Enhancing Performance of Conventional Image Codecs Using CNN Based Image Sub-sampling and Super Resolution

by

Susmita Saha

A thesis submitted to The Faculty of Graduate Studies

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering

The University of Manitoba

Winnipeg, Canada

October 2020

To my parents and my dear husband

Abstract

The goal of image compression is to reduce the number of bits required to represent an image with a minimum loss of visual quality. However, conventional image compression algorithms, such as JPEG, produce unpleasant artifacts in decoded images at high compression ratios. This thesis investigates a CNN-based approach to improve the performance of the widely used JPEG codec.

In recent times, there has been an increasing interest in using convolutional neural network (CNN) for various image processing tasks, owing to their ability to learn very compact features from images. However, the use of CNNs to improve the performance of existing image codecs is very limited in literature. Motivated by this, we investigate a CNN based image compression framework which improves the performance of the JPEG algorithm by optimally sub-sampling the input image with a CNN referred to as *compact convolutional neural network* (ComCNN) prior to JPEG encoding and by performing super resolution and enhancement of the decoded image with a CNN referred to as *enhancement based reconstruction convolutional neural network* (EBR-CNN). Both CNNs are optimally trained to minimize the end-to-end image distortion for a given value of the JPEG quality factor. Experimental results are presented which compare the performance of the proposed compression framework with several alternative learning and non-learning based image sub-sampling and super resolution methods. These results show that the proposed method provides noticeable

improvements in decoded image quality compared to the other alternatives.

Acknowledgment

First and foremost, all praises go to God for the strengths and His blessings in completing this thesis. Special appreciation goes to my supervisor, Dr. Pradeepa Yahampath for his continuous support and guidance. His continuous encouragement, suggestions, and experience helped me to complete this thesis work successfully. I am deeply indebted to my supervisor.

I would like to express my gratitude towards my University, University of Manitoba, and my Electrical and Computer Engineering Department for the opportunity and support to pursue my MSc program. I would also like to extend my gratitude to the Faculty of Graduate Studies for granting me International Graduate Student Entrance Scholarship.

With great warmth, I want to thank my beloved parents Dulal Chandra Saha and Radha Rani Saha for always believing in me and motivating me to pursue higher studies. I would also like to thank my in law parents Madhab Chandra Nath and Suniti Debnath for all their love and encouragement. I am also thankful for the guidance from my sister Tultul Saha to pursue my graduate studies. And last but not the least, I would like to heartfully thank Sudipto Debnath for being always a caring husband and an exceptional colleague to me. I cannot count how many times my problem is solved just by discussing it with him. It was comforting to know he was there to help me out. I will be always grateful to him. Thank you all!

Table of Contents

List of Figures

List of Tables

List of Acronyms

AdaGrad	adaptive gradient algorithm
AR-CNN	artifact removing convolutional neural network
bpp	bits per pixel
CAS-CNN	compression artifact suppression convolutional neural network
CNN	convolutional neural network
ComCNN	compact representation convolutional neural network
DCT	discrete cosine transform
DCN	densely connected network
DnCNN	denoising convolutional neural network
DST	discrete sine transform
EBR-CNN	enhancement based reconstruction convolutional neural network
FNN	feed forward neural network
FSRCNN	fast super resolution convolutional neural network
HR	high resolution
JPEG	joint photographic expert group
kB	kilobyte
MSE	mean square error
MLP	multi layer perceptron
PSNR	peak signal-to-noise ratio
QF	quality factor
RMSProp	root mean square propagation
ReLu	rectified linear unit
SGD	stochastic gradient descent
SSIM	structural similarity index modulation
SRCNN	super resolution convolutional neural network
VDSR	very deep super resolution network

Chapter 1

Introduction

A digital image is a two-dimensional array of pixels where each pixel represents a color intensity. These intensity values are represented by bits, and the number of bits required to represent an image determines the amount of computer memory required to store it. The first digital image was captured in 1957, and it was a black and white image that contained 176×176 pixels. By a black and white image, it means that each pixel has two states of intensity. Therefore, to represent a pixel digitally, we will require only one bit. As that image contains 30976 pixels in total, it will take 30976 bits or 3.78-kilobytes (kB) (1 byte = 8 bits) to store it in memory. However, back then, due to the unavailability of solid-state devices, the computers offered very little memory space. Also, the computer memory was too expensive. Therefore, the idea of image compression started receiving attention of the researchers. By compression, it means to represent any digital content with the smallest possible number of bits. As the technology evolved, the digital images were no more confined to black and white and color images also started to become more common place. Typically, a color image contains three separate color components. If each color component requires 8 bits, each pixel will require 24 bits. Clearly, a color image requires more bits than a

black and white image. The image size can be reduced by using data compression. However, with compression, the visual quality of an image degrades, and, therefore, how to achieve the best trade-off between the image quality and the file-size has become an important research problem. As a result, throughout the last few decades, we have observed the development of many image compression algorithms. Although we have achieved many technological breakthroughs in solid-state devices, allowing us to produce storage devices with high memory capacities at a lower price compared to the 1980s, our capability to capture very high resolution images has out paced the developments in storage technology. Moreover, we are using digital platforms in every aspect of our life, and these services often require the transmission of digital content, such as images and video wirelessly. However, we always have a very limited allotment of bandwidth to support these services. As a result, image compression remains an important research filed. Recently, the application of convolutional neural networks (CNN) to image compression has been received considerable research attention in last few years.

1.1 Concept of Image Compression

The main aim of image compression is to reduce the redundancy that presents in an image, and thereby, to encode an image using the smallest possible number of bits in order to store or transmit the image at a given reconstruction quality. Typically, there are three types of redundancies present in any image namely inter-pixel redundancy, psycho visual redundancy, and coding redundancy [3].

It has been observed that very often neighboring pixels in an image share nearly the same pixel magnitude. This type of phenomenon is called inter-pixel redundancy. Psycho visual redundancy occurs in an image due to the fact that the human vision

sensitivity is selective in nature. It is less sensitive to changes in chrominance (color) and more sensitive to changes in luminance (brightness). Coding redundancy arises as the frequency of occurrence of all quantization levels are not same, some are more frequent and some are less. Therefore, it is not required to represent each quantization level with the same number of bits. Rather we can assign more bits to the less frequent levels and less bits to more frequent levels and thereby, we can achieve a lower average number of bits/level.

Generally, image compression can be divided into two types: *lossless compression* and *lossy compression*. Lossless compression is a reversible process whereas lossy compression is irreversible. Once a file is compressed using a lossy compression method, it cannot be brought back to the original form exactly. Lossy compression methods are often used in consumer devices for their much higher compression rate [4]. Typically, lossless compression is required when information losses are intolerable, e.g., medical images used for diagnostics. Huffman coding [5], Golomb coding [6] and Arithmetic coding [7] are some of the common lossless methods of image compression. These methods adopt what is known as entropy coding to reduce the statistical redundancy within the image [8]. On the other hand, the state of the art image codecs, such as joint photographic experts group (JPEG) [9] and JPEG2000 [10] use linear transforms such as discrete cosine transform (DCT) [11], discrete sine transform (DST) [12], and wavelet transform together with scalar or vector quantization for very efficient lossy compression of images.

1.2 Motivation and Contribution of This Thesis

In recent times, there has been an increasing interest in using a convolutional neural network (CNN) for image compression with the goal of developing better image

compression algorithms than existing standards such as JPEG and JPEG2000. This is because the CNNs may have the ability to learn very compact hidden representations directly from training images. It has been observed in recent research that some of the CNN based techniques have the potential to outperform the current image compression standards. However, CNN based image compression research is still in infancy and the methods developed so far cannot be used to implement a complete codec with multiple functionalities. For example, it is not possible to vary the bit rate of a CNN-based image codec as required in most practical applications of image compression. Therefore, an interesting avenue of research is to enhance the performance of a state-of-the-art image codec such as JPEG by incorporating CNN based pre- and post-processing techniques. This is the goal of this thesis.

Recent research has shown that deep learning and CNNs produce better results than traditional methods in many image processing tasks. The main contribution of this thesis an investigation of an approach to improve the performance of the JPEG codec using CNNs for image sub-sampling, super resolution, and denoising. The proposed reconstruction framework, on one hand, is capable of reducing the noise produced by the image codec and on the other hand, is capable of enhancing the reconstructed image for better visual quality compared to what an lossy image codec can do alone, particularly under high compression.

1.3 Related Work

When dealing with image compression, some of the main challenges are image reconstruction, removal of noise, and improvement of visual quality. Therefore, the related work using deep learning can be divided into two categories: (1) image super-resolution, (2) image deblocking and artifact reduction.

1.3.1 Image Super-Resolution using Deep Learning

In this section, we will discuss three most popular CNN based image super-resolution models, namely super resolution convolutional neural network (SRCNN) [13], fast super resolution convolutional neural network (FSRCNN) [14], and very deep super resolution convolutional networks (VDSR) [15].

Dong et al. propose a CNN based image super resolution method SRCNN [13], which can directly learn an end-to-end mapping between a low resolution image and a high resolution image. The network architecture of this model is very simple, as it consists of only three layers. Each layer is a convolution layer followed by ReLu activation function. The hyperparameters of each conv layer are denoted by (f_i, n_i, c_i) where the variables f_i,n_i,c_i represent the filter size, the number of filters and the number of channels, respectively. The main tasks of these layers are patch extraction, non-linear mapping, and reconstruction. The input of the model is a bicubic interpolated version of a low-resolution image, and the sizes of the high-resolution output image and bicubic interpolated image are the same. The network architecture of SRCNN is given in the Figure 1.1.

Later Dong et al. again proposed a method called FSRCNN [14], which is an improved version of SRCNN. The main difference between SRCNN and FSRCNN is that in FSRCNN, no pre-processing of the image is needed to get the same size as the target high-resolution image because up-sampling is done by a transposed convolution

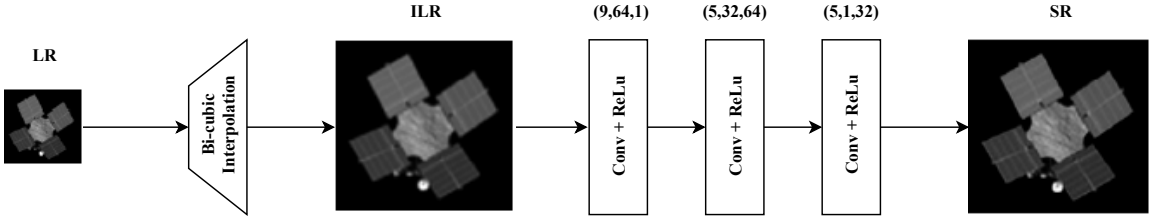


Figure 1.1: Network architecture of SRCNN.

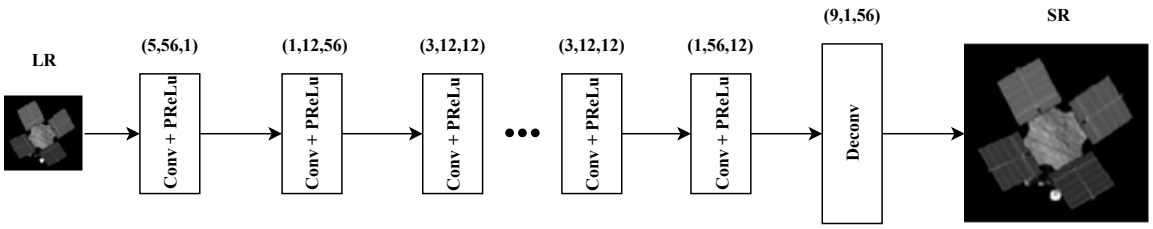


Figure 1.2: Network architecture of FSRCNN.

layer which makes the model faster than SRCNN. The non-mapping layer of SRCNN is replaced by shrinking, mapping, and expanding steps. The filter size is smaller, and the network size is deeper in FSRCNN than SRCNN. The network architecture of FSRCNN is given in the Figure 1.2.

The very deep super resolution network (VDSR) is proposed in [15]. The network architecture of this model is very deep, as it consists of 20 layers. The residual learning technique is used here for better performance and acceleration of the training speed. This technique allows the model to learn the difference between the high resolution and the interpolated images instead of an end-to-end mapping, as has been done in SRCNN [13]. The input of the model is just like SRCNN, which is a bicubic interpolated version of a low-resolution image, and the sizes of the feature maps are kept the same by doing zero padding. The network architecture of VDSR is given in Figure 1.3.

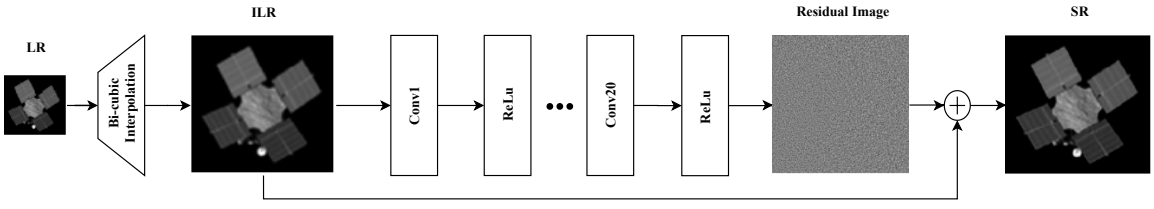


Figure 1.3: Network architecture of VDSR.

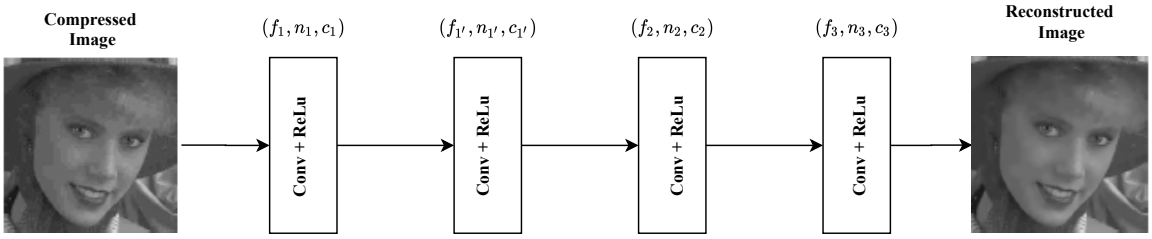


Figure 1.4: Network architecture of ARCNN.

