Learning Image-Adaptive Codebooks for Class-Agnostic Image Restoration

Kechun Liu\textsuperscript{1,3}\textsuperscript{*}, Yitong Jiang\textsuperscript{2}, Inchang Choi\textsuperscript{3}, Jinwei Gu\textsuperscript{2}

\textsuperscript{1} University of Washington \hspace{1em} \textsuperscript{2} The Chinese University of Hong Kong \hspace{1em} \textsuperscript{3} SenseBrain

Abstract

Recent work on discrete generative priors, in the form of codebooks, has shown exciting performance for image reconstruction and restoration, as the discrete prior space spanned by the codebooks increases the robustness against diverse image degradations. Nevertheless, these methods require separate training of codebooks for different image categories, which limits their use to specific image categories only (e.g., face, architecture, etc.), and fail to handle arbitrary natural images. In this paper, we propose AdaCode for learning image-adaptive codebooks for class-agnostic image restoration. Instead of learning a single codebook for each image category, we learn a set of basis codebooks. Given an input image, AdaCode learns a weight map with and computes a weighted combination of these basis codebooks for adaptive image restoration. Intuitively, AdaCode is a more flexible and expressive discrete generative prior than previous work. Experimental results demonstrate that AdaCode achieves state-of-the-art performance on image reconstruction and restoration tasks, including image super-resolution and inpainting. Codes are released at \url{https://github.com/kechunl/AdaCode}

*kechun@cs.washington.edu
Work done during an internship at SenseBrain.

Figure 1: We propose an image-adaptive codebook learning method, named AdaCode, for class-agnostic image restoration. For a number of image reconstruction and restoration tasks (e.g., super-resolution and inpainting), the proposed AdaCode achieved significantly better performance than latest prior work, such as VQGAN\cite{7} and VQGAN with our merged basis codebooks (referred to as VQGAN-aux) for reconstruction; KX-Net\cite{9}, Real-ESRGAN\cite{42} and FeMaSR\cite{3} for super-resolution; GPEN\cite{52}, MAT\cite{19} and our trained FeMaSR\cite{3} for image inpainting. (Zoom in for best view)
1. Introduction

In recent years, discrete generative priors (in the form of codebooks) [39, 7] have shown impressive performance for image synthesis [7, 11, 45, 61], exhibiting reduced mode collapse and more stable training. These learned codebooks essentially provide strong priors for compressing and reconstructing natural images, even in the presence of severe degradation. Nevertheless, these methods have a common limitation. The codebooks need to be learned separately for each image category (e.g., face, architecture), which restricts their applicability to arbitrary natural images [7, 45, 61]. Although FeMaSR [3] attempted to learn a single general codebook for all image categories, the expressiveness of the codebook is limited by the complexity of natural images. For example, as shown in Fig. 1, an image often includes textural and structural contents from multiple categories (e.g., face, man-made structural edges, repetitive texture, natural texture). It is challenging to rely on a single universal codebook to capture all. Prior work such as VQGAN [7] and FeMaSR [3] often introduce noticeable artifacts for image reconstruction and restoration.

Is it possible to learn a class-agnostic discrete generative prior for image reconstruction and restoration? Inspired by a recent work [55], we propose AdaCode, which learns image-adaptive codebooks for class-agnostic image reconstruction and restoration. Instead of learning a single codebook for all categories of images, we learn a set of basis codebooks. For a given input image, AdaCode learns a weight map that determines the contribution of each basis codebook to the final representation. Intuitively, this design allows AdaCode to learn a more flexible and expressive discrete generative prior than previous work, as demonstrated in Fig. 2. In contrast to VQGAN [7] and FeMaSR [3], which utilize a single partition for the latent space and assign each image feature an exclusive discrete representation, AdaCode learns various partitions to the latent space from different perspectives — each corresponding to the learning of one of the basis codebooks. The discrete generative prior for an arbitrary image is a weighted linear combination derived from these basis codebooks, resulting in a more flexible and expressive representation. As depicted in Fig. 1, AdaCode outperforms previous work in various image restoration tasks, effectively preserving scene structure and texture.

