Abstract

We propose the deep progressive image compression using trit-planes (DPICT) algorithm, which is the first learning-based codec supporting fine granular scalability (FGS). First, we transform an image into a latent tensor using an analysis network. Then, we represent the latent tensor in ternary digits (trits) and encode it into a compressed bitstream trit-plane by trit-plane in the decreasing order of significance. Moreover, within each trit-plane, we sort the trits according to their rate-distortion priorities and transmit more important information first. Since the compression network is less optimized for the cases of using fewer trit-planes, we develop a postprocessing network for refining reconstructed images at low rates. Experimental results show that DPICT outperforms conventional progressive codecs significantly, while enabling FGS transmission. Codes are available at https://github.com/jaehanlee-mcl/DPICT.

1. Introduction

Image compression is a fundamental problem in image processing and analysis. Classical image codecs, such as JPEG [44], JPEG2000 [37], WebP [18], and BPG [8], have been developed to achieve the goal of efficient data storage and transmission. They contain several modules to process hand-crafted features. For example, for transform coding, JPEG uses discrete cosine transform, and JPEG2000 adopts wavelet transform.

Recently, with the availability of substantial training data and computing resources, deep learning has been adopted for image compression, as well as other image and vision problems. Some learning-based codecs are based on convolutional neural networks (CNNs) [5, 6, 26, 30], while others on recurrent neural networks (RNNs) [19, 24, 40, 41]. In terms of the rate-distortion (RD) performance, recent learning-based codecs [11, 13, 47] are competitive with or even superior to the classical codecs.

Progressive compression, or scalable coding [32], is a crucial issue. A progressive codec encodes an image into a single bitstream that can be decoded at various bit-rates, as illustrated in Figure 1. There are various terminals from small wearable devices to big TVs, requiring different image qualities at different bit-rates. It is inefficient to encode multiple non-scalable bitstreams for these diverse devices. In contrast, a scalable bitstream can be efficiently truncated at multiple points to reconstruct the scene at different qualities. Moreover, when a network has a limited bandwidth, the receiver of a scalable bitstream can first check a preview image by receiving a small portion of the bitstream, and then reconstruct a higher quality image by decoding the remaining bits.

There are several learning-based progressive codecs [10, 19, 24, 40, 41], which, however, support coarse granular scalability only: a bitstream can be decoded at a limited number of rates. To the best of our knowledge, the proposed algorithm is the first learning-based codec with fine granular scalability (FGS) [28, 36]: a single bitstream can be truncated at any points to reconstruct the scene faithfully. Furthermore, despite this additional functionality, the proposed algorithm provides better RD performance than the conventional learning-based progressive codecs.

In this paper, we propose the deep progressive image codec using trit-planes (DPICT) with FGS. First, we transform an image into a latent tensor, each element of which is
represented by ternary digits (trits). Then, we encode the latent tensor trit-plane by trit-plane in the decreasing order of significance. Moreover, even in the same trit-plane, we sort the trits according to the RD priorities to transmit more important information first. At the decoder, when fewer trit-planes are used, the reconstructed image is degraded by quantization errors and contain noisy artifacts. To reduce such artifacts, we also develop postprocessing networks. Experimental results demonstrate that DPICT outperforms the conventional progressive codecs significantly, while supporting FGS.

2. Related Work

**Learning-based compression:** A typical learning-based image compression method [3, 5, 39] constructs a neural network, by integrating a quantizer into an autoencoder [43], which is trained end-to-end to minimize a loss function, including rate and distortion terms. Given an image, the encoder (or analysis network) generates a latent representation, which is then quantized. The decoder (or synthesis network) dequantizes the representation and reconstructs the image in a lossy manner. The quantizer is, however, not differentiable. For the backpropagation in the training phase, it is approximated by the binarization process [40], additive uniform noise [5], or stochastic rounding [39].

Rippel and Bourdev [35], Tschannen et al. [42], and Agustsson et al. [4] adopted generative adversarial networks [17] for image compression. Nakanishi et al. [31] developed an image codec based on a multi-scale autoencoder. Ballé et al. [6] proposed an additional autoencoder for a hyperprior. They assumed that latent elements have Gaussian distributions with zero mean, and used the hyperprior autoencoder to encode the standard deviations. Mentzer et al. [29], Minnen et al. [30], Lee et al. [26], and Li et al. [27] adopted context models. Also, more sophisticated hyperpriors have been studied. Cheng et al. [11] assumed Gaussian mixture models for latent elements, and Cui et al. [13] used asymmetric Gaussian distributions.