1.3.2 Image Deblocking and Artifact Reduction using Deep Learning

Different types of strategies have been proposed in the literature to remove the blocking artifact of the JPEG compressed image. Among them traditional filter-based methods [16], [17] for image denoising and the sparse coding-based methods [18], [19] for restoring the compressed images are the most popular. Though these methods generate sharper images, most of the time their results suffer from additional artifacts and time-consuming optimizations [20]. In recent times, deep neural networks, especially CNN-based methods have been gaining popularity in de-noising and artifact reduction tasks. Artifact removing convolutional neural network (AR-CNN) [21], denoising convolutional neural network (DnCNN) [22], compression artifact suppression convolutional neural network (CAS-CNN) [1] are the most popular CNN based methods. The authors of SRCNN [13] first introduced CNN based artifact removing network (AR-CNN) [21]. AR-CNN is an extended version of the SRCNN [13] obtained by adding a feature enhancement layer after the feature extraction layer in

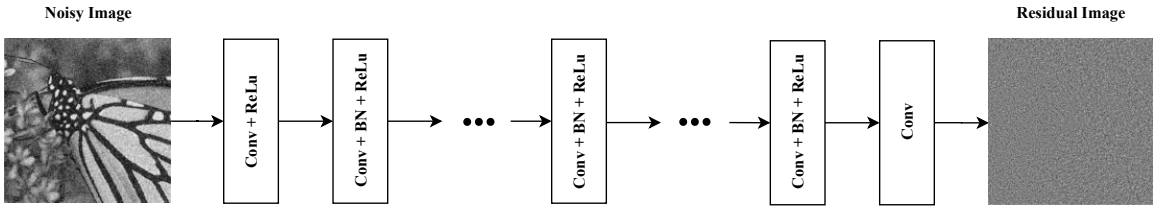


Figure 1.5: Network architecture of DnCNN.

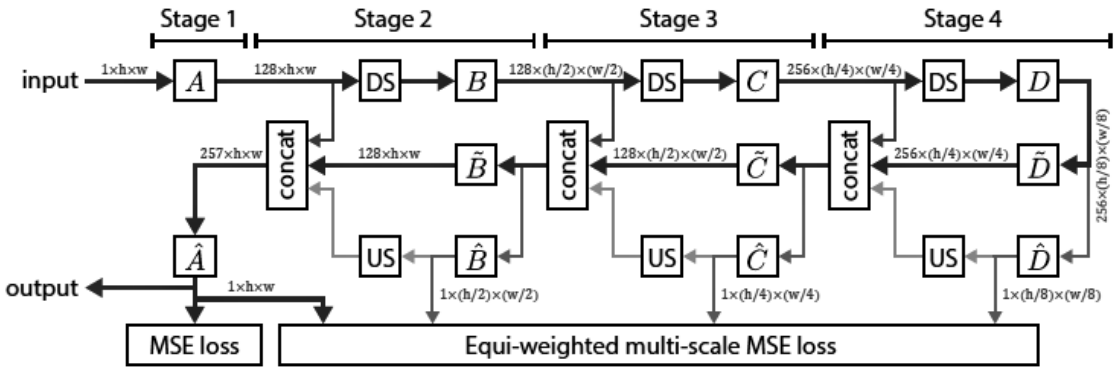


Figure 1.6: Network architecture of CAS-CNN [1].

SRCNN. Therefore, the four layers of the AR-CNN are feature extraction, feature enhancement, mapping, and reconstruction layer. To avoid training difficulties, the authors train the network in two stages. At first, a shallow network is trained, and then it is used to initialize the final four-layer CNN. The network architecture of ARCNN is given in the Figure 1.4.

Zhang et al. propose the deep denoising convolutional neural network (DnCNN) architecture [22] for several general image denoising tasks such as Gaussian noise, single image super-resolution, and JPEG image deblocking. The depth of the network is set to 20 layers. The DnCNN does not predict the denoised image directly. It is trained to predict the residual error, and a denoised image can be obtained by subtracting the output of the network from the input. Residual learning [23] and batch normalization [24] technique are adopted in this network for the improvement of denoising and fast training. The network architecture of DnCNN is given in the Figure 1.5.

CAS-CNN [1] is a 12-layer deep convolutional network for image compression artifact suppression. A residual architecture, an edge emphasized loss function, symmetric weight initialization, and skip connections are introduced to train deep networks for regression tasks without any difficulty. The authors propose a neural network with

hierarchical skip connections and a multi-scale loss function for compression artifact suppression. The network architecture of CAS-CNN is given in the Figure 1.6.

1.4 Outline of the Thesis

Chapter 2 represents some preliminaries about the thesis to follow. It describes a background to deep learning including backpropagation, optimization methods, and activation function.

Chapter 3 gives the description of problem formulation and provides the details of network architecture.

Chapter 4 presents the training procedure and the parameters settings used in this thesis. It also includes the experimental results with comparative and individual analyses of the results.

Chapter 5 concludes the thesis with a summary of findings and direction for future work.

Chapter 2

Overview of Convolutional Neural Networks

The objective of this thesis is to investigate an approach to improve the performance of the JPEG algorithm by using CNNs for pre- and post-processing. An overview of the basic JPEG algorithm is provided in Appendix A. In this chapter an overview of CNNs is provided.

2.1 Convolutional Neural Networks

CNNs are one of the most popular approaches for deep learning with image data. CNNs are originally proposed by LeCun in 1989 for handwritten ZIP code recognition [25]. A CNN consists of an input layer, several hidden layers, and an output layer. Examples of hidden layers are convolutional layers, pooling layers, and fully connected layers. Based on the specific application, CNN architectures vary in the number and type of layers. For example, for categorical purposes, the network should include a classification layer.

The neurons in a CNN receives input from only small portion of an image at a time. Pixels from such a small block is processed by a convolutional filter to extract fetures. Features are then captured by the respective feature maps of the network and multiple different feature maps are used to make the network robust to varying levels of contrast, brightness, noise, etc. in images.

2.2 Convolutional Layers

A convolutional layer performs the convolutional operation between the input and a specific filter. A convolutional operation is shown in Figure 2.1. The output of a convolutional operation is called a feature map. Normally, the input of the convolutional operation is an image if the layer is the first layer of the network. If the convolutional layer is not the first layer of the network, then the input is a feature map produced by the previous layer. The filters are smaller than the input, and for this reason, it is possible to perform the convolution operation multiple times at different points on the input. The filters are a set of weights and also known as the kernels. A layer with input $\boldsymbol{X}$ produces a feature map $\boldsymbol{Z}$ by performing a convolution of $\boldsymbol{X}$ with a filter $\boldsymbol{F}$ [26].

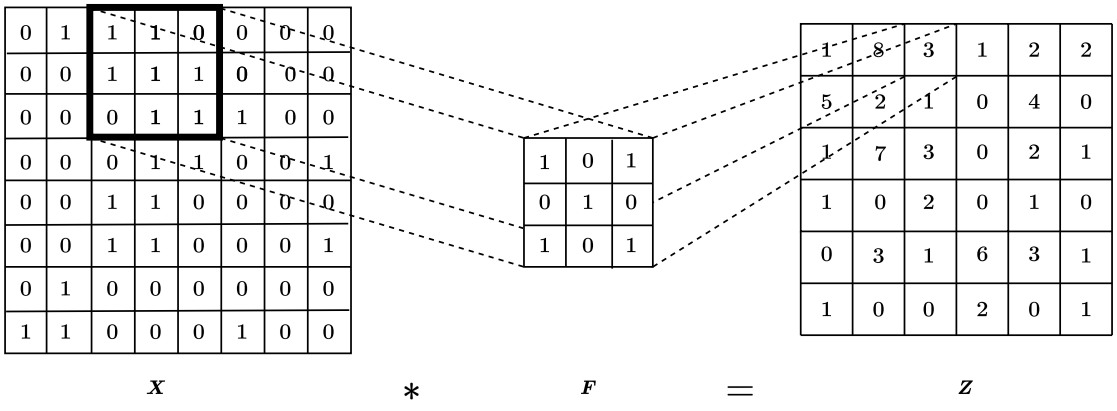


Figure 2.1: A convolutional operation between input image $\boldsymbol{X}$ and the the filter $\boldsymbol{F}$.

The size of the feature map depends on the size of the filters and padding preferences. If the requirement is to preserve the input volume in the output volume, then zero padding is required. Zero padding means adding the symmetrical amount of zeros around the edges of input. The amount of padding to be used along the filter dimension f is given by $p = \frac{f-1}{2}$. However, when the requirement is reducing the size of the feature map, then a stride parameter is needed. Any data volume size can be achieved by adjusting the stride parameter. The stride parameter controls how many pixels the filter should skip when moving the filter horizontally or vertically. The dimension of the feature map is given by

$$N_{out} = \frac{N_{in} + 2p - f}{s} + 1,$$

where N_{in} is input size, f is kernel/filter size, and s is stride.

2.3 Method of Learning the Parameters in CNN

The method of learning the weight parameters in CNN includes two steps: (1) *forward propagation* to determine the outputs and (2) *back propagation* to update the weights. To understand these steps, let us consider a convolutional layer with a filter $\boldsymbol{F}$, an input $\boldsymbol{X}$, and an output $\boldsymbol{O}$ as shown in Figure 2.2. During *forward propagation*, we move across CNN, moving through its layers to compute the $\boldsymbol{O}$ of each layer which is a convolution operation between $\boldsymbol{X}$ and filter $\boldsymbol{F}$ (see previous section) and at the end we obtain loss $\boldsymbol{L}$ using the loss function. In latter step, we feed the loss $\boldsymbol{L}$ backwards to get the loss gradient $\partial \boldsymbol{L}/\partial \boldsymbol{O}$ at each layer from the previous layer. Then the loss is further propagated to the inputs of each layer as shown in Figure 2.2. This step is known as *back propagation*. Now to update the weights of filter $\boldsymbol{F}$, we need to

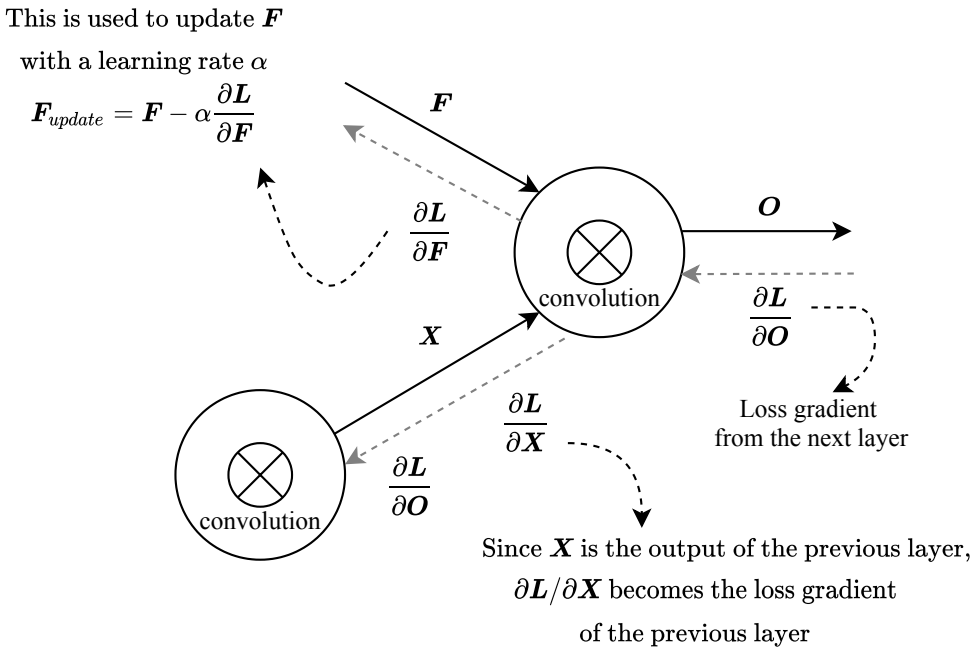


Figure 2.2: A schematic view of learning method.

compute $\partial \boldsymbol{L}/\partial \boldsymbol{F}$ and plug it in an weight updating rule based on an optimization algorithm (see Figure 2.2), and $\partial \boldsymbol{L}/\partial \boldsymbol{X}$ becomes the loss gradient of the previous layer. We will highlight more about the optimization algorithm in next section.

The value of these gradients are computed using the *chain rule*. It can be shown that the local gradient $\partial \boldsymbol{L}/\partial \boldsymbol{F}$ is nothing but a convolution operation between input $\boldsymbol{X}$ and the loss gradient $\partial \boldsymbol{L}/\partial \boldsymbol{O}$ while the local gradient $\partial \boldsymbol{L}/\partial \boldsymbol{X}$ is a *full* convolution operation between a 180-degree rotated filter $\boldsymbol{F}$ and loss gradient $\partial \boldsymbol{L}/\partial \boldsymbol{O}$ [27].

2.4 Optimization Algorithms

Different types of optimization algorithms are available in the literature, and each algorithm has a different approach to calculate, update, and find the optimal values

of model parameters. Some of the most important examples are briefly described below. For more details see [26].

2.4.1 Stochastic Gradient Descent

Stochastic gradient descent (SGD) is one of the most popular optimization algorithms in deep learning. Instead of using the whole dataset per gradient calculation, SGD uses a small portion of the dataset that is called a batch. Using batches during the training of a CNN, computational complexity per weight update can be reduced. The main aim of SGD is to reach the minima using approximate gradients for each batch of training data, where each batch is randomly sampled from a dataset. The pseudo-code of SGD optimization is given in Algorithm 1.

