Real-Time Adaptive Image Compression

Oren Rippel [* 1] **Lubomir Bourdev** [* 1]

Abstract

We present a machine learning-based approach to lossy image compression which outperforms all existing codecs, while running in real-time. Our algorithm typically produces files 2.5 times smaller than JPEG and JPEG 2000, 2 times smaller than WebP, and 1.7 times smaller than BPG on datasets of generic images across all quality levels. At the same time, our codec is designed to be lightweight and deployable: for example, it can encode or decode the Kodak dataset in around 10ms per image on GPU. Our architecture is an autoencoder featuring pyramidal analysis, an adaptive coding module, and regularization of the expected codelength. We also supplement our approach with adversarial training specialized towards use in a compression setting: this enables us to produce visually pleasing reconstructions for very low bitrates.

1. Introduction

Streaming of digital media makes 70% of internet traffic, and is projected to reach 80% by 2020 (CIS, 2015). However, it has been challenging for existing commercial compression algorithms to adapt to the growing demand and the changing landscape of requirements and applications. While digital media are transmitted in a wide variety of settings, the available codecs are "one-size-fits-all": they are hard-coded, and cannot be customized to particular use cases beyond high-level hyperparameter tuning.

In the last few years, deep learning has revolutionized many tasks such as machine translation, speech recognition, face recognition, and photo-realistic image generation. Even though the world of compression seems a natural domain for machine learning approaches, it has not yet benefited from these advancements, for two main reasons. First, our deep learning primitives, in their raw forms, are not well-suited to construct representations sufficiently compact. Recently, there have been a number of important efforts by Toderici et al. (2015; 2016), Theis et al. (2016), Ballé et al. (2016), and Johnston et al. (2017) towards alleviating this: see Section 2.2. Second, it is difficult to develop a deep learning compression approach sufficiently efficient for deployment in environments constrained by computation power, memory footprint and battery life.

In this work, we present progress on both performance and computational feasibility of ML-based image compression.

Our algorithm outperforms all existing image compression approaches, both traditional and ML-based: it typically produces files 2.5 times smaller than JPEG and JPEG 2000 (JP2), 2 times smaller than WebP, and 1.7 times smaller than BPG on the Kodak PhotoCD and RAISE-1k 512×768 datasets across all of quality levels. At the same time, we designed our approach to be lightweight and efficiently deployable. On a GTX 980 Ti GPU, it takes around 9ms to encode and 10ms to decode an image from these datasets: for JPEG, encode/decode times are 18ms/12ms, for JP2 350ms/80ms and for WebP 70ms/80ms. Results for a representative quality level are presented in Table 1.

To our knowledge, this is the first ML-based approach to surpass all commercial image compression techniques, and moreover run in real-time.

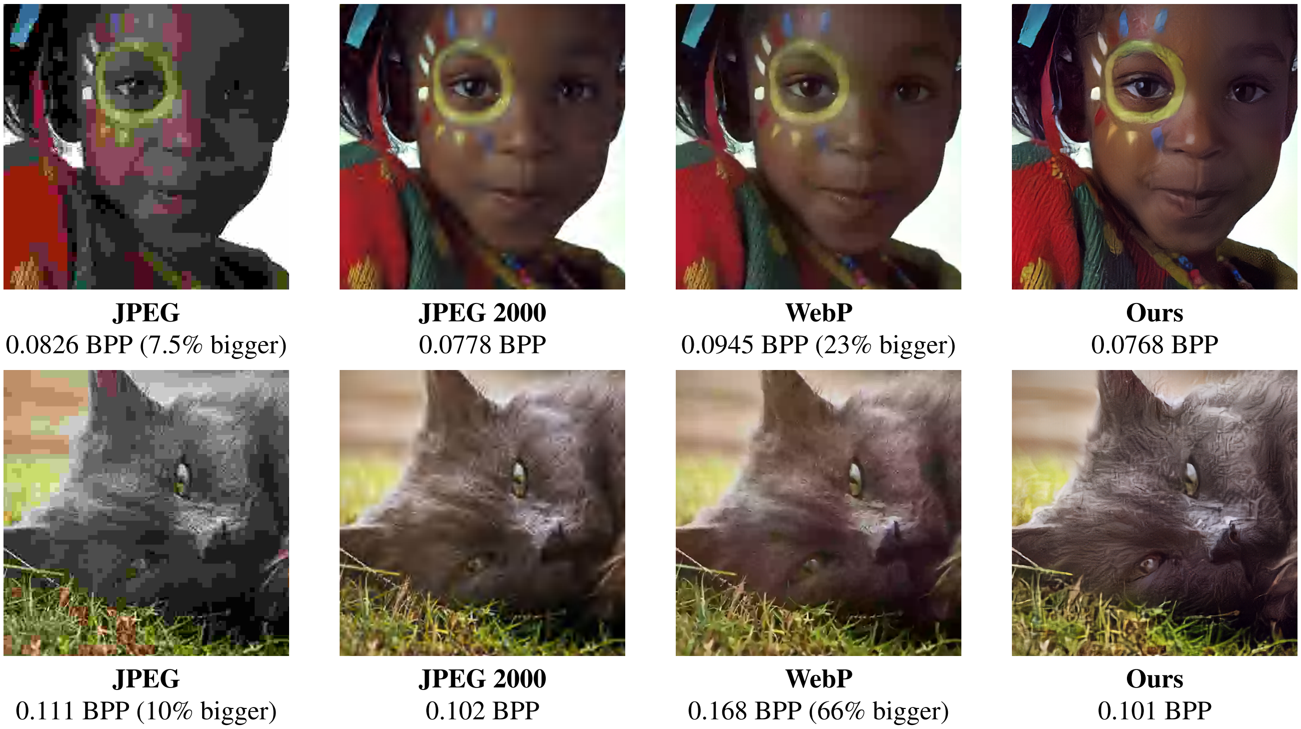
We additionally supplement our algorithm with adversarial training specialized towards use in a compression setting. This enables us to produce convincing reconstructions for very low bitrates.

