Supplementary Material to "Variational Bayesian Quantization"

Yibo Yang [*1] Robert Bamler [*1] Stephan Mandt [1]

This document provides details of the proposed compression method (Section S1), model and experiment details (Section S2), and additional examples of compressed images (Section S3).

S1. Delimitation Overhead

We elaborate on the "encoding" paragraph of Section 3.3 of the main paper. After finding a quantized code point $\hat{\xi}_i$ for each dimension $i \in \{1, \ldots, K\}$ of the latent space, these code points have to be losslessly encoded into a single bitstring for transmission or storage. We experimented with two encoding schemes, described in Subsections S1.1 and S1.2 below. Subsection S1.3 provides further analysis.

S1.1. Encoding via Concatenation

We first describe an encoding scheme that we did not end up using, but that makes it easier to understand the objective function of VBQ (Eq. 8 of the main text). This encoding scheme concatenates the binary representations of $\hat{\xi}_i$ (Eq. 4 of the main text) for all $i \in \{1, \ldots, K\}$ in to a single bitstring. As each dimension i contributes $\mathcal{R}(\hat{\xi}_i)$ bits to the concatenated bitstring, this encoding scheme justifies the rate penalty term "$\lambda\mathcal{R}(\hat{\xi}_i)$" in Eq. 4 of the main text.

One also has to transmit the rates $\mathcal{R}(\hat{\xi}_i)$ (in compressed form using traditional entropy coding) so that the decoder can split the concatenated bitstring at the correct positions. While this incurs some overhead, the variable-bitlength representation of $\hat{\xi}_i$ also saves one bit per dimension i because the last bit in the binary representation of each $\hat{\xi}_i$ does not need to be transmitted as it is always equal to one (otherwise, the optimization algorithm in VBQ would favor an equivalent shorter binary representation of $\hat{\xi}_i$).

[*]Equal contribution [1]Department of Computer Science, University of California, Irvine. Correspondence to: Yibo Yang <yibo.yang@uci.edu>, Robert Bamler <rbamler@uci.edu>.

S1.2. Encoding via Standard Entropy Coding

The actual encoding scheme we ended up using does not deal with the binary representation of each $\hat{\xi}_i$ explicitly. Instead, we treat each $\hat{\xi}_i$ as a discrete symbol and directly encode the sequence $(\hat{\xi}_i)_{i=1}^K$ of symbols via entropy coding (e.g., standard arithmetic coding). The entropy coder needs a model of the probability $p(\hat{\xi}_i)$ of each symbol. For model compression, we use the empirical frequencies, which we transmit as extra header information that counts towards the total bitrate. For data compression, we estimate the frequencies on training data and include them in the decoder.

Transmitting the empirical frequencies lead to a negligible overhead in the word embeddings experiment. Only a few hundred code points (depending on λ) had nonzero frequencies, so that the compressed file size was dominated by the encoding of $K = Vd = 10^7$ quantized latent variables.

S1.3. Justification of the Rate Penalty Term $\lambda\mathcal{R}(\hat{\xi}_i)$

All experimental results are reported with the encoding scheme of Section S1.2 as it lead to slightly lower bitrates in practice. A peculiarity of this encoding scheme is that it ignores the length $\mathcal{R}(\hat{\xi}_i)$ of the binary representation of each $\hat{\xi}_i$. For a sequence of symbols $\hat{\boldsymbol{\xi}} \equiv (\hat{\xi}_i)_{i=1}^K$ with an i.i.d. entropy model $p(\hat{\xi}_i)$, an optimal entropy coder (such as arithmetic coding) achieves the total bitrate $\mathcal{R}(\hat{\boldsymbol{\xi}}) = \lceil h(\hat{\boldsymbol{\xi}}) \rceil$ with the information content

$$h(\hat{\boldsymbol{\xi}}) = \sum_{i=1}^{K} h(\hat{\xi}_i) = -\sum_{i=1}^{K} \log_2 p(\hat{\xi}_i). \tag{S1}$$

In particular, Eq. S1 does not depend on $\mathcal{R}(\hat{\xi}_i)$. This poses the question whether the rate penalty term "$\lambda\mathcal{R}(\hat{\xi}_i)$" in the VBQ objective (Eq. 8 of the main text) is justified. Ideally, the algorithm would minimize $\lambda h(\hat{\xi}_i) = -\lambda \log_2 p(\hat{\xi}_i)$ instead, but this quantity is unknown until the quantizations $(\hat{\xi}_i)_{i=1}^K$ and therefore the empirical frequencies $p(\hat{\xi}_i)$ are obtained. Our experiments suggest that $\mathcal{R}(\hat{\xi}_i)$ is a useful proxy for the eventual value of $h(\hat{\xi}_i)$.

Figure S1 plots the rate estimate $\mathcal{R}(\hat{\xi}_i)$, i.e., the integer number of bits in the binary representation of $\hat{\xi}_i$ (x-axis) against

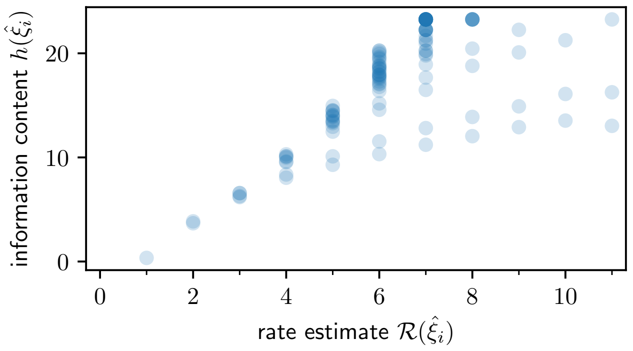


Figure S1. Relation between rate estimate $\mathcal{R}(\hat{\xi}_i)$ and the actual contribution $h(\hat{\xi}_i)$ of code point $\hat{\xi}_i$ to the total bitrate under entropy coding. The approximate affine linear relationship justifies minimizing $\mathcal{R}(\hat{\xi}_i)$ as a proxy for $h(\hat{\xi}_i)$ in VBQ.

the actual contribution $h(\hat{\xi}_i) = -\log_2 p(\hat{\xi}_i)$ to the total bitrate according to Eq. S1 (y-axis). The figure shows experimental data for compressed word embeddings at 1.32 bits per latent dimension. We make the following observations:

- For most code points $\hat{\xi}_i$, the dependency between $h(\hat{\xi}_i)$ and $\mathcal{R}(\hat{\xi}_i)$ can be approximated by an affine linear function, thus justifying the use of $\mathcal{R}(\hat{\xi}_i)$ in the optimization of Bayesian AC.

- The slope of the approximate linear dependency is larger than one. This may be understood by the penalty term $\lambda\mathcal{R}(\hat{\xi}_i)$ in the objective function (Eq. 8 of the main text), which causes the method to avoid code points $\hat{\xi}_i$ with large rate estimates $\mathcal{R}(\hat{\xi}_i)$, thus reducing their empirical frequencies $p(\hat{\xi}_i)$ and increasing their information content $h(\hat{\xi}_i) = -\log_2 p(\hat{\xi}_i)$. This observation does not invalidate the use of $\mathcal{R}(\hat{\xi}_i)$ as an estimate for $h(\hat{\xi}_i)$ since the different slope can be absorbed in a rescaling of the parameter λ

- For rates $\mathcal{R}(\hat{\xi}_i) \geq 4$, there are two code points for each rate with considerably lower information content. These code points correspond to the two extremes for each rate, i.e., $\hat{\xi}_i$ closest to zero or one, respectively. The observation that the two extremes have lower information content (i.e., higher empirical frequencies) can be explained by the fact that the empirical prior distribution whose CDF we use to map latent variables z_i to quantiles ξ_i does not fully capture the true distribution of variational means. Indeed, experiments with a more long tailed empirical prior distribution lead to marginally better performance, but the simplicity of a Gaussian empirical prior seemed more valuable to us.

S2. More Experimental Details

S2.1. Word Embeddings

The word embeddings experiment involved only minimal hyperparameter tuning, and we only optimized for performance of the uncompressed model since the goal of the experiment was to test the proposed compression method on a model that was not tuned for compression. We trained for 10^5 iterations with minibatches of 10^4 randomly drawn words and contexts due to hardware constraints. We tried learning rates 0.1 and 1 and chose 0.1.

S2.2. Experiments on Images

As mentioned in the main text, we used regular VAEs in the image experiments with standard normal prior $p(\mathbf{z})$ and factorized normal posterior $q(\mathbf{z}|\mathbf{x})$ with diagonal covariance.