In each iteration, the gradients of the objective function are calculated with respect to the weight. Then the difference is taken to update the weights. Before performing weight update, the gradients are scaled by learning rate α to balance out the contribution and avoid over-correction.

Algorithm 1 Stochastic Gradient Descent Algorithm

1: θ, Initial parameter
2: ϵ, Learning rate
3: D, Dataset
4: m, Batch size
5: **while** stop criterion not met **do**
6: $\{x_i, y_i\} \leftarrow$ sample batch pairs (D)
7: $d \leftarrow \frac{1}{m} \bigtriangledown_\theta \sum_i L(f(x_i; \theta), y_i)$ ▷ compute gradient
8: $\theta \leftarrow \theta - \epsilon d$ ▷ update weights

2.4.2 Adam

Currently, Adam is one of the most popular optimization algorithms because of its robustness to hyper-parameters, fast convergence speed, little memory requirements, and straightforward implementation. The authors of [28] state that the name Adam is derived from *adaptive moment estimation*. The main differences between SGD and Adam lies in the incorporation of adaptive learning rates and momentum. SGD does not consider the previous step while updating the weights; rather, it always considers constant steps towards the minimum. However, the addition of momentum with SGD means giving the gradient a better ability to update the weight by taking into consideration the previous step. Momentum helps to accelerate SGD in the relevant direction and dampening the oscillations [29]. When a method computes individual learning rates for different parameters, it is called an adaptive learning rate method. As Adam is an adaptive learning rate method, it computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients [28]. The authors of [28] describe Adam as combination of adaptive gradient algorithm (AdaGrad) [30] and root mean square propagation (RMSProp) [31]. These two methods are the adaptive learning rate methods and the extensions of SGD. By maintaining an adaptive learning rate, AdaGrad can operate effectively on sparse gradient problems, and RMSProp can work well on online and non-stationary problems [28]. In the RMSProp algorithm, the learning rates are adapted based on the average of recent magnitudes of the gradients for the weight. During the adaption of learning rates, Adam not only considers the average first moment (the mean) like RMSProp but also considers the average of the second moments of the gradients (the uncentered variance) [28].

The pseudo-code of Adam optimization is given in Algorithm 2.

Algorithm 2 Adam

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

2.5 Activation Function

The selection of a proper activation function is essential to the design a CNN. Different types of activation functions are presented in the literature. Among them, the most widely used activation function is called ReLu, which is also used in this research work. Mathematically, it is defined as

$$f(x) = max(0, x).$$

The ReLu function is shown in Figure 2.3. This type of function is differentiable and also easy to compute. It does not suffer from the vanishing gradient problem that arises in training of neural networks with a large number of layers [32].

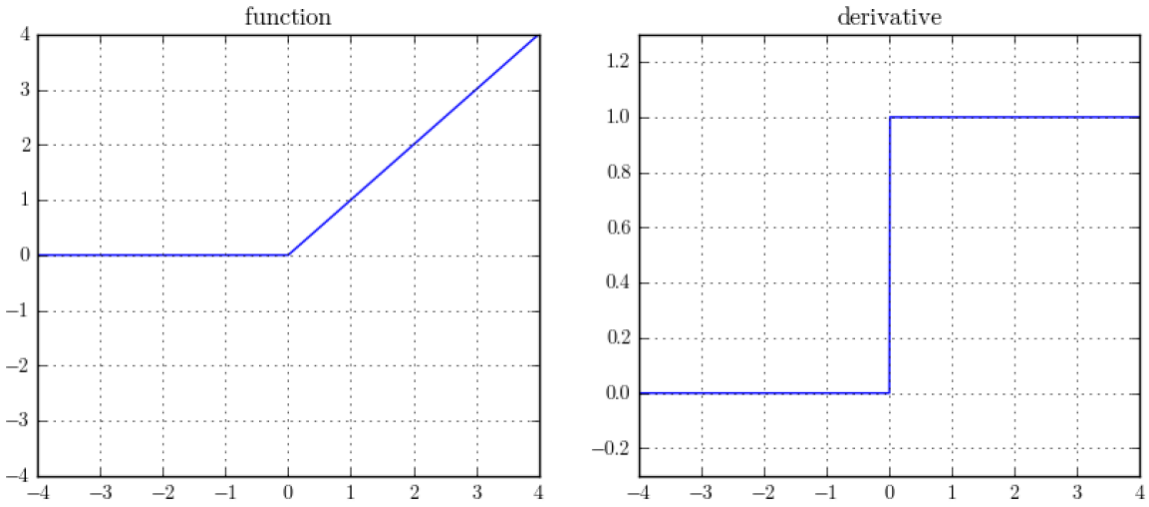


Figure 2.3: ReLu activation function.

2.6 Batch Normalization

During the training of deep neural networks, the distribution of each layer's input changes due to the change of the previous layers parameters. As a result, the training algorithm slows down and the training process becomes harder. This problem is referred to as the internal covariate shift [24]. Ioffe et al. [24] propose a method known as batch normalization to solve this problem. To reduce the internal covariate shift, batch normalization normalizes each scalar feature independently, by setting it to zero mean and unit variance and then scaling and shifting the normalized value for each training mini-batch according to

$$x_i = \frac{(x_i - \mu)}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta, \tag{2.1}$$

where γ is the scale parameter, and β is the shift parameter. γ and β are often initialized at 1.0 and 0.0, respectively. Mean (μ), and variance (σ^2) are computed across each batch. During training, μ and σ^2 are the current input mini-batch statis-

tics, and during the evaluation, they are replaced with running average statistics over the training data. Moreover, batch normalization allows to speed up the training by using a higher learning rate and reduces strong dependency on initialization. It also improves gradient flow through the network. More details can be found in [24].

In convolutional layers, the same normalizing mean and variance are applied at every location within a feature map to ensure the statistics of feature map remain same regardless of spatial location [26]. A minor modification is needed in equation 2.1 to perform batch normalization in CNN which is described in [24].

2.7 Residual Learning

Another problem in training CNN is the degradation of the training accuracy with the increasing network depth. The authors of [23] proposed residual learning of CNN to solve this problem. A residual neural network is composed of units where a layer output is not only fed into the next layer but also into a subsequent ones skipping the layers in between. It has been shown in [23] that the learning of a residual mapping is much easier than the original unreferenced mapping.

Chapter 3

Methodology

3.1 Problem Description

A general block diagram of the image compression system investigated in this thesis is shown in Figure 3.1. In this diagram $\boldsymbol{X}$ represents the image to be compressed and $\boldsymbol{Y}$ is a sub-sampled version of $\boldsymbol{X}$. The image $\boldsymbol{Y}$ is encoded and decoded with a

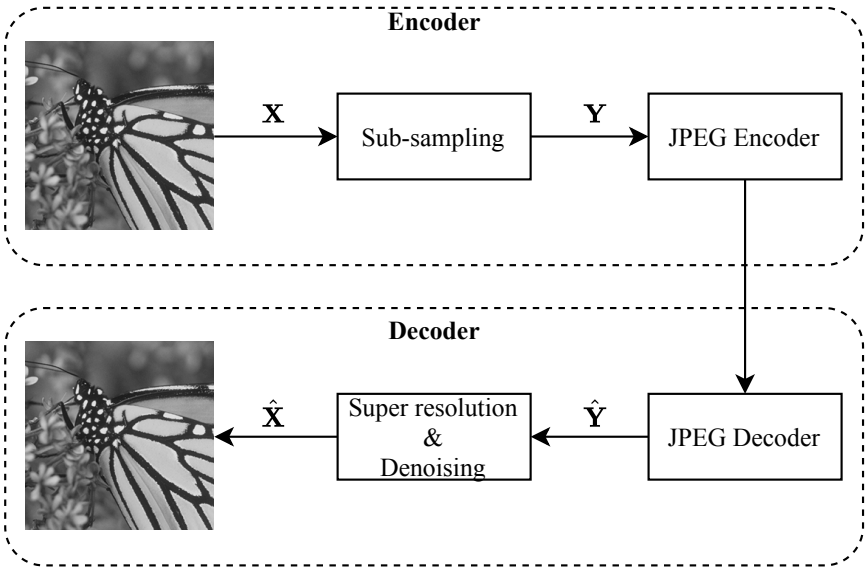


Figure 3.1: Block diagram of underlying problems.

standard JPEG image codec to produce a quantized version $\hat{\boldsymbol{Y}}$. The docoded image goes through denoising and super resolution operations to get an approximation close to the original image $\boldsymbol{X}$. By super resolution we refer to the task of restoring the sub-sampled version of the image back to high resolution (HR). The key advantage of this approach is due to the fact that we are quantizing a smaller image so that we can perform higher compression but with better visual quality compared to directly quantizing the original full size image. However, such an approach poses several challenges which are described below.

On one hand, the fundamental problem with image sub-sampling is that we are fixing the sampling-rate such as 1/2, 1/4 etc. and by doing so we are at the risk of getting *aliasing artifacts* if the sampling-rate is lower than the *Nyquist rate* of the image. On the other hand, quantization typically results in *blocking artifacts*, especially at low bit-rate. Thus the performance of the system critically depends on the sub-sampling and quantization processes. Additionally, restoring the image back to the HR is also challenging as there is no proper definition of a mapping between the sub-sampled space to the HR space. Also, the sub-sampled image could be a cropped portion of the original HR image and, therefore, it can be intractable to map the sub-sampled image to the HR space [33].

In the system we investigate, while the estimation of $\boldsymbol{Y}$ from $\boldsymbol{X}$ suffers only from sub-sampling and blurring, the estimation of $\hat{\boldsymbol{X}}$ from $\hat{\boldsymbol{Y}}$ suffers from quantization artifacts introduced by the JPEG codec. Therefore, the estimation of $\hat{\boldsymbol{X}}$ is more complex and harder compared to estimating $\boldsymbol{Y}$ from $\boldsymbol{X}$ especially under high compression. Previous research show that, a CNN can learn complex nonlinear mapping involved in for image sub-sampling, super resolution, and compression artifact removal from training data [1], [15], [22]. This is the motivation behind the approach

investigated in this thesis.

A more detailed block diagram of the proposed system is shown in Figure 3.2. In this system a CNN architecture referred to as the *compact representation convolutional neural network* (ComCNN) [34] is used for sub-sampling the image to half the original size. The sub-sampled image is then quantized using the JPEG image codec. The quantized version of the image is then restored back to the original size using another CNN which performs a joint image super resolution and removal of JPEG coding artifacts. In this thesis, we propose a CNN architecture referred to as the *enhancement based reconstruction convolutional neural network* (EBR-CNN) for this purpose. Once both CNNs have been trained, the encoding of an image is done by using the ComCNN and the JPEG encoder, while decoding is done by using the JPEG decoder and the EBR-CNN. The CNN based sub-sampled and super resolution techniques tend to produces superior results compared to traditional methods as CNNs can be explicitly trained to minimize the information loss via training. Traditional

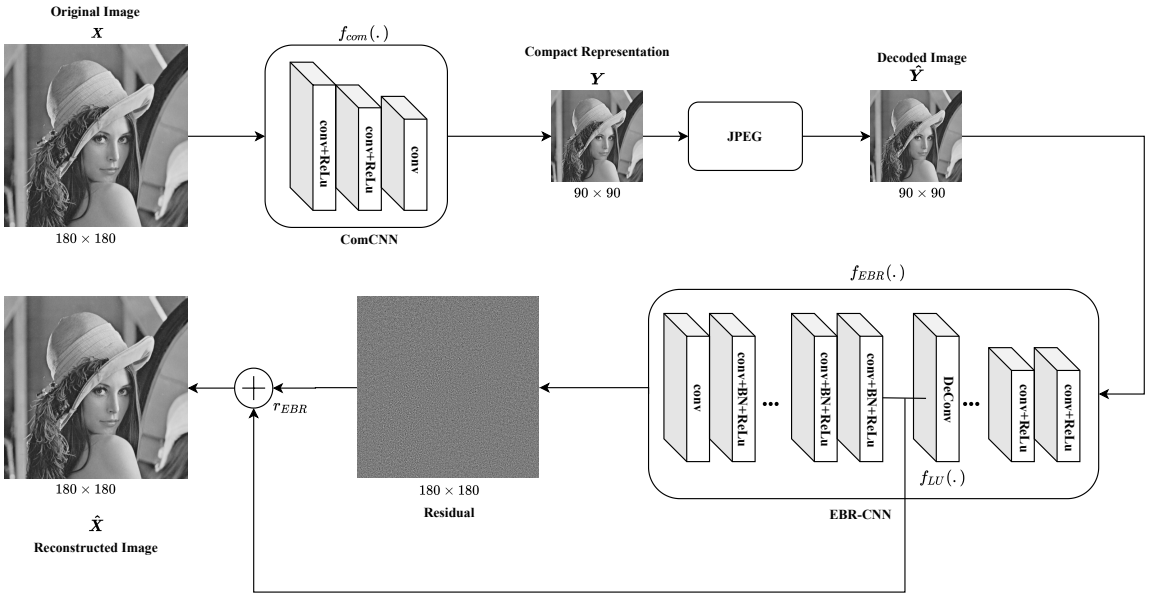


Figure 3.2: Block diagram representation of the proposed image compression framework.

methods such as bilinear or bicubic image sampling, does not involve minimization of a loss function [35].

3.2 Architecture of ComCNN

We have closely followed the ComCNN architecture described in [34]. A summary of this architecture is given here for completeness. The complete network consists of three layers as follows:

1. The first layer is a convolutional layer with 64 filters, each with size 3×3. This layer uses a stride of 1 and padding to maintain the dimension of the input. The output of this convolutional layer is followed by a ReLU activation function. The main purpose of this layer is to extract overlapping patches from input image [34].

