Hierarchical Modeling for Image Compression

Prakhar Srivastava[*]
University of California, Irvine
prakhs2@uci.edu

Alekhya Pyla
University of California, Irvine
apyla@uci.edu

Abstract

Neural compression is the application of neural networks and other machine learning methods to data compression. In this project, we first replicate a simple end-to-end optimized image compression paper and then propose an interesting hierarchical variant which better optimizes the entropy of the quantized latents. We finally test the methods against classical JPEG and report the rate-distortion curve along with a visual inspection of reconstruction quality.

1 Introduction

The world's dependency on multimedia communication has no end in sight. As digital gadgets become more pervasive in our daily lives, so does the need to store, send, and receive images and sounds in ever more efficient ways. Hence, there is a need for efficient data compression methods to make the load on communication channels more balanced and remove bottlenecks. Simply put, the goal of data compression is to reduce the number of bits needed to represent useful information.

Transform coding (TC) has almost invariably been the method of choice for compressing image data, mainly because it separates the task of decorrelating a data source from coding it. In theory, any source can be compressed optimally using vector quantization (VQ). However, for sources with more than a few dimensions, VQ quickly becomes computationally infeasible, owing to the fact that the codebook of reproduction vectors, as well as the computational complexity of finding the best reproduction of the source vector, grows exponentially with the number of dimensions. By first mapping the source vector into a latent space using a decorrelating invertible transform, and then individually quantizing and coding each of the latent dimensions, TC simplifies quantization and coding.

Because this assumption allows for closed-form solutions, most of the theory surrounding TC is predicated on an implicit or explicit assumption that the source is jointly Gaussian. Decorrelation is all that is required to make the latent dimensions independent if the source is Gaussian. Even if the source is not Gaussian, it is almost always assumed that the transforms are linear when discussing TC.

Determining nonlinear transforms with desirable qualities, such as increased independence between latent dimensions, is a tough challenge for high-dimensional sources, which was one of the primary constraints in constructing transform codes. As a result, there wasn't much practical research into employing nonlinear transformations for compression until few years ago. With the current emergence of artificial neural networks (ANNs), this premise has shifted. It is well known that ANNs can approximate arbitrary functions given the correct set of parameters. It turns out that a nearly universal set of tools for function approximation has evolved when stochastic optimization approaches like stochastic gradient descent (SGD) are combined with massively parallel computational resources. These technologies have also been applied to the compression of data. Despite the fact that these approaches were created from the ground up, they have quickly caught up to modern conventional compression methods like HEVC, which are the result of decades of incremental engineering efforts. This highlights the ease and convenience of prototyping that universal function approximation provides over constructing methods manually, as well as the power of developing methods in a data-driven manner, as it has in other domains.

In this project, we explore end-to-end trainable models for image compression based on variational autoencoders. More specifically, we start off with a baseline model akin to (Ballé et al., 2016) and compare it against a hierarchical variant. We evaluate their performance using Rate-Distortion Curve (more in Section 2) and finally analyze these models in the low bitrate regime qualitatively.

[*]report author

2 Background

All neural compression algorithms work on the same basic principle as outlined in Figure 1: the encoder transforms the image vector $\boldsymbol{x}$ using a generic parameterized analysis transform $g_a\left(\boldsymbol{x};\boldsymbol{\phi}_g\right)$ into a latent representation $\boldsymbol{y}$, which is then quantized to form $\hat{\boldsymbol{y}}$. Since $\hat{\boldsymbol{y}}$ is discrete valued, it can be losslessly compressed using entropy coding schemes like arithmetic coding (Rissanen and Langdon, 1981) and transmitted as a sequence of bits. On the other side, the decoder recovers $\hat{\boldsymbol{y}}$ from the compressed signal, and subjects it to a generic parametrized synthesis transform $g_s\left(\boldsymbol{x};\boldsymbol{\theta}_g\right)$ to recover the reconstructed image $\hat{\boldsymbol{x}}$.

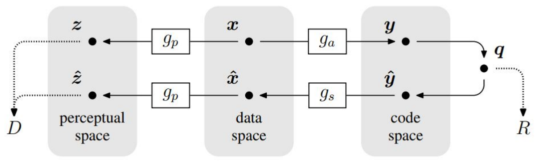


Figure 1: General nonlinear transform coding framework. In our project, g_p is simply an identity transform, meaning that we measure distortion directly in the image space.