Codec	RGB file size (kb)	YCbCr file size (kb)	Encode time (ms)	Decode time (ms)
Ours	21.4 (100%)	17.4 (100%)	8.6[*]	9.9[*]
JPEG	65.3 (304%)	43.6 (250%)	18.6	13.0
JP2	54.4 (254%)	43.8 (252%)	367.4	80.4
WebP	49.7 (232%)	37.6 (216%)	67.0	83.7

Table 1. Performance of different codecs on the RAISE-1k 512×768 dataset for a representative MS-SSIM value of 0.98 in both RGB and YCbCr color spaces. Comprehensive results can be found in Section 5. [*]We emphasize our codec was run on GPU.

[*]Equal contribution [1]WaveOne Inc., Mountain View, CA, USA. Correspondence to: Oren Rippel <oren@wave.one>, Lubomir Bourdev <lubomir@wave.one>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).



JPEG
0.0826 BPP (7.5% bigger)

JPEG 2000
0.0778 BPP

WebP
0.0945 BPP (23% bigger)

Ours
0.0768 BPP

JPEG
0.111 BPP (10% bigger)

JPEG 2000
0.102 BPP

WebP
0.168 BPP (66% bigger)

Ours
0.101 BPP

Figure 1. Examples of reconstructions by different codecs for very low bits per pixel (BPP) values. The uncompressed size is 24 BPP, so the examples represent compression by around 250 times. We reduce the bitrates of other codecs by their header lengths for fair comparison. For each codec, we search over bitrates and present the reconstruction for the smallest BPP above ours. WebP and JPEG were not able to produce reconstructions for such low BPP: the reconstructions presented are for the smallest bitrate they offer. More examples can be found in the appendix.

2. Background & Related Work

2.1. Traditional compression techniques

Compression, in general, is very closely related to pattern recognition. If we are able to discover structure in our input, we can eliminate this redundancy to represent it more succinctly. In traditional codecs such as JPEG and JP2, this is achieved via a pipeline which roughly breaks down into 3 modules: transformation, quantization, and encoding (Wallace (1992) and Rabbani & Joshi (2002) provide great overviews of the JPEG standards).

In traditional codecs, since all components are hard-coded, they are heavily engineered to fit together. For example, the coding scheme is custom-tailored to match the distribution of the outputs of the preceding transformation. JPEG, for instance, employs 8×8 block DCT transforms, followed by run-length encoding which exploits the sparsity pattern of the resultant frequency coefficients. JP2 employs an adaptive arithmetic coder to capture the distribution of coefficient magnitudes produced by the preceding multi-resolution wavelet transform.

However, despite the careful construction and assembly of these pipelines, there still remains significant room for improvement of compression efficiency. For example, the transformation is fixed in place irrespective of the distribution of the inputs, and is not adapted to their statistics in any way. In addition, hard-coded approaches often compartmentalize the loss of information within the quantization step. As such, the transformation module is chosen to be bijective: however, this limits the ability to reduce redundancy prior to coding. Moreover, the encode-decode pipeline cannot be optimized for a particular metric beyond manual tweaking: even if we had the perfect metric for image quality assessment, traditional approaches cannot directly optimize their reconstructions for it.

2.2. ML-based lossy image compression

In approaches based on machine learning, structure is automatically *discovered*, rather than manually engineered.

One of the first such efforts by Bottou et al. (1998), for example, introduced the DjVu format for document image compression, which employs techniques such as segmentation and K-means clustering separate foreground from background, and analyze the document's contents.

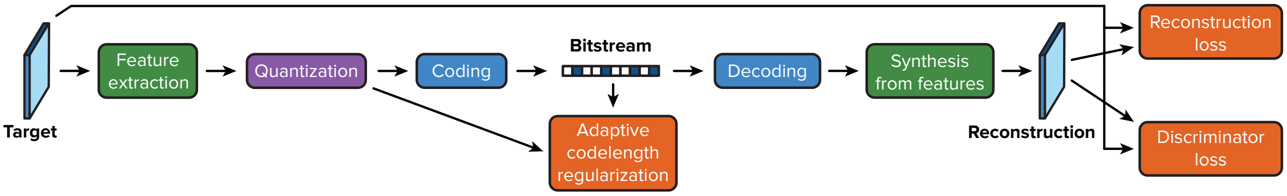


Figure 2. Our overall model architecture. The feature extractor, described in Section 3.1, discovers structure and reduces redundancy via the pyramidal decomposition and interscale alignment modules. The lossless coding scheme, described in Section 3.2, further compresses the quantized tensor via bitplane decomposition and adaptive arithmetic coding. The adaptive codelength regularization then modulates the expected code length to a prescribed target bitrate. Distortions between the target and its reconstruction are penalized by the reconstruction loss. The discriminator loss, described in Section 4, encourages visually pleasing reconstructions by penalizing discrepancies between their distributions and the targets'.

At a high level, one natural approach to implement the encoder-decoder image compression pipeline is to use an autoencoder to map the target through a bitrate bottleneck, and train the model to minimize a loss function penalizing it from its reconstruction. This requires carefully constructing a feature extractor and synthesizer for the encoder and decoder, selecting an appropriate objective, and possibly introducing a coding scheme to further compress the fixed-size representation to attain variable-length codes.

