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

In this Supplementary Materials document, we include some cross-references to the main paper, for clarity. References that are prefixed with "P-" refer to the main paper. References to material within this supplementary document are made without prefix.

1. Derivation of Spatial Support Equations

Our belief is that some of the improvement in the compression results with diffusion and, to some extent, with priming is due to the additional spatial context provided by these stages that is available to the hidden-state vectors of the recurrent units. This section provides the derivation of the spatial-support equations provided in Subsections P-3.1 and P-3.2.

Figure P-1 shows, at a high level, a single iteration through the network architecture that we use for our encoder and decoder networks. The name and type of layer is shown by the "$E_i : I/H$" for the encoder (or "$D_i : I/H$" for the decoder) label in the bottom, front corner of each plane: the input convolutional kernels will be (spatially) $I \times I$ (and fully connected across depth) and the hidden convolutional kernels will be $H \times H$, with $H = 0$ corresponding to layers which have no hidden state. The input to the encoder is the residual image: that is, the difference between the previous iteration's reconstruction and the original image. On the first iteration, this residual is just the original image itself. With k-priming and k-diffusion, the input that is to be used on the next iteration is processed k times, with the outputs (either the bit stacks or the reconstruction) discarded before the actual iteration is run and the output is kept.

The recurrent layers in the encoder and decoder (that is, E_1, E_2, E_3, D_1, D_2, D_3, and D_4) use Gated Recurrent Units [4] (GRU):

$$y_t = h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t, \tag{1}$$

$$\tilde{h}_t = \tanh(W x_t + U(r_t \odot h_{t-1})), \tag{2}$$

$$z_t = \sigma(W_z x_t + U_z h_{t-1}), \tag{3}$$

$$r_t = \sigma(W_r x_t + U_r h_{t-1}). \tag{4}$$

where t is the iteration number; $\odot$ denotes element-wise multiplication; and omitted operators correspond to spatial convolution (with fully connected operation across depth). Using the terms introduced by [4]: r_t is the reset gate; z_t is the update gate; h_t is the activation; and $\tilde{h}_t$ is the candidate activation. The output, y_t, is the same as the activation, h_t. The spatial extents of W, W_r, and W_z are $I \times I$: that is, 3×3 for all the GRU layers in Figure-P 1. The spatial extents of U, U_r, and U_z are $H \times H$: that is, 1×1 for all of the GRU layers in the encoder and for the first two GRU layers in the decoder and 3×3 for the last two GRU layers in the decoder.

Using the Baseline architecture (Figure P-2-a), the spatial context from the original image input to each binary code and to each pixel of the reconstruction can be computed by examining the stacked spatial supports of the encoder (for the first iteration's binary codes); of the encoder followed by the decoder (for the first iteration's reconstruction); or of encoder, decoder, and all state vectors (for subsequent iterations). We use $\mathcal{S}_I(F_{t-1})$ to denote the pixel-to-pixel support, starting from the input image through to the output image for iteration $t-1$ (where the subscript I is used to refer to the target input image, as opposed to subscript B, below, to refer to the bit rate). Then, for iteration t, in the encoder, since the hidden convolutions are all 1×1 (or none for the feedforward layers) each layer i in the encoder increases the spatial context from the previous layer $i-1$ to give a $\mathcal{S}_I(E_{i,t}) \times \mathcal{S}_I(E_{i,t})$ support equation of $\mathcal{S}_I(E_{i,t}) = (I_i - 1)2^i + \mathcal{S}_I(E_{i-1,t})$ with $\mathcal{S}_I(E_{-1,t}) = \mathcal{S}_I(F_{t-1})$ and $\mathcal{S}_I(F_{-1}) = 1$. On the first iteration, the spatial context of the last encoder GRU layer is $E_{3,0} = 31$. (The feedforward layer just before the binarizer ('1/-1') does not change the spatial support.)

The support from the binary code to the i^{th} layer of the decoder network goes as $\mathcal{S}_B(D_{i,t}) \times \mathcal{S}_B(D_{i,t})$ where

$$\begin{aligned}\mathcal{S}_B(D_{i,t}) = \max(& \\ & \frac{\max(H_i - 1, 0) + I_i - 1}{2^{\max(i-1,0)}} + \mathcal{S}_B(D_{i-1,t}), \\ & \frac{\max(2 * (H_i - 1), 0)}{2^{\max(i-1,0)}} + \mathcal{S}_B(D_{i,t-1}))\end{aligned} \tag{5}$$

and $\mathcal{S}_B(D_{0,t}) = D_{i,-1} = 1$. The concatenation of H and I in the expansion of the support is due to the chained dependence from x_t to r_t (using W_r with a support of $I \times I$) and then from r_t through $\tilde{h}_t$ to y_t (using U with a support of $H \times H$) while the doubling of H is due to the chained

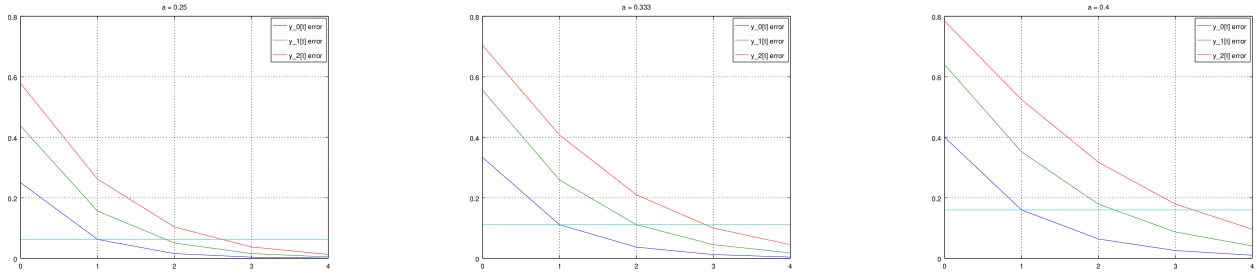

Figure 1. Error in stacked single-pole IIR filters, starting from a zero initialization state, using $a = 0.25$, $a = 0.333$, and $a = 0.4$, from left to right. The later filters in the stack take more iterations to achieve the same level of error, even though their coefficients and initializations are the same, due to the compounding effect of the incorrect inputs that they receive from the previous filters at early time steps and the zero-state initialization.

dependence from h_{t-1} to r_t (using U_r with a support $H \times H$) and then (again) from r_t through $\tilde{h}_t$ to y_t. For those layers where D_i is a fractional number, the floor and ceiling of the spatial dependence each happen at different offsets within the reconstruction. Using this formula, $D_{4,0} = 5.5$ (so the spatial dependence from the binarizer to the output of the last decoder layer is $\{5, 6\} \times \{5, 6\}$ depending on position).

