Supplementary Material
Unified Multivariate Gaussian Mixture for Efficient Neural Image Compression

Xiaosu Zhu[1] Jingkuan Song[1*] Lianli Gao[1] Feng Zheng[2] Heng Tao Shen[1]

[1]Center for Future Media, University of Electronic Science and Technology of China
[2]Southern University of Science and Technology

xiaosu.zhu@outlook.com, jingkuan.song@gmail.com, shenhengtao@hotmail.com

A. Proof of Eq. (8)

For simplicity to prove Eq. (8), we will not consider residuals and cascaded estimation. Therefore, overall compression pipeline will become: $\boldsymbol{x} \xrightarrow{g_a} \boldsymbol{y} \xrightarrow{Q} \mathfrak{y} = \tilde{\boldsymbol{y}} \xrightarrow{g_x} \tilde{\boldsymbol{x}}$, where $\boldsymbol{x}, \tilde{\boldsymbol{x}}$ are in two set $\boldsymbol{X}, \tilde{\boldsymbol{X}}$ with unknown distributions. According to rate-distortion theory, minimizing $\mathrm{d}\left(\boldsymbol{x}, \tilde{\boldsymbol{x}}\right)$ is equivalent to maximizing mutual information between $\boldsymbol{X}$ and $\mathfrak{Y}$:

$$\max \mathcal{I}\left(\boldsymbol{X};\ \mathfrak{Y}\right). \tag{1}$$

Since we do not add constraints on other network parameters, if network is fully trained, $\mathcal{I}\left(\boldsymbol{X}; \boldsymbol{Y}\right)$ should be maximized. Therefore Eq. (1) can be derived as:

$$\max \mathcal{I}\left(\boldsymbol{Y};\ \mathfrak{Y}\right). \tag{2}$$

Thus maximize the mutual information between latents and quantizeds. To solve $\mathfrak{Y}$ or equivalence $\mathfrak{y}$, Eq. (2) is wrote as a function w.r.t. $\mathfrak{y}$:

$$f\left(\mathfrak{y}\right) = \int_{\boldsymbol{y}} \sum_{\mathfrak{y}} p\left(\boldsymbol{y}, \mathfrak{y}\right) \log \frac{p\left(\boldsymbol{y}, \mathfrak{y}\right)}{p\left(\boldsymbol{y}\right) p\left(\mathfrak{y}\right)} d\boldsymbol{y}. \tag{3}$$

Noticed that $p\left(\mathfrak{y}\right)$ is under Categorical prior whose all entries have the same probability *i.e.* $p_{\mathfrak{Y}}\left(\mathfrak{y} = \boldsymbol{C}_i\right) = {}^1\!/_K,\ 1 \leq i \leq K$ [8]. Eq. (3) can be simplified as:

$$\begin{aligned} f\left(\mathfrak{y}\right) &= \int_{\boldsymbol{y}} \sum_{\mathfrak{y}} p\left(\mathfrak{y}\right) p\left(\boldsymbol{y}\right) \log \frac{p\left(\mathfrak{y} \mid \boldsymbol{y}\right)}{p\left(\mathfrak{y}\right)} d\boldsymbol{y} \\ &= const \cdot \int_{\boldsymbol{y}} p\left(\boldsymbol{y}\right) \log p\left(\mathfrak{y} \mid \boldsymbol{y}\right) d\boldsymbol{y}. \end{aligned} \tag{4}$$

Since $p\left(\boldsymbol{y}\right)$ is not an variable of $f\left(\mathfrak{y}\right)$, and $\log\left(\cdot\right)$ is monotonically increasing, maximizing Eq. (4) is equivalent to maximizing right part of function:

$$\max f\left(\mathfrak{y}\right) \Leftrightarrow \max \log p\left(\mathfrak{y} \mid \boldsymbol{y}\right). \tag{5}$$

*Corresponding author.

Methods		Latency (ms)			
		Encoder		Decoder	
		Abs	Rel	Abs	Rel
Ballé'18		30.66	1.09×	35.54	1.21×
Minnen'18	*w/o*	32.89	1.17×	36.24	1.24×
	→	2656.66	94.58×	1799.47	61.36×
	⚄	59.13	2.11×	40.40	1.38×
Cheng'20	→	2697.58	96.04×	1835.80	62.60×
	⚄	94.11	3.35×	88.04	3.00×
Ours		**28.09**	**1.00×**	**29.32**	**1.00×**
Our Additional		**12.03**	**0.43×**	**13.37**	**0.46×**

Table 1. Encoding and decoding latency comparisons for image size 768×512. Our additional model achieves even faster speed than our main model.

