# LOSSLESS IMAGE COMPRESSION USING REVERSIBLE INTEGER WAVELET TRANSFORMS AND CONVOLUTIONAL NEURAL NETWORKS

by

Eze Ahanonu



A Thesis Submitted to the Faculty of the

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

In Partial Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

In the Graduate College

THE UNIVERSITY OF ARIZONA

2018

## STATEMENT BY AUTHOR

This thesis has been submitted in partial fulfillment of the requirements for an advanced degree at the University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the Library.

Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: Eze Ahanonu

## APPROVAL BY THESIS DIRECTOR

This thesis has been approved on the date shown below:

Ali Bilgin, PhD

10/1/2018

Date

# ACKNOWLEDGEMENTS

First, I would like to thank Dr. Ali Bilgin who inspired me to pursue a graduate education, as well as provided me with outstanding feedback and support throughout my masters work.

I am also very grateful for the support of Dr. Michael Marcellin, who played a crucial role in my graduate success. Additionally, I would like to thank Dr. Amit Ashok for agreesing to be a member of my thesis committee.

I highly value my various discussions with Dr. Feng Liu, Dr. Miguel Herndez and Yuzhang Lin, which have both expanded and refined my understanding of several key concepts.

I would like to thank Maria Teresa Velez, Shetara OliwoOlabode and Jim Field for the unconditional funding and professional development resources they have provided during my masters.

I would like to thank Tami Whelan and Diana Wilson for their guidance and patience while ensuring I satisfied the various requirements to complete my degrees.

I would like to thank my friends for keeping me company over the years, in particular Corey Zammit, Matt Konen, Robert Blair, Otis Blank, and Shayan Milani. Additionally, I greatly appreciate the support of Savanna Weninger, who kept me motivated while completing my thesis.

Finally, I would like to thank my family. Throughout the years they have given me endless support, and have always believed in me more than I believed in myself.

# TABLE OF CONTENTS

TABLE OF CONTENTS – *Continued*

# LIST OF FIGURES

LIST OF FIGURES – *Continued*

LIST OF FIGURES – *Continued*

# LIST OF TABLES

# ABSTRACT

Image compression is an area of data compression which looks to exploit various redundancies that exist within images to reduce storage and transmission requirements. In information critical applications such as professional photography, medical diagnostics, and remote sensing, *lossless* image compression may be used to ensure the original data can be restored at a later time.

In this work, a lossless compression framework is proposed which incorporates Convolutional Neural Networks (*CNN*s) to predict wavelet detail coefficients from coefficients within neighboring subbands. The main premise of the proposed framework is that information which can be recovered at the decoder via $CNN$ prediction can be excluded from the compressed codestream, resulting in reduced file sizes.

An end-to-end encoder and decoder is implemented to test the validity of the proposed, model and compression performance is compared with current state of the art methods.

# CHAPTER 1

# INTRODUCTION

Image compression, an area of data compression, looks to reduce the data requirements associated with transferring and storing digital images; it may be divided into two general categories known as *lossless* and *lossy*. In lossless compression, we require that the original image data be recoverable without error during decompression; whereas in lossy compression, we introduce a controlled level of error into the data in order to achieve greater compression performance. Applications in which communication bandwidth or storage capacity take precedence over preservation of image contents will generally use lossy compression. Lossless compression is seen in applications where any loss of information within an image is not tolerable. These include some areas of medical imaging, remote sensing, and image archiving. Over the last two decades, researchers in both industry and academia have proposed numerous lossless compression frameworks to meet this demand, some of which include: CALIC (1996) [1], JPEG-LS (1999) [2], Lossless JPEG2000 (2000) [3], GLICBAWLS (2001) [4], and FLIF (2015) [5].

In recent years, machine learning has brought about many breakthroughs in different areas of science. In particular, deep learning fueled by significant growth of computational power, and availability of large datasets, has improved the state-of-the-art in many speech and image processing applications as well as other areas of science such as genomics. While these breakthroughs have occurred in many areas, the impact of deep learning techniques in image and video compression has so far been relatively modest. In large, work in this area has focused on developing generative image models, which can be used to predict small pixel patches [6] [7]. Compression is then achieved by lossless predictive coding, in which smaller magnitude residual values are coded in place of original pixel intensities. These models show an impressive ability to understand past and predict future structure

within an image. Unfortunately, there is a lack of end-to-end implementation of these frameworks to compare with existing standards.

In this work, we propose a lossless compression framework which uses *CNNs* to make predictions of wavelet coefficients from coefficients within neighboring subbands. Information which can be recovered later at the decoder using $CNN$ prediction can be excluded from the codestream resulting in reduced file sizes. We show that a $CNN$ may be developed which, on average, produces predictions whose corresponding prediction errors are easier to compress than original coefficients. An end-to-end encoder and decoder are implemented to validate the model and compression performance is compared with current state of the art methods.

## 1.1 Chapter Overview

This paper is organized as follows: Chapter 2 will introduce necessary concepts which will be discussed throughout this thesis. Chapter 3 gives an overview of the proposed compression framework, along with pseudo-code which can be used for implementation. In Chapter 4 we explore several *CNN* prediction models, outline a baseline prediction framework, and implement an end-to-end encoder and decoder. Concluding remarks and future work are discussed in Chapter 5.

# CHAPTER 2

# BACKGROUND

## 2.1 Lossless Image Compression

*Lossless* image compression is an area of data compression which looks to minimize communication and storage requirements of digital images without loss of information. In order to accomplish this task, lossless compression algorithms exploit various redundancies which exist in image data.

*Spatial redundancy* regards the strong correlations which exist among neighboring pixel values in natural images. After decorrelation, many data samples will be small in magnitude requiring fewer bits on average to code than original pixel values. Common decorrelation procedures include predictive coding (e.g. Differential Pulse Code Modulation [8]) and transform coding (e.g. Discrete Cosine Transform [9] and Discrete Wavelet Transform [10]).

*Coding redundancy* results from non-uniform representation of sample values within image data. Particularly after decorrelation, data samples will take values that are small in magnitude with far greater frequency than larger magnitude values. It becomes inefficient to use the same number of bits to represent each value, instead one should code frequent values with fewer bits than less common ones. The process of removing these redundancies is referred to as *entropy coding*, and is typically one of the last steps in a compression algorithm. Examples of entropy coding techniques include Huffman Coding [11], Run-Length Coding [12], and Arithmetic Coding [13].

## 2.2 Compression Performance

The performance of a lossless image compression algorithm is determined by its ability to reduce the number of bits required to uniquely represent an image. A

natural way to express this is by *compression ratio* (Equation 2.1), which is a ratio of bit-stream lengths before and after compression.

$$compression\ ratio\ = \frac{D_r D_c B}{L_C} \tag{2.1}$$

The numerator in Equation 2.1 gives the uncompressed bit-stream length, where $D_r$ and $D_c$ respectively represent the number of $B$ bit pixels in each image row and column. The denominator, $L_C$, represents the length of the compressed bit-stream in bits after coding by a given compression algorithm.

While compression ratio is easy to understand, it is a unit-less relative measure that does not indicate actual storage requirements of a compressed bit-stream. *Bit-rate (R)* is an absolute measure indicating on average the number of bits used to code individual data samples. For images, pixels are considered individual samples and bit-rate is described in *bits-per-pixel (bpp)* (Equation 2.2).

$$R = \frac{L_C}{D_r D_c} \quad bits/pixel \tag{2.2}$$

Throughout this thesis, bit-rate will be used to measure compression performance.

## 2.3 Compression and Entropy

For a discrete signal, $X$, which takes values in the set $\mathcal{A}_X$; a fixed length binary code which uniquely represents each value in $\mathcal{A}_X$ will have a minimum codeword length of $L = log_2\left[|\mathcal{A}_X|\right]$; where $|\mathcal{A}_X|$ denotes the cardinality of the set $\mathcal{A}_X$. However if the values in $\mathcal{A}_X$ follow a non-uniform distribution $f_X(x)$, so that certain values occur more frequently than others, a variable length prefix code may be constructed which on average requires fewer than $L$ bits to uniquely represent each value. This average codeword length is lower-bounded by a value known as *entropy* [14], which is defined as:

$$H(X) \triangleq -\sum_{x \in \mathcal{A}_X} f_X(x) log_2[f_X(x)] \tag{2.3}$$

In general, signals which take a small number of values with high probability will have lower entropy and can be represented with fewer bits on average than signals who take a large number of values with low probability. For the extreme case where values are uniformly distributed with $f_X(x) = \frac{1}{|\mathcal{A}_X|}$, entropy will reach its maximum of $log_2[|\mathcal{A}_X|]$ and no benefit is seen when using a variable length code.

Pixel intensities in typical grayscale image are assumed to follow an approximate uniform distribution, making them difficult to code at bit-rates significantly lower than $L$. However as mentioned in Section 2.1, pixels contain a large amount of spatial redundancy that can be removed. The resulting equivalent signal will contain primarily small magnitude samples (lower entropy) making it easier to compress. In the next section, we introduce the *Discrete Wavelet Transform* which has been used as a tool to remove redundancies in several high performance image and video codecs such as *JPEG2000* [3] and *CineForm* [15], and is also incorporated in the proposed framework.

## 2.4 Discrete Wavelet Transform

The Discrete Wavelet Transform ($DWT$) [10] [16] is a joint space-frequency transform which decomposes a discrete-time signal into frequency and spatially localized subband components. The $DWT$ is used in image compression to exploit spatial correlations among signal samples at various scales; with small scale (high frequency) components capturing short term correlations among neighboring samples and large scale (low frequency) components describing long term correlations which span over several signal samples.

The $DWT$ of a discrete-time signal $x$ may be computed by passing it through a perfectly reconstructing FIR filter bank. This involves convolution of $x$ with a series of low and high pass *analysis* filters ($\tilde{H}_0$ and $\tilde{H}_1$), followed by down-sampling to obtain critically sampled subbands ($y_L$ and $y_H$). Signal reconstruction is achieved by up-sampling each subband, applying a series of *synthesis* filters ($H_0$ and $H_1$), and summing the results. The high computational costs of convolution can make

this approach impractical for large signals. An alternative method is proposed in [17], which simplifies the *DWT* into a set of *lifting steps*. In this section, we derive the lifting steps for the Cohen-Daubechies-Feauveau (CDF) 5/3 wavelet transform [18] used in the lossless pipeline of *JPEG2000*. As in [17], we develop this transform through analysis in the spatial domain, which may provide a more intuitive understanding of the *DWT* than a Fourier analysis approach.

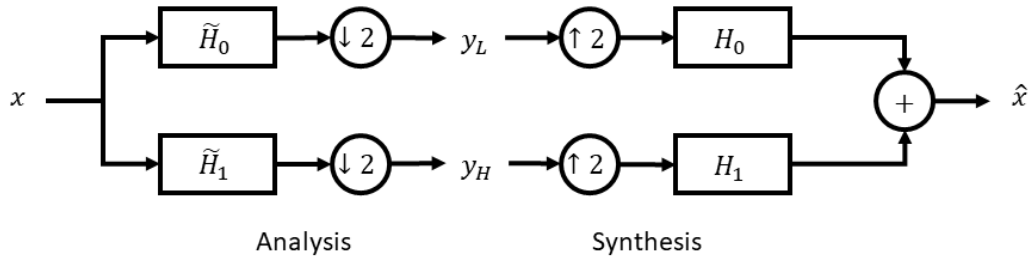


Figure 2.1: Implementation of the *1-D DWT* using a 2-channel perfectly reconstructing *FIR* filterbank.

The first step is to split $x$ into the even and odd signals $x_e[n] = x[2n]$ and $x_o[n] = x[2n+1]$, respectively. It is apparent that these signals are related and will show some correlations. Thus, a predictor function $P$ may be designed to make predictions of $x_o$ using $x_e$. If $P$ is well designed, then an error signal $y_d = x_o - P(x_e)$ may be computed which is sparse, only taking large values in regions where $x_e$ and $x_o$ become uncorrelated, resulting in bad predictions. These uncorrelated events will occur in regions where $x$ contains transient events, in which rapid variation occurs among neighboring sample values. If $P$ is chosen to make its prediction by taking the average of the neighboring values in $x_e$ then the prediction error signal is computed as:

$$y_d[n] = x_o[n] - \frac{x_e[n] + x_e[n+1]}{2} \tag{2.4}$$

$x$ may now be represented in terms of $x_o$ and $y_d$, where $y_d$ is a decorrelated high-pass subband of $x$. $y_d$ will be sparse when $x$ contains a small number of transient events, indicating that fewer bits on average may be required to store $x$ in this new

representation. As mentioned in the introduction of this section, the $DWT$ looks to decompose the input signal into frequency localized subbands. Currently, $x_e$ has frequency contents that overlap with $y_d$ and contain a large amount of aliasing due to subsampling. Daubechies and Sweldens [17] proposed a second lifting step to reduce this aliasing. A smoothing operator $U$ is introduced which is used to calculate a smoothed signal $y_a = x_e + U(y_d)$ given by:

$$y_a[n] = x_e[n] + \frac{y_d[n-1] + y_d[n]}{4} \tag{2.5}$$

$y_a$ is a low-pass subband, containing a low resolution approximation of $x$ for which smoothing has been applied to reduce aliasing. $x$ may now be represented in terms of $y_a$ and $y_d$; $y_a$ is called the *approximation subband*, and contains approximation coefficients, while $y_d$ is called the *detail subband* and contains detail coefficients. Equations 2.4 and 2.5 represent what are called *predict* and *update* lifting steps and together form the lifting implementation of the CDF 5/3 $DWT$ (omitting any normalizations). We may modify our notation so that the predict and updates steps are equivalently given by:

$$Predict: \quad y_H[n] = x[2n+1] - \frac{x[2n] + x[2n+2]}{2} \tag{2.6}$$

$$Update: \quad y_L[n] = x[2n] + \frac{y_H[n-1] + y_H[n]}{4} \tag{2.7}$$

where we have now chosen $y_L$ and $y_H$ to represent the approximation and detail subbands, respectively, to emphasize their respective low and high-pass nature. In theory, this transform may then be inverted by reversing the predict and update steps as follows:

$$Update: \quad x[2n] = y_L[n] - \frac{y_H[n-1] + y_H[n]}{4} \tag{2.8}$$

$$Predict: \quad x[2n+1] = y_H[n] + \frac{x[2n] + x[2n+2]}{2} \tag{2.9}$$

In a practical implementation, the potential introduction of floating-point values results in its reversibility being limited by the precision used to perform the computations. For image compression, requiring floating point operations would not allow for lossless compression to be achieved. It is then desirable to obtain a transform that maps integers to integers so that no issues related to floating-point values arise. This is easily achieved by introducing rounding into the lifting steps above. With this modification we obtain the *Reversible CDF 5/3 DWT*:

$$Prediction: \quad y_H[n] = x[2n+1] - \left\lfloor \frac{x[2n] + x[2n+2]}{2} + \frac{1}{2} \right\rfloor \tag{2.10}$$

$$Update: \quad y_L[n] = x[2n] + \left\lfloor \frac{y_H[n-1] + y_H[n]}{4} + \frac{1}{2} \right\rfloor \tag{2.11}$$

Again by inverting the lifting steps the original signal values may be recovered:

$$Update: \quad x[2n] = y_L[n] - \left\lfloor \frac{y_H[n-1] + y_H[n]}{4} + \frac{1}{2} \right\rfloor \tag{2.12}$$

$$Predict: \quad x[2n+1] = y_H[n] + \left\lfloor \frac{x[2n] + x[2n+2]}{2} + \frac{1}{2} \right\rfloor \tag{2.13}$$

This is the wavelet transform which will form the basis of the proposed compression framework.

### 2.4.1 2-D *DWT*

For applications in digital imaging, the $DWT$ may be separably extended to two dimensions by first applying a 1-D $DWT$ to each image row, followed by a 1-D $DWT$ of each resulting column. This procedure is demonstrated in Figure 2.2 for a single component image, resulting in one approximation ($LL$) and three detail ($HL, LH$ and $HH$) subbands. The $LL$ subband represents a low resolution approximation of the original image, while the detail subbands individually give edge information in the horizontal ($HL$), vertical ($LH$), and diagonal ($HH$) directions. The detail subbands will be sparse and biased towards zero, only taking on large values in

areas corresponding to rapid change in pixel intensity. This results in the wavelet representation of an image to be more compressible than the original pixel values.

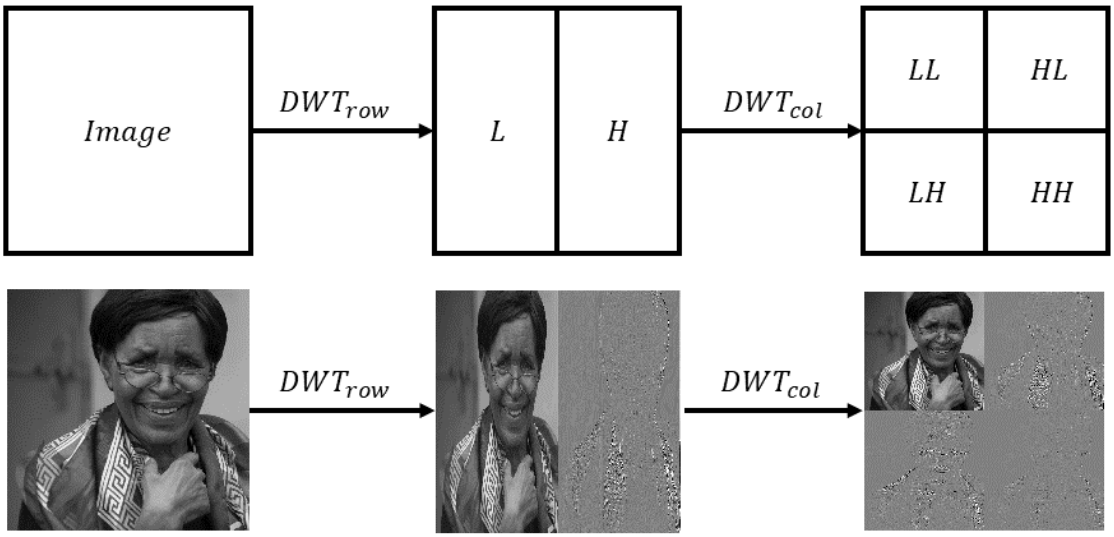


Figure 2.2: Implementation of the 2-D $DWT$ on a single component image.

To achieve greater sparsity, a multi-level 2-D $DWT$ may be implemented by recursively decomposing the approximation subband (Figure 2.3). These subbands may be organized into *decomposision levels*, denoted by $\mathcal{D}_n$, where each decomposition level contains the subbands which are produced during a given decomposition step (e.g. $LL_n, HL_n, LH_n, HH_n$). We note that even though the approximation subband ($LL_n$) in a given decomposition level ($\mathcal{D}_n$) may be decomposed after additional decomposition steps, we still consider it part of the repsective decomposition level after the full multi-level transform has been applied. This understanding will be convineient in later sections when describing the *CNN* prediction procedure. It should be understood that approximation subbands which are decomposed during the multi-level transform will not be included in the codestream. Throughout this thesis, subbands which are contained within the same decomposition level will be referred to as *neighboring subbands* (e.g. $LL_1$,$HL_1$,$LH_1$,$HH_1$).

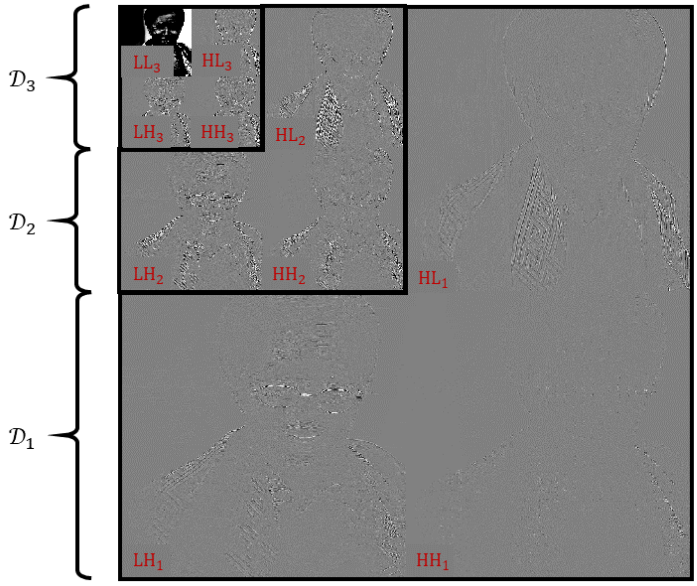


Figure 2.3: $N = 3$ level $DWT$ on an image resulting in $3 \cdot 5 + 1 = 16$ subbands.

## 2.5 Predictive Coding

Predictive coding is a technique used extensively in image compression to remove spatial redundancies which exist among neighboring pixel values. In a more general case, we consider a sequence of arbitrary sample values $x$, a predictor function $F$ and a set of causal information $\mathcal{I}_x$ associated with $x$; a residual signal may then be computed by $r = x - F(\mathcal{I}_x)$. If $F$ is a good predictor, the values in $r$ will be tightly distributed around zero, leading to higher compressibility.

While the $DWT$ is able to effectively decorrelate neighboring image pixels, the resulting wavelet coefficients are not independent and may have some remaining interdependencies which can be exploited to achieve greater compression performance. To achieve this, we wish to design a new predictor that can be used to make predictions of wavelet coefficients given adequate causal information. Due to the state-of-the-art performance demonstrated by neural networks in several prediction and regression tasks over the last decade, a neural network is chosen as the basis of our predictor.

## 2.6 Artificial Neural Networks

*Artificial Neural Networks* (*ANN*s) are computational models, which processes data using a large number of interconnected processing units called *neurons*. A basic *ANN* can be described by Equation 2.14

$$\mathbf{y} = \mathbf{W}_2 f(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 \tag{2.14}$$

where $\mathbf{x}$ is an input vector, $\mathbf{W}_1$ and $\mathbf{W}_2$ are linear transforms, $\mathbf{b}_1$ and $\mathbf{b}_2$ are bias vectors, $f$ is a element-wise non-linear function, and $\mathbf{y}$ is the resulting output vector. Augmenting the otherwise linear transform with $f$, allows representation of non-linear relationships between the input and output vectors. Compared to a standard linear model, the non-linear model is better suited for many real world regression tasks, for which the data is highly non-linear.

