A PIXEL-BY-PIXEL LEARNED LOSSLESS IMAGE COMPRESSION METHOD WITH PARALLEL DECODING

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

BY

SINEM GÜMÜŞ

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

JULY 2022

Approval of the thesis:

A PIXEL-BY-PIXEL LEARNED LOSSLESS IMAGE COMPRESSION METHOD WITH PARALLEL DECODING

submitted by **SINEM GÜMÜŞ** 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. Gözde Bozdağı Akar
Electrical and Electronics Engineering, METU _____

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

Prof. Dr. Alptekin Temizel
Graduate School of Informatics, METU _____

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

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

Date: 05.07.2022

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: Sinem Gümüş

Signature :

ABSTRACT

A PIXEL-BY-PIXEL LEARNED LOSSLESS IMAGE COMPRESSION METHOD WITH PARALLEL DECODING

Gümüş, Sinem
M.S., Department of Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. Fatih Kamışlı

July 2022, 71 pages

The success of deep learning in computer vision applications has led to the use of learning based algorithms also in image compression. Learning based lossless image compression algorithms can be divided into three categories, namely, pixel-by-pixel (or masked convolution based) algorithms, prior based algorithms and latent representation based algorithms. In the pixel-by-pixel algorithms, each pixel's probability distribution is obtained by processing the previously coded left and upper neighbouring pixels with a neural network (NN), which is then used by an arithmetic coder for lossless compression. In the prior based algorithms, the probability distribution of the image is conditioned on a prior that is obtained with a NN and transmitted to the decoder. In the latent representation based algorithms, the image is transformed to a latent domain with a learned invertible mapping, and the latent representation is lossless compressed.

This thesis studies a learned lossless image compression method that falls into the pixel-by-pixel (or masked convolution based) algorithms category. The study aims to provide a learned lossless image compression method by modelling each pixel's probability distribution with a Gaussian Mixture Model (GMM), whose parameters

are obtained by processing the pixel's causal neighbourhood (i.e. previously compressed pixels) with a relatively simple NN. This causality dependency causes the decoder to operate sequentially, i.e. the NN has to be run for each pixel sequentially, which increases decoding time significantly. The causality dependency can be easily alleviated at the encoder via masked convolutions. To reduce the decoding time, parallel encoding and decoding algorithms are studied and implemented. The obtained lossless image compression performance is competitive and is compared to both state-of-the art traditional and learning based methods.

Keywords: lossless image compression, conditional probability model, artificial neural networks (ANN)

ÖZ

PARALEL KOD ÇÖZME İLE PİKSEL-PİKSEL ÖĞRENİLMİŞ KAYIPSIZ GÖRÜNTÜ SIKIŞTIRMA YÖNTEMİ

Gümüş, Sinem
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. Fatih Kamışlı

Temmuz 2022 , 71 sayfa

Bilgisayarla görü uygulamalarında derin öğrenmenin başarısı, görüntü sıkıştırmada da öğrenme tabanlı algoritmaların kullanılmasına öncü olmuştur. Öğrenme tabanlı kayıpsız görüntü sıkıştırma algoritmaları, piksel-piksel (veya maskeli evrişim tabanlı) algoritmalar, ön bilgi tabanlı algoritmalar ve gizli temsil tabanlı algoritmalar olmak üzere üç kategoriye ayrılabilir. Piksel-piksel algoritmalarda, her pikselin olasılık dağılımı, önceden kodlanmış sol ve üst komşu piksellerin bir sinir ağı (SA) ile işlenmesiyle elde edilir ve sonra kayıpsız sıkıştırma için bir aritmetik kodlayıcı tarafından kullanılır. Ön bilgi tabanlı algoritmalarda, görüntünün olasılık dağılımı, bir SA ile elde edilen ön bilgiye göre koşullandırılır ve kod çözücüye iletilir. Gizli temsil tabanlı algoritmalarda, görüntü, öğrenilmiş bir tersinir eşleme ile gizli bir bilgi alanına dönüştürülür ve gizli temsil kayıpsız bir şekilde sıkıştırılır.

Bu tez, piksel-piksel (veya maskeli evrişim tabanlı) algoritmalar kategorisine giren öğrenilmiş ve kayıpsız bir görüntü sıkıştırma yöntemini araştırmaktadır. Yapılan çalışma, Gauss Karışım Modeli (GKM) ile her pikselin olasılık dağılımını modelleyerek öğrenilmiş kayıpsız bir görüntü sıkıştırma yöntemi sağlamayı amaçlamaktadır. Gauss

Karışım Modeli'ne ait parametreler, pikselin nedensel komşuluğunu (yani daha önce sıkıştırılmış pikselleri) nispeten basit bir SA ile işleyerek elde edilmiştir. Bu nedensellik bağımlılığı, kod çözücünün sıralı olarak çalışmasına neden olur, yani SA'nın her piksel için sıralı olarak çalıştırılması gerekir. Bu durum kod çözme süresini önemli ölçüde artırır. Kodlayıcı tarafında nedensellik bağımlılığı maskeli evrişimler yoluyla kolayca sağlanabilir. Kod çözme süresini azaltmak için paralel kodlama ve kod çözme algoritmaları araştırılmış ve uygulanmıştır. Elde edilen kayıpsız görüntü sıkıştırma performansı rekabetçidir ve hem en gelişmiş geleneksel yöntemlerle hem de öğrenme tabanlı yöntemlerle karşılaştırılmıştır.

Anahtar Kelimeler: kayıpsız görüntü sıkıştırma, koşullu olasılık modeli, yapay sinir ağları

To my family..

ACKNOWLEDGMENTS

I want to express my deep gratitude to Assoc. Prof. Dr. Fatih Kamışlı, who supported me at every stage of my thesis and my studies. He was an incredible supervisor to me and apart from my thesis work, he always guided me during my graduate education in the best way possible.

I would like to thank my colleague, Görkem Algan, for helping me. Especially on some specific topics related to deep learning, searching for solutions to encountered problems can be a very time-consuming action. In such situations, he has always been someone I could consult quickly.

I also want to thank Aselsan Inc. for their support.

Finally, I would like to thank my family for always being there for me and believing in me. They witnessed most of the stress I experienced during my graduate education, but they always motivated me and tried to make me believe that I could succeed.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xiv

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xix

CHAPTERS

1 INTRODUCTION 1

1.1 Lossless Image Compression 1

1.2 Problem Definition for Lossless Compression 3

1.3 Contributions 4

1.4 The Outline of the Thesis 5

2 BACKGROUND 7

2.1 Random Variable Basics 7

2.2 Density Estimation 8

2.2.1 Non-parametric Density Modelling 9

2.2.2 Parametric Density Modelling 9

2.3 Shannon's Entropy 10
2.4 Arithmetic Coding 12
2.5 Performance Measure for Lossless Compression 13
2.6 Convolutional Neural Network (CNN) 15
3 RELATED WORK 17
3.1 Engineered Lossless Coders 17
3.2 Learning Based Lossless Compression 20
3.2.1 Autoregressive Generative Models 20
3.2.2 Multi-scale Autoregressive Lossless Compression 26
3.2.3 Flow-based Compression 28
4 PROPOSED METHOD 31
4.1 Overview 31
4.2 Gaussian Mixture Model (GMM) 31
4.3 Discretized Mixture Model 33
4.4 The Probability Model of Pixels 35
4.5 Network Architecture and Optimization 37
4.6 Real Compression 38
4.6.1 Encoding Operation 40
4.6.2 Decoding Operation 40
4.7 Time Optimization for Decoding Operation 42
5 EXPERIMENTAL RESULTS 49
5.1 Compression Performance 49
5.2 Encoding and Decoding Times 50

5.3 Ablation Study 54

5.4 Visual Results 57

6 CONCLUSIONS 63

6.1 Future Work 63

REFERENCES 65

LIST OF TABLES

TABLES

Table 2.1 Possible outcomes for random variable X 8

Table 2.2 Example outcomes for a loaded die 11

Table 2.3 Probability of Symbols 13

Table 3.1 Prediction Modes for Lossless JPEG 19

Table 3.2 Prediction Modes for PNG 20

Table 5.1 Compression Results 49

Table 5.2 Encoding Times 50

Table 5.3 Decoding Times 51

Table 5.4 Time Durations of operations for Decoding Process 52

Table 5.5 Compression Results for Block-based Decoding 53

Table 5.6 Decoding Times L3C and Traditional Encoders (taken from [1]) . . 53

Table 5.7 Hyperparameters of the proposed model 54

Table 5.8 The Effect of different probability distribution functions 55

Table 5.9 The Effect of number of mixtures in GMM 55

Table 5.10 The Effect of channel size in CNN layers 56

Table 5.11 The Effect of Skip Connections 56

LIST OF FIGURES

FIGURES

Figure 1.1 Predictive lossless image compression scheme. The current pixel X in the original image is predicted by using its neighbor pixels A, B, C, and D. The error signal is calculated by subtracting the predicted value from the original pixel value X. Then, the error signal is fed to an entropy coder in order to generate a compressed bitstream. 2

Figure 1.2 Forward and reverse transform mappings between YCoCg and RGB color models (redrawn from [2]). 3

Figure 1.3 Lossless Image Compression System 4

Figure 2.1 Arithmetic coding example for sequence S=[s1,s3,s2] In table 2.3, probability values for all symbols are given. As it can be seen from the figure, we initialize the range in between [0.0,1.0) and the intervals for each symbol are determined due to their probabilities. After first symbol encoded, new range value becomes in between [0.0,0.5). Then, probability values are calculated again for the updated range. The whole process repeats itself until the end of the source sequence. 14

Figure 2.2 Receptive field of Convolutional Neural Network layers for filter size 3x3. 15

Figure 3.1 An illustration of Run-Length Coding Algorithm. The block of pixels are encoded with Run-Length coding method. 17

Figure 3.2 Encoded pixel and its neighbors. Most of the lossless algorithms first predict the pixels to be coded by using its already decoded neighbor pixels. In predictive schemes, all pixels are generated one at a time in the raster scan order. 18

Figure 3.3 The pixel x_i is conditioned on all the previous pixels which are left and above of x_i. 21

Figure 3.4 Illustration of generating a pixel by PixelCNN architecture. PixelCNN uses a region of neighbor pixels and maps the probability distribution to softmax function with 256 distinct values. (redrawn from [3]) 22

Figure 3.5 Left: A-Type Mask, Right: B-type Mask. The masking procedure can be done simply by zeroing the weight locations of future content. 23

Figure 3.6 Visualization of blind spot problem (above) and proposed solution with vertical and horizontal stacks (bottom). (reproduced from [3]) 24

Figure 3.7 Visualization of a single layer in the Gated PixelCNN architecture. The figure visualizes operation given in equation 3.5 Green blocks represents convolution operations and red blocks shows element-wise multiplications and additions. Blue represents combined operation of convolution with W_f and W_g. (reproduced from [3]) 25

Figure 3.8 Pixel grouping for 4x4 image. (redrawn from [4]) The generation order of pixels are different from other autoregressive generative models (not strictly above and left). 25

Figure 3.9 Visualization of L3C architecture (reproduced from [1]). There are three auxiliary feature representations and their probability distribution can be given to adaptive arithmetic coder for encoding. At the decoder side, the whole image can be generated sequentially with only three forward passes. 27

Figure 4.1 Visualization of Gaussian Mixture Model with number of mixture K=2. Red plot represents Gaussian distribution with $\mu_1 = 5$ and $\sigma_1 = 4$. Green plot represents Gaussian distribution with $\mu_1 = -5$ and $\sigma_1 = 4$. Blue plot represents mixture density of red and green Gaussian distributions with $\phi_1 = 0.6$ and $\phi_2 = 0.4$ 33

Figure 4.2 Probability Density Function (top) and Cumulative Distribution Function (bottom) of Gaussian Distribution with zero mean and unit variance. The probability of pixels can be computed at integer locations around bin width b. 34

Figure 4.3 The pixel to be encoded (red) and neighboring pixels $C_{i,j}$ (blue). To encode each pixel, a certain number of neighbor pixels are used. The future content is blocked using masks. 36

Figure 4.4 Network architecture of the proposed model. 37

Figure 4.5 The overall scheme for lossless image compression method and the proposed artificial neural network architecture. Neighboring pixels ($C_{i,j}$) of the current pixel are processed with the help of artificial neural network and the parameters which will be used in the Gaussian Mixture model are obtained. Then the pixel probability is calculated by using the Gaussian Mixture Model. 39

Figure 4.6 Encoding operation with arithmetic encoder 41

Figure 4.7 Decoding operation with arithmetic decoder 43

Figure 4.8 The sequential decoding order of pixels for an 8x8 block image. The numbers in figure indicate the decoding order of the pixels. All pixels are generated sequentially from left to right and top to bottom. Since there is no parallelization, the processing time will increase according to the resolution of the image to be decoded. 44

Figure 4.9 The parallel decoding order of pixels for the input filter size 5x5 with parallel decoding method [5]. This method is based on parallel decoding of each pixel whose conditional probability distribution model is independent of each other, according to the filter size used in input layer of the neural network. 45

Figure 4.10 The decoding order of pixels for the input filter size 5x5 with proposed diagonal decoding method. All pixels which are located on the diagonal can be obtained parallel. 46

Figure 4.11 Unavailable neighbor pixels (red pixels) for the diagonal parallel decoding algorithm. The green colored pixel represents the current pixel to be generated. Red pixels are assigned the nearest neighbor pixel value. 46

Figure 4.12 Block based decoding scheme (for the algorithm of Zhang et al.[5]). Each block is processed independently of each other. 47

Figure 5.1 The structure of skip connection 57

Figure 5.2 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 58

Figure 5.3 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 59

Figure 5.4 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 59

Figure 5.5 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 60

Figure 5.6 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 60

Figure 5.7 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 61

Figure 5.8 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 61

Figure 5.9 Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively 62

LIST OF ABBREVIATIONS

NN	Neural Network
ANN	Artificial Neural Network
YSA	Yapay Sinir Ağı
DCT	Discrete Cosine Transform
DWT	Discrete Wavelet Transform
bpp	bits per pixel
bpsp	bits per sub-pixel
KDE	Kernel Density Estimation
CDF	Cumulative Distribution Function
PDF	Probability Density Function ,
PMF	Probability Mass Function
GMM	Gaussian Mixture Model
GKM	Gauss Karışım Modeli
MLE	Maximum Likelihood Estimation
GIF	The Graphics Interchange Format
PDF	Portable Document Format
LZW	Lempel–Ziv–Welch
JPEG	Joint Photographic Expert Group
LOCO-I	Low Complexity Lossless Compression for Images
MED	Median Edge Detection
SAD	Sum of Absolute Differences
PNG	PNG Portable Network Graphics
WEBP	Web Picture format
CABAC	Context-adaptive binary arithmetic coding

FLIF	Free Lossless Image Format
MANIAC	Meta-Adaptive Near-zero Integer Arithmetic Coding
RNN	Residual Neural Network
CNN	Convolutional Neural Network
MS-PixelCNN	Multi Scale PixelCNN
LSTM	Long short-term memory
VAE	Variational Autoencoder
GMM	Gaussian Mixture Model
MLE	Maximum Likelihood Estimation
GPU	Graphics Processing Unit
CPU	Central Processing Unit

CHAPTER 1

INTRODUCTION

1.1 Lossless Image Compression

In recent years, the demand for high resolution pictures has increased in many areas, such as social media platforms, web applications, mobile phones, medical imaging, security industry, etc. Due to this demand, sensor and electronic technologies have been developed, and sensor resolution has increased. However, increased resolution also comes with a cost in terms of data samples that need to be processed and compressed. The advancement of technology in this regard has greatly increased the demand for better image compression. Many researchers and engineers are working on the development of algorithms and methods for better compression performance.

Image compression is the process of reducing the number of bits to store or represent an image, with or without preserving its original data. The spatial redundancy of the image signal is used to represent the image with fewer number of bits. In other words, compression process exploits the spatial correlation between pixels, allowing the image data to be represented with less number of bits compared to the uncompressed image. There are two types of image compression methods. These are lossy and lossless image compression. Lossy compression allows the decoded/reconstructed image to have some distortion compared to the original image in order to achieve better compression performance. It can achieve higher compression ratios even in situations where the human eye would not notice the distortion. Lossless compression, on the other hand, aims to reduce the number of bits to store the image while obtaining a decoded/reconstructed image that is identical (i.e. has no loss) to the original image.