The quantization introduces error, which is tolerated in the context of lossy compression, giving rise to a *rate–distortion* optimization problem. Rate is the expected code length (bit rate) of the compressed representation: assuming the entropy coding technique is operating efficiently, this can again be written as a cross entropy $R = \mathbb{E}_{\boldsymbol{x}\sim p_x}\left[-\log_2 p_{\hat{\boldsymbol{y}}}\left(Q\left(g_a\left(\boldsymbol{x};\boldsymbol{\phi}_g\right)\right)\right)\right]$ where Q represents the quantization function and $p_{\hat{\boldsymbol{y}}}$ is the entropy model. Distortion, on the other hand, is the expected difference between the reconstruction $\hat{\boldsymbol{x}}$ and the original image $\boldsymbol{x}$.

2.1 Quantization approximation

Quantization poses a problem for optimization because gradients with respect to $\boldsymbol{\phi}_g$ are zero almost everywhere. Hence, we need a continuous relaxation of Q to work with such that we can use gradient descent methods to optimize the parameters $\boldsymbol{\phi}_g$ and $\boldsymbol{\theta}_g$ jointly. Different optimization techniques have been explored like creating proxy gradients for the quantizer (Theis et al., 2017) and substituting the quantizer with additive uniform noise during the training process (Ballé et al., 2016). For the latter method, the model switches to true quantization when deploying the model as a compression method. For the rest of this report, *noisy* representation of this latent space will be referred as $\tilde{\boldsymbol{y}}$.

Independent uniform noise is a very common approximation model of quantization error (Gray and Neuhoff, 1998). Given an index i running over all spatial and channel locations, hard quantization $\hat{\boldsymbol{y}}_i = \boldsymbol{q}_i = \text{round}\left(\boldsymbol{y}_i\right)$. The marginal density of $\hat{\boldsymbol{y}}_i$ is then given by a train of Dirac-Delta functions with weights $P_{\boldsymbol{q}_i}(n) = \int_{n-\frac{1}{2}}^{n+\frac{1}{2}} p_{\boldsymbol{y}_i}(t)\,dt$. Continuous relaxation of this is $\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{\Delta y}$ where $\boldsymbol{\Delta y}$ is an additive i.i.d. uniform noise with same width as that of the quantization bins. This relaxation has a desirable property that $p_{\tilde{\boldsymbol{y}}}(n) = P_{\boldsymbol{q}}(n)$. Figure 2 clearly denotes the distinctions and similarities between all these quantities.

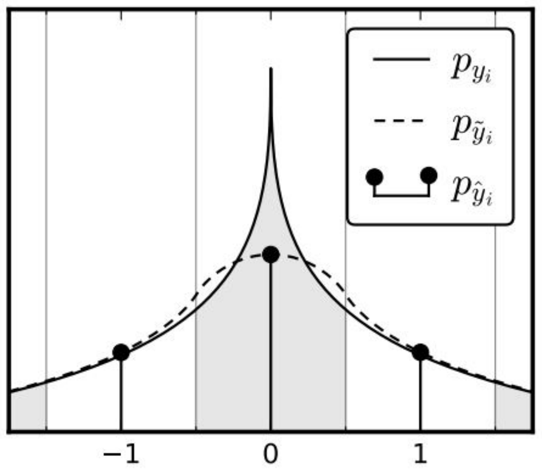


Figure 2: One-dimensional relationship between densities of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$ and $\tilde{\boldsymbol{y}}$.

2.2 Optimization formulation

In a style very similar to variational inference, the goal is to approximate the true intractable posterior $p_{\tilde{\boldsymbol{y}}|\boldsymbol{x}}\left(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}\right)$ with a parametric variational density $q\left(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}\right)$ by minimizing the KL divergence over the data distribution, i.e.,

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} D_{KL}\left[q \| p_{\tilde{\boldsymbol{y}}|\boldsymbol{x}}\right] = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}}\mathbb{E}_{\tilde{\boldsymbol{y}}\sim q}\Big[\cancelto{0}{\log q\left(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}\right)} - \underbrace{\log p_{\boldsymbol{x}|\tilde{\boldsymbol{y}}}\left(\boldsymbol{x} \mid \tilde{\boldsymbol{y}}\right)}_{\text{weighted distortion}} - \underbrace{\log p_{\tilde{\boldsymbol{y}}}\left(\tilde{\boldsymbol{y}}\right)}_{\text{rate}}\Big] + \text{const.}$$