Letting $\mathbf{x} = [x_1, ..., x_N]^T$, and $\mathbf{y} = [y_1, ..., y_P]^T$, this transformation can be represented by a directed graph, organized into layers of nodes, with weighted connections joining nodes within adjacent layers. This structure is depicted in Figure 2.4, where each node represents an individual neuron, with $n_i^l$ denoting the $i$th neuron within the $l$th layer. Each black arrow represents a weighted connection between neurons, where labels have been omitted from Figure 2.4 to prevent clutter. We assign the weight $w_{i,j}^l$ to the weighted connection joining the $i$th neuron in the $l$th layer, to the $j$th neuron in the $(l-1)$th layer. These weights are defined within the transform matrices, $\mathbf{W}_1$ and $\mathbf{W}_2$, and determine the input-output relationship of the *ANN*. Each neuron can be seen as taking the weighted sum of its inputs, and applying a non-linear function $f$ to produce an output. *ANNs* with this architecture are often referred to as *fully connected*, due to the interconnectivity of neurons within adjacent layers. The structure of each neuron within a fully connected *ANN* is given in Equation 2.5.

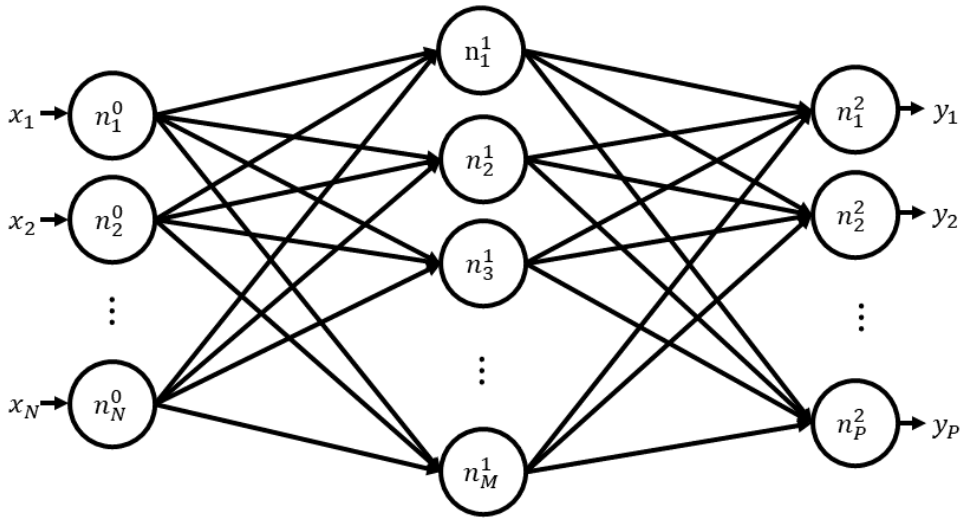


Figure 2.4: Graph representation of the *ANN* defined in Equation 2.14.

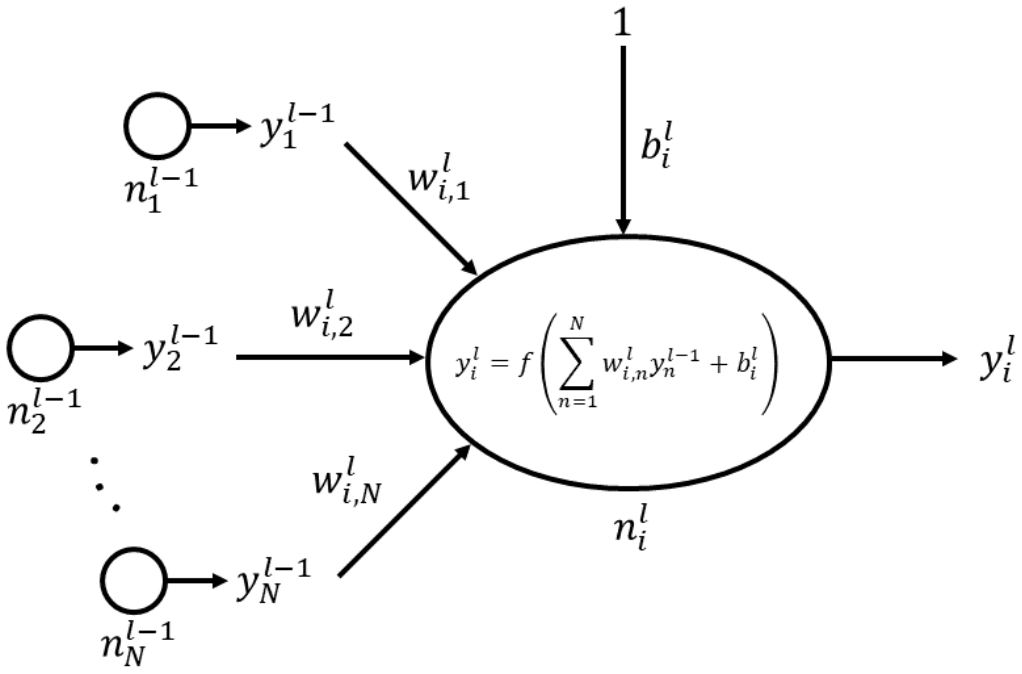


Figure 2.5: Architecture of neurons within an *ANN*.

Certain applications will require a model which can represent more complex non-

linear input-output relationships, this may be achieved through adding additional layers to the *ANN* model. The basic model in Equation 2.14 may be extended to the general form given in Equation 2.15

$$\mathbf{y} = \mathbf{W}_L(...f(\mathbf{W}_2 f(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2)) + ...) + \mathbf{b}_L \tag{2.15}$$

where there are $L$ linear transforms $(\mathbf{W}_1, ..., \mathbf{W}_L)$, each of which is augmented with a non-linear function $f$, with the exception of the last transform $\mathbf{W}_L$. As a directed graph, this is given in Figure 2.6.

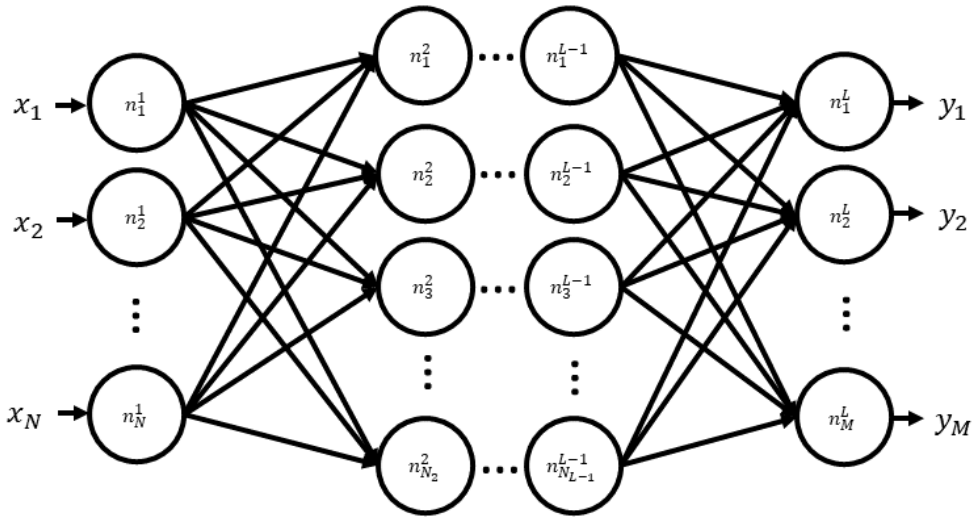


Figure 2.6: Graph representation of a simple $ANN$.

For a typical regression problem, we are given a set of input vectors, $\{\mathbf{x}_p = [x_{p,1}, ..., x_{p,N}],\ p = 1, ..., P\}$, along with a set of corresponding output vectors (or *labels*), $\{\mathbf{y}_p = [y_{p,1}, ..., y_{p,M}],\ p = 1, ..., P\}$. We wish to find an $ANN$ model which will result in the minimum average error when attempting to map each input $(\mathbf{x}_p)$ to each output $(\mathbf{y}_p)$. For a fixed network structure (predefined number of layers and neurons), we define the function $F_{\mathbf{W}}$ as the $ANN$ parameterized by the set of weights $\mathbf{W}$. This minimization may be given by Equation 2.16

$$\arg\min_{\mathbf{W}} \left\{ \frac{1}{P} \sum_{p=1}^{P} E(\mathbf{y}_p, \hat{\mathbf{y}}_p) \right\} \tag{2.16}$$

where $E(,)$ denotes a chosen error function, and $\hat{\mathbf{y}}_p = F_{\mathbf{W}}(\mathbf{x}_p)$ denotes the *ANN* output for a given input $\mathbf{x}_p$. This minimization is solved though an iterative algorithm called *Stochastic Gradient Descent (SGD)* [19]. At each minimization step, the *ANN* is presented with a batch of *training examples* ($\mathbf{s}_p,\ b = 1, ..., B$); where each *training example* consists of an input, $\mathbf{x}_b$, and corresponding output, $\mathbf{y}_b$, so that $\mathbf{s}_b = (\mathbf{x}_b, \mathbf{y}_b)$. Using the batch of training examples, the gradient of the error function ($E$) is approximated using a technique called *backwards propagation of errors* [20]. This gives us the partial derivative of the error with respect to each network weight ($\frac{\partial E}{\partial w_{i,j}}$). Each weight is then updated to minimize the error by moving opposite the error gradient, this is given by Equation 2.17

$$w_{i,j}^{l} = w_{i,j}^{l} - \eta \frac{\partial E}{\partial w_{i,j}^{l}} \tag{2.17}$$

where $\eta$ is called the *learning rate*, and determines the magnitude of the weight update. This procedure is performed for a predefined number of iterations, or until the error can no longer be minimized

### 2.6.1 Convolutional Neural Networks

*Convolutional Neural Networks* ($CNN$s) are a form of $ANN$, where vector transforms are replaced with convolutional filter banks. Networks with this architecture have been successful in numerous applications involving image regression, where the high dimensionality makes training efficiently traditional fully connected networks difficult. For this reason, we choose *CNN*s for subband to subband regression, which shares a nearly identical construction to many image to image regression problems (e.g. super-resolution).

The general structure of a *CNN* is given in Figure 2.7, and of each neuron in Figure 2.8. Each neuron ($n_i^l$) has an associated filter ($w_i^l$), and the neuron output ($y_i^l$) is calculated by convolving this filter with the input ($y^{l-1}$), followed by application of

an element-wise non-linear function ($f$). The value of the weights within each filter will determine the output of the *CNN*. This makes the filter weights analogous to the linear transform weights in the previously dicussed *ANN* architecture. The neuron input will be an array with spatial dimensions $D_{S_{l-1}} \times D_{S_{l-1}}$, with $D_{C_{l-1}}$ channels. The output will be a single channel array, with spatial dimensions $D_{S_l} \times D_{S_l}$. The dimensions of the output array will depend on the filter dimensions, as well as the type of convolution used.

The basic setup of a regression problem for *CNN* is the same as for a traditional *ANN*, with the weights within the convolutional filters are also trained using *SGD*. The previously discussed equations can be applied directly if the *CNN* output and associated label are linearized.

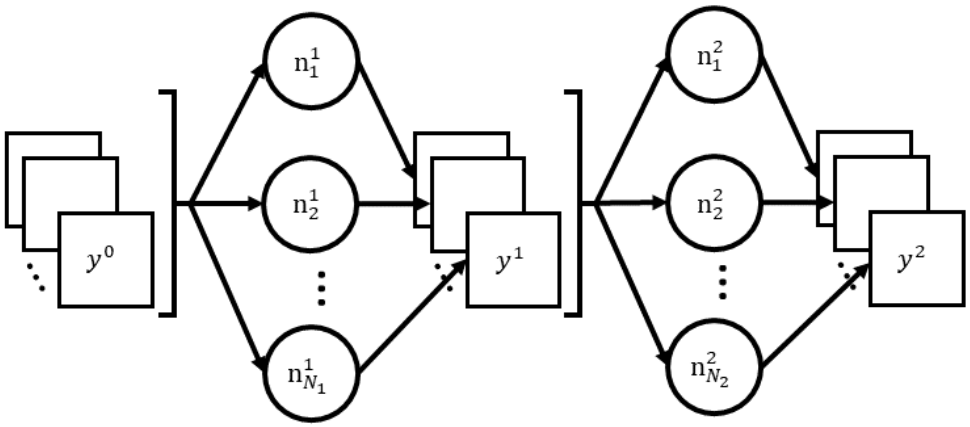


Figure 2.7: Structure of a *convolutional layer*.

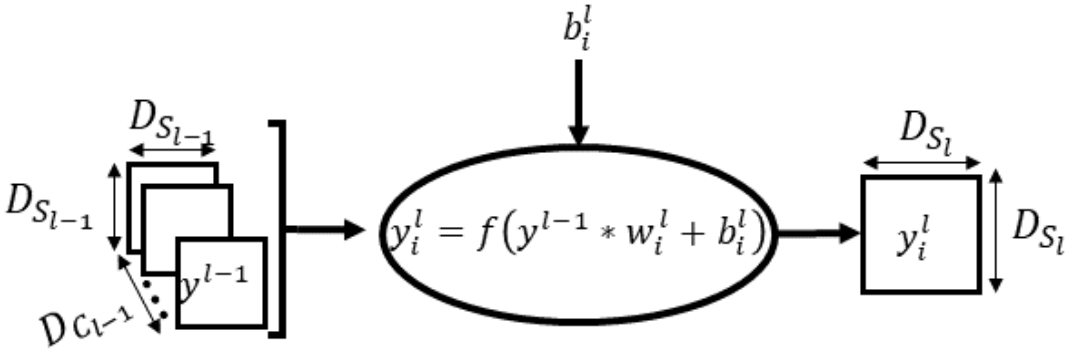


Figure 2.8: Architecture of neurons within a *CNN*.

# CHAPTER 3

# PROPOSED MODEL

## 3.1 Encoding

The proposed encoding framework (Figure 3.1) consists of three primary steps; *DWT*, *CNN Prediction*, and *Entropy Coding*. During the *DWT* step, an input image is decomposed into wavelet subbands.

This is followed by *CNN Prediction*, in which a *CNN* encoding structure ($CNN_{enc}$) predict detail coefficients using coefficients within neighboring subbands; the output of this structure is a *encoding subband*, which contains the information which will be included in the compressed codestream. Finally, a *block entropy coder* is used to produce a compressed codestream. These steps are detailed below in Sections 3.1.1 - 3.1.3 and the full encoding procedure is given in Algorithm 1.

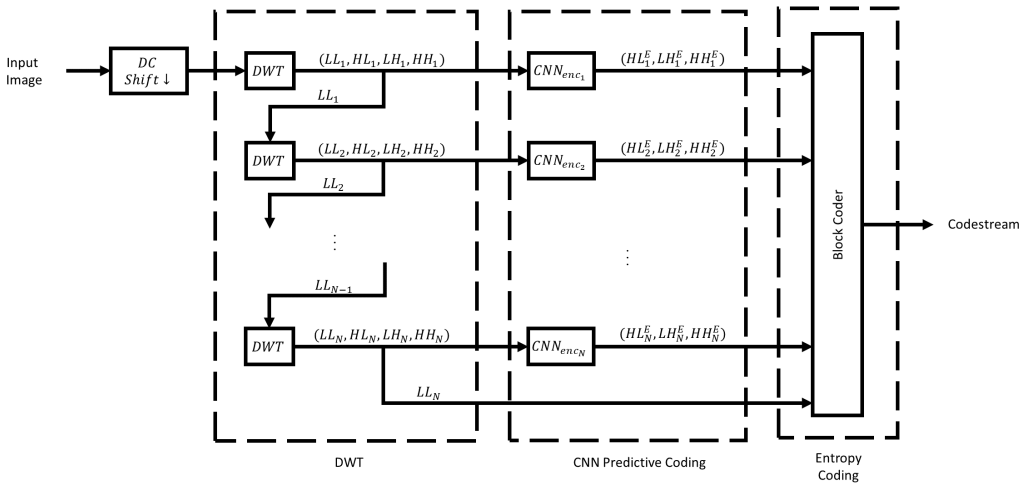


Figure 3.1: Proposed encoding framework.

### 3.1.1 DWT

A single component input image is decomposed into wavelet subbands using a reversible (integer) $DWT$ [21]. Decomposition is performed for a user defined $N$ decomposition steps to produce subbands organized into decomposition levels (See Section 2.4).

### 3.1.2 CNN Prediction

For each decomposition level ($\mathcal{D}_n,\ \ n = 1, ..., N$), the corresponding *CNN* encoding structure ($CNN_{dec_n}$) is used to generate *encoding subbands* ($HL_n^E, LH_n^E, HH_n^E$); with each encoding subband potentially containing a mixture of original coefficients, and prediction residuals. Within $CNN_{enc_n}$, a series of *CNN*s is used to produce subband predictions ($HL_n^P, LH_n^P, HH_n^P$), and residual subbands ($HL_n^R, LH_n^R, HH_n^R$) are determined as the difference between original and predicted coefficients. Corresponding original and residual subbands are then assessed in $C \times C$ coefficient blocks; the block with the lower entropy (Equation 2.3) between the two is inserted into the corresponding encoding subband. A 1-bit flag (called the *block decision flag*) is sent for each block to notify the decoder of the decision made during encoding. This procedure prevents codestream expansion in situations where *CNN* prediction is poor, resulting in residual values which are more difficult to compress than original coefficients. This procedure is called *Subband Block Encoding (SBE)*, and is detailed in Algorithm 2.

We note that no specific *CNN* structure is defined, the only requirement is that a corresponding *CNN* decoding structure ($CNN_{dec}$) is made available during decoding. The decoding structure should invert the steps of the decomposing structure, without requiring any information not readily available at the decoder. In Section 4.1, we explore various *CNN* configurations to determine an optimal prediction framework. Baseline encoding and decoding *CNN* structures are then developed in Section 4.4.

### 3.1.3 Entropy Coding

The blocks within each encoding subband ($HL_n^E, LH_n^E, HH_n^E,\ n = 1, ..., N$) are individually entropy coded. Additionally, the approximation subband ($LL_N$) is entropy coded.

**Algorithm 1** Encoding procedure

**Variables:**
$X \leftarrow$ Input image
$N \leftarrow$ Number of decomposition steps
$B \leftarrow$ Coding Block Size
$CS \leftarrow$ Compressed codestream

**procedure** ENCODE(X,N,B)
  **for** $n$ in 1 to $N$ **do**
    $LL_n, HL_n, LH_n, HH_n \leftarrow \mathbf{DWT}(LL_{n-1})$

    $HL_n^E, LH_n^E, HH_n^E \leftarrow \mathbf{CNN}_{enc_n}(LL_n, HL_n, LH_n, HH_n)$

    $CS \leftarrow \{CS,\ block\_coder(HL_E^C)\}$
    $CS \leftarrow \{CS,\ block\_coder(LH_E^C)\}$
    $CS \leftarrow \{CS,\ block\_coder(HH_E^C)\}$
  **end for**

  $CS \leftarrow \{CS,\ block\_coder(LL_N)\}$
**end procedure**

Notes:

1. $CS \leftarrow \{CS, X\}$ appends the bits in X to the codestream CS

**Algorithm 2** Subband Block Encoding (SBE) Procedure

**Variables:**
$CS \leftarrow$ Codestream
$X \leftarrow$ Original Subband
$\hat{X} \leftarrow$ Subband Prediction
$X^E \leftarrow$ Encoding Subband
$C \leftarrow$ Coding Block Size

**procedure** SBE(CS,X,$\hat{X}$,C)
    $X^R \leftarrow X - \hat{X}$ ▷ Prediction Residuals

    **for** $n_r$ in 0 to $\frac{X.rows}{C}$ **do**
        **for** $n_c$ in 0 to $\frac{X.columns}{C}$ **do**
            $X_{block} \leftarrow X[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1]$
            $X^R_{block} \leftarrow X^R[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1]$
            **if** $entropy(X^R_{block}) < entropy(X_{block})$ **then**
                $X^E.bdf[n_r, n_c] \leftarrow 1$ ▷ Block decision flag
                $X^E[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1] \leftarrow X^R_{block}$
            **else**
                $X^E.bdf[n_r, n_c] \leftarrow 0$ ▷ Block decision flag
                $X^E[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1] \leftarrow X_{block}$
            **end if**
        **end for**
    **end for**
**end procedure**

Notes:

1. $X$ and $X^R$ will have the same dimensions, this dimension is assumed to be an integer power of $C$.
2. $X.rows$ and $X.columns$ are methods which return the number of rows and columns in $X$ respectively.
3. $CS \leftarrow \{CS, x\}$ appends the bits in x to the codestream CS

## 3.2 Decoding

The decoding pipeline is comprised of three primary steps (Figure 3.2); *Decoding*, *CNN Prediction / Subband Reconstruction*, and *Inverse DWT*. Entropy coding is reversed during *decoding* to reproduce the approximation and encoding subbands. *CNN Prediction and Subband Reconstruction* is then performed, where original subbands are recovered using a *CNN* decoding structure ($CNN_{dec}$). Finally, an *inverse DWT* is applied to achieve image reconstruction.

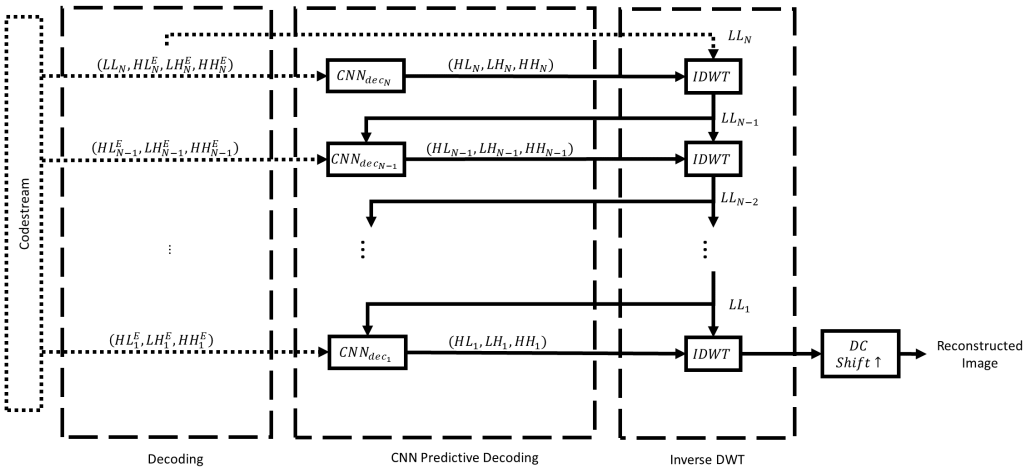


Figure 3.2: Proposed decoding framework.