2. The second layer is also a convolutional layer where the number of filters, size of the filters, and padding remain the same as of the first layer. Like the first layer, the output of this layer is also followed by a ReLU activation function. The main purpose of this layer is to downscale the input image to half of its original size, and, therefore, the stride is set to 2.

3. The final layer is used for generating the sub-sampled image and, therefore, a convolution layer with only 1 filter of size 3×3 is used. The stride is set to 1 and the padding remains unchanged.

We train the ComCNN independently from the rest of the system.

3.2.1 ComCNN Training Strategy

The image set $\{\boldsymbol{X}_i^{train}, \boldsymbol{X}_i^{train}\}$ is used to train the ComCNN, where $\boldsymbol{X}_i^{train}$ represents the original HR images and $i = 1, .., N$. The main goal of ComCNN is to learn a mapping $f_{com}(\boldsymbol{X}^{train})$ that predicts a compact representation of an image. As the compact image and the original HR image both have different spatial resolutions, they are not directly comparable in training the network. In this case, either we need to sub-sample the original HR image to match the size of the compact image or up-sample the compact image to match the original HR image. The exact method used to address this issue is not mentioned in [34]. Note that if we sub-sample the original HR image, there will be an inevitable loss of information. Therefore, we decided to up-sample the compact image to match with the original HR image using bicubic interpolation for training the network. Accordingly the loss function used to train the ComCNN is defined as

$$l_{com}(\boldsymbol{\theta}_{com}) = \frac{1}{N}\sum_{i=1}^{N} ||g(f_{com}(\boldsymbol{X}_i^{train}; \boldsymbol{\theta}_{com})) - \boldsymbol{X}_i^{train}||^2,$$

where $\boldsymbol{\theta}_{com}$ represents trainable weight parameters of the mapping function $f_{com}(.)$ and $g(.)$ denotes the bicubic up-sampling operator.

3.3 Proposed Architecture of EBR-CNN and Justification

In this section we describe in detail the architecture of the EBR-CNN proposed in this thesis for reconstruction of the JPEG codec output. EBR-CNN consists of eighteen weight layers. As mentioned earlier, this CNN is required to both perform super resolution of the JPEG codec output to the size of the original input image and the removal of compression artifacts produced by the JPEG codec. The main idea that

we propose here is to optimize a single CNN to jointly perform both tasks. This is achieved by cascading the FSRCNN proposed in [14] for image super resolution and DnCNN proposed in [22] for image denoising. The purpose of each layer in the EBR-CNN is described below:

1. **Feature extraction:** The first layer performs the convolution on the JPEG decoded output and uses ReLU as the activation function. This layer generates 56 feature maps of the sub-sampled image with 5×5 filters.

2. **Shrinking:** The dimension of the feature maps of the first layer is very large and, therefore, if directly mapped to HR feature space, it will make the computational complexity of mapping step very high. Thus a shrinking layer is adopted to reduce the feature dimension of the sub-sampled image. This layer is formed by a convolution layer and a ReLU activation function which generates 16 feature maps using 1×1 filters.

3. **Non-linear mapping:** This is the most important step which affects the performance of super-resolution. Multiple (third to sixth) convolution layers with ReLu activation functions are used instead of a single wide layer, which determines both the mapping accuracy and complexity. Each layer contains 12 filters of size 3×3.

4. **Expanding:** This seventh layer acts as the inverse of the shrinking layer. It is introduced to expand the low-dimensional features so that a good restoration quality can be achieved. This layer generates 56 feature maps using filters of size 1×1 for convolution and ReLU activation function is used at the output of each layer.

5. **Deconvolution:** The eighth layer up-samples and aggregates the feature maps

obtained from previous layer with deconvolution using a filter size of 9×9. The deconvolution, which is the opposite operation of the convolution, performs up-sampling by using a stride larger than 1. The output of this layer is an image which has the same size as the original HR image.

6. **Denoising:** Rest of the layers are designed to predict the residual error instead of predicting the denoised image directly. In this way, the final reconstruction image can be obtained by adding the deconvolutional layer output and the residual error. These layers are also convolutional layers which generates similar number (64) of feature maps using 64 filters of size 3×3 followed by batch normalization and ReLu activations except the last layer. The last layer is the convolutional layer. As this layer generates the final output, a single filter with size 3×3 is used.

   All hyperparameters used in EBR-CNN are listed in Table 3.1.

When designing EBR-CNN the ideal objective is to first remove compression artifacts from the sub-sampled image and then perform super resolution. This would

Table 3.1: Hyperparameters used in EBR-CNN.

Layer number	layer name	Filter size	Number of filter	Activation function	Batchnormalization
1	conv layer	5,5	56	ReLU	no
2	conv layer	1,1	16	ReLU	no
3	conv layer	3,3	12	ReLU	no
4	conv layer	3,3	12	ReLU	no
5	conv layer	3,3	12	ReLU	no
6	conv layer	3,3	12	ReLU	no
7	conv layer	1,1	56	ReLU	no
8	deconv layer	9,9	1	n/a	no
9	conv layer	3,3	64	ReLU	yes
10	conv layer	3,3	64	ReLU	yes
11	conv layer	3,3	64	ReLU	yes
12	conv layer	3,3	64	ReLU	yes
13	conv layer	3,3	64	ReLU	yes
14	conv layer	3,3	64	ReLU	yes
15	conv layer	3,3	64	ReLU	yes
16	conv layer	3,3	64	ReLU	yes
17	conv layer	3,3	64	ReLU	yes
18	conv layer	3,3	1	n/a	no

prevent compression artifacts being amplified during super resolution. However, as ground truth for sub-sampled image is not available, it is not possible to train the system to denoise the sub-sampled image first. Performing super resolution prior to denoising allows training of the combined network to minimize the loss between the original and reconstructed HR images.

3.3.1 EBR-CNN Training Strategy

Let $\hat{\boldsymbol{Y}}_i^{train}$ be the output of the JPEG codec for the training set $\boldsymbol{X}_i^{train}$, $i = 1, .., N$. The image set $\{\hat{\boldsymbol{Y}}_i^{train}, \boldsymbol{X}_i^{train}\}$ is used to train the EBR-CNN network. Residual learning [23] strategy is adopted during EBR-CNN training to achieve better performance and faster convergence of the training algorithm. Therefore, the goal of this network is to learn a residual mapping as well as predict the residual image $\boldsymbol{r}_{EBR}^{train} = \boldsymbol{X}^{train} - f_{LU}(\hat{\boldsymbol{Y}}_i^{train})$, where $f_{LU}(.)$ is the up-sampling operator performed by the deconvolutional layer of EBR-CNN. The loss function used to train the EBR-CNN is defined as

$$l_{EBR}(\boldsymbol{\theta}_{EBR}) = \frac{1}{N}\sum_{i=1}^{N} ||f_{EBR}(\hat{\boldsymbol{Y}}_i^{train}; \boldsymbol{\theta}_{EBR}) - \boldsymbol{r}_{EBR,i}^{train}||^2,$$

where $\boldsymbol{\theta}_{EBR}$ represents the weight parameters of the EBR-CNN mapping function $f_{EBR}(.)$.

In principle, a better approach is to train both ComCNN and EBR-CNN as a single network. However, this would involve back propagation of the derivatives from EBR-CNN to ComCNN through the JPEG codec. This is not straight forward as the quantization involved in the JPEG codec is a non-differentiable operation.

Chapter 4

Experimental Results

In this chapter, we present experimental results to demonstrate the performance of the image compression system proposed in chapter 3. These results include comparisons of the proposed system with several other alternatives.

4.1 Design Procedure

The design of the proposed image compression system consists of the following steps.

1. Image compression in the proposed system is performed by a standard JPEG codec. The bit rate in bits per pixel (bpp) of the system is therefore determined by the quality factor (QF) setting of the codec. QF can be varied in the range $1-100$, where a lower value corresponds to higher compression and, therefore, a lower bit rate. However, QF required to achieve a given bit rate depends on the image being compressed and, therefore, QF has to be chosen by trial-and-error to achieve the desired bit rate.

2. Training ComCNN using the available training sets of images.

3. Training EBR-CNN for the ComCNN obtained in step 2 and the JPEG codec with the chosen QF setting.

All experiments have been implemented using Python with the Keras library [36], [37] running on the Tensorflow backend. Training is performed on a ASUS ROG STRIX G531G computer with an Intel Core i7-9750H 2.60GHz CPU, 16 GB of RAM, and an NVIDIA GeForce GTX 1650 GPU. The operating system is Windows 10.

4.2 Parameters Settings

The proposed image compression system is trained using the stochastic gradient descent (SGD) algorithm with an Adam optimizer [28]. The optimizer is initialized with the following default parameters: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$.

One of the main advantages of Adam optimizer is that it performs well with the default parameter values in most problems [38]. Although Adam optimizer can work well without using learning rate decay, it is sometimes useful to decrease the learning rate in order to limit the maximum per-parameter learning rate value [38]. Taking this into account, the initial learning rate is set to 0.001 and decayed exponentially for each epoch according to the rule

$$lr = lr_0 \exp^{(-kt)},$$

where t is the epoch number, lr is the learning rate for current epoch. The initial learning rate lr_0 and the exponential constant k are the hyper parameters.

The filters weights of ComCNN and EBR-CNN are initialized using the *He initializer* [32]. In He initialization, the weights are subtracted from a uniform distribution with zero mean and a variance that depends on the size of the layer's input. The rea-

son behind is taking into account the number of inputs during initialization to make the training faster and more efficient. Both ComCNN and EBR-CNN are trained for 50 epochs. Due to memory constraints of the computer, the batch size has been set to 64.

4.3 Training and Testing Datasets

The training dataset is taken from [39] which contains 400 images with size 180×180. Instead of training the network using whole images, each image is divided into multiple number of patches so that the network can learn the features that are present in small areas (e.g. edges) and thus can improve generalization ability of the network [40].

The number of image patches depends on the stride size. The patch size is set to 40×40 and the stride is set to 20. Therefore, 64 patches are extracted from each image using the equation

$$N = \left\lfloor \frac{M - P}{S} + 1 \right\rfloor^2,$$

where N is the number of patches of size $P \times P$ extracted from an image of size $M \times M$, using a stride S. As the performance of a deep learning network depends on the size of the data set used to train it, we further increase the size of our data set by introducing data augmentation [40].

Data augmentation creates modified samples of the original data which expands the diversity of the dataset. This process involves augmenting each patch 8 times by random horizontal and vertical flips as well as rotations by random angles. As a result, we obtain $400 \times 64 \times 8 = 204800$ patches in total, which are used to train the two CNNs. For testing, we use 8 well known images shown in Figure 4.1 which are commonly

(a) Cameraman

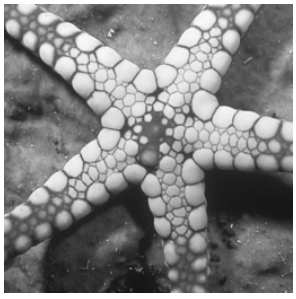
(b) House

(c) Pepper

(d) Butterfly

(e) lena

(f) Ship



(g) Star fish

(h) Parrot

Figure 4.1: Testset. The sub-figures (c), (e), (f) have dimension of 512×512 while the rest has dimension of 256×256 each.

used in literature to bench mark the performance of compression algorithms. Note that the test images are not included in the training set.

4.4 Performance Metrics

The most common performance metrics used for evaluating image compression algorithms are:

- Peak signal-to-noise ratio (PSNR)

- Mean square error (MSE)
- Structural similarity index modulation (SSIM)

4.4.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR is the most commonly used performance metric to measure the quality between the original and compressed images. The PSNR is calculated between two images in the logarithmic decibel scale. It is the ratio between the maximum possible power of the signal and the power of distorting noise that affects the quality of its representation. The PSNR is defined as

$$PSNR = 20 \log_{10} \left[\frac{255}{\sqrt{MSE}} \right] \text{dB}.$$

4.4.2 Mean Square Error (MSE)

The MSE refers to the average of the squared error between the original and compressed images. The MSE is defined as

$$MSE = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} [I'(i,j) - I(i,j)]^2,$$

where I' is the compressed image and I is the original image. The size of the image is $m \times n$.

4.4.3 Structural Similarity Index Modulation (SSIM)

SSIM is a method to calculate the perceptual difference between two similar images. This method is used to compare luminance, contrast, and structure of two images.

SSIM of original image (X) and compressed image (Y) can be defined as

$$SSIM(X,Y) = \frac{(2\mu_x\mu_y + C_1) + (2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

where μ_x and μ_y are the local means for image X and image Y. Similarly, σ_x and σ_y are the standard deviation of image X and Y, $\sigma_{xy} = \sigma_x \times \sigma_y$, and C_1, C_2 are regularization constants.

4.5 Alternatives Used for Comparison

Different types of learning- and non-learning based techniques have been used to evaluate the performance of the proposed compression framework along with JPEG codec alone.

- Pure non-learning based approach shown in Figure 4.2, which will refer to as the method-1. In this method, sub- and up-sampling operations are performed using bicubic interpolation.