The support from the input image to the decoded image is $\mathcal{S}_I(F_t) = (\mathcal{S}_B(D_{4,t}) - 1) * 16 + \mathcal{S}_I(E_{3,t})$.

Bringing all of these together for the architecture shown in Figure P-1 gives maximum supports of

$$\max(\mathcal{S}_B(F_t)) = 6 * t + \lceil 5.5 \rceil \quad (6)$$

$$\max(\mathcal{S}_I(F_t)) = 16 * \max(\mathcal{S}_B(F_t)) + 15 \quad (7)$$

Using the priming or diffusion architectures (similar to Figure P-2-b and -c, respectively), there are additional stages before the first set of used encoder bits and additional stages before the first used reconstruction from the decoder. Since the input values do not change during these priming stages, the support on the encoder (or decoder) does not change. However, the support provided by the hidden states within recurrent units that have a hidden-kernel support larger than one will change. On the first layer where the hidden-kernel support is larger than one, the path for the expanding support is through the hidden-state propagation: that is, $\frac{\max(2*(H_i-1),0)}{2^{\max(i-1,0)}} + \mathcal{S}_B(D_{i,t-1})$ in Equation 5. On layers that follow a previous layer with hidden-kernel support larger than one that themselves have hidden-kernel support larger than one, both terms in Equation 5 provide alternatives for increases in spatial support. With the kernels that we have used (see Figure P-1), these two alternatives provide the same increase in support on these later layers.

For priming, the increase only happens once, before the first iteration. It operates as if there were an additional k repetitions of the hidden layers with the spatially-expansive hidden-kernel support, but only for the $t = 0$ pass through Equation 5. The increase for diffusion is similar but occurs on all iterations (not just on the first one), so the final amount of increase in support depends on the number of iterations t. In both cases, using the kernels indicated in Figure P-1, each priming or diffusion stage in the decoder adds another $\lceil 1.5 \rceil$ bit stacks to dependence $\mathcal{S}_B(F_t)$. Pulling that all together gives

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

2. Error Propagation due to Hidden-State Initialization

In addition to a wider spatial support, priming allows the GRU units to initialize the hidden-state values to an input-dependent value before it is used in the output. We can make an analogy to the output of an infinite-impulse response (IIR) filter. Let's say that we have a simple (single-pole) IIR filter:

$$y[t] = ay[t-1] + (1-a)x[t] \quad (8)$$

for $0 < a < 1$. As with GRU, the hidden state is the previous output value. Comparing the GRU unit to a single pole filter is the best analogy, since both have memory of the previous–time-step output but not direct memory of further back in time.

If the input, $x[t]$, is a constant (e.g., 1), we ultimately want Equation 8 to converge to that same constant. If we start from an initialization of $y[-1] = 0$ (as we do with our GRU layers), the error in the value of Equation 8 is a^{t+1}. For this one single-pole filter, if we are willing to accept an error of a^2, then we would accept the output at $t = 1$: that is, with 1-priming.

When we have stacked recurrent units, then the layers after the first layer start, not only with the wrong hidden state but also with the wrong input value, since that input value includes the error from the previous recurrent layer. Continuing with our IIR example, let us consider if we were to stack three layers of the same single-pole IIR filter, similar to what we have in our encoder network. The error at the

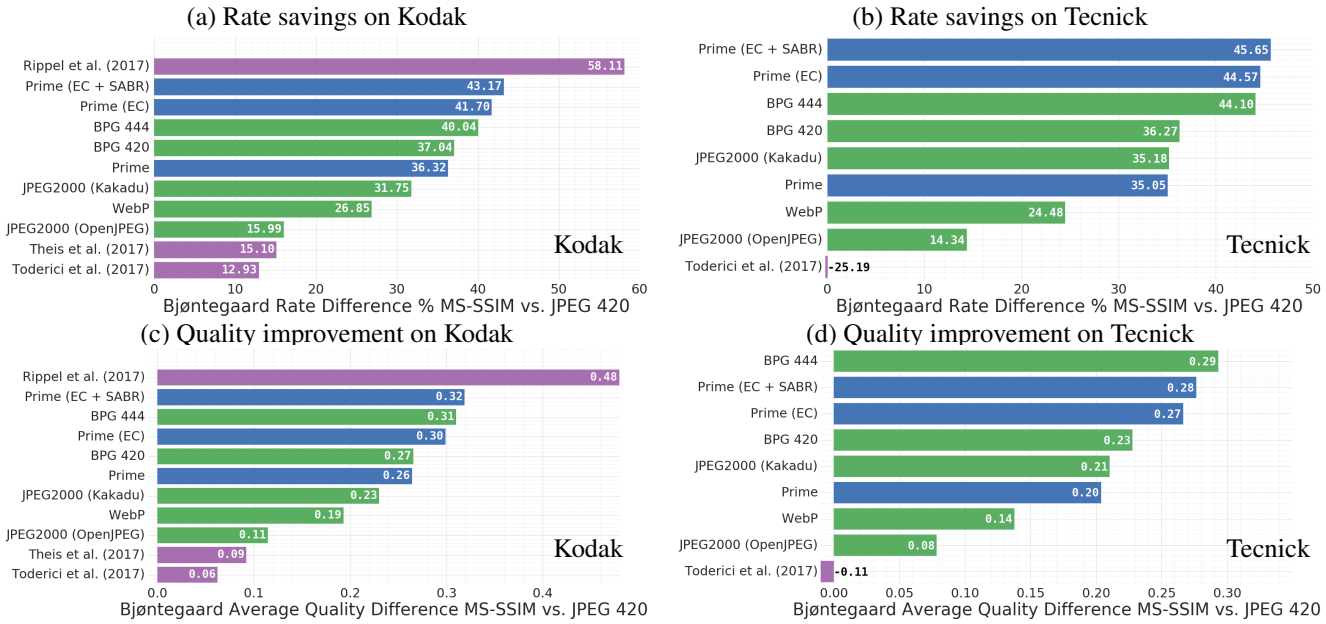

Figure 2. Rate savings (a & b) and quality improvement (c & d) for different codecs, relative to JPEG 420 under MS-SSIM for Kodak (a & c) and Tecnick (b & d), according to Bjøntegaard Delta.

output of the second filter is $(t+2)a^{t+1}-(t+1)a^{t+2}$, which is greater than a^2 for $t = 1$ and all values of $0 < a < 1$. For an error threshold of a^2, we would need to use 2-priming if $0 < a \leq \frac{1}{3}$ or 3-priming for $\frac{1}{3} < a < 1$. Note that, even though we have tied our error threshold to the value of a and that threshold gets *looser* as the value of a increases towards one, it still takes more priming steps to fall below that error threshold.