Starting from the highest encoded decomposition level ($\mathcal{D}_N$), the approximation ($LL_N$) and corresponding encoding subbands ($HL_N^E, LH_N^E, HH_N^E$) are fed into the appropriate *CNN* decoding structure ($CNN_{dec_N}$). Subband predictions ($\hat{HL}_n, \hat{LH}_n, \hat{HH}_n$) are reproduced using an identical series of *CNN*s as was used during encoding. This is followed by *Subband Block Decoding*, in which encoding and prediction subbands are assessed in the same $C \times C$ coefficient blocks as during encoding. Using the *block decision flags* as reference, the encoding and prediction block values will be combined when encoding block values are prediction residuals. The prediction block is otherwise discharged, as the encoding block will already contain original coefficients. After all blocks have been processed, the original subband will be reconstructed. An inverse $DWT$ is then applied to recover the approximation subband ($LL_{N-1}$) corresponding to the next lowest decomposition level ($\mathcal{D}_{N-1}$). The corresponding detail subbands ($HL_{N-1}, LH_{N-1}, HH_{N-1}$) are similarly recovered using an additional *CNN* decoding structure ($CNN_{dec_{N-1}}$), and an inverse $DWT$ is applied to recover the approximation subband ($LL_{N-2}$) at the next lowest decomposition level ($\mathcal{D}_{N-2}$). This procedure is repeated until the subbands at the lowest

decomposition level ($\mathcal{D}_1$) are recovered ($LL_1, HL_1, LH_1, HH1$), after which a final inverse $DWT$ is applied to reconstruct the image.

**Algorithm 3** Decoding procedure

**Variables:**
$CS \leftarrow$ Compressed codestream
$N \leftarrow$ Number of decomposition steps
$X \leftarrow$ Reconstructed Image

**procedure** DECODE(CS)
  **for** $n$ in $N$ to 1 **do**
    $HL_n, LH_n, HH_n \leftarrow \mathbf{CNN}_{rec_n}(LL_n, HL_n^E, LH_n^E, HH_n^E)$
    $LL_{n-1} \leftarrow \mathbf{IDWT}(LL_n, HL_n, LH_n, HH_n)$
  **end for**
  $X \leftarrow LL_0$
**end procedure**

**Algorithm 4** Subband Block Decoding (SBD) Procedure

**Variables:**
$\hat{X} \leftarrow$ Prediction Subband
$X^E \leftarrow$ Encoding Subband
$C \leftarrow$ Block Size
$X \leftarrow$ Original Subband

**procedure** SBD($\hat{X}$,$X^E$,C)
  **for** $n_r$ in 0 to $\frac{\hat{X}.rows}{C}$ **do**
    **for** $n_c$ in 0 to $\frac{\hat{X}.columns}{C}$ **do**
      $X_{block} \leftarrow X^E[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1]$

      **if** $X^E.bdf[n_r, n_c] == 1$ **then**
        $X_{block} \leftarrow X_{block} + \hat{X}[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1]$
      **end if**

      $X[C \cdot n_r : C \cdot (n_r + 1) - 1,\ C \cdot n_c : C \cdot (n_c + 1) - 1] \leftarrow X_{block}$
    **end for**
  **end for**
**end procedure**

Notes:

1. $X.rows$ and $X.columns$ are methods which return the number of rows and columns in $X$ respectively. $X$ and $X^R R$ will have the same dimensions

2. $C$ represents the coding block size

3. $X^E.bdf(,)$ returns the *block decision flag* for block $(n_r, n_c)$

# CHAPTER 4

# EXPERIMENTAL RESULTS

## 4.1 CNN Prediction

In this section, we consider multiple *CNN* prediction models for integration within the previously discussed end-to-end compression framework. A good prediction model will result in sparse prediction residuals, effectively minimizing the information which much be included in the compressed codestream. We divide these models into three categories called *one-to-one*, *many-to-one*, or *one-to-many* based on their input-output configuration (Figure 4.1).

In the *one-to-one* model, a single subband is passed as input to a *CNN* to generate a prediction of a single neighboring subband. For this, we propose a *CNN* which takes an approximation subband as input, and produces a prediction of a single neighboring detail subband as output. A unique *CNN* may be trained to produce a prediction of each detail subband within a given decomposition level; approximation-detail subband interdependencies can then be removed by replacing detail coefficients with prediction residuals.

The *many-to-one* model passes multiple subbands as *CNN* inputs to generate a prediction of a single neighboring subband. For this, we extend the proposed *one-to-one* model by passing one or more detail subbands as *CNN* inputs alongside the approximation subband. The prediction of a single neighboring detail subband is generated as the output. With this extension, we gain the ability to address approximation-detail subband interdependencies in addition to detail-detail subband interdependencies. Greater prediction performance may be seen over the one-to-one model when there is appreciable redundancy between neighboring detail subbands.

The *one-to-many* model takes as input a single subband and provides predictions

of multiple subbands as outputs. Similar to the *one-to-one* model, a single approximation subband is passed as *CNN* input; but in this model, the *CNN* is trained to simultaneously produce predictions of all three neighboring detail subbands as outputs. Because predictions of all three detail subbands may be produced using a single *CNN*, this model has a clear computational advantage over the *one-to-one* model. Additionally, training the *CNN* to make collective predictions may allow it to learn relationships among neighboring detail subbands that can be exploited to produce better predictions, giving it some of the advantages of the *many-to-one* model. On the contrary, it is possible that a degradation in performance may be seen when the *CNN* is not able to sufficiently capture and isolate the non-linear relationship between the input and each output.

While the proposed models offer a large variety of potential *CNN* input-output configurations, for the scope of this work we limit our attention to those provided in Table 4.1. We train networks using each configuration and compare prediction performance. Based off of these results, we then propose a baseline prediction framework.

Table 4.1: CNN Prediction Configurations

| Model | Input | Output |
|---|---|---|
| One-to-One | LL | HL |
| One-to-One | LL | LH |
| One-to-One | LL | HH |
| Many-to-One | LL,LH | HL |
| Many-to-One | LL,HL | LH |
| Many-to-One | LL,HL,LH | HH |
| One-to-Many | LL | HL, LH, HH |

### 4.1.1 *CNN* Architecture

The chosen *CNN* architecture is inspired by those introduced in [22] and is depicted in Figure 4.2. Each network consists of $N_c$ convolutional layers, with each layer containing $N_f$ $3\times3$ filters, with the exception of the output layer which contains $D_{c_{out}}$ $3 \times 3$ filters. The parameters $D_{c_{in}}$ and $D_{c_{out}}$ determine the number of network

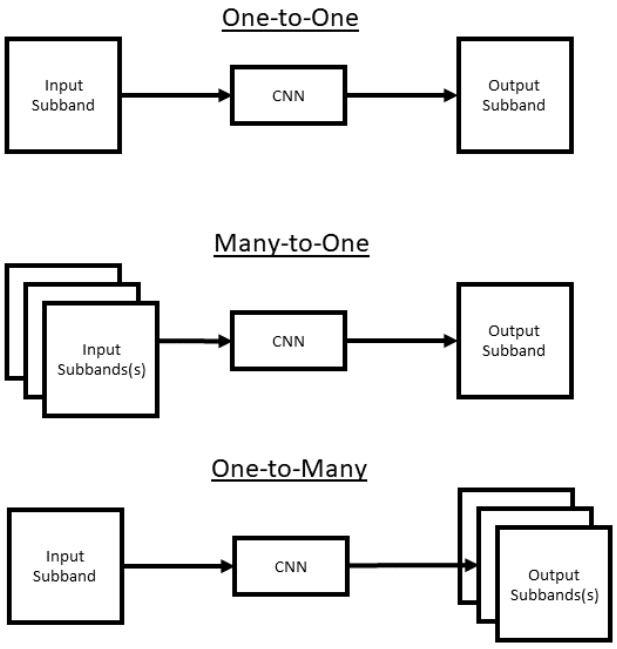


Figure 4.1: Structure of the *one-to-one* (top), *many-to-one* (middle), and *one-to-many* (bottom) prediction models.

inputs and outouts respectively and will vary based off of the configuration being trained (Table 4.1). All hidden layers use ReLU activation, while no activation is used in the output layer.

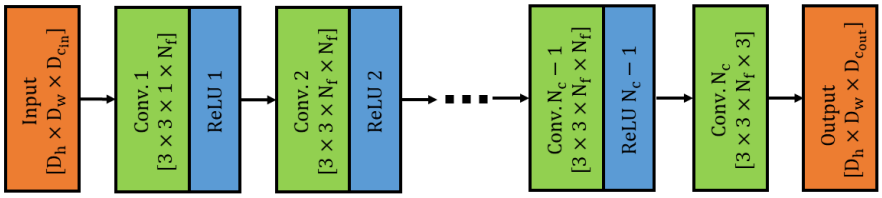


Figure 4.2: The proposed $CNN$ structure where filter dimensions are given as [width×height×channels×filter count].

As a baseline configuration, we find that good results may be obtained by setting the parameters $N_c$ and $N_f$ to 10 and 64, respectively.

### 4.1.2 *CNN* Training

All networks are implemented using the TensorFlow deep learning framework [23]. Training is carried out for 40 epochs using the Adam optimizer [24] with first and second moment estimates of 0.9 and 0.999, respectively. A batch size of $N_{batch} = 64$ with a fixed learning rate and $L_2$ weight regularization ($\lambda$) of 0.0001 are chosen based off of successful implementations of the *CNN* proposed in [25], which shares a similar architecture.

When evaluating network output during training, we would like to minimize the first-order entropy (Equation 2.3) of the prediction residuals. Unfortunately, we do not have an analytical solution for the gradient of this function, and so a surrogate must be used in its place. Both $L1$ (Mean-Absolute-Error) and $L2$ (Mean-Squared-Error) based loss functions have been used successfully in training *CNN* for regression tasks [26][25]. For this reason, we consider both methods as indirect metrics to minimize the first-order entropy. The $L1$ loss is insensitive to outliers, but has multiple solutions and can be instable; while the $L2$ loss is stable with a single solution, but is sensitive to outliers due to squaring of the error. We train networks using both metrics to determine which is better suited for the task; the batch-wise optimization for each is given in Equations 4.1 and 4.2 below.

$$\arg\min_{\mathbf{W}} \left\{ \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \|\mathbf{y}_i - F_{\mathbf{W}}(\mathbf{x}_i)\|_1 + \lambda \|\mathbf{W}\|_2 \right\} \tag{4.1}$$

$$\arg\min_{\mathbf{W}} \left\{ \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \|\mathbf{y}_i - F_{\mathbf{W}}(\mathbf{x}_i)\|_2 + \lambda \|\mathbf{W}\|_2 \right\} \tag{4.2}$$

where $\|\cdot\|_1$ and $\|\cdot\|_2$ respectively represent the $L1$ and $L2$ norms, $F_{\mathbf{W}}$ the *CNN* parameterized by the filter weights $\mathbf{W}$, $\mathbf{x}_i$ the input samples, and $\mathbf{y}_i$ the corresponding labels.

### 4.1.3 Training and Validation Data

Training data is generated by first decomposing each image within a training dataset into subband components using a reversible (integer) $DWT$ [21]. For a network to be trained for prediction at the $n$th decomposition level ($\mathcal{D}_n$), this requires an $n$-level decomposition to obtain the corresponding subbands $(LL_n, HL_n, LH_n, HH_n)$.

When training deep $CNN$ architectures, using input samples with large spatial dimensions can limit the batch size ($N_{batch}$) that may be used before memory resources are exhausted. A small batch size can result in an unstable error gradient and poor network convergence [27]. To avoid these issues, we train on coefficient patches rather than full wavelet subbands. We use training patches roughly twice the size of the theoretical receptive field of neurons within the last layer of the $CNN$. For the chosen network architecture, this leads to a patch size of $41 \times 41$.

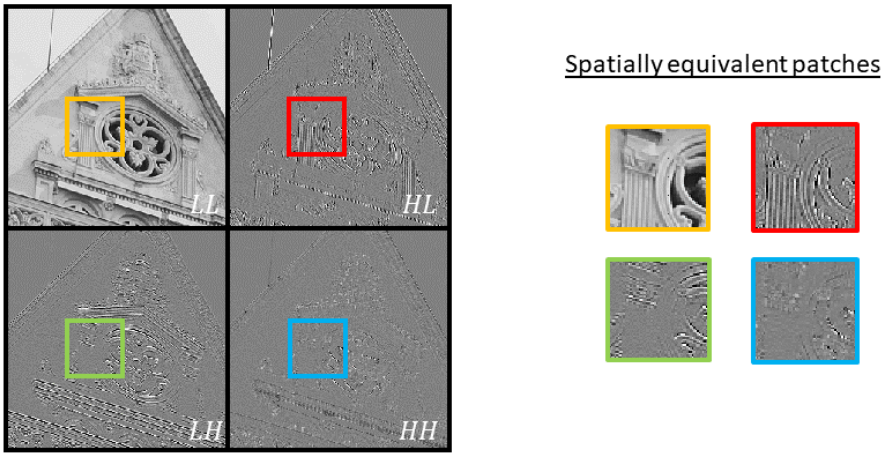


Figure 4.3: Spatially equivalent patches are extracted from each subband.

We train and test networks using data generated by decomposing images in three different datasets, where a separate set of networks is trained for each dataset. The first dataset is the RAISE Raw Image Database [28], which contains 8156 raw high resolution natural scene images. The second is a pathology dataset containing 18,922 pathology images cropped from a dataset of 12 whole-slide pathology images. The

last is a dataset of 23,259 high quality SVG computer graphic images from [29], for usability each image is converted to high resolution PNG using Inkskape [30]. The 8-bit luminance component is extracted from each image, followed by cropping so that all images are $2048 \times 2048$ pixels. We choose 4000 random images from each dataset and split these into individual training and testing sets each containing 2000 images.

For each image in each training dataset, we extract patches as described above. During testing, we perform prediction on full subbands and so no patching is performed. We produce a dataset for each image set at decomposition levels 1 and 2 for a total of 6 training and 6 testing datasets, a summary of these is given in Table 4.2.

Table 4.2: Summary of Training Datasets

| Dataset Set | *DWT* Level | Samples | Sample Dimensions |
|---|---|---|---|
| Natural (Training) | 1 | 1152000 | $41 \times 41$ |
| Natural (Training) | 2 | 288000 | $41 \times 41$ |
| Natural (Testing) | 1 | 2000 | $1024 \times 1024$ |
| Natural (Testing) | 2 | 2000 | $512 \times 512$ |
| Pathology (Training) | 1 | 1152000 | $41 \times 41$ |
| Pathology (Training) | 2 | 288000 | $41 \times 41$ |
| Pathology (Testing) | 1 | 2000 | $1024 \times 1024$ |
| Pathology (Testing) | 2 | 2000 | $512 \times 512$ |
| Graphics (Training) | 1 | 1152000 | $41 \times 41$ |
| Graphics (Training) | 2 | 288000 | $41 \times 41$ |
| Graphics (Testing) | 1 | 2000 | $1024 \times 1024$ |
| Graphics (Testing) | 2 | 2000 | $512 \times 512$ |

The above datasets were chosen due to their unique characteristics; this will help determine the versatility of the proposed prediction models. The *natural* images are captured from a DSLR camera, while the pathology images are captured using a high-resolution slide scanner; the graphics images are not captured, but rather generated in a pure digital environment. Two example images from each dataset are provided in Figures 4.4 - 4.6. From these figures, we see that both the *natural* and

*pathology* images contain sensor noise that is not apparent in the *graphics* images; this results in large regions of constant intensity within the *graphics* images. The $HH_1$ subband of the pathology images is highly contaminated with noise, resulting in no discernible structure. It is possible that predictions performed on this subband will not yield a sparse residual that will result in any significant entropy reduction.

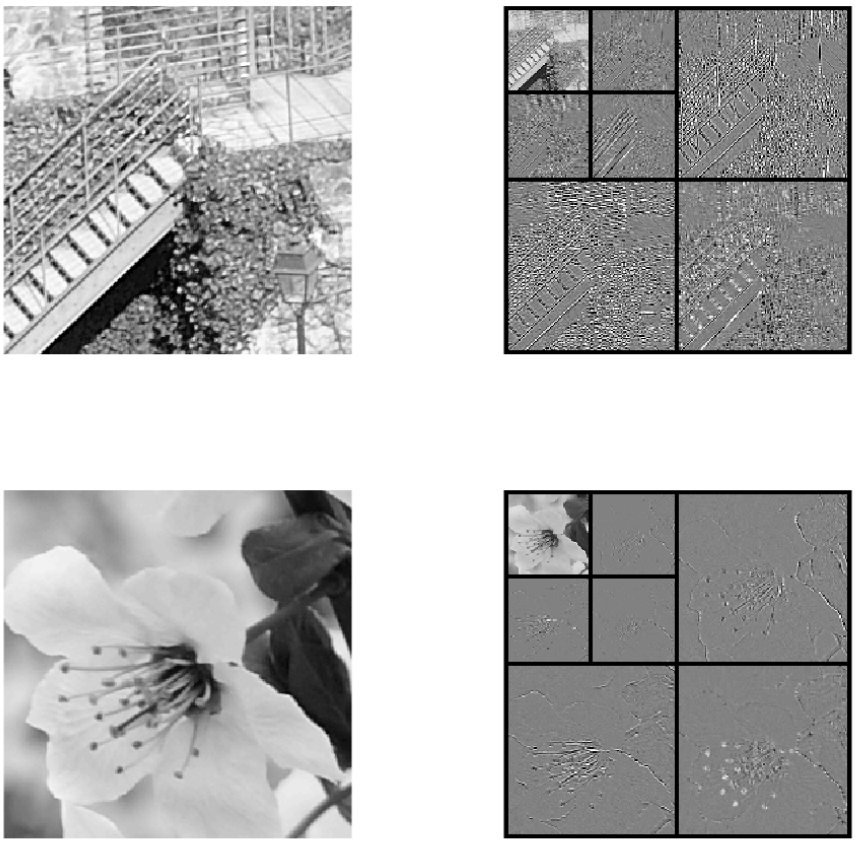

Figure 4.4: Example image and corresponding *2-Level DWT* from the *natural* dataset.

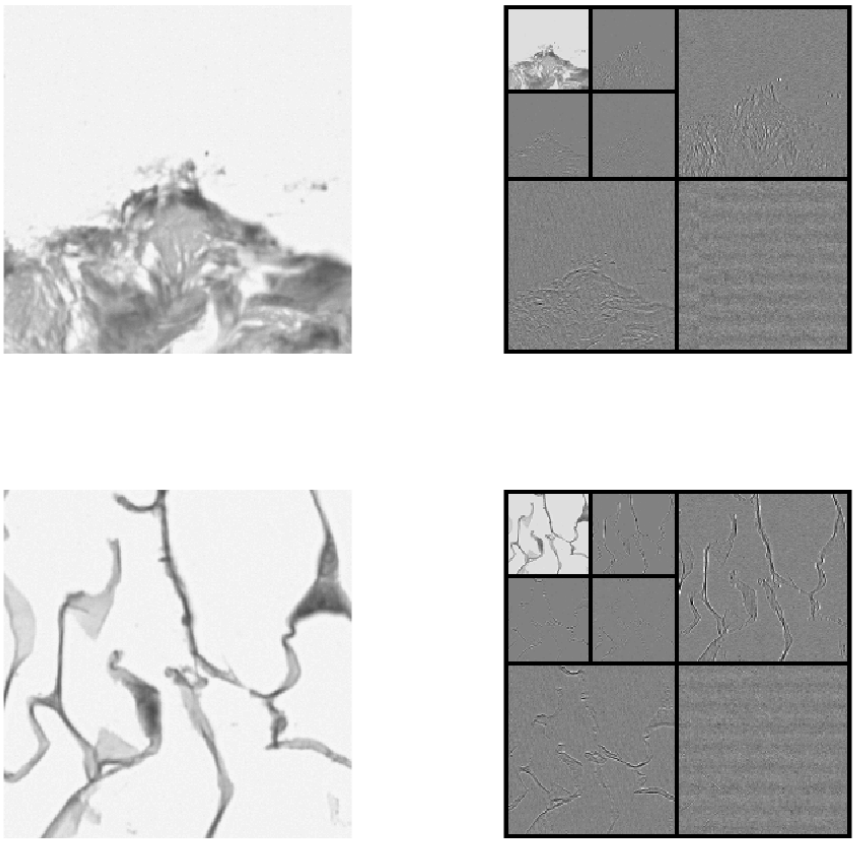

Figure 4.5: Example image and corresponding *2-Level DWT* from the *pathology* dataset.

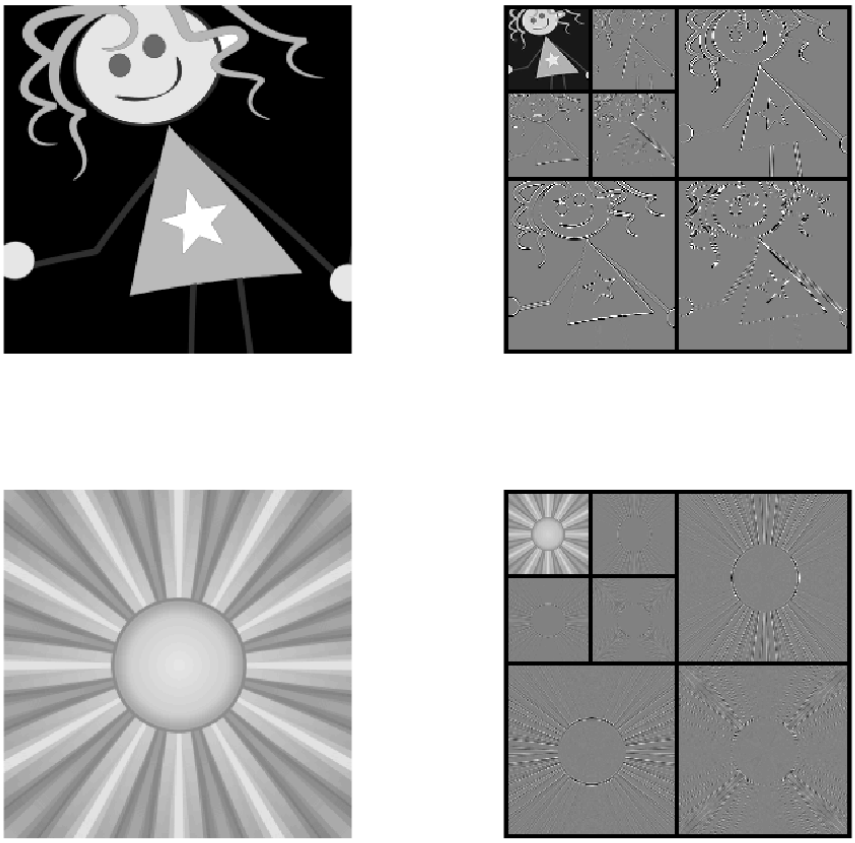

Figure 4.6: Example image and corresponding *2-Level DWT* from the *graphics* dataset.