Many of the existing ML-based image compression approaches (including ours) follow this general strategy. Toderici et al. (2015; 2016) explored various transformations for binary feature extraction based on different types of recurrent neural networks; the binary representations were then entropy-coded. Johnston et al. (2017) enabled another considerable leap in performance by introducing a loss weighted with SSIM (Wang et al., 2004), and spatially-adaptive bit allocation. Theis et al. (2016) and Ballé et al. (2016) quantize rather than binarize, and propose strategies to approximate the entropy of the quantized representation: this provides them with a proxy to penalize it. Finally, Pied Piper has recently claimed to employ ML techniques in its Middle-Out algorithm (Judge et al., 2016), although their nature is shrouded in mystery.

2.3. Generative Adversarial Networks

One of the most exciting innovations in machine learning in the last few years is the idea of Generative Adversarial Networks (GANs) (Goodfellow et al., 2014). The idea is to construct a *generator* network $\mathscr{G}_{\mathbf{\Phi}}(\cdot)$ whose goal is to synthesize outputs according to a target distribution p_{true}, and a *discriminator* network $\mathscr{D}_{\mathbf{\Theta}}(\cdot)$ whose goal is to distinguish between examples sampled from the ground truth distribution, and ones produced by the generator. This can be expressed concretely in terms of the minimax problem:

$$\min_{\mathbf{\Phi}} \max_{\mathbf{\Theta}} \mathbb{E}_{\mathbf{x} \sim p_{\text{true}}} \log \mathscr{D}_{\mathbf{\Theta}}(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim p_{\mathbf{z}}} \log \left[1 - \mathscr{D}_{\mathbf{\Theta}}(\mathscr{G}_{\mathbf{\Phi}}(\mathbf{z}))\right] .$$

This idea has enabled significant progress in photo-realistic image generation (Denton et al., 2015; Radford et al., 2015; Salimans et al., 2016), single-image super-resolution (Ledig et al., 2016), image-to-image conditional translation (Isola et al., 2016), and various other important problems.

The adversarial training framework is particularly relevant to the compression world. In traditional codecs, distortions often take the form of blurriness, pixelation, and so on. These artifacts are unappealing, but are increasingly noticeable as the bitrate is lowered. We propose a multiscale adversarial training model to encourage reconstructions to match the statistics of their ground truth counterparts, resulting in sharp and visually pleasing results even for very low bitrates. As far as we know, we are the first to propose using GANs for image compression.

3. Model

Our model architecture is shown in Figure 2, and comprises a number of components which we briefly outline below. In this section, we limit our focus to operations performed by the encoder: since the decoder simply performs the counterpart inverse operations, we only address exceptions which require particular attention.

Feature extraction. Images feature a number of different types of structure: across input channels, within individual scales, and across scales. We design our feature extraction architecture to recognize these. It consists of a pyramidal decomposition which analyzes individual scales, followed by an interscale alignment procedure which exploits structure shared across scales.

Code computation and regularization. This module is responsible for further compressing the extracted features. It quantizes the features, and encodes them via an adaptive arithmetic coding scheme applied on their binary expansions. An adaptive codelength regularization is introduced to penalize the entropy of the features, which the coding scheme exploits to achieve better compression.

Discriminator loss. We employ adversarial training to pursue realistic reconstructions. We dedicate Section 4 to describing our GAN formulation.

3.1. Feature extraction

3.1.1. Pyramidal decomposition

Our pyramidal decomposition encoder is loosely inspired by the use of wavelets for multiresolution analysis, in which an input is analyzed recursively via feature extraction and downsampling operators (Mallat, 1989). The JPEG 2000 standard, for example, employs discrete wavelet transforms with the Daubechies 9/7 kernels (Antonini et al., 1992; Rabbani & Joshi, 2002). This transform is in fact a linear operator, which can be entirely expressed via compositions of convolutions with only two *hard-coded and separable* 9×9 filters applied irrespective of scale, and independently for each channel.

The idea of a pyramidal decomposition has been employed in machine learning: for instance, Mathieu et al. (2015) uses a pyramidal composition for next frame prediction, and Denton et al. (2015) uses it for image generation. The spectral representations of CNN activations have also been investigated by Rippel et al. (2015) to enable processing across a spectrum of scales, but this approach does not enable FIR processing as does wavelet analysis.

We generalize the wavelet decomposition idea to learn optimal, nonlinear extractors individually for each scale. Let us assume an input $\mathbf{x}$ to the model, and a total of M scales. We perform recursive analysis: let us denote $\mathbf{x}_m$ as the input to scale m; we set the input to the first scale $\mathbf{x}_1 = \mathbf{x}$ as the input to the model. For each scale m, we perform two operations: first, we extract coefficients $\mathbf{c}_m = \mathbf{f}_m(\mathbf{x}_m) \in \mathbb{R}^{C_m \times H_m \times W_m}$ via some parametrized function $\mathbf{f}_m(\cdot)$ for output channels C_m, height H_m and width W_m. Second, we compute the input to the next scale as $\mathbf{x}_{m+1} = \mathbf{D}_m(\mathbf{x}_m)$ where $\mathbf{D}_m(\cdot)$ is some downsampling operator (either fixed or learned).

Our pyramidal decomposition architecture is illustrated in Figure 3. In practice, we extract across a total of $M = 6$ scales. The feature extractors for the individual scales are composed of a sequence of convolutions with kernels 3×3 or 1×1 and ReLUs with a leak of 0.2. We learn all downsamplers as 4×4 convolutions with a stride of 2.