S2.3. MNIST

The VAE’s inference network has two convolutional layers followed by a fully connected layer. The two conv layers use 32 and 64 filters respectively, with kernel size 3, stride size 2, and ReLU activation. The fully connected layer has output dimension 10 so that $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ of $q(\mathbf{z}|\mathbf{x})$ each has dimension 5.

The generative network architecture mirrors the inference network but in reverse, starting with a dense layer mapping 5 dimensional latent variables to 1568 dimensional, treated as 32-channel 7x7 activations, and followed by two deconvolutional layers of 64 and 32 filters (with identical padding and stride as the convolutional layers). The output is deconvolved with a single 3x3 filter with sigmoid activation function. For each pixel, the (scalar) output of the last layer parameterizes the likelihood of the pixel being white.

We trained the network on binarized MNIST images for 100 epochs, using the Adam optimizer with learning rate 10^{-4}.

S2.4. Frey Faces

On the Frey Faces dataset, we observe poor reconstruction quality by training on binarized images with a factorized Bernoulli likelihood model; instead, we treat each pixel as an observation from a factorized categorical likelihood model with 256 possible outcomes.

The VAE’s inference network has two layers. The first layer flattens the input image, converts each pixel value in $\{0, 1, ..., 255\}$ into a one-hot vector $\in \mathbb{R}^{256}$, and uses it to index a 128-dimensional dense vector. The second layer flattens the result of the first layer as its input (which has dimensionality equal to 128 $\times$ *number of pixels*), and fully connects its input to 8 hidden units. The final output is split to obtain 4-dimensional $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}^2$ of $q(\mathbf{z}|\mathbf{x})$.

The generative network has two fully connected layers. The first layer uses 4 hidden units and ReLU activation; the second layer uses 256 $\times$ *number of pixels* hidden units, and takes a 256-way softmax to compute the categorical probability of of each pixel value taking value in $\{0, 1, ..., 255\}$.

We obtained the Frey Faces images from https://cs.nyu.edu/~roweis/data.html. We trained on a random subset of 1800 images for 800 epochs, using the Adam optimizer with learning rate 10^{-4}.

On both MNIST and Frey Faces, we vary the rate-distortion trade-off parameter λ of Variational Bayesian Quantization between 10^{-5} and 10^{4}.

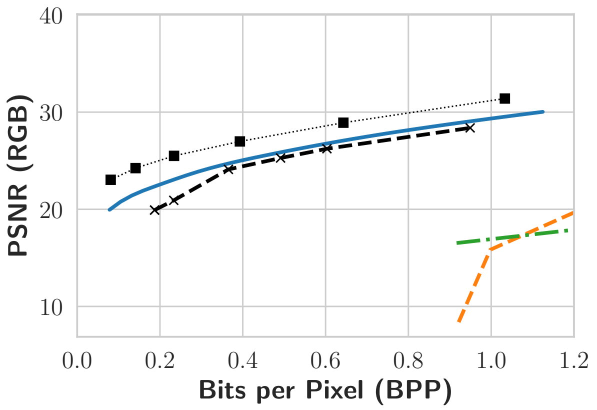
S2.5. Color Image Compression

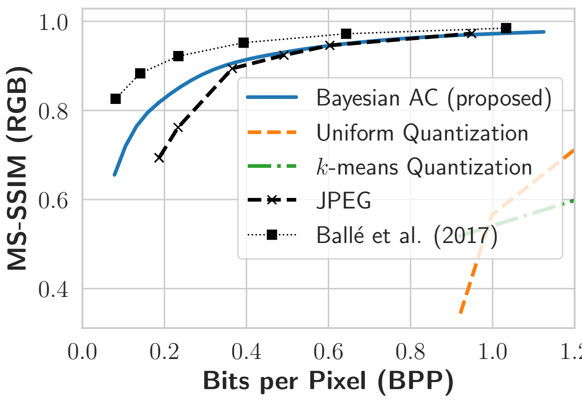
As mentioned in the main text, the VAE here uses a fully convolutional architecture with 3 layers of 256 filters each, the same as in (Balle et al., 2017); see the latter for detailed descriptions. We tuned the variance σ^2 of the likelihood model on a logarithmic grid from 10^{-4} to 0.1 and set it to 0.001. The VAE was trained on the same dataset as in (Balle et al., 2017) for 2 million steps, using Adam with learning rate 10^{-4}.

In the image compression R-D curves, λ ranges from 2^{-6} to 2^{16}. In Figure 5 of the main text, λ was set to 17.5 for VBQ to match the bitrate of the other methods. The uniform quantization result was obtained with 4 quantization levels, on a separately tuned model that had an additional convolutional layer of 64 channels. The additional conv layer was to reduce the latent dimensionality, as uniform quantization could not achieve bitrates lower than 0.5 even with only 2 grid points in the original 3-layer model.

S3. Additional Image Compression Examples

Starting on the next page, we provide detailed compression results for individual images from the Kodak dataset. For each image, we show the rate-distortion performance by various methods, followed by reconstructions using our proposed method and JPEG at equal bitrate.