## 4.2 Network Convergence

Network convergence is assessed by looking at network loss over training epochs. These results are provided in Figures 4.7 and 4.8 for networks trained on $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively. For networks trained in the *one-to-one* and *many-to-one* configurations, this loss represents the average per coefficient error within a single subband. Networks trained using the *one-to-many* configuration will have loss representing average per coefficient error over multiple subbands. Networks trained using $L1$

loss have error given in *Mean Absolute Error (MAE)*, while networks trained using $L2$ loss have error given in *Mean Squared Error (MSE)*. The entropy reduction associated with *CNN* predictions is also calculated at each training epoch. This is given as the difference in first-order entropy (Equation 2.3) between the original coefficients and prediction residuals, so that a positive reduction indicates bit-rate reduction. These plots are provided in Figures 4.9 and 4.10 for subbands in $\mathcal{D}_1$ and $\mathcal{D}_2$, respectively.

The stability of convergence for each network shows a dependence on the choice of loss function, training dataset, and decomposition level. Networks trained using $L2$ loss show large instabilities which are not present in those trained with $L1$ loss. Networks trained on the *pathology* datasets show the least instability, followed by the *natural* datasets, and lastly the *graphics* datasets. To understand this, we look at the distribution of wavelet coefficients within each dataset and across decomposition levels (Figure 4.11). Coefficients within the *natural* and *pathology* datasets follow an approximate Laplacian distribution. Coefficients within the graphics dataset are very tightly centered around 0 with long tails; this results in a tight distribution with a moderate variance. In natural photographs, coefficients with small magnitudes can generally be attributed to noise and subtle textural details. This type of content is not typically present in *graphics* images; as a result, smooth gradient regions result in zero magnitude detail coefficients. Performing prediction on these regions will result in a misleading evaluation of performance, as the network will just be mapping zeros to zeros. This will manifest as large fluctuations in network loss, leading to network convergence which appears unstable. On the other hand, smooth regions within the *natural* and *pathology* datasets will contain noise, which the network will not learn. Prediction on these regions will result in some level of error. This prevents the fluctuations in loss as is seen in the *graphics* data. The difference in stability between $\mathcal{D}_1$ and $\mathcal{D}_2$ can be attributed to the magnitude of coefficients between the subbands, with higher magnitude coefficients being seen in the second level of decomposition. Because $L2$ loss squares error, poor prediction will result in larger fluctuations in the network loss; this leads to increased instability in network convergence.

Minimization of the network loss results in an increase in entropy reduction for all networks. Networks which use the *many-to-one* prediction model, in some cases, show an appreciable gain in entropy reduction over those which use the *one-to-one* model. This is especially apparent in networks trained at the first level of decomposition on *natural* data; here we see an increase of approximately 0.19*bpc* on average. This implies that the detail subbands within a given decomposition level contain some level of redundancy.

Evaluation of entropy reduction within the *one-to-many* model is difficult, as a single value is given for all subbands. Still, it can be observed that as network loss decreases, the reduction in residual entropy increases. In Section 4.3, we evaluate entropy reduction on a per-subband basis over full wavelet subband.

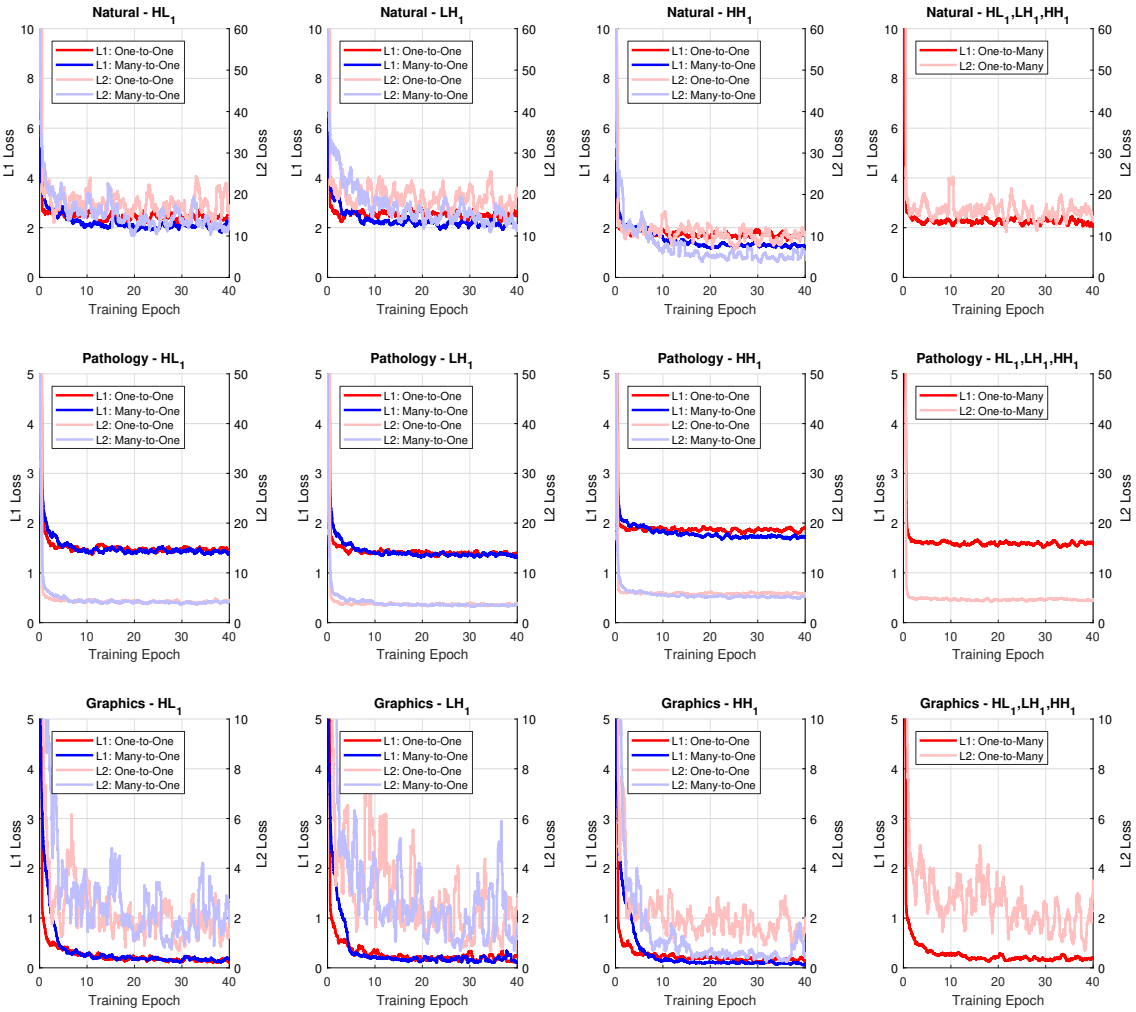


Figure 4.7: Network loss over training epochs for networks trained on the first level of decomposition.

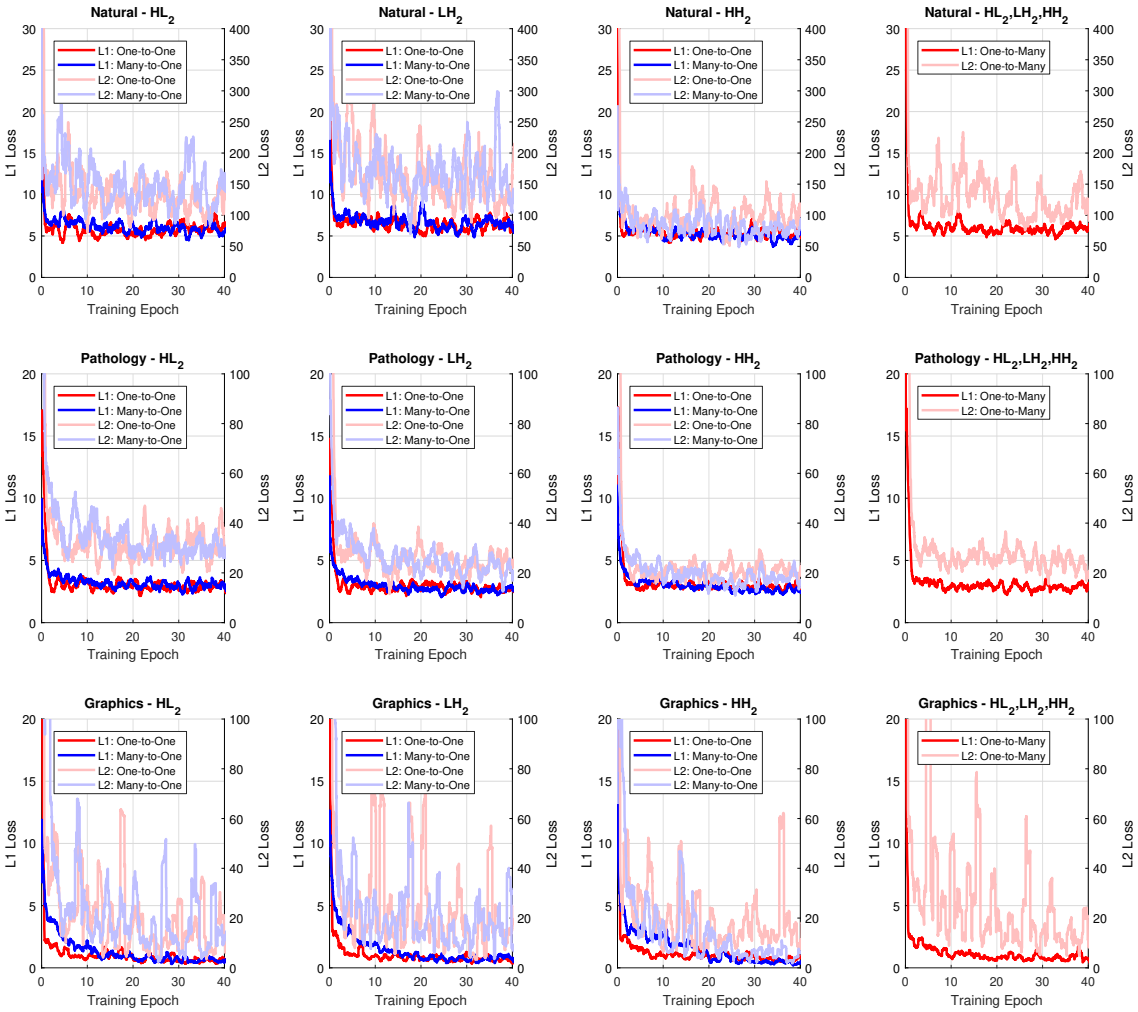


Figure 4.8: Network loss over training epochs for networks trained on the second level of decomposition.

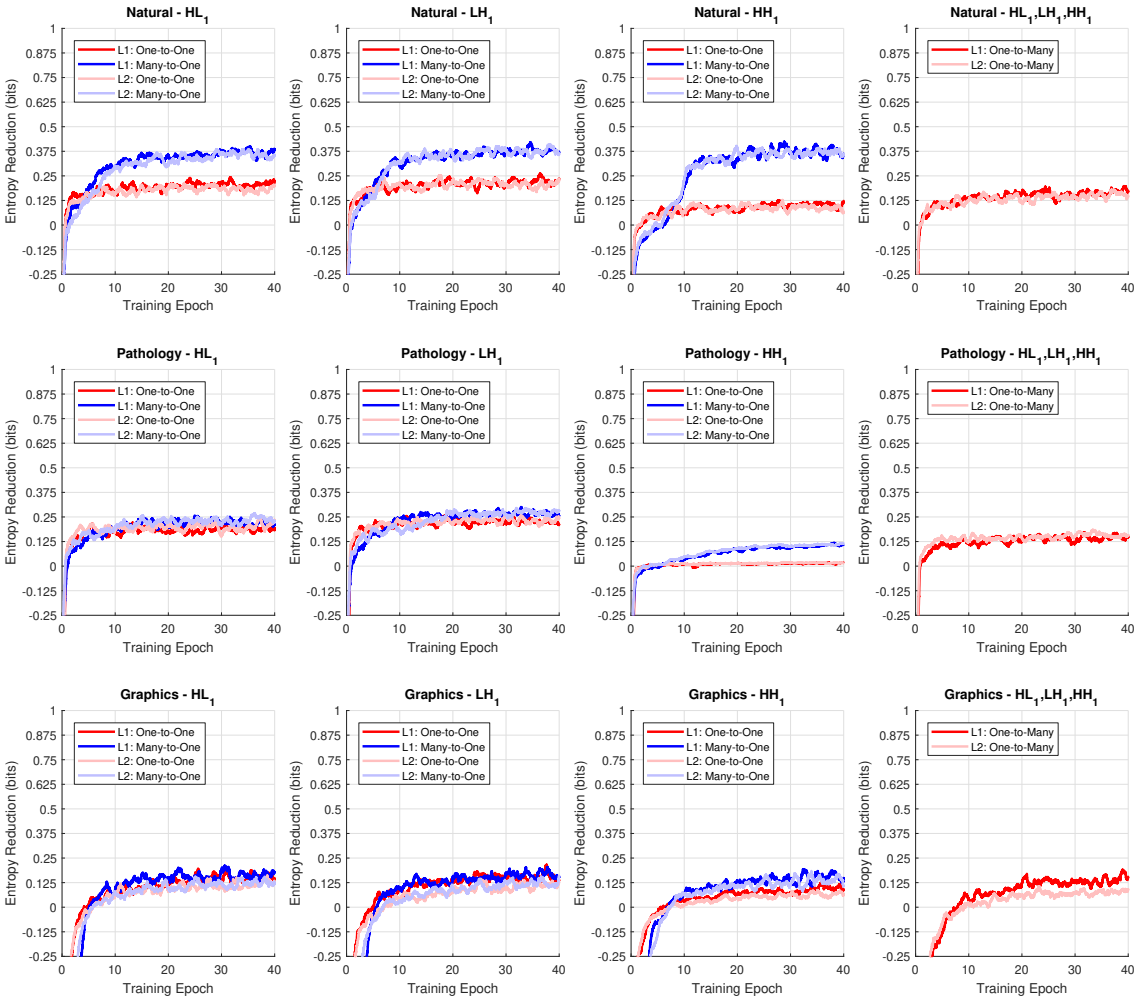


Figure 4.9: Entropy reductions over training epochs for networks trained on the first level of decomposition.

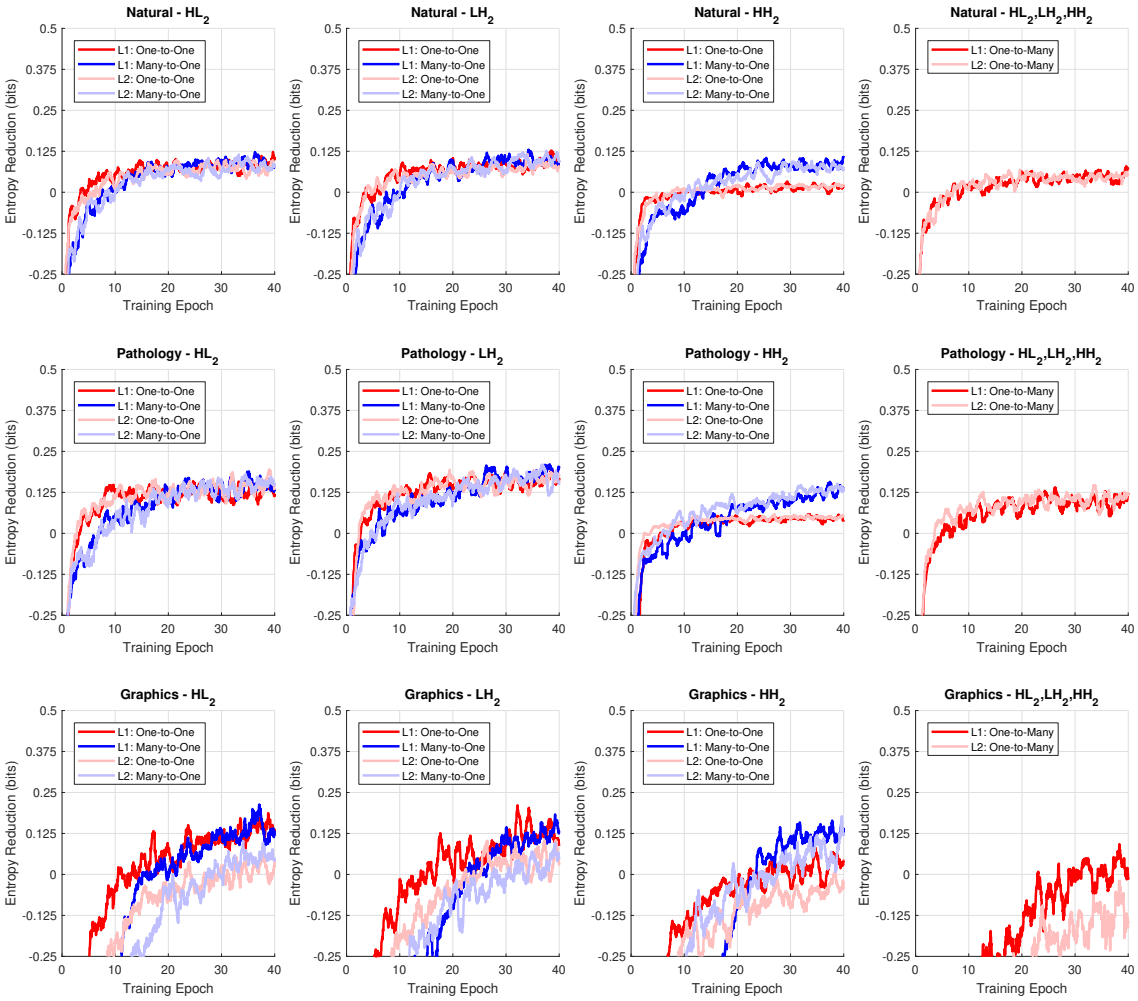


Figure 4.10: Entropy reduction over training epochs for networks trained on the second level of decomposition.

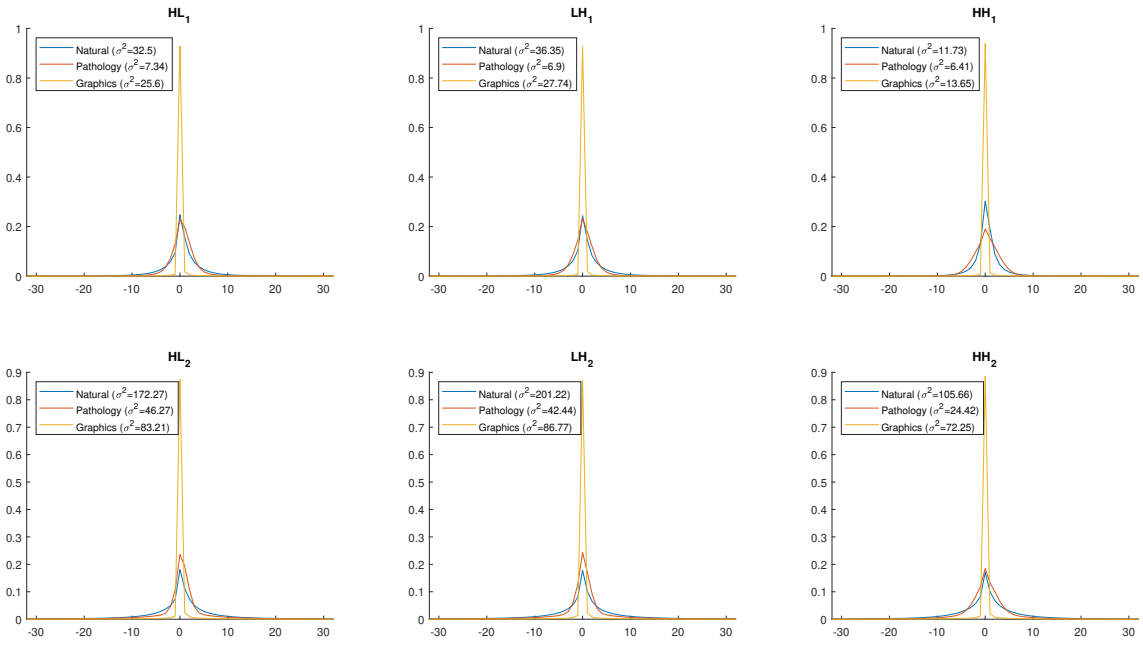


Figure 4.11: Distribution of wavelet coefficient within the first two levels of wavelet decomposition.

## 4.3 Reduction in Entropy

In this section, we consider the difference in entropy between the original wavelet coefficients and prediction residuals over the validation datasets. In the following analysis, entropy should be interpreted as bits-per-coefficient (bpc), since it is calculated on a per-subband basis. Rather than calculating entropy using full subband statistics, subbands are split into $C \times C$ blocks as described in the *Subband Block Encoding (SBE)* (Algorithm 2) and the entropy is calculated individually for each block. The subband entropy is then given as the average entropy over blocks, which provides a more appropriate measure of potential bit-rate reduction since blocks are entropy coded individually during encoding. The average entropy of blocks that are coded by the $SBE$ procedure are also compared to determine the bit-rate reduction that may be attributed to choosing optimal blocks. In both the $SBE$ procedure and the entropy calculations as described above, we use a block size of $C = 64$.

A summary of entropy reduction for $\mathcal{D}_1$ and $\mathcal{D}_2$ in *natural*, *pathology*, and *graphics*

data is provided in Tables 4.3, 4.4, and 4.5, respectively. Distributions of entropy reduction over $\mathcal{D}_1$ and $\mathcal{D}_2$ for *natural* data are given in Figures 4.12 and 4.13, respectively. Similarly, distributions over *pathology* data are shown in Figures 4.14 and 4.15; and distributions over *graphics* data are shown in Figures 4.16 and 4.17. The vertical red line in each figure represents the point in which no entropy reduction occurs, so that codestream expansion occurs to the left of the line.

In all cases, *one-to-one* and *one-to-many* prediction models result in largely similar results. This implies that the *one-to-many* model is sufficiently able to capture and isolate the information about each subband. The *one-to-many* model may then serve as an efficient alternative to the *one-to-one* prediction model. Since using the *one-to-many* model does not result in performance improvements over the *one-to-one* model, it is likely that the network is not exploiting any relationships between subbands to generate its prediction.

