Reduce Information Loss in Transformers for Pluralistic Image Inpainting
Supplementary Material

Qiankun Liu\textsuperscript{1*} Zhentao Tan\textsuperscript{1} Dongdong Chen\textsuperscript{2} Qi Chu\textsuperscript{1†} Xiyang Dai\textsuperscript{2}
Yinpeng Chen\textsuperscript{2} Mengchen Liu\textsuperscript{2} Lu Yuan\textsuperscript{2} Nenghai Yu\textsuperscript{1}
\textsuperscript{1}University of Science and Technology of China
\textsuperscript{2}Microsoft Cloud + AI
\{liuqk3, tzt\}@mail.ustc.edu.cn, \{qchu, ynh\}@ustc.edu.cn
ccdlyf@gmail.com, \{xiyang.dai, yiche, mengcliu, luyuan\}@microsoft.com

1. Overview

In this supplementary material, we provide more implementation details, experimental results and analysis, including:

- training of P-VQVAE (Section 2).
- sampling strategy for image inpainting (Section 3).
- network architecture of different models (Section 4).
- more results on different datasets (Section 5).
- more discussions on PUT (Section 6), including the inference speed of PUT and some artifacts in inpainted results.

2. Training of P-VQVAE

Given an image \( \mathbf{x} \) and two different masks \( \mathbf{m} \) and \( \mathbf{m}' \), the input of P-VQVAE is \( \hat{\mathbf{x}} = \mathbf{x} \otimes \mathbf{m} \). The overall loss for the training of P-VQVAE is:

\[
L_{\text{vae}} = L_{\text{rec}}(\hat{\mathbf{x}}, \hat{\mathbf{x}}') + \| \text{sgn} (\hat{\mathbf{f}}) \otimes \hat{\mathbf{e}} \|_2^2 + \beta \| \text{sgn} (\mathbf{e}) \otimes \hat{\mathbf{f}} \|_2^2,
\]

where \( \hat{\mathbf{e}} = \mathcal{E}(\hat{\mathbf{x}}) \) denotes the feature vectors extracted by the encoder and \( \hat{\mathbf{e}} \) is quantized vectors for \( \hat{\mathbf{f}} \). \( \hat{\mathbf{x}}' = \mathcal{D}(\hat{\mathbf{e}}, \mathbf{m} \otimes \mathbf{m}', \hat{\mathbf{x}} \otimes \mathbf{m}') \) is the reconstructed image and \( \text{sgn} \) refers to a stop-gradient operation that blocks gradients from flowing into its argument.

The last term in Eq. (1) is the so-called commitment loss \cite{4} with weighting factor \( \beta = 0.25 \). It is responsible for passing gradient information from decoder to encoder. The second term in Eq. (1) is the codebook loss for the optimization of latent vectors. Following previous works in \cite{13,15}, we replace the second term with the Exponential Moving Average (EMA) to optimize \( \mathbf{e} \) and \( \mathbf{e}' \).

\[
\begin{align*}
\mathbf{n}_k & = n_{k-1} \gamma + n_k \lambda \left(1 - \gamma\right), \\
\mathbf{e}_{k} & = \mathbf{e}_{k-1} \gamma + \sum_{j} (\hat{\mathbf{f}}_k)^j \lambda \left(1 - \gamma\right), \\
\mathbf{e}_k & = \frac{\mathbf{e}_k}{\| \mathbf{e}_k \|}.
\end{align*}
\]

where \( \hat{\mathbf{f}}_k \) denotes the set of feature vectors in \( \hat{\mathbf{f}} \) that assigned to \( \mathbf{e}_k \) and \( n_k \) is the number of feature vectors in \( \hat{\mathbf{f}}_k \). \( \gamma \) is the
Table 1. Architecture of P-VQVAE. For MSG-Dec, the bracketed layers in the bottom four rows denotes the layers in reference branch. Except the convolution layer marked by †, all the other layers are followed by a ReLU [9] activation function. The structure of Linear and Conv ResBlocks are shown in Figure 1.

