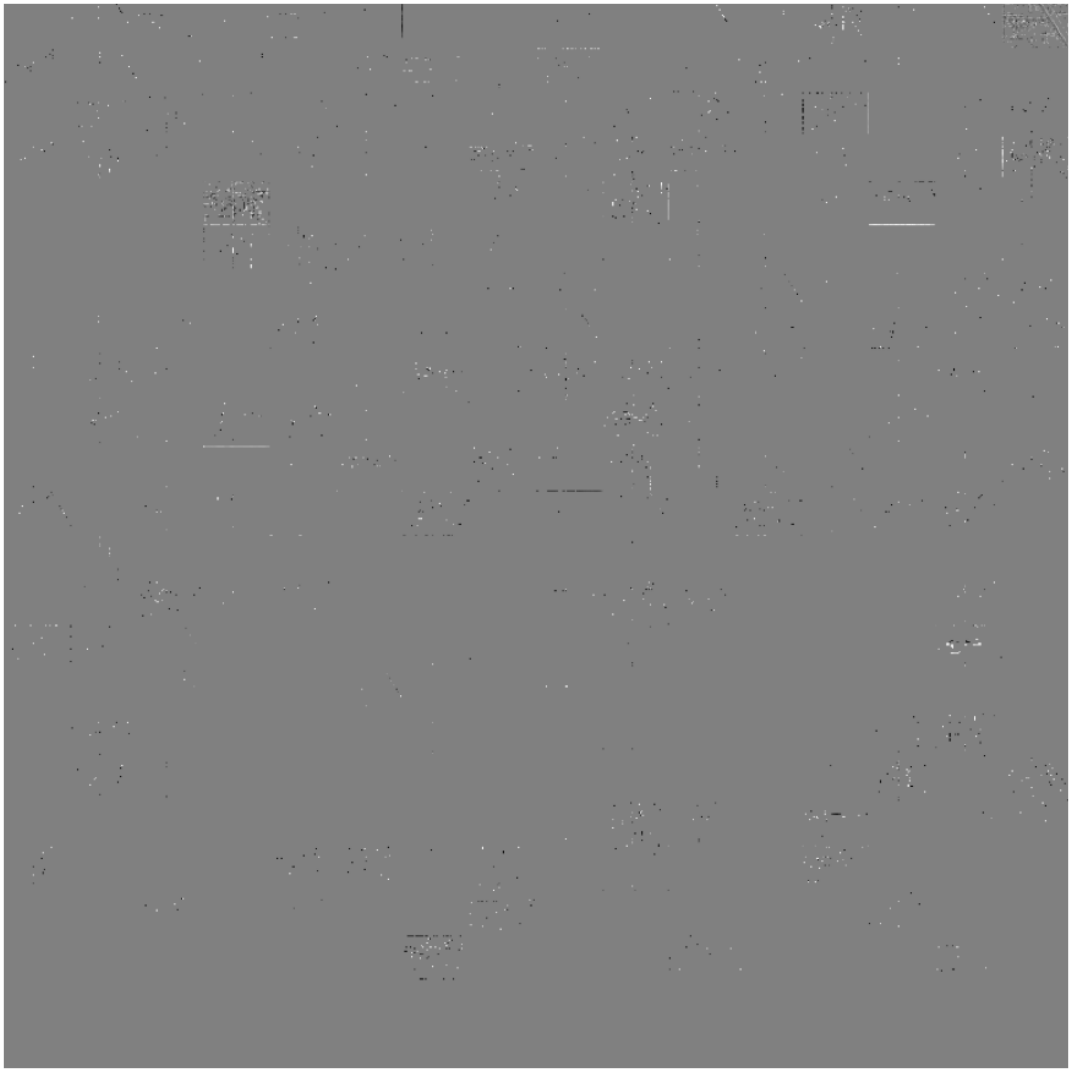


Figure 5.6: Mean Subtracted Scale Normalized Latent

5.5 Compression Latency and Parallelized Entropy Model

One of the most important metric when comparing the performance of a compression model is the time it takes to encode and decode an image. Since there are no auto-regressive computation involved in hyperprior model [6], it can be run on GPU flawlessly, dramatically reducing the computation time. Moreover, because of the low complexity of hyperprior model, it is the fastest deep compression model. Among the auto-regressive models, the channel auto-regressive model [2] is the faster one due to low number of sequential iterations. The spatial auto-regressive model [1] has to compute mean and scales values for each element in the latent variable sequen-

