Improved Lossy Image Compression with Priming and Spatially Adaptive Bit Rates for Recurrent Networks

Nick Johnston, Damien Vincent, David Minnen, Michele Covell, Saurabh Singh, Troy Chinen,
Sung Jin Hwang, Joel Shor, George Toderici

{nickj, damienv, dminnen, covell, saurabhsingh, tchinen, sjhwang, joelshor, gtoderici}
@google.com, Google Research

Abstract

We propose a method for lossy image compression based on recurrent, convolutional neural networks that outperforms BPG (4:2:0), WebP, JPEG2000, and JPEG as measured by MS-SSIM. We introduce three improvements over previous research that lead to this state-of-the-art result using a single model. First, we modify the recurrent architecture to improve spatial diffusion, which allows the network to more effectively capture and propagate image information through the network's hidden state. Second, in addition to lossless entropy coding, we use a spatially adaptive bit allocation algorithm to more efficiently use the limited number of bits to encode visually complex image regions. Finally, we show that training with a pixel-wise loss weighted by SSIM increases reconstruction quality according to multiple metrics. We evaluate our method on the Kodak and Tecnick image sets and compare against standard codecs as well as recently published methods based on deep neural networks.

1. Introduction

Previous research showed that deep neural networks can be effectively applied to the problem of lossy image compression [21, 22, 23, 10, 17, 4, 19]. Those methods extend the basic autoencoder structure and generate a binary representation for an image by quantizing either the bottleneck layer or the corresponding latent variables. Several options have been explored for encoding images at different bit rates including training multiple models [4], learning quantization-scaling parameters [21], and transmitting a subset of the encoded representation within a recurrent structure [10, 23].

Our method takes the recurrent approach and builds on the architecture introduced by [23]. The model uses a recurrent autoencoder where each iteration encodes the residual between the previous reconstruction and the original image (see Figure 1). At each step, the network extracts new information from the current residual and combines it with context stored in the hidden state of the recurrent layers. By saving the bits from the quantized bottleneck after each iteration, the model generates a progressive encoding of the input image.

Our method provides a significant increase in compression performance over previous models due to three improvements. First, by "priming" the network, that is, running several iterations before generating the binary codes (in the encoder) or a reconstructed image (in the decoder), we expand the spatial context, which allows the network to represent more complex representations in early iterations. Second, we add support for spatially adaptive bit rates (SABR), which dynamically adjusts the bit rate across each image depending on the complexity of the local image content. Finally, we train our model with a more sophisticated loss function that guides the pixel-wise loss using structural similarity (SSIM) [26, 28]. Combining three techniques yields a rate-distortion (RD) curve that exceeds state-of-the-art codecs (BPG 444 (YCbCr 4:4:4) [5], BPG 420 (YCbCr 4:2:0), WebP [9], JPEG2000 [12], and JPEG [25]) as well as other learned models based on deep neural networks ([21] and [23]), as measured by MS-SSIM [27].

We review previous work in Section 2 and describe our method in detail in Section 3. The description focuses on the network architecture (Section 3.1), how we combine that with hidden-state priming and diffusion (Section 3.2), and how we use spatially adaptive bit rates (Section 3.3). Section 3 also covers our training loss (Section 3.4), which provides better generalization results than unweighted L_1 or L_2 loss. In Section 4 we discuss the training setup used for our networks. Section 5 summarizes the results and compares them to existing codecs and to other recent research in neural-network-based compression [21, 19].

2. Related Work

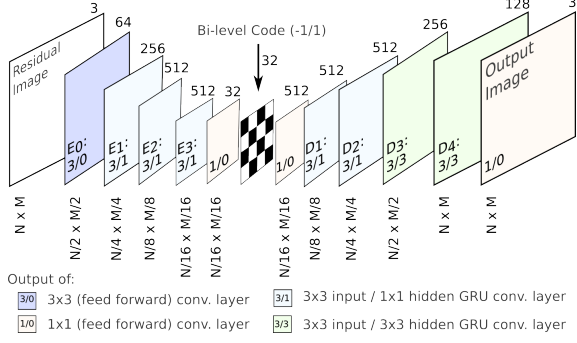Lossy image compression is a long-standing problem with many standard codecs. JPEG [25] remains the

Figure 1. The layers in our compression network, showing the encoder (E_i), binarizer (checkerboard), and decoder (D_j). Each layer is labeled with its relative resolution (below) and depth (above). The inner label ("I / H") represents the size of the convolutional kernels used for input (I) and for the hidden state (H).

most widely used method for lossy compression of digital photographs [7] while several more sophisticated standards have gained in popularity including JPEG2000 [12], WebP [9], and Better Portable Graphics (BPG) [5]. To our knowledge, BPG currently has the highest coding efficiency for lossy image compression amongst public codecs.

Recently, there has been a surge in research applying neural networks to the problem of image compression [21, 22, 23, 10, 4, 20, 2, 19]. While such methods were explored since at least the late 1980s [18, 13], few neural-network-based systems improve upon JPEG or match the coding efficiency of JPEG2000.

Autoencoders with a bottleneck have been used to learn compact representations for many applications [11, 16, 24] and form the basis for most network-based compression models. Theis *et al.* used an ensemble of encoders and target multiple bit rates by learning a scaling parameter that changes the effective quantization granularity. Ballé *et al.* use a similar architecture but use a form of local gain control called generalized divisive normalization [3] and replace the non-differentiable quantization step with continuous relaxation by adding uniform noise [4]. Rippel *et al.* achieve impressive performance by directly training for the target metric (MS-SSIM). In addition they use an ensemble of multi-scale models, an adaptive coding module, and adversarial loss.