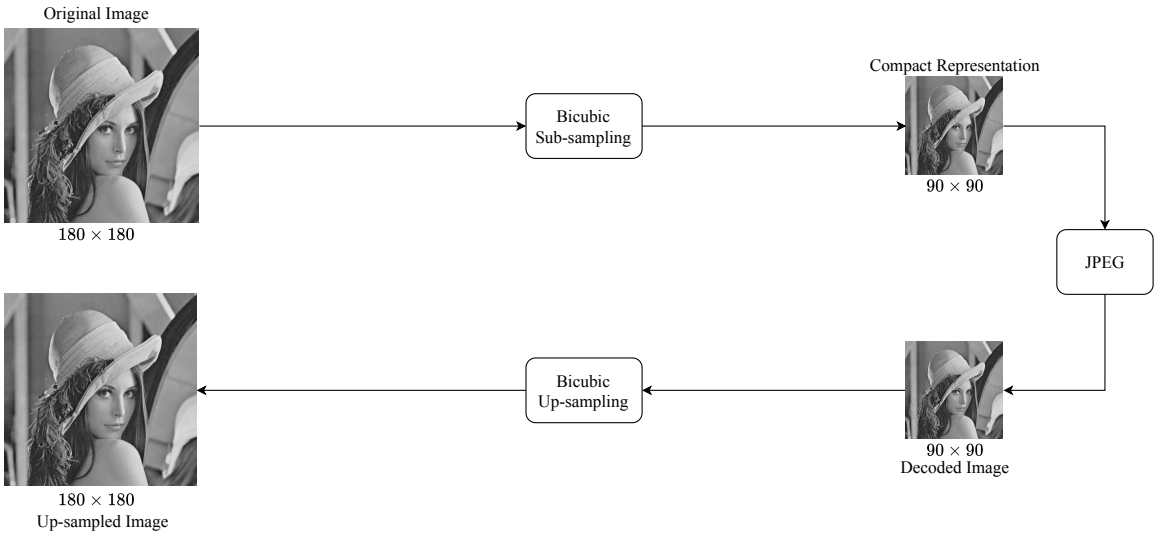


Figure 4.2: System architecture of Method-1.

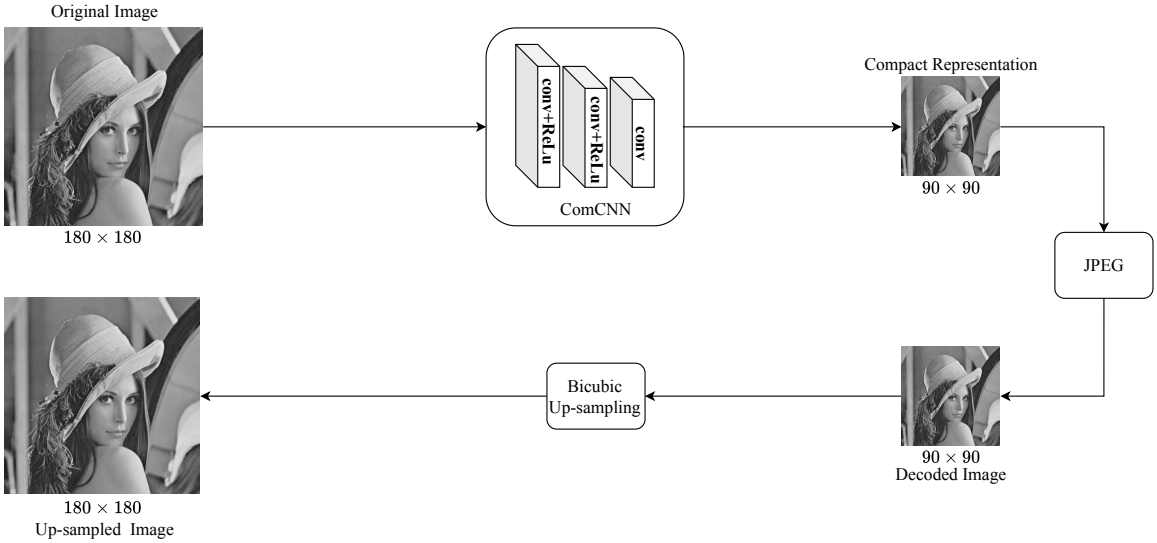


Figure 4.3: System architecture of Method-2.

- The method-2 (see Figure 4.3) adopts the learning based approach ComCNN for sub-sampling the image. However, bicubic interpolation is used for upsampling operation.

- In method-3 (Figure 4.4) sub-sampling operation is performed using ComCNN

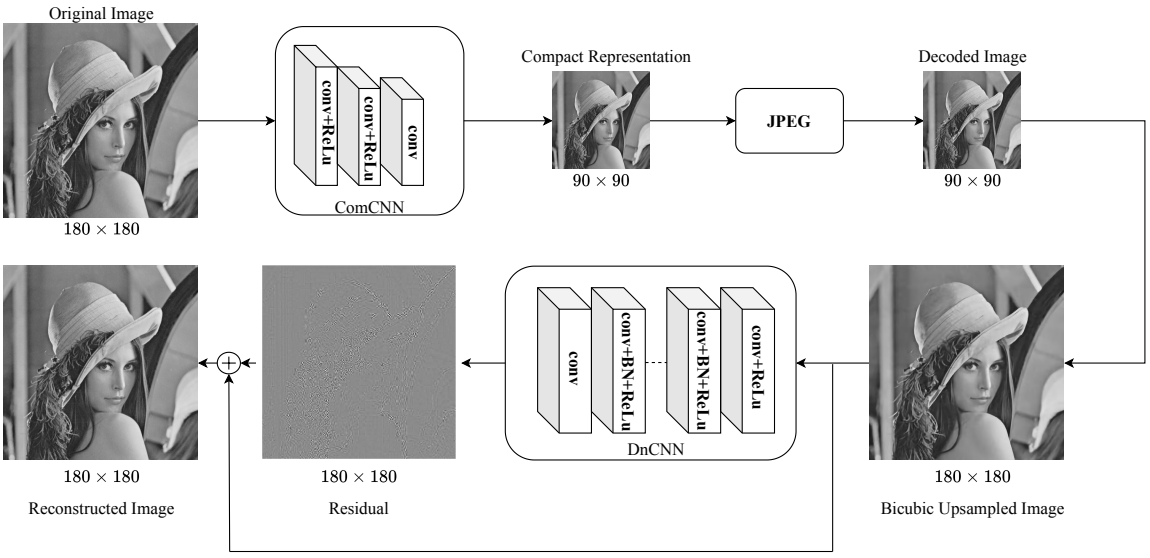


Figure 4.4: System architecture of Method-3.

but reconstruction is performed using bicubic up-sampling and DnCNN [22].

All systems use the same bit rate (bpp). However, each system requires a different QF to achieve a given bit rate which has been found by trial-and-error, using the Algorithm 3 in Appendix B.

To compare the experimental results quantitatively, we have used PSNR and SSIM as quality metrics. We also compare the visual quality of the reconstructed images for qualitative evaluation. Table 4.1−4.6 present the PSNR and SSIM comparisons of the investigated methods for all test images. Each table compares the performance of all methods at a given compression ratio as determined by a value of QF. However, note that, in each table, all methods use the same bit rate for a given image.

Table 4.1: Comparison of PSNRs (dB) for JPEG QF=5.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.2	24.45	24.73	24.89	25.76	**26.21**
House	0.2	27.77	29.61	29.70	30.81	**31.11**
Peppers	0.2	27.26	29.20	29.34	30.35	**30.64**
Butterfly	0.3	23.80	24.39	24.46	26.86	**27.07**
Lena	0.2	27.32	29.84	29.88	30.89	**31.08**
Ship	0.2	25.56	26.49	26.58	27.41	**27.56**
Starfish	0.3	23.99	24.71	24.75	25.83	**26.04**
Parrot	0.3	24.51	25.00	25.15	25.58	**25.58**
Average	0.2	25.59	26.75	26.84	27.94	**28.16**

Table 4.2: Comparison of PSNRs (dB) for JPEG QF=10.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.3	26.47	25.44	25.73	27.23	**27.33**
House	0.3	30.56	30.69	30.92	32.63	**32.75**
Peppers	0.2	30.15	30.22	30.39	31.61	**31.63**
Butterfly	0.5	26.67	26.57	26.72	29.22	**29.32**
Lena	0.2	30.41	31.35	31.44	32.37	**32.51**
Ship	0.3	28.13	28.23	28.45	29.44	**29.53**
Starfish	0.4	26.72	26.57	26.73	28.34	**28.55**
Parrot	0.4	26.84	26.01	26.28	26.99	**27.09**
Average	0.3	28.24	28.14	28.33	29.73	**29.84**

Table 4.3: Comparison of PSNRs (dB) for JPEG QF=15.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.4	27.71	25.99	26.25	27.84	**27.84**
House	0.3	32.07	31.15	31.43	33.08	**33.32**
Peppers	0.3	31.55	30.63	30.86	31.94	**32.16**
Butterfly	0.6	28.15	27.59	27.61	30.32	**30.53**
Lena	0.3	31.95	32.04	32.14	33.14	**33.24**
Ship	0.4	29.53	28.69	28.95	30.01	**30.00**
Starfish	0.6	28.20	27.54	27.50	29.45	**29.47**
Parrot	0.5	**28.08**	26.57	26.72	27.53	27.68
Average	0.4	29.66	28.78	28.93	30.41	**30.53**

Table 4.4: Comparison of SSIMs for JPEG QF=5.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.2	0.7262	0.7569	0.7602	0.8030	**0.8111**
House	0.2	0.7731	0.8003	0.8012	0.8359	**0.8388**
Peppers	0.2	0.7118	0.7977	0.7964	0.8329	**0.8358**
Butterfly	0.3	0.7412	0.7872	0.7871	0.8647	**0.8673**
Lena	0.2	0.7324	0.8226	0.8236	0.8487	**0.8510**
Ship	0.2	0.6582	0.7163	0.7193	0.7472	**0.7492**
Starfish	0.3	0.7083	0.7513	0.7540	0.7984	**0.8002**
Parrot	0.3	0.7124	0.7926	0.7957	0.8276	**0.8309**
Average	0.2	0.7205	0.7781	0.7797	0.8198	**0.8231**

Table 4.5: Comparison of SSIMs for JPEG QF=10.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.3	0.7951	0.7948	0.7982	0.8412	**0.8429**
House	0.3	0.8184	0.8259	0.8301	0.8609	**0.8623**
Peppers	0.2	0.7916	0.8282	0.8276	**0.8570**	0.8567
Butterfly	0.5	0.8417	0.8551	0.8566	0.9173	**0.9187**
Lena	0.2	0.8214	0.8586	0.8608	0.8792	**0.8802**
Ship	0.3	0.7673	0.7891	0.7942	0.8167	**0.8167**
Starfish	0.4	0.8162	0.8292	0.8314	0.8701	**0.8724**
Parrot	0.4	0.8136	0.8329	0.8378	0.8634	**0.8652**
Average	0.3	0.8082	0.8267	0.8296	0.8633	**0.8644**

Table 4.6: Comparison of SSIMs for JPEG QF=15.

Test Images	bpp	JPEG only	Method-1	Method-2	Method-3	Proposed
Cameraman	0.4	0.8359	0.8237	0.8207	0.8589	**0.8589**
House	0.3	0.8470	0.8371	0.8417	0.8680	**0.8704**
Peppers	0.3	0.8275	0.8413	0.8415	0.8648	**0.8649**
Butterfly	0.6	0.8826	0.8854	0.8792	0.9326	**0.9356**
Lena	0.3	0.8586	0.8740	0.8759	0.8891	**0.8913**
Ship	0.4	0.8137	0.8069	0.8118	0.8329	**0.8336**
Starfish	0.6	0.8615	0.8554	0.8528	**0.8910**	0.8906
Parrot	0.5	0.8559	0.8559	0.8532	0.8762	**0.8784**
Average	0.4	0.8478	0.8474	0.8471	0.8767	**0.8779**

From Table 4.1, it can be seen that methods which use image sub-sampling and super resolution perform better than the JPEG algorithm used alone. Even the non-learning based approach achieves a 1.16 dB gain in PSNR and a 0.185 gain in SSIM. However, as QF is increased (reduced compression), the JPEG codec outperforms the non-learning based approach, method-1. This is quite expected, as a JPEG algorithm introduces less blocking artifacts with less compression. However, the problem with non-learning based image sub-sampling and reconstruction is that, only bicubic up-sampling are not able to significantly reduce the noise introduced by sub-sampling and compression, rather they amplify noise during the up-sampling operation. A better solution can be achieved by the addition of learning based approach ComCNN for image sub-sampling which is trained to preserve more useful information [34]. However, as we still use bicubic up-sampling during image reconstruction, the gains in PSNR over JPEG algorithm decreases with increasing QF.

It is well known in literature that CNN based image reconstruction methods often outperform traditional methods such as simple bicubic/bilinear methods [41]. Our experimental results also seem to support this observation. The relative performance gains of the proposed method over the other methods are presented in Table 4.7 and Table 4.8. It is noticeable from these experimental results that, as the QF is increased, the performance gap between the JPEG algorithm alone and the CNN based approaches closes. Such behavior of CNN based approaches is to be expected as CNNs are trained to eliminate artifacts due to image sub-sampling and JPEG compression. As the QF is increased, compression artifacts largely disappear and hence the performance gap closes.

