Enhanced Image Compression Through Advanced Residual Network Architectures

Abstract

This manuscript provides an in-depth explanation of the methodology developed for a recent image compression challenge. The method primarily incorporates two innovative aspects: the application of advanced residual networks for enhanced compression and the utilization of sub-pixel convolution techniques for efficient up-sampling during decompression. The efficacy of these methodologies, which achieved a high Multiscale Structural Similarity Index (MS-SSIM) of 0.972 under a strict bit rate constraint of 0.15 bits per pixel (bpp) while maintaining reasonable computational demands during the evaluation stage.

1 Introduction

Image compression remains a crucial research area within the field of signal processing, aiming to facilitate more efficient data storage and transfer. Conventional image compression algorithms, like the various JPEG standards, often employ manually designed encoder/decoder frameworks. However, with the emergence of novel image formats and the proliferation of high-resolution mobile devices, there is a growing recognition that existing standards may not represent the most effective or universal solutions for image compression.

Recently, deep learning-based techniques have shown a surge of progress in the image compression domain. Some of these methods employ generative models, trained adversarially, to effectively learn the underlying distribution of images, resulting in impressive subjective quality even at exceptionally low bit rates. Other works utilize recurrent neural networks to iteratively compress residual information, enabling progressive coding which allows for multiple quality levels within a single compression operation. Further advancements have been made by focusing on relaxing quantization constraints and improving entropy modeling, leading to enhanced performance compared to established image compression methods.

Nevertheless, identifying an optimal network structure presents a formidable challenge across various machine learning applications, including image compression. This paper primarily discusses two important aspects of network design for image compression. The first concerns the selection of kernel size, a parameter that significantly influences compression effectiveness in traditional algorithms. Motivated by its impact in classical methods, this paper presents experiments that use different filter sizes to prove that larger kernel sizes contribute to improved coding efficiency. Building upon this, a strategy is presented that utilizes a deep residual learning approach, allowing for the maintenance of a broad receptive field while utilizing a reduced number of parameters. This approach not only decreases the model's overall size but also substantially enhances its performance. Additionally, the architecture of up-sampling operations within the decoder plays a pivotal role in determining the quality of reconstructed images and the presence of artifacts. This issue, extensively studied in the context of super-resolution, involves various implementations for up-sampling layers, such as interpolation, transposed convolution, and sub-pixel convolution. This work compares two commonly used up-sampling methods, transposed and sub-pixel convolutions, to demonstrate their relative performance in the context of image compression.

.

2 Methodology

The fundamental network architectures employed in this research are based on prior works that have demonstrated state-of-the-art compression performance. The network is structured as a pair of autoencoders. The primary autoencoder is responsible for optimizing the rate-distortion tradeoff inherent in image compression. The loss function can be expressed as:

$$J = \lambda d(x, \hat{x}) + R(\hat{y}) \tag{1}$$

where λ is a parameter that balances the importance of rate and distortion. The secondary autoencoder handles the encoding of side information, which is used to model the probability distribution of the compressed data. A Gaussian scale mixture approach is utilized to develop an image-adaptive entropy model, with scale parameters conditioned on a hyperprior.

2.1 From Small Kernel Size to Large Kernel Size

In traditional image compression techniques, the size of transform filters significantly affects coding efficiency, especially for high-definition videos. Initially, transform sizes were small, but as the field progressed, there was a gradual shift towards larger sizes to better capture spatial correlations and semantic details. The experiments detailed in this paper, using a standard dataset, explore the impact of different filter sizes in both the main and auxiliary autoencoders. Table 1 indicates that for the Baseline architecture, larger kernel sizes lead to better rate-distortion outcomes. Similarly, Table 2 demonstrates comparable improvements for the HyperPrior architectures. Table 3 reveals that employing large kernels in the auxiliary autoencoder does not enhance rate-distortion performance and may even negatively impact it. This is likely due to the small size of the compressed codes, which makes smaller kernels sufficient for effective encoding. An excessive number of trainable parameters can hinder the learning process.

Table 1: The effect of kernel size on Baseline on Kodak, optimized by MSE with λ = 0.015.

Method	PSNR	MS-SSIM	Rate
Baseline-3	32.160	0.9742	0.671
Baseline-5	32.859	0.9766	0.641
Baseline-9	32.911	0.9776	0.633

Table 2: The effect of kernel size on HyperPrior on Kodak, optimized by MSE with λ = 0.015.