Lossless compression is a method of compression that guarantees obtaining the orig-

inal image after decoding and is generally preferred in medical imaging, professional photography, satellite imaging, etc., i.e. in applications where data loss is not wanted. Traditional lossless compression methods can be categorized as pixel-by-pixel and invertible transform-based methods.

For a basic pixel-by-pixel lossless image compression system, pixel values are predicted from their surrounding pixels (left and above) using predefined modes in the encoder. The difference signal is obtained by subtracting the prediction signal from the original pixel value, and then the obtained error pixel is lossless compressed using an entropy coder.

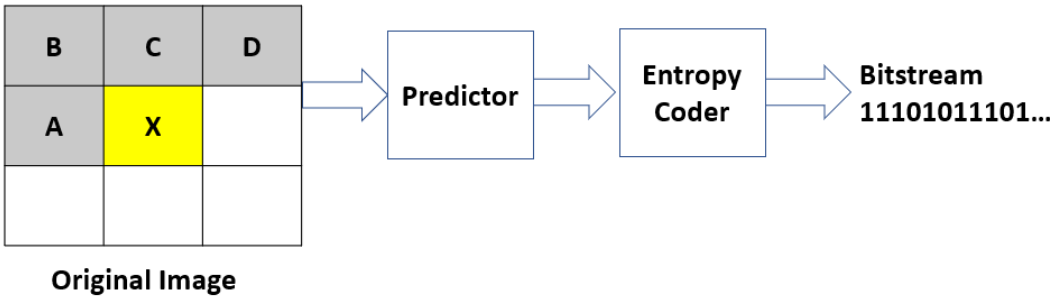

Figure 1.1: Predictive lossless image compression scheme. The current pixel X in the original image is predicted by using its neighbor pixels A, B, C, and D. The error signal is calculated by subtracting the predicted value from the original pixel value X. Then, the error signal is fed to an entropy coder in order to generate a compressed bitstream.

Transformation-based applications are widely used in lossy compression. These transforms are preferred in image compression as they provide good energy compaction and can allow progressive decoding. Most transforms, such as DCT (Discrete Cosine Transform) and DWT (Discrete Wavelet Transform), used in lossy compression are real-valued and a quantization process is required. Since quantization is an irreversible process, it can not be used in lossless compression methods. In the case of lossless compression, the transform to be used must be losslessly invertible and satisfy integer-to-integer mapping in order to obtain the original image without loss at the decoder side. In the literature, different approaches [7][8][9][10] have been proposed by researchers for the use of Wavelet transform in lossless compression.

In particular, the introduction of a lifting-based scheme [11] has introduced a simple method to obtain invertible integer-to-integer transforms for lossless compression.

Traditional encoders also use color space transformations to decorrelate the color channels of the image. For example, YCbCr is the commonly used color space in lossy compression and makes an important contribution in terms of coding gain. In this color space, the Y component represents the brightness channel, while the Cb and Cr components represent the chroma channels. YCoCg [2] (luminance + offset orange + offset green) is a color space that is especially proposed for lossless compression methods, and it allows a reversible transform between itself and the RGB color space. Figure 1.2 shows RGB to YCoCg transformation and its inverse.

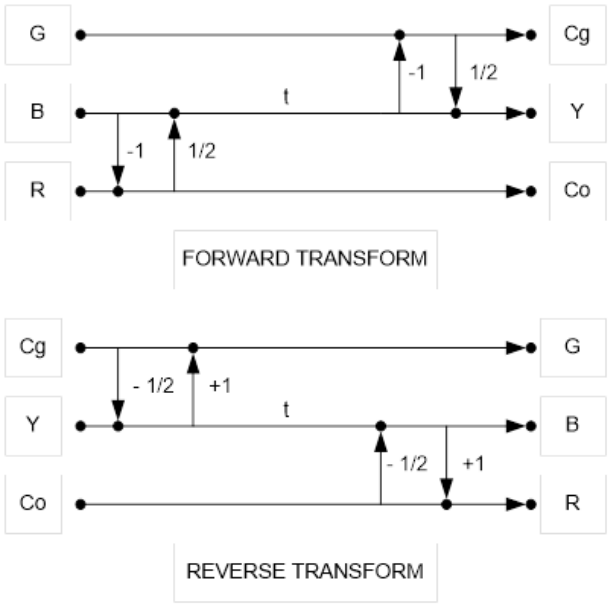

Figure 1.2: Forward and reverse transform mappings between YCoCg and RGB color models (redrawn from [2]).

1.2 Problem Definition for Lossless Compression

A compression system processes an original image with an encoding system (encoder) and generates a bitstream. For lossless compression, when the resulting bitstream is processed by a decoder, the reconstructed image is expected to be equal to the original image. Entropy coding methods are used to create a bitstream in an image

compression system. They enable the compression system to approximate the theoretical lower bound of entropy by taking advantage of the redundancy in the source. Entropy coding methods need symbols, generated by the encoder from the image, to be encoded and probability distribution functions of these symbols in order to generate a bitstream. Fewer bits are used for more probable symbols, and more bits are used for less probable symbols. The symbols given to the entropy encoder in image compression can be the original pixel, the difference signal calculated from the estimated pixel values, or the feature representation obtained by processing the image with an artificial neural network.

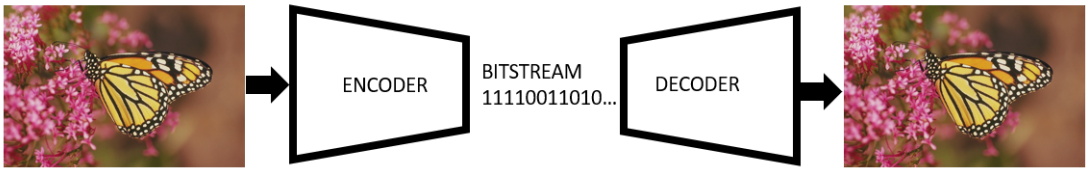

Figure 1.3: Lossless Image Compression System

It is important to learn the probability distribution of the symbols so that the entropy coder can compress efficiently, i.e. near the entropy bound. However, estimating the probability distribution of symbols requires careful design. With deep learning-based structures, it is possible to learn this distribution with satisfactory performance. In recent years, artificial neural networks have also been used in the field of lossless compression to increase the compression performance. Despite the fact that deep learning based encoders achieve lower compression rates compared to traditional encoders, they have longer encoding and decoding times due to the computational complexity of neural network based structures.

1.3 Contributions

Autoregressive generative models are an artificial neural network architecture whose main purpose is to learn the probability distribution of the training data and to generate new samples. In the studies where these models are used, the results are generally shared only on the log likelihood scores obtained for low resolution datasets such as MNIST [12], CIFAR10 [13], and Imagenet32/64 [14]. In order to use these models in

lossless image compression, a system combined with an entropy encoder is required.

In this thesis, a deep learning based lossless image compression method is presented.An autoregressive generative model-based neural network architecture with pixel-by-pixel processing is used. The Gaussian Mixture Model (GMM) is preferred to model the probability distribution function of the pixels. In this study, real compression results of the presented method were obtained with a test set prepared by the authors of [1] and compared with learning based methods and traditional engineering methods. Better compression results are obtained with a simple neural network that has a very low number of parameters compared to its competitors.

In autoregressive generative models, the image generation algorithm can not work in parallel because each pixel is dependent on the neighboring pixels before it. This means that the pixels of the entire image are generated sequentially in raster scan order and causes long decoding times on the decoder side. A parallelization algorithm [5] proposed in the literature has been implemented in order to optimize this problem. In addition, different parallelization approaches are presented and the results are analyzed.

1.4 The Outline of the Thesis

In Chapter 1, brief information about image compression is provided, and an overview of lossless image compression is presented. The problem definition for lossless compression is discussed. Then, the contributions of our method are explained.

In Chapter 2, key concepts like the entropy, probability density modelling, arithmetic coding etc. are discussed to make the frequently used terms in the thesis and the details of the study more understandable.

In Chapter 3, basic working principles and some essential details of the learning-based and non-learning lossless compression studies, are summarized. The state-of-the-art methods are analyzed and a perspective on the subject is provided.

In Chapter 4, the proposed method is explained in detail and the architecture of the artificial neural network is presented with illustrative figures.

In Chapter 5, the results of the study are presented with the state-of-the-art methods in the field of lossless compression, and the results are analyzed.

CHAPTER 2

BACKGROUND

2.1 Random Variable Basics

In probability theory, a random variable can be considered as a function that maps the outcome of a random experiment to numbers. They are usually expressed with capital letters. The set of possible outcomes, i.e. the sample space, is a countable set for discrete random variables and an uncountable set for continuous (general) random variables.

The probabilities of every possible outcome of a random variable form the probability distribution of that random variable. In other words, the distribution function can be used to calculate probabilities. For a discrete random variable, the probability distribution is also called Probability Mass Function (PMF).

PMF of a discrete random variable X can be defined as:

$$p_X(x) = p(x) = P(X = x). \tag{2.1}$$

where x represents possible values that the random variable X can take and P() represents probability. There are two important properties of probability mass function.

PMF is a non-negative function taking values between 0 and 1.

1) $p(x) \geq 0$.

The sum of all probability values of PMF must be equal to 1.

2) $\sum_x P(X = x) = 1$.

Random variable is a very basic concept that we can associate with many problems in our lives. For example, we can think of rolling a die. Let X be a random variable that represents the number showing on the die. This experiment has six possible outcomes (1,2,3,4,5,6) and all of them are equally probable. The probability values in table 2.1 present PMF values of each value that the random variable X can take.

Table 2.1: Possible outcomes for random variable X

Possible outcomes	Probability
X=1	1/6
X=2	1/6
X=3	1/6
X=4	1/6
X=5	1/6
X=6	1/6

Cumulative Distribution Function (CDF) represents the probability value of random variable X which is less than or equal to x. CDF of a random variable is defined as:

$$F_X(x) = P(X \leq x). \tag{2.2}$$

2.2 Density Estimation

In general, the probability distribution of a random variable to be processed should be known. If the probability density function of a random variable is known, one can easily decide on the distribution of the data and extract all the statistical information like mean, standard deviation, etc. But not all cases are as simple as rolling a die example, and one cannot obtain all possible outcomes or the exact probability distribution for this random variable. In this case, it may be necessary to estimate the probability density function for this random variable. There are two types of density estimation methods, parametric and non-parametric.

2.2.1 Non-parametric Density Modelling

Nonparametric estimation methods make no assumptions about the unknown distribution of the data. These methods directly use sample data to approximate the actual probability distribution. Sometimes parametric density models fail when the distribution of the data is relatively complex and can not be fit to a known probability function like multimodal distributions. Nonparametric models are more flexible than parametric models; hence there are no certain parameters, but they need more data for a good approximation, which may not always be available.

There are some methods for non-parametric density estimation. Histograms [15] are the most basic way of estimating the probability distribution of a random variable. According to the specific bin size, the data samples are placed in the estimated probability distribution. Bin size directly affects performance and can change the quality of density modelling. Another well-known non-parametric method is known as kernel density estimation (KDE) [16]. This method chooses a non-negative kernel function K, which is a certain probability distribution such as Gaussian, Exponential etc.

For example, X is a random variable with an unknown univariate distribution function and we want to estimate its probability distribution with kernel density estimation. Let samples of random variable X be $(x_1, x_2...., x_n)$. Then for the random variable X, the estimated distribution function can be defined as:

$$\widehat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big) \tag{2.3}$$

where K is the kernel function and h is smoothing parameter which controls the bandwidth of the kernel function.

2.2.2 Parametric Density Modelling

Given samples of data, the parametric model assumes that the unknown density function can be fit to a known probability distribution like Gaussian, Binomial, Logistic, etc. Assuming that a certain probability is suitable for the density estimation makes

things simpler. In addition, more satisfactory results can be obtained with less data compared to non-parametric models.

For example, the probability distribution of a random variable X is unknown, and it is assumed that the unknown distribution can be fit to a Gaussian distribution. Let observed samples of random variable X be $(x_1, x_2...., x_n)$. Since Gaussian distribution is used for the parametric model, two parameters, mean and variance, are needed. Using the observed examples, these parameters can be estimated as follows:

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.4}$$

$$\sigma^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \mu)^2 \tag{2.5}$$

where μ is sample mean and σ is sample variance. After finding the neccesary parameters we can calculate estimated probability distribution as:

$$\widehat{f}(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}. \tag{2.6}$$

Whether our fitting gives a good result should be analyzed by comparing it with the actual distribution.

2.3 Shannon's Entropy

The main objective of information theory is to transfer data from source to destination in the most efficient way possible. The term information entropy is introduced by Shannon [17] in 1948 and can be considered as a measure of the information a random variable can have. According to this theory, uncertain events contain more information and have higher entropy.

The entropy of a random variable X is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log p(x_i). \tag{2.7}$$

Here, $\log p(x_i)$ term actually refers to the self-information of an event x_i, and entropy can be considered as the average information content of the random variable X. For the digital world, we have bits whose value can be 0 or 1. Therefore, log base 2 is used in the formula to calculate entropy in bits.

For example, an experiment of rolling a fair die was mentioned in the section 2.1. It was seen that the probability distribution was uniform and each outcome had a probability of 1/6. In this case, entropy for this random variable can be calculated as:

$$H(X) = -6 * (1/6 * \log_2(1/6)) = 2.585\, bits. \tag{2.8}$$

But if we consider a loaded die, the situation will be different. Let Y be a random variable that represents the number showing on the loaded die and probability distribution function is as in the table 2.2.

Table 2.2: Example outcomes for a loaded die

Possible outcomes	Probability
Y=1	1/15
Y=2	1/15
Y=3	1/15
Y=4	10/15
Y=5	1/15
Y=6	1/15

Then entropy for the random variable Y can be calculated as:

$$H(Y) = -5 * (1/15 * log_2(1/15)) + 10/15(log_2(10/15)) = 1.692\, bits. \tag{2.9}$$

The intuition behind the difference between the theoretical entropy values calculated for the two cases can be explained by the varying uncertainties of different events. The fact that some events are more probable on a loaded die reduces uncertainty relatively and results in a lower entropy value compared to a fair die case.

2.4 Arithmetic Coding

Arithmetic coding [18] is an entropy coding algorithm which takes all symbols in the source sequence and maps them into a single number in between 0 and 1. The detailed pseudocode for the arithmetic encoding and decoding process is presented in algorithm 1.

Algorithm 1 ARITHMETIC ENCODING AND DECODING ALGORITHM [18]

```
/* ARITHMETIC ENCODING ALGORITHM. */
/* Call encode_symbol repeatedly for each symbol in the message. */
/* Ensure that a distinguished "terminator" symbol is encoded last, then */
/* transmit any value in the range [low, high). */
function encode_symbol (symbol,cum_freq)
    range = high - low
    high = low + range*cum_freq[symbol-1]
    low = low + range*cum_freq[symbol]
end
/* ARITHMETIC DECODING ALGORITHM. */
/* "Value" is the number that has been received. */
/* Continue calling decode_symbol until the terminator symbol is returned. */
function decode_symbol(cum_freq)
    Find symbol such that
        /* This ensures that value lies within the new */
        /* (low, high) range that will be calculated by */
        /* the following lines of code. */
        cum_freq[symbol] <= (value-low)/(high-low) < cum_freq[symbol-1]
    range = high - low
    high = low + range*cum_freq[symbol-1]
    low = low + range*cum_freq[symbol]
    return symbol
end
```

For the encoding operation, arithmetic encoder needs the cumulative distribution function to obtain the probability values of all symbols in the source sequence. According to the given function, non-overlapping intervals, also called ranges, are determined for each symbol. The range value is high for symbols with high probability. At the beginning of the encoding process, the range value is between 0 and 1 . After the first symbol is coded, the interval is updated according to the range value of that symbol, and this process continues for each symbol in the source sequence. As the symbols continue to be coded, this range gets narrower and finally, any value within this range can be used to encode the entire sequence. In figure 2.1, a simple example of the coding procedure of the sequence [s1,s3,s2] is illustrated according to the probability distribution function of symbols given in Table 2.3.