This means when $\boldsymbol{y}$ is given to produce $\mathfrak{y}$, that specific probability should be *one i.e.* fully confident. Recall that

$$\begin{gathered} p_{\mathfrak{Y}|\boldsymbol{Y}}\left(\mathfrak{y} \mid \boldsymbol{y};\ \boldsymbol{C}\right) = \prod_{k=1}^{K} \zeta\left(\boldsymbol{\phi}\right)_k^{\mathbb{1}\{\mathfrak{y} = \boldsymbol{C}_k\}}, \\ where\ \boldsymbol{\phi}_k = -\left\|\boldsymbol{y} - \boldsymbol{C}_k\right\|_2^2,\ 1 \leq k \leq K. \end{gathered} \tag{6}$$

So,

$$\begin{gathered} p_{\mathfrak{Y}|\boldsymbol{Y}}\left(\mathfrak{y} = \boldsymbol{C}_k \mid \boldsymbol{y}\right) \propto -\left\|\boldsymbol{y} - \boldsymbol{C}_k\right\|_2^2, \\ p_{\mathfrak{Y}|\boldsymbol{Y}}\left(\mathfrak{y} = \boldsymbol{C}_k \mid \boldsymbol{y}\right) = 1 \Leftrightarrow \left\|\boldsymbol{y} - \boldsymbol{C}_k\right\|_2^2 = 0. \end{gathered} \tag{7}$$

Therefore, if $\boldsymbol{y}$ has a nearest codeword $\boldsymbol{C}_k$, then $\boldsymbol{\phi}_k$ should be *zero* in order to pick $\boldsymbol{C}_k$ with maximized confidence. So, for a subset $\boldsymbol{Y}_k = \{\boldsymbol{y} \in \boldsymbol{Y} \mid \boldsymbol{\Phi}_k = 1\}$, all $\boldsymbol{y}$s in this subset "*pull*" codeword $\boldsymbol{C}_k$ to be close to them, making $\boldsymbol{C}_k$ to be the mean embedding of $\boldsymbol{Y}_k$. So Eq. (8) is derived.

B. Model Architecture

Now, we show our detailed model design in Fig. 1. We adopt similar structure from [3] *i.e.* Residual blocks and attention blocks. You could check structures in their paper.

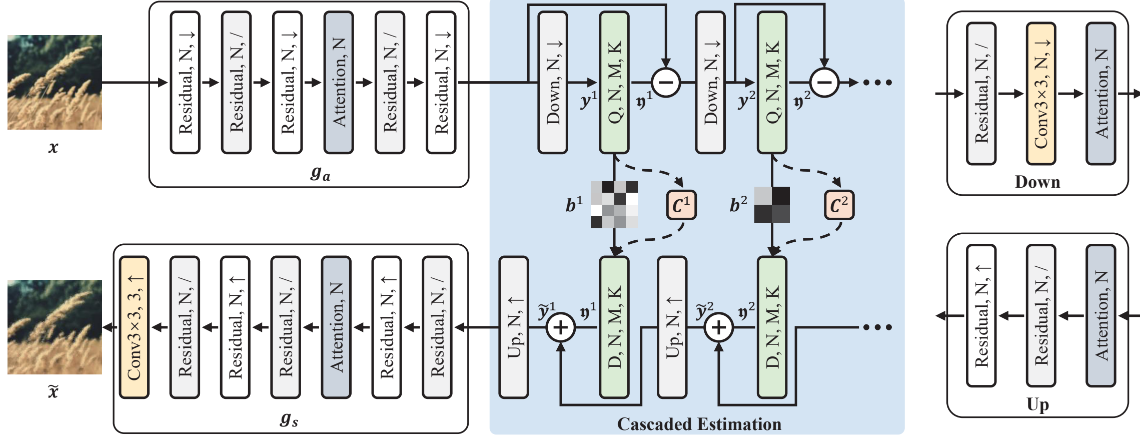


Figure 1. The detailed framework. "Residual", "Attention" are residual block and attention block from [3]. "Down" and "Up" blocks are placed on the right. "N" is number of channels. "↓" means output is $2\times$ down-sampled and vice-versa. "M", "K" are codebook sizes (M sub-codebooks, K codewords for each.).