If we look closely at the first term of this optimization equation,

$$q\left(\tilde{\boldsymbol{y}} \mid \boldsymbol{x}, \boldsymbol{\phi}_g\right) = \prod_i \mathcal{U}\left(\tilde{\boldsymbol{y}}_i \mid \boldsymbol{y}_i - \frac{1}{2}, \boldsymbol{y}_i + \frac{1}{2}\right)$$
$$\text{with } \boldsymbol{y} = g_a\left((x); \boldsymbol{\phi}_g\right)$$

where $\mathcal{U}$ is a uniform distribution that is centered around $\boldsymbol{y}_i$. Since the width of this distribution is constant (one), the first term in the optimization equation turns out to be zero.

Assuming the likelihood (second term in the optimization equation) to be a Gaussian,

$$p_{\boldsymbol{x}|\tilde{\boldsymbol{y}}}\left(\boldsymbol{x} \mid \tilde{\boldsymbol{y}}, \boldsymbol{\theta}_g\right) = \mathcal{N}\left(\boldsymbol{x} \mid \tilde{\boldsymbol{x}}, (2\lambda)^{-1}\mathbf{1}\right)$$
$$\text{with } \tilde{\boldsymbol{x}} = g_s\left((\tilde{y}); \boldsymbol{\theta}_g\right)$$

then it can be formulated as the mean squared error between original image and the reconstruction, that's weighted by λ.

The third term is simply the cross entropy between the true marginal $m(\tilde{\boldsymbol{y}}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\boldsymbol{x}}} q(\tilde{\boldsymbol{y}} \mid \boldsymbol{x})$ and the prior $p_{\tilde{\boldsymbol{y}}}\left(\tilde{\boldsymbol{y}}\right)$ which basically is the cost of encoding the $\tilde{\boldsymbol{y}}$, that has been generated from the encoder, using an entropy model $p_{\tilde{\boldsymbol{y}}}$.

2.3 Entropy model

(Ballé et al., 2018) models the prior using a non-parametric, fully factorized density model

$$p_{\tilde{\boldsymbol{y}}|\boldsymbol{\psi}}(\tilde{\boldsymbol{y}} \mid \boldsymbol{\psi}) =$$
$$\prod_i \left(p_{\boldsymbol{y}_i|\boldsymbol{\psi}^{(i)}}\left(\boldsymbol{\psi}^{(i)}\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)\right)\left(\tilde{\boldsymbol{y}}_i\right)$$

where $\boldsymbol{\psi}^{(i)}$ is the parameter for each univariate distribution $p_{\boldsymbol{y}_i|\boldsymbol{\psi}^{(i)}}$. Each density is convolved with a standard uniform density to enable a better match of the prior to the marginal. Often, this prior is referred to as the *Noisy Deep Factorized* prior. Further details on the explicit differentiable form of this prior and its parameters can be found in Appendix 6.1 of (Ballé et al., 2018).

2.4 Scale hyperprior

As discussed in the beginning of Section 2, the rate R is given by the Shannon cross entropy between the true marginal and the prior. The rate is minimized if both distributions are close to each other. However, as we just saw in Section 2.3, the prior is modeled using a fully factorized density model, which means that when statistical dependencies do exist, in the actual distribution of the latent representation, there will be suboptimal compression performance.

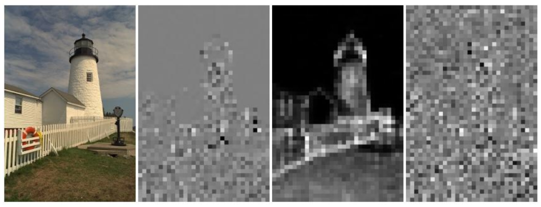


Figure 3: Left: an image from dataset. Middle left: subset of latent representation $\boldsymbol{y}$ learned by *Noisy Deep Factorized* model, notice the structure in the response, indicating statistical dependency. Middle right: standard deviations $\hat{\boldsymbol{\sigma}}$ predicted by the model augmented with hyperprior. Right: latents divided elementwise by their standard deviation, notice the reduced dependency.

For example, consider Figure 3. The center left panel visualizes the quantized responses $\hat{\boldsymbol{y}}$ of a compression model trained in this way. It is quite clear that the choice of using a factorized prior is a stark simplification since non-zero responses are clustered in areas of high contrast. This means that there is some sort of probabilistic coupling in the response, but our prior hasn't been designed to capture that. To alleviate this issue, (Ballé et al., 2018) introduced scale hyperpriors.