The error at the output of third filter is $\frac{(t+2)(t+3)}{2}a^{t+1} - (t+1)(t+3)a^{t+2} + \frac{(t+1)(t+2)}{2}a^{t+3}$ which is greater than the second-filter output error for all values of $0 < a < 1$. If a is less than about $\frac{1}{3}$ and we want an output error on the third filter that is a^2 or less, we need to use 3-priming; for larger values of a and an a^2-error threshold, we need to use 4-priming. Figure 1 shows the error curves (and error threshold of a^2) for values of a at, just above, and just below $\frac{1}{3}$.

Since the encoder has a stack of three GRU layers and since, in their linear operating range, they will act like a (vector) single-pole IIR filter, we believe this analogy provides some reasoning for why 3-priming might work so well.

3. Bjøntegaard Delta (BD) rate differences

Bjøntegaard Delta (BD) measures summarize the rate savings or quality improvements of on codec compared to another [3]. BD rate differences are the percent differences in area between two RD curves, after a logarithmic transform on the bitrate. 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 bitrate since the human visual system is more sensitive to low-bit-rate areas than to the high-bit-rate areas.

The RD curves that are being compared typically do not have samples at the exact same bit rates. So, before computing the difference in area, Bjøntegaard also proposed using a polynomial fit of the curves, then sampling the fitted curve over 100 points (common for both curves), and using the trapezoidal integration method to compute the areas.

The BD difference was originally defined for PSNR, but since its publication, better measures of quality have been proposed [11]. As a result, we are reporting the BD rate computed on the logarithmic transform of MS-SSIM.

Table P-3 and Figures P-8 and P-9 compare various codecs to JPEG 420. Due to the lower quality of JPEG at 2 bpp than the quality of the other codecs, we computed the RD curve for JPEG out to 4 bpp, before computing the intersection in quality levels. This allowed us to provide BD numbers that are influenced by the high-quality bitrates (around 1 to 2 bpp) for the other codecs that were examined.

The BD rate-savings measure takes the difference in bit rate for each quality level that is being integrated over. Similarly, we can examine the quality improvement by taking the difference in quality for each bit rate that is being integrated over. Figure 2 shows the rate savings and quality improvements using this measure, according to MS-SSIM.

Figure 3 shows the BD measures using SSIM (instead

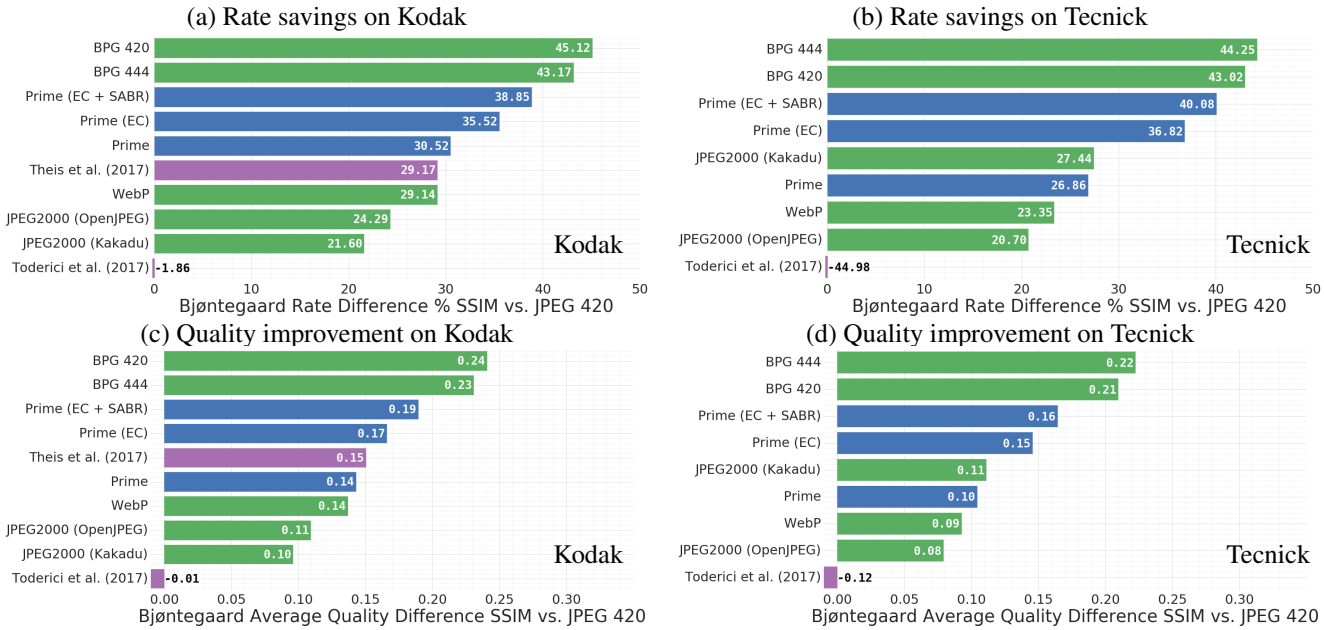

Figure 3. Rate savings (a & b) and quality improvement (c & d) for different codecs, relative to JPEG 420 under SSIM for Kodak (a & c) and Tecnick (b & d), according to Bjøntegaard Delta.