**Variable-rate compression:** While traditional codecs, such as JPEG [44], support variable bit-rates, most learning-based codecs [5, 6, 11, 26, 27, 30, 39] can generate bitstreams at fixed rates only. For multiple rates, they should train as many models, which incur inefficiency in testing, as well as in training.

Hence, several learning-based codecs have been developed to achieve variable-rate compression using a single network. Theis et al. [39] proposed a variable-rate training scheme for a single autoencoder using a scale parameter for quantization. Choi et al. [12] also employed the scaled quantization, while training rate-specific parameters for a few selected rates. Similarly, Yang et al. [48] proposed the modulated autoencoders containing separate modules for selected rates. Cai et al. [9] generated multi-scale representations and performed content-adaptive rate allocation. Cui et al. [13] proposed gain units to guide the network to allocate more bits to specific channels and also to control rates. Yang et al. [47] used the slimmable neural networks [49, 50] to perform low-rate compression using only a fraction of the parameters and the highest-rate compression using all parameters. These codecs [9, 12, 13, 39, 47, 48] can adapt to different rates with a single trained model, but their bitstreams are not scalable, i.e. a lower-rate bitstream is not embedded in a higher-rate one.

**Progressive compression:** Progressive codecs to encode scalable bitstreams have also been studied. For example, JPEG [44] and JPEG2000 [37] are traditional progressive codecs. There are several learning-based progressive codecs, most of which are based on RNNs [19, 24, 40, 41]. Toderici et al. [40] proposed the first RNN-based progressive codec. Their network, utilizing the long short-term memory (LSTM) [20], transmits bits progressively: At stage $t + 1$, the encoder transmits the residual error at stage $t$, and the decoder reconstructs it and adds it to the reconstructed image at stage $t$. If this is repeated $T$ times, the single bitstream can support $T$ different rates progressively. Gregor et al. [19] also introduced a recurrent image codec, which improves conceptual quality using a generative model. However, these codecs [19, 40] are for low-resolution patches. Toderici et al. [41] developed a codec for higher-resolution images by expanding the previous work in [40]. Johnston et al. [24] proposed an effective initializer for hidden states of their RNN and a spatially adaptive rate controller. However, all these RNN-based algorithms support only coarse granular scalability: a bitstream can be reconstructed at $T$ different rates only, where $T$ is the number of recurrent stages. Moreover, their compression performances are inferior even to those of the traditional codecs [8]. On the other hand, Cai et al. [10] proposed a network of a single encoder and two decoders. The encoder decomposes an image into two representations. Then, the preview decoder uses only one representation, and the full-quality decoder uses both representations. Hence, their network supports two rates only.

The proposed DPICT algorithm supports FGS in contrast to these learning-based progressive codecs [10, 19, 24, 40, 41]. Furthermore, DPICT provides significantly better RD performance than the conventional progressive codecs.

3. Proposed Algorithm

3.1. Compression network

We adopt the compression framework in Figure 2, which consists of an encoder $g_a$, a decoder $g_s$, a hyper encoder $h_a$, and a hyper decoder $h_s$, as done in [6, 11–13, 26, 30, 47]. An image $X$ is transformed to a latent representation $Y$ and
a hyper latent representation $\mathbf{Z}$ sequentially by $g_a$ and $h_a$. Using the factorized prior model \cite{6}, denoted by $q_f(\cdot)$ in Figure 2, $\mathbf{Z}$ is digitized to $\tilde{\mathbf{Z}}$. From $\tilde{\mathbf{Z}}$, $h_s$ yields $\mathbf{M}$ and $\Sigma$, which contain the means and standard deviations of the elements in $\mathbf{Y}$, respectively. These elements are assumed to be independent Gaussian random variables. Then, the mean-removed (or centered) $\check{\mathbf{Y}}_c = \mathbf{Y} - \mathbf{M}$ is quantized to

$$\check{\mathbf{Y}}_c = q(\mathbf{Y}_c)$$  \hspace{1cm} (1)

where rounding is used for the quantizer $q(\cdot)$. Finally, the decoder $g_s$ adds $\mathbf{M}$ back to $\check{\mathbf{Y}}_c$ to yield

$$\hat{\mathbf{Y}} = \check{\mathbf{Y}}_c + \mathbf{M}$$  \hspace{1cm} (2)

and uses it to reconstruct $\hat{\mathbf{X}}$.