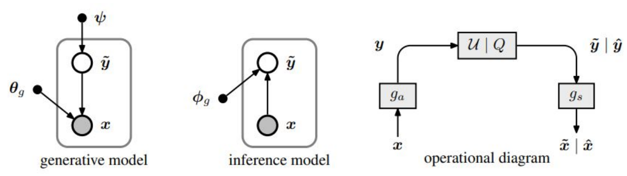
Each element $\tilde{\boldsymbol{y}}$ is now modeled as a zero-mean Gaussian with its own standard deviation $\boldsymbol{\sigma}_i$, where these deviations are predicted by a hyper-synthesis transform h_s applied to $\tilde{\boldsymbol{z}}$.:

$$p_{\tilde{\boldsymbol{y}}|\tilde{\boldsymbol{z}}}\left(\tilde{\boldsymbol{y}} \mid \tilde{\boldsymbol{z}}, \boldsymbol{\theta}_h\right) = \prod_i \left(\mathcal{N}\left(0, \tilde{\boldsymbol{\sigma}}_i^2\right) * \mathcal{U}\left(-\frac{1}{2}, \frac{1}{2}\right)\right)\left(\tilde{\boldsymbol{y}}_i\right)$$
$$\text{with } \tilde{\boldsymbol{\sigma}} = h_s\left(\tilde{\boldsymbol{z}}; \boldsymbol{\theta}_h\right)$$

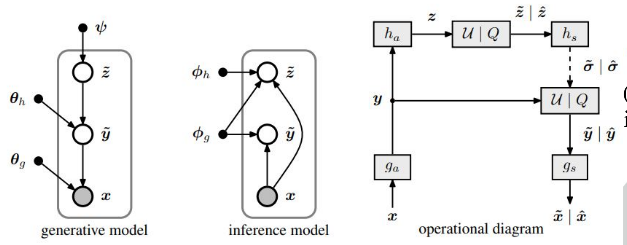
Note that it has been convolved with standard uniform to better fit the target distribution. It is also commonly referred to as the *Noisy Normal*. On the encoder side of things, a hyper-analysis transform h_a is simply stacked over $\boldsymbol{y}$ to get $\boldsymbol{z}$. Since we have no prior belief about the hyperprior, we can simply model it with the *Noisy Deep Factorized* model from Section 2.3. Figure 4 summarises the difference between the two approaches we have seen so far, one with the simple factorized prior and the other with the hyperprior.

3 Hierarchical architecture

In this project, we extend the hyperprior approach to multiple hierarchical levels, where the topmost level is the hyperprior that dictates the standard

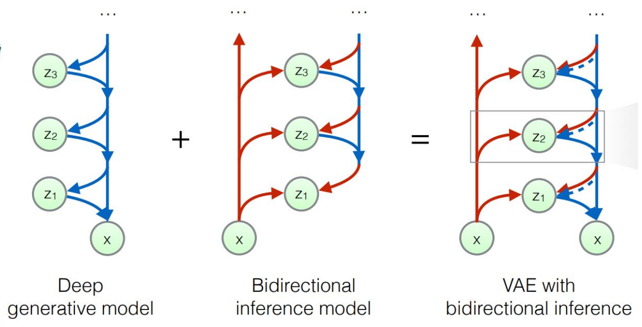


(a) Bayesian model of the simple case, along with the operational structure of the compression model



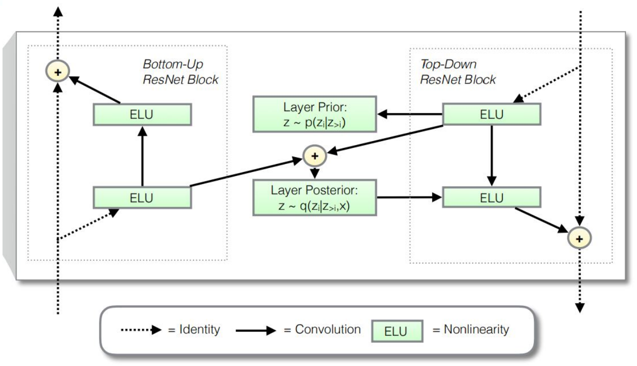
(b) Same as Figure 4a, but for the hyperprior case

Figure 4: Representation of the overall scheme as a Bayesian generative model and the corresponding Bayesian inference model (decoder and encoder respectively). Nodes are the random variables and arrows represent the conditional dependence between them. For the operational diagram, the boxes labeled $\mathcal{U} \mid Q$ represent additive uniform noise applied during training, or true quantization applied during testing.