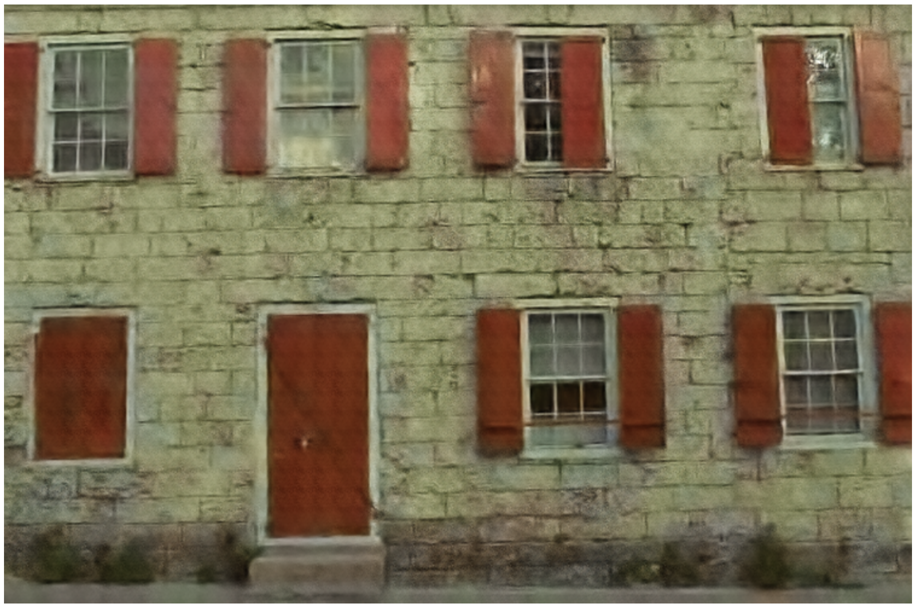


Figure S2. Proposed. bits-per-pixel: 0.27, PSNR: 23.595, MS-SSIM: 0.871

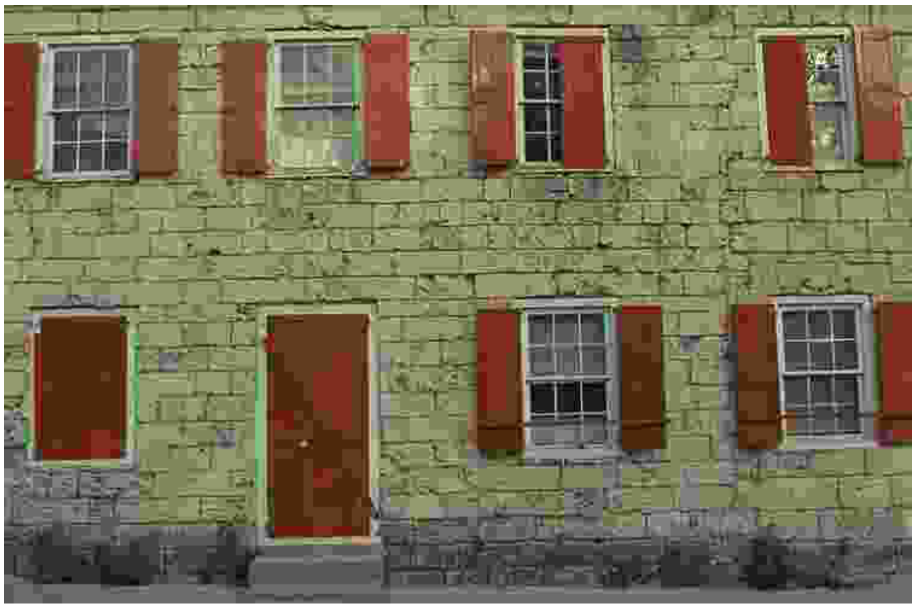
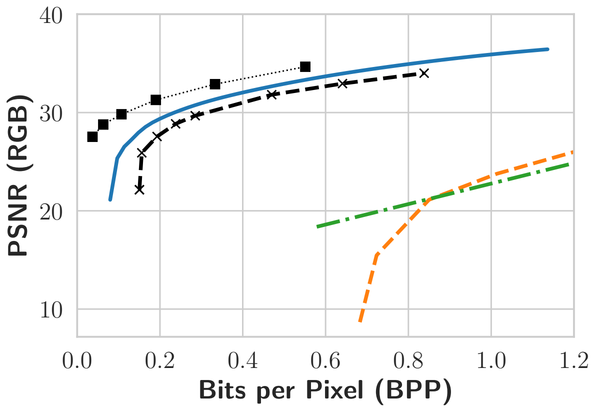
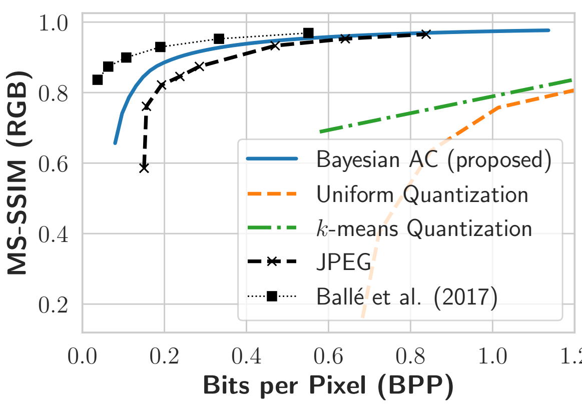


Figure S3. JPEG. bits-per-pixel: 0.27, PSNR: 22.015, MS-SSIM: 0.816





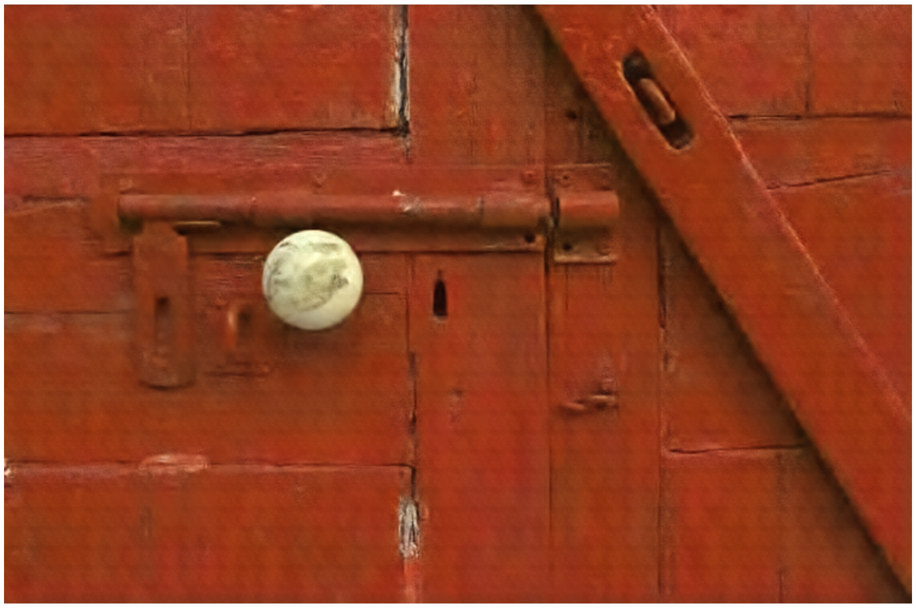


Figure S4. Proposed. bits-per-pixel: 0.19, PSNR: 29.226, MS-SSIM: 0.882

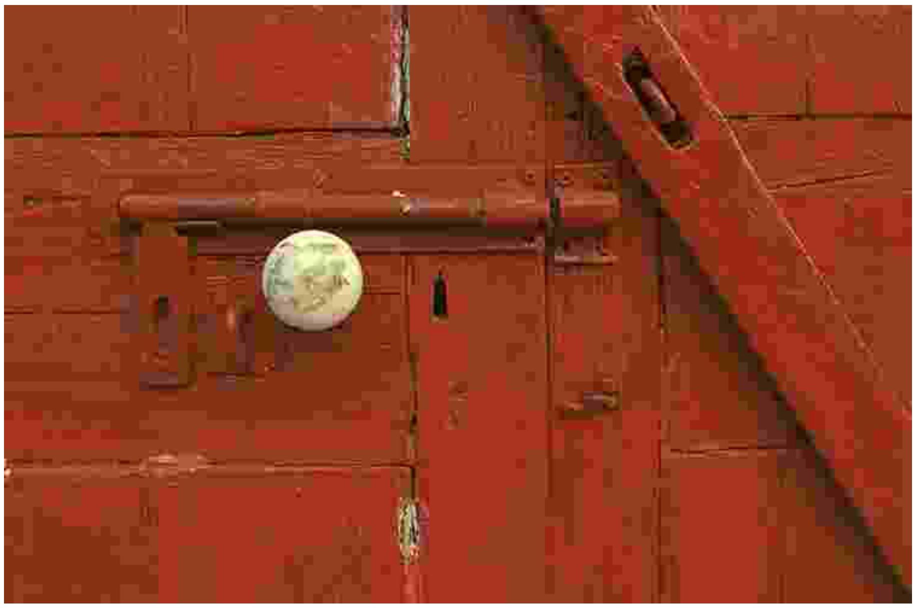
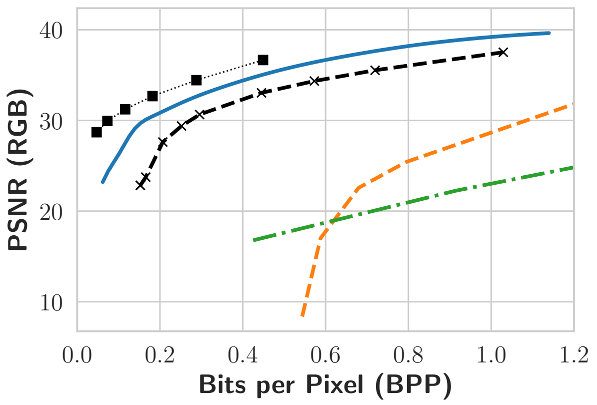
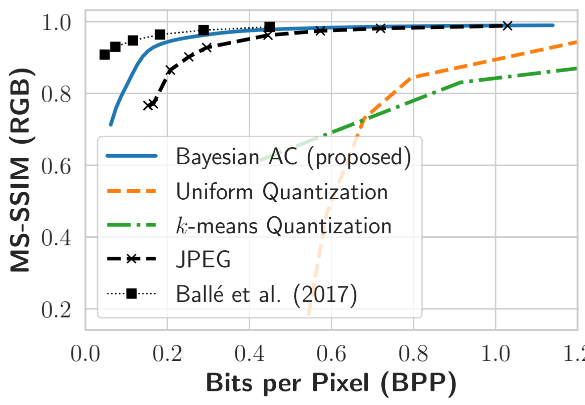