In $\mathcal{D}_1$ for *natural* data, prediction using the *many-to-one* model results in significant performance improvements (+0.22 bpc) over the other two models. For *pathology* and *graphics* prediction, only modest gains are seen in $\mathcal{D}_1$ (+0.06 bpc in both cases). In $\mathcal{D}_2$ for *natural* data, $HL_2$ shows no improvement and $LH_2$ shows marginal improvement (+0.03 bpc), though modest improvements are seen in $HH_2$ (+0.09 bpc). In *pathology*, $HL_2$ shows no improvement, $LH_2$ shows marginal improvements (+0.04 bpc), and $HH_2$ show modest improvements (+0.1 bpc). For *graphics*, $HL_2$ shows marginal improvements (+0.02 bpc) and $HH_2$ modest improvements (+0.14 bpc); interestingly, a performance reduction is seen of -0.04 bpc. We point out that the results in Figure 4.8 and 4.10 indicate that the networks used for *graphics* prediction were not fully converged after 40 training epochs. As a result, these values may not completely reflect the optimal performance for the network configuration.

The benefit of the *SBE* procedure is evident when comparing the distributions before and after its application. The *SBE* procedure prevents any residual blocks from being coded which will result in a increase in bitrate. As a result, the average bitrate across an image is at most equal to that of the original coefficients. This

benefit is seen most prominently in results for the *one-to-one* and *one-to-many* models, though there are cases in which the *many-to-one* models also benefit. The benefit of the *SBE* procedure may not be fully appreciated when only considering the average entropy reduction, in Tables 4.3 - 4.5, the minimum and maximum entropy reductions are also provided (shown as [minimum, maximum]). In a majority of cases, the minimum reduction is negative, indicating an increase in bitrate; while after *SBE* the minimum reduction is 0 bpc.

Table 4.3: Entropy Reduction (bpc) - *Natural*

| Input | Output | Original | $L1$ | $L1+SBE$ | $L2$ | $L2+SBE$ |
|---|---|---|---|---|---|---|
| $LL_1$ | $HL_1$ | 3.50 | 0.23 | 0.24 | 0.21 | 0.23 |
| | | | $[-0.12, 0.94]$ | $[0, 0.94]$ | $[-0.27, 0.95]$ | $[0, 0.95]$ |
| $LL_1, LH_1$ | $HL_1$ | 3.50 | 0.42 | 0.42 | 0.41 | 0.41 |
| | | | $[-0.06, 1.06]$ | $[0.02, 1.06]$ | $[-0.06, 0.98]$ | $[0.01, 0.98]$ |
| $LL_1$ | $HL_1$ | 3.50 | 0.22 | 0.22 | 0.20 | 0.22 |
| | (One-to-Many) | | $[-0.19, 0.89]$ | $[0, 0.89]$ | $[-0.31, 0.84]$ | $[0, 0.84]$ |
| $LL_1$ | $LH_1$ | 3.57 | 0.25 | 0.25 | 0.25 | 0.25 |
| | | | $[-0.15, 1.16]$ | $[0, 1.16]$ | $[-0.12, 1.2]$ | $[0, 1.2]$ |
| $LL_1, HL_1$ | $LH_1$ | 3.57 | 0.43 | 0.43 | 0.44 | 0.44 |
| | | | $[-0.14, 1.22]$ | $[0.02, 1.22]$ | $[-0.14, 1.27]$ | $[0.02, 1.27]$ |
| $LL_1$ | $LH_1$ | 3.57 | 0.24 | 0.24 | 0.21 | 0.23 |
| | (One-to-Many) | | $[-0.1, 1.14]$ | $[0, 1.14]$ | $[-0.23, 1.07]$ | $[0, 1.07]$ |
| $LL_1$ | $HH_1$ | 2.95 | 0.13 | 0.16 | 0.13 | 0.16 |
| | | | $[-0.63, 0.69]$ | $[0, 0.69]$ | $[-0.61, 0.68]$ | $[0, 0.68]$ |
| $LL_1, HL_1, LH_1$ | $HH_1$ | 2.95 | 0.43 | 0.43 | 0.44 | 0.44 |
| | | | $[-0.07, 1.03]$ | $[0, 1.03]$ | $[-0.05, 1.05]$ | $[0, 1.05]$ |
| $LL_1$ | $HH_1$ | 2.95 | 0.12 | 0.16 | 0.11 | 0.13 |
| | (One-to-Many) | | $[-0.57, 0.65]$ | $[0, 0.65]$ | $[-0.53, 0.55]$ | $[0, 0.55]$ |
| $LL_2$ | $HL_2$ | 4.39 | 0.12 | 0.12 | 0.11 | 0.11 |
| | | | $[-0.01, 0.58]$ | $[0, 0.58]$ | $[-0.03, 0.53]$ | $[0, 0.53]$ |
| $LL_2, LH_2$ | $HL_2$ | 4.39 | 0.12 | 0.13 | 0.12 | 0.12 |
| | | | $[-0.16, 0.60]$ | $[0, 0.60]$ | $[-0.14, 0.56]$ | $[0.12, 0.56]$ |
| $LL_2$ | $HL_2$ | 4.39 | 0.11 | 0.12 | 0.09 | 0.10 |
| | (One-to-Many) | | $[-0.01, 0.58]$ | $[0, 0.58]$ | $[-0.41, 0.46]$ | $[0, 0.46]$ |
| $LL_2$ | $LH_2$ | 4.49 | 0.10 | 0.11 | 0.12 | 0.12 |
| | | | $[-0.36, 0.54]$ | $[0, 0.54]$ | $[-0.36, 0.52]$ | $[0, 0.52]$ |
| $LL_2, HL_2$ | $LH_2$ | 4.49 | 0.13 | 0.13 | 0.12 | 0.12 |
| | | | $[-0.24, 0.65]$ | $[0, 0.65]$ | $[-0.29, 0.55]$ | $[0, 0.55]$ |
| $LL_2$ | $LH_2$ | 4.49 | 0.12 | 0.12 | 0.10 | 0.10 |
| | (One-to-Many) | | $[-0.30, 0.60]$ | $[0, 0.60]$ | $[-0.36, 0.47]$ | $[0, 0.47]$ |
| $LL_2$ | $HH_2$ | 4.33 | 0.03 | 0.03 | 0.03 | 0.03 |
| | | | $[-0.10, 0.30]$ | $[0, 0.30]$ | $[-0.19, 0.25]$ | $[0, 0.25]$ |
| $LL_2, HL_2, LH_2$ | $HH_2$ | 4.33 | 0.12 | 0.12 | 0.09 | 0.11 |
| | | | $[-0.01, 0.66]$ | $[0, 0.66]$ | $[-0.39, 0.65]$ | $[0, 0.65]$ |
| $LL_2$ | $HH_2$ | 4.33 | 0.03 | 0.03 | 0.02 | 0.02 |
| | (One-to-Many) | | $[-0.1, 0.37]$ | $[0, 0.37]$ | $[-0.35, 0.25]$ | $[0, 0.26]$ |

Table 4.4: Entropy Reduction (bpc) - *Pathology*

| Input | Output | Original | $L1$ | $L1+SBE$ | $L2$ | $L2+SBE$ |
|---|---|---|---|---|---|---|
| $LL_1$ | $HL_1$ | 3.09 | 0.22 | 0.23 | 0.23 | 0.23 |
| | | | $[-0.01, 0.77]$ | $[0, 0.77]$ | $[-0.09, 0.77]$ | $[0, 0.77]$ |
| $LL_1, LH_1$ | $HL_1$ | 3.09 | 0.25 | 0.26 | 0.26 | 0.26 |
| | | | $[-0.04, 0.81]$ | $[0, 0.81]$ | $[-0.04, 0.81]$ | $[0, 0.81]$ |
| $LL_1$ | $HL_1$ (One-to-Many) | 3.09 | 0.22 | 0.23 | 0.22 | 0.22 |
| | | | $[-0.11, 0.76]$ | $[0, 0.76]$ | $[-0.06, 0.76]$ | $[0, 0.76]$ |
| $LL_1$ | $LH_1$ | 3.10 | 0.27 | 0.27 | 0.28 | 0.28 |
| | | | $[-0.02, 0.80]$ | $[0, 0.80]$ | $[-0.01, 0.80]$ | $[0, 0.8]$ |
| $LL_1, HL_1$ | $LH_1$ | 3.10 | 0.31 | 0.32 | 0.32 | 0.32 |
| | | | $[0, 0.86]$ | $[0, 0.86]$ | $[0, 0.87]$ | $[0, 0.87]$ |
| $LL_1$ | $LH_1$ (One-to-Many) | 3.10 | 0.27 | 0.27 | 0.27 | 0.27 |
| | | | $[-0.01, 0.79]$ | $[0, 0.79]$ | $[-0.02, 0.79]$ | $[0, 0.79]$ |
| $LL_1$ | $HH_1$ | 3.28 | 0.02 | 0.02 | 0.02 | 0.02 |
| | | | $[-0.01, 0.09]$ | $[0, 0.09]$ | $[-0.04, 0.09]$ | $[0, 0.09]$ |
| $LL_1, HL_1, LH_1$ | $HH_1$ | 3.28 | 0.14 | 0.14 | 0.14 | 0.14 |
| | | | $[0, 0.24]$ | $[0, 0.24]$ | $[0, 0.25]$ | $[0, 0.25]$ |
| $LL_1$ | $HH_1$ (One-to-Many) | 3.28 | 0.02 | 0.02 | 0.02 | 0.02 |
| | | | $[0, 0.09]$ | $[0, 0.09]$ | $[-0.02, 0.09]$ | $[0, 0.09]$ |
| $LL_2$ | $HL_2$ | 3.53 | 0.16 | 0.17 | 0.16 | 0.17 |
| | | | $[-0.31, 0.50]$ | $[0, 0.50]$ | $[-0.03, 0.51]$ | $[0, 0.51]$ |
| $LL_2, LH_2$ | $HL_2$ | 3.53 | 0.16 | 0.17 | 0.15 | 0.18 |
| | | | $[-0.15, 0.55]$ | $[0, 0.55]$ | $[-0.43, 0.57]$ | $[0, 0.57]$ |
| $LL_2$ | $HL_2$ (One-to-Many) | 3.53 | 0.15 | 0.15 | 0.17 | 0.17 |
| | | | $[-0.02, 0.48]$ | $[0, 0.48]$ | $[-0.01, 0.51]$ | $[0, 0.51]$ |
| $LL_2$ | $LH_2$ | 3.54 | 0.17 | 0.18 | 0.18 | 0.18 |
| | | | $[-0.06, 0.49]$ | $[0, 0.49]$ | $[-0.01, 0.49]$ | $[0, 0.49]$ |
| $LL_2, HL_2$ | $LH_2$ | 3.54 | 0.21 | 0.21 | 0.20 | 0.20 |
| | | | $[-0.02, 0.59]$ | $[0, 0.59]$ | $[-0.03, 0.59]$ | $[0, 0.59]$ |
| $LL_2$ | $LH_2$ (One-to-Many) | 3.54 | 0.17 | 0.17 | 0.18 | 0.18 |
| | | | $[-0.01, 0.48]$ | $[0, 0.48]$ | $[-0.01, 0.52]$ | $[0, 0.52]$ |
| $LL_2$ | $HH_2$ | 3.70 | 0.06 | 0.06 | 0.06 | 0.06 |
| | | | $[-0.05, 0.20]$ | $[0, 0.20]$ | $[-0.01, 0.19]$ | $[0, 0.19]$ |
| $LL_2, HL_2, LH_2$ | $HH_2$ | 3.70 | 0.16 | 0.16 | 0.17 | 0.17 |
| | | | $[-0.03, 0.48]$ | $[0, 0.48]$ | $[-0.01, 0.49]$ | $[0, 0.49]$ |
| $LL_2$ | $HH_2$ (One-to-Many) | 3.70 | 0.03 | 0.04 | 0.06 | 0.06 |
| | | | $[-0.22, 0.15]$ | $[0, 0.15]$ | $[-0.01, 0.20]$ | $[0, 0.20]$ |

Table 4.5: Entropy Reduction (bpc) - *Graphics*

| Input | Output | Original | $L1$ | $L1+SBE$ | $L2$ | $L2+SBE$ |
|---|---|---|---|---|---|---|
| $LL_1$ | $HL_1$ | 0.53 | 0.07 | 0.10 | 0.15 | 0.15 |
| | | | $[-0.37, 2.15]$ | $[0, 2.15]$ | $[-0.35, 2.46]$ | $[0, 2.46]$ |
| $LL_1, LH_1$ | $HL_1$ | 0.53 | 0.20 | 0.21 | 0.16 | 0.17 |
| | | | $[-0.21, 2.62]$ | $[0, 2.62]$ | $[-0.30, 2.48]$ | $[0, 2.48]$ |
| $LL_1$ | $HL_1$ | 0.53 | 0.19 | 0.19 | 0.12 | 0.13 |
| | (One-to-Many) | | $[-0.28, 2.60]$ | $[0, 2.60]$ | $[-0.48, 2.04]$ | $[0, 2.04]$ |
| $LL_1$ | $LH_1$ | 0.55 | 0.17 | 0.18 | 0.16 | 0.16 |
| | | | $[-0.18, 2.50]$ | $[0, 2.50]$ | $[-0.34, 2.39]$ | $[0, 2.39]$ |
| $LL_1, HL_1$ | $LH_1$ | 0.55 | 0.19 | 0.19 | 0.18 | 0.18 |
| | | | $[-0.21, 2.55]$ | $[0, 2.55]$ | $[-0.27, 2.53]$ | $[0, 2.53]$ |
| $LL_1$ | $LH_1$ | 0.55 | 0.18 | 0.18 | 0.13 | 0.13 |
| | (One-to-Many) | | $[-0.27, 2.40]$ | $[0, 2.40]$ | $[-0.46, 1.94]$ | $[0, 1.94]$ |
| $LL_1$ | $HH_1$ | 0.45 | 0.10 | 0.10 | 0.08 | 0.08 |
| | | | $[-0.12, 1.71]$ | $[0, 1.71]$ | $[-0.18, 1.48]$ | $[0, 1.48]$ |
| $LL_1, HL_1, LH_1$ | $HH_1$ | 0.45 | 0.13 | 0.13 | 0.17 | 0.17 |
| | | | $[-0.08, 2.19]$ | $[0, 2.19]$ | $[-0.13, 2.23]$ | $[0, 2.23]$ |
| $LL_1$ | $HH_1$ | 0.45 | 0.12 | 0.12 | 0.04 | 0.06 |
| | (One-to-Many) | | $[-0.27, 2.15]$ | $[0, 2.15]$ | $[-0.66, 1.25]$ | $[0, 1.27]$ |
| $LL_2$ | $HL_2$ | 0.94 | 0.17 | 0.18 | 0.06 | 0.12 |
| | | | $[-0.33, 2.49]$ | $[0, 2.49]$ | $[-0.45, 2.34]$ | $[0, 2.34]$ |
| $LL_2, LH_2$ | $HL_2$ | 0.94 | 0.19 | 0.19 | 0.16 | 0.18 |
| | | | $[-0.36, 2.86]$ | $[0, 2.86]$ | $[-0.45, 2.44]$ | $[0, 2.44]$ |
| $LL_2$ | $HL_2$ | 0.94 | 0.16 | 0.16 | 0.15 | 0.16 |
| | (One-to-Many) | | $[-0.27, 2.32]$ | $[0, 2.32]$ | $[-0.91, 2.36]$ | $[0, 2.36]$ |
| $LL_2$ | $LH_2$ | 0.96 | 0.22 | 0.22 | 0.14 | 0.15 |
| | | | $[-0.30, 2.35]$ | $[0, 2.35]$ | $[-0.37, 2.04]$ | $[0, 2.04]$ |
| $LL_2, HL_2$ | $LH_2$ | 0.96 | 0.18 | 0.19 | 0.18 | 0.18 |
| | | | $[-0.34, 2.7]$ | $[0, 2.7]$ | $[-0.34, 2.59]$ | $[0, 2.59]$ |
| $LL_2$ | $LH_2$ | 0.96 | 0.15 | 0.15 | 0.14 | 0.16 |
| | (One-to-Many) | | $[-0.29, 2.39]$ | $[0, 2.39]$ | $[-0.43, 2.28]$ | $[0, 2.28]$ |
| $LL_2$ | $HH_2$ | 0.86 | 0.05 | 0.10 | 0.05 | 0.08 |
| | | | $[-0.57, 2.14]$ | $[0, 2.14]$ | $[-0.62, 2.06]$ | $[0, 2.06]$ |
| $LL_2, HL_2, LH_2$ | $HH_2$ | 0.86 | 0.19 | 0.19 | 0.03 | 0.12 |
| | | | $[-0.40, 2.69]$ | $[0, 2.69]$ | $[-0.67, 3.02]$ | $[0, 3.02]$ |
| $LL_2$ | $HH_2$ | 0.86 | 0.06 | 0.08 | 0.06 | 0.09 |
| | (One-to-Many) | | $[-0.68, 2.04]$ | $[0, 2.04]$ | $[-1.07, 2.21]$ | $[0, 2.21]$ |

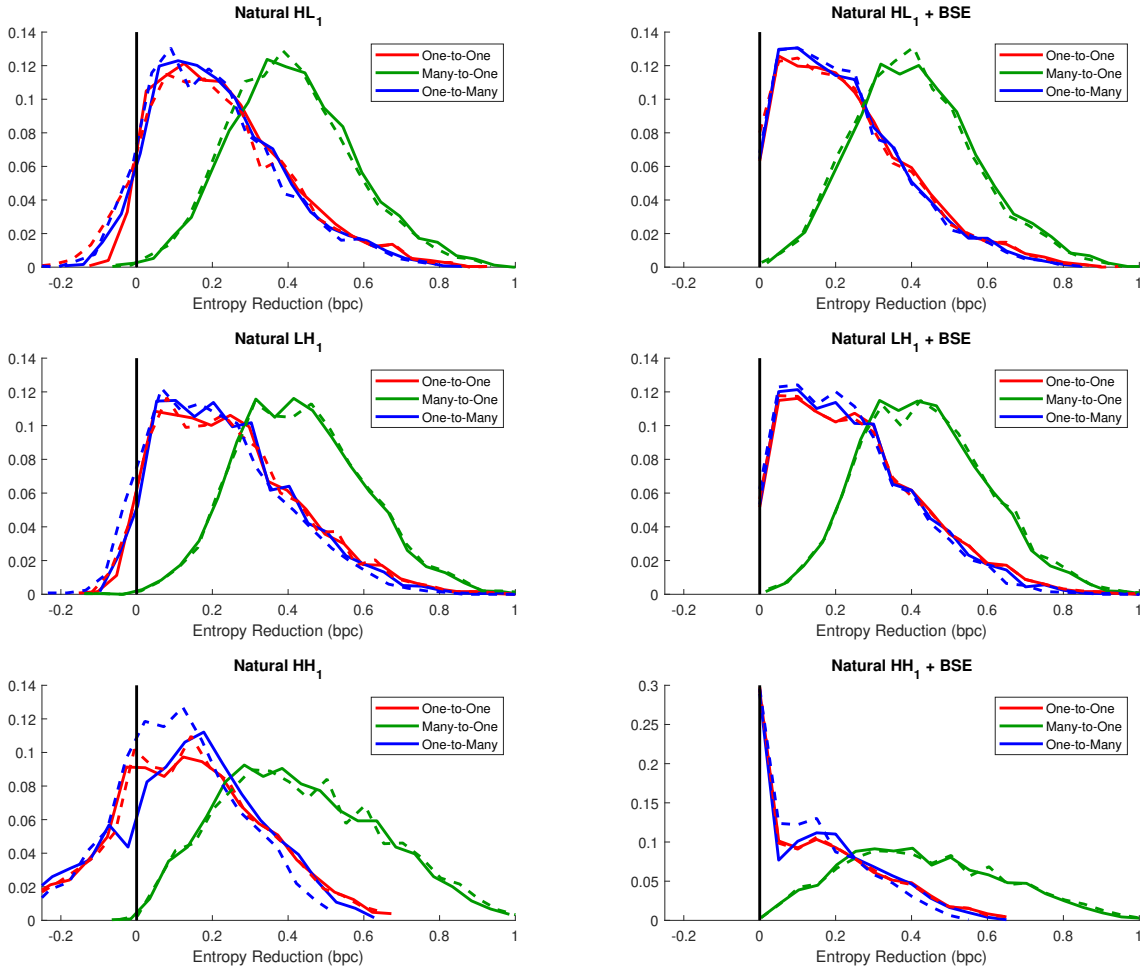


Figure 4.12: Distributions of entropy reduction over the *natural* dataset for subbnads in the 1st decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