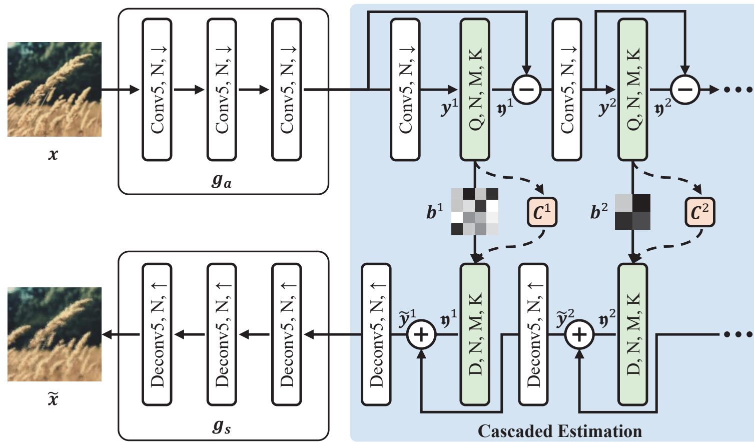


Figure 2. An additional model with *Conv5* and *Deonv5* layers. The structure is similar as Ballé's [2] and Minnen's [5] in order to test generalization ability of our method. Specifically, *Conv5* uses $kernel_size = 5 \times 5$, $stride = 2$, $padding = 2$, and output size is $2\times$ smaller than input. *Deconv5* is reverse of *Conv5*.

Variants								
M	1	2	4	6	8	12	16	24
K	64	128	256	512	1024	2048	4096	8192

Table 2. Models with different codebook sizes. We test codebooks from small to large to validate scalability.

"Down" and "Up" blocks are placed on the right. "↓" means output is $2\times$ down-sampled and vice-versa. N the number of channels, M, K the codebook sizes (M sub-codebooks, K codewords for each.).

For instance, given an image of size 768×512, we first $16\times$ down-sample it to 96×64, then go to cascaded estimation. Each $\boldsymbol{y}$ is obtained by one more down-sampling. So, size of $\boldsymbol{y}^1$ is 48×32, $\boldsymbol{y}^2$ is 24×16, *etc.*, and vice versa for decoding.

How to Calculate sup bpp**?** As shown above, output code size will be $16^2\times, 32^2\times, 64^2\times$ smaller than original images. According to Sec.4, the upper bound of bpp is:

$$M \cdot \frac{\sum_l \log_2 K \cdot h^\ell \cdot w^\ell}{H \cdot W}. \tag{8}$$

For example, when we employ model №1, where $M = 2$ and $K = [8192, 2048, 512]$, the above result is:

$$2 \cdot \left(\frac{13}{16^2} + \frac{11}{32^2} + \frac{9}{64^2} \right) \approx 0.1274 \tag{9}$$

```python
class Quantizer(Module):
    """
    Quantizer with `m` sub-codebooks,
        `k` codewords for each, and
        `n` total channels.
    Args:
        m (int): Number of sub-codebooks.
        k (int): Number of codewords for each sub-codebook.
        n (int): Number of channels of latent variables.
    """
    def __init__(self, m: int, k: int, n: int):
        super().__init__()
        # A codebook, feature dim `d = n // m`.
        self._codebook = Parameter(torch.empty(m, k, n // m))
        self._initParameters()

    def forward(self, x: Tensor, t: float = 1.0) -> (Tensor, Tensor):
        """
        Module forward.
        Args:
            x (Tensor): Latent variable with shape [b, n, h, w].
            t (float, 1.0): Temperature for Gumbel softmax.
        Return:
            Tensor: Quantized latent with shape [b, n, h, w].
            Tensor: Binary codes with shape [b, m, h, w].
        """
        b, _, h, w = x.shape
        # [b, m, d, h, w]
        x = x.reshape(b, len(self._codebook), -1, h, w)
        # [b, m, 1, h, w], square of x
        x2 = (x ** 2).sum(2, keepdim=True)
        # [m, k, 1, 1], square of codebook
        c2 = (self._codebook ** 2).sum(-1, keepdim=True)[..., None]
        # [b, m, d, h, w] * [m, k, d] -sum-> [b, m, k, h, w]
        # dot product between x and codebook
        inter = torch.einsum("bmdhw,mkd->bmkhw", x, self._codebook)
        # [b, m, k, h, w], pairwise L2-distance
        distance = x2 + c2 - 2 * inter
        # [b, m, k, h, w], distance as logits to sample
        sample = F.gumbel_softmax(-distance, t, hard=True, dim=2)
        # [b, m, d, h, w], use sample to find codewords
        quantized = torch.einsum("bmkhw,mkd->bmdhw", sample, self._codebook)
        # back to [b, n, h, w]
        quantized = quantized.reshape(b, -1, h, w)
        # [b, n, h, w], [b, m, h, w], quantizeds and binaries
        return quantized, sample.argmax(2)
```