Z3
Z2
Z1
X
+
=
Deep generative model
Bidirectional inference model
VAE with bidirectional inference

(a) Bayesian model of bidirectional inference and the corresponding VAE



(b) Details for the bidirectional inference block

Figure 5: Modeled using ResNet blocks. The approximate posterior for each layer is conditioned both on bottom-up input and top-down input. The sample from this distribution, after application of non-linearity, is treated as part of the hidden layer of the top-down residual function and then used for downstream layers.

deviation of each latent variable in the hierarchies below it.

More specifically, instead of a regular deep VAE, we work with 6-layer deep hierarchical VAE which follows bidirectional inference. We take inspiration from (Kingma et al., 2016) for its architecture which basically outlines a deep ResNet-style VAE as outlined in Figure 5.

For bidirectional inference, there is a fully deterministic bottom-up pass, before sampling from the posterior in top-down order in the topological ordering of the generative models.

Exact details for how the priors and the posteriors have been modeled can be found in our attached codebase.

4 Experiments

The dataset that we have used for training and validating our model is COCO 2017 (Lin et al., 2014). The dataset consists of 164K images. In the 2017 version used, the training, validation and test split is 118K, 5K and 41K respectively. Some examples of images are shown in Figure 7.

Both models described in the report, the simple and hierarchical version, have been built using the Tensorflow Compression library. The simple version has three layers in the encoder and encoder whereas the hierarchical version has six. The exact manner in which the information flows through the latent blocks in the hierarchical model have been properly explained in the codebase. We train the simple model for 200 epochs and the hierarchical model for 30 epochs for 9 different values of lambda - 0.0006, 0.001, 0.004, 0.015, 0.05, 0.19, 0.65, 2.29, 8.02. Once trained, we report the rate and PSNR values for the validation set and plot the rate-PSNR curve as shown in Figure 6.

5 Results

Figure 6 illustrates the rate-PSNR curve. Since PSNR is inversely related to distortion, higher the curve is for a model, better is its performance. More specifically, for the same bits-per-pixel (BPP), PSNR values of the reconstructed images are better for the neural compression models than for JPEG. Amongst the neural compression models themselves, the hierarchical model (HCAE) outperform the simple baseline model (BLS). This is because HCAE models the entropy in a better

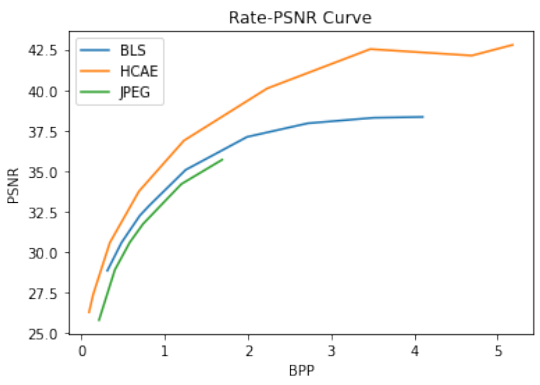


Figure 6: Rate-PSNR Curve, higher the curve, better the performance, neural compression methods outperform classical codec like JPEG.

Figure 7: Example images from the dataset.

way by taking into account the statistical dependencies amongst the latent variables. We also qualitatively analyze the results by looking at the actual reconstructions at various bitrates for images in the Kodak dataset. Note that the model maintains its performance on this new dataset, but that's also because both COCO and Kodak contain natural images. Had it been some other domain of images like medical images, we would have seen unnatural artifacts. As we can see in Figures 8 and 9 (last page), high bitrate JPEG and neural model images look quite similar. However, it is in the low bitrate regime where the neural compression methods truly shine. JPEG images in low bitrate contain a lot of blocky artifacts, owing to the fact that the images are actaully processed in blocks of 8 and this inductive bias is visually apparent in the reconstruction. Moreover, the the blocks have solid color patches which seem to be clustered and distinct. In contrast, we see a much smoother (albeit slightly blurry) reconstruction by the neural compression models. More specifically, HCAE seems to preserve the high level details more precisely than BLS. Similar results are visible in the medium bitrate regime, where the water texture and waves are more accurately captured by HCAE than BLS.