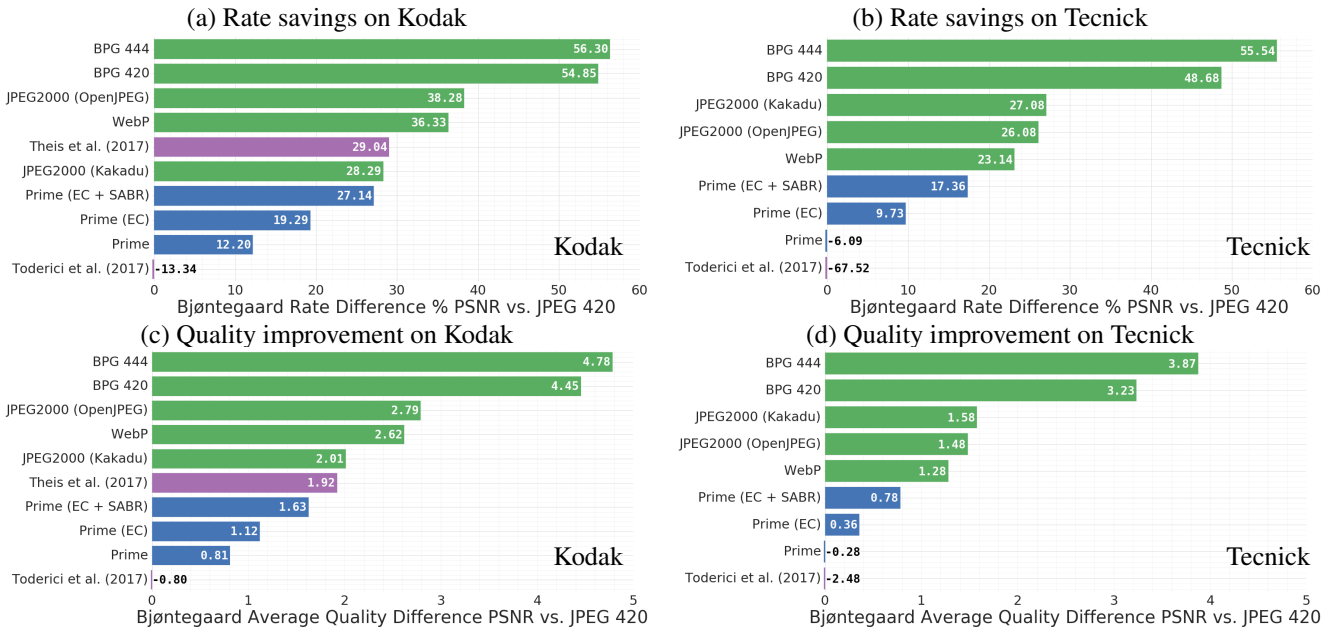

Figure 4. Rate savings (a & b) and quality improvement (c & d) for different codecs, relative to JPEG 420 under PSNR for Kodak (a & c) and Tecnick (b & d), according to Bjøntegaard Delta.

of MS-SSIM) and Figure 4 shows them using PSNR. Our methods do much worse, compared to other state-of-the-art codecs when PSNR is used as the quality metric. However, other research has shown the MS-SSIM is more strongly correlated with human perception that PSNR [11].

4. Rate Distortion Curves

Figures 5, 6 and 7 show the RD curves on the Kodak data set [8] using MS-SSIM, SSIM, and PSNR, respectively. The figures include multiple existing codecs as well as different implementations (Kakadu vs. Open-

Jpeg for JPEG2000) and parameter settings (YCbCr 4:4:4 and 4:2:0 for BPG). The curves marked "Prime", "Prime (EC)", and "Prime (EC + SABR)" are the RD results for our Best model. "Prime" is the base model that uses the nominal bitrate (without entropy coding or SABR), "Prime (EC)" includes entropy coding but not SABR, and "Prime (EC + SABR)" uses both SABR and entropy coding. "BPG (4:4:4)" and "BPG (4:2:0)" use code available from [2] in the YCbCr colorspace with the indicated chroma subsampling. "JPEG2000 (OpenJPEG)" and "JPEG2000 (Kakadu)" use the OpenJPEG (v2.1.2) [5] and Kakadu Software (v7.9) [9] implementation of the JPEG2000 standard. Both results use five layers, which gave slightly better results than six, the other commonly recommended setting. "WebP" uses the code available from Google (v0.6.0) [6] with automatic filter strength (-af), sharp YUV conversion (-sharp_yuv), and the slowest (highest quality) compression method (-m 6). "Theis et al. (ICLR 2017)" uses the RD curves reported by [10] (note that evaluation numbers are only available for Kodak for this method). "Toderici et al. (CVPR 2017)" uses the models that were open-sourced at [7]. All of these RD curves are computed in the RGB space by averaging across the color channels.

Similar to the above, figures 8, 9 and 10 show the RD curves on the Tecnick data set [1] using the same distortion metrics (MS-SSIM, SSIM, and PSNR). The Tecnick data set is made up of 100 digital photographs (compared to 24 for Kodak), each with a resolution of 1200×1200 (vs. 768×512 for Kodak).

5. Full Image Examples

In this section, we show the full (un-cropped) images for the examples we provided in Figure P-7. As with Figure P-7, we have set a nominal target bitrate which we would like (e.g., $\frac{1}{4}$ bpp); selected the reconstructions from JPEG 2000, WebP, and BPG that use that bitrate or higher (e.g., 0.293 bpp for BPG in Figure 12); and selected the reconstruction from our Best model that uses that bitrate or lower (e.g., 0.234 in Figure 12). This approach puts our method at a disadvantage, sometimes a quite large disadvantage: in Figure 12, BPG uses a 20% higher bitrate than our Best model. However, since none of the codecs provide fine-grain control of the output bitrate, we chose this to avoid any question of providing our model with a bitrate advantage. Figure 11 shows reconstructions near $\frac{1}{2}$ bpp for 768×512 pixels. Figures 12 and 13 show reconstructions near $\frac{1}{4}$ bpp across two different resolutions (768×512 for Figure 12 and 1200×1200 for Figure 13). Figures 14 and 15 show reconstructions near $\frac{1}{8}$ bpp for 1200×1200 pixels. Suggestions for areas of the images that show differences in encoding approaches are included in the captions.