Figure S5. JPEG. bits-per-pixel: 0.19, PSNR: 26.658, MS-SSIM: 0.747





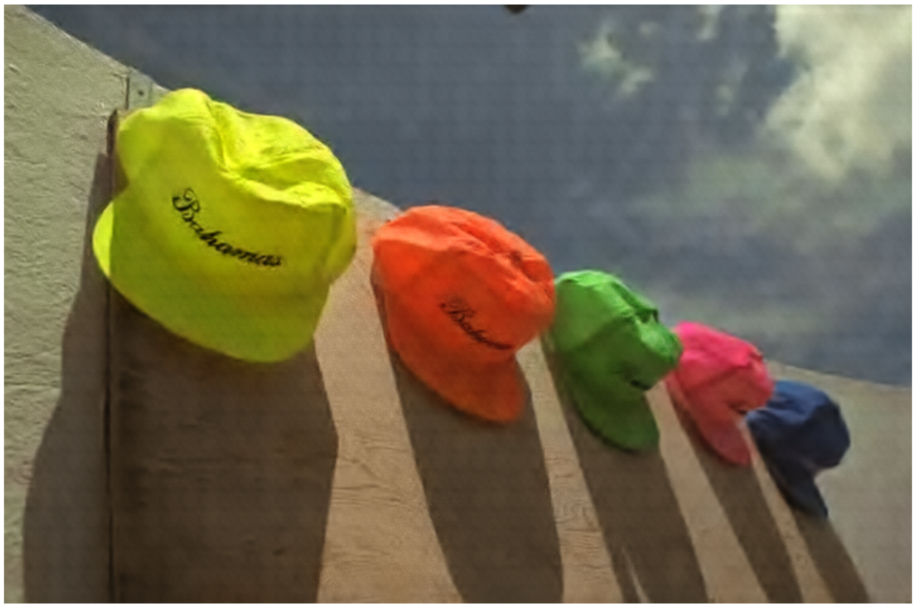


Figure S6. Proposed. bits-per-pixel: 0.19, PSNR: 30.664, MS-SSIM: 0.94

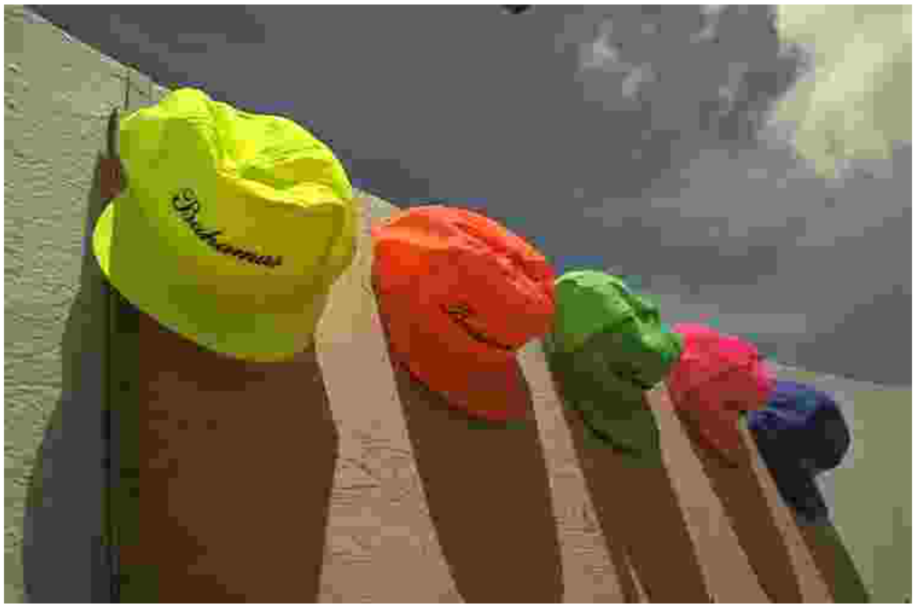
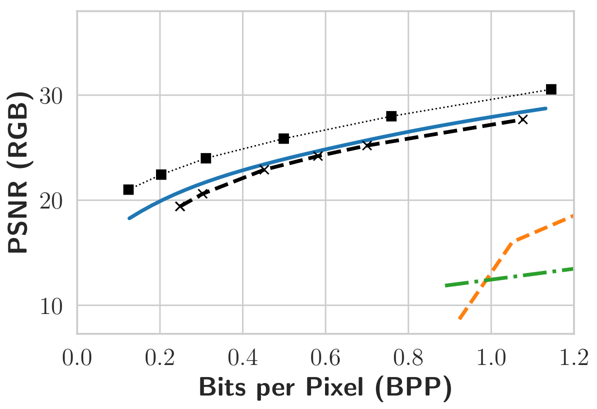
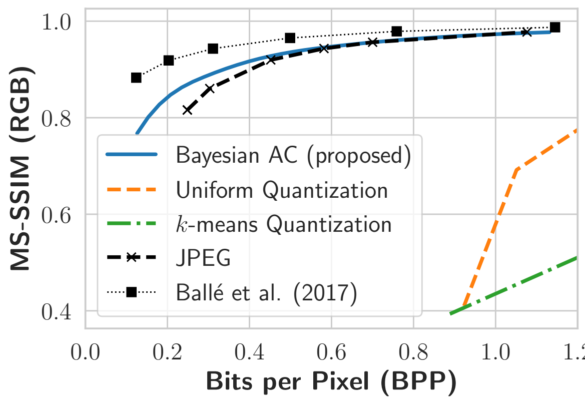