6 Conclusion

The task was fairly difficult for the computer algorithm to solve considering the vast information theory background that is needed to understand the task at hand. Also, optimizing for the entropy model and distortion jointly is also a challenge over the high dimensional latent space. The results were limited due to the high computational requirement needed. A smoother rate-distortion performance curve could be possible if the training was done for more values of lambda. This approach could be prone to high errors in testing in out-of-distribution images, leading to artifacts in the compressed images. Also, when tested on images of different dataset-but in the domain, artifacts were found. The success of the approach is that the method gave an optimized rate-distortion performance. Also, although the method used Mean Squared Error for distortion in loss optimization, the compressed images are much more natural in appearance than JPEG which suffers from severe artifacts in the reconstructed images. We believe this visual improvement arises because biologically-inspired nonlinear transformations in the model might have been better performant to capture and eliminate the components which the human eye cannot detect and thus giving compressed images with good quality. Further, it is observed that the hierarchical model performed better due to efficient modeling of the entropy to generate lower bit rate and distortion. A better system can be built by replacing the convolutional blocks in the architecture with Vision transformers.

A Appendix

The starter code, a pipeline to run the models, was from one of our previous project which we had reused to enable this project. The reused code are boilerplate.py, keras_utils.py, models_utils.py. We have implemented from scratch the bls2017.py, hcae.py and some of the utils.py. The utils.py was modified to fit the needs of the project.

References

Johannes Ballé, Valero Laparra, and Eero P Simoncelli. 2016. End-to-end optimized image compression. *arXiv preprint arXiv:1611.01704*.

Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. 2018. Variational image compression with a scale hyperprior. *arXiv preprint arXiv:1802.01436*.

Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. 2016. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Jorma Rissanen and Glen Langdon. 1981. Universal modeling and coding. *IEEE Transactions on Information Theory*, 27(1):12–23.

Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. 2017. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*.