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

[3] 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. 3

[4] 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. 1

[5] U. de Louvain (UCL). Openjpeg: An open-source jpeg 2000 codec written in c. 5

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

[7] N. Johnston. Image compression with neural networks: tensorflow/models. 5

[8] E. Kodak. Kodak lossless true color image suite (PhotoCD PCD0992). 4

[9] K. Software. Kakadu jpeg 2000 codec. 5

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

[11] 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. 3, 4

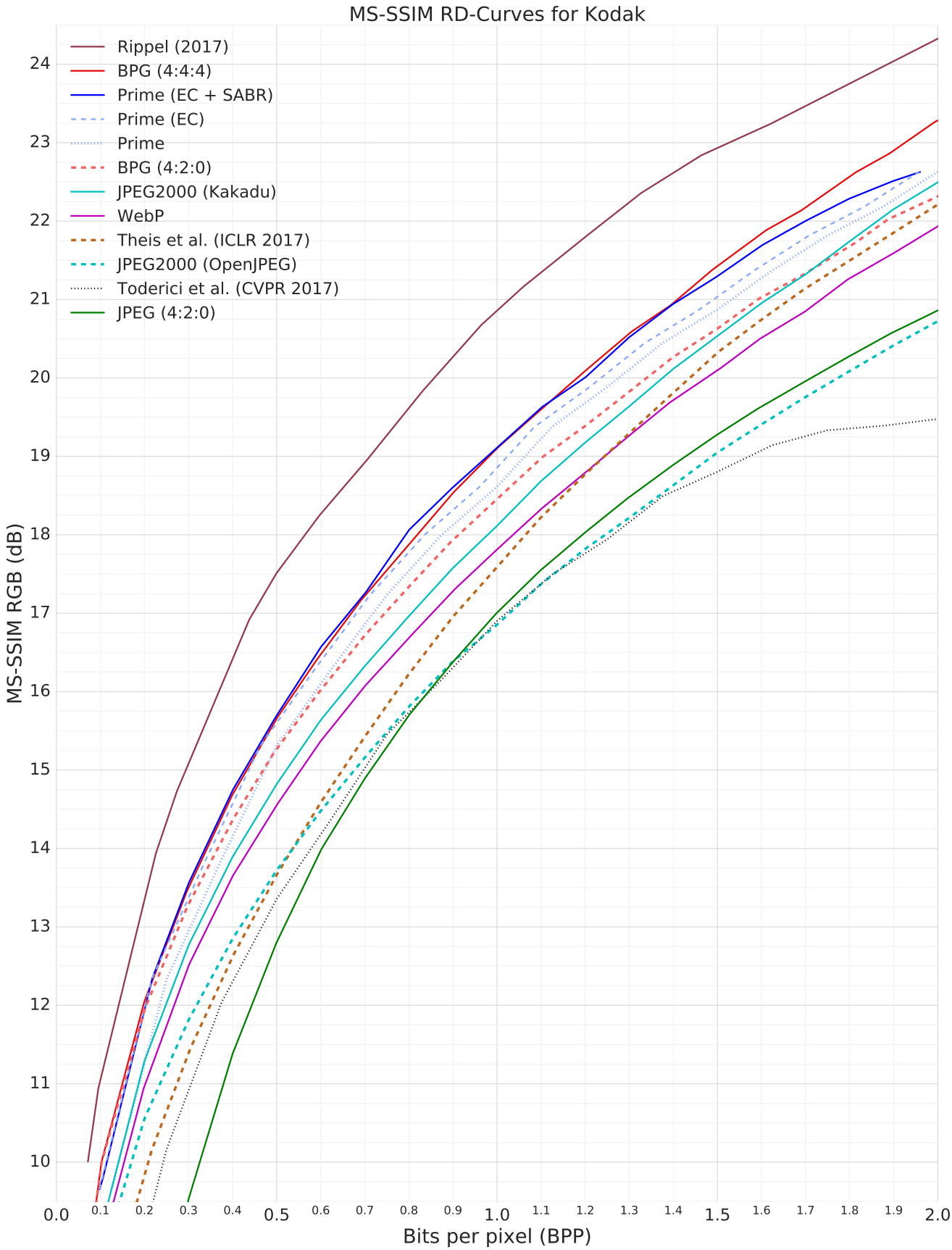

Figure 5. Rate distortion curve for MS-SSIM on Kodak.