We evaluated AdaCode on both image reconstruction and image restoration tasks (i.e., super-resolution and image inpainting). Across multiple benchmark datasets, AdaCode achieved state-of-the-art performance, while maintaining a comparable codebook size and computational cost.

2. Related Work

Visual Representation Dictionary Learning

Learning representation dictionaries in visual understanding has demonstrated its great power in image restoration tasks such as super-resolution [51, 62], denoising [59], and image inpainting [8, 36]. Using DNNs, VQVAE [39] first introduces a generative autoencoder model that learns discrete latent representations, also known as “codebook”. The following VQGAN [7] employs perceptual and adversarial loss to train the visual codebook, resulting in better image generation quality with a relatively small codebook size. The representation dictionary-based generative model has inspired various impressive image generation work [11, 61, 53, 45], as well as our AdaCode.

The use of dictionaries is not limited to image restoration. Referenced as lookup tables (LUTs), the dictionaries are also applied to optimize color transforms [55, 23, 49, 50]. In 3D-LUT [55], multiple LUTs are learned to serve as the bases of the LUT space. And a CNN is trained to predict weights to fuse the bases into an image-adaptive LUT. Inspired by 3D-LUT, we leverage the discrete codebooks from VQGAN as the bases of the image latent space to build our image-adaptive codebook, AdaCode. Such design allows our method to fully and flexibly exploit the latent codes to represent diverse and complex natural images.
3. Methodology

DNN-based Image Restoration  DNN has been widely used for image restoration tasks, i.e. single image super-resolution (SISR) and image inpainting. In the field of super-resolution (SR), recent studies focus on recovering low-resolution input images with unknown degradation types, which is more relevant to real-world scenarios. They restore the images either by learning the degradation representations [33, 21, 41, 42, 57, 9], or by training unified generative networks with LR-HR pairs [16, 46, 32]. Some studies introduce additional image priors, such as latent representations [27, 2, 30] and discrete codebooks [7, 61, 3], to address unrealistic textures or over-smoothed areas commonly observed in GAN-based methods. However, a single partition of the latent space is often insufficient to model the intricate patterns in natural images, resulting in specific textures being generated regardless of the image content.

In the case of image inpainting, researchers often leverage deep generative models to fill missing image regions with plausible content [31, 25, 44, 56, 5]. Some approaches incorporate additional discriminators [35], partial or gated convolutions [24, 53], semantic texture or context [28, 48, 34, 13, 38, 15], or transformers [19, 6, 26, 40] to enhance the quality of inpainted results. However, these methods often require separate experiments for different image patterns, i.e. natural scenes, faces, etc., due to the significant variations among them.

Both SR and inpainting methods face the challenge of effectively modeling complex visual patterns with a single model or codebook. In our proposed approach, AdaCode, we address this challenge by leveraging adaptive codebooks, enabling realistic and robust restoration results for general images.

3.1. Codebook Pretraining (Stage I)

Diversify Basis Codebooks  To enhance the expressiveness of AdaCode, we aim to diversify the basis codebooks. Rather than applying various initializations as 3D-LUT [55], we achieve codebook divergence by training them on different HR subsets of our dataset. To accomplish this, we utilize an off-the-shelf SegFormer model [47] to perform semantic segmentation with 150 classes from the ADE20K dataset [60]. Each HR patch is labeled according to the semantic class with the largest area. We then group the 150 classes into 5 super-classes: Architectures, Indoor objects, Natural scenes, Street views, Portraits, and obtain the 5 semantic HR subsets accordingly. It is worth noting that the separation of subsets is not rigorous and each subset may
Learning Basis Codebooks. Given a HQ subset of class \( k \), we train a quantized autoencoder to learn the class-specific basis codebook. As shown in Fig. 3, the input HR patch \( y \in \mathbb{R}^{H \times W \times 3} \) is first passed through the encoder \( E \) to generate the embedding \( \hat{z} = E(y) \in \mathbb{R}^{h \times w \times n_z} \). Following VQVAE \(^{[39]}\) and VQGAN \(^{[7]}\), each entry \( \hat{z}_i \in \mathbb{R}^{n_z} \) in \( \hat{z} \) is replaced with its nearest code in the learnable codebook \( Z_k \in \mathbb{R}^{N \times n_z} \) to construct the quantized embedding \( z_{qk} \):