Method	PSNR	MS-SSIM	Rate
HyperPrior-3	32.488	0.9742	0.543
HyperPrior-5	32.976	0.9757	0.518
HyperPrior-9	33.005	0.9765	0.512

Table 3: The effect of kernel size in the auxiliary autoencoder on Kodak, optimized by MS-SSIM with λ = 5.

Method	PSNR	MS-SSIM	Rate
HyperPrior-9-Aux-5	26.266	0.9591	0.169
HyperPrior-9-Aux-9	26.236	0.9590	0.171

2.2 From Shallow Network to Deep Residual Network

In terms of receptive field coverage, a sequence of four 3x3 kernels can encompass the same area as a single 9x9 kernel but with a reduced parameter count. Initial attempts to substitute a large kernel with multiple 3x3 filters encountered convergence issues during training. To address this, shortcut connections were incorporated between adjacent 3x3 kernels. The resultant deep residual network

architecture for image compression is denoted as ResNet-3x3(4), signifying that a stack of four 3x3 kernels achieves an equivalent receptive field to a 9x9 kernel. To minimize parameter overhead, GDN/IGDN activation functions are applied only once within each residual unit when the output dimensions change. For the remaining convolutional layers, parameter-free Leaky ReLU activations are employed to introduce non-linearity. As indicated in Table 4, ResNet-3x3(4) surpasses both ResNet-3x3(3) and Hyperprior-9 in terms of performance.

Table 4: Comparison of residual networks and upsampling operations on Kodak, optimized by MS-SSIM with $\lambda = 5$.

Method	PSNR	MS-SSIM	Rate
Hyperprior-9	26.266	0.9591	0.1690
ResNet-3x3(3)	26.378	0.9605	0.1704
ResNet-3x3(4)-TConv	26.457	0.9611	0.1693
ResNet-3x3(4)-SubPixel	26.498	0.9622	0.1700

2.3 Upsampling Operations at Decoder Side

The encoder-decoder structure is characterized by its symmetrical design. While down-sampling at the encoder is typically achieved using strided convolution filters, up-sampling at the decoder can be implemented through various methods, such as bicubic interpolation, transposed convolution, and sub-pixel convolution. Considering the importance of rapid end-to-end learning, bicubic interpolation was excluded, and a comparison was made between the two widely used up-sampling techniques: transposed convolution (TConv) and sub-pixel convolution (SubPixel). To implement sub-pixel convolution, the channel count is expanded fourfold, followed by the application of a depth-to-space operation. The results presented in Table 4 demonstrate that sub-pixel convolution filters offer slight improvements in both PSNR and MS-SSIM compared to transposed convolution filters.

3 Experiments

For the training process, 256x256 image patches were extracted from a large-scale image dataset. A batch size of 8 was employed, and training was conducted for up to 2 million iterations to ensure stable convergence. Optimization was performed using the Adam optimizer, with an initial learning rate of 1 x 10⁻⁴, reduced to 1 x 10⁻⁵ for the final 80,000 iterations.

Two primary strategies were implemented. The first strategy, termed "Wide Bottleneck," involves increasing the model's capacity by expanding the number of filters. Since increasing filters in large feature maps significantly increases computational cost (FLOPs), the filter count was only raised in the encoder's final layer, from 128 to 192. This results in a minor FLOPs increase, as detailed in Table 5. While Bottleneck192 effectively reduces the bit rate, it also leads to some quality degradation compared to Bottleneck128.

Table 5: The effect of wide bottleneck on Kodak dataset.

Method	PSNR	MS-SSIM	Rate
ResNet-3x3(4)-Bottleneck128	26.498	0.9622	0.1700
ResNet-3x3(4)-Bottleneck192	26.317	0.9619	0.1667

The second strategy is "Rate Control." For achieving a target bit rate, two models are trained at distinct bit rates by adjusting the λ parameter. This allows for adaptive selection during encoding to approach the target bit rate while maximizing MS-SSIM, as shown in Table 6. A single bit is added to the bitstream to indicate the model used for decoding, without increasing decoder complexity.

4 Results

Table 7 summarizes the compression performance of the proposed methods on a validation dataset.

Table 6: Rate control on validation dataset.

Method	λ	PSNR	MS-SSIM	Rate
ResNet-3x3(4)-Bottleneck192	5	29.708	0.9697	0.1369
ResNet-3x3(4)-Bottleneck192	10	30.710	0.9765	0.1816

Table 7: Results on validation dataset.