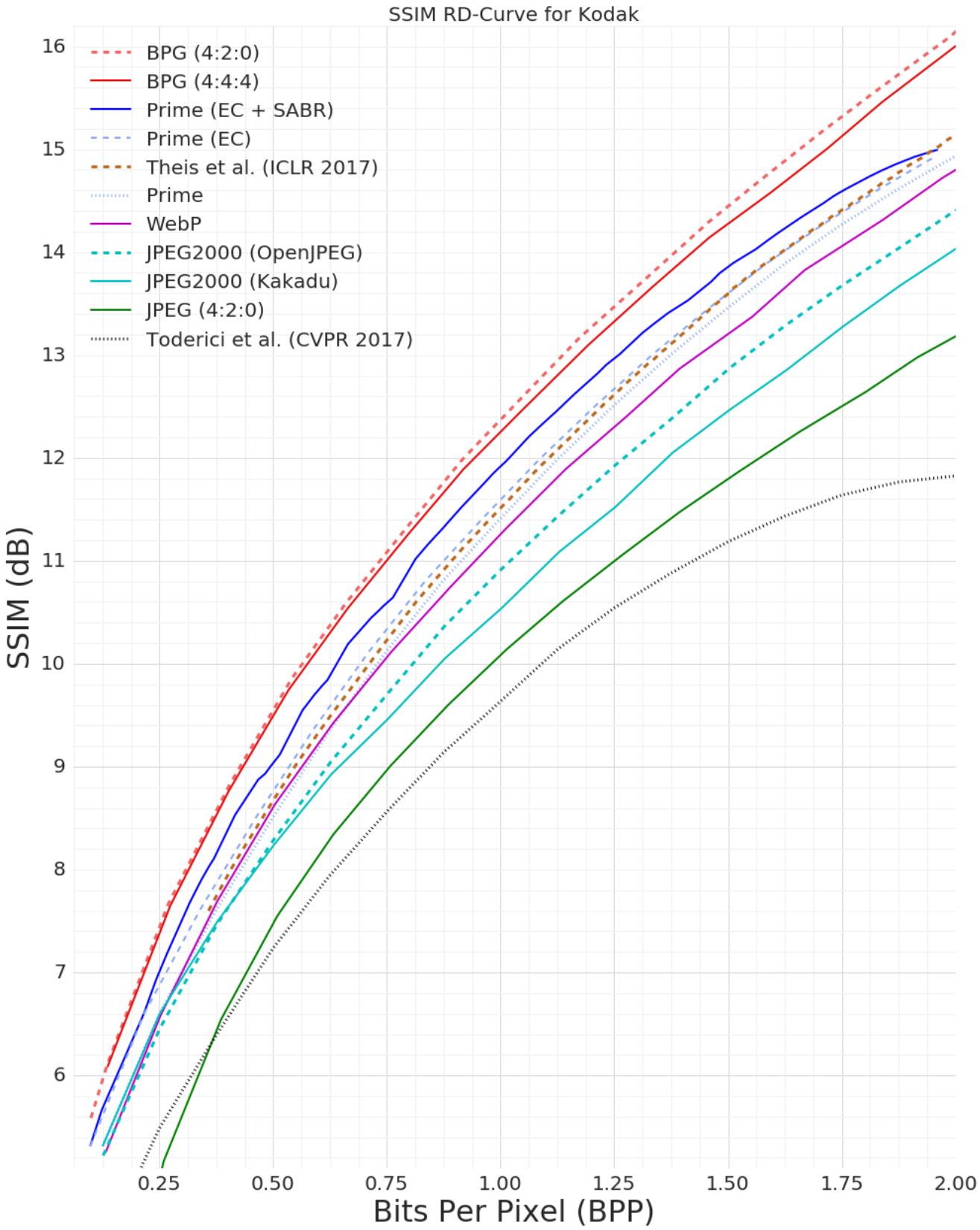

Figure 6. Rate distortion curve for SSIM on Kodak.

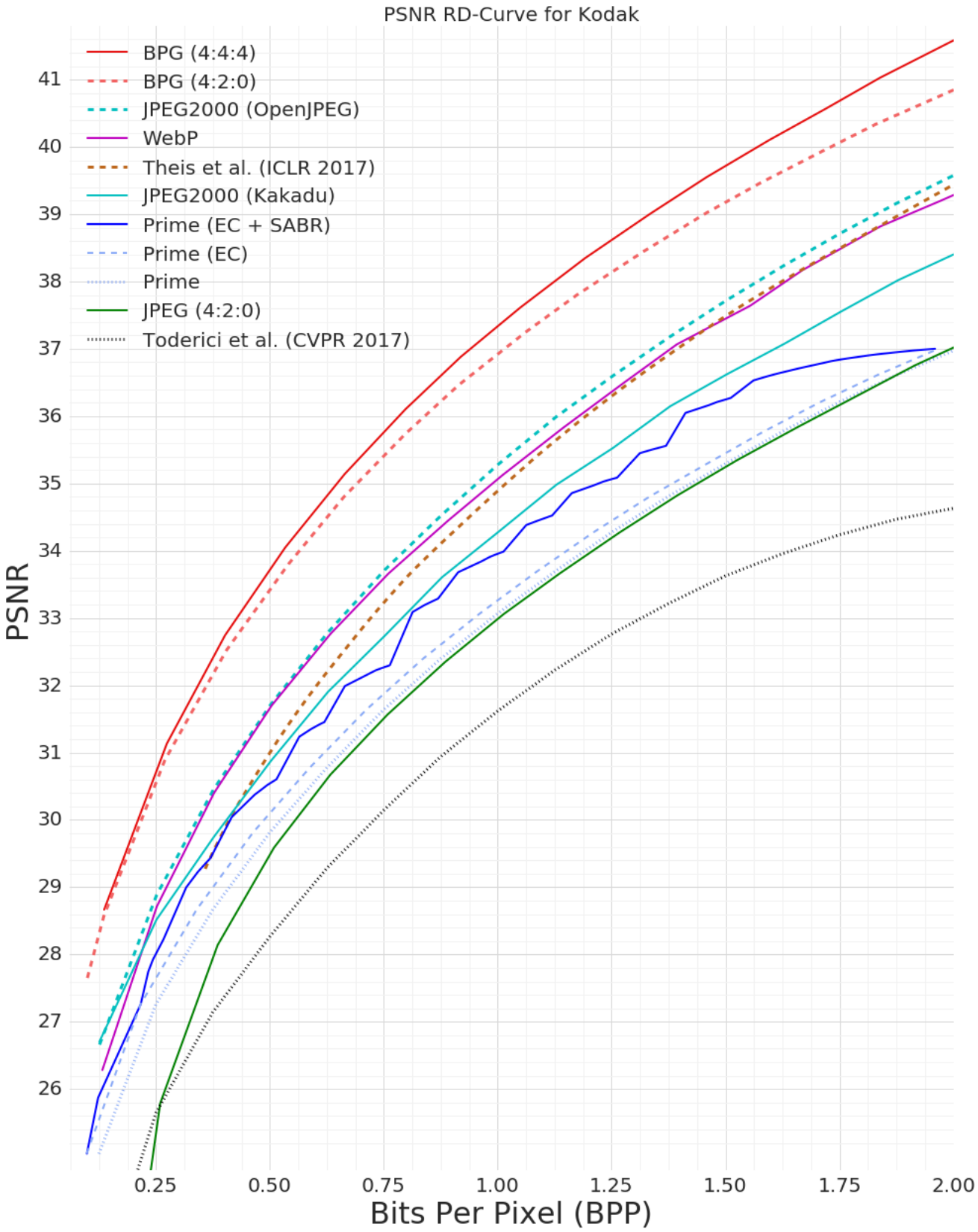

Figure 7. Rate distortion curve for PSNR on Kodak.