Table 2.3: Probability of Symbols

Symbol	Probability
s1	0.5
s2	0.2
s3	0.1
s4	0.1
s5	0.1

2.5 Performance Measure for Lossless Compression

For lossless compression systems, there are certain parameters that determine performance in terms of compression and complexity. Compression performance is measured by compression ratio or bit rate. The compression ratio is defined as the ratio of the original image size to the compressed image size. Bit rate refers to the average number of bits to represent a pixel in a digital image and is measured in bits per pixel (bpp) or bits per sub-pixel (bpsp). For a raw 24-bit RGB image, there are 8 bits in each sub-pixel (red, green, and blue) and 24 bits in total to represent a pixel value. So, the bpp value is 24 and the bpsp value is 8 for an uncompressed RGB image. The relation between the compression ratio and the bitrate is as follows:

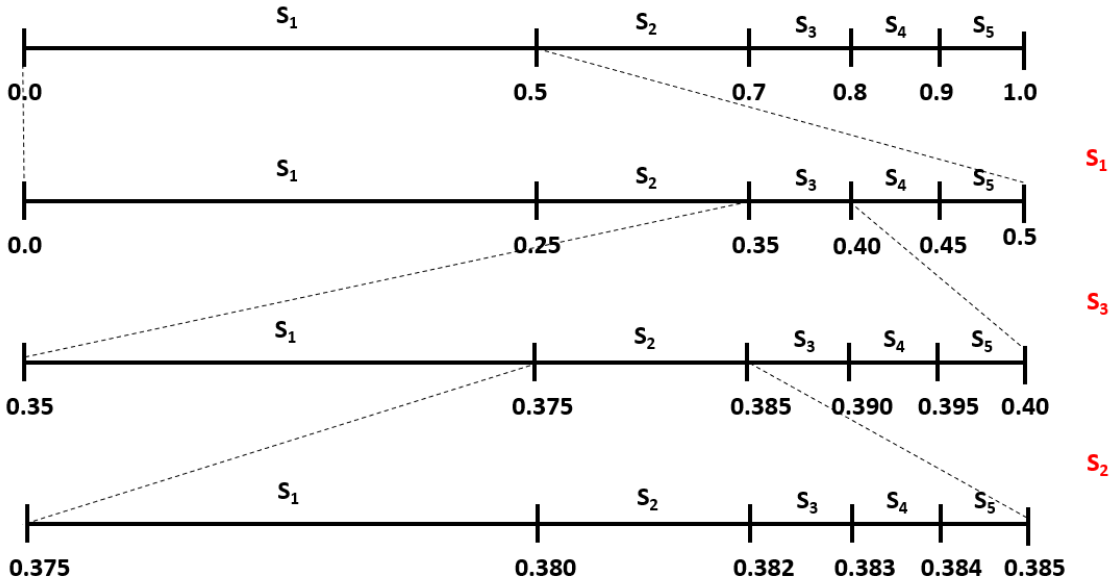

Figure 2.1: Arithmetic coding example for sequence S=[s1,s3,s2] In table 2.3, probability values for all symbols are given. As it can be seen from the figure, we initialize the range in between [0.0,1.0) and the intervals for each symbol are determined due to their probabilities. After first symbol encoded, new range value becomes in between [0.0,0.5). Then, probability values are calculated again for the updated range. The whole process repeats itself until the end of the source sequence.

$$\text{Compression ratio} = \frac{\text{Bit rate of the original image}}{\text{Bit rate of compressed image}} \tag{2.10}$$

The performance of image compression algorithms is measured against the entropy of the image. According to the source coding theorem, the entropy of the source sets a lower limit for the average code length per symbol L_{avg} as given below:

$$L_{avg} \geq \frac{H(X_1, X_2, ..., X_n)}{n}. \tag{2.11}$$

where $H(X_1, X_2, ..., X_n)$ is the joint entropy of the symbols $X_i (i = 1, 2, ...n)$. The lower bound of entropy for image compression depends on how the compression scheme models the joint probability distribution of all pixels in the image. For example, assuming that all pixels are independent in a special scenario, the joint entropy will be the sum of the individual entropies of the pixels. Encoding and decoding times

are also critical parameters for image compression systems because complex algorithms or neural network-based structures can result in impractical run times. Therefore, the main goal of lossless compression is to improve compression performance while providing acceptable encoding and decoding times for a given application.

2.6 Convolutional Neural Network (CNN)

A convolutional neural network is a type of artificial neural network architecture that is frequently preferred in deep learning, especially for image data. It is used for many problems, such as classification, face recognition, image captioning, etc. Convolutional neural networks preserve the spatial structure of the image (i.e., without flattening the input image like in fully connected networks). Basically, each layer has filters (weights and bias) of a certain size that are optimized over a loss function. The main operation is to calculate the dot products for each location by spatially shifting the filters over the input image. This process is done in every layer with CNN structures, and the activation map of the relevant layer is obtained as an output. The area that each neuron in the activation map can see at the input is called the receptive field, and it directly depends on the size of the filter. In fact, this concept is a hyperparameter and is directly equal to the size of the filter.

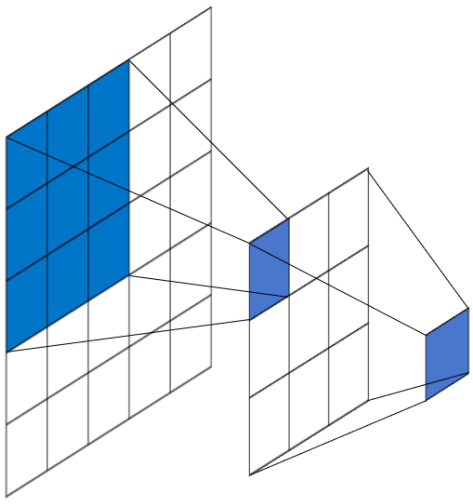

Figure 2.2: Receptive field of Convolutional Neural Network layers for filter size 3x3.

CHAPTER 3

RELATED WORK

3.1 Engineered Lossless Coders

There are several algorithms developed in the lossless compression field. Run length coding, LZW coding, and arithmetic coding are the most commonly used methods which are used by different encoders. Run length coding [19]is a compression method that combines a given pixel value and how many times that value continues uninterrupted into a symbol, which is then entropy coded. This compression method is quite simple and shows its greatest advantage in large homogeneous areas where the pixel values do not change.

12	12	12	12	12
12	12	12	12	12
12	12	12	9	9
9	9	9	10	10
11	11	11	11	11

RESULT: (12,13),(9,5),(10,2),(11,5)

Figure 3.1: An illustration of Run-Length Coding Algorithm. The block of pixels are encoded with Run-Length coding method.

Entropy coding methods try to achieve the lower bound of the entropy of the source. Arithmetic coding and Huffman coding [20] are the most commonly used entropy coding methods. A pixel value is considered as a symbol of an alphabet, and the

occurrence of the pixel value directly effects the length of the symbol. These methods reduce the bitrate by assigning fewer bits for more frequently used symbols and more bits for less frequently used symbols. While arithmetic coding offers good compression performance under many circumstances, Huffman coding is generally more advantageous in terms of run-time.

LZW [21] is a dictionary-based compression algorithm which is used in GIFs, PDFs etc. It uses a dictionary that includes codewords that represents the symbols in the bitstream. This dictionary is created by analyzing the source to be compressed. As the compression process continues, different codes are added to the dictionary. The LZW coding method aims to reduce the redundancy in the source by adding repeated or duplicate data combinations in the source to the dictionary.

Predictive coding makes prediction for the encoded pixel by using its neighbors. Instead of the exact pixel value, the difference between the original and predicted pixel is compressed by using an entropy coding algorithm. If the estimation is done well enough, significant bit rate savings can be achieved.

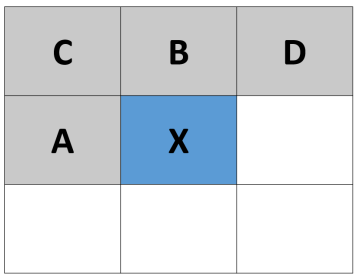

Figure 3.2: Encoded pixel and its neighbors. Most of the lossless algorithms first predict the pixels to be coded by using its already decoded neighbor pixels. In predictive schemes, all pixels are generated one at a time in the raster scan order.

JPEG [22] presents a lossless operation mode which uses a predictive coding scheme for compression. Neighboring pixels A, B, C and D are used to predict the current X pixel as shown in the figure 3.2 . There are eight predefined prediction modes which are generated as different linear combination of neighbor pixels.

On the other hand, JPEG-LS [23], is based on the LOCO-I (Low Complexity Lossless Compression for Images) algorithm and is similar to lossless JPEG in terms of

Table 3.1: Prediction Modes for Lossless JPEG

Mode Selection	Prediction
0	No prediction
1	A
2	B
3	C
4	A + B – C
5	A + (B – C)/2
6	B + (A – C)/2
7	(A + B)/2

operation. It increases compression performance by using an enhanced decorrelation process with different prediction modes and improved error modelling. Instead of only linear estimators used in lossless JPEG, the context model has been further developed and a nonlinear estimation structure called the MED estimator (Median Edge Detector) has been used. The predictor chooses between three estimation modes depending on edge information from the relationship between neighboring pixels.

$$X = \begin{cases} \min(A, B) & \text{if } C \geq \max(A, B) \\ \max(A, B) & \text{if } C \leq \min(A, B) \\ A + B - C & \text{otherwise.} \end{cases} \tag{3.1}$$

PNG [24] has a preprocessing step which performs different filtering options for each row. The filtering process uses the neighbor pixels and this reduces the bit rate significantly. For each line, five different filters are available, and the filter which minimizes the sum of absolute differences (SAD) can be selected. To generate a bitstream, PNG uses an algorithm named deflate, which can be considered as a specific combination of dictionary-based LZ77 and Huffman coding.

WEBP [25] is an encoder developed especially for web pages, offering both lossy and lossless compression options. Spatial redundancy in the image is exploited using a predictive coding scheme. The whole picture is divided into square blocks, and one

Table 3.2: Prediction Modes for PNG

Filter Name	Prediction
None	No prediction (Zero)
Sub	A
Up	B
Average	(A+B)/2 (rounded down)
Paeth	A, B, or C, whichever is closest to p = A + B - C

of the 13 estimators is used. After the estimation step, it applies different transformations for decorrelation of color channels and to achieve more efficient compression. An algorithm based on LZ77 is used in the encoding phase.

FLIF (Free Lossless Image Format) [26] is the best performing encoder in terms of compression among non-learning based methods. A reversible YCoCg color transformation [2] is applied to provide decorrelation between color channels. FLIF uses an entropy coding method called MANIAC (Meta-Adaptive Near-Zero Integer Arithmetic Coding) based on CABAC (Context-adaptive binary arithmetic coding). This method provides a powerful context modeling by adapting to the data. Although it offers strong compression performance, it has relatively longer run times than other traditional methods.

3.2 Learning Based Lossless Compression

3.2.1 Autoregressive Generative Models

The most important function of generative models is to learn the distribution of the training data in an unsupervised way. Mainly, they produce an explicit and tractable probability distribution based on the training data. These methods are commonly used in deep learning and have a wide range of usage, like density modelling, image inpainting, etc. Inıtially, they were not developed for compression purposes, but they can be used with an entropy coder for lossless compression.

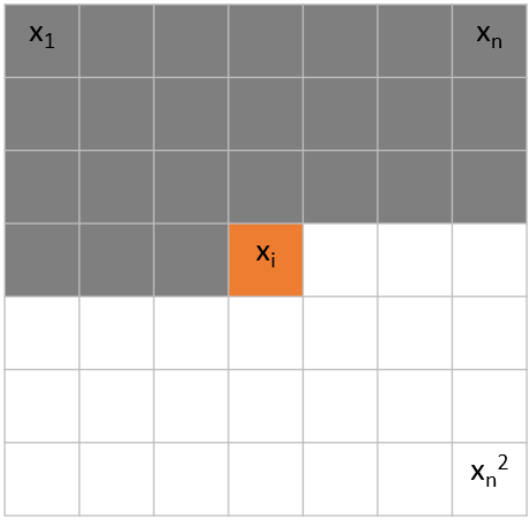

Figure 3.3: The pixel x_i is conditioned on all the previous pixels which are left and above of x_i.

The probability distribution function of the image is formulated as the joint probability distribution of its pixels. For a nxn image x which has total number of n^2 pixels (row by row), the joint distribution p(x) can be defined as follows:

$$p(x) = p(x_1, x_2, ..., x_{n^2}). \tag{3.2}$$

The joint distribution p(x) can be rearranged in terms of conditional distributions over the pixels with the help of chain rule in probability theory as below [27]:.

$$p(x) = \prod_{i=1}^{n^2} p\left(x_i \,|\, x_1, x_2....., x_{i-1}\right) \tag{3.3}$$

where x_i represents the single pixel of image x. All pixels in the image are generated pixel by pixel in a sequential manner. The probability distribution of each pixel is conditioned to the values of all previous pixels while the last pixel depends on all previous pixel values. This is the reason why these network structures are called autoregressive because probability model uses past values while predicting current value. PixelRNN [27], PixelCNN [3], and MS-PixelCNN [4] are well-known generative network architectures.

The pixel distribution is jointly determined by its color channels red (R), green (G) and blue (B). Green channel of the image is conditioned to the red channel, while the blue channel is conditioned to both red and green channels. The probability distribution of pixel x_i can be defined in terms of the conditional probability of the subpixels as follows [27].

$$p(x_i \mid x_{<i}) = p(x_{i,R} \mid x_{<i})p(x_{i,G} \mid x_{i,R}, x_{<i})p(x_{i,B} \mid x_{i,R}, x_{i,G}, x_{<i}). \tag{3.4}$$

The modelling of Equation 3.3 with a deep learning based solution requires long-term dependencies. PixelRNN models long dependencies by using two dimensional LSTM network structure. In the architecture, there are residual skip connections in order to improve training process. At the output layer, 256-way softmax function is used to obtain discrete probability distributions for each channel. The network optimizes log-likelihood as a loss function. The biggest disadvantage of Pixel RNN network is that the training process is slow because of the states in the LSTM layers which can not be parallelized.

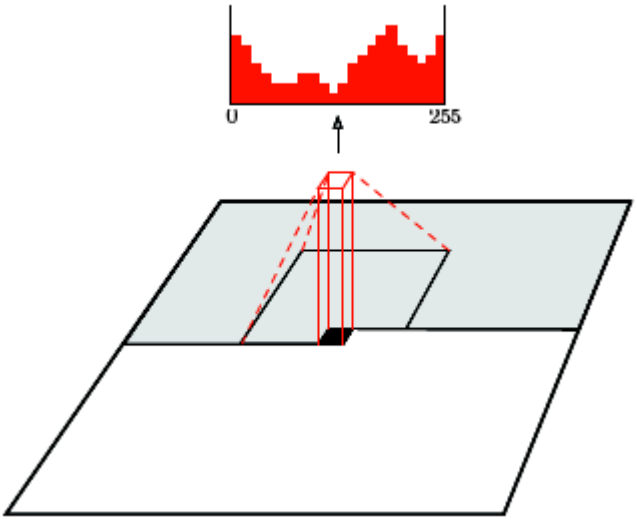

Figure 3.4: Illustration of generating a pixel by PixelCNN architecture. PixelCNN uses a region of neighbor pixels and maps the probability distribution to softmax function with 256 distinct values. (redrawn from [3])

PixelCNN offers a CNN based architecture to speed up training. It reduces compu-

tational complexity and provides a bounded receptive field by using certain neighbor pixels. The range of neighboring pixels to be used for the probability distribution of the pixel is controlled by structures called masked convolutions. The masks in the architecture are arranged to see only previous context. There are two mask structures called A and B. Mask A, is only used in the first CNN layer and breaks all connections with the current pixel to be generated. On the other hand, mask B allows to use current pixel information thorough the rest of the model.