The visual quality comparisons for QF = 5 and QF = 10 are presented in Figure 4.5−4.8. It can be seen that blocking artifacts due to JPEG compression are

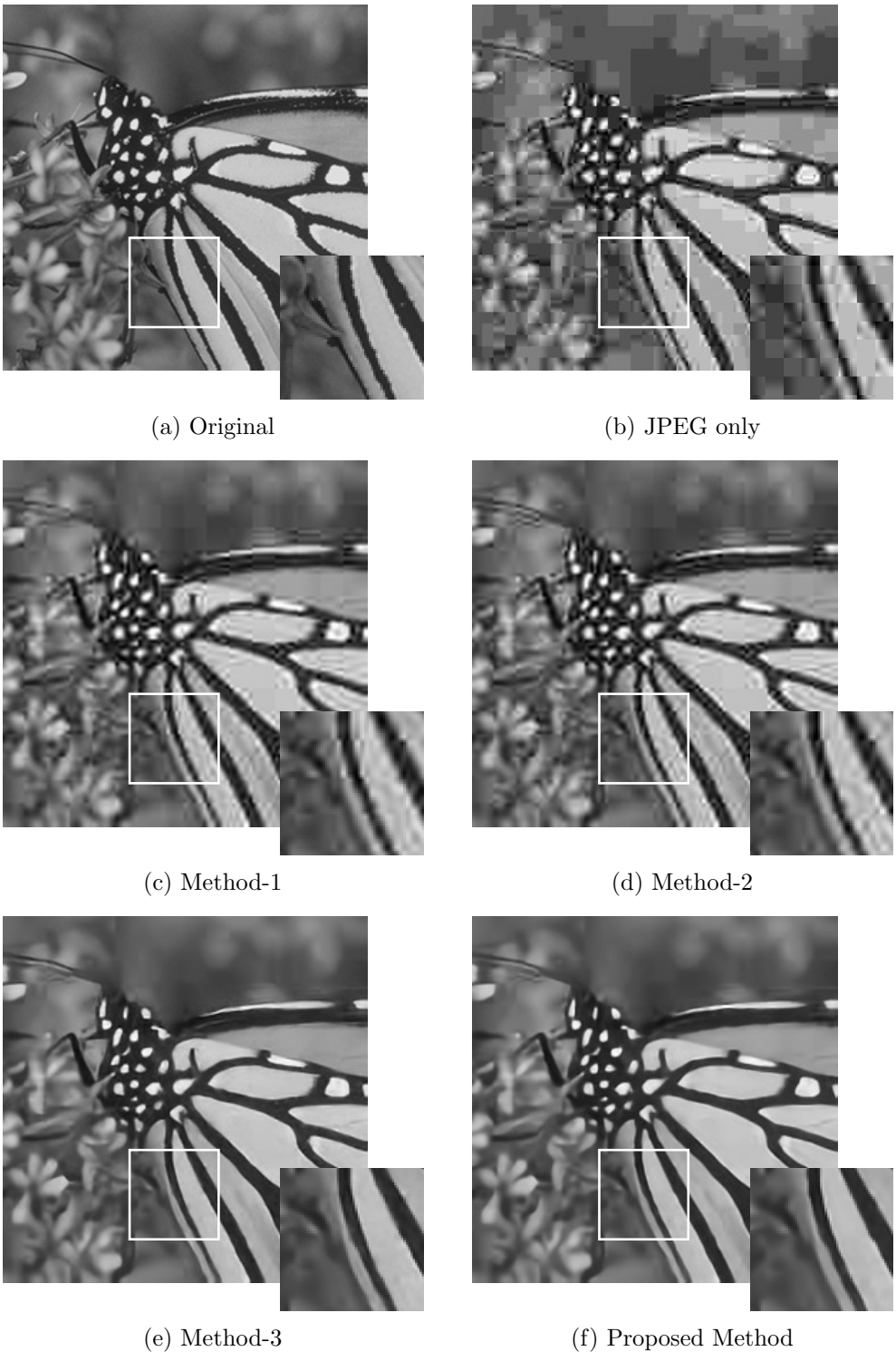
Table 4.7: Gain over other investigated methods in terms of PSNR.

Test Images	QF=5				QF=10				QF=15			
	JPEG only	**Method 1**	**Method 2**	**Method 3**	**JPEG only**	**Method 1**	**Method 2**	**Method 3**	**JPEG only**	**Method 1**	**Method 2**	**Method 3**
Cameraman	1.76	1.48	1.32	0.45	0.86	1.89	1.6	0.1	0.13	1.85	1.59	0
House	3.34	1.5	1.41	0.3	2.19	2.06	1.83	0.12	1.25	2.17	1.89	0.24
Peppers	3.38	1.44	1.3	0.29	1.48	1.41	1.24	0.02	0.61	1.53	1.3	0.22
Butterfly	3.27	2.68	2.61	0.21	2.65	2.75	2.6	0.1	2.38	2.94	2.92	0.21
Lena	3.76	1.24	1.2	0.19	2.1	1.16	1.07	0.14	1.29	1.2	1.1	0.1
Ship	2.00	1.07	0.98	0.15	1.4	1.3	1.08	0.09	0.47	1.31	1.05	-0.1
Starfish	2.05	1.33	1.29	0.21	1.83	1.98	1.82	0.21	1.27	1.93	1.97	0.02
Parrot	1.07	0.58	0.43	0	0.25	1.08	0.81	0.1	-0.4	1.11	0.96	0.15
Average	**2.58**	**1.42**	**1.32**	**0.23**	**1.60**	**1.70**	**1.51**	**0.11**	**0.88**	**1.76**	**1.60**	**0.11**

Table 4.8: Gain over other investigated methods in terms of SSIM.

Test Images	QF=5				QF=10				QF=15			
	JPEG only	**Method 1**	**Method 2**	**Method 3**	**JPEG only**	**Method 1**	**Method 2**	**Method 3**	**JPEG only**	**Method 1**	**Method 2**	**Method 3**
Cameraman	0.0849	0.0542	0.0509	0.0081	0.0478	0.0481	0.0447	0.0017	0.0230	0.0352	0.0382	0
House	0.0657	0.0385	0.0376	0.0029	0.0439	0.0364	0.0322	0.0014	0.0234	0.0333	0.0287	0.0024
Peppers	0.124	0.0381	0.0394	0.0029	0.0651	0.0285	0.0291	-0.0003	0.0374	0.0236	0.0234	0.0001
Butterfly	0.1261	0.0801	0.0802	0.0026	0.077	0.0636	0.0621	0.0014	0.0530	0.0502	0.0564	0.0030
Lena	0.1186	0.0284	0.0274	0.0023	0.0588	0.0216	0.0194	0.0010	0.0327	0.0173	0.0154	0.0022
Ship	0.091	0.0329	0.0299	0.002	0.0494	0.0276	0.0225	0	0.0199	0.0267	0.0218	0.0007
Starfish	0.0919	0.0489	0.0462	0.0018	0.0562	0.0432	0.0410	0.0023	0.0291	0.0352	0.0378	-0.0004
Parrot	0.1185	0.0383	0.0352	0.0033	0.0516	0.0323	0.0274	0.0018	0.0225	0.0225	0.0252	0.0022
Average	**0.1025**	**0.0449**	**0.0433**	**0.0032**	**0.4498**	**0.0376**	**0.2784**	**0.0012**	**0.0301**	**0.0305**	**0.0309**	**0.0013**

more obvious in the images decoded directly with the JPEG codec. However, these artifacts are less visible when JPEG compression is performed on a compact representation of the original image. Both method-3 and the proposed method achieve better visual quality than rest of the methods investigated here. Method-3 and the proposed approach not only reduce the visual artifacts significantly but also preserve more visual details on both edges and textures as well.

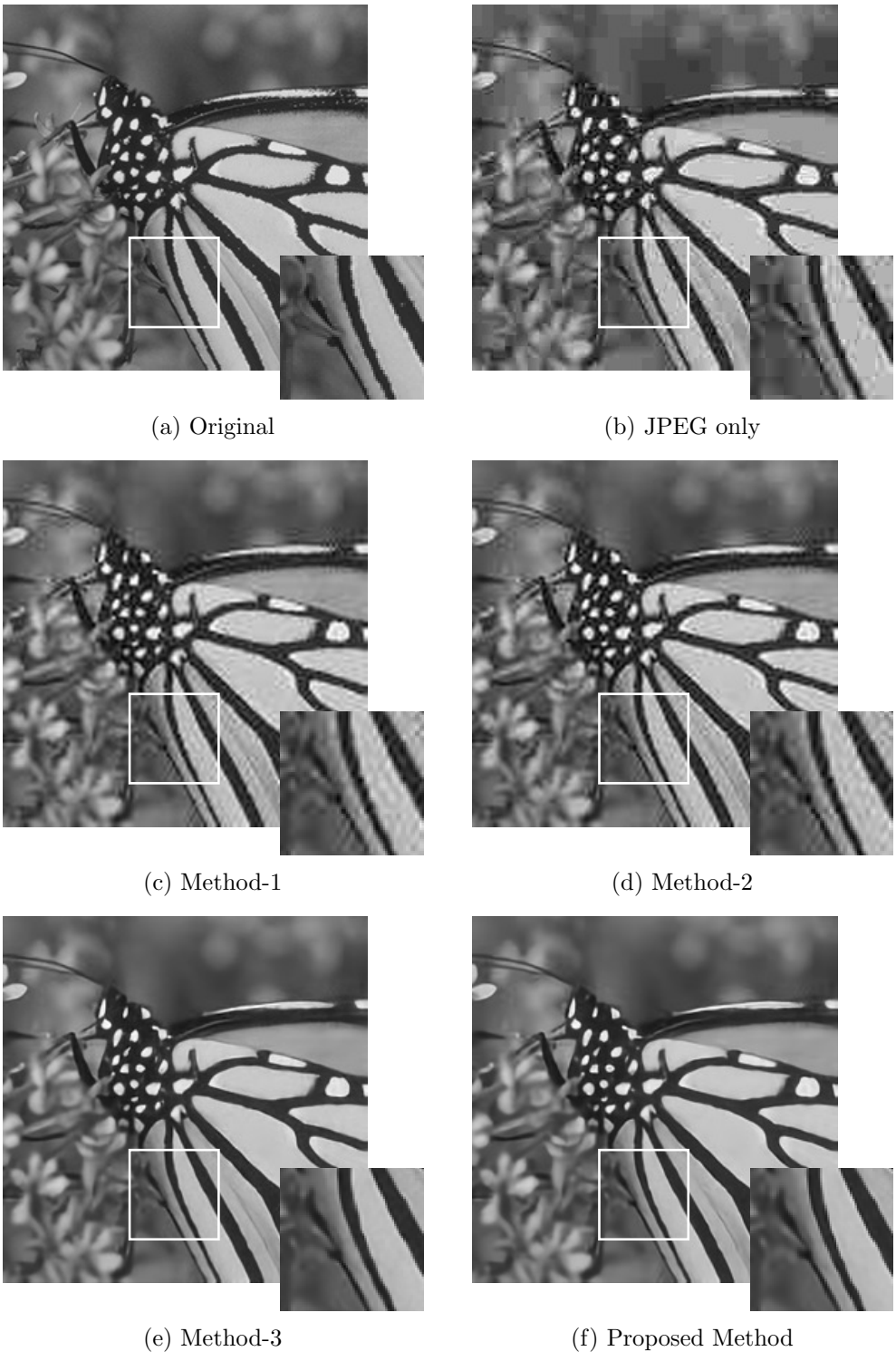


(a) Original (b) JPEG only

(c) Method-1 (d) Method-2

(e) Method-3 (f) Proposed Method

Figure 4.5: Visual quality comparison of reconstructed images (*Butterfly*, QF=5).



(a) Original (b) JPEG only

(c) Method-1 (d) Method-2

(e) Method-3 (f) Proposed Method

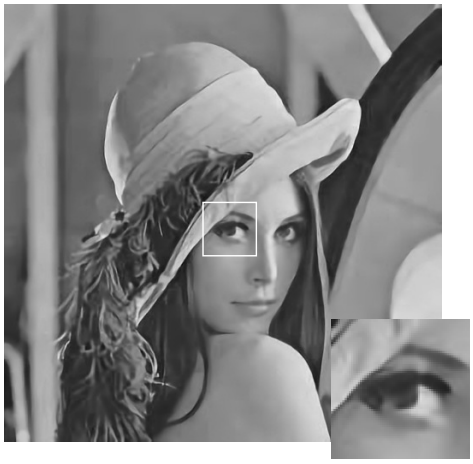
Figure 4.6: Visual quality comparison of reconstructed images (*Butterfly*, QF=10).

(a) Original

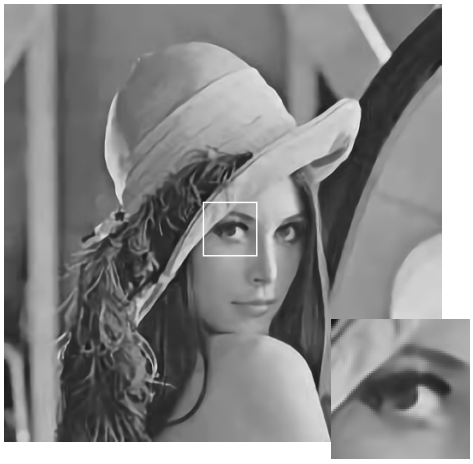
(b) JPEG only

(c) Method-1

(d) Method-2



(e) Method-3



(f) Proposed Method

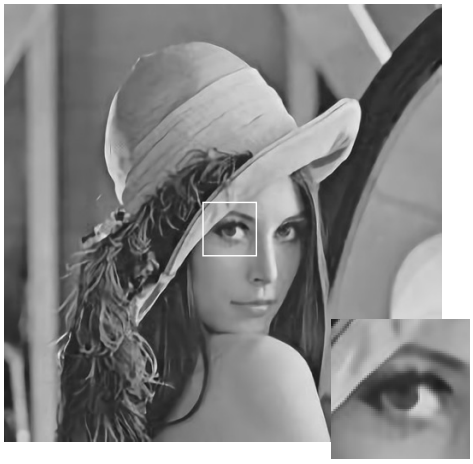
Figure 4.7: Visual quality comparison of reconstructed images (*Lena*, QF=5).

(a) Original

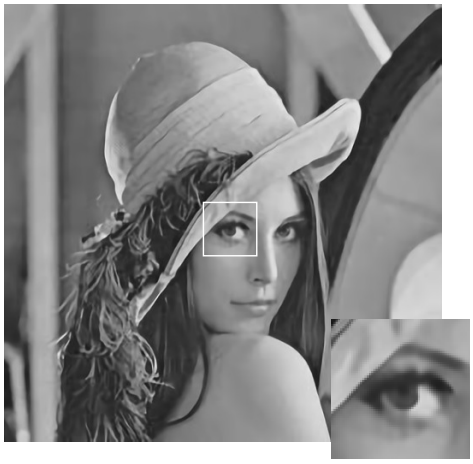
(b) JPEG only

(c) Method-1

(d) Method-2



(e) Method-3



(f) Proposed Method

Figure 4.8: Visual quality comparison of reconstructed images (*Lena*, QF=10).