In addition to $\tilde{\mathbf{Z}}$, $\check{\mathbf{Y}}_c$ in (1) is encoded into a bitstream. Let $\check{y}_c$, $y_c$, and $\sigma$ be corresponding elements in $\check{\mathbf{Y}}_c$, $\mathbf{Y}_c$, and $\Sigma$. Then, the number of bits for encoding $\check{y}_c$ is given by

$$N(\check{y}_c) = -\log_2 P(\check{y}_c - \frac{1}{2} \leq y_c < \check{y}_c + \frac{1}{2})$$  \hspace{1cm} (3)

where $y_c \sim \mathcal{N}(0, \sigma^2)$. Unlike conventional algorithms, we compress $\check{\mathbf{Y}}_c$ progressively using trit-planes in Section 3.2.

In the training phase, since the quantizer is not differentiable, it is approximated by an additive noise function $u(\cdot)$,

$$\check{\mathbf{Y}} = u(\mathbf{Y}) = \mathbf{Y} + U(-\frac{1}{2}, \frac{1}{2})$$  \hspace{1cm} (4)

where $U(-\frac{1}{2}, \frac{1}{2})$ is a uniform noise tensor in range $(-\frac{1}{2}, \frac{1}{2})$. The noise function $u(\cdot)$ is assumed to generate white noise and not considered during the back-propagation of gradients \cite{5}. Similarly, $\tilde{\mathbf{Z}}$ is obtained from $\tilde{\mathbf{Z}}$ \cite{30}, and then $\mathbf{M}$ and $\Sigma$ are estimated. Finally, $\hat{\mathbf{X}}$ is reconstructed from $\check{\mathbf{Y}}$.

The loss function $\ell$ consists of a distortion term $\ell_D$ and a rate term $\ell_R$,

$$\ell = \ell_D(\mathbf{X}, \hat{\mathbf{X}}) + \lambda \cdot \ell_R(\hat{\mathbf{Y}}, \hat{\mathbf{Z}}, \hat{\mathbf{M}}, \hat{\Sigma})$$  \hspace{1cm} (5)

where $\ell_D$ is defined as the mean squared error between $\mathbf{X}$ and $\hat{\mathbf{X}}$, and $\ell_R$ is the rate for encoding the elements in $\hat{\mathbf{Y}}$ and $\hat{\mathbf{Z}}$. Notice that the rate for $\hat{\mathbf{Y}}$ is given by the sum of the numbers of bits $N(\check{y}_c)$, similar to (3), but based on $\hat{\mathbf{M}}$ and $\Sigma$. In (5), $\lambda$ is a Lagrangian multiplier to control the RD trade-off.

3.2. Trit-plane coding

We represent $\check{\mathbf{Y}}_c$ in (1) in a ternary number system and encode it trit-plane by trit-plane to achieve FGS. There are $K = H \times W \times C$ elements in $\check{\mathbf{Y}}_c$, where $H$, $W$, and $C$ are the height, the width, and the number of channels, respectively. Conventional algorithms transmit these $K$ elements in a raster scan order, so they cannot decompress a partial bitstream meaningfully; the reconstructed image is severely degraded if some of the elements are missing. In contrast, we express each element using $L$ trits and compress the $L$ trit-planes in the decreasing order of significance from the most significant trit (MST) to the least significant trit (LST), as shown in Figure 3. In this way, front parts of the bitstream contain more important information, enabling the decoder to perform progressive reconstruction faithfully.

Figure 4 illustrates how to progressively compress an element $y_c = q(\check{y}_c)$ in $\check{\mathbf{Y}}_c$, where $y_c \sim \mathcal{N}(0, \sigma^2)$. In this example, $L = 3$, so it is assumed that $y_c$ is rounded to one of the integers between $-13$ and $13 = \frac{3^L-1}{2}$. First, the number line is partitioned into three intervals, and the MST indicates which interval contains $y_c$. Specifically, $0(3), 1(3)$, and $2(3)$ correspond to the left, middle, and right intervals, respectively. In Figure 4, $y_c$ belongs to the middle interval $I_1 = [-4.5, 4.5)$, so the trit $3(3)$ is encoded into the bitstream. Second, $I_2 = [-4.5, 4.5)$ is partitioned into three sub-intervals, and the next trit $3(3)$ informs that $y_c \in I_2 = [1.5, 4.5)$. Finally, $I_3$ is partitioned again, and the third trit (LST) $3(3)$ means that $y_c \in I_3 = [1.5, 2.5)$.