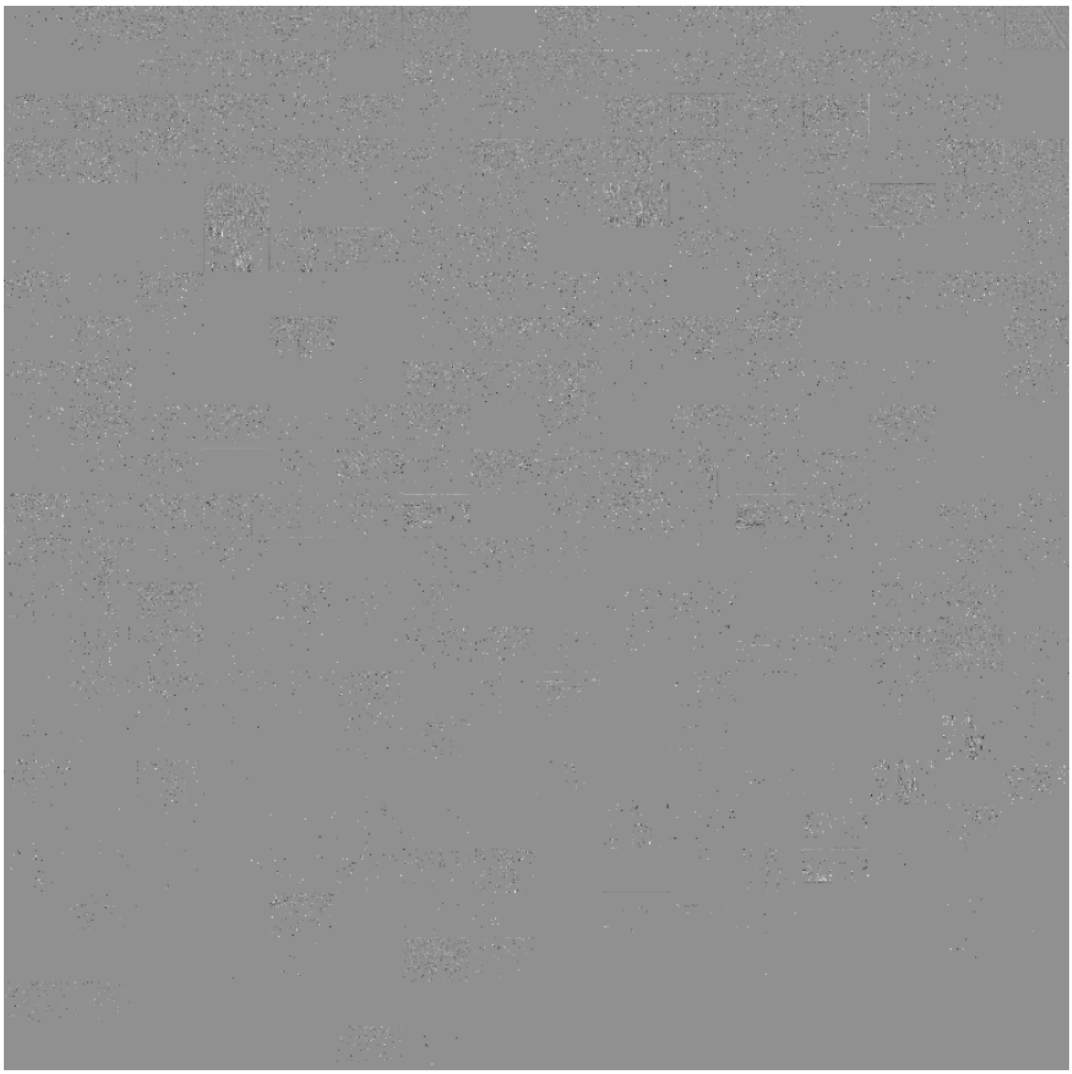


Figure 5.7: Mean Subtracted Scale Normalized Latent at PSNR: 35.5 dB - bpp: 0.380

tially. This processes renders the model fully sequential, slowing down the encoding and decoding of the image in magnitudes. On the other hand, each iteration of the channel auto-regressive model can be run parallel on GPU. The number of sequential iterations equals to the number of slices, which is defined as 10 in the paper [2].