Figure 3. Minimal implementation of our probabilistic vector quantization.

C. Implementation

To implement a probabilistic vector quantization with multi-codebooks, we could seek help from a few PyTorch built-in functions such as `einsum` and `gumbel_softmax`. Our implementation is shown in Fig. 3. Specifically, to calculate the pair-wise Euclidean distance in order to produce ϕ, we can use the expanded version for speed up, *e.g.*, for two matrix $\boldsymbol{U} \subseteq \mathbb{R}^{k_1 \times n}, \boldsymbol{V} \subseteq \mathbb{R}^{k_2 \times n}$, the pairwise distance $\boldsymbol{D} \subseteq \mathbb{R}^{k_1 \times k_2}$ is calculated by: $\boldsymbol{U}^2 + \boldsymbol{V}^2 - 2\boldsymbol{U}\boldsymbol{V}^\intercal$. We could utilize `einsum` to perform above calculation in M ways separately with very few line of codes (line № 31 $\sim$ 38). Then, the calculated distance will be input of `gumbel_softmax` to sample one-hot vectors (line № 40). The indices of where "*one*"s present are collected as $\boldsymbol{b}$ (line № 46).

D. Additional Experiments

We also conduct a few additional experiments to investigate detailed latencies, model generalization ability, *etc*.

D.1. Compression Latencies w.r.t. Codebook Size

We conduct latency tests of our models by varying M, K in codebook, to see if codebook size could affect model efficiency. Specifically, we set $N = 192$, $L = 1$, M varies

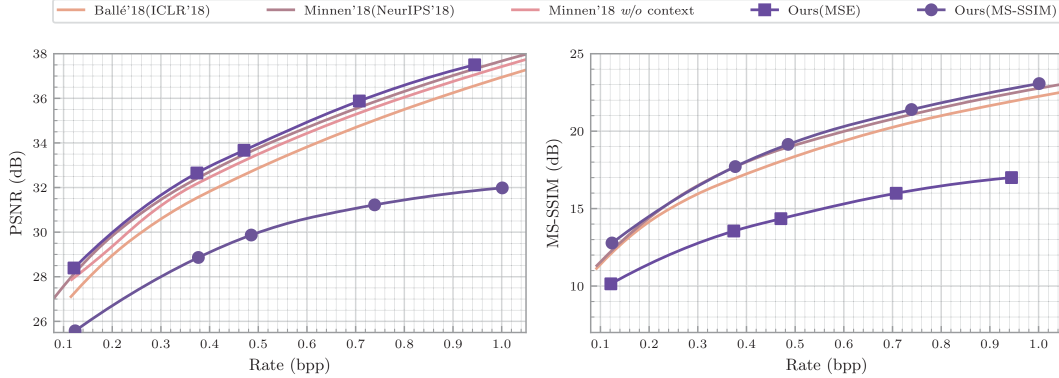


Figure 4. Rate-Distortion performance with our additional *conv5*-based model on Kodak dataset. Ours is slightly better than Ballé'18 and Minnen'18, which have similar backbone.

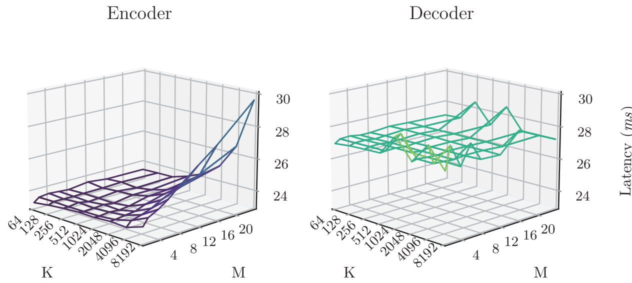


Figure 5. Latencies w.r.t. codebook sizes. Some bumps in decoder is considered to be within margin of error.