A different method for targeting multiple bit rates uses recurrent autoencoders [10, 22, 23]. In this approach, a single model generates a progressive encoding that grows with the number of recurrent iterations. Different bit rates are achieved by transmitting only a subset (prefix) of the progressive code. Gregor *et al.* use a generative model so missing codes are replaced by sampling from the learned distribution [10]. Our model uses a recurrent architecture similar to Toderici *et al.* where missing codes are ignored [23]. The decoder thus runs fewer iterations for low bit rate encodings and will generate a valid, but less accurate, reconstruction compared to high bit rate encodings.
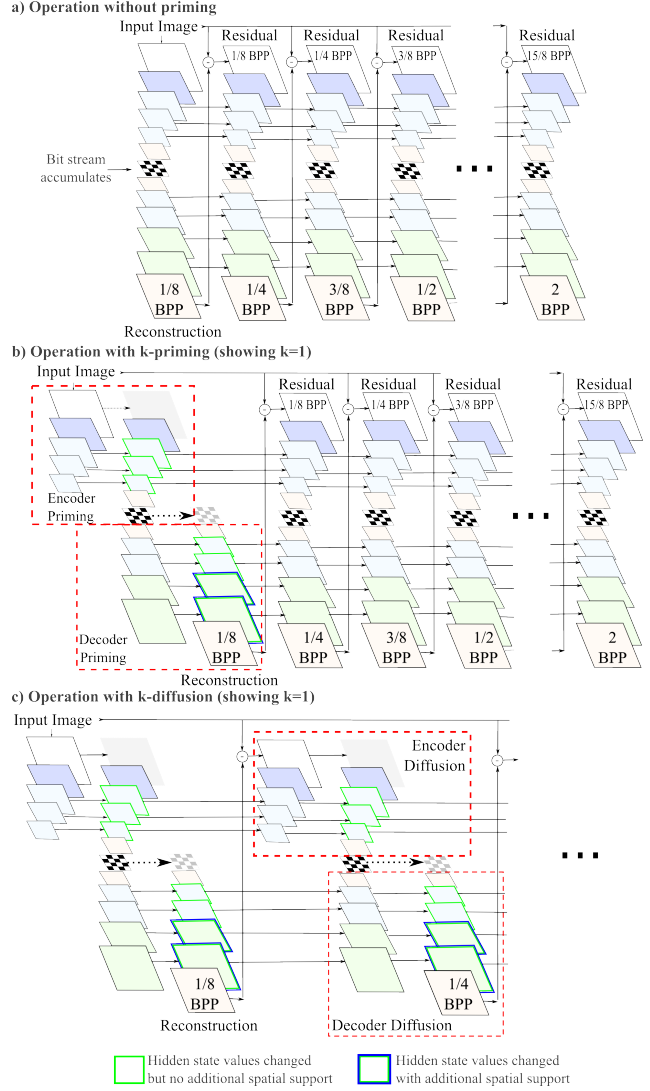

Figure 2. Network operation: (a) without priming, (b) with priming, and (c) with diffusion.

3. Methods

In this section, we first describe the network architecture used in our research along with an analysis of its spatial support. We then describe each of the three techniques that we leverage to achieve our results: hidden-state priming, spatially adaptive bit rates, and a perceptually-weighted training loss.

3.1. Network Architecture

Figure 1 shows the architecture used for our encoder and decoder networks. The depth of each layer is marked above the back corner of each plane. The name and type of layer is depicted as "$E_i : I/H$" for the encoder (and "$D_j : I/H$" for the decoder) inside the bottom of each plane. The con-

volutional kernels for input have size $I \times I$ and the convolutional kernels for the hidden state are $H \times H$. Feed-forward, non-recurrent layers have $H = 0$. The input to the encoder is the residual image: the difference between the original image and previous iteration's reconstruction. For the first iteration, this residual is simply the original image.

The first and last layers on both the encoder and decoder networks use feed-forward convolutional units ($H = 0$) with tanh activations. The other layers contain convolutional Gated Recurrent Units (GRU) [8].

To ensure accurate bit-rate counts, the binarizer (shown as a checkerboard in Figure 1) quantizes its input to be ± 1 [22]. This will give us our nominal (pre-entropy coding) bit rate. Given our choice of downsampling rates and binarizer depths, each iteration adds $\frac{1}{8}$ bpp to the previous nominal bit rate.

The spatial context used by each reconstruction pixel, as a function of either the "bit stacks" (that is, the outputs of the binarizer at a single spatial position) or the original image pixels, can be computed by examining the combined spatial supports of the encoder, the decoder, and all state vectors.[1] The dependence of the output reconstruction on the bit stacks varies by output-pixel position by one bit stack (in each spatial dimension), so we will discuss only the maximum spatial support:

$$\max(\mathcal{S}_B(F_t)) = 6t + 6 \tag{1}$$
$$\mathcal{S}_I(F_t) = 16\mathcal{S}_B(F_t) + 15 \tag{2}$$

where $\mathcal{S}_B(F_t) \times \mathcal{S}_B(F_t)$ and $\mathcal{S}_I(F_t) \times \mathcal{S}_I(F_t)$ are the spatial support of the reconstruction on the bit stacks and on the original image pixels, respectively.

3.2. Hidden-state Priming

On the first iteration of our compression networks, the hidden states of each GRU layer are initialized to zero (Figure 2-a). In our experiments, we have seen a strong visual improvement in image quality across the first several iterations. Our hypothesis is that not having a good hidden-state initialization degrades our early bit-rate performance. Since both encoder and decoder architectures stack several GRU layers sequentially, it takes several iterations for the hidden-state improvement from the first GRU layer to be observable at the binarizer (for the encoder) or in the reconstruction (for the decoder). Our approach to tackling this problem is to generate a better initial hidden-state for each layer with a technique called hidden-state priming.