Hence, with all three trits, the decoder knows that $y_c = 2$.

In general, let $I_n = [l_n, r_n)$ denote the interval containing $y_c$ when the first $n$ trits are encoded. Also, let $t_n \in \{0(3), 1(3), 2(3)\}$ denote the $n$th interval. $I_n$ is partitioned into three sub-intervals of the same length (except for the leftmost and rightmost intervals):

$$I_n^0 = [l_n^0, r_n^0) = [l_n, \frac{2l_n + r_n}{3}],$$

$$I_n^1 = [l_n^1, r_n^1) = [\frac{2l_n + r_n}{3}, \frac{4l_n + 2r_n}{3}],$$

$$I_n^2 = [l_n^2, r_n^2) = [\frac{4l_n + 2r_n}{3}, r_n).$$

Then, the next $(n+1)$th trit $t_{n+1}$ reveals which subinterval contains $y_c$. It becomes $I_{n+1}$. 

![Figure 2. Image compression framework.](image-url)

![Figure 3. Illustration of the trit-plane coding of $K$ elements.](image-url)
The conditional probability of \( t_n \) given \( \{t_{n-1}, \cdots, t_1\} \) is given by
\[
P(t_n|t_{n-1}, \cdots, t_1) = P(y_c \in I_n | y_c \in I_{n-1}) = \frac{\Phi(r_n/\sigma) - \Phi(r_{n-1}/\sigma)}{\Phi(r_1/\sigma) - \Phi(r_0/\sigma)}
\]
(10)
where \( \Phi(\cdot) \) is the CDF of the standard normal distribution. The number of bits for encoding \( t_n \) is then given by
\[
N(t_n) = -\log_2 P(t_n|t_{n-1}, \cdots, t_1).
\]
(11)
Also, by the chain rule,
\[
P(y_c) = P(t_L, \cdots, t_1) = \prod_{n=1}^{L} P(t_n|t_{n-1}, \cdots, t_1).
\]
(12)
From (3), (11), and (12), we have
\[
N(y_c) = \sum_{n=1}^{L} N(t_n).
\]
(13)
In other words, the trit-plane coding of \( y_c \) requires the same number of bits as the straightforward coding does. However, it allows progressive reconstruction of \( y_c \). Suppose that the first \( n \) trits are received. Then, the decoder reconstructs \( y_c \) to
\[
\hat{y}_c^n = E[y_c|y_c \in I_n],
\]
(14)
which is the minimum mean squared error (MMSE) estimate \([16]\). In other words, \( \hat{y}_c^n \) in (14) yields the minimum mean squared distortion
\[
D_n = E[(y_c - \hat{y}_c^n)^2|y_c \in I_n].
\]
(15)

**Bit-plane coding:** Bit-planes may be used instead of trit-planes. However, note that the most frequent \( y_c \) is 0 because \( y_c \sim N(0, \sigma^2) \). When \( y_c = 0 \), \( \hat{y}_c^n \) in (14) is 0 regardless of \( n \), which enables faithful reconstruction even at a low bit-rate. This is impossible in the bit-plane coding, i.e. \( \hat{y}_c^n \neq 0 \) for \( n \leq L - 1 \) when \( y_c = \hat{y}_c^n = 0 \). Hence, we use trit-planes. It is shown in Section 4 that the trit-plane coding outperforms the bit-plane coding.

**Coding of \( \hat{Z} \):** We do not compress \( \hat{Z} \) progressively. Since the entropy coding of \( \hat{y} \) depends on \( \mu \) and \( \sigma \) estimated from \( \hat{Z} \), it is impossible to decompress \( \hat{y} \) from a partially reconstructed \( \hat{Z} \). Also, \( \hat{Z} \) occupies only about 1% of the overall bitrate. Hence, we transmit all bits for \( \hat{Z} \) before the progressive transmission of \( Y_c \).