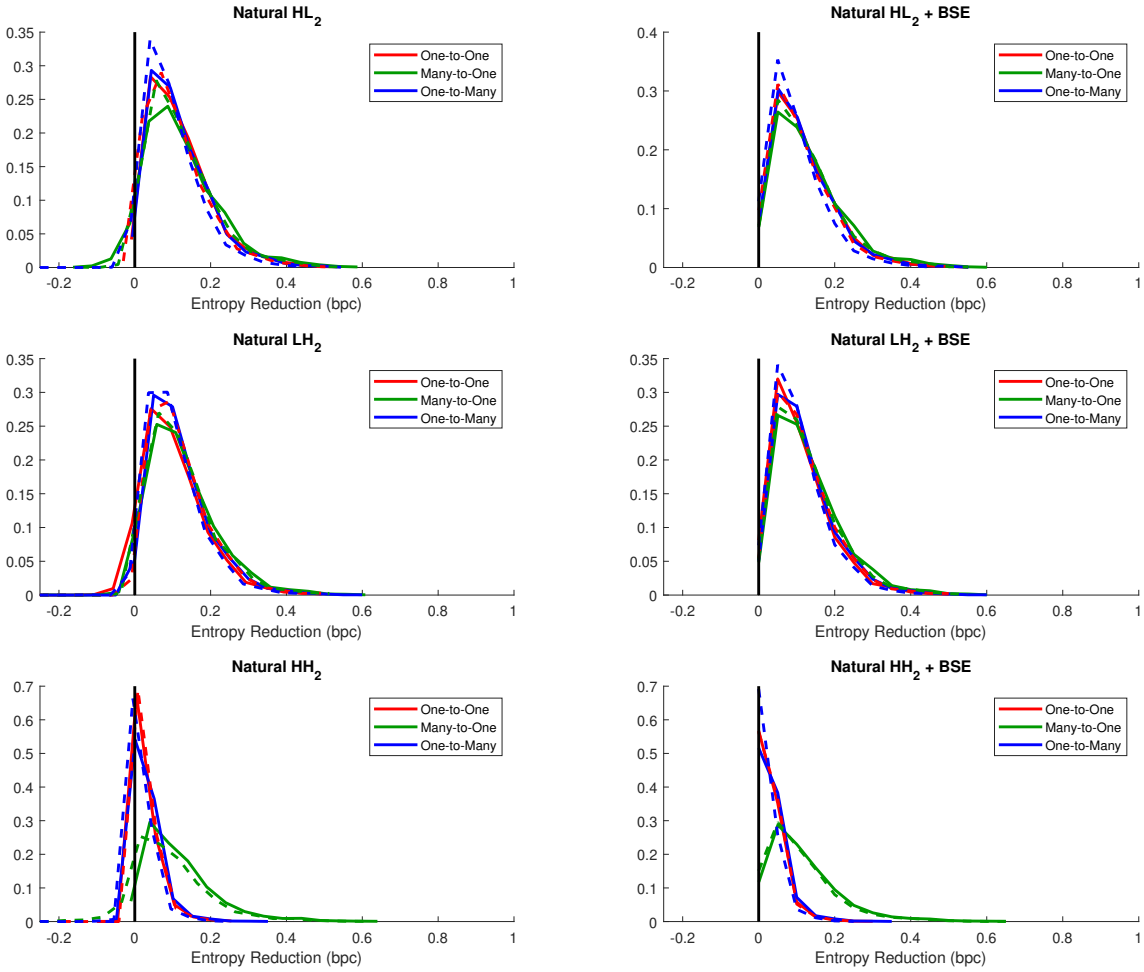


Figure 4.13: Distributions of entropy reduction over the *natural* dataset for subbnads in the 2nd decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

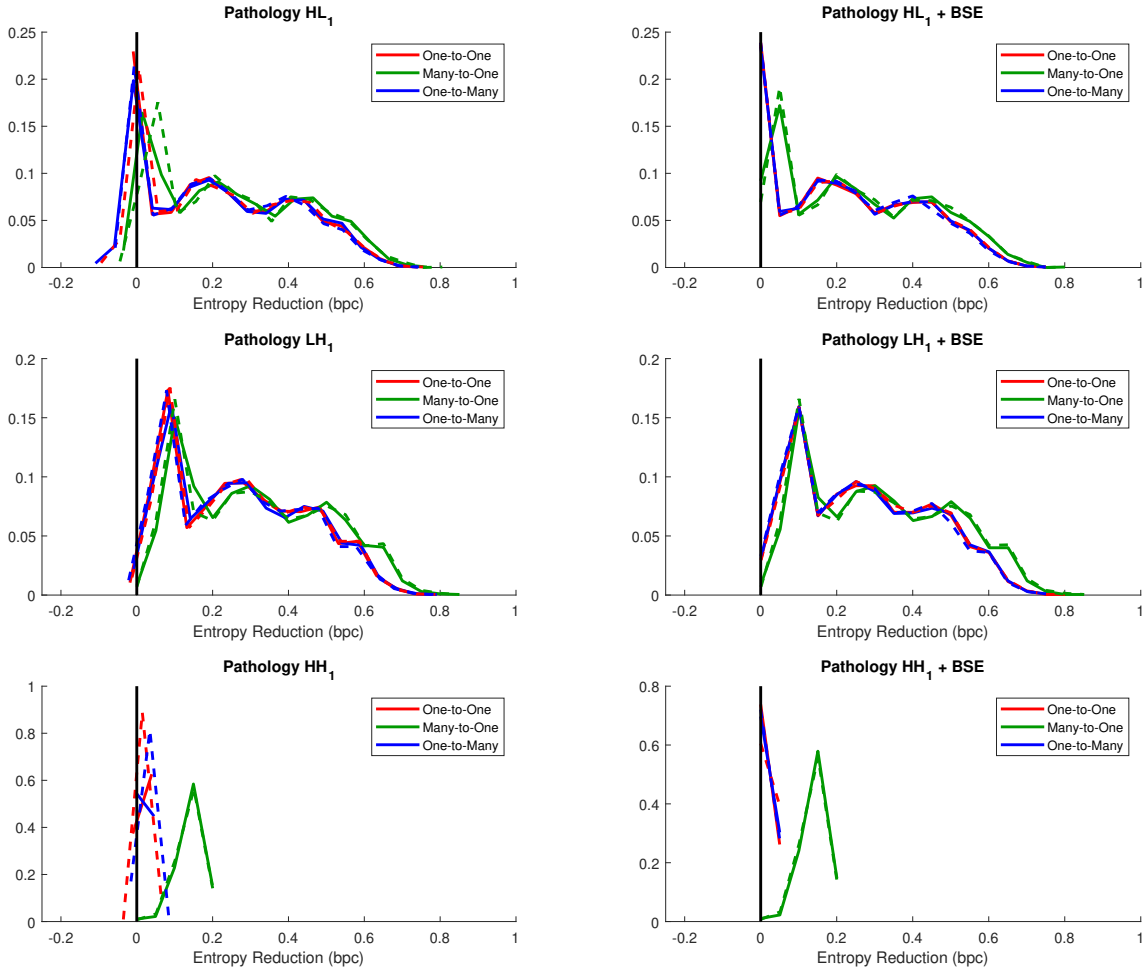


Figure 4.14: Distributions of entropy reduction over the *pathology* dataset for subbnads in the 1st decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

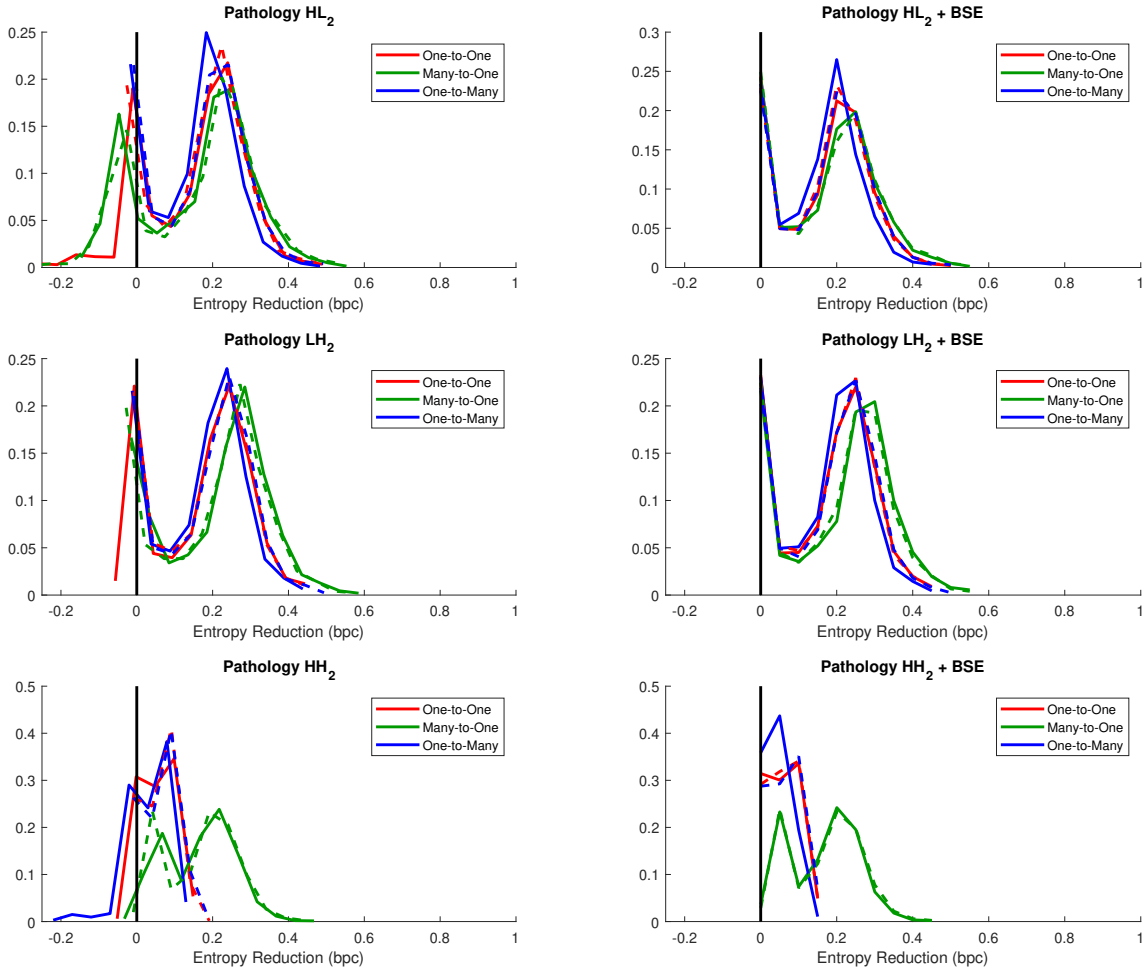


Figure 4.15: Distributions of entropy reduction over the *pathology* dataset for subbnads in the 2nd decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

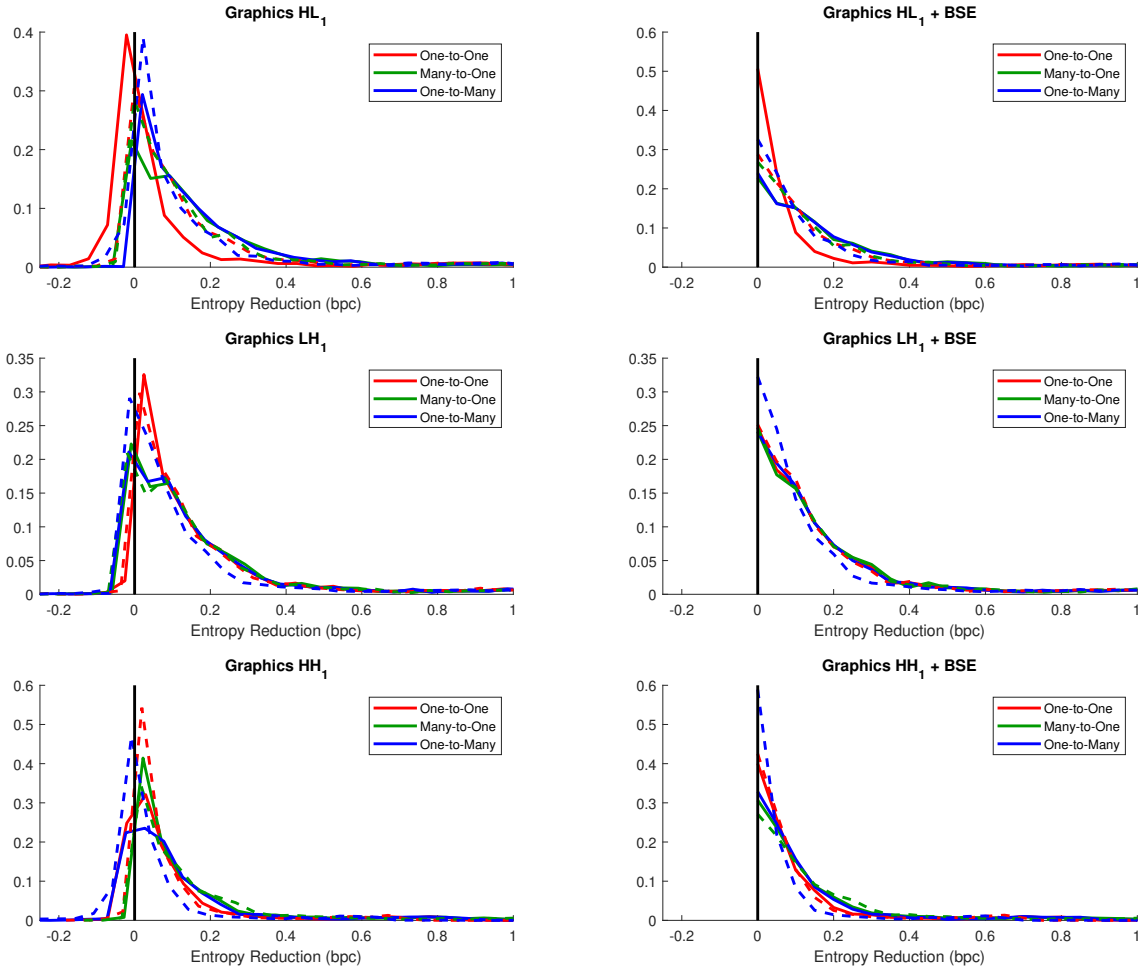


Figure 4.16: Distributions of entropy reduction over the *graphics* dataset for subbnads in the 1st decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

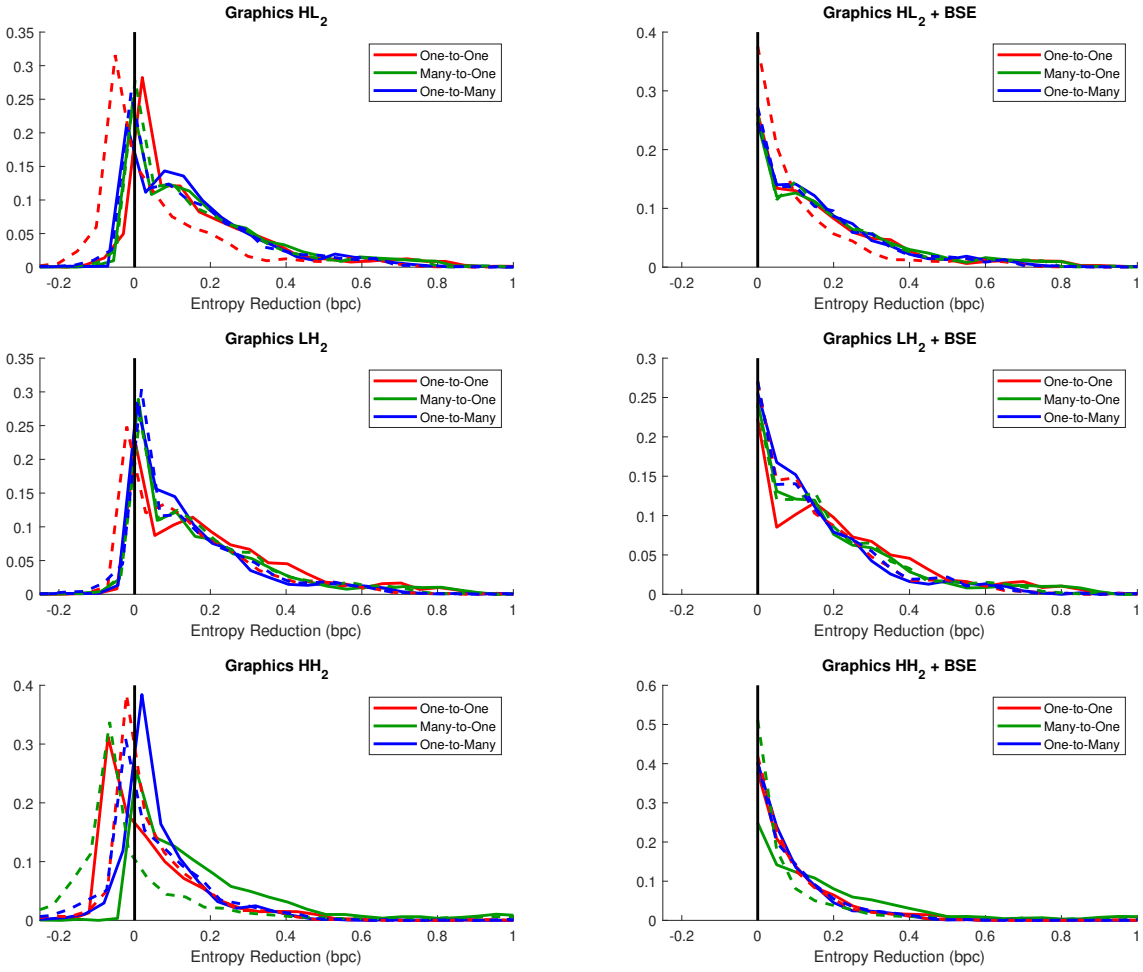


Figure 4.17: Distributions of entropy reduction over the *graphics* dataset for subbnads in the 2nd decomposition level. Solid and dashed lines represent networks trained with $L1$ and $L2$ loss, respectively.

### 4.3.1 *CNN* Prediction Examples

In this section, several visual examples are given to demonstrate the efficacy of the proposed *CNN* prediction method. For simplicity, we only show results for networks which were trained using $L1$ loss on $\mathcal{D}_1$. Within each figure, the entropy of the original coefficients and prediction residuals are provided for comparison.

In Figures 4.18 through 4.23, we look at examples from the *natural* dataset. In general, we see that all networks are capable of effectively recovering the significant structural information within a subband. Noticeable differences in prediction

performance occur when looking at the ability to correctly predict signs, as well as capture subtle textural details contained within small magnitude coefficients.

In Figures 4.18 through 4.20, incorrect sign prediction by the *one-to-one* and *one-to-many* models results in poor prediction results and low entropy reductions, while the *many-to-one* model produces good sign predictions and high entropy reductions. Issues related to sign prediction appear to occur primarily in the $HH_1$ subband. We note that the nature of this incorrect sign prediction is largely uniform over a given subband prediction, where nearly all predicted values may have the opposite sign of the original coefficients. This may be attributed to the non-shift invariance of the *DWT*, where shifting of a signal can result in completely different subband coefficients. This phenomenon is demonstrated in Figure 4.24. In this example, two images are cropped from the same image, the only difference being a 1 pixel shift in the row dimension during cropping. While the images are virtually identical, the resulting subband after decomposition show prominent structural, as well as sign, differences. In the *many-to-one* mode, having access to information within neighboring detail subbands may provide contextual information that allows the network to better compensate for this effect.

In Figures 4.21 through 4.23, the ability of the *many-to-one* model to effectively recover small magnitude coefficients is demonstrated. This is most apparent in Figure 4.23, in which substantial improvements are seen across all subbands.