<table>
<thead>
<tr>
<th>Module</th>
<th>Layer</th>
<th>Parameter size / Stride</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv-Enc</td>
<td>Linear</td>
<td>192 × 256</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td></td>
<td>ResBlock</td>
<td>(256 × 128) × 8</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td></td>
<td>Linear</td>
<td>256 × 256</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td>D-Codes</td>
<td>e</td>
<td>512 × 256</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>e′</td>
<td>512 × 256</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Conv</td>
<td>256 × 3 × 3 × 256/1</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td></td>
<td>ResBlock</td>
<td>(256 × 3 × 3 × 128/1) × 8</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td></td>
<td>Deconv (Conv)</td>
<td>256 × 4 × 4 × 256/2</td>
<td>64 × 64 × 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(256 × 4 × 4 × 256/2)</td>
<td>(32 × 32 × 256)</td>
</tr>
<tr>
<td></td>
<td>Deconv (Conv)</td>
<td>128 × 4 × 4 × 256/2</td>
<td>64 × 64 × 256</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(128 × 4 × 4 × 256/2)</td>
<td>(64 × 64 × 256)</td>
</tr>
<tr>
<td></td>
<td>Deconv (Conv)</td>
<td>64 × 4 × 4 × 128/2</td>
<td>256 × 256 × 64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(64 × 4 × 4 × 128/2)</td>
<td>(128 × 128 × 256)</td>
</tr>
<tr>
<td></td>
<td>Conv1</td>
<td>(3 × 3 × 3 × 64/1)</td>
<td>256 × 256 × 3</td>
</tr>
<tr>
<td></td>
<td>Conv</td>
<td>(3 × 3 × 3 × 64/1)</td>
<td>(256 × 256 × 64)</td>
</tr>
</tbody>
</table>

Table 2. Architecture of the encoder in P-VQVAE_{conv}. The learnable codebook and decoder are the same with those in P-VQVAE in Table 1. All layers are followed by a ReLU [9] activation function.

<table>
<thead>
<tr>
<th>Module</th>
<th>Layer</th>
<th>Parameter size / Stride</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conv-Enc</td>
<td>Conv</td>
<td>3 × 4 × 4 × 64/2</td>
<td>128 × 128 × 64</td>
</tr>
<tr>
<td></td>
<td>Conv</td>
<td>64 × 4 × 4 × 128/2</td>
<td>64 × 64 × 128</td>
</tr>
<tr>
<td></td>
<td>Conv</td>
<td>128 × 4 × 4 × 256/2</td>
<td>64 × 128 × 256</td>
</tr>
<tr>
<td></td>
<td>ResBlock</td>
<td>(256 × 3 × 3 × 128/1) × 8</td>
<td>32 × 32 × 256</td>
</tr>
<tr>
<td></td>
<td>Conv</td>
<td>256 × 3 × 3 × 256</td>
<td>32 × 32 × 256</td>
</tr>
</tbody>
</table>

decay parameter with the value between 0 and 1. We set γ = 0.99 in all our experiments.

The first term in Eq. (1) is the reconstruction loss and \( L_{rec}(\cdot) \) is the function to get the difference between the inputted and reconstructed images. It consists of five parts, including L1 loss between the pixel values in two images (denoted as \( L_{pixel} \)) and the gradients of two images (denoted as \( L_{grad} \)), the adversarial loss \( L_{adv} \), as well as the perceptual loss [5] \( L_{perc} \) and style loss [4] \( L_{style} \) between the two images. The design of the last three losses are inspired by the work in [10]. In the following, we describe the aforementioned losses in detail. Among them:

\[
L_{pixel} = \mathcal{M}([\hat{x} \odot \hat{x}^R]), 
\]

\[
L_{grad} = \mathcal{M}([\text{grad}[\hat{x}] \odot \text{grad}[\hat{x}^R]]),
\]

where \( \mathcal{M}(\cdot) \) refers to a mean-value operation, \( \text{grad}[\cdot] \) is the function calculating the gradient of the given image.