The expected number of bits for encoding \( \hat{y}_c^n \) at level \( k \) is
\[
q_k = \frac{\text{Prob}(\text{rate} | \text{est} \in I_k)}{\text{Prob}(\text{rate} | \text{est} \in I_{k-1})}, \quad k = 0, 1, 2 \quad \text{via} \ (9).
\]
Then, the expected number of bits for encoding \( t_n \) is given by the entropy
\[
\Delta R = H(\{q_0, q_1, q_2\}),
\]
(16)
We use the ANS coder \([15]\) for the entropy coding. Also, for the three cases of \( t_n = 0, 1, 2 \), we compute the distortions via (15), respectively, which are denoted by \( D_n^0, D_n^1, D_n^2 \).
Then, the expected distortion change is given by
\[
\Delta D = E[D_n] - D_{n-1} = \sum_{k=0}^{2} q_k D^k_n - D_{n-1} \tag{17}
\]
where \(D^k_n\) is
\[
D^k_n = E[(y_c - \bar{y}_k)^2 | y_c \in \mathcal{X}^k_n] \tag{18}
\]
with \(\bar{y}_k = E[y_c | y_c \in \mathcal{X}^k_n]\). Figure 6 illustrates how to compute \(\bar{y}_k, D^k_n, \) and \(\Delta D\) with a toy example.

Finally, we compute the RD priority of the \(n\)th trit, which is defined as
\[
-\frac{\Delta D}{\Delta R} = \frac{\sum_{k=0}^{2} q_k D^k_n - D_{n-1}}{\sum_{k=0}^{2} q_k \log_2 q_k}. \tag{19}
\]

Then, we transmit the \(K\) trits in each trit-plane after sorting them in the decreasing order of their RD priorities. In this way, more important information is transmitted first, even in the same trit-plane.

### 3.4. Postprocessing network

To train the proposed algorithm, in (4), we use a uniform noise tensor in range \((-\frac{1}{2}, \frac{1}{2})\) to consider the case of using all \(L\) trit-planes. Thus, the network is less optimized for the cases of using fewer trit-planes. Figure 7 shows histograms of \(\hat{Y}_c^L, \hat{Y}_c^{L-1}, \) and \(\hat{Y}_c^{L-2}\). With fewer trit-planes, the gaps between reconstruction levels get wider, resulting in bigger quantization errors, which generate more artifacts in reconstructed images. Let \(\hat{X}^n\) denote a reconstructed image from a partial representation \(\hat{Y}^n\) consisting of \(\hat{y}^n\)’s. As shown in Figure 8(b), \(\hat{X}^{L-3}\) contains noisy artifacts.

We develop a postprocessing network \(g_p\) to reduce such artifacts. In other words, the goal of \(g_p\) is to convert a reconstructed image \(\hat{X}^n\) to a more faithful one \(\hat{X}^n + \Delta \hat{X}^n\) with less artifacts. Figure 9 shows the training schema of \(g_p\). In the upper pass, we first obtain \(\hat{Y}_c^n\) by \((14)\) and then reconstruct \(\hat{X}^n = g_s(\hat{Y}_c^n + M)\). Similarly, in the lower pass, we obtain the reconstructed image \(\hat{X}^L\) from \(\hat{Y}_c^L + M\) using all trit-planes. We regard \(\hat{X}^L\) as a clean image without artifacts. Then, we train the postprocessing network to take \(\hat{X}^n\) and yield the residual \(\Delta \hat{X}^n\), by employing the postprocessing loss
\[
\ell_p = \|\Delta \hat{X}^n - (\hat{X}^L - \hat{X}^n)\|_2 \tag{20}
\]

In the testing phase, using \(g_p\), we refine a reconstructed image \(\hat{X}^n\) into \(\hat{X}^n + g_p(\hat{X}^n)\). Figure 8(c) shows how \(g_p\) improves the reconstructed images \(\hat{X}^{L-3}\). We see that, compared to \(\hat{X}^{L-3}\), the refined images \(\hat{X}^{L-3} + \Delta \hat{X}^{L-3}\) contain less artifacts and render the scenes more faithfully.