The proposed model in this thesis uses both sequential and channel auto-regressive models. While it performs better compared to the other models, the encoding and decoding times are much higher. To prevent this, a novel approach to partition the latent into smaller patches and process the items with the same indexes together is proposed. This approach reduces the encoding and decoding times dramatically.

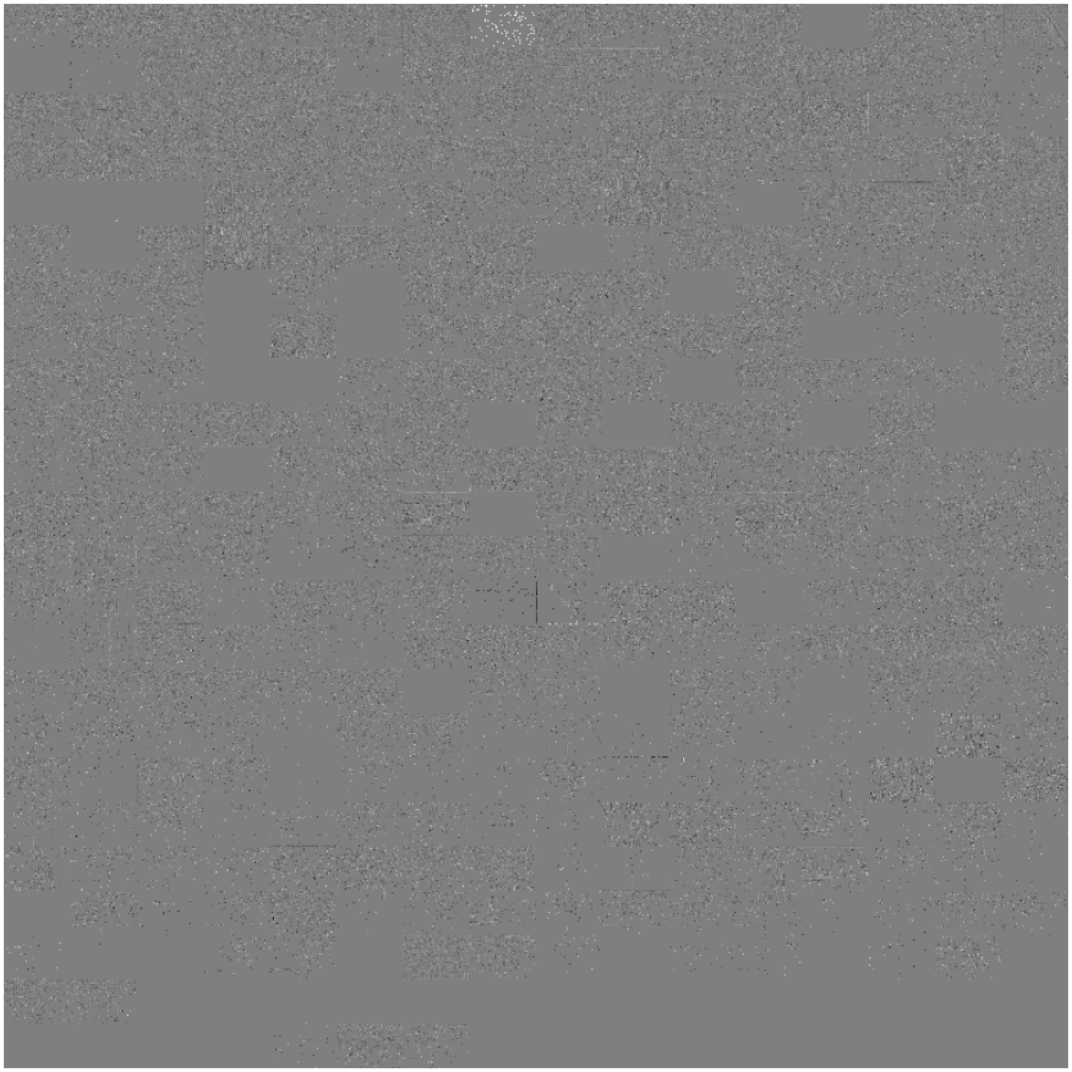
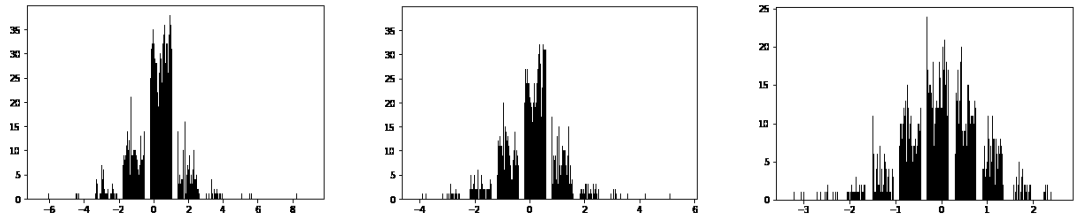