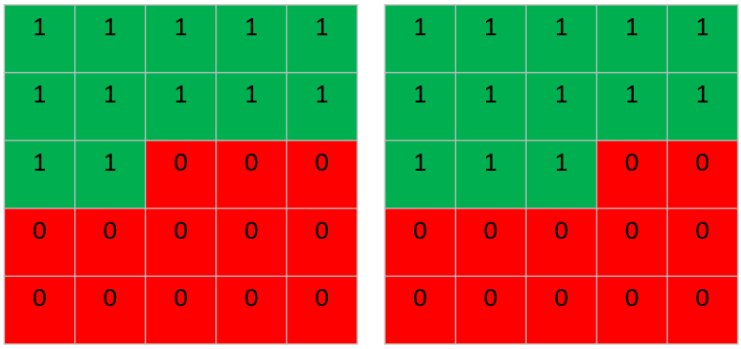

Figure 3.5: Left: A-Type Mask, Right: B-type Mask. The masking procedure can be done simply by zeroing the weight locations of future content.

The mask structures used to provide the causality principle in the PixelCNN architecture cause blind spot problem (see figure 3.6) in the receptive field as they mask the right and bottom (future) pixels. In other words, the generated pixels can not use all of the past pixel values to process the current pixel. This problem cannot be solved by adding more layers, on the contrary, the blind spot area grows for each added layer. PixelRNN outperforms the PixelCNN architecture thanks to the spatial LSTM layers it uses and the ability to process all past pixels.

Aäron van den Oord et al. [3] proposed the Gated PixelCNN architecture to overcome these problems. The improvements made are as follows:

- In order to further improve the performance of PixelCNN and to model more complex structures, gated activation units have been proposed instead of the ReLU activation function between masked convolution layers.

$$y = \tanh(W_{k,f} * x) \odot \sigma(W_{k,g} * x). \tag{3.5}$$

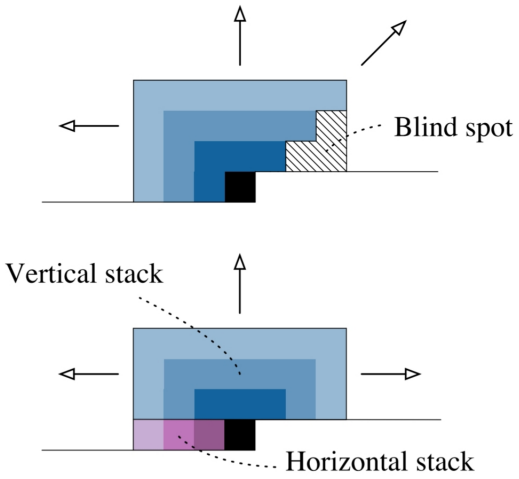

Figure 3.6: Visualization of blind spot problem (above) and proposed solution with vertical and horizontal stacks (bottom). (reproduced from [3])

where σ is the sigmoid function, k is the number of the layer, $\odot$ is the element-wise product and * is the convolution operator.

- They proposed gated convolutional layer structures (see figure 3.7) to solve the blind spot problem. By combining horizontal and vertical stack structures in these layers, the loss in the receptive area caused by the mask is eliminated. With the horizontal stack, it is ensured that the current pixel can be conditioned to all neighboring pixels on the same line and to the left of it. The vertical stack, on the other hand, aims to be conditioned by accessing all rows above the current pixel.

Although PixelCNN increases training speed, both of PixelCNN and PixelRNN are slow during generation of pixels due to sequential nature. Increasing the generation speed is only possible by parallelizing this process, which can be done by assuming that some pixel groups are independent of each other. MS-PixelCNN presents a parallellized version of PixelCNN by assuming group of pixels are independent. It reduces the computational cost significantly by allowing acceptable bit overhead. The joint probability of image x which is divided into G groups of T pixels can be defined as[4]:

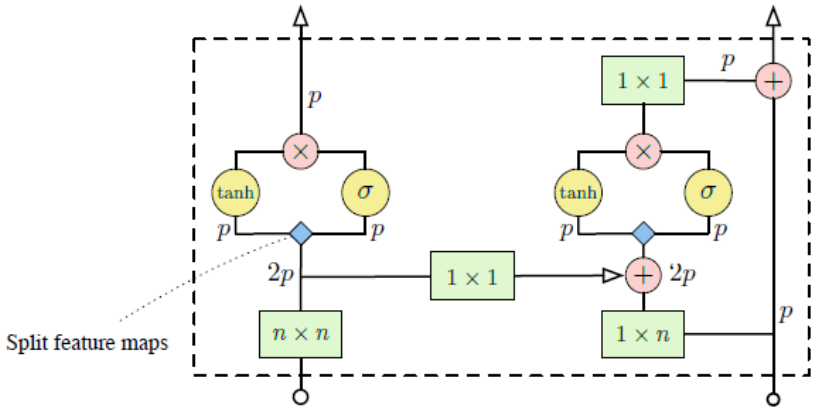

Figure 3.7: Visualization of a single layer in the Gated PixelCNN architecture. The figure visualizes operation given in equation 3.5 Green blocks represents convolution operations and red blocks shows element-wise multiplications and additions. Blue represents combined operation of convolution with W_f and W_g. (reproduced from [3])

$$p(x_{1:T}^{1:G}) = \prod_{g=1}^{G} p\left(x_{1:T}^{(g)} \mid x_{1:T}^{(1:g-1)}\right). \tag{3.6}$$

The main working principle is based on generating upper scale image conditioned on the lower scale image. For each scale, image is divided into 2x2 pixel groups and the pixel grouping scheme is organized so that maximum available neighbor pixels are used. Given the first scale, the whole image can be generated in a autoregressive manner.

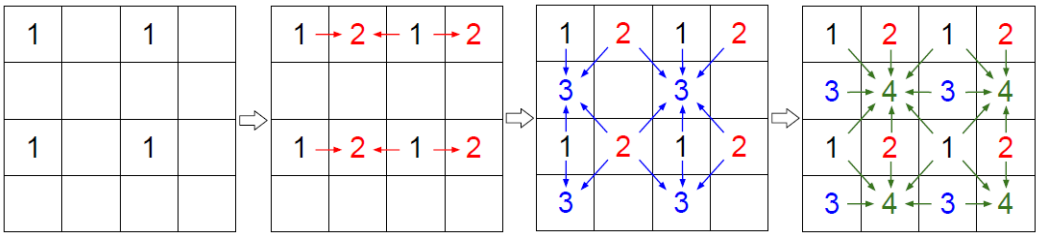

Figure 3.8: Pixel grouping for 4x4 image. (redrawn from [4]) The generation order of pixels are different from other autoregressive generative models (not strictly above and left).

PixelCNN++ [28] proposed some changes to enhance PixelCNN's performance and

achieve a more flexible architecture. Instead of a 256-way softmax function,discretized logistic mixture likelihood is used for the pixel probability distribution. They also implemented shortcut connections in their architecture and used dropout for regularization. PixelCNN defines the pixel probability in terms of factorized conditional probabilities of sub-pixels. On the other hand, PixelCNN+ argues that there is no need for a deep neural network to model the dependency between color channels. So, generation of subpixels is simplified by conditioning on whole pixels. PixelCNN++ uses convolutions with stride 2 in order to increase receptive field. Loss of information coming from the strides is compensated with short-cut connections they proposed.

3.2.2 Multi-scale Autoregressive Lossless Compression

Autoregressive generative models generate pixels in the order from left to right and top to bottom. The generation of each pixel depends on the value of the previous generated pixel. Therefore, the pixels have to be produced sequentially and the generation algorithm can not work in parallel. As a practical method to avoid this disadvantage, L3C [1] proposes a multi-scale compression scheme. The probability distribution of pixels is conditioned on a separate auxiliary representations obtained by a neural network rather than being conditioned on previous pixels. The joint distribution of image x and feature representations is defined as[1]:

$$p(x, z^{(1)}, z^{(2)}, ..., z^{(S)}) = p(x|z^{(1)}, z^{(2)}, ..., z^{(S)}) \prod_{s=1}^{S} p\left(z^{(s)}|z^{(s+1)}, ..., z^{(S)}\right). \quad (3.7)$$

where $z^{(s)}$ represents the learned or hand designed feature representations for each scale $s = 1, 2, ..S$. Probability distribution $p(z^{(S)})$ of last feature representation is assumed to be uniform distribution and independent from other auxiliary features.

The feature representation $z^{(s)}$ can be obtained by using feature extractor $F^{(s)}$ which can be represented as a combination of a scalar differentiable quantizer and feature extractor block $E^{(s)}$ as below [1].

$$z^{(s)} = F^{(s)}(x), \quad (3.8)$$

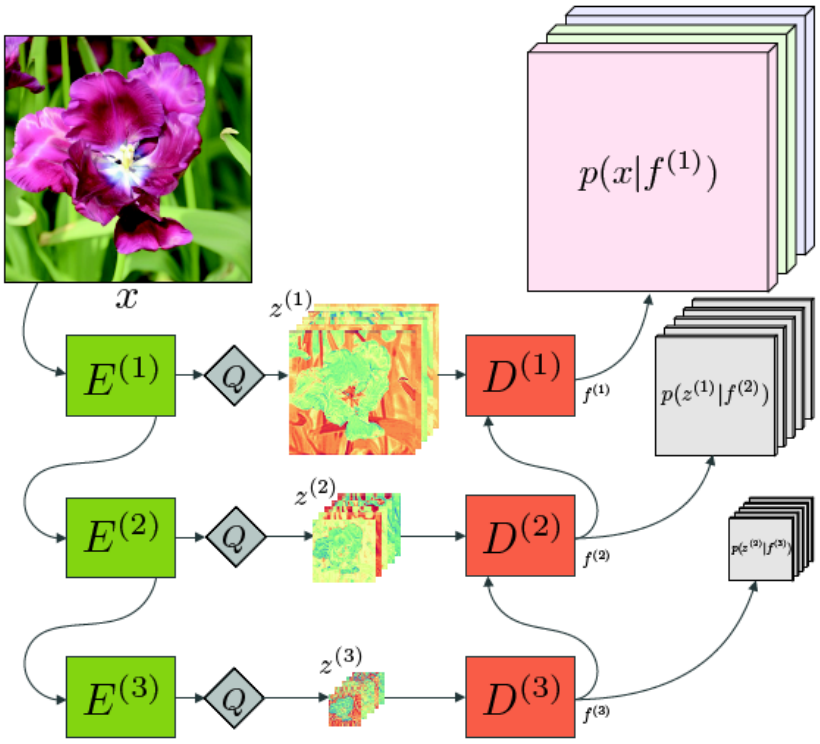

Figure 3.9: Visualization of L3C architecture (reproduced from [1]). There are three auxiliary feature representations and their probability distribution can be given to adaptive arithmetic coder for encoding. At the decoder side, the whole image can be generated sequentially with only three forward passes.

$$F^{(s)} = Q\, o\, E^{(s)}\, o.... o\, E^{(1)}. \tag{3.9}$$

The first feature representation $z^{(1)}$ is generated by feeding the original image through the first feature extractor $F^{(1)}$. The resulting auxiliary representation (before quantization) is used as a input for the next scale. For encoding operation, probability distribution for all auxiliary representations need to be calculated. The conditional distributions is parametrized for all scales by using predictor $D^{(s)}$ as below [1].

$$p(z^{(s)}|z^{(s+1)}, ..., z^{(S)}) = p(z^{(s)}|f^{(s+1)}), \tag{3.10}$$

$$f^{(s)} = D^{(s)}(f^{(s+1)}, z^{(s)}). \tag{3.11}$$

There are some similarities between PixelCNN++ architecture mentioned in the previous section. L3C also uses discretized logistic mixtures for the modelling of probability distribution. The number of feature representations is chosen as S=3. Firstly, the third scale is calculated from previous feature extractor and encoded by using uniform distribution function. For the upper scales, the probability distribution is calculated by conditioning on the previous feature representations. The bitstream is generated sequentially as the probability distributions of the scales are obtained.

At the decoder side, the last scale $z^{(S)}$ is decoded by using uniform distribution. Since, predictor $D^{(S)}$ is available at the decoder, the probability distribution of the upper scale $z^{(S-1)}$ can be calculated. Then, decoder side generates the feature representations according to given probability distribution. This way, only one feature representation is calculated for each forward pass until original image is obtained. Although this method offers practical encoding and decoding times, it lags behind autoregressive generative models in terms of compression performance.

3.2.3 Flow-based Compression

As a well-known generative model, flow-based approaches [29] [30] [31] learn an invertible and stable mapping function that converts an input image into a latent representation with a predefined distribution function. IDF (Integer Discrete Flows) [31] obtains an integer latent representation that is invertible. Compression performance of IDF is superior to conventional methods, but it cannot provide practical decoding times for high resolution images. In order to perform lossless compression, the mapping function must be invertible and bijective. Assuming f is a function which takes an observed data variable x $\sim p_X$ and maps it to a latent variable z $\sim p_Z$ in the latent space, the relation between them can be defined as:

$$z = f(x). \tag{3.12}$$

Let probability distribution of latent variable be a predefined function $p_Z(z)$. By changing of variables, probability distribution of x can be obtained as in the following[29].

$$p_X(x) = p_Z(f(x)) \left| \det \frac{\partial f(x)}{\partial x^T} \right|. \tag{3.13}$$

where the partial derivative term can be expressed as Jacobian of f at x.

Flow-based methods learns data distribution explicitly and optimizes the negative log likelihood function. The invertible function f is modeled by an artificial neural network. For the encoding operation, entropy coder takes the latent representation and probability distribution $p_Z(z)$ to generate bitstream. Decoder side takes bitstream and $p_Z(z)$ to obtain latent representation z. Then the original image can be reconstructed by applying the inverse of the encoding function to z.

CHAPTER 4

PROPOSED METHOD

4.1 Overview

In this thesis, a pixel-based autoregressive and learned lossless compression method is presented. The proposed end-to-end learned method is designed without using complex neural network structures. Gaussian Mixture probability model is used to model the probability density function of the pixels, and a simple neural network architecture is presented to provide a practical approach. In addition, the proposed parallel decoding algorithm [5] for autoregressive based models is applied to optimize the decoding time. Also, alternative parallel decoding methods are introduced and the results are compared.

In this section, theoretical background information about the thesis work is presented. First, the pixel probability model and the Gaussian mixture model are explained in detail. The selection of hyper-parameters, datasets, loss function, and optimization of the presented architecture are shared. Finally, the solutions for time optimization on the decoder side are explained.

4.2 Gaussian Mixture Model (GMM)

The Gaussian Mixture Model is a well-known unsupervised method which can be used for data clustering or density estimation. Recently, Gaussian Mixture Model has been preferred in the field of image compression as it can provide a flexible and accurate entropy model [32] [33]. It can be considered as a parametric estimation method that assumes the probability density function of a random variable can be

represented as a linear combination of different Gaussian distributions.

The parameters needed for a univariate Gaussian Mixture Model (GMM) can be explained as:

K = number of mixture components
$\pi_{i=1...K}$ = weight of component i
$\mu_{i=1...K}$ = mean of component i
$\sigma_{i=1...K}$ = standard deviation of component i

Each Gaussian in the univariate model has its own mean and standard deviation, and it is multiplied by a weight value. According to the parameters mentioned above, a univariate Gaussian Mixture Model of the number of K Gaussians can be defined as:

$$p(x) = \sum_{i=1}^{K} \pi_i \mathcal{N}(x|\mu_i, \sigma_i). \tag{4.1}$$

where

$$\mathcal{N}(x|\mu_i, \sigma_i) = \frac{1}{\sigma_i\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_i}{\sigma_i}\right)^2}. \tag{4.2}$$

and the weight parameters sum to 1 by satisfying equation:

$$\sum_{j=1}^{K} \pi_j = 1. \tag{4.3}$$

In order to fit the probability distribution, the parameters of the Gaussian Mixture Model must be estimated. Maximum Likelihood Estimation (MLE) is the most common way of estimating the parameters of a model. For the mixture case, MLE can not provide a closed form solution. So, an iterative type of maximum likelihood estimation algorithm called expectation-maximization is used frequently for the GMM model. As the name suggests, the algorithm consists of two steps. The expectation step introduces hidden (latent) variables and tries to maximize the posterior probability, while the maximization step updates the parameters of the model. This procedure continues until parameters converge to a value. In this thesis, the parameters of Gaussian Mixture Model are obtained by using a neural network and a loss function that expresses the entropy of the image data under a GMM model.