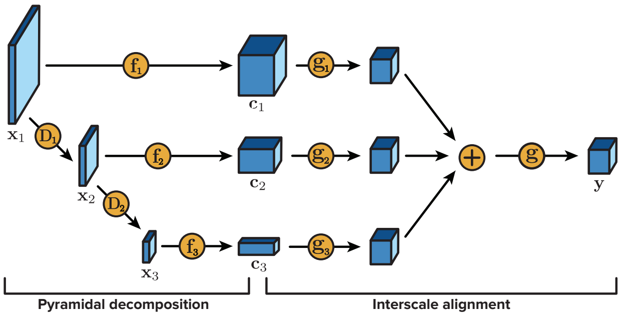


Figure 3. The coefficient extraction pipeline, illustrated for 3 scales. The pyramidal decomposition module discovers structure within individual scales. The extracted coefficient maps are then aligned to discover joint structure across the different scales.

3.1.2. Interscale alignment

Interscale alignment is designed to leverage information shared across different scales — a benefit not offered by the classic wavelet analysis. It takes in as input the set of coefficients extracted from the different scales $\{\mathbf{c}_m\}_{m=1}^{M} \subset \mathbb{R}^{C_m \times H_m \times W_m}$, and produces a single tensor of a target output dimensionality $C \times H \times W$.

To do this, we first map each input tensor $\mathbf{c}_m$ to the target dimensionality via some parametrized function $\mathbf{g}_m(\cdot)$. This involves ensuring that this function spatially resamples $\mathbf{c}_m$ to the appropriate output map size $H \times W$, and outputs the appropriate number of channels C. We then sum $\mathbf{g}_m(\mathbf{c}_m), m = 1, \ldots, M$, and apply another parametrized non-linear transformation $\mathbf{g}(\cdot)$ for joint processing.

The interscale alignment module can be seen in Figure 3. We denote its output as $\mathbf{y}$. In practice, we choose each $\mathbf{g}_m(\cdot)$ as a convolution or a deconvolution with an appropriate stride to produce the target spatial map size $H \times W$; see Section 5.1 for a more detailed discussion. We choose $\mathbf{g}(\cdot)$ simply as a sequence of 3×3 convolutions.

3.2. Code computation and regularization

Given the extracted tensor $\mathbf{y} \in \mathbb{R}^{C \times H \times W}$, we proceed to quantize it and encode it. This pipeline involves a number of components which we overview here and describe in detail throughout this section.

Quantization. The tensor $\mathbf{y}$ is quantized to bit precision B:

$$\hat{\mathbf{y}} := \textsc{Quantize}_B(\mathbf{y}) \; .$$

Bitplane decomposition. The quantized tensor $\hat{\mathbf{y}}$ is transformed into a binary tensor suitable for encoding via a lossless bitplane decomposition:

$$\mathbf{b} := \textsc{BitplaneDecompose}_B(\hat{\mathbf{y}}) \in \{0,1\}^{B \times C \times H \times W} \; .$$

Adaptive arithmetic coding. The adaptive arithmetic coder (AAC) is trained to leverage the structure remaining in the data. It encodes $\mathbf{b}$ into its final variable-length binary sequence $\mathbf{s}$ of length $\ell(\mathbf{s})$:

$$\mathbf{s} := \textsc{AACEncode}(\mathbf{b}) \in \{0,1\}^{\ell(\mathbf{s})} \; .$$

Adaptive codelength regularization. The adaptive codelength regularization (ACR) modulates the distribution of the quantized representation $\hat{\mathbf{y}}$ to achieve a target expected bit count across inputs:

$$\mathbb{E}_{\mathbf{x}}[\ell(\mathbf{s})] \quad \longrightarrow \quad \ell_{\text{target}} \ .$$

3.2.1. Quantization

Given a desired precision of B bits, we quantize our feature tensor $\mathbf{y}$ into 2^B equal-sized bins as

$$\hat{y}_{chw} := \textsc{Quantize}_B(y_{chw}) = \frac{1}{2^{B-1}} \left\lceil 2^{B-1} y_{chw} \right\rceil \ .$$

For the special case $B = 1$, this reduces exactly to a binary quantization scheme. While some ML-based approaches to compression employ such thresholding, we found better performance with the smoother quantization described. We quantize with $B = 6$ for all models in this paper.

3.2.2. Bitplane decomposition

We decompose $\hat{\mathbf{y}}$ into bitplanes. This transformation maps each value $\hat{y}_{chw}$ into its binary expansion of B bits. Hence, each of the C spatial maps $\hat{\mathbf{y}}_c \in \mathbb{R}^{H\times W}$ of $\hat{\mathbf{y}}$ expands into B binary *bitplanes*. We illustrate this transformation in Figure 4, and denote its output as $\mathbf{b} \in \{0,1\}^{B\times C\times H\times W}$. This transformation is lossless.

As described in Section 3.2.3, this decomposition will enable our entropy coder to exploit structure in the distribution of the activations in $\mathbf{y}$ to achieve a compact representation. In Section 3.2.4, we introduce a strategy to encourage such exploitable structure to be featured.

3.2.3. Adaptive arithmetic coding

The output $\mathbf{b}$ of the bitplane decomposition is a binary tensor, which contains significant structure: for example, higher bitplanes are sparser, and spatially neighboring bits often have the same value (in Section 3.2.4 we propose a technique to guarantee presence of these properties). We exploit this low entropy by lossless compression via adaptive arithmetic coding.

Namely, we associate each bit location in $\mathbf{b}$ with a *context*, which comprises a set of features indicative of the bit value. These are based on the position of the bit as well as the values of neighboring bits. We train a classifier to predict the value of each bit from its context features, and then use these probabilities to compress $\mathbf{b}$ via arithmetic coding.