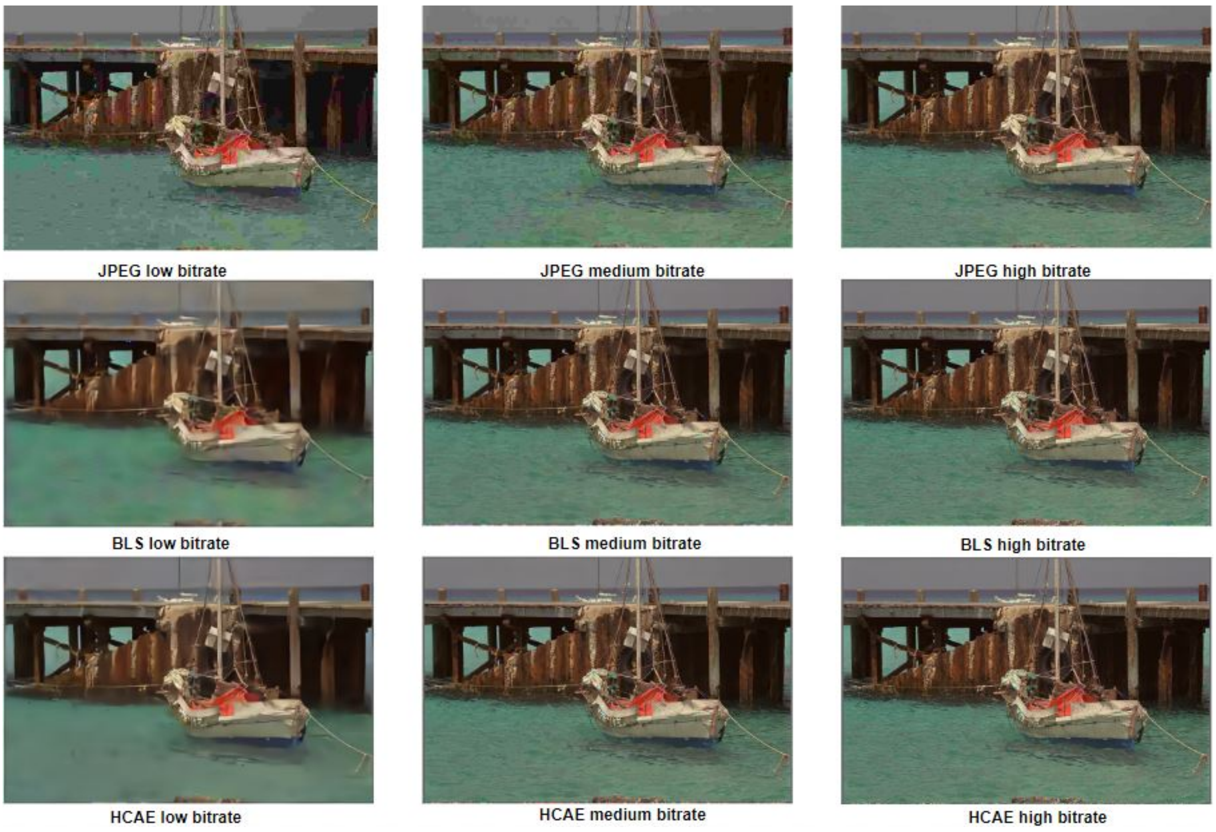


Figure 8: Visual inspection of performance.

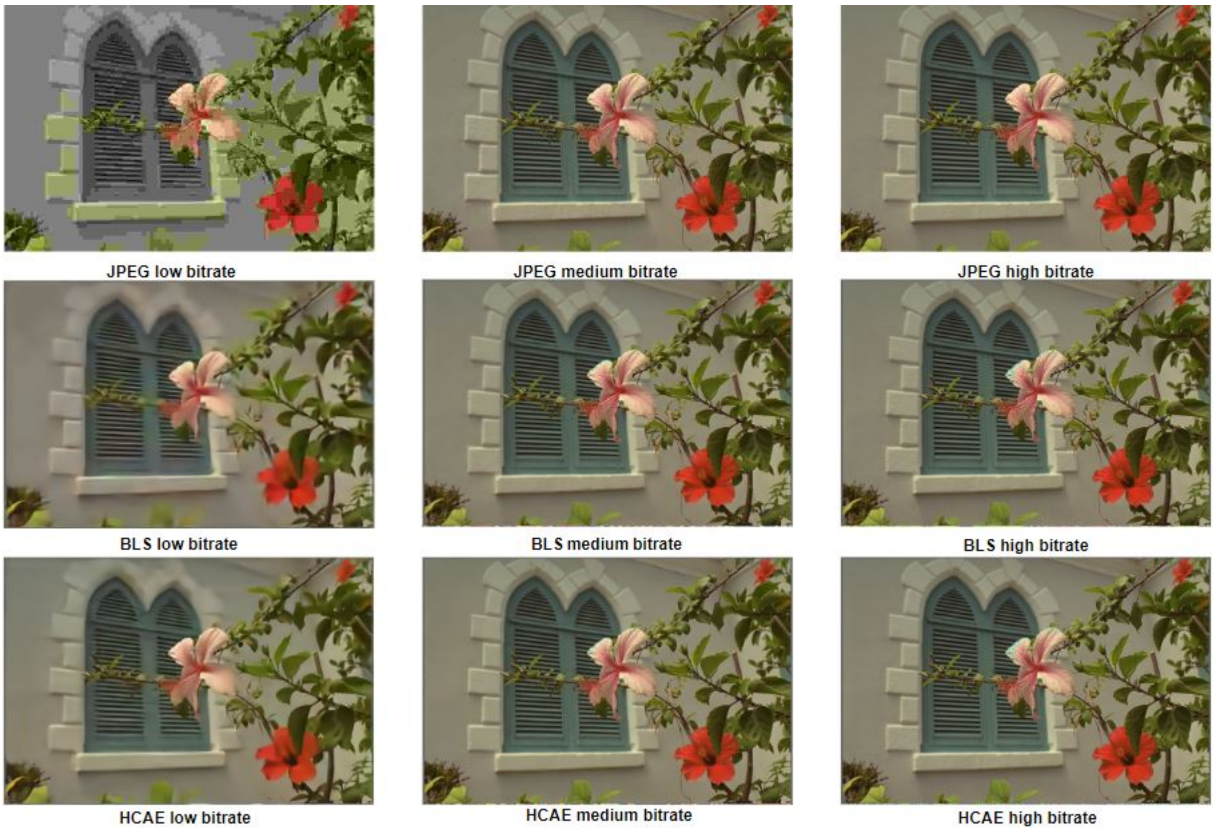


Figure 9: Visual inspection of performance.