Figure S7. JPEG. bits-per-pixel: 0.19, PSNR: 26.201, MS-SSIM: 0.842





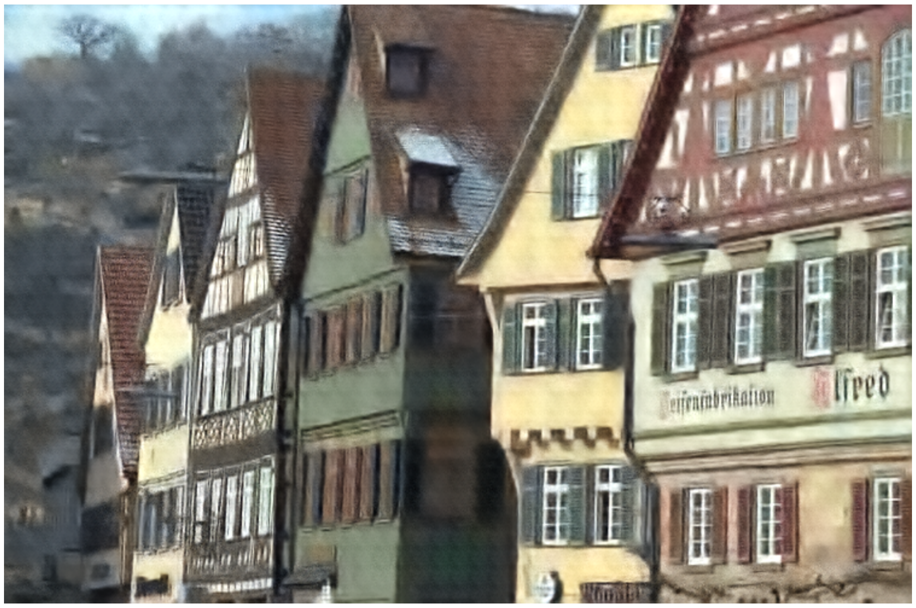


Figure S8. Proposed. bits-per-pixel: 0.34, PSNR: 22.177, MS-SSIM: 0.903

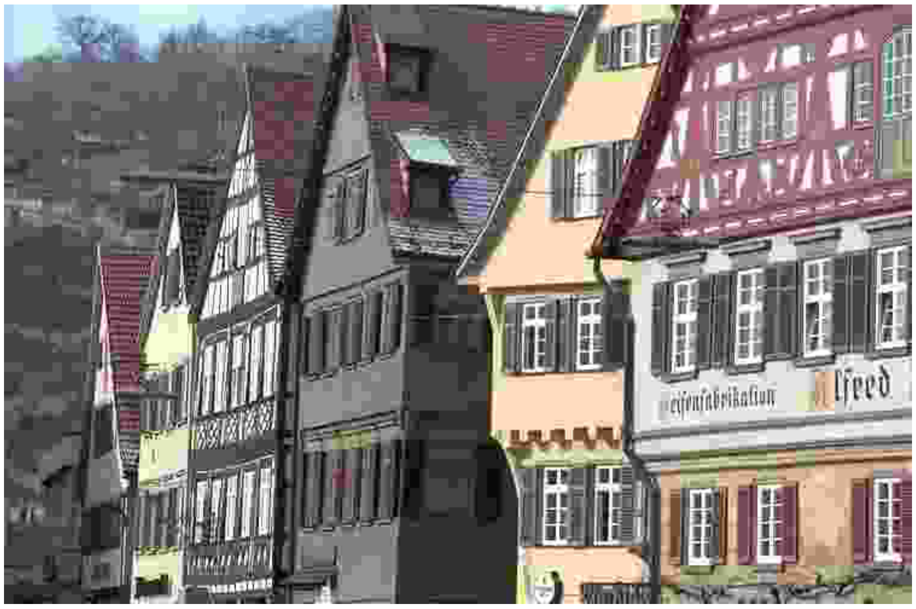
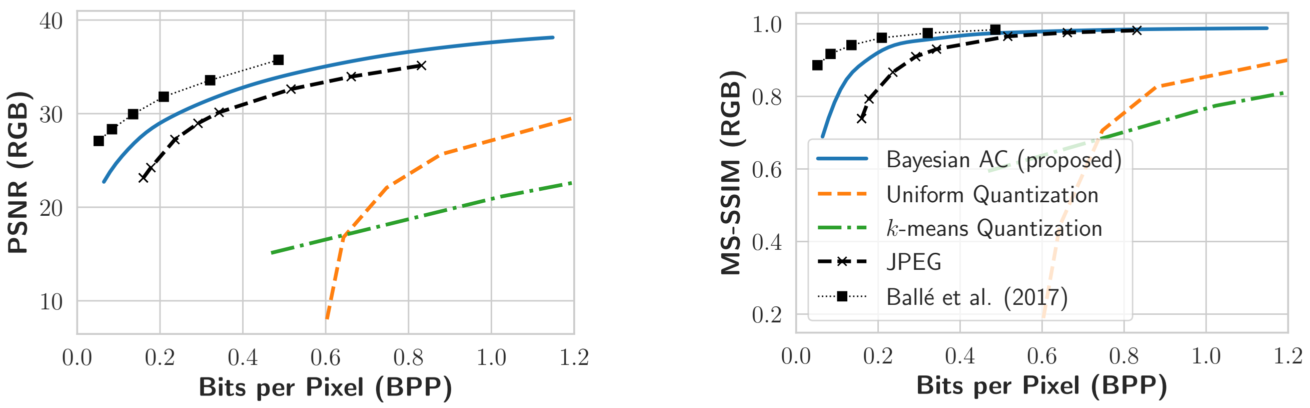
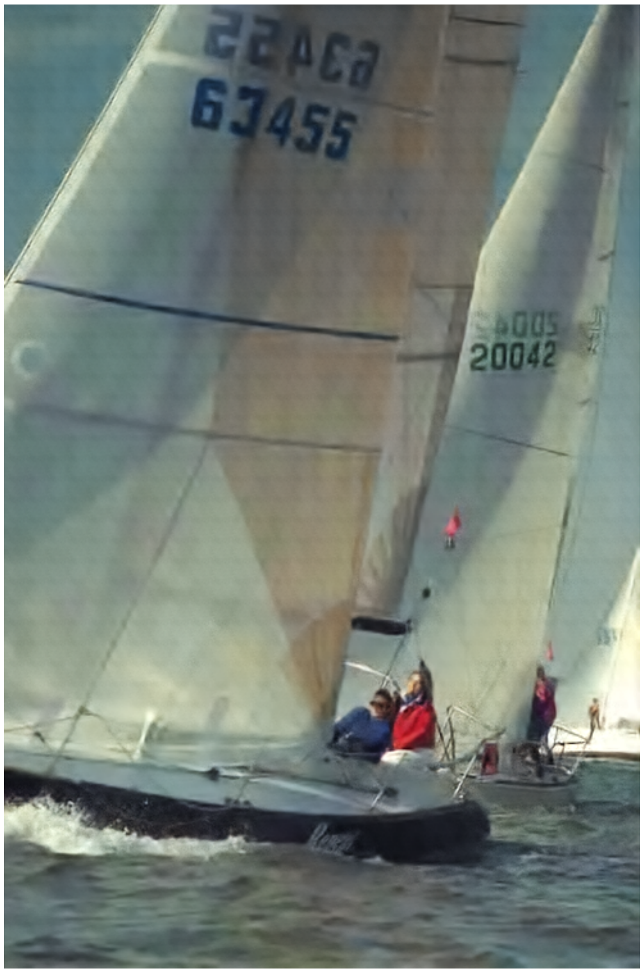
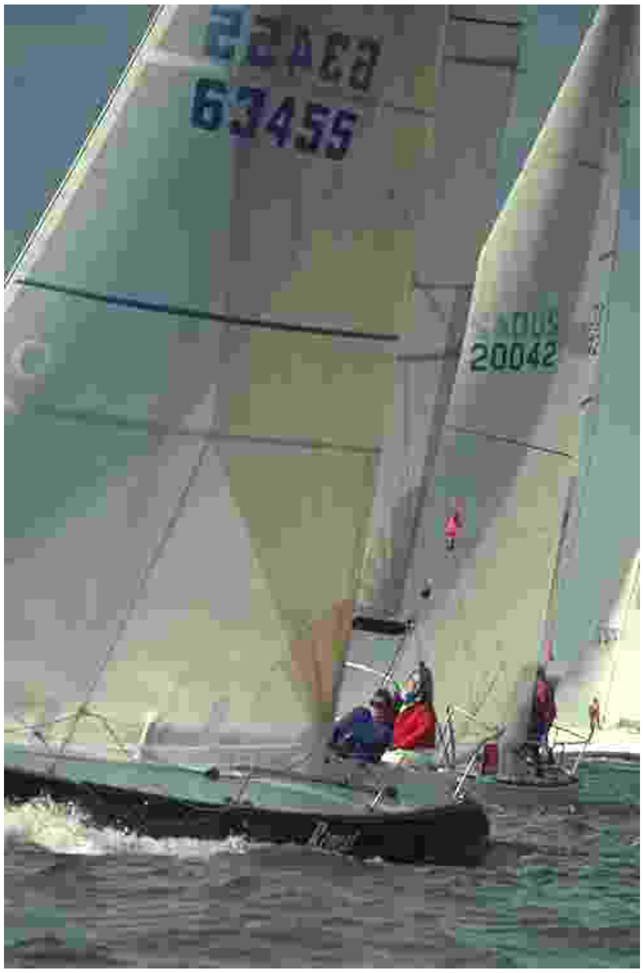
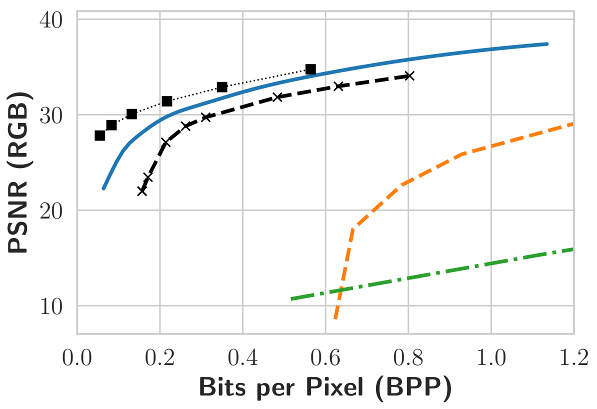


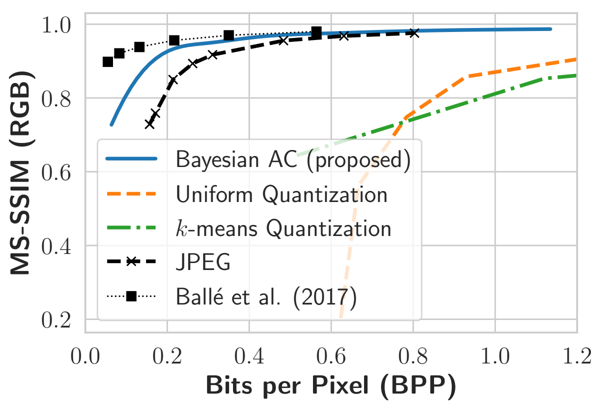
Figure S9. JPEG. bits-per-pixel: 0.34, PSNR: 21.331, MS-SSIM: 0.882