Figure 5.8: Mean Subtracted Scale Normalized Latent at PSNR: 40.65 dB - bpp: 1.170

Figure 5.2 shows the time it takes to encode and decode an image. The data for hyperprior and spatial auto-regressive models are taken from CompressAI. The proposed method is tested locally on RTX 3070 and Intel i7 10700KF, while proposed parallel method is tested on Google Colab. Since there was no data for channel auto-regressive model, it is not included in the table. However, it is expected to accomplish encoding and decoding slightly faster than the proposed parallel method.

The parallel model has proved that faster compression is possible. Unfortunately, per-



(a) PSNR: 30.1 dB
bpp: 0.063

(b) PSNR: 35.5 dB
bpp: 0.380

(c) PSNR: 40.65 dB
bpp: 1.170

Figure 5.9: Histograms of mean subtracted scale normalized latents at different bit-rates

(seconds)	Hyperprior	Spatial Auto-regressive	Proposed	Proposed (Parallel)
Encoding Time (CPU)	0.72	5.37	24.61	6.29
Encoding Time (GPU)	0.035	2.94	18.57	1.94
Decoding Time (CPU)	0.70	9.68	75.05	5.07
Decoding Time (GPU)	0.0249	6.25	56.24	1.70

Table 5.2: Encoding and decoding times of various models.

formance experiments are limited to two quality values because of time constraints. Nevertheless, the obtained data shows that the proposed model can be accelerated with minimal loss in compression performance. The comparison is shown in Table 5.3.

Patch Size	$\lambda = 0.0035$	$\lambda = 0.013$
2	PSNR:30.4 bpp:0.222	PSNR:33.4 bpp:0.445
4	PSNR:30.4 bpp:0.216	PSNR:33.4 bpp:0.409

Table 5.3: Experimental results of parallel context model for two lambda values. The first elements in each patch are encoded without spatial context information. There are more elements to encode without spatial context information when patch size decreases. Hence, performance tends to decrease with the patch size.

CHAPTER 6

CONCLUSION

In this thesis, a deep learning model to compress natural images is presented. The model is trained in a end-to-end fashion, minimizing the rate-distortion loss calculated with a Lagrange multiplier to control the trade-off between bit-per-pixel and quality.

This work adapts the auto-encoder for image compression framework presented in [27]. Basically, the image is transformed into the latent representation which is encoded with a arithmetic coder to be transmitted or stored with least amount of bits given the entropy model. Afterwards, the encoded tensor is decoded with the same entropy model and transformed back into the image space. The compression model learns the most optimal transformation for rate-distortion performance. The entropy model given in [27] is improved with hyperprior which can be considered as a side information packed with the actual latent representation to increase the performance. Additionally, effects of auto-regressive models on compression performance are explored. The spatial auto-regressive model introduced in [1] uses a pixel-wise conditional model to reduce information redundancy between pixels. On the other hand, the channel auto-regressive model presented in [2] adopts a channel-wise conditional probability model to prevent unnecessary repetition of information between channels.

The entropy model in this thesis utilizes both channel and spatial context information. First, the latent tensor is divided into smaller slices along the channel dimension. These slices are encoded and decoded sequentially while each channel slice is conditioned on the previously processed slices. Furthermore, a masked convolution extracts useful information from the earlier pixels. The context information obtained from pixel-wise and channel-wise auto-regressive models are fed into parameter estimator network together with hyperprior tensor. Even though this approach increases

rate-distortion performance, the computation time increases dramatically because of pixel-by-pixel processing of spatial context model. A patch-based spatial context model is proposed to reduce the compression latency. The latent tensor is divided further into patches before the entropy network. Each element inside a patch is indexed in raster scan order, and every element with the same index is processed in parallel. Moreover, the element being processed is conditioned on formerly processed elements inside neighboring patches as well as the elements inside the same patch. The increased receptive field compensates the overhead caused by increased bits allocated for earlier indexes. With a small performance trade-off, great computational efficiency can be achieved with patch division.