Hidden-state priming, or "k-priming", increases the recurrent-depth of the first iteration of the encoder and decoder networks, separately, by an additional k steps (Fig-

---

[1]Detailed derivations of these equations, as well as the ones for the priming and diffusion supports, are given in the Supplementary material. This is additionally available at https://storage.googleapis.com/compression-ml/prime_sabr_results/supplemental_2018.pdf
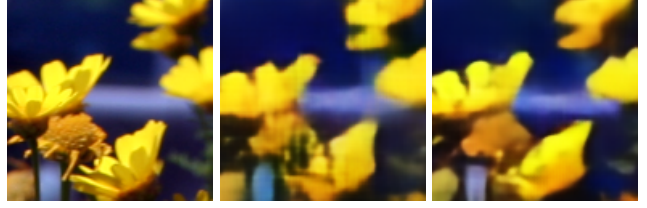

Figure 3. Left: Crop of the original Tecnick image 98. Center: Reconstruction using the DSSIM network at 0.125 bpp. Right: Reconstruction using the Prime network at 0.125 bpp. Notice the reduced artifacts from right versus center, especially in the sunflower in the lower left corner. Best viewed with zoom.
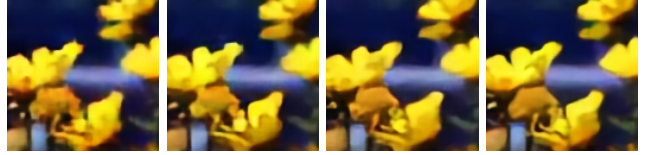

Figure 4. Cropped reconstructions of Tecnick image 98, taken at 0.25 bpp. From left to right, the results are from networks with no diffusion (0-diffusion) up to 3-diffusion. Notice the increased petal definition as more diffusion is used. Best viewed with zoom.

ure 2-b). To avoid using additional bandwidth, we run these additional steps separately, without adding the extra bits produced by the encoder to the actual bit stream. For the encoder, this means processing the original image k times, while discarding the generated bits but keeping the changes to the hidden state within the encoder's recurrent units. For the decoder, this means taking the first valid set of bits transmitted and generating a decoded image multiple times but only keeping the final image reconstruction (and the changes to the decoder's hidden states). Figure 3 depicts an example patch of an image from our evaluation set and the reconstructions from our networks trained with and without priming. The reconstruction with priming is both visually and quantitatively better than without priming, without using any additional bits.

Priming can be performed between iterations as well. When k steps are added in between each emitting iteration, we call this "k-diffusion" (Figure 2-c). Diffusion has experimentally shown better results (Figure 4), but at the cost of runtime and training time. As we increase k, we both increase the maximum support of the system along with computation and training time.

For example, in a 16 iteration network with "k-priming", k iterations of the encoder would take place before we generate our first set of bits, expanding the number of steps of computation from 16 to $16 + k$. This is done similarly on the decoder. In a 16 iteration network with "k-diffusion", the k iterations of the encoder would happen between every generation of bits, increasing computation from 16 steps to $16 \times k$ steps. So instead of taking the output at $O(i)$ we take use the output at $O(k \times i)$

In addition to achieving a better hidden-state representa-

tion for our networks, priming and diffusion also increase the spatial extent of the hidden-states, in the decoder, where the last two layers of the hidden kernels are 3×3, and in later iterations of the encoder, when the increased decoder support propagates to increased encoder support. This changes $\max(\mathcal{S}_B(F_t))$ from Equation 1 to

$$\max(\mathcal{S}_B(F_t)) = \lceil 1.5k_d + 5.5 \rceil\, t + \lceil 1.5k_p + 5.5 \rceil$$

with $k_p = k_d$ when $k_d > 0$.

3.3. Spatially Adaptive Bit Rates

By construction, our recurrent models generate image representations at different bit rates according to the number of iterations used, but those bit rates are constant across each image. That means that the local (nominal) bit rate is fixed regardless of the complexity of the underlying image content, which is inefficient in terms of quantitative and perceptual quality (e.g., consider the number of bits needed to accurately encode a clear sky compared to a flower bed).

In practice, the entropy coder introduces some spatial adaptivity based on the complexity and predictability of the binary codes, but our training procedure does not directly encourage the encoder to generate low-entropy codes. Instead, the loss function only pushes the network to maximize reconstruction quality over image patches. In order to maximize quality across a full image for a target (average) bit rate, we introduce a spatially adaptive bit rate (SABR) post-process to dynamically adjust the local bit rate according to a target reconstruction quality.

The results presented in Section 5 use a very simple bit allocation algorithm, though a more sophisticated method can be easily substituted. Given a target quality, each image tile is reconstructed using as many iterations as necessary to meet the target. As shown in Figure 1, each spatial location in the code tensor corresponds to a 16×16 tile of pixels in the original image. We calculate the per-tile quality by first dividing the image into a grid of 8×8 blocks and computing the mean L_1 error for each block. The quality score for the 16×16 tiles is then taken as the maximum error over its four sub-blocks. We use this approach because it empirically balances noise-tolerance with local adaptivity, e.g. we found that averaging over the full 16×16 tile led to visible artifacts for tiles that span both simple and visually complex image content. Finally, we enforce a heuristic that every tile must use between 50% and 120% of the target bit rate to avoid potential boundary artifacts between tiles with significantly different bit rates. We expect that the use of a more accurate perceptual metric would make this heuristic unnecessary.

Our decoder architecture requires a full tensor of bits so missing entries must be filled. Although the network was trained by mapping binary values to ± 1, we found that using a fill value of zero led to the best reconstruction quality. We believe zero works well because the convolutional

layers use zero-padding, which pushes the network to learn that zero values are uninformative. Zero is also halfway between the standard bit values, which can be interpreted as the least biased value.