During decoding, we decompress the code by performing the inverse operation. Namely, we interleave between computing the context of a particular bit using the values of previously decoded bits, and using this context to retrieve the activation probability of the bit and decode it. We note that this constrains the context of each bit to only include features composed of bits already decoded.

3.2.4. Adaptive codelength regularization

One problem with classic autoencoder architectures is that their bottleneck has fixed capacity. The bottleneck may be too small to represent complex patterns well, which affects quality, and it may be too large for simple patterns, which results in inefficient compression. What we need is a model capable of generating long representations for complex patterns and short for simple ones, while maintaining an expected codelength target over large number of examples. To achieve this, the AAC is necessary, but not sufficient.

We extend the architecture by increasing the dimensionality of $\mathbf{b}$ — but at the same time controlling its information content, thereby resulting in shorter compressed code $\mathbf{s} = \textsc{AACEncode}(\mathbf{b}) \in \{0,1\}$. Specifically, we introduce the adaptive codelength regularization (ACR), which enables us to regulate the expected codelength $\mathbb{E}_{\mathbf{x}}[\ell(\mathbf{s})]$ to a target value ℓ_{target}. This penalty is designed to encourage structure exactly where the AAC is able to exploit it. Namely, we regularize our quantized tensor $\hat{\mathbf{y}}$ with

$$\mathscr{P}(\hat{\mathbf{y}}) = \frac{\alpha_t}{CHW} \sum_{chw} \Big\{ \log_2 |\hat{y}_{chw}| + \sum_{(x,y)\in S} \log_2 \left|\hat{y}_{chw} - \hat{y}_{c(h-y)(w-x)}\right| \Big\} ,$$

for iteration t and difference index set $S = \{(0,1),(1,0),(1,1),(-1,1)\}$. The first term penalizes the magnitude of each tensor element, and the second penalizes deviations between spatial neighbors. These enable better prediction by the AAC.

As we train our model, we continuously modulate the scalar coefficient α_t to pursue our target codelength. We do this via a feedback loop. We use the AAC to monitor the mean number of effective bits. If it is too high, we increase α_t; if too low, we decrease it. In practice, the model reaches an equilibrium in a few hundred iterations, and is able to maintain it throughout training.

Hence, we get a knob to tune: the ratio of total bits, namely the $BCHW$ bits available in $\mathbf{b}$, to the target number of effective bits ℓ_{target}. This allows exploring the trade-off of increasing the number of channels or spatial map size of

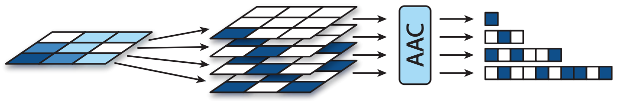


Figure 4. Each of the C spatial maps $\hat{\mathbf{y}}_c \in \mathbb{R}^{H\times W}$ of $\hat{\mathbf{y}}$ is decomposed into B bitplanes as each element $\hat{y}_{chw}$ is expressed in its binary expansion. Each set of bitplanes is then fed to the adaptive arithmetic coder for variable-length encoding. The adaptive codelength regularization enables more compact codes for higher bitplanes by encouraging them to feature higher sparsity.

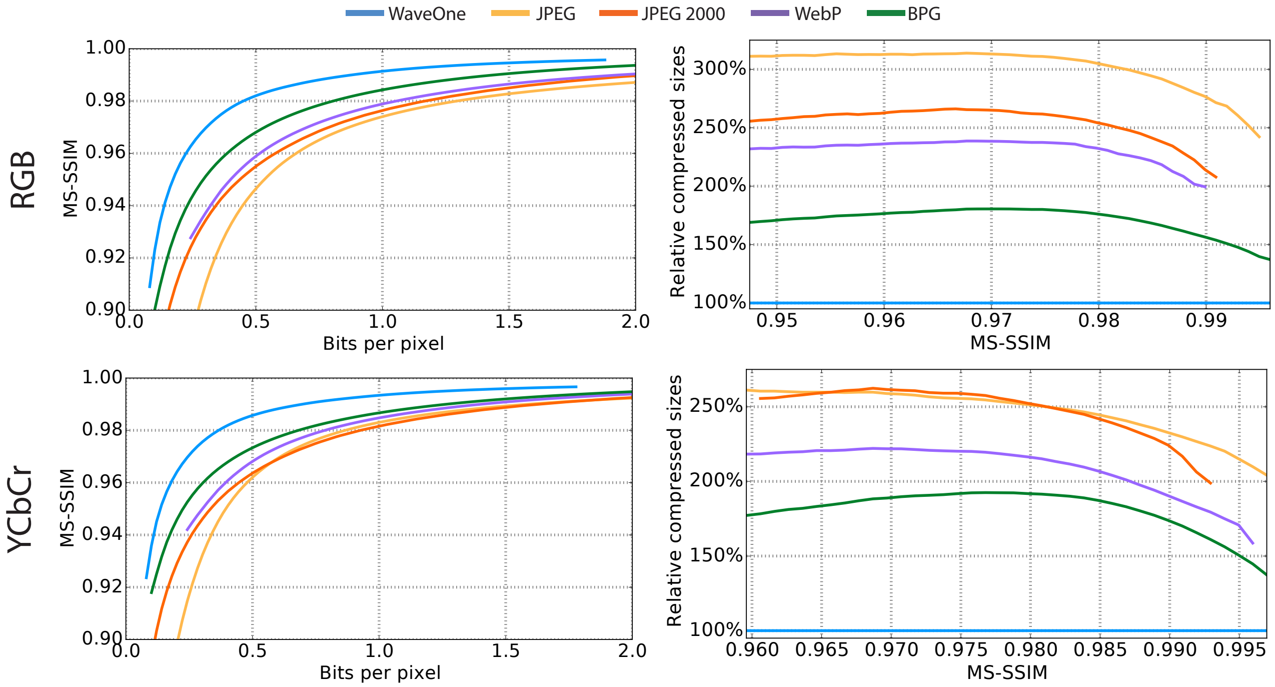


Figure 5. Compression results for the RAISE-1k 512×768 dataset, measured in the RGB domain (top row) and YCbCr domain (bottom row). We compare against commercial codecs JPEG, JPEG 2000, WebP and BPG[5] (4:2:0 for YCbCr and 4:4:4 for RGB). The plots on the left present average reconstruction quality, as function of the number of bits per pixel fixed for each image. The plots on the right show average compressed file sizes relative to ours for different target MS-SSIM values for each image. In Section 5.2 we discuss the curve generation procedures in detail.