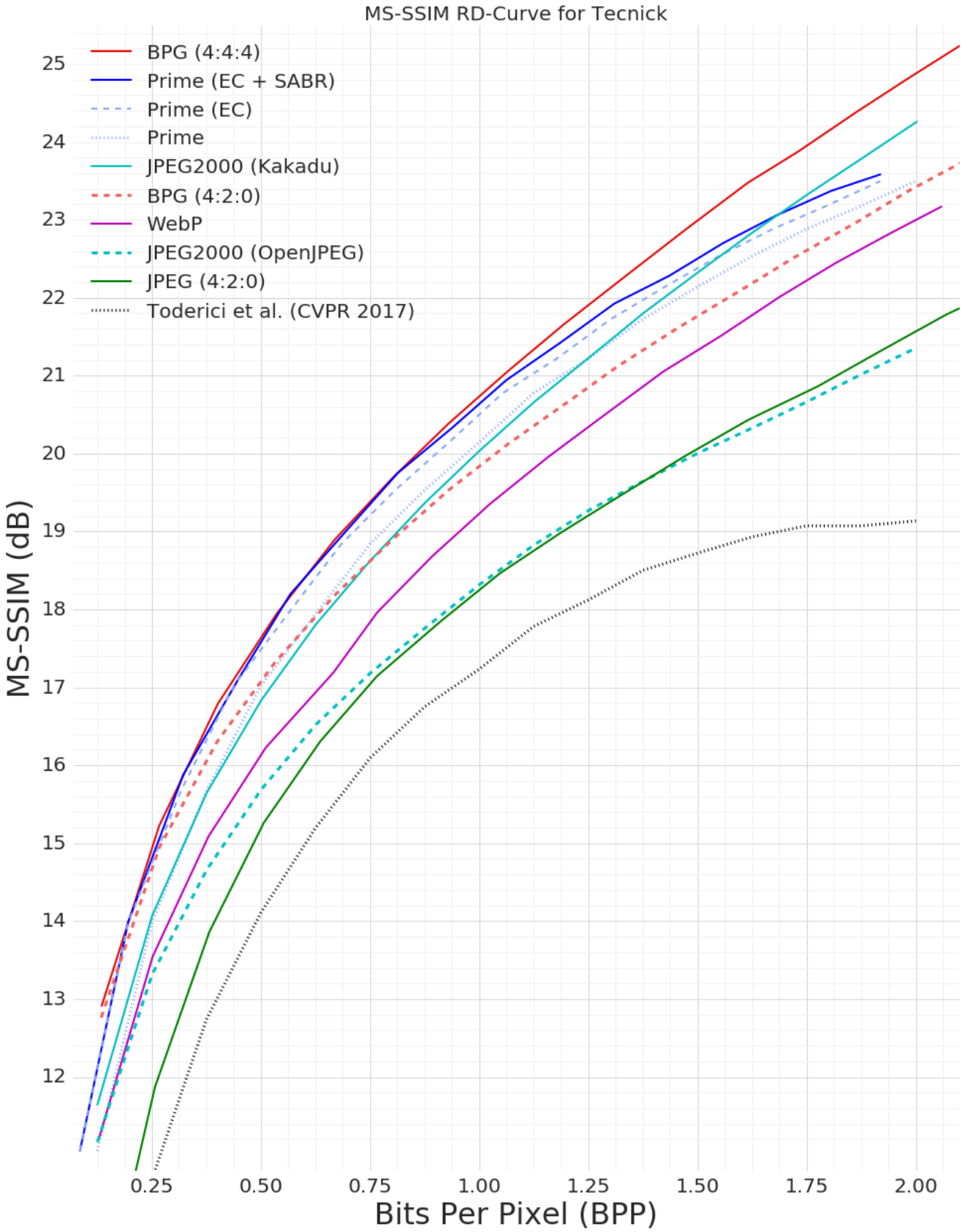

Figure 8. Rate distortion curve for MS-SSIM on Tecnick.

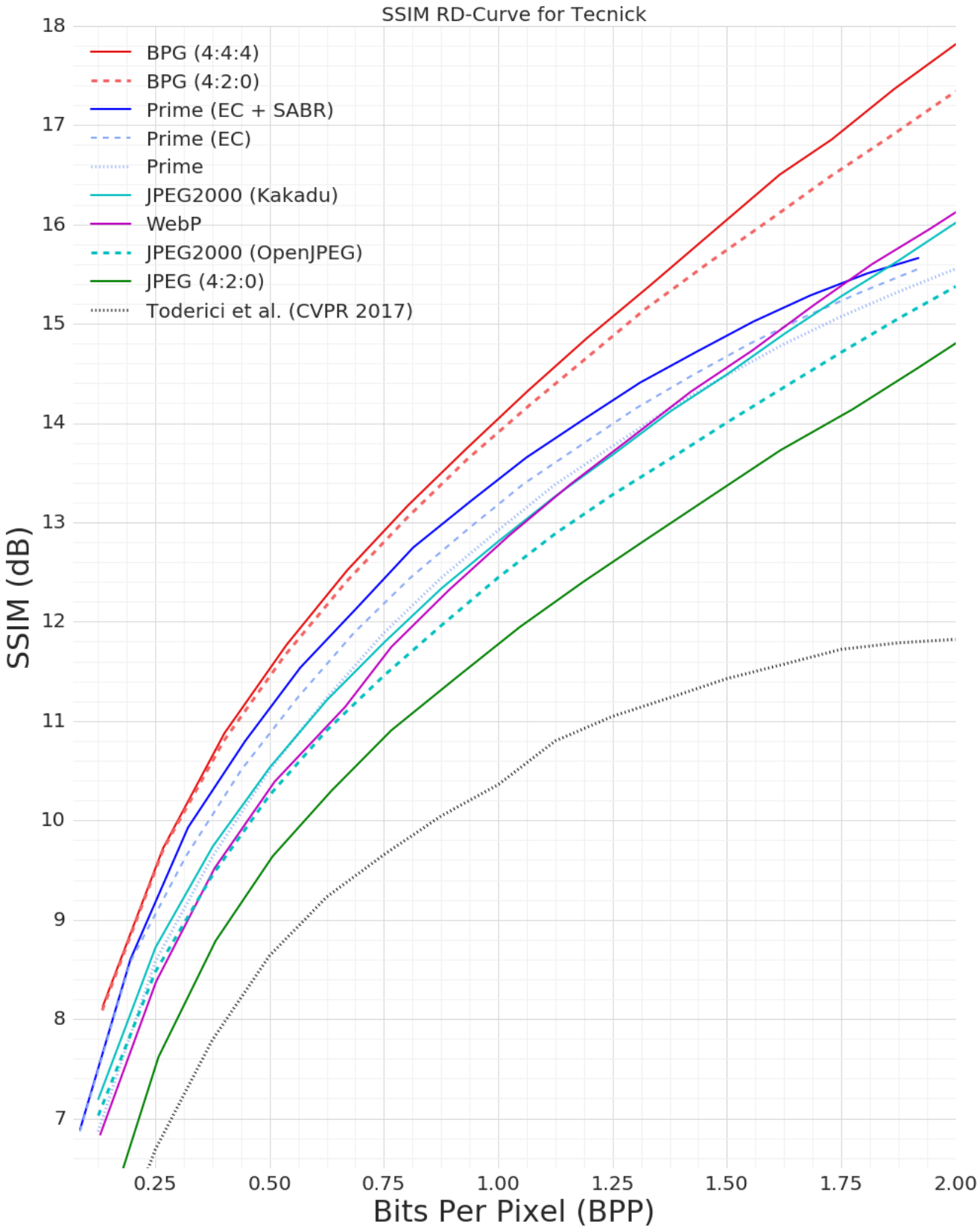

Figure 9. Rate distortion curve for SSIM on Tecnick.