Entry	Description	PSNR	MS-SSIM	Rate
Kattolab	HyperPrior-9	28.902	0.9674	0.134
Kattolab	HyperPrior-9 + Rate Control	29.102	0.9701	0.150
Kattolab	ResNet-3x3(4)-TConv + Rate Control	29.315	0.9716	0.150
Kattolabv2	ResNet-3x3(4)-SubPixel+ Rate Control	29.300	0.9720	0.150
KattolabSSIM	ResNet-3x3(4)-SubPixel + Wide Bottleneck + Rate Control	29.211	0.9724	0.150

While deep residual networks enhance coding gain, they also lead to a substantial increase in model size. This section analyzes the parameter count and model complexity in terms of floating-point operations per second (FLOPs) for various architectures. Specifically, using the HyperPrior-9 architecture as an example, Table 8 provides a layer-wise breakdown of model size. The number of parameters and FLOPs are calculated as follows:

$$Para = (h \times w \times C_{in} + 1) \times C_{out} \tag{2}$$

$$FLOPs = Para \times H' \times W' \tag{3}$$

where $h \times w$ represents the kernel size, $H' \times W'$ denotes the output dimensions, and C_{in} and C_{out} are the number of input and output channels, respectively. The +1 term is omitted when no bias is used. Quantization and leaky-ReLU are parameter-free. GDN operates across channels but not spatial positions, resulting in a parameter count of $(C_{in} + 1) \times C_{out}$. The total FLOPs for GDN and inverse GDN calculations are minimal. This analysis primarily focuses on the backbone of convolutional layers, so the FLOPs of GDN, inverse GDN, and factorized prior are not included in the comparison. Table 9 presents a comparison of different architectures, with the last column showing the relative FLOPs using Baseline-5 as the reference. The proposed models achieve improved coding performance with relatively low computational complexity.

5 Conclusion

This manuscript details the proposed deep residual learning framework and sub-pixel convolution technique for image compression, forming the foundation of the submitted entries: Kattolab, Kattolabv2, and KattolabSSIM. The results demonstrate that these approaches achieve a high MS-SSIM of 0.972 under a bit rate constraint of 0.15 bpp, while maintaining a moderate level of computational complexity during the validation phase.

Table 8: The model size analysis of HyperPrior-9.

Layer	Kernel		Channel		Output	Para
FLOPs	h	w	C_{in}	C_{out}	H x W	
conv1	9	9	3	128	128 x 128	31232
5.12 x 10⁹						
conv2	9	9	128	128	64 x 64	1327232
5.44 x 10⁷						
conv3	9	9	128	128	32 x 32	1327232
1.36 x 10⁷						
conv4	9	9	128	128	16 x 16	1327104
3.40 x 10⁶						
GDN/IGDN						99072
-						
Hconv1	3	3	128	128	16 x 16	147584
3.78 x 10⁶						
Hconv2	5	5	128	128	8 x 8	409728
2.62 x 10⁶						
Hconv3	5	5	128	128	4 x 4	409728
6.56 x 10⁵						
FactorizedPrior						5888
-						
HTconv1	5	5	128	128	8 x 8	409728
2.62 x 10⁶						
HTconv2	5	5	128	192	16 x 16	614592
1.57 x 10⁷						
HTconv3	3	3	192	256	16 x 16	442624
1.13 x 10⁷						
layer1			256	640	16 x 16	164480
4.21 x 10⁶						
layer2			640	512	16 x 16	328192
8.40 x 10⁶						
layer3			512	256	16 x 16	131072
3.36 x 10⁶						
Tconv1	9	9	128	128	32 x 32	1327232
1.36 x 10⁷						
Tconv2	9	9	128	128	64 x 64	1327232
5.44 x 10⁷						
Tconv3	9	9	128	128	128 x 128	1327232
2.17 x 10¹⁰						
Tconv4	9	9	128	3	256 x 256	31107
2.04 x 10⁷						
Total						11188291
3.88 x 10¹⁰						

Table 9: The model complexity of different architectures.

Method	Para	FLOPs	Relative
Baseline-3	997379	4.25 x 10⁹	0.36
Baseline-5	2582531	1.18 x 10¹⁰	1.00
Baseline-9	8130563	3.82 x 10¹⁰	3.24
HyperPrior-3	4055107	4.78 x 10⁹	0.40
HyperPrior-5	5640259	1.23 x 10¹⁰	1.04
HyperPrior-9	11188291	3.88 x 10¹⁰	3.28
ResNet-3x3(3)	5716355	1.75 x 10¹⁰	1.48
ResNet-3x3(4)	6684931	2.43 x 10¹⁰	2.06
ResNet-3x3(4)-SubPixel	8172172	2.50 x 10¹⁰	2.12
ResNet-3x3(4)-SubPixel-Bottleneck192	11627916	2.56 x 10¹⁰	2.17