(a) Proposed. bits-per-pixel: 0.22, PSNR: 29.584, MS-SSIM: 0.935


(b) JPEG. bits-per-pixel: 0.22, PSNR: 26.543, MS-SSIM: 0.846





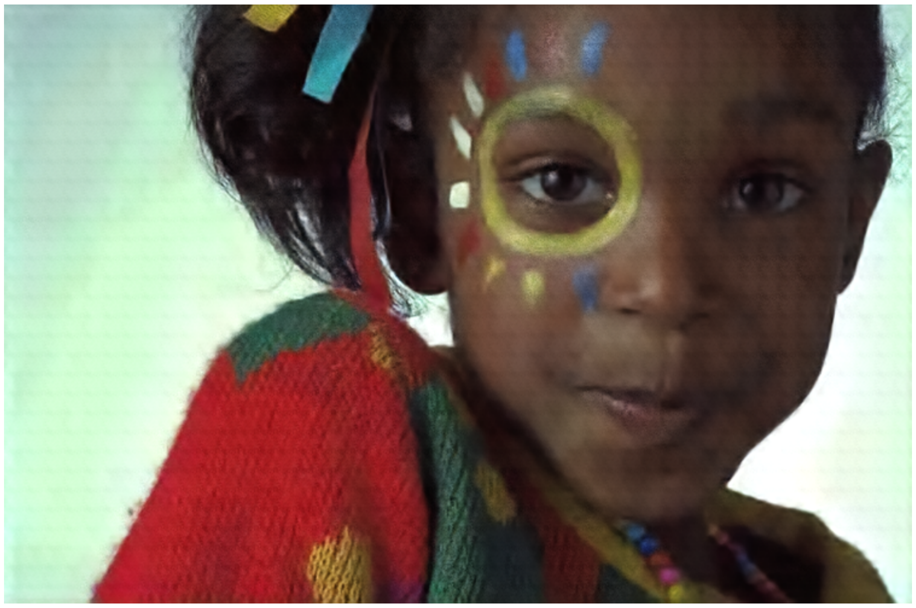


Figure S11. Proposed. bits-per-pixel: 0.2, PSNR: 29.523, MS-SSIM: 0.928

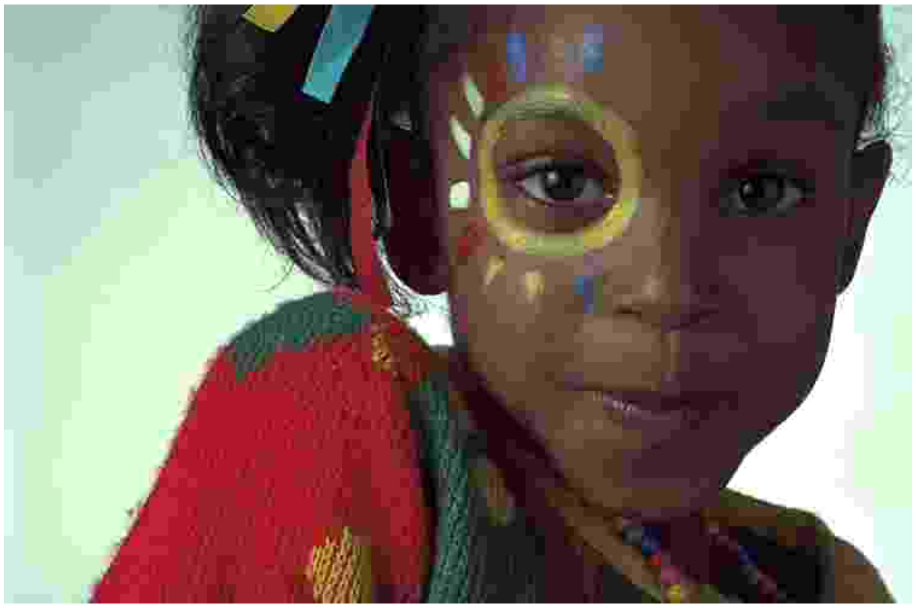
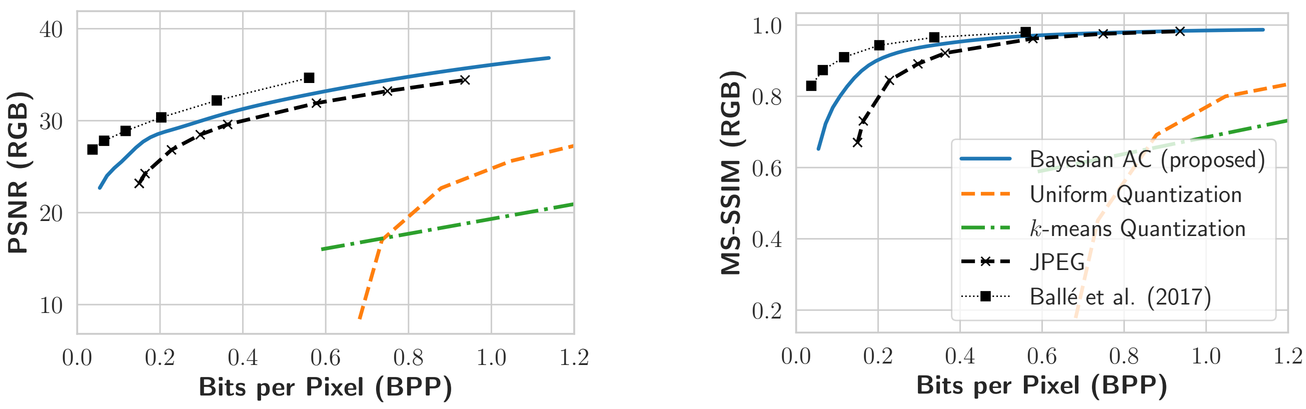