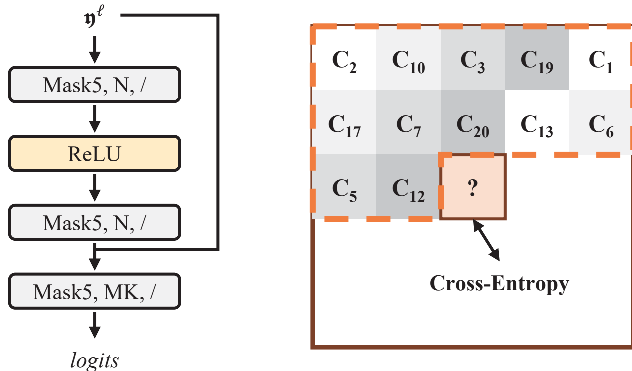


Figure 6. **Left**: Residual-based masked convolutional block. We use `MaskedConv` with kernel size 5×5. The last layer produces $M \times K$ logits and trained by picked codeword indices and cross-entropy loss. **Right**: Demonstration of masked prediction. It uses left and top $\mathfrak{y}$ as context information to predict index of next picked codeword.

from 1 to 24 and K varies from 64 to 8192. Detailed settings are placed in Tab. 2 while results are placed in Fig. 5.

From Fig. 5, we could draw following conclusions. Firstly, encoder's latency is linearly correlated to K. This is because computation of ϕ consumes $\mathcal{O}(NKD)$ time complexity.

Codecs	BPP	ΔE	LPIPS	IS
VVC VTM 14.2	0.1455	**3.927**	0.159	3.593
Cheng'20 (MSE)	0.1281	4.577	0.161	3.542
Cheng'20 (MS-SSIM)	0.1279	4.782	0.149	3.791
Ours (MSE)	0.1234	4.505	0.163	3.535
Ours (MS-SSIM)	0.1256	4.966	**0.144**	**3.875**

Table 3. Perceptual comparisons between VVC, Cheng'20 and ours on Kodak dataset. The results indicate that deep models targeting MS-SSIM generally perform better than MSE on LPIPS and IS. And in contrast, MSE models as well as VVC perform better on ΔE.

Decoder's latency is smooth and flat. Since it is not affected by K or M. Decoding only involves $\mathcal{O}(1)$ lookup and operations between sub-codebooks are highly paralleled. Therefore, no matter how many codewords are employed in quantization, decoding can be still treated as $\mathcal{O}(1)$ roughly.

D.2. Additional Perceptual Evaluations

To make a comprehensive study on image restoration quality of out network as well as other codecs, we notice that there are a lot of perceptual metrics can be adopted. In this study, we choose ΔE [1], LPIPS [9] and Inception score (IS) [6][1]. We pick "VVC", "Cheng'20" and "ours" to test since they have similar performance in main paper. Due to limitation of computation resources, we only test with our model №2 and tune quantization parameter of other codecs to target similar *bpp*. Results are shown in Tab. 3. From this table, we could confirm codecs that target MSE generally perform worse than MS-SSIM on LPIPS and IS scores. In contrast, MSE models have lower ΔE than MS-SSIM models.

[1] We use their open-source PyTorch implementations.

Context $Acc.$	bpp $w/$	bpp w/o
2.24%	0.1262	0.1265

Table 4. Prediction accuracy of the auxiliary context model. With context model, we only obtain 2.24% accuracy for context prediction on the average among all levels and all groups, and bpp is nearly the same with original model.

D.3. R-D Performance with Other Backbones

In main paper, we only report R-D performance based on [3]'s backbone. It is important to test with other backbones to evaluate the generalization ability of our method. Therefore, we design an additional model, as shown in Fig. 2. Specifically, we use $Conv5$ and $Deconv5$ with 5×5 kernels to perform $2\times$ down-sampling and up-sampling.