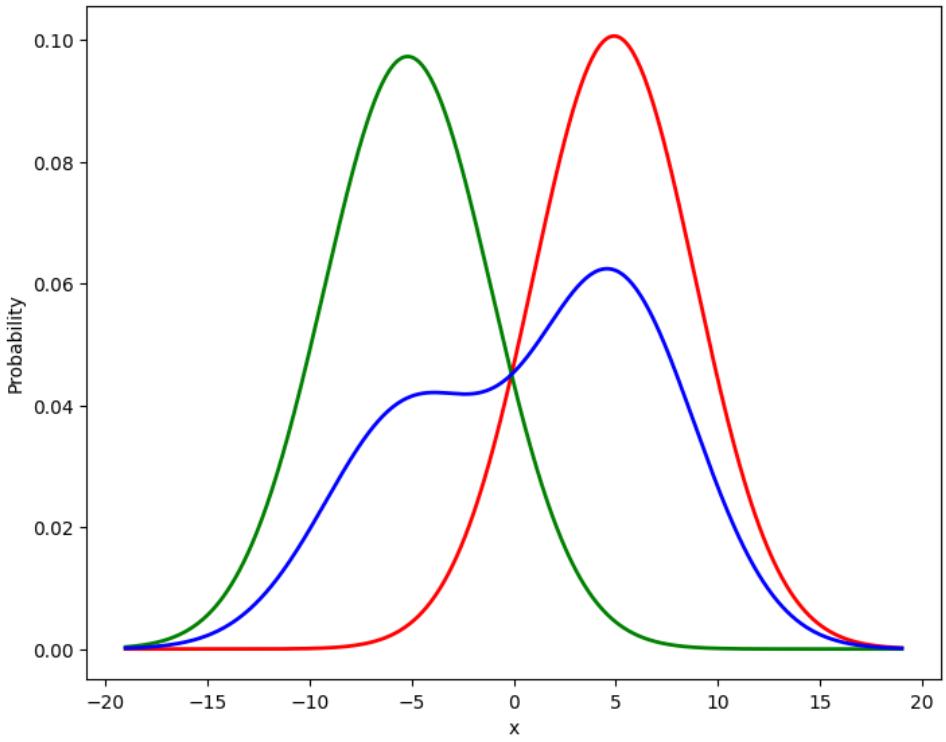

Figure 4.1: Visualization of Gaussian Mixture Model with number of mixture K=2. Red plot represents Gaussian distribution with $\mu_1 = 5$ and $\sigma_1 = 4$. Green plot represents Gaussian distribution with $\mu_1 = -5$ and $\sigma_1 = 4$. Blue plot represents mixture density of red and green Gaussian distributions with $\phi_1 = 0.6$ and $\phi_2 = 0.4$

4.3 Discretized Mixture Model

PixelCNN++ [28] uses a univariate continuous logistic distribution for computing the conditional probability of pixels. Using a parametric and continuous probability model provides a more efficient training process.

For lossless compression, probability values need to be calculated for integer pixel values. For example, let $f_X(x)$ be a probability density function of continuous Gaussian distribution with mean μ and standard deviation σ. The probability mass value at the integer value $\hat{x}$ can be calculated within a given bin width 2b, where b=0.5, as follows:

$$P(\hat{x}) = \int_{\theta=\hat{x}-b}^{\theta=\hat{x}+b} f_X(\theta)\, d\theta. \tag{4.4}$$

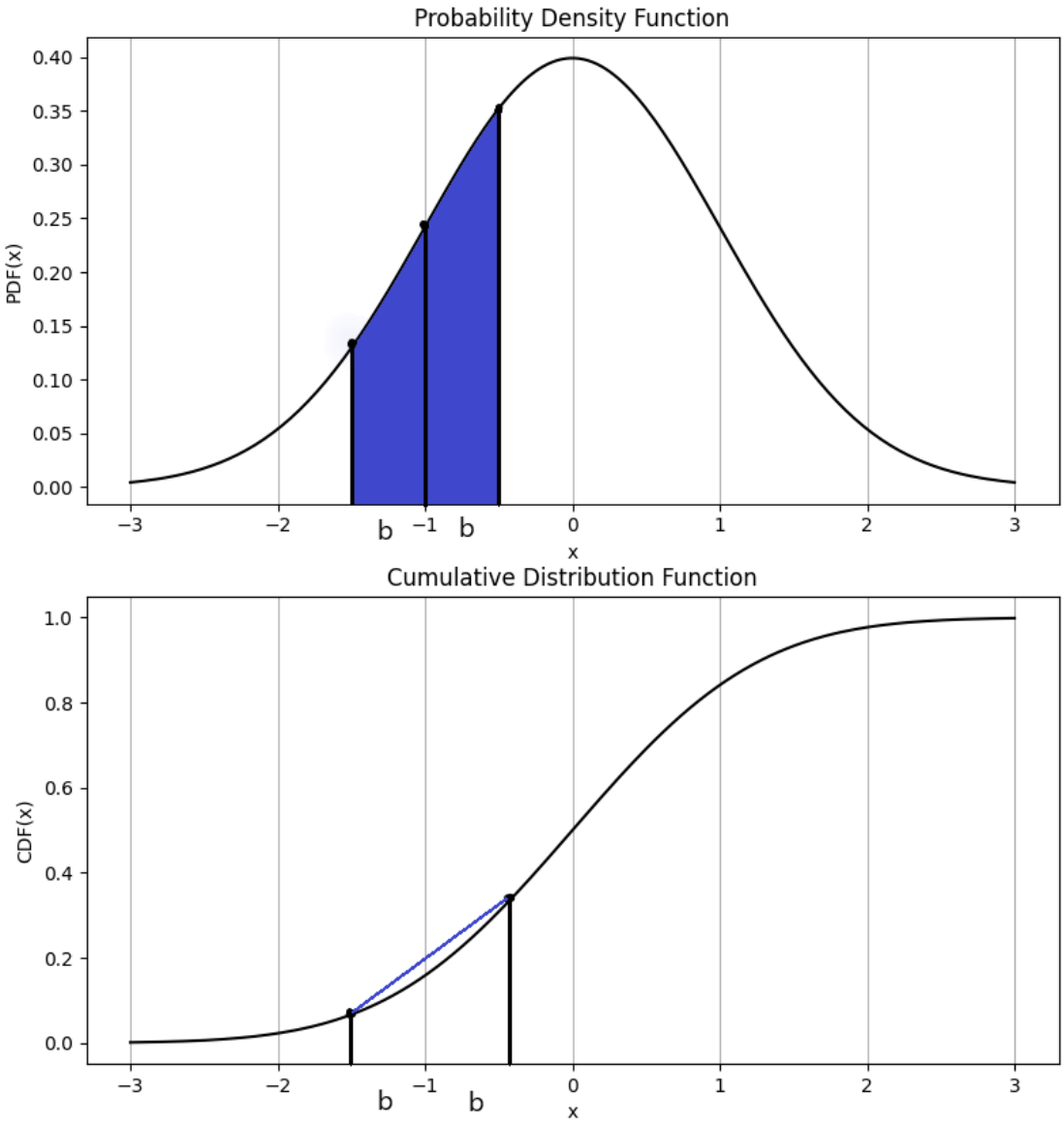

Figure 4.2: Probability Density Function (top) and Cumulative Distribution Function (bottom) of Gaussian Distribution with zero mean and unit variance. The probability of pixels can be computed at integer locations around bin width b.

In order to update the parameters of the neural network, the loss function must be differentiable for the training process. Since log probabilities will be used in the loss function of the artificial neural network, it will be an easier approach to write the calculation of probability mass values in terms of the cumulative distribution function as below.

$$P(\hat{x}) = F_X(\hat{x} + b) - F_X(\hat{x} - b) \tag{4.5}$$

where $F_X(.)$ is cumulative distribution function of random variable X.

4.4 The Probability Model of Pixels

In lossless image compression, the entropy encoder needs the probability distribution function of the current pixel to generate a bitstream. PixelCNN [3] obtains the probability distribution function of the pixels by using the 256-way softmax function as mentioned before. However, this is an approach that can be disadvantageous in terms of both memory and training performance. In addition, this approach does not take into account the probabilistic relationship between pixels whose values are close to each other. In recent studies, parametric density estimation methods have been frequently preferred for the pixel probability model, and satisfactory results have been obtained in terms of speed and compression rate. In this study, the conditional probability mass function of each pixel is obtained using a Gaussian Mixture Model. The conditional probability distribution function of each sub-pixel $x \in \{r, g, b\}$ is defined as [28]:

$$P(x|\pi, \mu, \sigma) = \sum_{i=1}^{K} \pi_i [F((x + 0.5 - \mu_i)/\sigma_i) - F((x - 0.5 - \mu_i)/\sigma_i)] \tag{4.6}$$

where $F()$ is the cumulative density function of the Gaussian distribution with mean μ_i and standard deviation σ_i. The Gaussian mixture model consists of K Gaussian distribution functions, and each has a weight parameter of π_i. A pixel at location

(i,j) in the image consists of sub-pixels $r_{i,j}$ (red), $g_{i,j}$ (green) and $b_{i,j}$ (blue). The probability of a pixel is represented as the joint probability of these three channels.

$$\begin{aligned} P(r_{i,j}, g_{i,j}, b_{i,j}|C_{i,j}) = P(r_{i,j}|\pi_r(C_{i,j}), \mu_r(C_{i,j}), \sigma_r(C_{i,j}))\times \\ P(g_{i,j}|\pi_g(C_{i,j}), \mu_g(C_{i,j}, r_{i,j}), \sigma_g(C_{i,j}))\times \\ P(b_{i,j}|\pi_b(C_{i,j}), \mu_b(C_{i,j}, r_{i,j}, g_{i,j}), \sigma_b(C_{i,j})) \end{aligned} \tag{4.7}$$

where $C_{i,j}$ represents a region containing neighboring pixels (left and top) of the pixel positioned at (i,j) location (blue region in 4.3). $\pi_r(C_{i,j}), \mu_r(C_{i,j})$ and $\sigma_r(C_{i,j})$ represent the parameters produced by the artificial neural network processing the $C_{i,j}$ pixels and which will be used in the equation 4.7.

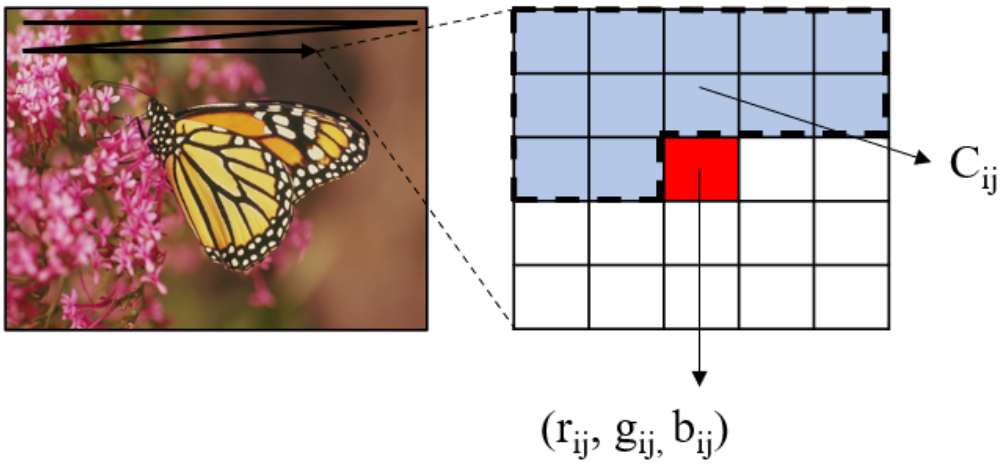

Figure 4.3: The pixel to be encoded (red) and neighboring pixels $C_{i,j}$ (blue). To encode each pixel, a certain number of neighbor pixels are used. The future content is blocked using masks.

$$\mu_g(C_{i,j}, r_{i,j}) = \mu_g(C_{i,j}) + \alpha(C_{i,j})r_{i,j} \tag{4.8}$$

$$\mu_b(C_{i,j}, r_{i,j}, g_{i,j}) = \mu_b(C_{i,j}) + \beta(C_{i,j})r_{i,j} + \gamma(C_{i,j})b_{i,j}. \tag{4.9}$$

As seen in equation 4.8 and 4.9 (taken from [28]), the average values of the green and blue sub-pixels are updated according to the previous sub-pixels. The α, β and γ parameters used for this update are also learned by the artificial neural network. In

this way, the necessary parameters for the probability distribution model of all three sub-pixels are calculated in one pass of the artificial neural network, and three passes are not required as in the case of PixelCNN.

4.5 Network Architecture and Optimization

The proposed neural network has a simple architecture with 5 layers (see figure 4.4). In the first layer, 5x5 A-type masked convolution [27] is used. The A-type mask ensures that only previous (i.e. left and top) pixels are used , and the resulting conditional probability distribution function conditions on only previous pixels. There are 128 filters in each CNN layer, and Leaky ReLU is used as activation function.

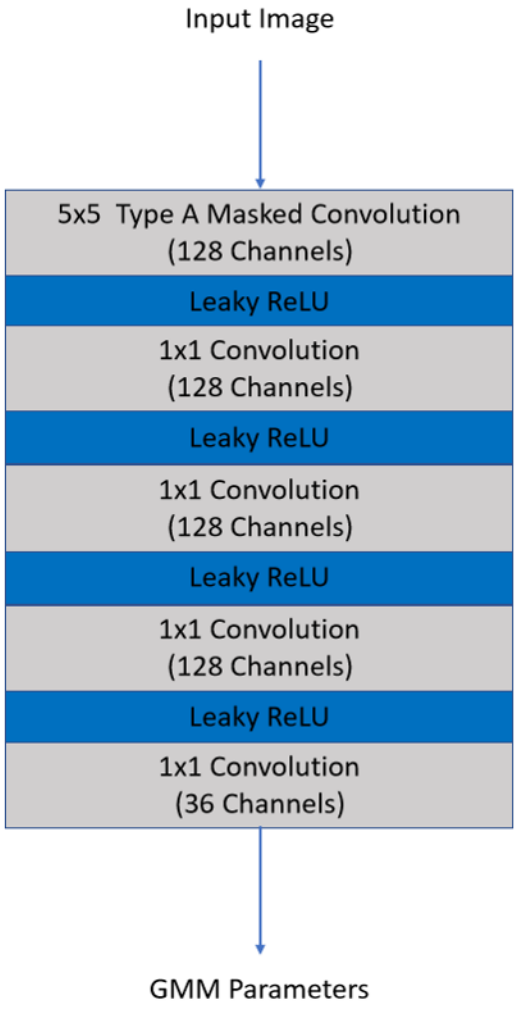

Figure 4.4: Network architecture of the proposed model.

We trained our model with 16x16 block images cut from the L3C training dataset [1]. While creating our own dataset, image patches were created using at least 10 percent of each image. The training and evaluation datasets of the L3C authors were obtained using the Open Images [34] datasets by eliminating high saturation/non-photographic images. We used the Adam optimizer for optimization with a batch size of 64. The learning rate is initialized as 0.0001 and updated by multiplying with 0.95 after every five training epoch. Pytorch [35] framework is used in both training and testing processes.

For the training of the proposed neural network structure, we minimized the average negative log-likelihood function, which gives the entropy of the random variables. The loss function is given in equation 4.10.

$$L = -\frac{1}{N}\sum_{i,j}\log_2 P(r_{i,j}, g_{i,j}, b_{i,j}|C_{i,j}) \tag{4.10}$$

4.6 Real Compression

During the training process, the neural network is trained to minimize the average negative log likelihood of the training data. In real compression, symbols are encoded using the cumulative distribution functions obtained with the trained neural network. Thus, the compression performance of the neural network can be measured in bits per sub-pixel (bpsp). In our case, symbols represent the original pixel values of the image to be coded.