The adversarial loss \( L_{adv} \) is computed with the help of a discriminator network \( D_{adv}(\cdot) \):

\[
L_{adv} = -\mathcal{M}(\log[1 \odot D_{adv}(\hat{x}^R)]) - \mathcal{M}(\log[D_{adv}(\hat{x})]),
\]

where \( \log[\cdot] \) denotes element-wise logarithm operation. The architecture of the discriminator network is the same with that in [10].

The conceptual loss \( L_{perc}(\cdot) \) and style loss \( L_{style}(\cdot) \) are computed based on the activation maps from VGG-19 [14]:

\[
L_{perc} = \sum_{l} \mathcal{M}(\|\phi_l(\hat{x}) \odot \phi_l(\hat{x}^R)) \]

\[
L_{style} = \sum_{l} \mathcal{M}(\|\mathcal{G}(\phi_l(\hat{x})) \odot \mathcal{G}(\phi_l(\hat{x}^R)))
\]

where \( \phi_l(\cdot) \) corresponds to different layers in VGG-19 [14], \( \mathcal{G}(\cdot) \) denotes the function that gets the Gram matrix of its argument. For \( L_{perc} \) and \( L_{style} \), we set \( L_{perc} = \{\text{relu1}_1, \text{relu2}_1, \text{relu3}_1, \text{relu4}_1, \text{relu5}_1\} \) and \( L_{perc} = \{\text{relu2}_2, \text{relu3}_4, \text{relu4}_4, \text{relu5}_2\} \). The overall reconstruction loss is:

\[
L_{rec} = L_{pixel} + \lambda_p L_{grad} + \lambda_a L_{adv} + \lambda_p L_{perc} + \lambda_s L_{style}
\]

In our implementation, we set \( \lambda_p = 5, \lambda_a = 0.1, \lambda_p = 0.1 \) and \( \lambda_s = 250 \).

### 3. Sampling Strategy for Image Inpainting

The overall procedure can be divided into three steps: 1) get the feature vectors \( \hat{f} \) from the masked image \( \hat{x} \) using encoder and get the tokens \( \hat{t} \) by quantizing \( \hat{f} \) with latent vectors in dual-codebook. The tokens for masked patches are not required; 2) get the tokens for masked patches using transformer. Note that the tokens are iteratively sampled with Gibbs sampling following previous transformer-based works [3, 11, 12]; 3) retrieve quantized vectors \( \hat{e}^t \) from codebook \( e \) based on the tokens and reconstruct the inpainted image \( \hat{x}^t \) using decoder by referencing to masked image \( \hat{x} \). The detailed sampling strategy is shown in Algorithm 1.
Figure 1. Architecture of different blocks. For Linear and Conv ResBlocks, each layer is followed by a ReLU [9] activation function. For transformer block, there is a GELU [6] activation function between the two linear layers. MSA: Multi-head Self-Attention. MLP: Multi-Layer Perceptron.