SABR requires a small addition to the bitstream generated by our model to inform the decoder about how many bits are used at each location. This "height map" is losslessly compressed using gzip and added to the bitstream. To ensure a fair comparison, the total size of this metadata is included in all of the bit rate calculations in Section 5.

3.4. SSIM Weighted Loss

Training a lossy image compression network introduces a dilemma: ideally we would like to train the network using a perceptual metric as the underlying loss but these metrics are either non-differentiable or have poorly conditioned gradients. The other option is to use the traditional L_1 or L_2 loss; however, these two metrics are only loosely related to perception. To keep the best of both worlds, we propose a weighted L_1 loss between image y and a reference image x

$$L(x,y) = w(x,y)||y-x||_1, \quad w(x,y) = \frac{S(x,y)}{\bar{S}}$$

where $S(x,y)$ is a perceptual measure of dissimilarity between images x and y and where $\bar{S}$ is a dissimilarity baseline. When doing compression, y is the decompressed version of x: $y = f_\theta(x)$ where θ are the compression model parameters. During training, the baseline $\bar{S}$ is set to the moving average of $S(x,y)$. It is not constant but can be considered as almost constant over a short training window. In our experiments, the moving average decay was $\alpha = 0.99$. To actually perform the gradient update, the trick is to consider the weight $w(x,y) = \frac{S(x,f_\theta(x))}{\bar{S}}$ as fixed. This leads to updating using $\theta' = \theta - \eta w(x, f_\theta(x))\nabla_\theta ||f_\theta(x) - x||_1$.

Intuitively, this weighted L_1 loss is performing dynamic importance sampling: it compares the perceptual distortion of an image against the average perceptual distortion and weighs more heavily the images with high perceptual distortion and less heavily the images for which the compression network already performs well.

In practice, we use a local perceptual measure of dissimilarity. The image is first split into 8×8 blocks. Over each of these blocks, a local weight is computed using $D(x,y) = \frac{1}{2}(1 - SSIM(x,y))$ as the dissimilarity measure (DSSIM), where SSIM refers to the structural similarity index [26]. The loss over the whole image is then the sum of all these locally weighted losses. The weighting process can then be thought as a variance minimization of the perceptual distortion across the image, trying to ensure the quality of the image is roughly uniform: any 8×8 block whose perceptual distortion is higher than the average will be over-weighted in the loss.

	Kodak AUC (dB)			Tecnick AUC (dB)		
Method	MS-SSIM	SSIM	PSNR	MS-SSIM	SSIM	PSNR
Baseline	32.96	19.06	59.42	35.49	22.35	64.16
DSSIM	33.43	20.17	60.46	36.02	23.03	64.82
Prime	33.84	20.56	60.94	36.34	23.29	65.19
Best	**34.20**	**21.02**	**61.40**	**36.85**	**23.67**	**65.66**

Table 1. AUC for MS-SSIM (dB), SSIM (dB), and PSNR across Kodak and Tecnick. Baseline uses Figure 2-a and is trained using L_1 reconstruction loss. DSSIM also uses Figure 2-a but is trained using DSSIM reconstruction loss. Prime uses 3-priming (similar to Figure 2-b) and DSSIM training loss. Best is the same as Prime after more training steps.[3] 3-priming shows the best results, which then continue to improve with additional training (last row).

4. Training

All experiments use a dataset of a random sampling of 6 million 1280×720 images on the web. Each minibatch uses 128×128 patches randomly sampled from these images. The Adam optimizer [14] is used with an $\epsilon = 1.0$, $\beta_1 = 0.9$ and a $\beta_2 = 0.999$. All experiments were run with 10 asynchronous workers on NVIDIA Tesla K80 GPUs and clipping all gradient norms over 0.5.

To understand the improvement due to perceptual training metric, separate from those due to hidden-state refinements, we trained two baseline models (Figure 2-a): one using L_1 error for our training loss and the second using our DSSIM loss. Both of these models were trained with a learning rate of 0.5 and a batch size of 8, for 3.8M steps.

We then built on the improvements seen with DSSIM training to investigate the improvements from hidden-state priming (Figure 2-b) for 3-priming. This 3-Prime model as trained in the same way as our two baseline models: with same hyperparameters as above.

Finally, we trained additional models (all using DSSIM training) to investigate k-diffusion for $k = 0$ (which is the same as the DSSIM-trained baseline model), 1, 2, and 3. For $k = 1$ to 3, we repeat the "Encoder Diffusion" and "Decoder Diffusion" steps (Figure 2-c) k times before taking the next step's outputs (bits, for the encoder, or reconstructions, for the decoder) and we do that before every iteration (not just the first, as in priming). For a fair comparison between these models and the DSSIM-trained baseline, we used a learning rate of 0.2, a batch size of 4, and a total of 2.2M steps.[2]

5. Results

In this section, we first evaluate the performance improvements provided to our compression architecture, using our proposed techniques: DSSIM training; priming; and diffusion. Due to the fact that our methods are intended to preserve color information, the computation of all the

k steps of	Kodak AUC (dB)			Tecnick AUC (dB)		
Diffusion	MS-SSIM	SSIM	PSNR	MS-SSIM	SSIM	PSNR
0	31.89	18.75	58.73	34.34	21.78	63.18
1	33.05	19.62	59.91	35.41	22.52	64.23
2	32.85	19.38	59.81	35.28	22.12	64.13
3	**33.40**	**19.87**	**60.35**	**35.68**	**22.70**	**64.70**

Table 2. AUC for MS-SSIM (dB), SSIM (dB), and PSNR across Kodak and Tecnick. All methods in this table used DSSIM for training loss and used diffusion (similar to Figure 2-c) with different numbers of steps between iterations.[3] 3-diffusion provides the best performance in this test (but at a high computational cost).

metrics we report is performed in the RGB domain, following [21, 23].