b at the cost of increasing sparsity. We find that a total-to-target ratio of $BCHW/\ell_{\text{target}} = 4$ works well across all architectures we have explored.

4. Realistic Reconstructions via Multiscale Adversarial Training

4.1. Discriminator design

In our compression approach, we take the generator as the encoder-decoder pipeline, to which we append a discriminator — albeit with a few key differences from existing GAN formulations.

In many GAN approaches featuring both a reconstruction and a discrimination loss, the target and the reconstruction are treated independently: each is separately assigned a label indicating whether it is real or fake. In our formulation, we consider the target and its reconstruction jointly as a single example: we compare the two by asking *which* of the two images is the real one.

To do this, we first swap between the target and reconstruction in each input pair to the discriminator with uniform probability. Following the random swap, we propagate each set of examples through the network. However, instead of producing an output for classification at the very last layer of the pipeline, we accumulate scalar outputs along branches constructed along it at different depths. We average these to attain the final value provided to the terminal sigmoid function. This multiscale architecture allows aggregating information across different scales, and is motivated by the observation that undesirable artifacts vary as function of the scale in which they are exhibited. For example, high-frequency artifacts such as noise and blurriness are discovered by earlier scales, whereas more abstract discrepancies are found in deeper scales.

We apply our discriminator $\mathscr{D}_{\Theta}$ on the aggregate sum across scales, and proceed to formulate our objectives as described in Section 2.3. The complete discriminator architecture is illustrated in the appendix.

4.2. Adversarial training

Training a GAN system can be tricky due to optimization instability. In our case, we were able to address this by designing a training scheme adaptive in two ways. First, the reconstructor is trained by both the confusion signal gradient as well as the reconstruction loss gradient: we balance the two as function of their gradient magnitudes. Second, at any point during training, we either train the discriminator or propagate confusion signal through the reconstructor, as function of the prediction accuracy of the discriminator.

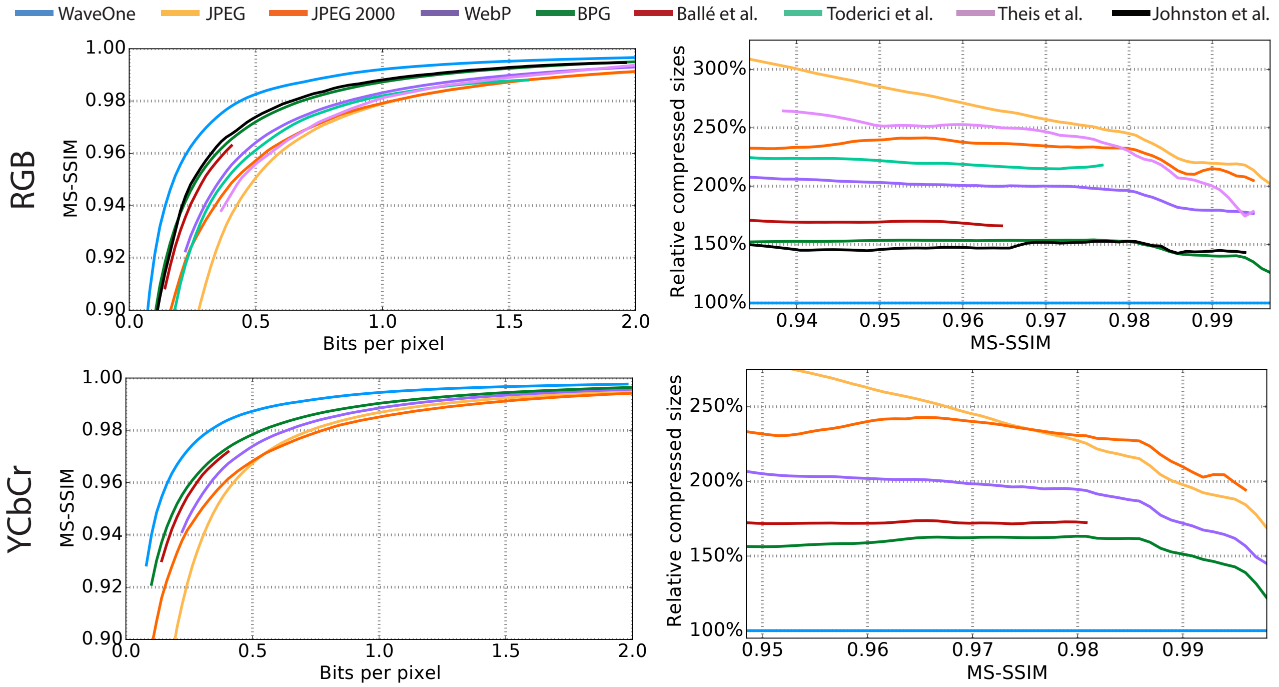


Figure 6. Performance on the Kodak PhotoCD dataset measured in the RGB domain (top row) and YCbCr domain (bottom row). We compare against commercial codecs JPEG, JPEG 2000, WebP and BPG[5] (4:2:0 for YCbCr and 4:4:4 for RGB), as well as recent ML-based compression work by Toderici et al. (2016)[2], Theis et al. (2016)[3], Ballé et al. (2016)[4], and Johnston et al. (2017)[3] in all settings where results exist. The plots on the left present average reconstruction quality, as function of the number of bits per pixel fixed for each image. The plots on the right show average compressed file sizes relative to ours for different target MS-SSIM values for each image.