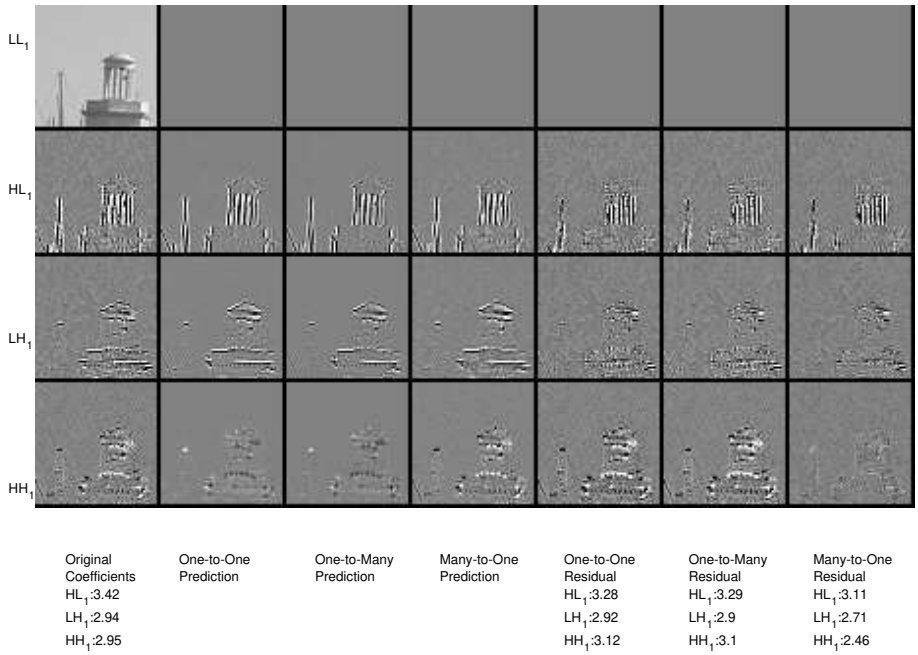


Figure 4.18: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

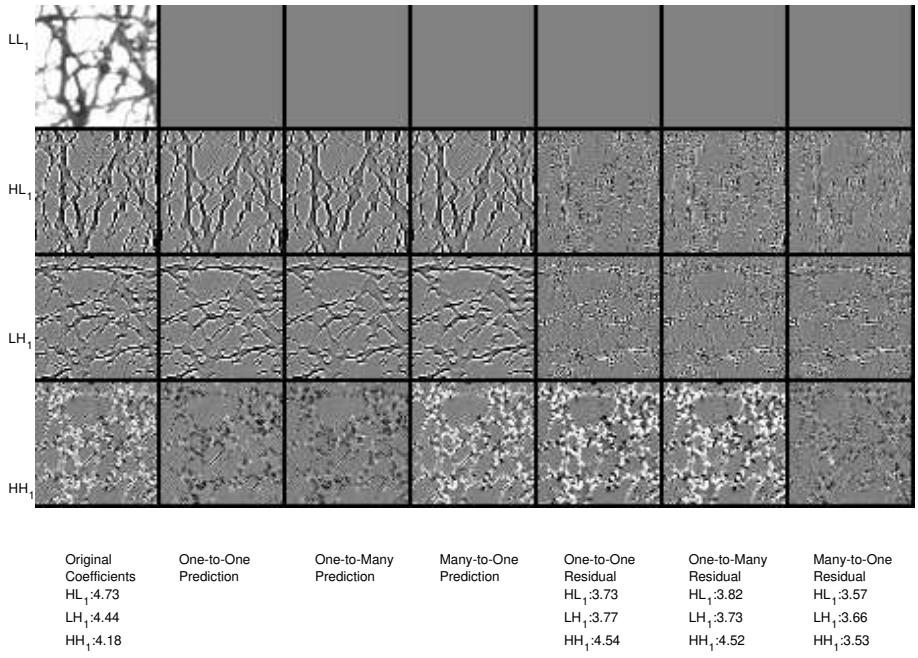


Figure 4.19: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

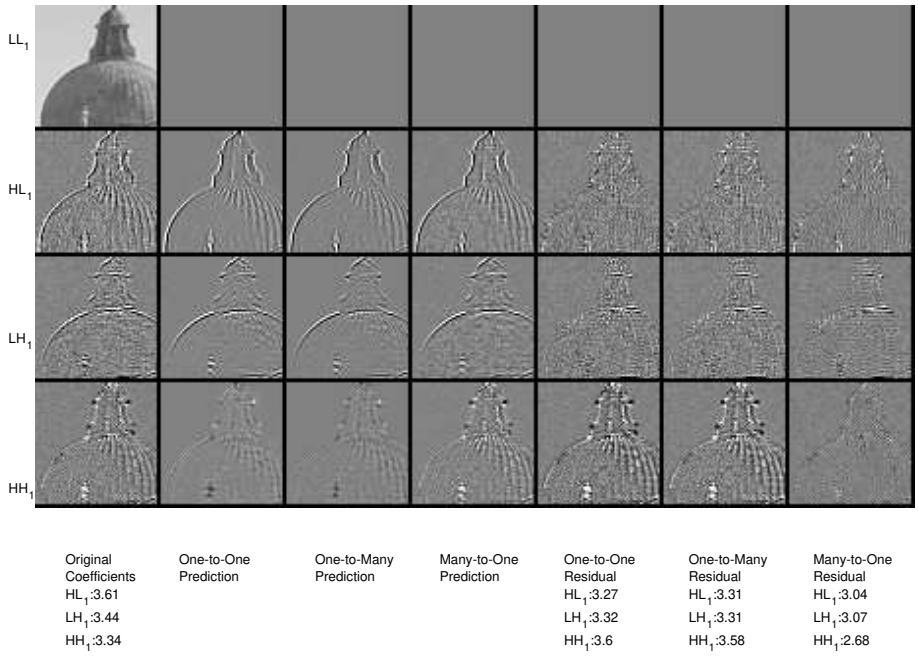


Figure 4.20: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

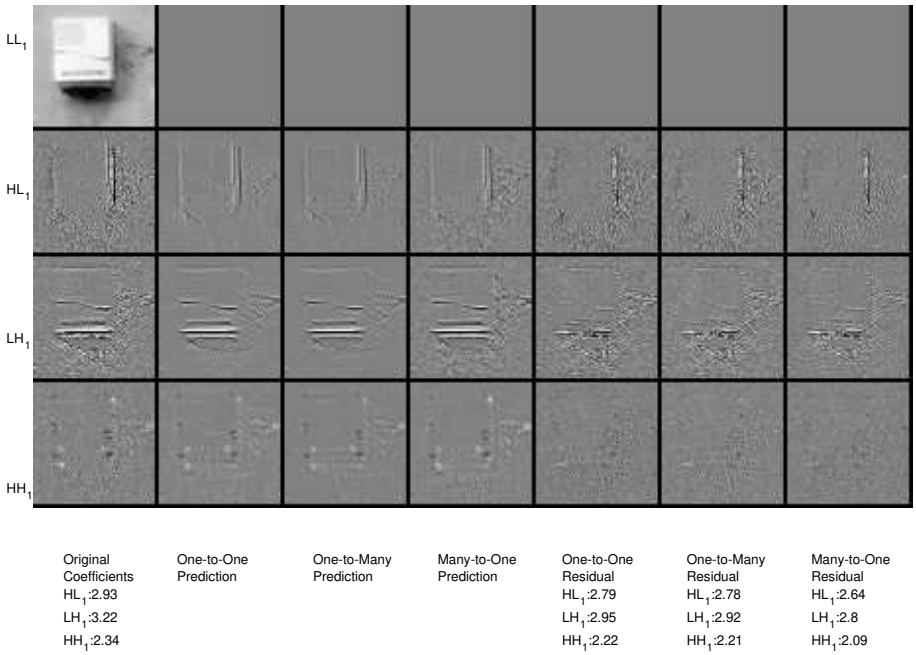


Figure 4.21: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

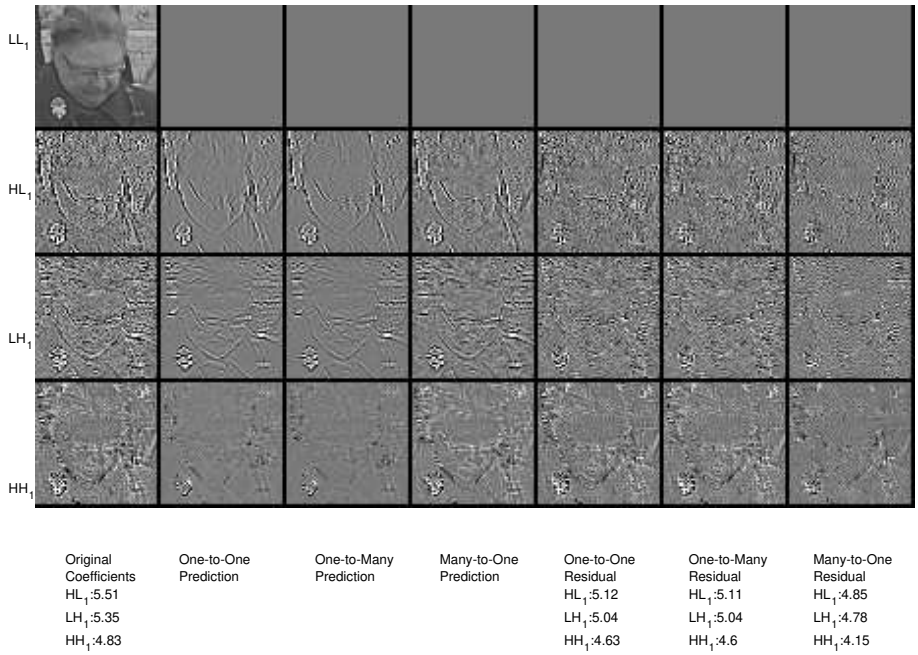


Figure 4.22: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

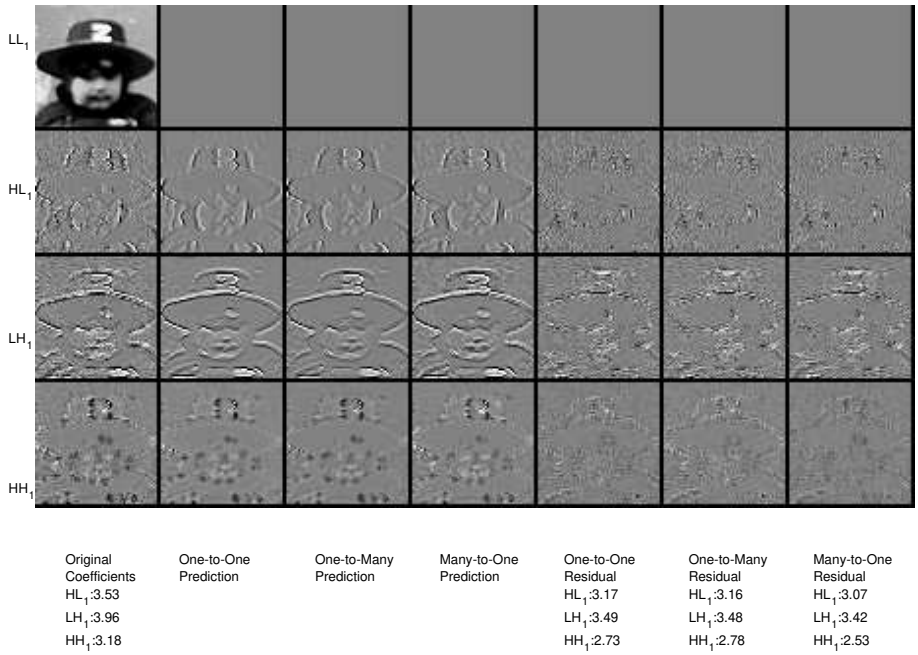


Figure 4.23: Example prediction from the *natural* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

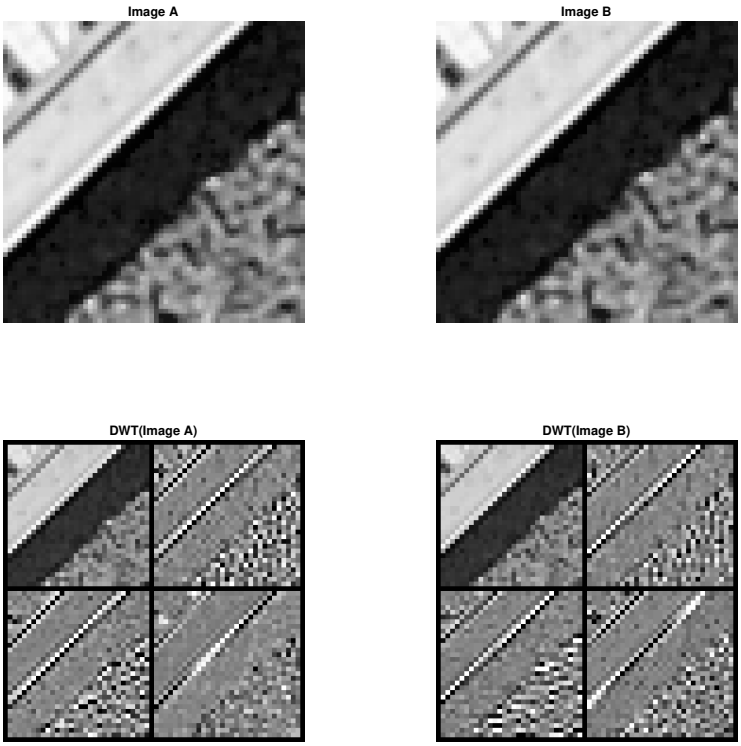


Figure 4.24: Demonstration of the non-shift invariance of the $DWT$ by looking at the $DWT$ of two image crops which only differ by a 1 pixel shift in the row dimension.

Figures 4.25 through 4.28 gives prediction examples from the *pathology* dataset. The results in Table 4.4 implied that no benefit may be seen by using the *many-to-one* prediction model on pathology images, except for on the $HH_1$ subband. We see this in the examples provided below, where in all cases, the *many-to-one* model produced nearly identical predictions to the *one-to-one* and *one-to-many* models on $HL_1$ and $LH_1$.

The examples in Figures 4.25 through 4.27 demonstrates the noise contamination in $HH_1$ mentioned in Section 4.1.3. While the network does not learn the noise, it does learn the banding within the noise; this banding is likely the result of a scanning procedure used to capture the images. By learning this banding, the network is able to achieve modest entropy reductions. In Figure 4.28 we see an

example in which structure does exist within the $HH_1$ subband. All networks do a poor job of predicting this structure, which is likely results from a lack of training examples from which to learn.

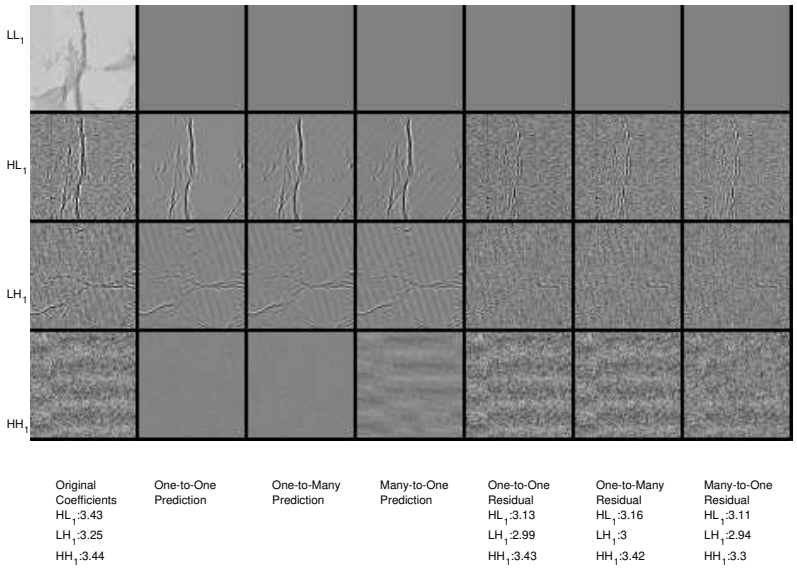


Figure 4.25: Example prediction from the *pathology* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

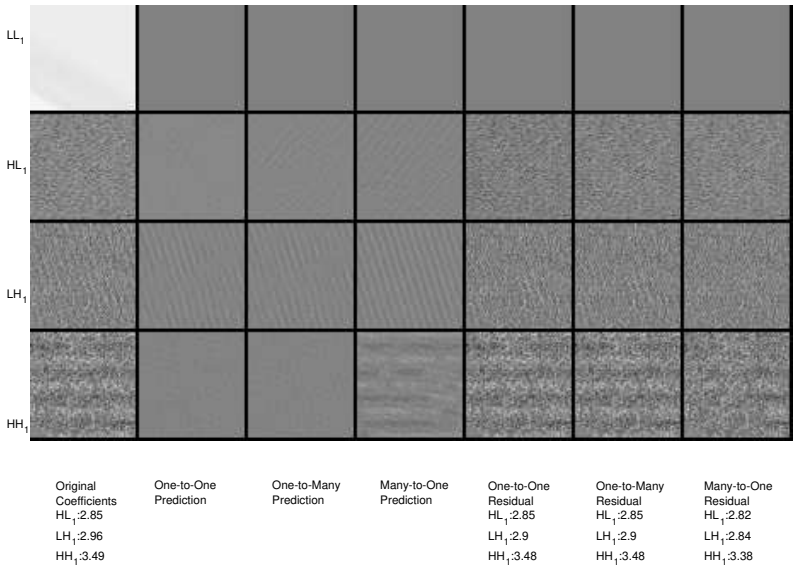


Figure 4.26: Example prediction from the *pathology* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

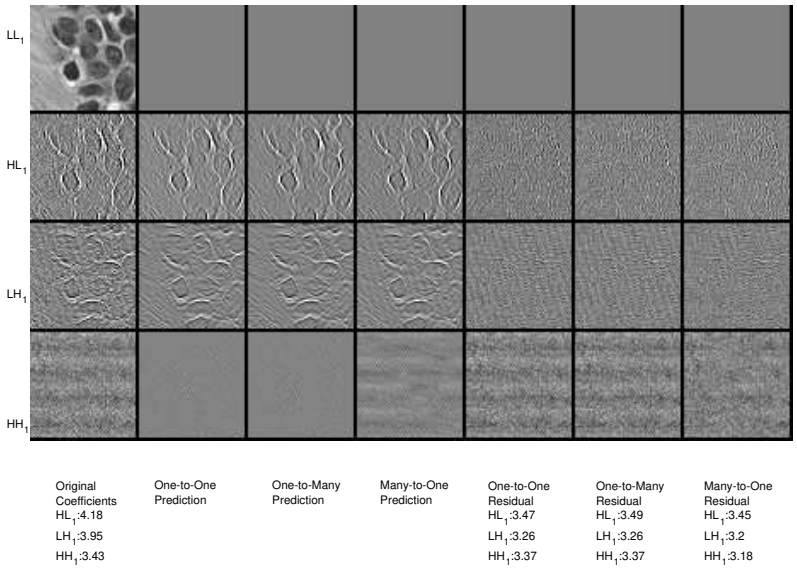


Figure 4.27: Example prediction from the *pathology* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

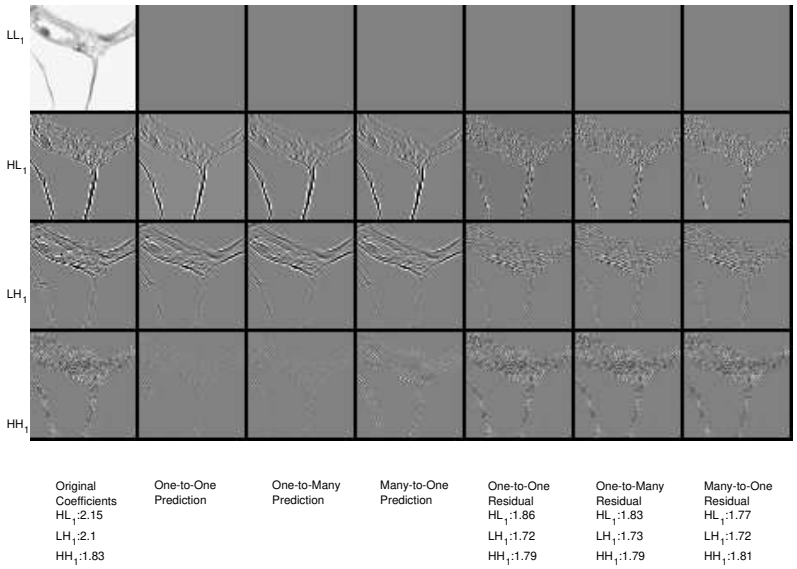


Figure 4.28: Example prediction from the *pathology* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

Figures 4.29 through 4.34 provide several examples from the *graphics* dataset. Between networks, we see that predictions made by the *one-to-one* network for $HL_1$ containing a glowing effect around edges. This is likely the result of network weights which did not converge well, which agrees with our observations in 4.2. In all examples, networks are able to sufficiently capture structural information. The textural details which gave the *many-to-one* network the largest advantage on the *natural* dataset are not present in the *graphics* dataset. This likely contributes to it having similar performance to the *one-to-one* and *one-to-many* models in this case.