Next, we show the results for the best model that we have trained to date, which uses 3-priming, trained with DSSIM (but has trained for more steps than the models used in Section 5.1). We compare this model against contemporary image compression codecs (BPG (4:2:0); JPEG2000; WebP; and JPEG) as well as the best recently published neural-network-based approach [21] and [23].

We present results on both Kodak [15] and Tecnick [1] datasets. The Kodak dataset is a set of 24 768×512 images (both landscape and portrait) commonly used as a benchmark for compression. We also compare using the Tecnick SAMPLING dataset (100 1200×1200 images). We feel the Tecnick images are more representative of contemporary, higher resolution content.

5.1. Comparative Algorithm Evaluation

In this subsection, all of our experiments use nominal bit rates: neither entropy coding nor SABR were applied to the RD curves before computing the area under the curve (AUC) values listed in Tables 1 and 2.

We evaluate our results using AUC for peak signal-to-noise ratio (PSNR), SSIM (dB) and MS-SSIM (dB). SSIM (dB) and MS-SSIM (dB) are $-10 \log_{10}(1 - Q)$ where Q is either SSIM [26] or MS-SSIM [27]. Both of these metrics tend to have significant quality differences in the range between 0.98 and 1.00, making them difficult to see on linear graphs and difficult to measure with AUC. This dB transform is also supported by the original MS-SSIM [27] publication, which showed the mean opinion score is linearly correlated with the MS-SSIM score after transforming that score to the log domain. Subsequent compression studies have also adopted this convention, if the methods were able to achieve high-quality–compression results [9].

The Baseline and DSSIM models differ only in the training loss that was used (L_1 or DSSIM). As shown in Table 1, the DSSIM model does better (in terms of AUC) for all of the metrics on both image test sets. Surprisingly, this is true even of PSNR, even though the L_1 loss function should be

---

[2]The smaller batch size was needed due to memory constraints, which forced our learning rate to be lower.

[3]For Tables 1 and 2, no entropy compression or SABR was used: these AUC numbers can not be compared to those in Section 5.2.
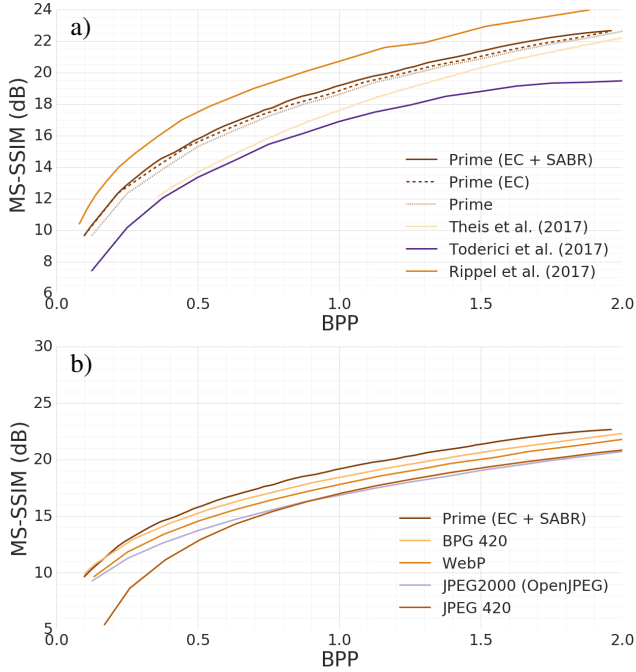
Figure 5. Our full method outperforms existing codecs at all but the lowest bit rate where only BPG 420 matches our performance. This figure shows MS-SSIM (dB) on Kodak: (a) our method compared to [21], [19] and [23] (without entropy coding), and (b) compared to standard image compression codecs. Graphs best viewed on a display.
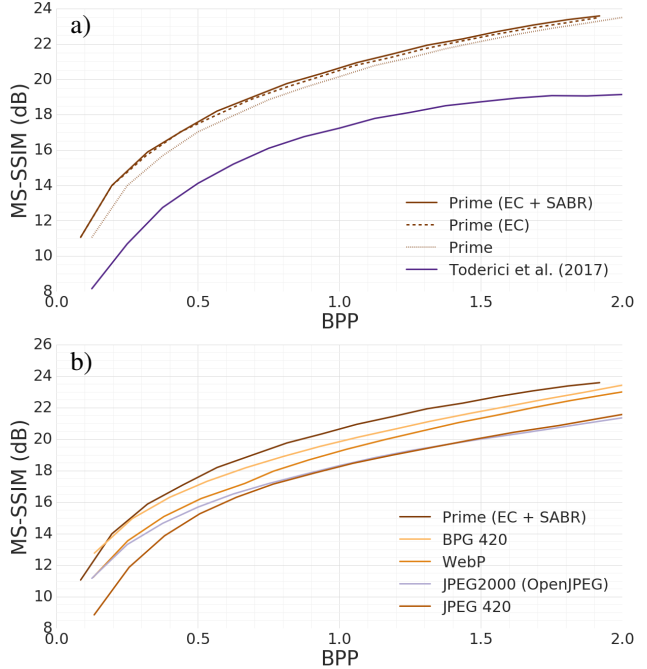

Figure 6. On the larger Tecnick dataset, our full method outperforms existing codecs at all but the lowest bit rate where BPG 420 has a small advantage. This figure shows MS-SSIM (dB) on Tecnick: (a) our method compared to [23] (results on Tecnick were not available for [21] and [19]), and (b) compared to standard image codecs. Graphs best viewed on a display.

closer to PSNR than the DSSIM-weighted L_1 loss. The Prime model (trained with DSSIM loss) does better than the non-priming model, even when both are evaluated at the same number of training steps ("Prime" versus "DSSIM" in Table 1). The Prime model continues to improve with additional training, as shown by the results labeled "Best" in Table 1. While the runtime computation is increased by the use of priming, the percent increase is limited since these extra steps only happen before the first iteration (instead of between all iterations, as with diffusion).