Results on Kodak dataset are placed in Fig. 4. We only compare our additional model with methods that have similar backbone *i.e*. Ballé'18 [2] and Minnen'18 [5]. Similar as results in main paper, our method has a slightly better R-D performance against Ballé'18 and Minnen'18. These results indicate our method is suitable for different backbones. It is foreseeable that our method would be effective if incorporates with other backbones *e.g*. [4].

We also test latencies with this additional model, placed in Tab. 1 at the last row. From the table we find that latencies of the additional model are further reduced. This is because our additional model has fewer layers than main model. Compared to Ballé'18 and Minnen'18, our model is much faster. We will release two types of models in the future.

D.4. Incorporating with Auxiliary Context Model

In main paper, we claim that our model does not need auxiliary context models for side information prediction. But we still want to know whether context models could help for better Rate-Distortion performance. To incorporate with a context model, we adopt the widely-used PixelCNN [7]. Specifically, we build a residual-like block with full of $MaskedConv$ layers (Fig. 6) as the causal prediction network. We insert these blocks directly after $\mathfrak{y}^{\ell}$ on every level. Then, they produce $M \times K$ logits and are trained with picked codeword indices and cross-entropy loss. This procedure is also demonstrated in Fig. 6.

To evaluate how well these introduced networks predict, prediction accuracy of next picked codeword is calculated. For instance, if they could predict 50% of picked codewords' indices, bpp will be reduced 50% approximately.

Results are reported in Tab. 4. With context model, we only obtain 2.24% accuracy for context prediction on the average among all levels and all groups, and bpp is nearly the same with original model. This indicates that our model has encoded binary codes with high information entropy. Introducing extra context model do not further reduce rate.

(a) Perturbed area. 15% codes are changed to random new values. (b) Visualization result after perturbation. BPP: 0.1278 → 0.1283.

Figure 7. Effects of code perturbation.

D.5. Effects of Code Perturbation

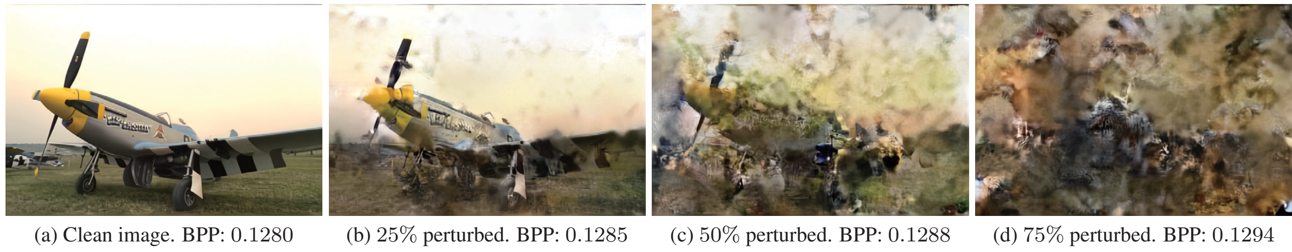
As mentioned in limitations and broader impacts, we could craft images that corrupt vectorized prior by *e.g*. adversarial attack. Therefore, a simple study is conducted by perturbing partial of compression codes to simulate this approach.

Specifically, We randomly perturb 15% of $\boldsymbol{b}$ that produced by our MS-SSIM model №1. Result based on Fig.7 is shown in Fig. 7. In Fig. 7(a), lighter area indicates more codes are perturbed. Fig. 7(b) is reconstruction result that appears to be artifacts on it. For whole Kodak dataset, after perturbation, bpp increases $0.1256 \to 0.1267$ and MS-SSIM decreases $14.33 \to 8.83$.

From above observations, firstly we think these reveal model's ability to choose appropriate codes for good rate-distortion. If any codes are misplaced, not only performance will drop, but also rate will increase. And intuitively, if perturbation rate increases, model performance will continuously drop. This can be confirmed by following experiments in Fig. 8. We think this is a valuable research problem and would like to conduct future studies on robustness and mechanisms of proposed vectorized prior.

D.6. Visualization

We further pick 2 images in Kodak and 3 images in CLIC Professional valid set for comprehensive visualization in Figs. 9 to 13. From these figures we could find our model preserves rich details.



(a) Clean image. BPP: 0.1280 (b) 25% perturbed. BPP: 0.1285 (c) 50% perturbed. BPP: 0.1288 (d) 75% perturbed. BPP: 0.1294

Figure 8. Restored images with various perturbation proportions.