### 4. Experimental Results

#### 4.1. Implementation

We develop a compression network similar to the Cheng et al.’s network [11], but we make some modifications to encode scalable bitstreams. First, we remove the autoregressive mask convolution, which predicts entropy parameters from a latent tensor. The autoregressive prediction assumes that the decoder can reconstruct the same latent tensor \(\hat{Y}\) in the raster scan order as the encoder does. However, in DPict, the information in \(\hat{Y}\) is progressively transmitted. Therefore, before the decoder reconstructs all trit-planes, the assumption fails and the entropy decoding breaks down. Second, we assume that each latent element \(y\) has a Gaussian distribution, instead of a Gaussian mixture model. This is to simplify the MMSE estimation and RD optimization in (9) and \((14)\sim(19)\).

A postprocessing network \(g_p\) also has the encoder-decoder architecture. It is implemented with residual blocks, attention modules, and subpixel convolution layers [11]. However, \(g_p\) has 35 layers only, while the encoder \(g_s\) and the decoder \(g_s\) in the compression network in Figure 2 have 68 layers. We train two postprocessing networks, targeting at different bit-rates. Note that \(\hat{X}^n\) denotes a reconstructed image using the first \(n\) trit-planes. When \(n\) is not an integer, \(\hat{X}^n\) means that, in addition to the first \(\lfloor n \rfloor\) trit-planes, \(100 \times (n - \lfloor n \rfloor)\) % of the trits in the \((\lfloor n \rfloor + 1)\)th trit-plane are used for the reconstruction. The first \(g_p\) targets at \(\hat{X}^n\) for \(n \in [0, L-2.9]\), and the second \(g_p\) for \(n \in (L-2.9, L-1.8]\).
We decided these ranges empirically and also observed that the postprocessing is not effective for $\hat{X}^n$, $n \in (L - 1.8, L]$, which is already of high quality.

For training, we use the Vimeo90k dataset [46]. Since it contains frames with overlapping contents, we sample 80,000 frames and randomly crop $256 \times 256$ patches from each sampled frame. We use the Adam optimizer [25] with a batch size of 16, a learning rate of $2 \cdot 10^{-5}$, and $\lambda = 5$. We perform the scheduled learning according to cosine annealing cycles [21]. We train the compression network for 200 epochs and then the postprocessing networks for 20 epochs.

For evaluation, we use the Kodak lossless image dataset [1] and the CLIC professional validation dataset [2]. The Kodak dataset consists of 24 images of resolution $512 \times 768$ or $768 \times 512$, while the CLIC dataset contains 41 images of higher quality up to 2K resolution. For each image, we determine the number $L$ of trit-planes to cover the values of all elements, after truncating outliers, in $\hat{Y}_c$. We measure the rate by bits per pixel (bpp), and the distortion by peak-to-signal ratio (PSNR) and multi-scale structural similarity (MS-SSIM) [45]. For MS-SSIM, we convert it to dB scale by $\text{MS-SSIM (dB)} = -10 \cdot \log_{10}(1 - \text{MS-SSIM})$.

We conduct all experiments using Pytorch [34] and CompressAI [7]. Network architecture and implementation details are available in the supplemental document.

### 4.2. Performance comparison

First, we compare the proposed DPICT with conventional progressive codecs: JPEG [44] and JPEG2000 [37] and learning-based progressive codecs [10, 14, 22, 24, 38, 41]. Figure 10 plots the RD curves on the Kodak dataset. At every rate, DPICT provides the highest PSNR and the
highest MS-SSIM, outperforming the second best codecs JPEG2000 and Johnson et al. [24] considerably. For example, at 0.75bpp, DPICT yields about 1.7dB higher PSNR than JPEG2000, and about 1.1dB higher MS-SSIM than Johnson et al. Moreover, it is worth pointing out that DPICT is the only learning-based codec with FGS. The same bitstream of DPICT for an image is reconstructed at 164 different rates, as indicated by red dots in Figure 10, which can be increased further if needed. On the other hand, Su et al. [38] support coarse granular scalability for four rates only, and their rate range is much narrower than that of DPICT. Cai et al. [10] support two rates only, one for preview images and the other for full-quality images. The RNN-based codecs [24, 41] support 16 rates, but their PSNR performances are poorer than those of JPEG2000.