Table 2 reports our AUC results on a second set of models, comparing different numbers of diffusion steps (extensions of Figure 2-c). The results from this experiment show that more diffusion (up to the 3 we tested) increases reconstruction quality. However, as the number of diffusion steps goes up, the resources used also increases: for a k-diffusion network, compression/decompression computation and training times goes up linearly with k. In light of these practical trade offs, we have focused on the Prime model for our comparisons in Section 5.2.

5.2. Best Results Evaluation

The Prime model trained for 4.1 million steps is our best model to date (called "Best" in the tables above). This section evaluates the results when adding entropy coding and SABR to this model.

In Figure 5-a, we compare our best model, with and without entropy coding, to the work reported by Theis et al. [21]. For our entropy coding we train the probability modeler, described in [23], using the codes generated by our model operating on the set of 6 million web images, mentioned in Section 4.

Figures 5-a and 6-a also show our results using SABR (in conjunction with entropy coding) to obtain even higher compression rates. It should be noted that we do not retrain the compression model (or the entropy-coding model) to handle SABR: we use the previously trained models unchanged. This is an area in which we could expect even better performance from our model, if we did some amount of retraining for SABR.

Compared to neural network-based methods, our best model has a better MS-SSIM RD curve than [21, 23]. Our model's curve improves with entropy coding and improves further with SABR.

In Figures 5-b and 6-b, we compare our best model against many popular image compression codecs. We provide examples of our compression results, and those of other popular codecs, in Figure 7.[4] For these image examples, since each of the codecs allows only coarse-level control of the output bit rate, we bias our comparisons against

---

[4]Full-image examples are available in Supplementary material.

JPEG2000	WebP	BPG 420	Our Method

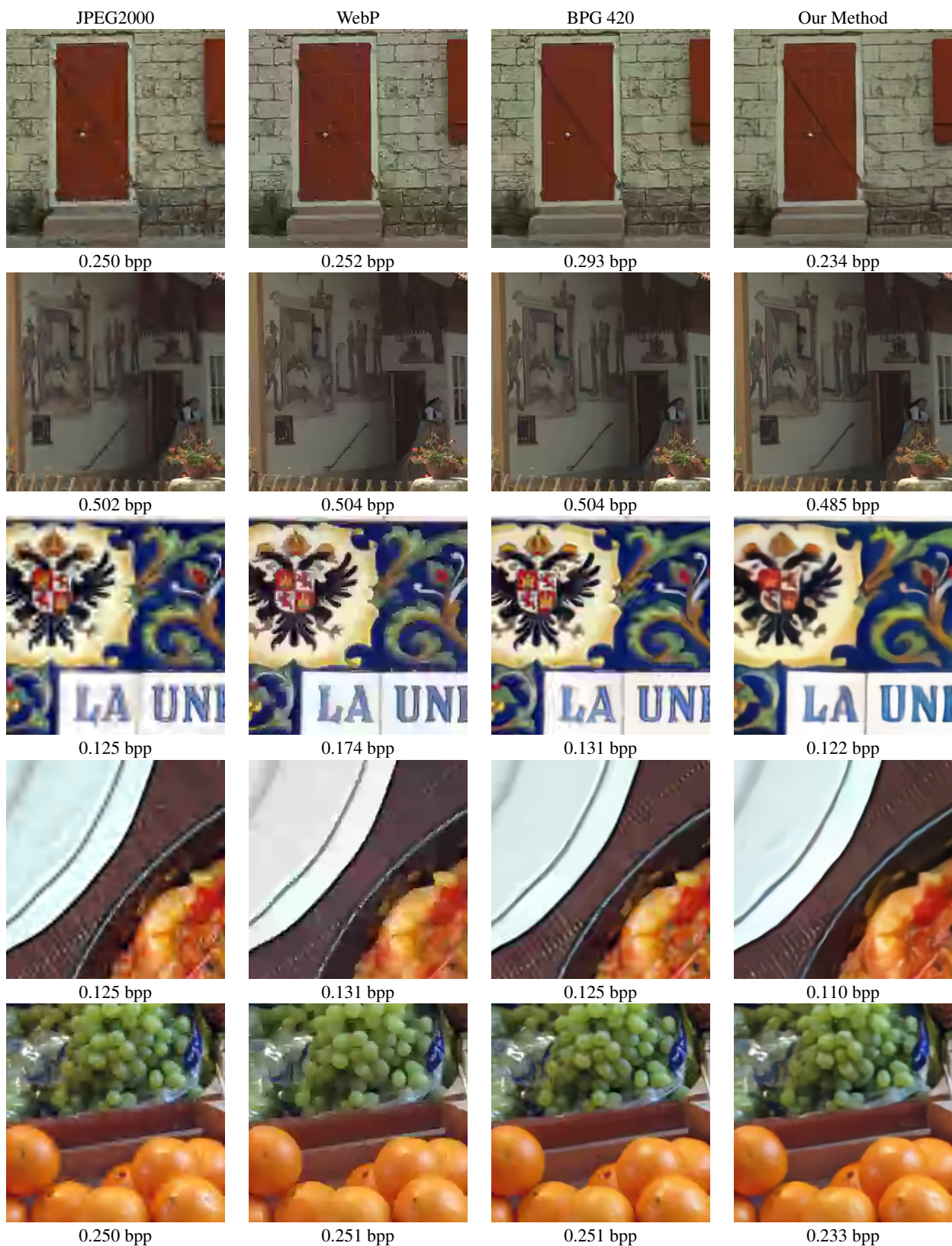| 0.250 bpp | 0.252 bpp | 0.293 bpp | 0.234 bpp |
0.502 bpp	0.504 bpp	0.504 bpp	0.485 bpp
0.125 bpp	0.174 bpp	0.131 bpp	0.122 bpp
0.125 bpp	0.131 bpp	0.125 bpp	0.110 bpp
0.250 bpp	0.251 bpp	0.251 bpp	0.233 bpp