\[
z_{qk} = \arg \min_{c \in \{0, \ldots, N-1\}} \| \hat{z} - Z_{k,c} \|
\]

where \( N \) is the number of codes in the corresponding codebook, \( z_{qk} \) denotes the quantized representation using \( Z_k \), and \( Z_{k,c} \) represent the \( c \)-th entry in codebook \( Z_k \). After the feature quantization, the decoder \( G \) reconstructs the HR patch \( \hat{y} \) using \( z_{qk} \):

\[
\hat{y} = G(z_{qk}) \approx y
\]

The adversarial learning scheme is employed to train the encoder \( E \), codebook \( Z \), and decoder \( G \) with the discriminator \( D \). The detailed architectures of \( E \), \( D \), and \( G \) are provided in the Appendix.

Training Objective. To train the quantized autoencoder, we adopt 3 image-level losses: L1 loss \( \mathcal{L}_1 \), perceptual loss \( \mathcal{L}_{per} \) \(^{[17]}\), and adversarial loss \( \mathcal{L}_{adv} \) \(^{[10]}\), which are calculated using \( \hat{y} \) and \( y \).

Since the quantization in Eqn. 1 is non-differentiable, we adopt the straight-through gradient estimator in \(^{[39]}\)\(^{[7]}\), which directly copies the gradients from decoder \( G \) to encoder \( E \), enabling back-propagation and allowing end-to-end training using the code-level loss function \( \mathcal{L}_{VQ} \):

\[
\mathcal{L}_{VQ}(E, G, Z_k) = \| sg[\hat{z}] - z_{qk} \|_2^2 + \beta \cdot \| \hat{z} - sg[z_{qk}] \|_2^2
\]

where \( sg[\cdot] \) denotes the stop-gradient operation and \( \beta = 0.25 \) is a hyper-parameter to control the update frequency of the codebook.

To further reinforce the semantics in the latent codebook and improve the texture restoration \(^{[43]}\), we incorporate a VGG19-based regularization term \( \mathcal{L}_{sem} \) into the codebook training process, following the approach in \(^{[3]}\):

\[
\mathcal{L}_{sem} = \| CONV(\hat{z}) - \Phi(y_k') \|_2^2
\]

where \( \Phi \) denotes the feature extractor of VGG19 \(^{[37]}\), and \( CONV \) denotes a single convolutional layer to match the dimension of \( \hat{z} \) and \( \Phi(y_k') \).

With the above image-level and code-level losses, we can summarize the training objective in Stage I as:

\[
\mathcal{L}_{stage1} = \mathcal{L}_1 + \mathcal{L}_{per} + \lambda \cdot \mathcal{L}_{adv} + \mathcal{L}_{VQ} + \lambda \cdot \mathcal{L}_{sem}
\]

where the loss weight \( \lambda \) is set to 0.1.

3.2. AdaCode Representation Learning (Stage II)

Given the pretrained class-specific basis codebooks \( Z_k, k \in \{1, \ldots, K\} \), the latent feature space can be partitioned into non-overlapping cells in \( K \) different ways. Specifically, for a given input HR patch \( y \), \( K \) quantized representations can be generated. Each distinct quantized representation \( z_{qk} \) obtains its code token from its corresponding semantic codebook. To combine the discrete representation \( z_{q1}, \ldots, z_{qK} \) into the AdaCode representation \( z \), we employ a weight predictor module, which generates a \( K \)-channel weight map \( w \in \mathbb{R}^{h \times w \times K} \), as illustrated in Fig. 3 and Fig. 5. The weight predictor module consists of four residual swin transformer blocks (RSTBs) \(^{[20]}\) and a convolution layer to match the channels of weight map and \( K \). \( z \) is computed following Eqn. 6. Finally, the adaptive feature \( z \) is reconstructed to HR patch \( \hat{y} \) via the decoder \( G \).