Figure S12. JPEG. bits-per-pixel: 0.2, PSNR: 26.6, MS-SSIM: 0.838



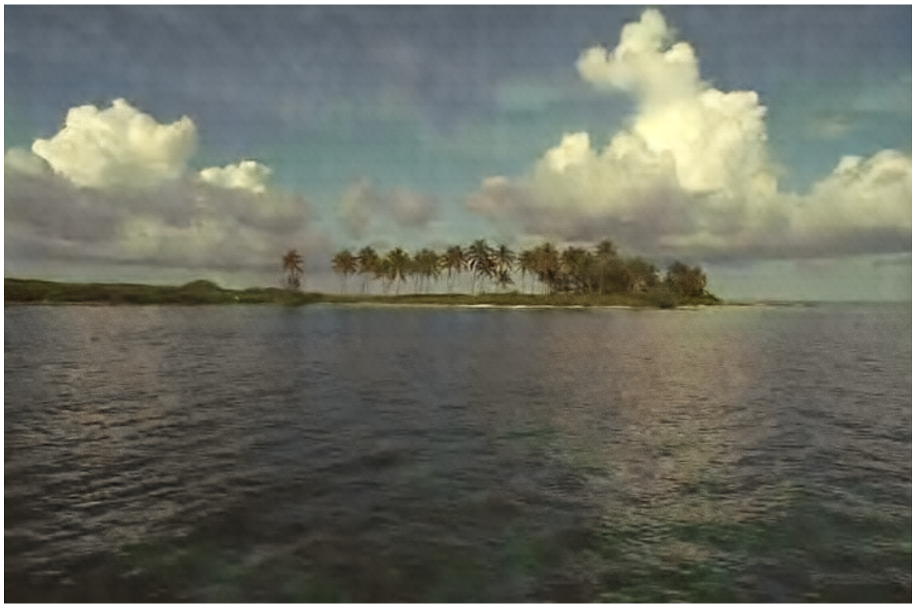


Figure S13. Proposed. bits-per-pixel: 0.21, PSNR: 28.77, MS-SSIM: 0.908

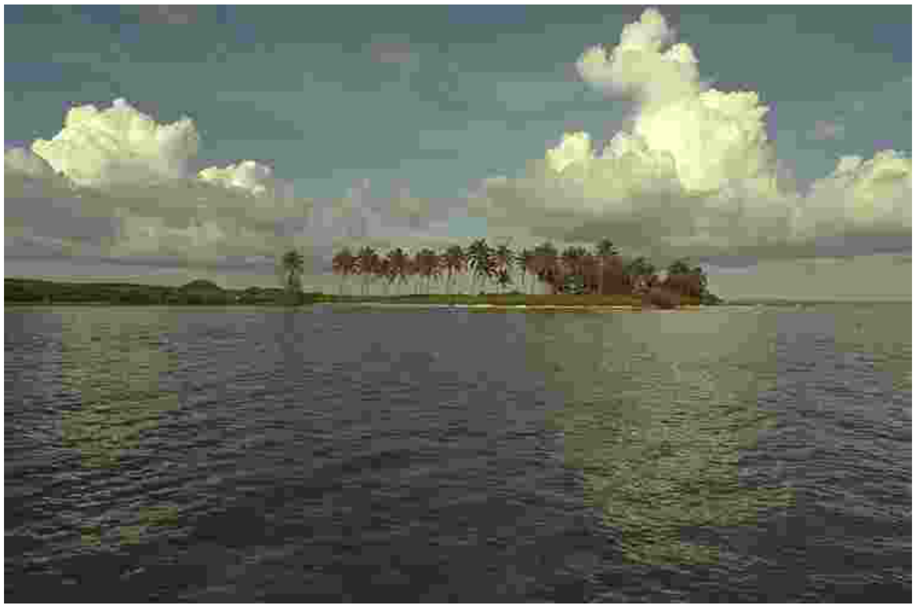
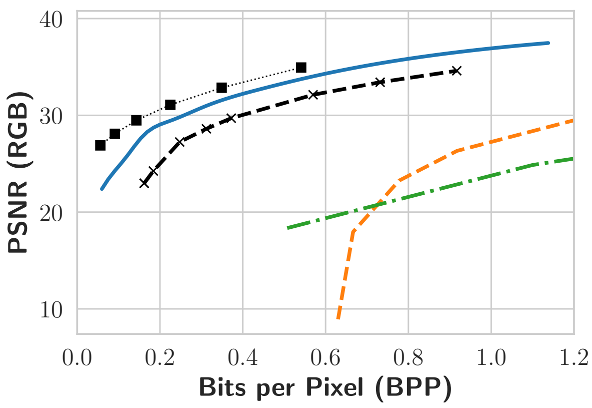
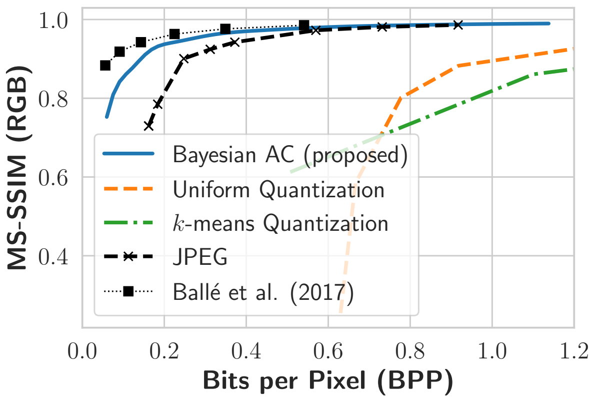
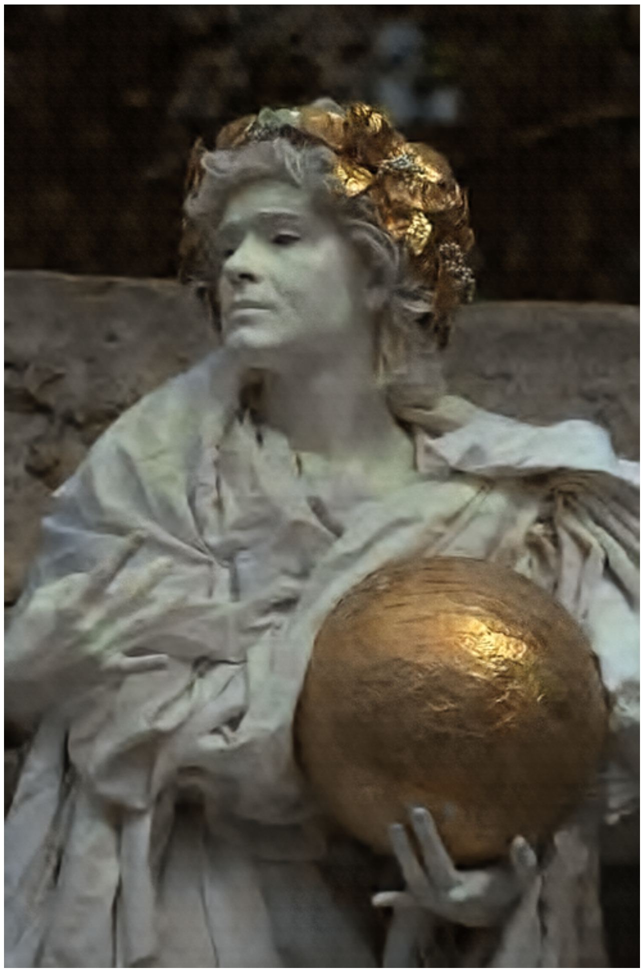
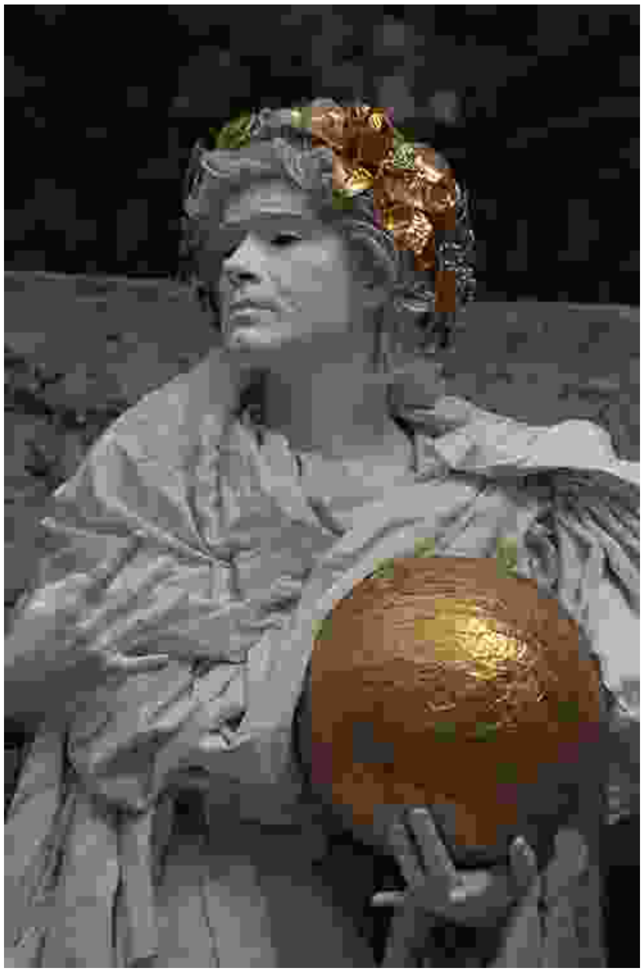


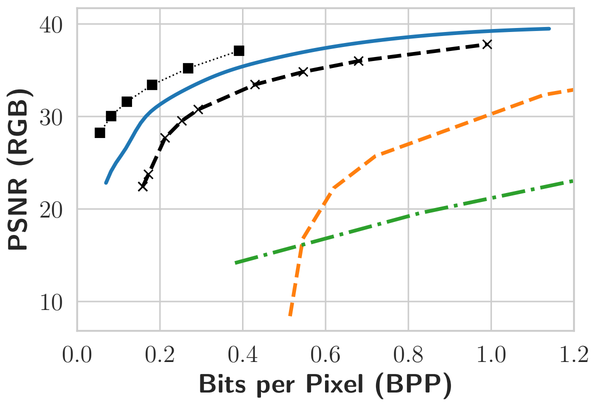
Figure S14. JPEG. bits-per-pixel: 0.21, PSNR: 26.247, MS-SSIM: 0.818