Figure 7. Example patches comparing our Best-model results with JPEG2000 (OpenJPEG), WebP and BPG 420. For the most visible differences, consider: (first row) the cross bar on door; (second row) the handrail and the hanging light in front of the dark wood; (third row) the text; (fourth row) the pan edge and the plate rim; (fifth row) the outlines of the oranges and the crate edge. Image best viewed zoomed in on a display.

Method	Kodak Rate Difference %			Tecnick Rate Difference %		
	MS-SSIM	SSIM	PSNR	MS-SSIM	SSIM	PSNR
Rippel et al.[19]	**58.11**	–	–	–	–	–
Prime (EC + SABR)	43.17	39.97	27.14	**45.65**	40.08	17.36
Prime (EC)	41.70	36.51	19.29	44.57	36.82	9.73
BPG 444	40.04	44.86	**56.30**	44.10	**44.25**	**55.54**
BPG 420	37.04	**46.94**	54.85	36.27	43.02	48.68
Prime	36.32	30.89	12.20	35.05	26.86	-6.09
JPEG2000 (Kakadu)	31.75	22.23	28.29	35.18	27.44	27.08
WebP	26.85	29.85	36.33	24.28	23.35	23.14
JPEG2000 (OpenJPEG)	15.99	24.80	38.28	14.34	20.70	26.08
Theis et al.[21]	15.10	28.69	29.04	–	–	–
Toderici et al.[23]	12.93	-1.86	-13.34	-25.19	-44.98	-67.52

Table 3. Bjøntegaard rate-difference on MS-SSIM, SSIM and PSNR for Kodak and Tecnick datasets. This shows the bit rate difference across each metric (larger numbers are better). Codecs are sorted in order of MS-SSIM bit-rate difference, while the best result in each metric is bolded.
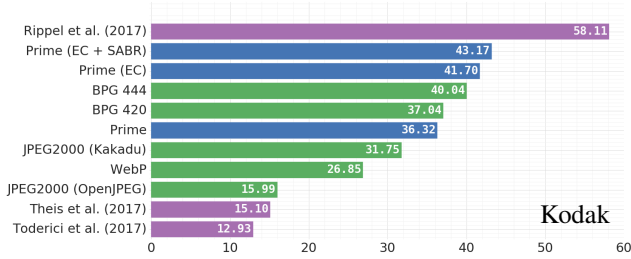

Figure 8. Our approach (Prime) outperforms standard codecs and many existing neural-network-based methods. This figure shows rate savings (Bjøntegaard Delta) relative to JPEG under MS-SSIM for the Kodak dataset. Standard codecs are shown in green, purple represents recent research using neural networks [21, 23, 19], and our methods are shown in blue.
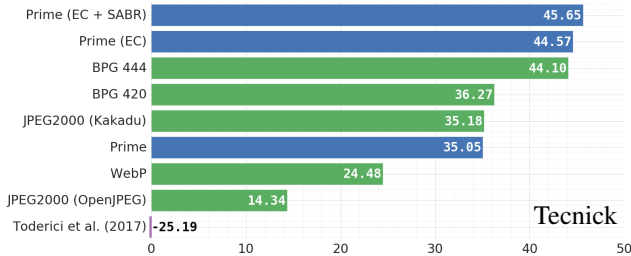

Figure 9. Rate savings (Bjøntegaard Delta) relative to JPEG under MS-SSIM for the Tecnick dataset. By this measure, two of our approaches outperform current standard codecs and all of our approaches outperform previous research in network-based codecs.

our own model. Specifically, when doing our comparisons, we always choose an average bit rate that is the same or larger than the bit rate produced by our method (giving an advantage to the other methods in the comparison). Qualitatively, our method tends to oversmooth at low bitrates at the cost of improved color fidelity.

Finally, Table 3 and Figures 8 and 9 summarizes the rate savings using Bjøntegaard Delta (BD) rate differences [6]. BD rate differences are the percent difference in area between two RD curves, after a logarithmic transform of the bit rate. When computing BD rate savings on methods that fail to deliver the full quality range, the difference in area is only computed across quality levels provided by both curves. BD rate differences use the log rate since the human visual system is more sensitive to low-bit-rate areas than to the high-bit-rate areas.[5] The BD difference was originally defined for PSNR, but since its publication, better measures of quality have been proposed [27]. As a result, we are reporting the BD rate computed on the logarithmic transform of MS-SSIM.

6. Conclusion

We introduced three techniques — hidden-state priming, spatially adaptive bit rates, and perceptually-weighted training loss — and showed that they boost the performance of our baseline recurrent image compression architecture. Training with a perceptually-weighted L_1 loss improved our model's performance on MS-SSIM, SSIM, and PSNR. Hidden-state priming provides further improvements to reconstruction quality, similar to that of diffusion but with lower computational requirements during inference and training. The quality improvements seen with priming are related to initializing the hidden states within the network with content-dependent values. Additionally, we're confident that this technique can be applied to other recurrent networks, not just compressive recurrent autoencoders.