More concretely, given lower and upper accuracy bounds $L, U \in [0, 1]$ and discriminator accuracy $a(\mathscr{D}_\Theta)$, we apply the following procedure:

- If $a < L$: freeze propagation of confusion signal through the reconstructor, and train the discriminator.
- If $L \leq a < U$: alternate between propagating confusion signal and training the disciminator.
- If $U \leq a$: propagate confusion signal through the reconstructor, and freeze the discriminator.

In practice we used $L = 0.8, U = 0.95$. We compute the accuracy a as a running average over mini-batches with a momentum of 0.8.

5. Results

5.1. Experimental setup

Similarity metric. We trained and tested all models on the Multi-Scale Structural Similarity Index Metric (MS-SSIM) (Wang et al., 2003). This metric has been specifically designed to match the human visual system, and has been established to be significantly more representative than losses in the ℓ_p family and variants such as PSNR.

Color space. Since the human visual system is much more sensitive to variations in brightness than color, most codecs represent colors in the YCbCr color space to devote more bandwidth towards encoding luma rather than chroma. In quantifying image similarity, then, it is common to assign the Y, Cb, Cr components weights $6/8, 1/8, 1/8$. While many ML-based compression papers evaluate similarity in the RGB space with equal color weights, this does not allow fair comparison with standard codecs such as JPEG, JPEG 2000 and WebP, since they have not been designed to perform optimally in this domain. In this work, we provide comparisons with both traditional and ML-based codecs, and present results in both the RGB domain with equal color weights, as well as in YCbCr with weights as above.

Reported performance metrics. We present both compression performance of our algorithm, but also its runtime. While the requirement of running the approach in real-time severely constrains the capacity of the model, it must be met to enable feasible deployment in real-life applications.

Training and deployment procedure. We trained and tested all models on a GeForce GTX 980 Ti GPU and a custom codebase. We trained all models on 128×128 patches sampled at random from the Yahoo Flickr Creative Commons 100 Million dataset (Thomee et al., 2016).

We optimized all models with Adam (Kingma & Ba, 2014). We used an initial learning rate of 3×10^{-4}, and reduced it twice by a factor of 5 during training. We chose a batch size of 16 and trained each model for a total of 400,000 iterations. We initialized the ACR coefficient as $\alpha_0 = 1$. During runtime we deployed the model on arbitrarily-sized images in a fully-convolutional way. To attain the rate-distortion (RD)curves presented in Section 5.2, we trained models for a range of target bitrates ℓ_{target}.

5.2. Performance

We present several types of results:

1. Average MS-SSIM as function of the BPP fixed for each image, found in Figures 5 and 6, and Table 1.

2. Average compressed file sizes relative to ours as function of the MS-SSIM fixed for each image, found in Figures 5 and 6, and Table 1.

3. Encode and decode timings as function of MS-SSIM, found in Figure 7, in the appendix, and Table 1.

4. Visual examples of reconstructions of different compression approaches for the same BPP, found in Figure 1 and in the appendix.

Test sets. To enable comparison with other approaches, we first present performance on the Kodak PhotoCD dataset[1]. While the Kodak dataset is very popular for testing compression performance, it contains only 24 images, and hence is susceptible to overfitting and does not necessarily fully capture broader statistics of natural images. As such, we additionally present performance on the RAISE-1k dataset (Dang-Nguyen et al., 2015) which contains 1,000 raw images. We resized each image to size 512×768 (backwards if vertical): we intend to release our preparation code to enable reproduction of the dataset used.

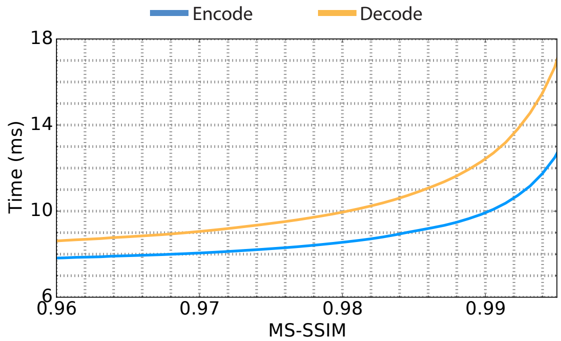


Figure 7. Average times to encode and decode images from the RAISE-1k 512×768 dataset using our approach.

We remark it is important to use a dataset of raw, rather than previously compressed, images for codec evaluation. Compressing an image introduces artifacts with a bias particular to the codec used, which results in a more favorable RD curve if it compressed *again* with the same codec. See the appendix for a plot demonstrating this effect.

Codecs. We compare against commercial compression techniques JPEG, JPEG 2000, WebP, as well as recent ML-based compression work by Toderici et al. (2016)[2], Theis et al. (2016)[3], Ballé et al. (2016)[4], and Johnston et al. (2017)[3] in all settings in which results are available. We also compare to BPG[5] (4:2:0 and 4:4:4) which, while not widely used, surpassed all other codecs in the past. We use the best-performing configuration we can find of JPEG, JPEG 2000, WebP, and BPG, and reduce their bitrates by their respective header lengths for fair comparison.