\[
z = \sum_i w_i \times z_{qi}
\]

To efficiently train AdaCode and maintain a comparable number of parameters in the codebooks to VQGAN \(^{[7]}\) and FeMaSR \(^{[3]}\), which both set the codebook dimension to be \( 1024 \times 512 \), we set each of our class-specific codebooks to be \( 256 \times 256 \) or \( 512 \times 256 \). These codebooks are fixed during the training of stage II while the rest of the model, \( i.e. \) the encoder \( E \), the weight predictor, the decoder \( G \), and the discriminator \( D \), are trained using the objective in Eqn. 7. Each term in this equation is defined in Sec. 3.1.

\[
\mathcal{L}_{stage2} = \mathcal{L}_1 + \mathcal{L}_{per} + \lambda \cdot \mathcal{L}_{adv} + \mathcal{L}_{VQ}(E, G)
\]
Figure 5: An example showing the learned weight maps. The input image contains multiple semantically-meaningful content (e.g., pyramid, person, animal, sky) which cannot be well represented with a single codebook. Instead, AdaCode uses multiple basis codebooks and weight maps for discrete representations. As shown, the weight maps correlate to the semantics to some extent.

### 3.3. Restoration via AdaCode (Stage III)

With the powerful decoder $G$, various image restoration tasks can be turned into a feature refinement problem through AdaCode scheme. From the perspective of latent space partition, each representation of the degraded input $x$ is pulled towards its nearest HR code entry, allowing for the information loss in $x$ to be relatively compensated. In comparison to the quantized representation $z_q$ using only one general codebook, the combination of $z_{q1},...,z_{qk}$ with the weight map can be considered as adding an offset to $z_q$. The offset helps to alleviate the discontinuity among the discrete codes, which is demonstrated in our ablation study. To further showcase the effectiveness of AdaCode, we train our model on two ill-posed problems, i.e., Single Image Super-Resolution and Image Inpainting.

In super-resolution and image inpainting tasks, the mapping between the input and the HR output has more than one solutions. Despite the benefits of the AdaCode scheme, restoring damaged content or missing details remains challenging given the uncertain degradations and the diversity of natural images. To better account for the degradation and improve the gradient flow, we adopt the encoder design in [3] which utilizes a feature extraction module and a residual shortcut module during stage III.

Thanks to the excellent reconstruction model in stage II, given a Degraded-HR image pair, we can obtain the groundtruth representation $z_{gt}$ via the fixed model. Since the decoder $G$ is fixed in this stage, the restoration problem can be formulated as minimizing the distance between HR feature $z_{gt}$ and degraded feature $z$. To achieve this, we use a code-level loss that includes the InfoNCE loss in [29] and the style loss in [17]. Following the design as SimCLR [4], given a degraded image feature $z$, we use the HR feature $z_{gt}$ as the positive sample, while other $z_{qj}$ and $z$ from different source images in the same batch are treated as the negative samples. The code-level loss is defined as follows.

$$L_{code} = L_{InfoNCE}(z_{gt}, z) + L_{style}(z_{gt}, z) + \beta \cdot ||\hat{z} - sg[z_{gt}]||^2_2$$  \hspace{1cm} (8)

And the overall loss is summarized as:

$$L_{stage3} = L_1 + L_{per} + \lambda \cdot L_{adv} + L_{code}$$  \hspace{1cm} (9)

### 4. Experiments

#### Datasets
Our training dataset includes images from DIV2K train set [1], Flickr2K [22], DIV8K train set [12], and 10,000 face images from FFHQ [18]. We generate the training patches by cropping images into non-overlapping patches at $512 \times 512$ resolution (face images in FFHQ are randomly resized with scale factors between [0.5, 1.0] before cropping). We adopt the same degradation model as BSRGAN [57] to generate LR patches. The final training dataset consists of 198,061 patches.