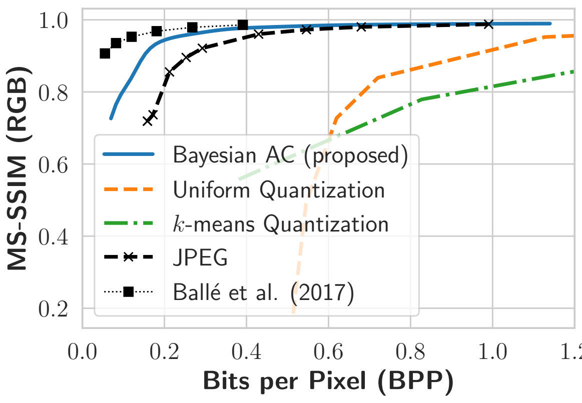





(a) Proposed. bits-per-pixel: 0.2, PSNR: 29.006, MS-SSIM: 0.936


(b) JPEG. bits-per-pixel: 0.2, PSNR: 25.389, MS-SSIM: 0.835





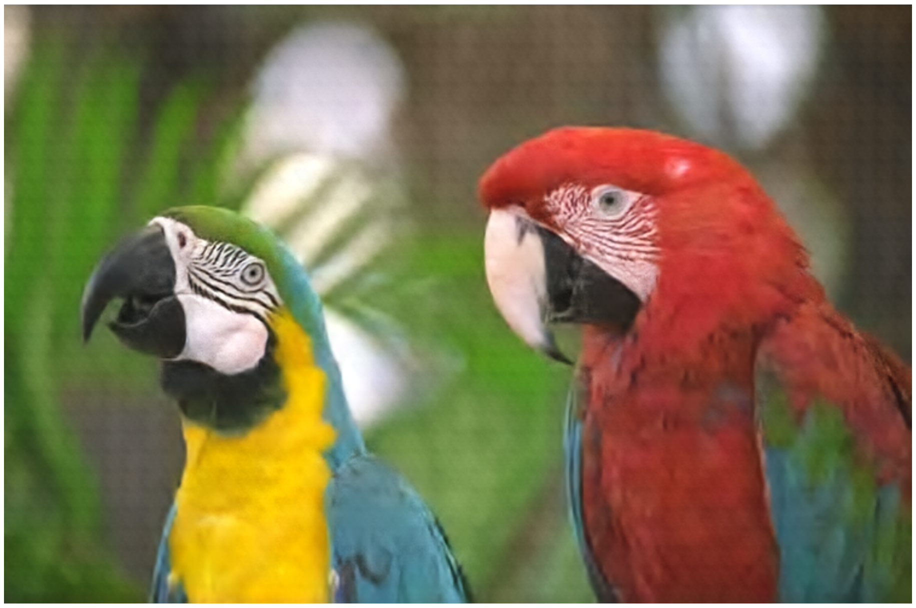


Figure S16. Proposed. bits-per-pixel: 0.2, PSNR: 31.425, MS-SSIM: 0.945

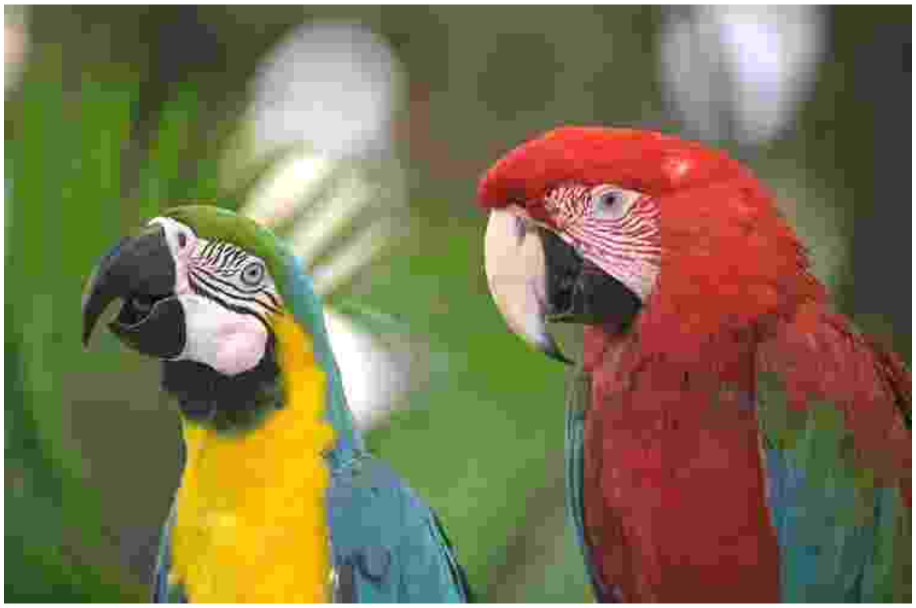
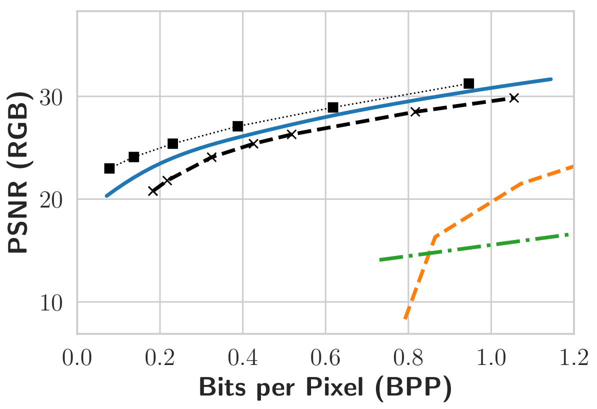
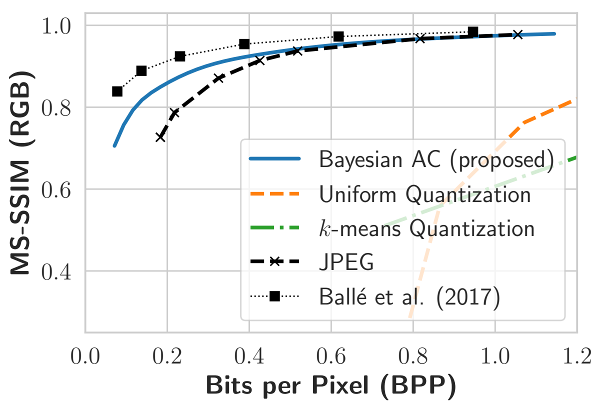


Figure S17. JPEG. bits-per-pixel: 0.2, PSNR: 26.976, MS-SSIM: 0.833





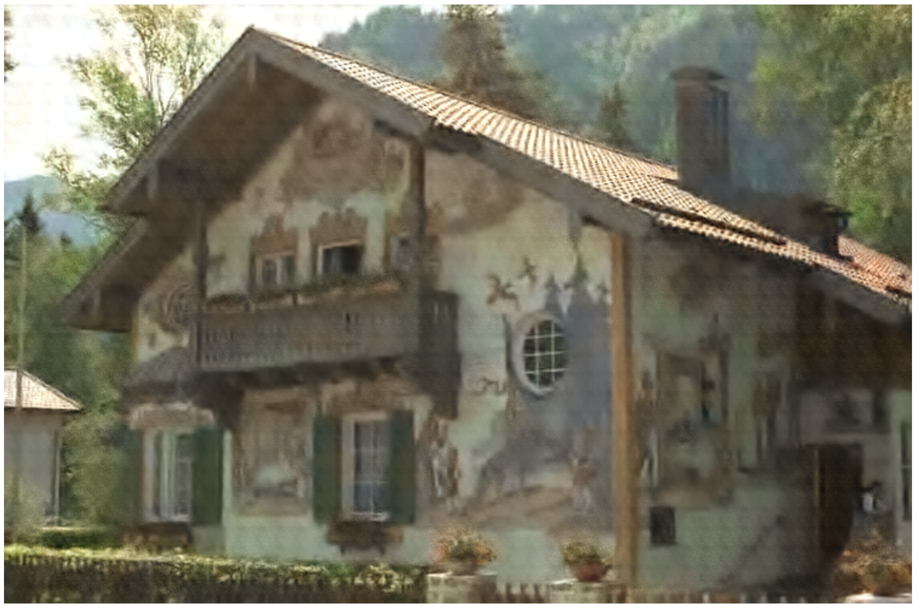


Figure S18. Proposed. bits-per-pixel: 0.24, PSNR: 24.254, MS-SSIM: 0.882

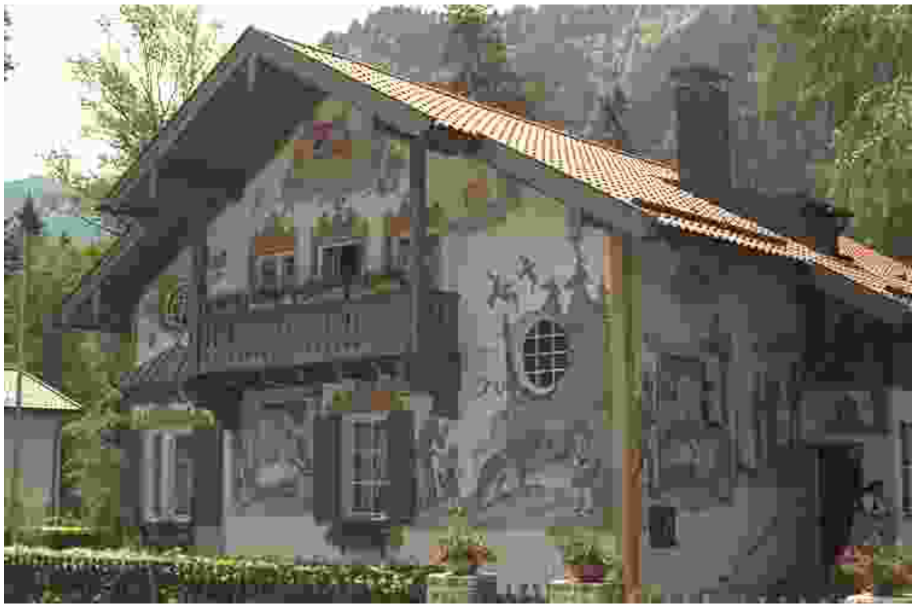


Figure S19. JPEG. bits-per-pixel: 0.24, PSNR: 22.562, MS-SSIM: 0.818