In this study, an adaptive arithmetic coder is used for entropy coding. An arithmetic coder basically takes the symbol sequence and cumulative distribution function to generate a bitstream as discussed in 2.4. The term "adaptive" means that different probability distribution functions can be used for each symbol. There is a specific case where the probability of a symbol being coded can be zero. For the lossless case, we need to be sure that every symbol has a non-negative probability so that the arithmetic coder can work consistently. In order to satisfy this requirement, the probability distribution must be normalized. We used a python module called "NumpyAc:

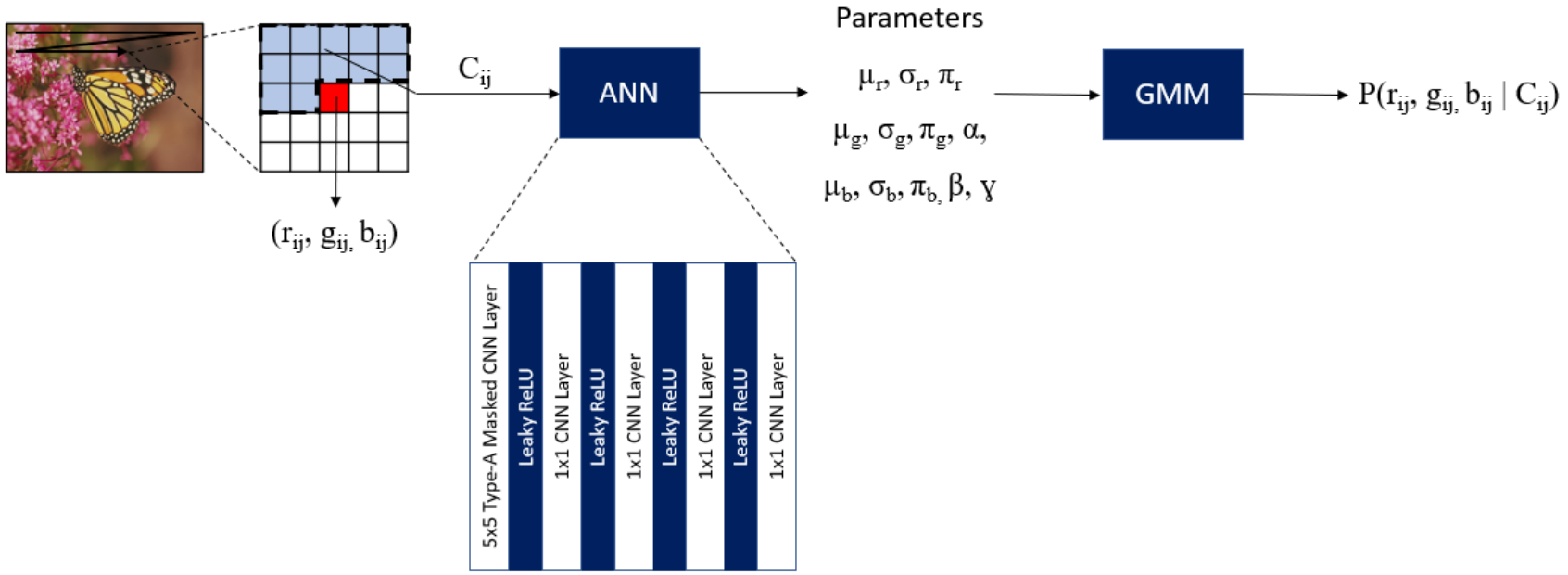

Figure 4.5: The overall scheme for lossless image compression method and the proposed artificial neural network architecture. Neighboring pixels ($C_{i,j}$) of the current pixel are processed with the help of artificial neural network and the parameters which will be used in the Gaussian Mixture model are obtained. Then the pixel probability is calculated by using the Gaussian Mixture Model.

Fast Autoregressive Arithmetic Coding" [36] which is a version of the module "torchac: Fast Arithmetic Coding for PyTorch" [37] provided by Mentzer et al. and allows autoregressive decoding. Our tests showed that this module was faster than the available pure Python modules because of implementing the back-end in C++. It also handles the zero probability problem by proposing an optional parameter for CDF normalization.

4.6.1 Encoding Operation

The real encoding process consists of three steps (see figure 4.6) . In the first step, the region containing the neighboring pixels (i.e., left and above) around the pixel to be encoded is processed with the artificial neural network and the parameters which will be used by the Gaussian Mixture Model are obtained. In the second step, the obtained parameters are given to the Gaussian Mixture Model and a 257-valued CDF matrix of the pixel to be encoded is created. All the operations up to this step are performed on the GPU. In the third step, since the arithmetic encoder operates on the CPU, the obtained CDF matrix and the symbol to be encoded (i.e., the original pixel value) are transferred to the CPU memory. The arithmetic encoder encodes the symbols using the CDF matrices as described in the algorithm 1 and generates the bitstream. All of the described steps are performed separately for all pixels in the image. Since all the pixels of the image are available on the encoder side, these operations can be performed in parallel on the GPU. When encoding with the sequential decoding method, the pixels are encoded in raster scan order. On the other hand, it is necessary to adjust the encoding order of the pixels (i.e., the same as the decoding order) for the parallel decoding method.

4.6.2 Decoding Operation

For decoding operation (see figure 4.7), an image buffer is defined according to the resolution of the image to be decoded. Zero padding is performed for the corner pixels which have not got required neighboring pixels. From the defined image buffer, an image patch is cut so that the pixel to be decoded is in the center. The image patch

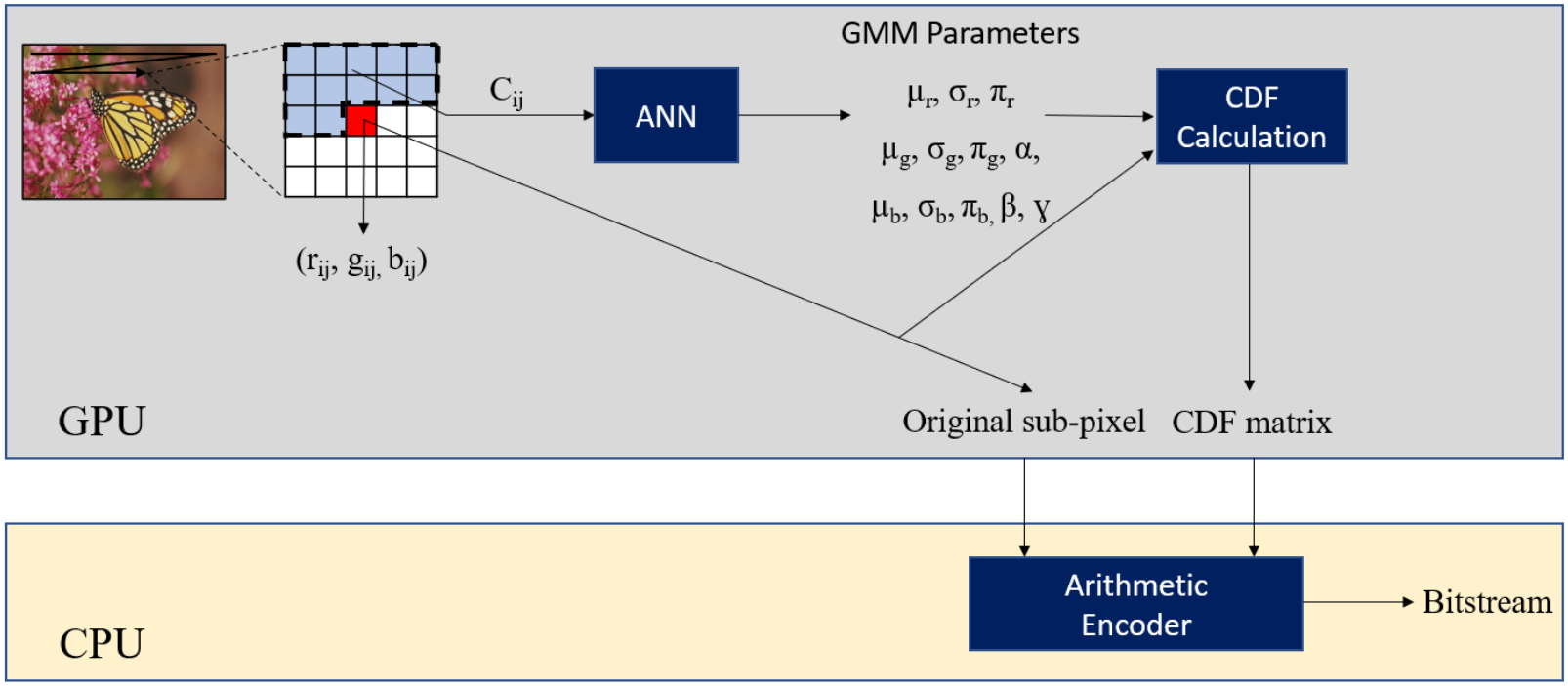

Figure 4.6: Encoding operation with arithmetic encoder

is given to the artificial neural network and the parameters which will be used by the Gaussian Mixture Model are obtained. Since the mean value of the green pixel is updated according to the red pixel, and the mean value of the blue pixel is updated according to the red and green pixels, the CDF matrices of sub-pixels can not be calculated in parallel as in the encoding operation. First, a 257-valued CDF matrix of the red pixel is created. The obtained CDF matrix is transferred to the CPU memory since the arithmetic decoder runs on the CPU. The red pixel value is obtained by the arithmetic decoder using the CDF matrix of the red pixel. The mean values of the Gaussian Mixture Model of the green sub-pixel are updated according to the decoded red pixel value and the CDF matrix is calculated. Then, the CDF matrix is transferred to the CPU memory and the green pixel is obtained by the arithmetic decoder. The operations for the green pixel are also applied for the blue sub-pixel and all sub-pixels are obtained in this way. The corresponding locations in the image buffer are updated according to the obtained sub-pixel values. All operations up to this point continue until all pixels in the image buffer are generated. In the sequential decoding method, all pixels are obtained one by one in raster scan order. In the parallel decoding method, the image patches belonging to the pixels to be decoded in parallel are obtained from the image buffer and are combined in the batch dimension. Then, these patches are given to the artificial neural network, and the parameters of the Gaussian Mixture Model are obtained. Obtaining the parameters of the sub-pixels and computation of the CDF matrices are run in parallel on the GPU. Since the arithmetic decoder works in a sequential manner, the pixels are obtained one by one and the corresponding locations in the image buffer are updated.

4.7 Time Optimization for Decoding Operation

In learning-based autoregressive lossless compression, the encoder can perform parallel processing (i.e. process entire image with masked convolutions based CNN) as all pixels of the image are available. However, the same situation is not possible for the decoder side. Because each pixel is conditioned to the neighboring pixels to its left and above. Therefore, a certain number of pixels above and to the left of each pixel must be obtained by the decoder. This means that the entire image is obtained

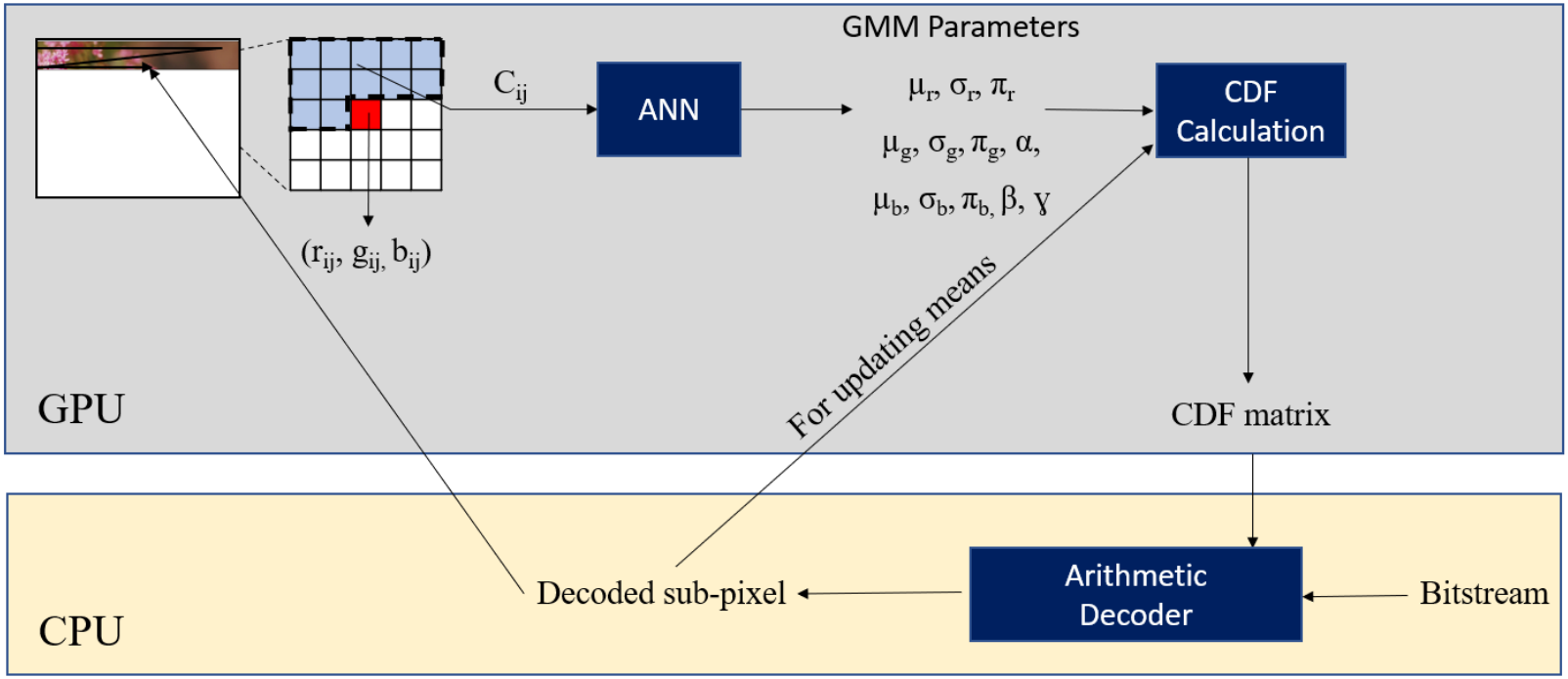

Figure 4.7: Decoding operation with arithmetic decoder

sequentially (see figure 4.8), making autoregressive models impractical in terms of decoding time. In learning-based approaches, making the network architecture less complex will not be enough to improve the decoding time. Therefore, there is a need for approaches and algorithms that can parallelize operations on the decoder side as well as on the encoder side.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Figure 4.8: The sequential decoding order of pixels for an 8x8 block image. The numbers in figure indicate the decoding order of the pixels. All pixels are generated sequentially from left to right and top to bottom. Since there is no parallelization, the processing time will increase according to the resolution of the image to be decoded.

Zhang et al. [5] proposed a parallelization algorithm to speed up the decoding process for autoregressive generative models. According to the algorithm, each pixel for which the necessary neighbor pixels are available can be decoded in parallel. The numbers in figure 4.9 indicate the order in which the pixels are generated by the decoder and the pixels which have the same color can be obtained in parallel. As an example, consider an image with D x D resolution and a local region K x K (region with a dashed line in figure 4.3). For a full autoregressive decoder, each pixel is generated sequentially and it takes D^2 forward passes to obtain the entire image. Parallelization method reduces number of passes from D^2 to [5]:

$$T = D + (D - 1)(h + 1) \tag{4.11}$$

where $K = 2h + 1$ and h changes with the kernel size K.

1	2	3	4	5	6	7	8
4	5	6	7	8	9	10	11
7	8	9	10	11	12	13	14
10	11	12	13	14	15	16	17
13	14	15	16	17	18	19	20
16	17	18	19	20	21	22	23
19	20	21	22	23	24	25	26
22	23	24	25	26	27	28	29

Figure 4.9: The parallel decoding order of pixels for the input filter size 5x5 with parallel decoding method [5]. This method is based on parallel decoding of each pixel whose conditional probability distribution model is independent of each other, according to the filter size used in input layer of the neural network.