In the test stage, we evaluate the reconstruction task on OST dataset [43], which contains 300 images with rich textures. For super-resolution, we evaluate the performance on five classical benchmarks, i.e., Set5, Set14, BSD100, Urban100, and Manga109, with $\times2$ and $\times4$ scales. For image inpainting, we apply a publicly available script [52] to randomly draw irregular polylines masks and generate masked images. The inpainting performance is evaluated on the validation sets of DIV2K [1] and DIV8K [12].

#### Evaluation Metrics
For reconstruction, we adopt PSNR and SSIM as the evaluation metrics. For super-resolution, we employ an additional well-known perceptual score, LPIPS [58]. For image inpainting task, we use PSNR, LPIPS and a widely-used non-reference metric, FID [14].

#### Implementation Details
According to the size of each semantic dataset, we empirically set the codebook bases sizes to be $\{512, 256, 512, 256, 256\} \times 256$ for Architectures, Indoor Objects, Natural Scenes, Street Views, and Portraits. For all stages, we represent the input image as a $32 \times 32$ code sequence. We train each stage for 350k iterations with an Adam optimizer and a batch size of 32. The learning rates for the generator and discriminator are fixed as 1e-4 and 4e-4 separately. Our method is implemented with PyTorch and trained with 4 NVIDIA Tesla V100 GPUs.

#### 4.1. Expressiveness of AdaCode

The key design in our work is to leverage the class-specific basis codebooks to construct an adaptive codebook, which supports more expressiveness even with a smaller
codebook size. To verify our method’s superiority, we evaluate the reconstruction performances using three codebook and model settings: (1) VQGAN [7] with its single general codebook. (2) VQGAN with a merged codebook concatenating all the basis codebooks, referred to as VQGAN-aux. (3) AdaCode with our adaptive codebook. The three approaches are trained with the same dataset (see Section 4) to guarantee a fair comparison.

Table 1: Quantitative comparison of reconstruction performance. PSNR/SSIM↑: the higher, the better.

<table>
<thead>
<tr>
<th>Method</th>
<th>Overall Codebook Size</th>
<th>Performance PSNR</th>
<th>SSIM</th>
</tr>
</thead>
<tbody>
<tr>
<td>VQGAN [7]</td>
<td>1024 × 512</td>
<td>21.3557</td>
<td>0.5664</td>
</tr>
<tr>
<td>VQGAN-aux</td>
<td>1792 × 256</td>
<td>21.9219</td>
<td>0.6030</td>
</tr>
<tr>
<td>AdaCode (Ours)</td>
<td>1792 × 256</td>
<td>25.7629</td>
<td>0.7705</td>
</tr>
</tbody>
</table>

As shown in Table 1, our AdaCode obtains overwhelming reconstruction results with a comparable or smaller codebook size. The gap between (1) and (2) certifies that training class-specific codebooks helps the codes to capture more image textures, while the great improvement between (2) and (3) justifies the expressiveness facilitated by our adaptive codebook design. Fig. 6 shows multiple scenarios, including plants, buildings, streets, portraits, and text which has distinct patterns but does not have a corresponding codebook in our experiments. AdaCode achieves exceedingly excellent results in all semantic cases, producing realistic and fidelitous reconstruction results.

4.2. Benchmarking Image Restoration Results

Super-Resolution We compare AdaCode with state-of-the-art models for Image Super-Resolution, including KX-Net [7], Real-ESRGAN [42], and FeMaSR [3]. Specifically, KX-Net iteratively learns the degradation kernels from the LR images; Real-ESRGAN learns super-resolution using pure synthetic data with high-order degradation model; FeMaSR utilizes a single perceptually rich codebook to restore the images. We use the original codes and weights from each method’s official public repository to conduct comparisons, as shown in Table 2 and Fig. 7.

Image Inpainting We compare AdaCode with state-of-the-art inpainting methods GPEN [52] and MAT [19]. To
conduct a fair comparison, we retrain MAT on our training dataset as discussed in Section 4. Moreover, we train FeMaSR [3] for this task to demonstrate the effectiveness of our adaptive codebooks over the single codebook in FeMaSR. As shown in Table 3, AdaCode achieves state-of-the-art performance on various metrics. Qualitative comparisons in Fig. 8 also illustrate that AdaCode consistently produce high-quality inpainting results across a wide range of scenes with a single model.