Performance evaluation. For each image in each test set, each compression approach, each color space, and for the selection of available compression rates, we recorded (1) the BPP, (2) the MS-SSIM (with components weighted appropriately for the color space), and (3) the computation times for encoding and decoding.

It is important to take great care in the design of the performance evaluation procedure. Each image has a separate RD curve computed from all available compression rates for a given codec: as Ballé et al. (2016) discusses in detail, different summaries of these RD curves lead to disparate results. In our evaluations, to compute a given curve, we sweep across values of the independent variable (such as bitrate). We interpolate each individual RD curve at this independent variable value, and average all the results. To ensure accurate interpolation, we sample densely across rates for each codec.

Acknowledgements We are grateful to Trevor Darrell, Sven Strohband, Michael Gelbart, Robert Nishihara, Albert Azout, and Vinod Khosla for meaningful discussions and input.

[1] The Kodak PhotoCD dataset can be found at http://r0k.us/graphics/kodak. We do not crop or process the images in any way.

[2] The results of Toderici et al. (2016) on the Kodak RGB dataset are available at http://github.com/tensorflow/models/tree/master/compression.

[3] We have no access to reconstructions by Theis et al. (2016) and Johnston et al. (2017), so we carefully transcribed their results, only available in RGB, from the graphs in their paper.

[4] Reconstructions by Ballé et al. (2016) of images in the Kodak dataset can be found at http://www.cns.nyu.edu/~lcv/iclr2017/ for both RGB and YCbCr and across a spectrum of BPPs. We use these to compute RD curves by the procedure described in this section.

[5] An implementation of the BPG codec is available at http://bellard.org/bpg.

References

White paper: Cisco vni forecast and methodology, 2015-2020. 2015.

Antonini, Marc, Barlaud, Michel, Mathieu, Pierre, and Daubechies, Ingrid. Image coding using wavelet transform. *IEEE Trans. Image Processing*, 1992.

Ballé, Johannes, Laparra, Valero, and Simoncelli, Eero P. End-to-end optimized image compression. *preprint*, 2016.

Bottou, Léon, Haffner, Patrick, Howard, Paul G, Simard, Patrice, Bengio, Yoshua, and LeCun, Yann. High quality document image compression with djvu. 1998.

Dang-Nguyen, Duc-Tien, Pasquini, Cecilia, Conotter, Valentina, and Boato, Giulia. Raise: a raw images dataset for digital image forensics. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pp. 219–224. ACM, 2015.

Denton, Emily L, Chintala, Soumith, Fergus, Rob, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *NIPS*, pp. 1486–1494, 2015.

Goodfellow, Ian, Pouget-Abadie, Jean, Mirza, Mehdi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron, and Bengio, Yoshua. Generative adversarial nets. In *NIPS*, pp. 2672–2680, 2014.

Isola, Phillip, Zhu, Jun-Yan, Zhou, Tinghui, and Efros, Alexei A. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

Johnston, Nick, Vincent, Damien, Minnen, David, Covell, Michele, Singh, Saurabh, Chinen, Troy, Hwang, Sung Jin, Shor, Joel, and Toderici, George. Improved lossy image compression with priming and spatially adaptive bit rates for recurrent networks. *arXiv preprint arXiv:1703.10114*, 2017.

Judge, Mike, Altschuler, John, and Krinsky, Dave. Silicon valley (tv series). 2016.

Kingma, Diederik and Ba, Jimmy. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Ledig, Christian, Theis, Lucas, Huszár, Ferenc, Caballero, Jose, Cunningham, Andrew, Acosta, Alejandro, Aitken, Andrew, Tejani, Alykhan, Totz, Johannes, Wang, Zehan, et al. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

Mallat, S. G. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(7):674–693, July 1989.

Mathieu, Michael, Couprie, Camille, and LeCun, Yann. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

Rabbani, Majid and Joshi, Rajan. An overview of the jpeg 2000 still image compression standard. *Signal processing: Image communication*, 17(1):3–48, 2002.

Radford, Alec, Metz, Luke, and Chintala, Soumith. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Rippel, Oren, Snoek, Jasper, and Adams, Ryan P. Spectral representations for convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 2449–2457, 2015.

Salimans, Tim, Goodfellow, Ian, Zaremba, Wojciech, Cheung, Vicki, Radford, Alec, and Chen, Xi. Improved techniques for training gans. In *NIPS*, pp. 2226–2234, 2016.

Theis, Lucas, Shi, Wenzhe, Cunningham, Andrew, and Huszar, Ferenc. Lossy image compression with compressive autoencoders. *preprint*, 2016.

Thomee, Bart, Shamma, David A, Friedland, Gerald, Elizalde, Benjamin, Ni, Karl, Poland, Douglas, Borth, Damian, and Li, Li-Jia. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 2016.

Toderici, George, O'Malley, Sean M, Hwang, Sung Jin, Vincent, Damien, Minnen, David, Baluja, Shumeet, Covell, Michele, and Sukthankar, Rahul. Variable rate image compression with recurrent neural networks. *arXiv preprint arXiv:1511.06085*, 2015.

Toderici, George, Vincent, Damien, Johnston, Nick, Hwang, Sung Jin, Minnen, David, Shor, Joel, and Covell, Michele. Full resolution image compression with recurrent neural networks. *arXiv preprint arXiv:1608.05148*, 2016.

Wallace, Gregory K. The jpeg still picture compression standard. *IEEE transactions on consumer electronics*, 38(1):xviii–xxxiv, 1992.

Wang, Zhou, Simoncelli, Eero P, and Bovik, Alan C. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004.*, volume 2, pp. 1398–1402. Ieee, 2003.

Wang, Zhou, Bovik, Alan C, Sheikh, Hamid R, and Simoncelli, Eero P. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.