Asymmetric Feature Fusion for Image Retrieval

Hui Wu1  Min Wang2*  Wengang Zhou1,2*  Zhenbo Lu2  Houqiang Li1,2
1CAS Key Laboratory of Technology in GIPAS, University of Science and Technology of China
2Institute of Artificial Intelligence, Hefei Comprehensive National Science Center
wh241300@mail.ustc.edu.cn, {wangmin,luzhenbo}@ial.ustc.edu.cn, {zhwg,lihq}@ustc.edu.cn

1. More Details about Experiment Setup

Testing details. For all testing datasets, images are resized with the larger dimension equal to 1,024 pixels, which preserve the original aspect ratio. Besides, image feature is extracted at three scales, i.e., \( \{1/\sqrt{2}, 1, \sqrt{2}\} \). \( L_2 \) normalization is performed for each scale independently and features of three scales are mean averaged, followed by another \( L_2 \) normalization to form the final feature. Under the asymmetric retrieval setting, queries are embedded with the lightweight query model \( \phi_q \), while the gallery images are embedded by various large models, which are further aggregated into compact embedding via the proposed mixer \( \phi_{mix} \) for efficient retrieval.

Gallery features. Tab. 1 provides details about the gallery features adopted in this work. In real-world applications, retrieval latency, feature extraction latency and storage overhead are three important considerations.

(1) Retrieval latency. Although inverted index greatly speeds up image retrieval based on local features, the online retrieval latency is still larger compared to compact global features, e.g., 0.995s for HOW [8] vs. 0.345s for DELG [1], when searching in a gallery of 1 million images. Our approach aggregates local features into compact descriptors at the gallery side, which reduces online retrieval latency while improving asymmetric retrieval accuracy.

(2) Feature extraction latency. To achieve scale invariance, existing deep local features are extracted at multiple image scales. An image is scaled to different sizes and passed through feature extractor multiple times, which notably increases the computational complexity, e.g., it takes about 258.1ms to extract local features of an image for HOW [8]. In contrast, compact global feature is derived from the feature maps of the deep representation model by spatial pooling, which is suitable for resource-constrained scenarios. For example, it only takes about 16.5ms to extract a global feature when MobileNetV2 is adopted as feature extractor.

(3) Storage overhead. Typically, there are thousands of local features in a single image, which greatly increases the storage overhead of the gallery set. For example, for R1M [6], it takes about 480G to store all the local features adopted in this work, absolute (ABS) and relative (‰) to the gallery model. All the models are modified slightly to adapt to image retrieval. The FLOPs are calculated with a input image of \( 362 \times 362 \).
extracted by HOW [8]. Even being binarized, the inverted index still takes up 14.2G of memory. While for the global feature DELG [1], it only needs 7.7G to store all the gallery features. In our method, multiple local and global features are aggregated into compact embedding on the gallery side. It exploits the complementarity of different features to enhance the discriminativeness of gallery features while ensuring a small storage overhead, e.g., 7.7G for R1M.

2. Additional Ablation and Analysis

**Impact of $C$ in the Eq. (10) of main paper.** The proposed mixer dynamically extracts helpful features from various gallery features via a learnable fusion token. In this section, we explore the impact of different iteration numbers on the retrieval accuracy. As shown in Fig. 1, retrieval accuracy increases gradually as the number of iterations increases under both asymmetric and symmetric settings. The results show that helpful features are progressively aggregated to form discriminative image representations.

**Inference computational overhead.** In previous experiments, we follow the common setup to scale a test image to different sizes and forward them through feature extractor multiple times to extract multi-scale feature. However, multi-scale feature extraction notably increases the computational overhead for the query side. In this section, we investigate the relationship between retrieval performance and computational overhead. MobileNetV2 is deployed as the query model and single-scale feature extraction is performed. We scale test images to different sizes, which correspond to different computational overhead (FLOPs), to ex-
tract single-scale query features. As shown in Fig. 2, both asymmetric and symmetric retrieval accuracy increases and then saturates as inference computational complexity increases. In some real-world applications, we need to scale an image to an appropriate size for the trade-off between retrieval accuracy and efficiency.

**Different lightweight query models.** In this section, various lightweight models, with different computational complexity and model size, are deployed as query models \( \phi_q \). As shown in Tab. 3, when various lightweight models are deployed as query model, our approach consistently advances the accuracy of asymmetric retrieval without introducing any additional overhead to the query side.

**Output dimension.** Since our method jointly trains the mixer and query model, it is easy to adjust the dimension of image feature. In the real-world retrieval system, there are billions of images and large feature dimension leads to huge storage overhead. In this section, we explore the impact of different output dimension on retrieval accuracy. As shown in Fig. 3, both asymmetric and symmetric retrieval accuracy increases gradually and then saturates as the dimension increases. In practical scenarios, we should choose appropriate feature dimension to achieve the trade-off between storage footprint and retrieval accuracy.

**More details about generalizability analysis.** In the main paper, our asymmetric feature fusion is combined with various existing asymmetric retrieval methods. Here, we provide more implementation details, when it is combined with CSD [10]. For an image, CSD aims to constrain the query model to maintain its neighborhood structure in the embedding space of the gallery model. It assumes that a well-trained gallery model is adopted to embed the images of a query image to an appropriate size for the trade-off between retrieval accuracy and efficiency.

As shown in Tab. 3, when various lightweight models are deployed as query model, our approach consistently advances the accuracy of asymmetric retrieval without introducing any additional overhead to the query side.

In this section, variant asymmetric retrieval methods are deployed as query model, our approach consistently advances the accuracy of asymmetric retrieval without introducing any additional overhead to the query side.

**Output dimension.** Since our method jointly trains the mixer and query model, it is easy to adjust the dimension of image feature. In the real-world retrieval system, there are billions of images and large feature dimension leads to huge storage overhead. In this section, we explore the impact of different output dimension on retrieval accuracy. As shown in Fig. 3, both asymmetric and symmetric retrieval accuracy increases gradually and then saturates as the dimension increases. In practical scenarios, we should choose appropriate feature dimension to achieve the trade-off between storage footprint and retrieval accuracy.

**More details about generalizability analysis.** In the main paper, our asymmetric feature fusion is combined with various existing asymmetric retrieval methods. Here, we provide more implementation details, when it is combined with CSD [10]. For an image, CSD aims to constrain the query model to maintain its neighborhood structure in the embedding space of the gallery model. It assumes that a well-trained gallery model is adopted to embed the images of a training gallery \( G^T \) into features \( G \in \mathbb{R}^{N \times d} \). These features are then adopted to mine neighbors of each training image during the learning of query model.

However, in our approach, the gallery features keep evolving with training. To approximate the fixed gallery features \( G \) in CSD, we maintain a memory to store features from the immediate preceding mini-batches, which is denoted as \( \tilde{G} \in \mathbb{R}^{M \times d} \). The introduction of the memory makes the size of training gallery much larger than that of mini-batch, which provides sufficient neighbors for each training image. Similar to MOCO [3], the features in the memory are progressively replaced, following the first-in-first-out principle. To improve the consistency of features in the memory, we also follow MOCO to maintain a momentum-updated version of mixer \( \phi_{mix}^\text{mix} \) to aggregate multiple gallery features into more smooth embedding \( \tilde{g}^\text{mix} \), which is stored in the memory. During query model training, for each training image, we mine its neighbors just in the memory with the embedding generated by \( \tilde{g}^\text{mix} \). The final objective function consists of two losses. One is ArcFace loss [2] (Eq. (11) in the main paper) for training mixer, the other is contextual similarity consistency loss defined in CSD, with the modifications mentioned above.

**References**