4.6 Comparisons of Computational Time

In the previous section, we have seen that both the DnCNN and EBR-CNN based image compression frameworks perform quite similarly in terms of gains in PSNR, SSIM, and reducing visual artifacts. Therefore, it is necessary to evaluate the performance of these two frameworks through a different performance metric to better understand why adopting EBR-CNN in post-processing phase can be beneficial over DnCNN in compression framework. To do so, we computed the total time required to train both the networks over 50 training epochs with similar training strategies as mentioned in chapter 3 and the results are presented in Figure 4.9. Also, we computed the MSE during each training epoch and is presented in Figure 4.10. From Figure 4.9, it can be seen that the proposed image reconstruction method has considerably lower training time compared to the DnCNN (around 8.5 hours which is nearly half of the times required to train DnCNN). That is, the proposed method converges faster to a

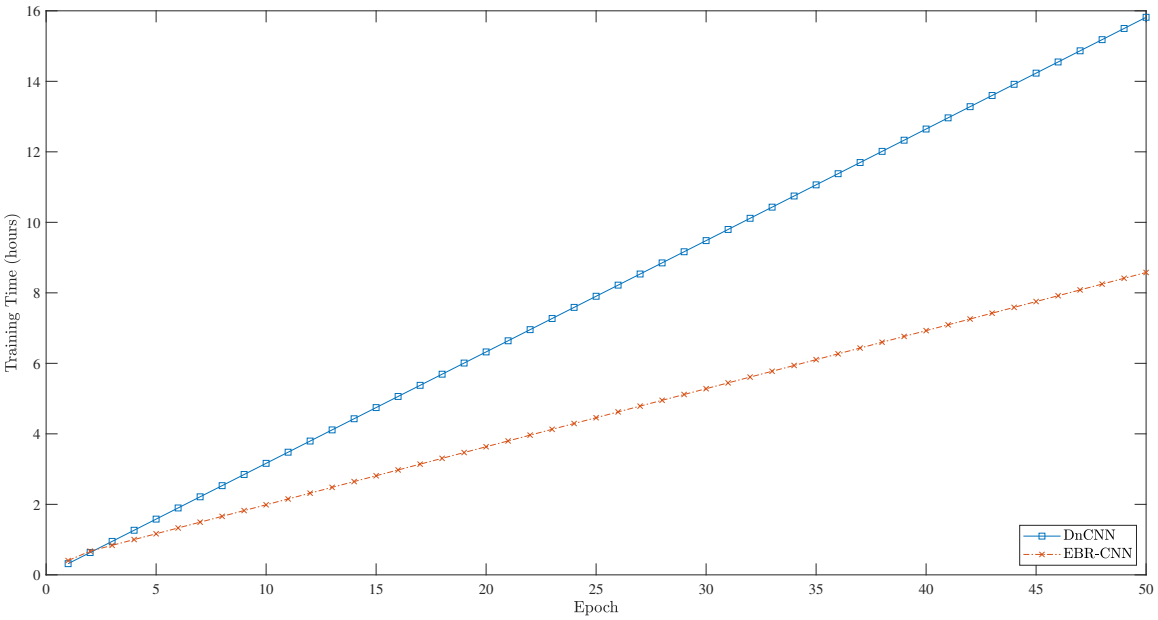


Figure 4.9: Total Training time as a function of epochs.

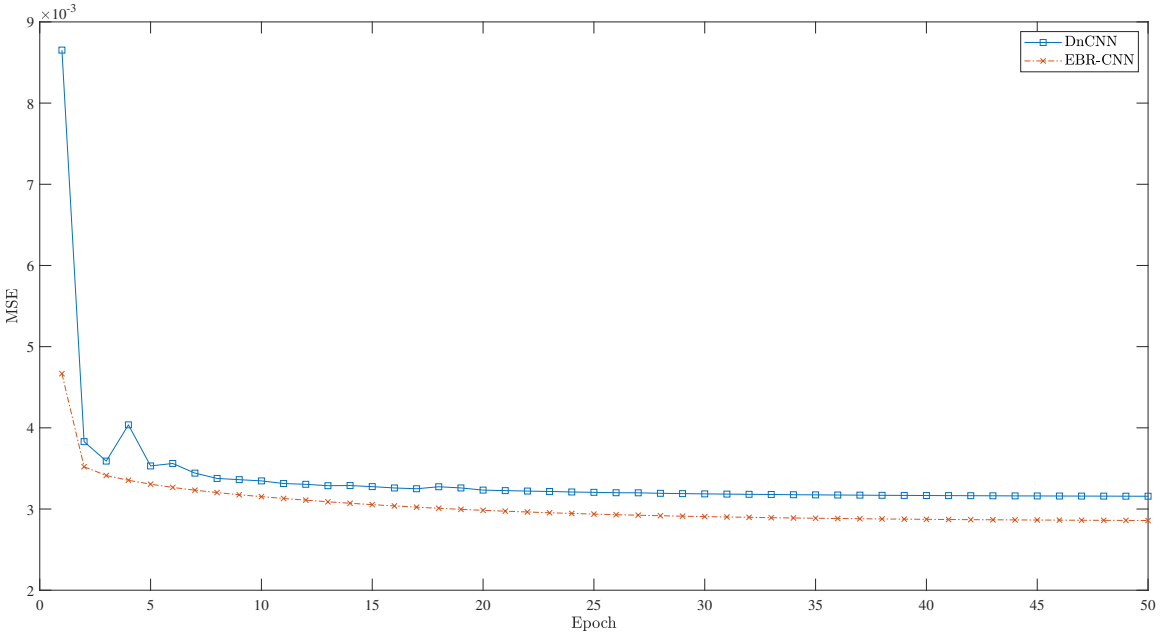


Figure 4.10: MSE vs epoch.

lower MSE than the DnCNN which is also evident in Figure 4.10.

4.7 EBR-CNN Depth Analysis

When designing the EBR-CNN network, it is important to pay attention to the numbers of layers being used in the network. The number of layers is referred to as the *depth* of the network. The first 8-layers of EBR-CNN are designed according to FS-RCNN [14]. During depth analysis we fix the first 8-layers which perform different tasks such as feature extraction, shrinking, non-linear mapping, expanding, and deconvolution. The purpose of the rest of the layers are to learn the residue which are designed as mentioned in DnCNN [22]. From the experimental results available in the literature we can conclude that increasing the depth in general increases the performance of CNNs [23], [42]. Since the enhancement layers are designed in a similar way, we perform depth analysis on the enhancement layers by varying the number of layers

Table 4.9: PSNR results for different depth sizes of EBR-CNN

Image	Depth=13	Depth=18	Depth=28
Cameraman	27.26	27.33	27.37
House	32.65	32.75	32.75
Pepper	31.52	31.63	31.64
Butterfly	29.40	29.32	29.36
Lena	32.59	32.51	32.41
Ship	29.47	29.53	29.53
Starfish	28.37	28.55	28.54
Parrot	27.09	27.09	27.03
Average	29.79	**29.84**	29.83

between 5 to 20 along with the 8 initial layers. Table 4.9 compares the performance of EBR-CNN with different numbers of layers. It can be seen that increasing depth size of 13 layers (8 initial layers + 5 enhancement layers) to 18 layers (8 initial layers + 10 enhancement layers) also increases the performance. The performance with a depth size of 28 layers (8 initial layers + 20 enhancement layers) is quite similar that with depth size of 18 layers but involves a higher complexity and a lot more training time. We, therefore, used 18 layers to produce the final results.

4.8 Robustness to Quality Factors Variations

In the proposed method, EBR-CNN is trained for a given QF and hence a given bit rate. Using the same system with a different QF setting will inevitably results in a performance degradation. To investigate the sensitivity of the proposed method to QF, we decided to test the proposed method on never seen artifacts that arise due to different JPEG QFs. In order to conduct the experiment, the proposed method was trained for QF=5, QF=10, and QF=15 and was then tested on QFs in the range 5 to 30. During testing, each image from the testset is first compressed by setting QF to a given value in the interval 5 to 30 and then passed through an EBR-CNN trained for a

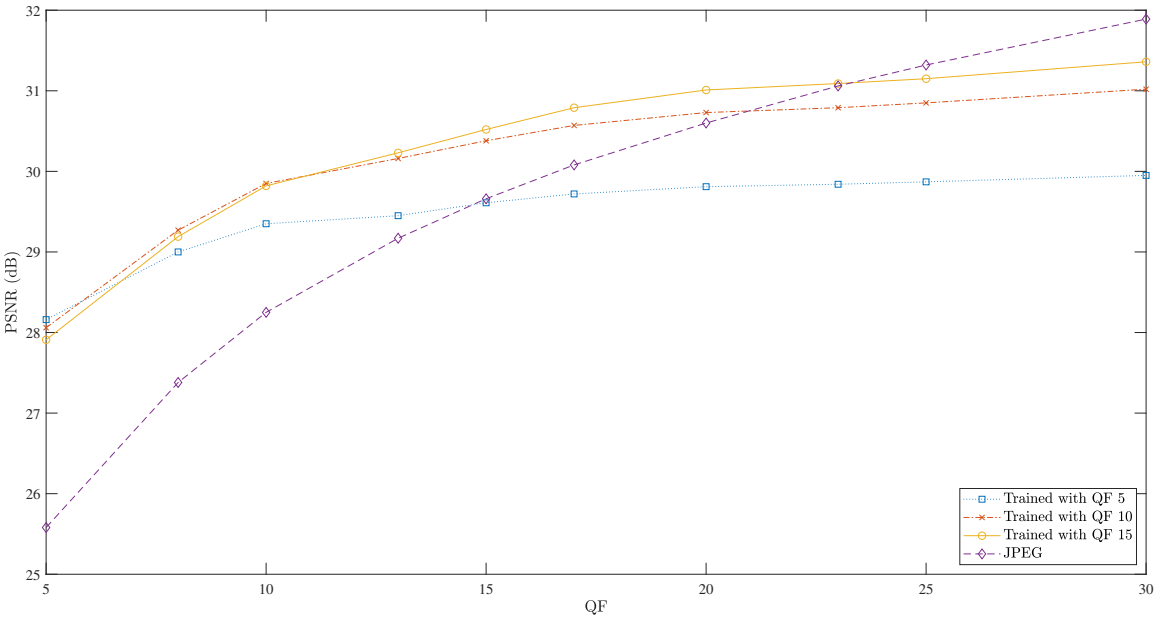


Figure 4.11: Robustness of PSNR performance against QF mismatch.

different QF. Then the PSNR and SSIM of each reconstructed image were measured. The results shown here have been obtained by taking the average PSNR and SSIM over whole set of test images. It should be noted that the test images are not included in training set. Figure 4.11 and 4.12 show performance comparisons of EBR-CNN trained with different QFs. It is clear that the EBR-CNN performs best only if the QF used during training matches the QF used to compress the images under testing. However, it can be seen from the results that the EBR-CNN shows fair amount of robustness to deviation of QF from the value used during training. For example, the performance of EBR-CNN trained for QF=10 remains within about 1dB in PSNR and 0.2 in SSIM of the performance EBR-CNNs trained for QF=5 and QF=15.

4.9 Experimental Results on Standard Image Datasets

In order to evaluate the effectiveness and robustness of the proposed method, we conducted experiments on three standard datasets including Set5 [43], Set14 [44]

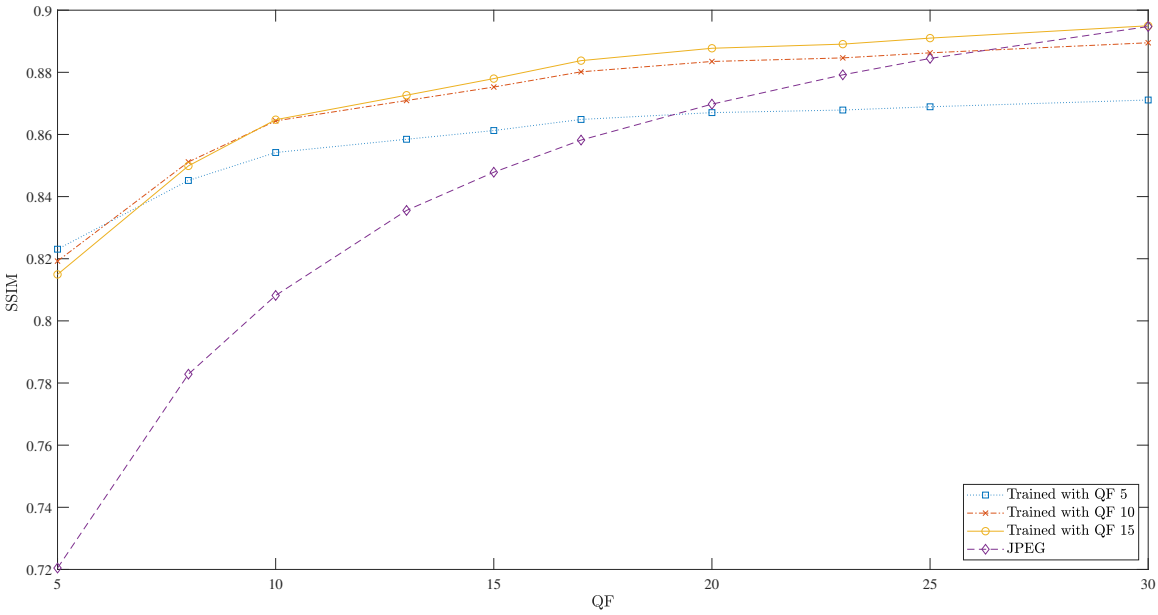


Figure 4.12: Robustness of SSIM performance against QF mismatch.

and Urban100 [45]. The average PSNR and SSIM results are shown in Table 4.10. From these results, it can be observed that the performance of the proposed method consistently outperforms the JPEG algorithm alone for all three datasets.