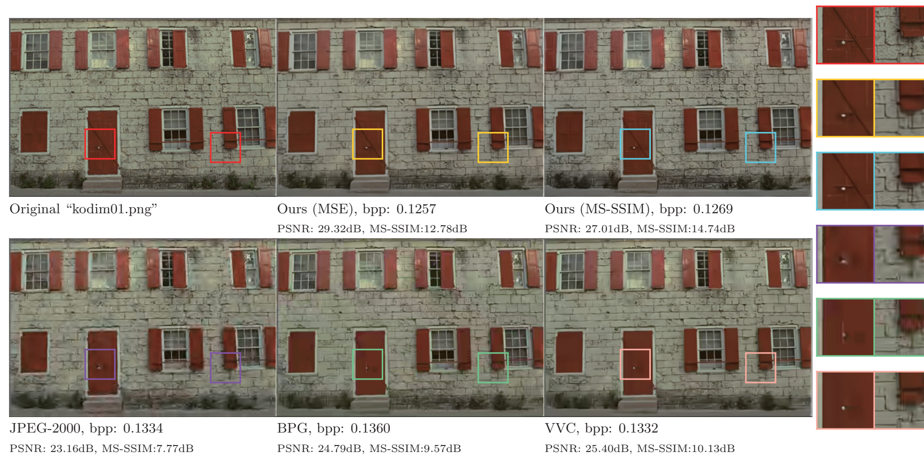


Figure 9. Comparisons of "`kodim01.png`" with other codecs.

Figure 10. Comparisons of "`kodim07.png`" with other codecs.

Figure 11. Ours bpp = 0.1236, PSNR = 34.29dB, MS-SSIM = 19.40dB.

Figure 12. Ours bpp = 0.1265, PSNR = 26.95dB, MS-SSIM = 11.64dB.

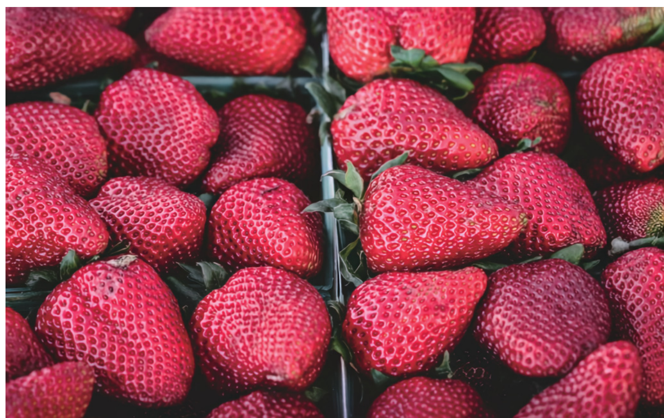


Figure 13. Ours bpp = 0.1259, PSNR = 27.04dB, MS-SSIM = 13.51dB.

References

[1] Werner G.K. Backhaus, Reinhold Kliegl, and John S. Werner. *Color Vision: Perspectives from Different Disciplines*. De Gruyter, 2011. 4

[2] Johannes Ballé, David Minnen, Saurabh Singh, Sung Jin Hwang, and Nick Johnston. Variational image compression with a scale hyperprior. In *ICLR*, 2018. 2, 5

[3] Zhengxue Cheng, Heming Sun, Masaru Takeuchi, and Jiro Katto. Learned image compression with discretized gaussian mixture likelihoods and attention modules. In *CVPR*, pages 7936–7945, 2020. 1, 2, 5

[4] Ge Gao, Pei You, Rong Pan, Shunyuan Han, Yuanyuan Zhang, Yuchao Dai, and Hojae Lee. Neural image compression via attentional multi-scale back projection and frequency decomposition. In *ICCV*, pages 14677–14686, 2021. 5

[5] David Minnen, Johannes Ballé, and George Toderici. Joint autoregressive and hierarchical priors for learned image compression. In *NeurIPS*, pages 10794–10803, 2018. 2, 5

[6] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *NeurIPS*, 2016. 4

[7] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017. 5

[8] Aäron van den Oord, Oriol Vinyals, and Koray Kavukcuoglu. Neural discrete representation learning. In *NeurIPS*, pages 6306–6315, 2017. 1

[9] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018. 4