### 4.3. Ablation Study

We investigate AdaCode’s expressiveness given a various number of basis codebooks. We fix the five basis codebooks trained in Stage I and train Stage II with various combinations of basis codebooks. We adopt PSNR and SSIM to evaluate the expressiveness on the reconstruction task. Fig. 9 empirically shows that the adaptive codebook benefits from the bases in a large extent. Meanwhile, it also indicates that our basis codebooks are “non-multicollinear” even if the semantic sub-datasets have overlapping patches.

### 5. Conclusion and Limitation

In this work, we propose AdaCode, a novel approach for class-agnostic image reconstruction and restoration. In par-

![Figure 7: Qualitative comparisons on super-resolution task with \( \times 2 \) and \( \times 4 \) upscale factor. AdaCode restores LR with realistic and faithful details, while the competitive work either fails to deblur the LR, i.e., KX-Net [9], or generates artifacts or over-smooth areas, i.e., Real-ESRGAN [42] and FeMaSR [3]. See Appendix for more results. (Zoom in for best view)](image)


![x2](image)

![x4](image)

Table 2: Quantitative comparison with state-of-the-art SISR methods. PSNR/SSIM↑: the higher, the better; LPIPS↓: the lower, the better. The best and second best performance are marked in red and blue. (Zoom in for best view)

<table>
<thead>
<tr>
<th>Method</th>
<th>Scale</th>
<th>Set5 PSNR</th>
<th>Set5 SSIM</th>
<th>Set5 LPIPS</th>
<th>Set14 PSNR</th>
<th>Set14 SSIM</th>
<th>Set14 LPIPS</th>
<th>BSDS100 PSNR</th>
<th>BSDS100 SSIM</th>
<th>BSDS100 LPIPS</th>
<th>Urban100 PSNR</th>
<th>Urban100 SSIM</th>
<th>Urban100 LPIPS</th>
<th>Manga109 PSNR</th>
<th>Manga109 SSIM</th>
<th>Manga109 LPIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeMaSR</td>
<td>x2</td>
<td>27.513</td>
<td>0.8250</td>
<td>0.1054</td>
<td>25.404</td>
<td>0.7565</td>
<td>0.1292</td>
<td>22.713</td>
<td>0.7573</td>
<td>0.1102</td>
<td>21.584</td>
<td>0.6938</td>
<td>0.3056</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KX-Net</td>
<td>x2</td>
<td>27.837</td>
<td>0.8884</td>
<td>0.0859</td>
<td>25.929</td>
<td>0.8291</td>
<td>0.1301</td>
<td>22.675</td>
<td>0.8016</td>
<td>0.0972</td>
<td>19.561</td>
<td>0.5049</td>
<td>0.5312</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real-ESRGAN</td>
<td>x2</td>
<td>30.032</td>
<td>0.8930</td>
<td>0.1051</td>
<td>27.019</td>
<td>0.8379</td>
<td>0.1382</td>
<td>26.670</td>
<td>0.7916</td>
<td>0.1526</td>
<td>20.815</td>
<td>0.6817</td>
<td>0.3056</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>x2</td>
<td>29.213</td>
<td>0.8444</td>
<td>0.0844</td>
<td>26.249</td>
<td>0.7702</td>
<td>0.1179</td>
<td>26.038</td>
<td>0.7529</td>
<td>0.1374</td>
<td>22.023</td>
<td>0.7029</td>
<td>0.2150</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FeMaSR</td>
<td>x4</td>
<td>24.039</td>
<td>0.7452</td>
<td>0.1500</td>
<td>22.724</td>
<td>0.6354</td>
<td>0.2045</td>
<td>21.957</td>
<td>0.5819</td>
<td>0.2517</td>
<td>20.509</td>
<td>0.6426</td>
<td>0.1983</td>
<td>18.131</td>
<td>0.5757</td>
<td>0.3449</td>
</tr>
<tr>
<td>KX-Net</td>
<td>x4</td>
<td>21.922</td>
<td>0.7050</td>
<td>0.2080</td>
<td>21.182</td>
<td>0.6126</td>
<td>0.3044</td>
<td>21.708</td>
<td>0.5768</td>
<td>0.3972</td>
<td>18.603</td>
<td>0.5576</td>
<td>0.2704</td>
<td>18.181</td>
<td>0.5323</td>
<td>0.6909</td>
</tr>
<tr>
<td>Real-ESRGAN</td>
<td>x4</td>
<td>25.263</td>
<td>0.7665</td>
<td>0.1710</td>
<td>24.100</td>
<td>0.7004</td>
<td>0.2338</td>
<td>23.776</td>
<td>0.6261</td>
<td>0.2819</td>
<td>21.351</td>
<td>0.6625</td>
<td>0.2140</td>
<td>18.222</td>
<td>0.5917</td>
<td>0.4091</td>
</tr>
<tr>
<td>Ours</td>
<td>x4</td>
<td>25.868</td>
<td>0.7731</td>
<td>0.1505</td>
<td>24.158</td>
<td>0.6662</td>
<td>0.2031</td>
<td>23.129</td>
<td>0.6041</td>
<td>0.2485</td>
<td>21.446</td>
<td>0.6568</td>
<td>0.2007</td>
<td>18.145</td>
<td>0.5664</td>
<td>0.3425</td>
</tr>
</tbody>
</table>