The additional quality improvements seen with diffusion are probably related to the increased spatial context available in the decoder: with 3-diffusion, the decoder's context nearly doubles, increasing from about $6(t+1)$ to $10(t+1)$ "bit stacks" (where a bit stack occurs each 16×16 pixels). Finally, SABR can reduce the bit rate in the areas of the image that are easier to compress. In our models, this adaptivity more than makes up for the additional overhead needed to send the SABR height map.

Combining these three techniques improves the MS-SSIM rate-distortion curve for our GRU-based architecture, surpassing that of recent neural-network-based methods ([21] and [23]) and many standard image codecs (BPG 420, WebP, JPEG2000, and JPEG). Our approach is still not state of the art in MS-SSIM on Kodak when comapred to [19]. The first main difference in our approach is we present one model for all points on the rate-distortion plot instead of needing multiple models for the entire range. Secondly, our perceptually-weighed L1 loss function gave a boost across all three of our tracked metrics, MS-SSIM, SSIM and PSNR, while initial evidence showed optimizing for MS-SSIM directly can give a large boost in SSIM based metrics with a substantial decrease in others.

Our approach is the first recurrent neural-network-based codec shown to outperform WebP and provide competitive coding efficiency with some variants of BPG.

---

[5] The Supplementary material provide additional details about computing Bjøntegaard measures, as well as the quality-improvement BD results.

References

[1] N. Asuni and A. Giachetti. TESTIMAGES: A large-scale archive for testing visual devices and basic image processing algorithms. In *STAG: Smart Tools and Apps for Graphics*, 2014. 5

[2] M. H. Baig and L. Torresani. Multiple hypothesis colorization and its application to image compression. *Computer Vision and Image Understanding*, 2017. 2

[3] J. Ballé, V. Laparra, and E. P. Simoncelli. Density modeling of images using a generalized normalization transformation. In *Int'l. Conf. on Learning Representations (ICLR2016)*, May 2016. 2

[4] J. Ballé, V. Laparra, and E. P. Simoncelli. End-to-end optimized image compression. In *Int'l. Conf. on Learning Representations (ICLR2017)*, Toulon, France, April 2017. Available at http://arxiv.org/abs/1611.01704. 1, 2

[5] F. Bellard. BPG image format (http://bellard.org/bpg/). Accessed: 2017-01-30. 1, 2

[6] G. Bjøntegaard. Calcuation of average PSNR differences between RD-curves. *Doc. VCEG-M33 ITU-T Q6/16, Austin, TX, USA, 2-4 April 2001*, 2001. 8

[7] D. Bull. Communicating pictures. In *Communicating Pictures*. Academic Press, 2014. 2

[8] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 3

[9] Google. WebP: Compression techniques (http://developers.google.com/speed/webp/docs/compression). Accessed: 2017-01-30. 1, 2, 5

[10] K. Gregor, F. Besse, D. Jimenez Rezende, I. Danihelka, and D. Wierstra. Towards conceptual compression. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3549–3557. Curran Associates, Inc., 2016. 1, 2

[11] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006. 2

[12] Information technology–JPEG 2000 image coding system. Standard, International Organization for Standardization, Geneva, CH, Dec. 2000. 1, 2

[13] J. Jiang. Image compression with neural networks–a survey. *Signal Processing: Image Communication*, 14:737–760, 1999. 2

[14] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. 5

[15] E. Kodak. Kodak lossless true color image suite (PhotoCD PCD0992). 5

[16] A. Krizhevsky and G. E. Hinton. Using very deep autoencoders for content-based image retrieval. In *European Symposium on Artificial Neural Networks*, 2011. 2

[17] D. Minnen, G. Toderici, M. Covell, T. Chinen, N. Johnston, J. Shor, S. J. Hwang, D. Vincent, and S. Singh. Spatially adaptive image compression using a tiled deep network. *International Conference on Image Processing*, 2017. 1

[18] P. Munro and D. Zipser. Image compression by back propagation: An example of extensional programming. *Models of cognition: A review of cognitive science*, 1989. 2

[19] O. Rippel and L. Bourdev. Real-time adaptive image compression. In *The 34th International Conference on Machine Learning*, 2017. 1, 2, 6, 8

[20] S. Santurkar, D. Budden, and N. Shavit. Generative compression. *arXiv:1703.01467*, 2017. 2

[21] L. Theis, W. Shi, A. Cunningham, and F. Huszar. Lossy image compression with compressive autoencoders. In *Int'l. Conf. on Learning Representations (ICLR2017)*, 2017. 1, 2, 5, 6, 8

[22] G. Toderici, S. M. O'Malley, S. J. Hwang, D. Vincent, D. Minnen, S. Baluja, M. Covell, and R. Sukthankar. Variable rate image compression with recurrent neural networks. *ICLR 2016*, 2016. 1, 2, 3

[23] G. Toderici, D. Vincent, N. Johnston, S. J. Hwang, D. Minnen, J. Shor, and M. Covell. Full resolution image compression with recurrent neural networks. *CVPR*, abs/1608.05148, 2017. 1, 2, 5, 6, 8

[24] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol. Extracting and composing robust features with denoising autoencoders. *Journal of Machine Learning Research*, 2012. 2

[25] G. K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, pages 30–44, 1991. 1

[26] Z. Wang, A. Bovik, A. Conrad, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004. 1, 4, 5

[27] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multiscale structural similarity for image quality assessment. In *Signals, Systems and Computers, 2004. Conference Record of the Thirty-Seventh Asilomar Conference on*, volume 2, pages 1398–1402. IEEE, 2003. 1, 5, 8

[28] H. Zhao, O. Gallo, I. Frosio, and J. Kautz. Loss functions for image restoration with neural networks. In *IEEE Transcations on Computational Imaging*, volume 3, March 2017. 1