The parallel decoding method can be further improved by compromising some neighboring pixels, which will introduce a degradation in compression performance. We proposed a parallel decoding method that allows for more parallel processing. In the previous method, the number of pixels which are decoded in parallel changes according to the range of neighboring pixels used (or kernel size K). In our method, a full diagonal decoding scheme (see figure 4.10) is proposed considering the generation order of autoregressive models (top to bottom and right to left). Thus, each pixel can be decoded in parallel with all the pixels located on its upper and lower diagonal.

This approach compromises compression performance while increasing parallelization, especially for high resolution images. The proposed parallelization method reduces the number of processing independent of kernel size as :

$$T = 2D - 1. \tag{4.12}$$

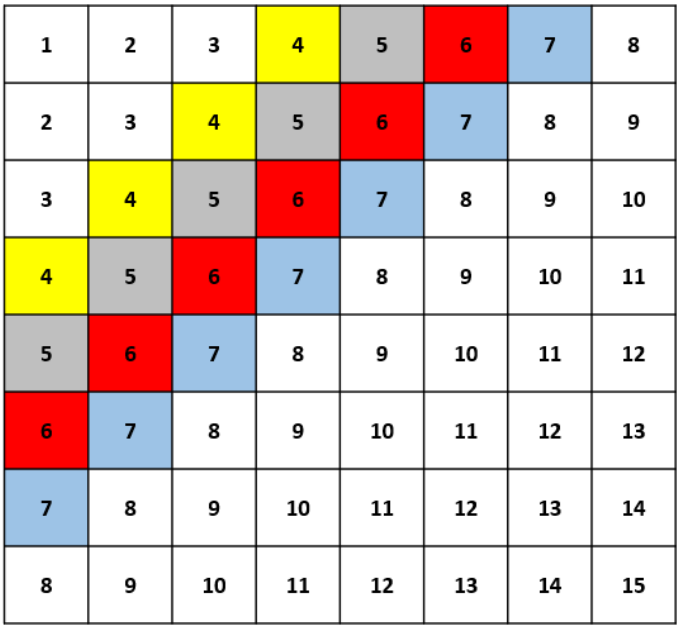

Figure 4.10: The decoding order of pixels for the input filter size 5x5 with proposed diagonal decoding method. All pixels which are located on the diagonal can be obtained parallel.

However, neighboring pixels in the upper diagonal can not be used for this method (see figure 4.11). The unavailable neighboring pixels are replaced by their closest neighbors. For simplicity, the available top pixel (pixel number 4 in the first row) is assigned to all unknown (red) pixels.

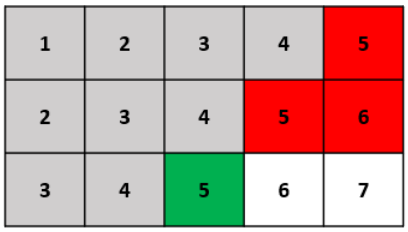

Figure 4.11: Unavailable neighbor pixels (red pixels) for the diagonal parallel decoding algorithm. The green colored pixel represents the current pixel to be generated. Red pixels are assigned the nearest neighbor pixel value.

Dividing the entire image into independent blocks of a certain size is also an approach that will increase parallelism. In this method, since each block is processed independently of the other, zero padding operation must be performed for non-available

neighboring pixels in each block. This will cause a degradation in compression performance, as in the diagonal decoding method. In the previously mentioned parallelization methods, the number of processing depends on the resolution of the image. In the block-based approach, as long as the image fits on the GPU, the number of forward passes depends only on the block size.

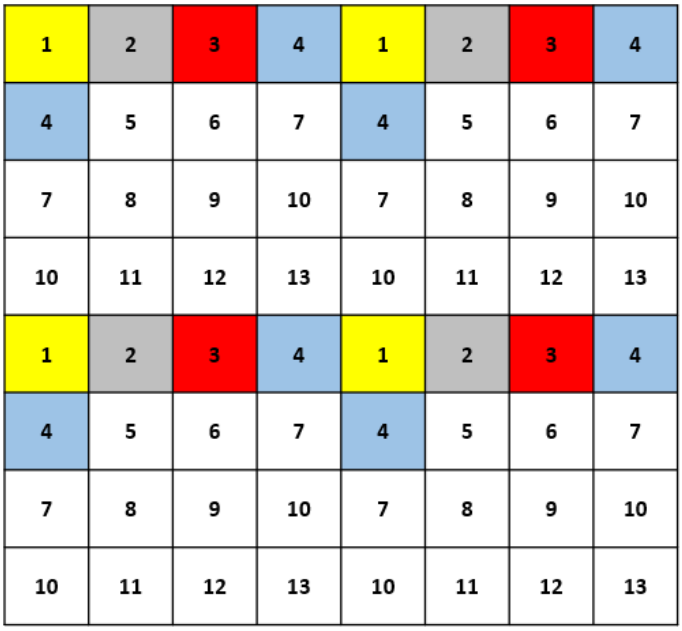

Figure 4.12: Block based decoding scheme (for the algorithm of Zhang et al.[5]). Each block is processed independently of each other.

CHAPTER 5

EXPERIMENTAL RESULTS

5.1 Compression Performance

Our model is evaluated with the test dataset test set [1] which includes 500 images created by the L3C authors. While creating this set, L3C authors randomly selected images from the Open Images dataset[34], and these images are cropped (maximum of 768 pixels in horizontal and vertical resolution) to fit the GPU. Test results are obtained from a system which has NVIDIA GeForce RTX 2080 as GPU and Intel i7-9700 @3.60 GHz as CPU. Our code is available at the link [38].

Table 5.1: Compression Results

	Method	bpsp	Number of Parameters
Traditional Encoders	PNG	4.005	-
	JPEG2000	3.055	-
	WebP	3.047	-
	FLIF	2.867	-
Learning Based Methods	L3C [1]	2.99	$5.01 \cdot 10^6$
	IDF [31]	2.76	$84.33 \cdot 10^6$
	SREC [6]	2.70	$4.20 \cdot 10^6$
	Our Method	2.56	63909

The actual compression performance results of the test dataset are given in Table 5.1. The bpsp (bits per sub-pixel) results (i.e. with arithmetic encoding and decoding) given in the table represent the average number of bits spent per sub-pixel in lossless

compression. Our model outperforms all engineered encoders and learned lossless compression methods.

Generally, processing time and the number of parameters are factors that limit the usability of learning-based systems in real-time applications. The artificial neural network model we presented has significantly fewer parameters compared to other learning-based methods.

5.2 Encoding and Decoding Times

In a learning-based lossless compression system, encoding and decoding times are important depending on the application used. Ideally, these times are expected to be as small as possible. Tables 5.2 and Table 5.3 show the encoder and decoder times for images with and without the parallelization methods. Since all pixels of the image to be encoded are available at the encoder, the cumulative probability distribution function of all pixels can be generated in parallel by a single forward pass of entire image through the artificial neural network. The resulting cumulative distribution function and pixel values are given to the adaptive arithmetic encoder to create a bitstream. The only limit on the encoder side is that the pixel of each channel has a cumulative function of 255 values, and this data cannot fit into the GPU as the resolution gets larger. In this case, instead of calculating all the cumulative distribution functions at once, they are obtained by multiple forward passes. Since all pixels can be compressed in parallel, practical compression times can be achieved with both sequential and parallel decoding methods.

Table 5.2: Encoding Times

Resolution	Encoding Time (s) (With Sequential Decoding Method)	Encoding Time (s) (With Parallelization Method of Zhang et al. [5])	Encoding Time (s) (With Proposed Diagonal Parallelization Method)
64x64	0.08	0.89	0.91
128x128	0.24	1.05	1.04
256x256	0.53	1.51	1.52
512x512	1.64	3.10	3.08

As can be seen in the table 5.2, a time overhead is observed when the image is compressed according to the parallel decoding methods. This is because the encoding order of the pixels changes in the parallel decoding algorithms. In the sequential decoding method, the movement direction of the kernel of the CNN layer and the order of generating the pixels at the decoder are the same (right to left and top to bottom). For this reason, the image can be given directly to the artificial neural network. However, in parallel decoding methods, image patches containing pixels to be encoded and their neighbors are created. In order to perform parallel decoding, the pixels must be obtained in a certain order. Therefore, the generated image patches and the pixels to be encoded must be organized in the same order (decoding order at the decoder) at the encoder side. Additional time losses occur due to these preprocessing steps being applied to correctly decode the image on the decoder side.

During the image decoding process, image patches belonging to each pixel (for the sequential method) or pixel group (for parallel decoding methods) are given to the artificial neural network and the parameters of the GMM model are obtained. Then, using these parameters, the cumulative probability distribution functions of the pixels are calculated. Since the arithmetic encoder runs on the CPU, the calculated cumulative distribution function is moved to the CPU memory. It has been investigated that GPU-CPU transfers are a time-consuming process as the resolution increases, especially in the sequential method. The pixels are decoded using the cumulative distribution functions that are moved to the CPU memory by the arithmetic encoder. Since the arithmetic encoder works in a serial manner, this part works sequentially for each decoding method and is added as a fixed bias to the time for each resolution.

Table 5.3: Decoding Times

	Decoding Time (s)			Number of Forward Passes		
Resolution	With Sequential Decoding Method	With Parallelization Method of Zhang et al. [5]	With Proposed Diagonal Parallelization Method	With Sequential Decoding Method	With Parallelization Method of Zhang et al. [5]	With Proposed Diagonal Parallelization Method
64x64	11.57	1.39	1.16	4096	253	127
128x128	45.94	3.04	2.82	16384	509	255
256x256	181.13	7.93	7.72	65536	1021	511
512x512	711.03	26.26	27.22	262144	2045	1023

The decoding times of block images of different resolutions and the number of forward passes required for decoding operation are given in Table 5.3. As expected, the decoding time increases as the resolution increases in both parallel and sequential de-

coding methods. This is because the number of pixels to be processed increases. In the sequential decoding method, one pixel is produced at a time. Since the GPU can not parallelize operations, this method can not provide efficient results. However, in the parallel decoding method, the pixels to be decoded in parallel in the picture are grouped into a batch and given to the artificial neural network. Therefore, practical times can be obtained compared to the sequential decoding method.

Table 5.4: Time Durations of operations for Decoding Process

Decoding Method	With Sequential Decoding Method				With Parallelization Method of Zhang et al. [5]				With Proposed Diagonal Parallelization Method			
Block Size	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
Obtaining GMM parameters with ANN (s)	1.56	5.94	22.82	90.71	0.28	0.39	0.73	1.57	0.32	0.65	1.45	4.02
CDF Calculation (s)	8.73	34.92	138.41	551.64	0.48	0.91	1.97	5.20	0.25	0.55	1.31	4.05
GPU to CPU transactions (s)	0.97	3.93	15.78	51.43	0.29	0.60	0.97	2.30	0.32	0.45	0.87	1.77
Arithmetic Decoding (s)	0.31	1.15	4.12	17.25	0.34	1.14	4.26	17.19	0.27	1.17	4.09	17.38
Total (s)	11.57	45.94	181.13	711.03	1.39	3.04	7.93	26.26	1.16	2.82	7.72	27.22

When the sequential and parallel methods are compared, it is observed that the proportional difference increases as the resolution increases. As the resolution increases, parallelization methods provide a great improvement since a long time is spent in the operations performed on the GPU in the sequential decoding method (calculating the parameters of the GMM model and calculating the cumulative distribution function of the pixels using the relevant parameters). When the method of Zhang et al. [5] is compared with the diagonal decoding method, it is seen that the diagonal method provides better results at small resolutions. In parallel decoding methods, as the resolution increases, the time spent by the arithmetic encoder becomes more dominant than the time spent on the GPU (see table 5.4), and the diagonal decoding method cannot provide an extra improvement.

The table 5.5 shows the decoding time and compression performance when the image with a resolution of 512x512 is divided into blocks of a certain size. The same evaluation test set was used for compression performance. When the results are examined, it was seen that the block-based approach, especially for the sequential decoding method, improved the decoding time. However, as the block size decreases and the image is divided into more blocks, the compression performance gets worse because there are more unavailable neighboring pixels to be used in the left and upper borders. Splitting into blocks does not provide much improvement for parallel decoding

Table 5.5: Compression Results for Block-based Decoding

Decoding Method	Block size	Encoding time(s)	Decoding time(s)	bpsp
With Sequential Decoding Method	32x32	1.79	21.42	2.615
	64x64	1.82	27.25	2.587
	128x128	1.84	55.47	2.574
	256x256	1.86	161.97	2.568
	Entire image	1.64	711.03	2.563
With Parallelization Method of Zhang et al. [5]	32x32	3.39	21.21	2.615
	64x64	3.41	21.35	2.587
	128x128	3.42	21.76	2.574
	256x256	3.44	23.40	2.568
	Entire image	3.10	26.26	2.563
With Proposed Diagonal Parallelization Method	32x32	3.40	22.47	2.785
	64x64	3.42	22.80	2.761
	128x128	3.43	23.82	2.751
	256x256	3.46	24.60	2.746
	Entire image	3.08	27.22	2.741

methods. Although the diagonal method can do more parallelization on the GPU, it gave a worse result than the method of Zhang et al. [5]. The reason for this can be interpreted as the unstable long times the GPU spends in very large and very small batch sizes. The arithmetic decoder takes about 17 seconds for a 512x512 picture. The remaining time comes from the operations done on the GPU, and more parallel processing on the GPU does not affect the time. Since all pixels are available on the encoder side, the difference in encoding times comes from preprocessing steps.

Table 5.6: Decoding Times L3C and Traditional Encoders (taken from [1])

	Codec	Encoding (s)	Decoding (s)
Traditional Encoders	PNG	0.213	$6.09 \cdot 10^{-5}$
	JPEG200	$1.48 \cdot 10^{-2}$	$2.26 \cdot 10^{-4}$
	WEBP	0.157	$7.12 \cdot 10^{-2}$
	FLIF	1.72	0.133
Learning Based Methods	L3C	0.242	0.374

The encoding and decoding times of non-learning-based compression methods are shorter compared to our method as can be seen in table 5.6. For example, for a 512x512 resolution image, classical methods can encode for 1-200 milliseconds, while FLIF is slower than other classical methods, and this time varies around 1-2 seconds. Decoder times are shorter than compression times. Although L3C is a learning-based system, it can parallelize operations with the method it offers and can perform compression/decoding around 200-400 milliseconds.

5.3 Ablation Study

In this study, the effects of different parameters on the complexity and compression performance were investigated. The purpose of the ablation study is that the artificial neural network used has both relatively low complexity and competitive compression performance.

Table 5.7: Hyperparameters of the proposed model

	Hyperparameter
Number of Mixture	3
Channel size of CNN layers	128
Number of layers	5
Mixture model	Gaussian Mixture Model

The hyperparameters of the proposed network architecture we used for comparison are given in Table 5.7. Experiments were made to change one parameter at a time. The selected probability density function, the number of mixtures used in the mixture model, the number of filters used in the CNN layers, and the effect of the presence of skip connections are shared as below.

The values given in the table 5.8 are obtained by changing the probability distribution function used in the mixture model of the reference network. PixelCNN++ [28] and L3C [1] used the logistic distribution function in their probability models. Probability density function of a logistic distribution is defined as:

$$f(x;\mu,s) = \frac{e^{-(x-\mu)/s}}{s\left(1+e^{-(x-\mu)/s}\right)^2} \tag{5.1}$$

where μ is the location parameter and s is the scale parameter of the logistic distribution. In fact, the Logistic and Gaussian probability distribution functions are also similar in shape, and no major difference is expected in terms of compression results. As a result of the experiment, it was observed that using Gaussian as a probability distribution made a slight improvement in terms of bpsp results.

Table 5.8: The Effect of different probability distribution functions

Model	Gaussian	Logistic
bpsp	2.563	2.607

The number of mixtures used in the Gaussian mixture model is a parameter that will directly affect the compression performance. However, it is seen that increasing the number of mixtures does not significantly affect the compression performance (see Table 5.9).