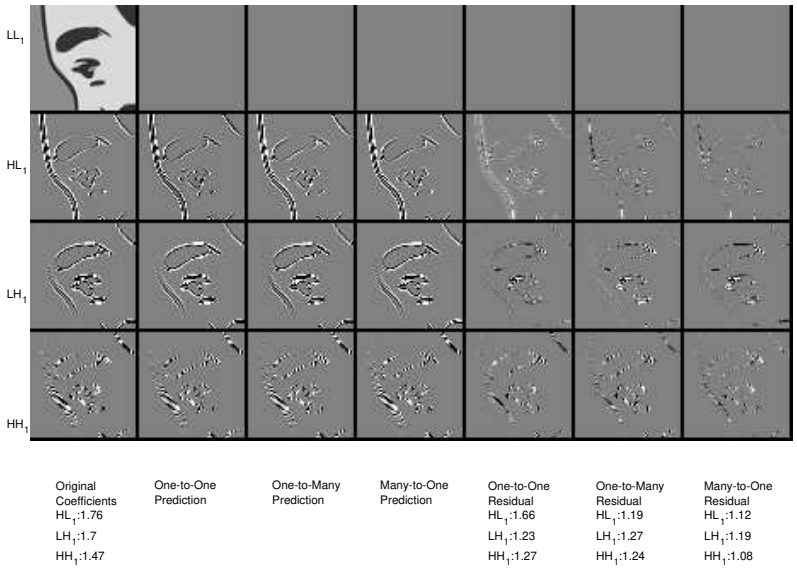


Figure 4.29: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

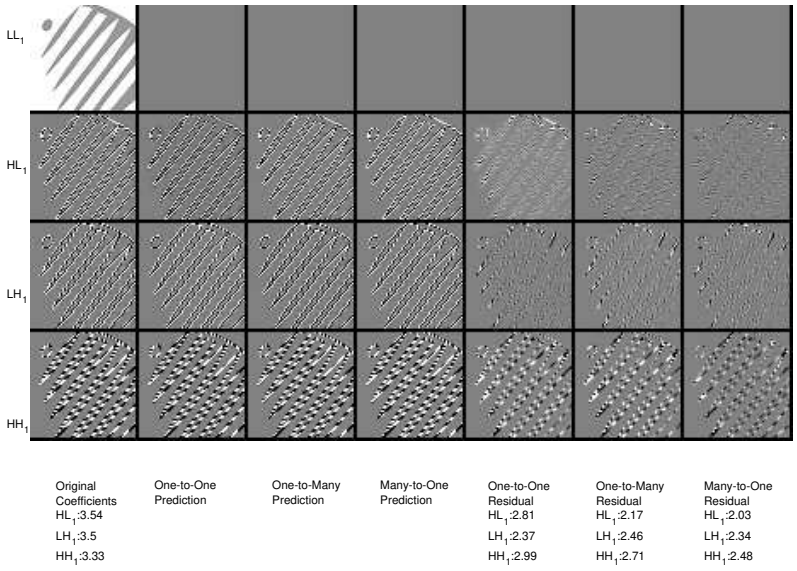


Figure 4.30: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

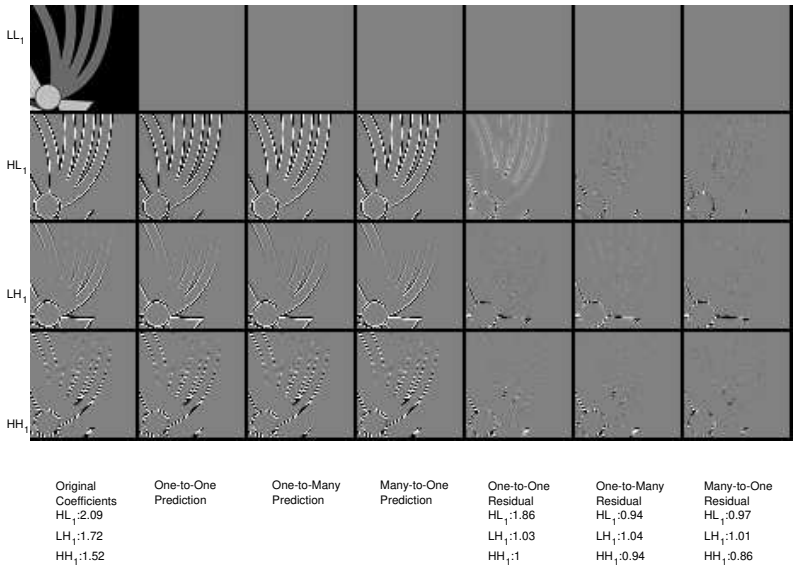


Figure 4.31: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

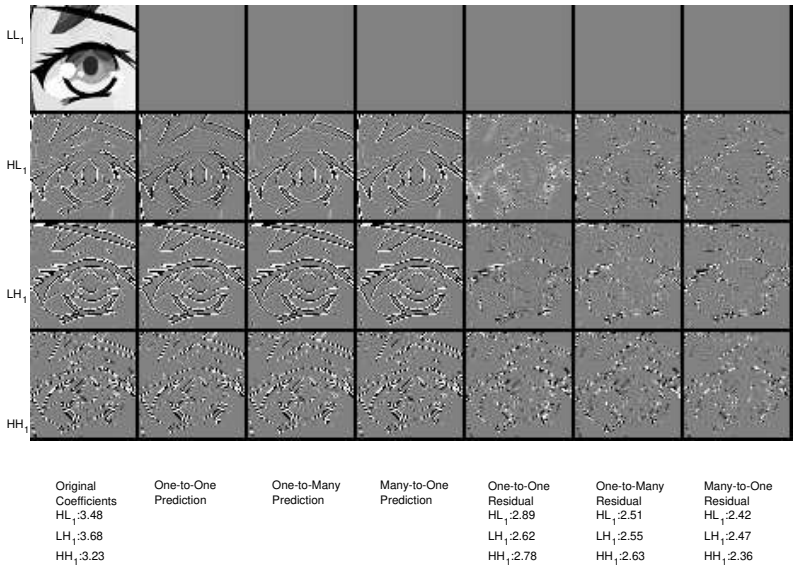


Figure 4.32: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

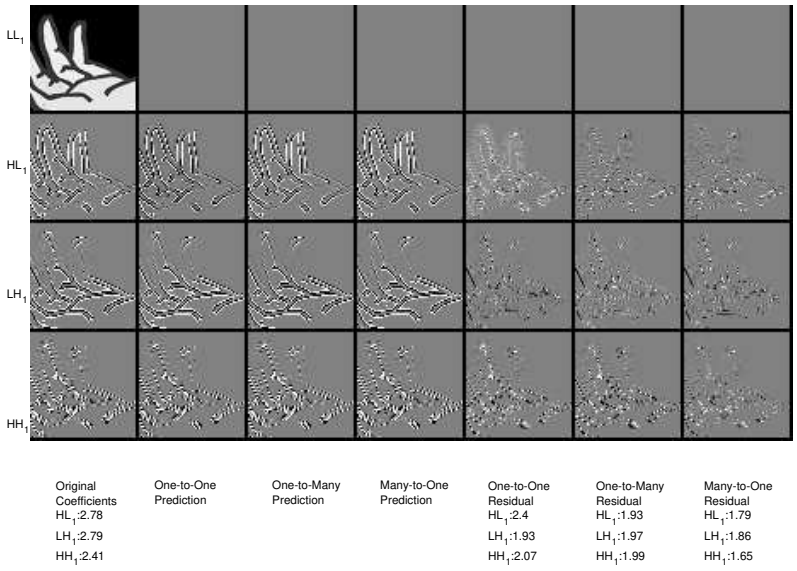


Figure 4.33: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

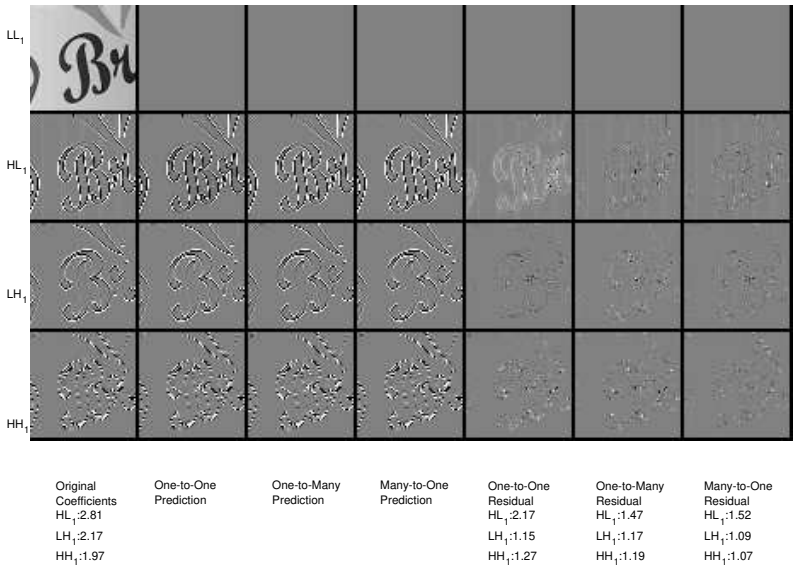


Figure 4.34: Example prediction from the *graphics* dataset of subbands within $\mathcal{D}_1$. The entropy of the original coefficients, and prediction residuals, are provided below each corresponding column.

## 4.4 Baseline Prediction Framework

A baseline prediction framework is now developed, guided by the results obtained throughout this section. For this, we look to minimize the potential bit-rate of the entire image, and so we must now consider bit-rate in bits-per-pixel (bpp). Since the total compressed file consists of compressed data from multiple subbands, we must scale the entropy reduction of each subband by its relative contribution to the full image codestream to obtain bit-rate reduction in bits-per-pixel (*bpp*). The average relative contribution for each detail subband in the first two decomposition levels is given in Table 4.6, where compressed data from test images within the *natural*, *pathology*, and *graphics* datasets are used to generate statistics. With these values, along with those provided in Tables 4.3-4.5, we estimate the potential bit-rate reduction of each proposed framework using Equation 4.3

$$\widehat{\Delta bpp} = r_{HL_1}C_{HL_1} + r_{LH_1}C_{LH_1} + r_{HH_1}C_{HH_1} + r_{HL_2}C_{HL_2} + r_{LH_2}C_{LH_2} + r_{HH_2}C_{HH_2} \quad (4.3)$$

where $r_x$ denotes the relative contribution of subband $x$ to the compressed bitstream, and $C_x$ denotes the bit-ate reduction of subband $x$ in *bpc*.

For each proposed framework, we are limited to which prediction models may be used in conjunction, as well as the order of their application. We may understand the first point by observing that, when the *many-to-one* model is used for both *HL* and *LH* prediction, the framework will not be reversible. This is due to the conflicting dependency on information used to generate each prediction. The second point follows the first in that, if we wish to perform *many-to-one* prediction at the decoder, we must first reconstruct the subbands used in making the prediction.

With these points in mind, we consider the prediction frameworks given in Table 4.7 and detailed in Algorithms 5 - 8. The resulting potential bit-rate reduction for each is given in Table 4.8. Unsurprisingly, the frameworks which employ many-to-one prediction have the high potential bit-rate reduction. In particular, the *MO-OO-MO* framework gives the best results for *natural* and *pathology* data, while the *OO-MO-MO* framework gives the best results for *graphics* images. The difference in potential bit-rate reduction between these two frameworks is negligible, as a result we choose *MO-OO-MO* as a baseline.

Table 4.6: Subband Relative Codestream Contributions

| Subband | Natural | Pathology | Graphics |
|---|---|---|---|
| $HL_1$ | 0.240 | 0.233 | 0.150 |
| $LH_1$ | 0.243 | 0.238 | 0.149 |
| $HH_1$ | 0.204 | 0.257 | 0.130 |
| $HL_2$ | 0.075 | 0.064 | 0.076 |
| $LH_2$ | 0.076 | 0.064 | 0.078 |
| $HH_2$ | 0.075 | 0.070 | 0.071 |

Table 4.7: Prediction Frameworks

| Framework | HL | LH | HH |
|---|---|---|---|
| OO-OO-OO | One-to-One | One-to-One | One-to-One |
| MO-OO-MO | Many-to-One | One-to-One | Many-to-One |
| OO-MO-MO | One-to-One | Many-to-One | Many-to-One |
| OM-OM-OM | One-to-Many | One-to-Many | One-to-Many |

Table 4.8: Estimated Average Bit-rate Reduction ($\widehat{\Delta bpp}$)

| Framework | Natural | Pathology | Graphics |
|---|---|---|---|
| OO-OO-OO | 0.171 | 0.149 | 0.043 |
| MO-OO-MO | 0.275 | 0.195 | 0.052 |
| OO-MO-MO | 0.276 | 0.199 | 0.048 |
| OM-OM-OM | 0.166 | 0.148 | 0.037 |

**Algorithm 5** OO-OO-OO $CNN_{enc}$ and $CNN_{dec}$ procedures

**procedure** $CNN_{enc_n}(LL_n, HL_n, LH_n, HH_n)$
  $\widehat{HL}_n \leftarrow CNN_{LL\rightarrow HL}(LL_n)$
  $HL_n^E \leftarrow SBE(HL_n,\ \widehat{HL}_n)$
  $\widehat{LH}_n \leftarrow CNN_{LL\rightarrow LH}(LL_n)$
  $LH_n^E \leftarrow SBE(LH_n,\ \widehat{LH}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL\rightarrow HH}(LL_n)$
  $HH_n^E \leftarrow SBE(HH_n,\ \widehat{HH}_n)$
**end procedure**

**procedure** $CNN_{dec_n}(LL_n, HL_n^E, LH_n^E, HH_n^E)$
  $\widehat{HL}_n \leftarrow CNN_{LL\rightarrow HL}(LL_n)$
  $HL_n \leftarrow SBD(HL_n^E, \widehat{HL}_n)$
  $\widehat{LH}_n \leftarrow CNN_{LL\rightarrow LH}(LL_n)$
  $LH_n \leftarrow SBD(LH_n^E, \widehat{LH}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL\rightarrow HH}(LL_n)$
  $HH_n \leftarrow SBD(HH_n^E, \widehat{HH}_n)$
**end procedure**

**Algorithm 6** MO-OO-MO $CNN_{enc}$ and $CNN_{dec}$ procedures

**procedure** $CNN_{enc_n}(LL_n,HL_n,LH_n,HH_n)$
  $\widehat{LH}_n \leftarrow CNN_{LL\rightarrow LH}(LL_n)$
  $LH_n^E \leftarrow SBE(LH_n,\ \widehat{LH}_n)$
  $\widehat{HL}_n \leftarrow CNN_{LL,LH\rightarrow HL}(LL_n, LH_n)$
  $HL_n^E \leftarrow SBE(HL_n,\ \widehat{HL}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL,HL,LH\rightarrow HH}(LL_n, HL_n, LH_n)$
  $HH_n^E \leftarrow SBE(HH_n,\ \widehat{HH}_n)$
**end procedure**

**procedure** $CNN_{dec_n}(LL_n,HL_n^E,LH_n^E,HH_n^E)$
  $\widehat{LH}_n \leftarrow CNN_{LL\rightarrow LH}(LL_n)$
  $LH_n \leftarrow SBD(LH_n^E, \widehat{LH}_n)$
  $\widehat{HL}_n \leftarrow CNN_{LL,LH\rightarrow HL}(LL_n, LH_n)$
  $HL_n \leftarrow SBD(HL_n^E, \widehat{HL}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL,HL,LH\rightarrow HH}(LL_n, HL_n, LH_n)$
  $HH_n \leftarrow SBD(HH_n^E, \widehat{HH}_n)$
**end procedure**

**Algorithm 7** OO-MO-MO $CNN_{enc}$ and $CNN_{dec}$ procedures

**procedure** $CNN_{enc_n}(LL_n,HL_n,LH_n,HH_n)$
  $\widehat{HL}_n \leftarrow CNN_{LL\rightarrow HL}(LL_n)$
  $HL_n^E \leftarrow SBE(HL_n,\ \widehat{HL}_n)$
  $\widehat{LH}_n \leftarrow CNN_{LL,HL\rightarrow LH}(LL_n, HL_n)$
  $LH_n^E \leftarrow SBE(LH_n,\ \widehat{LH}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL,HL,LH\rightarrow HH}(LL_n, HL_n, LH_n)$
  $HH_n^E \leftarrow SBE(HH_n,\ \widehat{HH}_n)$
**end procedure**

**procedure** $CNN_{dec_n}(LL_n,HL_n^E,LH_n^E,HH_n^E)$
  $\widehat{HL}_n \leftarrow CNN_{LL\rightarrow HL}(LL_n)$
  $HL_n \leftarrow SBD(HL_n^E, \widehat{HL}_n)$
  $\widehat{LH}_n \leftarrow CNN_{LL,HL\rightarrow LH}(LL_n, HL_n)$
  $LH_n \leftarrow SBD(LH_n^E, \widehat{LH}_n)$
  $\widehat{HH}_n \leftarrow CNN_{LL,HL,LH\rightarrow HH}(LL_n, HL_n, LH_n)$
  $HH_n \leftarrow SBD(HH_n^E, \widehat{HH}_n)$
**end procedure**

**Algorithm 8** OM-OM-OM $CNN_{enc}$ and $CNN_{dec}$ procedures

**procedure** $CNN_{enc_n}(LL_n, HL_n, LH_n, HH_n)$
    $\widehat{HL}_n, \widehat{LH}_n, \widehat{HH}_n \leftarrow CNN_{LL\rightarrow HL,LH,HH}(LL_n)$
    $HL_n^E \leftarrow SBE(HL_n,\ \widehat{HL}_n)$
    $LH_n^E \leftarrow SBE(LH_n,\ \widehat{LH}_n)$
    $HH_n^E \leftarrow SBE(HH_n,\ \widehat{HH}_n)$
**end procedure**

**procedure** $CNN_{dec_n}(LL_n, HL_n^E, LH_n^E, HH_n^E)$
    $\widehat{HL}_n, \widehat{LH}_n, \widehat{HH}_n \leftarrow CNN_{LL\rightarrow HL,LH,HH}(LL_n)$
    $HL_n \leftarrow SBD(HL_n^E, \widehat{HL}_n)$
    $LH_n \leftarrow SBD(LH_n^E, \widehat{LH}_n)$
    $HH_n \leftarrow SBD(HH_n^E, \widehat{HH}_n)$
**end procedure**

## 4.5 Compression Performance

Using the proposed model outlined in Section 3, and the baseline *CNN* prediction framework given in Section 4.4, we implement an end-to-end encoder and decoder. The compression performance of this implementation is compared with current standards, as well as state of the art methods.

All compression experiments are performed as follows: Test images consist of $2048 \times 2048$ 8-bit grayscale images stemming from the *natural*, *pathology*, and *graphics* test image sets discussed in Section 4.1.3. Each image is broken down into subbands using a 5-level integer 5/3 *DWT*. Encoding and decoding is performed in accordance with Algorithm 1 and 3, respectively, with the $CNN_{enc_n}$ and $CNN_{dec_n}$ procedures given by Algorithm 6. Entropy coding is done using the context driven binary arithmetic corder, as implemented in OpenJPEG [31]. While OpenJPEG produces JPEG2000 compliant codestreams, our modified version is not JPEG2000 compliant. Here, OpenJPEG is only used for its arithmetic coder and ability to produce a compressed file.

The results of these experiments are given in Table 4.9. For *natural* imags, the proposed method achieved an average bit-rate which is 7.6%, 5.8%, 0.6%, and 1.7% lower than Lossless JPEG2000, JPEG-LS, CALIC, and FLIF, respectively; while

GLICBAWLS achieves a bit-rate which is 1.2% lower than the proposed method. For *pathology* images, the proposed method sees bit-rate reductions of 5.1% and 3.9% over Lossless JPEG2000 and JPEG-LS, respectively; CALIC, GLICBAWLS, and FLIF achieve bit-rates which are lower than the proposed method by 1.9%, 5.0%, and 1.9%, respectively. For *graphics* images, the proposed method achieves bit-rate reductions of 25.9% over Lossless JPEG2000 and GLICBAWLS, but results in a bit-rate which is 17.5%, 32.5%, and 42.5% higher than JPEG-LS, CALIC, and FLIF, respectively.

While the proposed method achieves near state of the art performance for *natural* images, its performance may be considered average for *pathology* and *graphics* images. This may be attributed to a reduced level of structural information within the *pathology* and *graphics* images, leading to wavelet subbands with less structure. While the *natural* images contain complex structure which leads to rich content within wavelet subbands, the *pathology* and *graphics* images generally contain large smooth or constant intensity regions, leading to a reduced amount of exploitable information.

Table 4.9: Compression Performance (bpp)

| Method | Natural | Pathology | Graphics |
|---|---|---|---|
| Lossless J2K | 3.66 | 3.36 | 0.54 |
| JPEG-LS | 3.59 | 3.32 | 0.33 |
| CALIC | 3.40 | 3.13 | 0.27 |
| GLICBAWLS | 3.34 | 3.03 | 0.54 |
| FLIF | 3.44 | 3.13 | 0.23 |
| **Proposed** | 3.38 | 3.19 | 0.40 |

The encoding and decoding time for each method is provided in Table 4.10; we additionally provide the encoding and decoding throughput, measured in Mega-Pixels per second (MPixels/s). For the proposed method, the *CPU+GPU* configuration uses the GPU for *CNN* prediction and the CPU for all other tasks. The *CPU Only* implementation performs *CNN* prediction using the CPU, along with all other tasks. It should be noted that the encoding and decoding for a given method

will be effected by its implementation. For this reason, the values provided should not be considered as strict measures of complexity.

Table 4.10: Encoding and Decoding Time (ms) and Throughput (MPixels/s)

| Method | Encoding Time | Decoding Time | Encoding Throughput | Decoding Throughput |
|---|---|---|---|---|
| Lossless J2K | 1340 | 641 | 3.13 | 6.54 |
| JPEG-LS | 254 | 236 | 16.50 | 17.75 |
| CALIC | 1290 | 1260 | 3.25 | 3.33 |
| GLICBAWLS | 170827 | 170865 | 0.02 | 0.02 |
| FLIF | 17977 | 1300 | 0.23 | 3.23 |
| **Proposed (CPU+GPU)** | 1831 | 1132 | 2.29 | 3.70 |
| **Proposed (CPU Only)** | 12882 | 12183 | 0.33 | 0.34 |

For Lossless JPEG2000, JPEG-LS, GLICBAWLS, and FLIF, compression experiments were ran on a machine running *Ubuntu 16.04.4 LTS*, with *Intel(R) Xeon(R) E5-2699A* CPUs, and *Nvidia Tesla P100* GPU. For CALIC, only windows binaries were available, these experiments were ran on a machine running Windows 10.0.17134, with *Intel(R) Core(TM) i5-4670K* CPU.

# CHAPTER 5

# CONCLUSION

In this thesis, we proposed a lossless compression framework which incorporates *CNN*s for prediction of wavelet coefficients. Multiple *CNN* prediction frameworks were assess for usage with *natural*, *pathology*, and *graphics* image data. From these, a baseline prediction framework was proposed by analyzing optimal potential bit-rate rate reductions of each framework. Using this framework, an end-to-end implementation was developed, and compared with current standards, as well as state of the art image compression techniques. In these experiments, we found that the proposed model produces bit-rates which compete strongly with state of the art techniques for *natural* images. Weaker, but still competitive, performance was seen in models trained for *pathology* and *graphics* images. Additionally, the proposed model has a computational complexity which is practical, even when implemented using only CPUs.

In future work, the proposed model can be extended to multi-component imagery (e.g. RGB, YCbCr, or hyperspectral). With these, a *CNN* prediction framework may be developed which employs cross-component prediction. The proposed model may also be extended to lossy image compression. For this we must study the effects of quantization on *CNN* prediction, then a rate allocation model may be developed which takes into consideration the information required for prediction at the decoder.

# REFERENCES


[1] X. Wu and N. Memon, "CALIC-a context based adaptive lossless image codec," in *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. On Conference Proceedings., 1996 IEEE International Conference - Volume 04*, ICASSP '96, (Washington, DC, USA), pp. 1890–1893, IEEE Computer Society, 1996.

[2] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," *Trans. Img. Proc.*, vol. 9, pp. 1309–1324, Aug. 2000.

[3] D. Taubman and M. Marcellin, *JPEG2000 Image Compression Fundamentals, Standards and Practice*. Springer Publishing Company, Incorporated, 2002.

[4] B. Meyer and P. Tischer, "GLICBAWLS- grey level image compression by adaptive weighted," in *Least Squares,, DCC 2001, Data Compression Conference 2001*, p. 503, 2001.

[5] J. Sneyers and P. Wuille, "FLIF: free lossless image format based on MANIAC compression," in *2016 IEEE International Conference on Image Processing, ICIP 2016, Phoenix, AZ, USA, September 25-28, 2016*, pp. 66–70, 2016.

[6] A. van den Oord and B. Schrauwen, "The student-t mixture as a natural image patch prior with application to image compression," *Journal of Machine Learning Research*, vol. 15, pp. 2061–2086, 2014.

[7] K. Gregor, F. Besse, D. Jimenez Rezende, I. Danihelka, and D. Wierstra, "Towards conceptual compression," in *Advances in Neural Information Processing Systems 29* (D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, eds.), pp. 3549–3557, Curran Associates, Inc., 2016.

[8] C. C. Cutle, "Differential quantization of communication signals," 1950. US Patent 2605361 A.

[9] N. Ahmed, T. Natarajan, and K. R. Rao, "Discrete cosine transfom," *IEEE Trans. Comput.*, vol. 23, pp. 90–93, Jan. 1974.

[10] G. Strang and T. Nguyen, *Wavelets and filter banks*. Wellesley-Cambridge Press, 1997.

[11] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, pp. 1098–1101, September 1952.

[12] A. Robinson and C. Cherry, "Results of a prototype television bandwidth compression scheme," vol. 55, pp. 356 – 364, 04 1967.

[13] I. H. Witten, R. M. Neal, and J. G. Cleary, "Arithmetic coding for data compression," *Commun. ACM*, vol. 30, pp. 520–540, 06 1987.

[14] C. Shannon, "A mathematical theory of communication," *Bell Systems Technical Journal*, vol. 27, pp. 379–423, 1948.

[15] D. Taylor, D. Newman, and B. Schunck, "Cineform," Dec 2017.

[16] S. Mallat, *A Wavelet Tour of Signal Processing, Third Edition: The Sparse Way*. Academic Press, 3rd ed., 2008.

[17] I. Daubechies and W. Sweldens, "Factoring wavelet transforms into lifting steps," *J. Fourier Anal. Appl.*, vol. 4, no. 3, pp. 245–267, 1998.

[18] I. Daubechies, "Biorthogonal bases of compactly supported wavelets," 1992.

[19] J. Kiefer and J. Wolfowitz, "Stochastic estimation of the maximum of a regression function," *Ann. Math. Statist.*, vol. 23, pp. 462–466, 09 1952.

[20] Y. Chauvin and D. E. Rumelhart, eds., *Backpropagation: Theory, Architectures, and Applications*. Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1995.

[21] A. Calderbank, I. Daubechies, W. Sweldens, and B.-L. Yeo, "Wavelet transforms that map integers to integers," *Applied and Computational Harmonic Analysis*, vol. 5, pp. 332–369, 1998.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[23] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2014.

[25] J. Kim, J. K. Lee, and K. M. Lee, "Accurate image super-resolution using very deep convolutional networks," June 2016.

[26] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, pp. 295–307, Feb. 2016.

[27] P. M. Radiuk, "Impact of training set batch size on the performance of convolutional neural networks for diverse datasets," *Information Technology and Management Science*, vol. 20, no. 1, pp. 20–24, 2017.

[28] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, "Raise a raw images dataset for digital image forensics," 2015.

[29] "Public domain vectors," June 2018.

[30] Contributors, "Inkscape." Online, March 2018.

[31] D. Janssens, K. Hgihara, J. Fimes, Giuseppe, M. Savinaud, M. Malaterre, Y. Verschueren, H. Drolon, F.-O. Devaux, and A. Descampe, "OpenJPEG," Oct 2018.