4. Network Architecture

4.1. Auto-Encoder

For different datasets, we use P-VQVAE with the same model size, and the architecture of our default P-VQVAE is shown in Table 1. The structure of Linear and Conv ResBlocks are shown in Figure 1 (a) and (b). In the paper, Section 4.3, several models are designed to show the effectiveness of different components in our method, including PUT\textsuperscript{conv}, PUT\textsuperscript{one}, PUT\textsuperscript{no,ref}, PUT\textsuperscript{mask} and PUT\textsuperscript{tok}. The auto-encoders in the last two models are the same with our default P-VQVAE. However, the auto-encoders in PUT\textsuperscript{conv}, PUT\textsuperscript{one} and PUT\textsuperscript{no,ref} are different. For the auto-encoder in PUT\textsuperscript{conv} (denoted as P-VQVAE\textsuperscript{conv}), all the linear layers in the encoder are replaced with convolution layers, and the input image is processed in a sliding window manner. Other modules in P-VQVAE\textsuperscript{conv} are the same with those in P-VQVAE. The architecture of encoder in P-VQVAE\textsuperscript{conv} (denoted as Conv-Enc) is shown in Table 2. The architecture of the auto-encoder in PUT\textsuperscript{one} is the same with P-VQVAE, except only one codebook $e$ is used for training and testing. While for the auto-encoder in PUT\textsuperscript{no,ref}, it can be obtained from P-VQVAE by removing the reference branch in decoder.

4.2. Transformer

The architecture of transformer block is depicted in Figure 1 (c). There are several (denoted as $n'$) successive transformer blocks in UQ-Transformer. Within each transformer block, the input features will be enhanced by self-attention. Formally, let $\tilde{f} \in \mathbb{R}^{HW \times D}$ be the input of transformer block. At the $b$-th transformer block, the feature vectors are processed as:

$$
\tilde{f}^{b-1} = \tilde{f}^{b-1} + \text{MSA}(\text{LN}(\tilde{f}^{b-1})),
$$
$$
\tilde{f}^b = \tilde{f}^{b-1} + \text{MLP}(\text{LN}(\tilde{f}^{b-1})),
$$

(9)

where LN(·), MLP(·), MSA(·) denote layer normalization [1], multi-layer perceptron and multi-head self-attention respectively. More specifically, given input $f \in \mathbb{R}^{HW \times D}$, MSA(·) could be formatted as:

$$
h_j = \text{softmax}(\frac{(fw_0^j)(fw_1^j)}{\sqrt{D'}}),
$$

where $h$ is the number of head, $w_0^j, w_1^j \in \mathbb{R}^{D \times D'}$, $w_o \in \mathbb{R}^{hD' \times D}$ are the learnable parameters. $[: ; :]$ is the operation that concatenates the given arguments along the last dimension. By changing the values of $h, D, D'$ and $n'$, we can easily scale the size of UQ-Transformer.

We use UQ-Transformer with different model sizes for different datasets, which are shown in Table 3. As a reminder, the configuration of transformers are the same with those in ICT [17].

5. More Results

We show more qualitative comparisons for FFHQ [8] (Figure 3), Places2 [18] (Figure 4) and ImageNet [2] (Figure 5 and Figure 6).

6. More Discussions

**Inference speed.** As mentioned in Section 5 in the paper, the main limitation of PUT is the inference speed, which is also a common issue of existing transformer-based auto-regressive methods [3, 12, 16, 17]. Here we present the inference speed of PUT in Table 4. Note that the time consumption of inpainting a masked image depends on the area of masked regions.

**Artifacts.** We experimentally find that there sometimes contain some artifacts in the generated results of PUT, as

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>UQ-Transformer</td>
<td>37.138</td>
<td>32.048</td>
<td>17.186</td>
</tr>
<tr>
<td>( # tokens/second)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P-VQVAE</td>
<td></td>
<td></td>
<td>62.949</td>
</tr>
<tr>
<td>( # images/second)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
shown in Figure 2. These artifacts can be divided into two categories. 1) Color distortion: the color of generated contents may not be consistent with the color of provided contents in the image. 2) Black region: PUT may produce black regions if the provided masked image contain lots of black pixels.

Figure 2. Results with artifacts. Top: color distortion. Bottom: black regions. Please pay attention to the contents in yellow rectangles.
Figure 3. Qualitative comparisons between different methods on FFHQ [8].
Figure 4. Qualitative comparisons between different methods on Places2 [18].
Figure 5. Qualitative comparisons between different methods on ImageNet [2].
Figure 6. Qualitative comparisons between different methods on ImageNet [2].
References