Table 3: Quantitative comparison with state-of-the-art inpainting methods. PSNR↑: the higher, the better; LPIPS/FID↓: the lower, the better. The best and second best performance are marked in red and blue. (Zoom in for best view)

<table>
<thead>
<tr>
<th>Method</th>
<th>DIV2K PSNR</th>
<th>DIV2K LPIPS</th>
<th>DIV8K PSNR</th>
<th>DIV8K LPIPS</th>
<th>All LPIPS</th>
<th>All FID</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPEN</td>
<td>29.129</td>
<td>0.0933</td>
<td>31.191</td>
<td>0.0703</td>
<td>3.0924</td>
<td></td>
</tr>
<tr>
<td>FeMaSR</td>
<td>29.790</td>
<td>0.0581</td>
<td>32.233</td>
<td>0.0416</td>
<td>1.6741</td>
<td></td>
</tr>
<tr>
<td>MAT</td>
<td>30.124</td>
<td>0.0676</td>
<td>32.335</td>
<td>0.0406</td>
<td>1.2385</td>
<td></td>
</tr>
<tr>
<td>Ours</td>
<td>30.1151</td>
<td>0.0516</td>
<td>32.701</td>
<td>0.0372</td>
<td>1.1657</td>
<td></td>
</tr>
</tbody>
</table>
ticular, we train a set of class-specific basis codebooks and learn a weight map to construct an image-adaptive codebook for better image representation. Unlike previous methods that use a general codebook to represent images, our image-adaptive codebook is more flexible and suited for natural images. Extensive comparisons on image reconstruction, super-resolution, and image inpainting tasks validate our method’s superiority.

Our work is a first step towards class-agnostic generative prior for arbitrary images. It has several limitations we plan to explore in future work. First, it is yet unclear how many basis codebooks and how many code entries in each codebook we need. Stage I is trained separately from Stage II&III, which may be suboptimal. Second, we do not yet incorporate high-level explicit semantic information such as semantic segmentation into the framework, which may be also useful for general image restoration tasks. Finally, it would be interesting to extend AdaCode for high-dimensional visual appearance, such as videos and multispectral images.

Figure 8: **Qualitative comparisons on image inpainting task.** The first two rows demonstrate the inpainting results of face images, while the last two rows show the recovered results of place images. See Appendix for more results.

Figure 9: **Ablation Study.** PSNR and SSIM score of reconstructed results with varying number of basis codebooks. See supplementary for more result images.
References


[26] Qiankun Liu, Zhentao Tan, Dongdong Chen, Qi Chu, Xiyang Dai, Yinpeng Chen, Mengchen Liu, Lu Yuan, and Nenghai


Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-Quantized Image Modeling with Improved VQGAN. *Proceedings of International Conference on Learning Representations (ICLR)*, 2022.