Table 5.9: The Effect of number of mixtures in GMM

Number of mixtures	K=3	K=5	K=7
bpsp	2.563	2.548	2.545
total number of parameters	63909	67005	70101
trainable number of parameters	58917	62013	65109

From this result, it can be concluded that the probability distribution of the pixels can be modeled with relatively good performance by using fewer number of mixtures. Using a larger number of mixtures can be advantageous if a dataset with a different distribution is used or if more neighboring pixels (more context information) are processed. In the table, the total and trainable parameters in the networks are also given. The total number of parameters refers to the number of parameters in the artificial neural network. However, there are parameters that are not trained due to masked convolution structures and are given as "trainable number of parameters" in the table.

Increasing the number of mixtures or channels also increases the number of parameters in the neural network and adds complexity. In cases where the compression performance does not change significantly, the architecture with less complexity can be preferred.

In order to understand the effect of neural network capacity on compression performance, the number of channels used in CNN layers has been changed. According to the results obtained (see Table 5.10), it is observed that the effect of channel number change on the number of parameters and compression performance is significant.

Table 5.10: The Effect of channel size in CNN layers

Number of channels	C=64	C=128	C=192
bpsp	2.619	2.563	2.529
total number of parameters	19685	63909	132709
trainable number of parameters	17189	58917	125221

In general, skip connections, also known as shortcut connections, are preferred in deep neural network architectures. They are also used in learning-based image compression methods. For this reason, the effect of skip connections has been investigated. The skip connection structure used in the neural network is given in figure 5.1. In the test phase, skip connections were used instead of the layers between the input and output layers of the reference model. According to the results obtained, the added skip connections improved the compression performance.

Table 5.11: The Effect of Skip Connections

	Reference Model	Reference Model with Skip Connections
bpsp	2.563	2.549
total number of parameters	63909	113445
trainable number of parameters	58917	108453

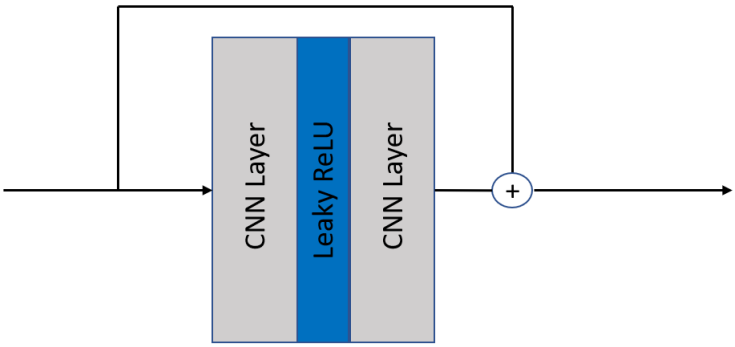

Figure 5.1: The structure of skip connection

5.4 Visual Results

The compression results for a few sample images from the test set used are shared below.

4.63 bpsp/ 5.02 bpsp/ 5.83 bpsp/ 4.91 bpsp/ 4.61 bpsp

Figure 5.2: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

1.48 bpsp/ 1.55 bpsp/ 2.16 bpsp/ 1.33 bpsp/ 1.02 bpsp

Figure 5.3: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

4.08 bpsp/ 4.41 bpsp/ 5.92 bpsp/ 4.61 bpsp/ 3.97 bpsp

Figure 5.4: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

1.56 bpsp/ 1.55 bpsp/ 2.25 bpsp/ 1.54 bpsp/ 1.17 bpsp

Figure 5.5: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

4.07 bpsp/ 4.38 bpsp / 5.18 bpsp/ 4.25 bpsp/ 3.94 bpsp

Figure 5.6: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

4.16 bpsp/ 4.53 bpsp / 5.38 bpsp/ 4.43 bpsp/ 4.11 bpsp

Figure 5.7: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

4.41 bpsp/ 4.80 bpsp/ 6.01 bpsp/ 4.85 bpsp/ 4.30 bpsp

Figure 5.8: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

5.46 bpsp/ 5.98 bpsp/ 6.60 bpsp/ 5.75 bpsp/ 5.41 bpsp

Figure 5.9: Bpsp results of SREC[6] /L3C [1]/ PNG/ WEBP/ Ours respectively

CHAPTER 6

CONCLUSIONS

In this thesis, a deep learning based lossless image compression method with pixel-by-pixel processing is presented. The probability distribution of each pixel is modeled by conditioning the values of the pixels to its left and above, and this model is learned with an artificial neural network. This model prevents parallel decoding/processing of neighboring pixels in the decoder, leading to impractical decoding times and causing the entire image to be generated sequentially from left to right and top to bottom. In order to reduce the decoding time, the special acceleration algorithm [5] presented in the literature has been implemented. In addition, different acceleration approaches are proposed and the results are compared. It is concluded that parallelization algorithms significantly improve decoder time and improve autoregressive generative models in terms of practicality. Compression performance was compared with other methods in the literature. The compression performance of our proposed model is better than both traditional encoders and many learning-based coders. However, the encoding and especially decoding times are significantly higher and, need improvement for practical use.

6.1 Future Work

Autoregressive generative models can learn a tractable and explicit probability distribution of the training data, and they provide better log likelihood scores than latent variable models such as Variational Autoencoder (VAE) [**?**][39]. This makes them competitive in terms of compression performance. It is possible to improve artificial neural network based lossless compression methods both in terms of compression

performance and processing time. New methods and algorithms are needed to reduce the impractical decoding times of pixel-by-pixel learned compression methods to compete with traditional encoders. In future studies, it is useful to develop different parallelization methods that can improve this situation.

REFERENCES

[1] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. V. Gool, "Practical full resolution learned lossless image compression," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 10629–10638, 2019.

[2] H. Malvar and G. Sullivan, "Ycocg-r: A color space with rgb reversibility and low dynamic range," *ISO/IEC JTC1/SC29/WG11 and ITU-T SG16 Q*, vol. 6, 2003.

[3] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu, "Conditional image generation with pixelcnn decoders," *arXiv preprint arXiv:1606.05328*, 2016.

[4] S. Reed, A. Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. Freitas, "Parallel multiscale autoregressive density estimation," in *International Conference on Machine Learning*, pp. 2912–2921, PMLR, 2017.

[5] M. Zhang, J. Townsend, N. Kang, and D. Barber, "Parallel neural local lossless compression," *arXiv preprint arXiv:2201.05213*, 2022.

[6] S. Cao, C.-Y. Wu, and P. Krähenbühl, "Lossless image compression through super-resolution," *arXiv preprint arXiv:2004.02872*, 2020.

[7] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and computational harmonic analysis*, vol. 5, no. 3, pp. 332–369, 1998.

[8] A. R. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Lossless image compression using integer to integer wavelet transforms," in *Proceedings of International Conference on Image Processing*, vol. 1, pp. 596–599, IEEE, 1997.

[9] J. Reichel, G. Menegaz, M. J. Nadenau, and M. Kunt, "Integer wavelet transform for embedded lossy to lossless image compression," *IEEE Transactions on Image Processing*, vol. 10, no. 3, pp. 383–392, 2001.

[10] S. Dewitte and J. Cornelis, "Lossless integer wavelet transform," *IEEE signal processing letters*, vol. 4, no. 6, pp. 158–160, 1997.

[11] W. Sweldens, "The lifting scheme: A custom-design construction of biorthogonal wavelets," *Applied and computational harmonic analysis*, vol. 3, no. 2, pp. 186–200, 1996.

[12] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010.

[13] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[14] P. Chrabaszcz, I. Loshchilov, and F. Hutter, "A downsampled variant of imagenet as an alternative to the cifar datasets," *arXiv preprint arXiv:1707.08819*, 2017.

[15] D. Freedman and P. Diaconis, "On the histogram as a density estimator: L 2 theory," *Zeitschrift für Wahrscheinlichkeitstheorie und verwandte Gebiete*, vol. 57, no. 4, pp. 453–476, 1981.

[16] S. Węglarczyk, "Kernel density estimation and its application," in *ITM Web of Conferences*, vol. 23, p. 00037, EDP Sciences, 2018.

[17] C. Shannon and W. Weaver, "A mathematical theory of communication (paperback ed.; urbana, il," 1962.

[18] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Communications of the ACM*, vol. 30, no. 6, pp. 520–540, 1987.

[19] S. Golomb, "Run-length encodings (corresp.)," *IEEE transactions on information theory*, vol. 12, no. 3, pp. 399–401, 1966.

[20] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[21] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 06, pp. 8–19, 1984.

[22] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still image data compression standard*. Springer Science & Business Media, 1992.

[23] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The loco-i lossless image compression algorithm: Principles and standardization into jpeg-ls," *IEEE Transactions on Image processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[24] T. Boutell and T. Lane, "Png (portable network graphics) specification version 1.0," *Network Working Group*, pp. 1–102, 1997.

[25] W. B. P. Version, "1.0 hardware reference guide, xp-002202892, network engines," *Inc., Jun*, vol. 1, p. 92, 2000.

[26] J. Sneyers and P. Wuille, "Flif: Free lossless image format based on maniac compression," in *2016 IEEE international conference on image processing (ICIP)*, pp. 66–70, IEEE, 2016.

[27] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *International Conference on Machine Learning*, pp. 1747–1756, PMLR, 2016.

[28] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[29] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[30] J. Ho, E. Lohn, and P. Abbeel, "Compression with flows via local bits-back coding," *arXiv preprint arXiv:1905.08500*, 2019.

[31] E. Hoogeboom, J. W. Peters, R. v. d. Berg, and M. Welling, "Integer discrete flows and lossless compression," *arXiv preprint arXiv:1905.07376*, 2019.

[32] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned image compression with discretized gaussian mixture likelihoods and attention modules," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7939–7948, 2020.

[33] Z. Cheng, H. Sun, M. Takeuchi, and J. Katto, "Learned lossless image compression with a hyperprior and discretized gaussian mixture likelihoods," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2158–2162, IEEE, 2020.

[34] I. Krasin, T. Duerig, N. Alldrin, V. Ferrari, S. Abu-El-Haija, A. Kuznetsova, H. Rom, J. Uijlings, S. Popov, A. Veit, *et al.*, "Openimages: A public dataset for large-scale multi-label and multi-class image classification," *Dataset available from https://github. com/openimages*, vol. 2, no. 3, p. 18, 2017.

[35] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[36] C. Fu, "Numpyac: Fast autoregressive arithmetic coding." `https://github.com/zb12138/NumpyAc`, Feb. 2022.

[37] F. Mentzer, "torchac: Fast arithmetic coding for pytorch." `https://github.com/fab-jul/torchac`, Feb. 2022.

[38] F. Kamisli, "a-pixel-by-pixel-learned-lossless-image-compression-method-with-parallel-." `https://github.com/metu-kamisli/a-pixel-by-pixel-learned-lossless-image-compression-method-with-parallel-decoding`, July 2022.

[39] D. J. Rezende, S. Mohamed, and D. Wierstra, "Stochastic backpropagation and approximate inference in deep generative models," in *International conference on machine learning*, pp. 1278–1286, PMLR, 2014.

[40] D. Liu, Y. Li, J. Lin, H. Li, and F. Wu, "Deep learning-based video coding: A review and a case study," *ACM Computing Surveys (CSUR)*, vol. 53, no. 1, pp. 1–35, 2020.

[41] H. Zhang, F. Cricri, H. R. Tavakoli, N. Zou, E. Aksu, and M. M. Hannuksela, "Lossless image compression using a multi-scale progressive statistical model," in *Proceedings of the Asian Conference on Computer Vision*, 2020.

[42] N. Zou, H. Zhang, F. Cricri, H. R. Tavakoli, J. Lainema, M. Hannuksela, E. Aksu, and E. Rahtu, "L 2 c–learning to learn to compress," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2020.

[43] A. Liu, S. Mandt, and G. V. d. Broeck, "Lossless compression with probabilistic circuits," *arXiv preprint arXiv:2111.11632*, 2021.

[44] K. Unno, Y. Kameda, I. Matsuda, S. Itoh, and S. Naito, "Lossless color image coding based on probability model optimization utilizing example search and adaptive prediction," *ITE Transactions on Media Technology and Applications*, vol. 8, no. 3, pp. 132–139, 2020.

[45] I. Schiopu and A. Munteanu, "Deep-learning-based lossless image coding," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 7, pp. 1829–1842, 2019.

[46] I. Schiopu and A. Munteanu, "Residual-error prediction based on deep learning for lossless image compression," *Electronics Letters*, vol. 54, no. 17, pp. 1032–1034, 2018.

[47] F. Mentzer, E. Agustsson, M. Tschannen, R. Timofte, and L. Van Gool, "Conditional probability models for deep image compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402, 2018.

[48] Y.-H. Ho, C.-C. Chan, W.-H. Peng, H.-M. Hang, and M. Domański, "Anfic: Image compression using augmented normalizing flows," *IEEE Open Journal of Circuits and Systems*, vol. 2, pp. 613–626, 2021.

[49] H. Rhee, Y. I. Jang, S. Kim, and N. I. Cho, "Lossless image compression by joint prediction of pixel and context using duplex neural networks," *IEEE Access*, vol. 9, pp. 86632–86645, 2021.

[50] H. Rhee, Y. I. Jang, S. Kim, and N. I. Cho, "Channel-wise progressive learning for lossless image compression," in *2020 IEEE International Conference on Image Processing (ICIP)*, pp. 1113–1117, IEEE, 2020.

[51] M. Zhang, A. Zhang, and S. McDonagh, "On the out-of-distribution generalization of probabilistic image modelling," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[52] H. Ma, D. Liu, N. Yan, H. Li, and F. Wu, "End-to-end optimized versatile image compression with wavelet-like transform," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[53] Y. Ruan, K. Ullrich, D. S. Severo, J. Townsend, A. Khisti, A. Doucet, A. Makhzani, and C. Maddison, "Improving lossless compression rates via monte carlo bits-back coding," in *International Conference on Machine Learning*, pp. 9136–9147, PMLR, 2021.

[54] T. Strutz, "Improved probability modelling for exception handling in lossless screen content coding," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2173–2177, IEEE, 2020.

[55] G. Xin and P. Fan, "Soft compression for lossless image coding," *arXiv preprint arXiv:2012.06240*, 2020.

[56] F. Mentzer, L. V. Gool, and M. Tschannen, "Learning better lossless compression using lossy compression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6638–6647, 2020.

[57] S. Reed, Y. Chen, T. Paine, A. v. d. Oord, S. Eslami, D. Rezende, O. Vinyals, and N. de Freitas, "Few-shot autoregressive density estimation: Towards learning to learn distributions," *arXiv preprint arXiv:1710.10304*, 2017.

[58] M. Rahman, M. Hamada, *et al.*, "Lossless image compression techniques: A state-of-the-art survey," *Symmetry*, vol. 11, no. 10, p. 1274, 2019.

[59] M. Rahman, M. Hamada, J. Shin, *et al.*, "The impact of state-of-the-art techniques for lossless still image compression," *Electronics*, vol. 10, no. 3, p. 360, 2021.

[60] S. Elakkiya and K. Thivya, "Comprehensive review on lossy and lossless compression techniques," *Journal of The Institution of Engineers (India): Series B*, pp. 1–10, 2021.

[61] J. Townsend and I. Murray, "Lossless compression with state space models using bits back coding," *arXiv preprint arXiv:2103.10150*, 2021.

[62] D. T. Nguyen, M. Quach, G. Valenzise, and P. Duhamel, "Multiscale deep context modeling for lossless point cloud geometry compression," in *2021 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)*, pp. 1–6, IEEE, 2021.

[63] H. Huang, I. Schiopu, and A. Munteanu, "Low-complexity angular intra-prediction convolutional neural network for lossless hevc," in *2020 IEEE 22nd International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1–6, IEEE, 2020.

[64] D. Mishra, S. K. Singh, and R. K. Singh, "Deep architectures for image compression: A critical review," *Signal Processing*, vol. 191, p. 108346, 2022.

[65] M. Yang and N. Bourbakis, "An overview of lossless digital image compression techniques," in *48th Midwest Symposium on Circuits and Systems, 2005.*, pp. 1099–1102, IEEE, 2005.

[66] J. Shukla, M. Alwani, and A. K. Tiwari, "A survey on lossless image compression methods," in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 6, pp. V6–136, IEEE, 2010.

[67] G. Xin and P. Fan, "Soft compression for lossless image coding based on shape recognition," *Entropy*, vol. 23, no. 12, p. 1680, 2021.