Next, Figure 11 compares DPICT with non-progressive codecs [6, 8, 11, 13, 18, 30, 47]. Despite the additional functionality of FGS, DPICT outperforms the two traditional codecs, WebP and BPG, and is competitive with the state-of-the-art learning-based codecs. Figure 12 compares RD curves on the CLIC dataset. We see that DPICT outperforms progressive codecs [37,44] significantly and competes with Cheng et al. [11], which is a state-of-the-art non-progressive codec.

Figure 13 compares reconstructed images at similar rates. In areas with complicated texture and sharp edges, such as dirt floor or patterned window, the traditional codecs yield blurring artifacts, but DPICT restores high quality images without noticeable artifacts.

Figure 14 shows reconstructed images at different rates from a single bitstream. At the top of the figure, we specify how many trit-planes are decoded. At all rates, DPICT provides better qualities than the progressive JPEG2000.

4.3. Ablation study

We analyze the compression performance of DPICT in Figure 15. Here, each curve represents the result of replacing or removing certain components of DPICT. First, we compare the proposed DPICT with four baselines excluding the trit-plane coding. In ‘without sorting,’ latent elements are transmitted channel by channel in the raster scan order without priority. This approach underperforms badly. In ‘channel-wise sorting,’ the $C$ channels of the latent tensor are sorted and transmitted according to the RD priorities. Instead of channels, ‘latent-wise sorting’ sorts the $K = H \times W \times C$ latent elements. These alternatives can support progressive transmission, but are much inferior to DPICT. ‘Bit-plane’ is the result of replacing the trit-plane coding with the bit-plane coding. As mentioned in
Section 3.2, $\hat{g}_c^n$ cannot be reconstructed to 0 in the bit-plane coding unless $n = L$. This causes significant performance degradation at low rates, i.e. when a bitstream is partially decoded. These results indicate that the trit-plane coding is an essential component of DPICT.

Second, we change the training schema. ‘Multi-rate’ is the result of changing the loss function so that the compression network is trained for multiple rates. Specifically, we replace the loss function in (5) with

$$\ell = \sum_{n=L-3}^{L} \left( \ell_D(X, \hat{X}^n) + \lambda_n \ell_R(Y^n, \hat{Z}; \hat{M}, \hat{\Sigma}) \right)$$

(21)
to consider partially reconstructed $\hat{X}^{L-3}, \hat{X}^{L-2}, \hat{X}^{L-1}$, as well as fully reconstructed $\hat{X}^L = X$. Note that loss functions for different rates are also combined in [13,47]. However, ‘multi-rate’ severely narrows the range of supported rates and is effective only at low rates.

Third, we replace the prioritized transmission. ‘Without priority’ is the result of not using the RD priority in (19). In this case, trits in each trit-plane are transmitted in the raster scan order. We see that the prioritized transmission is essential for reconstructing high quality $\hat{X}^n$ when $n$ is not an integer. ‘Reverse priority’ transmits the trits in a trit-plane in the increasing order of the RD priorities. It yields even poorer performance than ‘without priority.’

Last, ‘without $g_p$’ is the result of not using the postprocessing networks. Note that the postprocessing networks improve the quality of a reconstructed image when the rate is lower than about 0.4bpp. Figure 16 shows the impacts of the postprocessing networks $g_p$. For easier comparison, improvement maps are also provided. When $L - 2$ or fewer trit-planes are used, the postprocessing networks consistently improve reconstructed images both qualitatively and quantitatively. On the other hand, for reconstructed images using more than $L - 2$ trit-planes, they do not provide clear improvements. Thus, we use the postprocessing networks when fewer than $L - 1.8$ trit-planes are decoded.

We provide more experimental results and analysis in the supplemental document. Also, we describe how to implement the proposed DPICT algorithm efficiently in [23].

5. Conclusions

In this paper, we proposed the DPICT algorithm supporting FGS. In DPICT, an image is transformed into a latent tensor using an analysis network. The latent tensor is then represented in a ternary number system and is encoded trit-plane by trit-plane in the decreasing order of significance. Furthermore, in each trit-plane, the trits are transmitted in the decreasing order of the RD priorities. We also developed the post-processing networks to reduce artifacts due to quantization errors when fewer trit-planes are used. Experiments demonstrated that the proposed DPICT provides state-of-the-art performance by outperforming other progressive codecs significantly.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. NRF-2021R1A4A1031864 and No. NRF-2022R1A2B5B03002310).
References