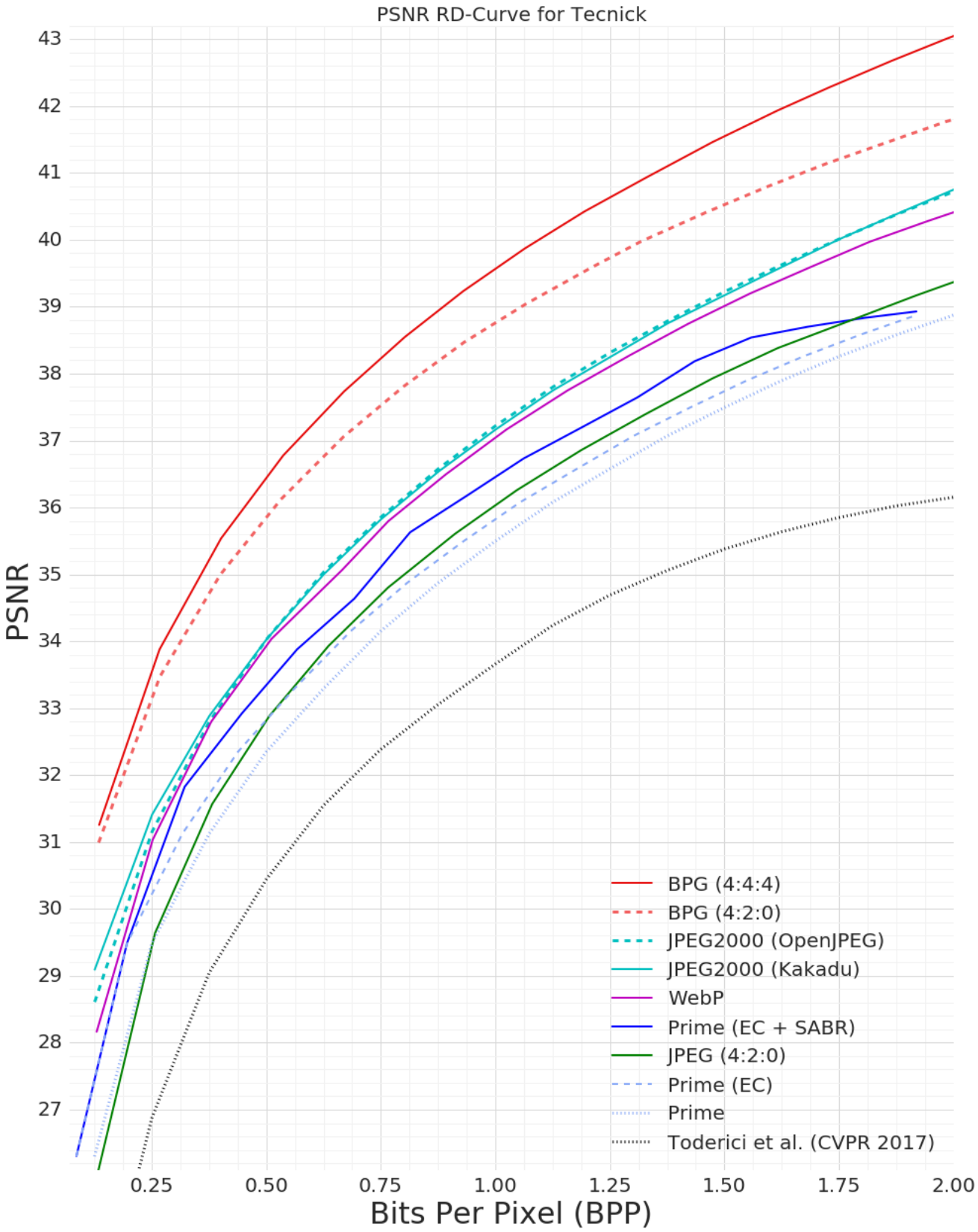

Figure 10. Rate distortion curve for PSNR on Tecnick.

JPEG2000 @ 0.502 bpp

WebP @ 0.504 bpp

BPG @ 0.504 bpp

Our Best @ 0.485 bpp

Figure 11. Compression results for Kodak image 24 near $\frac{1}{2}$ bpp. Note that BPG and WebP use nearly 4% more bandwidth than our Best, in this comparison. For the most visible differences, consider the handrail and hanging light (in front of the dark wood) on the right; the mural of the sun in the middle; the fencing at the bottom; the forest texture near the top; and the silhouetted tree in the upper left.

JPEG2000 @ 0.250 bpp

WebP @ 0.252 bpp

BPG @ 0.293 bpp

Our Best @ 0.234 bpp

Figure 12. Compression results for Kodak image 1 near $\frac{1}{4}$ bpp. Note that BPG uses more than 25% more bandwidth than our Best, in this comparison. For the most visible differences, consider the cross bar on the door; the texture on the stones and window shutters; and the weeds in front of the building.

JPEG2000 @ 0.250 bpp

WebP @ 0.251 bpp

BPG @ 0.251 bpp

Our Best @ 0.233 bpp

Figure 13. Compression results for Tecnick image 30 near $\frac{1}{4}$ bpp. Note that BPG (and WebP) use nearly 8% more bandwith than our Best, in this comparison. For the most visible differences, consider the outlines of the oranges; the orange-crate edges on the left; the color of the Clementine oranges at the bottom; and the transition from the Clementine oranges to the navel oranges at the bottom right.

JPEG2000 @ 0.125 bpp

WebP @ 0.174 bpp

BPG @ 0.131 bpp

Our Best @ 0.122 bpp

Figure 14. Compression results for Tecnick image 90 near $\frac{1}{8}$ bpp. Note that BPG uses more than 7% more bandwith (and WebP uses more than 42% more bandwidth) than our Best, in this comparison. For the most visible differences, consider the text; the eagle in the upper left; the outer edges of the mosaic tiles; and the texture in the concrete near the top.

JPEG2000 @ 0.125 bpp

WebP @ 0.131 bpp

BPG @ 0.125 bpp

Our Best @ 0.110 bpp

Figure 15. Compression results for Tecnick image 32 near $\frac{1}{8}$ bpp. Note that BPG uses nearly 14% more bandwith (and WebP uses more than 19% more bandwidth) than our Best, in this comparison. For the most visible differences, consider the pan edge; the plate rim; the colors in the coaster on the left; and the texture in the placemat, especially near the spatula, in the lower right.