Table 4.10: Average PSNR(dB) and SSIM results of JPEG and the proposed method for quality factors 5,10 and 15 for Set5, Set14 and Urban100.

Datasets	Quality Factors	JPEG		Proposed method	
		PSNR	SSIM	PSNR	SSIM
Set5	5	26.20	0.7216	**29.40**	**0.8397**
	10	29.06	0.8184	**31.33**	**0.8805**
	15	30.55	0.8562	**32.05**	**0.8947**
Set14	5	24.92	0.6791	**26.93**	**0.7587**
	10	27.52	0.7889	**28.34**	**0.8204**
	15	28.91	0.8348	**28.97**	**0.8433**
Urban100	5	23.39	0.6988	**25.05**	**0.7759**
	10	25.65	0.7984	**26.51**	**0.8358**
	15	26.94	0.8378	**26.99**	**0.8449**

Chapter 5

Conclusion and Future Work

In this thesis, we have investigated an approach to improve the performance of the JPEG algorithm at high compression using a CNN framework, which uses a CNN to sub-sample the input image prior to the JPEG encoding and another to up-sample and denoise the decoded image. The main contribution of this thesis is the investigation of a novel CNN, referred to as the EBR-CNN, used for joint denoising and super resolution of the JPEG compressed images.

The experimental results have demonstrated that proposed method can consistently outperform the standalone JPEG codec. The results have also shown that the proposed EBR-CNN can be trained much faster and achieves better performance than the previously reported DnCNN which has been widely used in the literature for image denoising.

Although EBR-CNN shows very promising results, there is room for further improvements. One possibility is to train both ComCNN and EBR-CNN to minimize an end-to-end loss function. There is evidence in the literature which suggest that such end-to-end training can lead to improvements. However, end-to-end training of two CNNs which sandwich a JPEG image codec is an open research problem as rounding

function used in quantization is not differentiable as required by the back propagation algorithm. It would be also interesting to explore new ways to increase the performance of the trained models while maintaining their competitive speed by varying the number and size of the layers and filters. Finally, another interesting avenue of research is to incorporate variable rate compression into the proposed method.

Appendix A

An overview of JPEG algorithm

JPEG algorithm is the most widely used lossy compression algorithm for images. Typically, a lossy compression method discards some amount of information from an image to obtain higher compression rates than purely lossless methods. Since it throws out a certain level of information, it is not possible to recover the original image from compressed data. Obviously, the decoded image quality degrades with the level of compression. Block diagram representation of the JPEG encoder and decoder are shown in Figure A.1 and Figure A.2 respectively.

It is known that neighboring pixels of an image are highly correlated, which gives rise to spatial redundancy in an image. Therefore, the first step of JPEG encoding

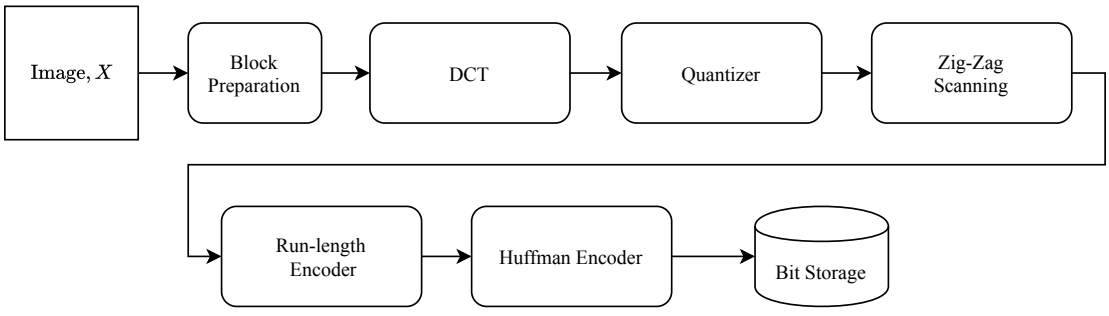


Figure A.1: JPEG encoder.

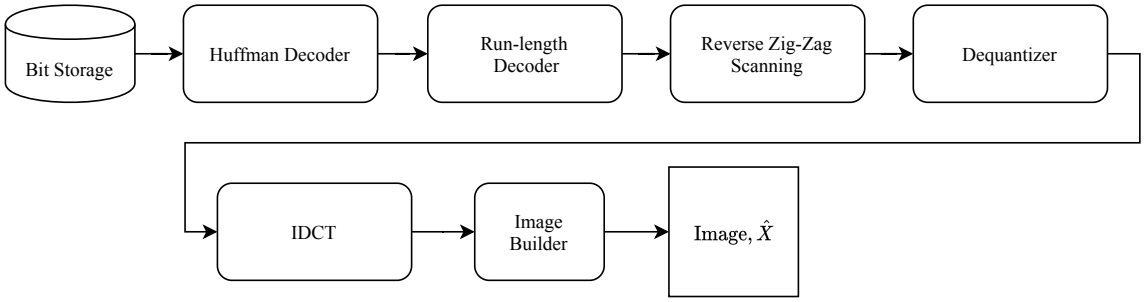


Figure A.2: JPEG decoder.

is block preparation, which captures some neighboring pixels in a block. Typically, a blocks of size 8×8 are considered. The spatial frequency of magnitudes can vary significantly within such a block [46]. It has been found out that the human visual system is less sensitive to high spatial frequencies than the lower ones. If the magnitude of high spatial frequency components falls below a specific threshold, then they will not be detected by human eyes [46]. Therefore, it is possible to introduce some level of compression without degrading the perceptual quality of an image just by discarding some or all of higher frequency components in an image. To do so, first, we need to transform the original spatial domain image input into the equivalent frequency domain, which will help to capture the high-frequency components. In the JPEG algorithm, this is done by using the discrete cosine transform (DCT). overall compression can be achieved by applying relatively coarse quantization to these frequency components.

The forward DCT can be expressed as

$$F(u,v) = \frac{2}{N} C(u)C(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \cos[\frac{\pi(2x+1)u}{2N}] \cos[\frac{\pi(2y+1)v}{2N}],$$

where u and $v = 0, ...N-1$, $N = 8$, and

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } k = 0 \\ 1 & \text{otherwise,} \end{cases}$$

and the inverse DCT can be expressed as

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u)C(v)F(u,v) \cos[\frac{\pi(2x+1)u}{2N}] \cos[\frac{\pi(2y+1)v}{2N}],$$

where x and $y = 0, ...N-1$. The Forward DCT is a part of the JPEG encoder, while the inverse DCT belongs to the JPEG decoder. With forward DCT, we will get a matrix of DCT coefficients. The top-left DCT coefficient in the matrix is the DC coefficient, and the rest are AC coefficients. Now to eliminate the higher frequency components, quantization is performed over the matrix of DCT coefficients. This is done by dividing each DCT coefficient by its corresponding quantizer step size and

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Figure A.3: Sample quantization table [2].

then rounding to the nearest integer [46]. Quantized coefficient are obtained as

$$F_Q(u,v) = Round(\frac{F(u,v)}{Q(u,v)}),$$

where $Q(u,v)$ represents the quantization step sizes as shown in Figure A.3, and the decoded DCT coefficients are given by

$$F'_Q(u,v) = F_Q(u,v) \times Q(u,v).$$

After quantization, many zeros appear in higher frequencies. These quantized coefficients can be stored more efficiently by rearranging the elements in zigzag order from left to bottom to get an array of quantized coefficients and applying run length coding for lossless compression. The rest of quantized coefficients are encoded with an entropy encoder, e.g., Huffman encoder.

Appendix B

Algorithm 3

Initialize: $O_{BPP} = 0$, $T_{BPP} = 0$, $QF = x$, $QF_t = 50$, $f_o = 0$,$f_t = 0$
Get an image
Get image height h and width w
Encode the image with QF
Compute the binary file size f_o
Calculate, $O_{BPP} \leftarrow \frac{f_o}{h \times w}$
Get compact representation of original image using ComCNN
while true **do**
    Encode with QF_t
    Compute binary file size f_t
    Calculate, $T_{BPP} \leftarrow \frac{f_t}{h \times w}$
    $V_{BPP} \leftarrow (O_{BPP} - T_{BPP})$
    if $TH_L \leq V_{BPP} \leq TH_H$ **then** ▷ where TH_L represents low threshold value and TH_H represents high threshold value
        return QF_t
        break
    else if $V_{BPP} > TH_H$ **then**

17: $a \leftarrow QF_t + 1$

18: $b \leftarrow QF_t + QF_t$

19: $QF_t \leftarrow \lfloor \frac{a+b}{2} \rfloor$

20: **else if** $V_{BPP} < TH_H$ **then**

21: $c \leftarrow \lfloor QF_t - \frac{QF_t}{2} + 1 \rfloor$

22: $d \leftarrow QF_t - 1$

23: $QF_t \leftarrow \lfloor \frac{c+d}{2} \rfloor$

24: break

Bibliography

[1] L. Cavigelli, P. Hager, and L. Benini, "Cas-cnn: A deep convolutional neural network for image compression artifact suppression," in *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2017, pp. 752–759.

[2] K. Sayood, *Introduction to Data Compression, (Morgan Kaufmann Series in Multimedia Information and Systems)*. Elsevier, 2005.

[3] K. Karadimitriou and M. Fenstermacher, "Image compression in medical image databases using set redundancy," in *Data Compression Conference DCC*. IEEE, 1997, p. 445.

[4] Z. Wang, A. C. Bovik, and L. Lu, "Why is image quality assessment so difficult?" in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 4. IEEE, 2002, pp. IV–3313.

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

[6] S. Golomb, "Run-length encodings (corresp.)," *IEEE Transactions on Information Theory*, vol. 12, no. 3, pp. 399–401, 1966.

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

[8] S. Ma, X. Zhang, C. Jia, Z. Zhao, S. Wang, and S. Wang, "Image and video compression with neural networks: A review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 30, no. 6, pp. 1683–1698, 2020.

[9] G. K. Wallace, "The jpeg still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. 18–34, 1992.

[10] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The jpeg 2000 still image compression standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.

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

[12] A. Gupta and K. R. Rao, "A fast recursive algorithm for the discrete sine transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 38, no. 3, pp. 553–557, 1990.

[13] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, 2015.

[14] C. Dong, C. C. Loy, and X. Tang, "Accelerating the super-resolution convolutional neural network," in *European Conference on Computer Vision*. Springer, 2016, pp. 391–407.

[15] J. Kim, J. Kwon Lee, and K. Mu Lee, "Accurate image super-resolution using very deep convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1646–1654.

[16] K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image denoising by sparse 3-d transform-domain collaborative filtering," *IEEE Transactions on Image Processing*, vol. 16, no. 8, pp. 2080–2095, 2007.

[17] A. Foi, V. Katkovnik, and K. Egiazarian, "Pointwise shape-adaptive dct for high-quality denoising and deblocking of grayscale and color images," *IEEE Transactions on Image Processing*, vol. 16, no. 5, pp. 1395–1411, 2007.

[18] X. Liu, X. Wu, J. Zhou, and D. Zhao, "Data-driven sparsity-based restoration of JPEG-compressed images in dual transform-pixel domain," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5171–5178.

[19] H. Chang, M. K. Ng, and T. Zeng, "Reducing artifacts in jpeg decompression via a learned dictionary," *IEEE Transactions on Signal Processing*, vol. 62, no. 3, pp. 718–728, 2013.

[20] X. Zhang, W. Yang, Y. Hu, and J. Liu, "Dmcnn: Dual-domain multi-scale convolutional neural network for compression artifacts removal," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 390–394.

[21] C. Dong, Y. Deng, C. Change Loy, and X. Tang, "Compression artifacts reduction by a deep convolutional network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 576–584.

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

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning*, 2015, pp. 448–456.

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[26] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, 2016.

[27] S. Y. Fei-Fei Li, Justin Johnson. Lecture 4: Backpropagation and neural networks. [Online]. Available: https://http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference for Learning Representations (ICLR)*, 2015. [Online]. Available: https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:main.html

[29] S. Ruder. (2016, Jan) An overview of gradient descent optimization algorithms. [Online]. Available: https://ruder.io/optimizing-gradient-descent/

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. null, p. 2121–2159, Jul. 2011.

[31] G. Hinton, N. Srivastava, and K. Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[32] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 2011, pp. 315–323.

[33] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao, "Deep learning for single image super-resolution: A brief review," *IEEE Transactions on Multimedia*, vol. 21, no. 12, pp. 3106–3121, 2019.

[34] F. Jiang, W. Tao, S. Liu, J. Ren, X. Guo, and D. Zhao, "An end-to-end compression framework based on convolutional neural networks," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 28, no. 10, pp. 3007–3018, 2017.

[35] M. Unser, A. Aldroubi, and M. Eden, "Enlargement or reduction of digital images with minimum loss of information," *IEEE Transactions on Image Processing*, vol. 4, no. 3, pp. 247–258, 1995.

[36] F. Chollet *et al.* (2015) Keras. Online. [Online]. Available: https://keras.io

[37] N. Ketkar, "Introduction to keras," in *Deep learning with Python*. Springer, 2017, pp. 97–111.

[38] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht, "The marginal value of adaptive gradient methods in machine learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4148–4158.

[39] Y. Chen and T. Pock, "Trainable nonlinear reaction diffusion: A flexible framework for fast and effective image restoration," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1256–1272, 2016.

[40] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[41] Y. Li, D. Liu, H. Li, L. Li, Z. Li, and F. Wu, "Learning a convolutional neural network for image compact-resolution," *IEEE Transactions on Image Processing*, vol. 28, no. 3, pp. 1092–1107, 2018.

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

[43] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. Alberi-Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," 2012, pp. 1–10.

[44] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *International Conference on Curves and Surfaces*. Springer, 2010, pp. 711–730.

[45] J.-B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5197–5206.

[46] F. Halsall, *Multimedia Communications: Applications, Networks, Protocols, and Standards*. Pearson education, 2001.