6.1 Future Work

Future work may include utilizing a Gaussian mixture model or another probability model to better express the distribution of latent representations. A more successful approximation will improve the coding efficiency.

The current patch division method divides the latent tensor into patches with equal size. Division into variable sized patches as in [4] can increase efficiency of patch division method. An additional network can be implemented to decide on the optimal patch sizes. Smoother regions can be divided into larger patch sizes while regions with high variance can have smaller patches. Similar to latent residual network taking mean estimation input, the patch division network can take the scale predictor's input tensor since size of the patches would depend on the local scale.

REFERENCES

[1] D. Minnen, J. Ballé, and G. Toderici, "Joint autoregressive and hierarchical priors for learned image compression," 2018.

[2] D. Minnen and S. Singh, "Channel-wise autoregressive entropy models for learned image compression," 2020.

[3] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1992.

[4] ISO/IEC 23008-2:2020, "ISO/IEC 23008-2:2020 information technology — high efficiency coding and media delivery in heterogeneous environments — part 2: High efficiency video coding," 2020.

[5] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimized image compression," 2016.

[6] J. Ballé, D. Minnen, S. Singh, S. J. Hwang, and N. Johnston, "Variational image compression with a scale hyperprior," 2018.

[7] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 1948.

[8] D. Huffman, "A method for the construction of minimum redundancy codes," *Proceedings IRE 40*, no. 10, pp. 1098–1101, 1952.

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

[10] G. G. Langdon, "An introduction to arithmetic coding," *IBM J. Res. Dev.*, vol. 28, pp. 135–149, 1984.

[11] D. R. Bull and F. Zhang, *Intelligent image and video compression: Communicating Pictures*. Academic Press, 2 ed., 2021.

[12] Z. Wang, E. Simoncelli, and A. 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, 2003.

[13] K. Ding, K. Ma, S. Wang, and E. P. Simoncelli, "Image quality assessment: Unifying structure and texture similarity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2020.

[14] Z. Tu, Y. Wang, N. Birkbeck, B. Adsumilli, and A. C. Bovik, "UGC-VQA: Benchmarking blind video quality assessment for user generated content," *IEEE Transactions on Image Processing*, vol. 30, pp. 4449–4464, 2021.

[15] P. Manocha, A. Kumar, B. Xu, A. Menon, I. D. Gebru, V. K. Ithapu, and P. Calamia, "Saqam: Spatial audio quality assessment metric," 2022.

[16] F. Rosenblatt, "The perceptron - a perceiving and recognizing automaton," Tech. Rep. 85-460-1, Cornell Aeronautical Laboratory, Ithaca, New York, January 1957.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[18] G. Hinton, "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent," 2012.

[19] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture in two nonstriate visual areas (18 and 19) of the cat," *Journal of Neurophysiology*, vol. 28, pp. 229–289, 1965.

[20] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction, and functional architecture in the cat's visual cortex," *Journal of Physiology (London)*, vol. 160, pp. 106–154, 1962.

[21] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.

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

[23] D. J. Heeger, "Normalization of cell responses in cat striate cortex," *Visual Neuroscience*, vol. 9, no. 2, p. 181–197, 1992.

[24] M. Carandini and D. Heeger, "Normalization as a canonical neural computation. nat," *Nature reviews. Neuroscience*, vol. 13, pp. 51–62, 11 2011.

[25] S. Chen and R. Gopinath, "Gaussianization," in *Advances in Neural Information Processing Systems* (T. Leen, T. Dietterich, and V. Tresp, eds.), vol. 13, MIT Press, 2000.

[26] M. Carandini and D. Heeger, "Normalization as a canonical neural computation. nat," *Nature reviews. Neuroscience*, vol. 13, pp. 51–62, 11 2011.

[27] J. Ballé, V. Laparra, and E. P. Simoncelli, "End-to-end optimization of nonlinear transform codes for perceptual quality," 2016.

[28] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2013.

[29] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," 2016.

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

[31] J. Bégaint, F. Racapé, S. Feltman, and A. Pushparaja, "Compressai: a pytorch library and evaluation platform for end-to-end compression research," 2020.

[32] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," 2019.

[33] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," 2020.

[34] R. Zou, C. Song, and Z. Zhang, "The devil is in the details: Window-based attention for image compression," 2022.