Efficient Image Retrieval via Decoupling Diffusion into Online and Offline Processing

Fan Yang,[1,2] **Ryota Hinami,**[1,2] **Yusuke Matsui,**[1] **Steven Ly,**[2,3] **Shin'ichi Satoh**[1,2]
[1]The University of Tokyo, Japan
[2]National Institute of Informatics, Japan
[3]University of Southern California, USA

Abstract

Diffusion is commonly used as a ranking or re-ranking method in retrieval tasks to achieve higher retrieval performance, and has attracted lots of attention in recent years. A downside to diffusion is that it performs slowly in comparison to the naive k-NN search, which causes a non-trivial online computational cost on large datasets. To overcome this weakness, we propose a novel diffusion technique in this paper. In our work, instead of applying diffusion to the query, we pre-compute the diffusion results of each element in the database, making the online search a simple linear combination on top of the k-NN search process. Our proposed method becomes 10$\sim$ times faster in terms of online search speed. Moreover, we propose to use late truncation instead of early truncation in previous works to achieve better retrieval performance.

Introduction

The success of deep neural networks on feature representation has led it to become a standard technique in image retrieval. Models pre-trained on popular datasets such as ImageNet (Deng et al. 2009), Landmarks (Babenko et al. 2014) etc. can be used to extract features of images. Particularly, convolutional layers have been proved to be most beneficial at retrieving images (Babenko et al. 2014; Radenović, Tolias, and Chum 2016; Gordo et al. 2016; Razavian et al. 2016). Nearest neighbor search is then used on the feature vectors to find the most similar images to a query.

Datasets are usually scraped from the internet, resulting in images with the same object/landmark shown in a variety of angles, lighting, and other conditions. The diversity often results in a manifolds in the feature space that are not conducive to ranking using a distance-based metric. Unlike the rigid distance metric used in k-NN search, diffusion (Zhou et al. 2004b; 2004a; Donoser and Bischof 2013; Grady 2006) exploits the intrinsic manifold structure of data based on a neighborhood graph. Such a graph consists of nodes and edges, where each node represent a feature vector from the database, with the edges connect each node to its neighbors with corresponding weights proportional to the pairwise affinities between nodes. Using this graph, diffusion performs a restartable random walk given a query as the initial state. The final state of random walk can be viewed as ranking scores showing the similarities of each image in database to the query. To obtain the convergence of the final state, there are two main approaches: running iterative random walk or computing the convergence state by the closed-form theorem proposed in (Zhou et al. 2004b). Diffusion has demonstrated its potential in improving retrieval performance (Donoser and Bischof 2013; Iscen et al. 2017; Radenovic et al. 2018), and is also utilized in other fields such as unsupervised learning (Iscen et al. 2018b). Recently, other works have attempted to improve the efficiency of diffusion (Iscen et al. 2017), but their speed up is still not sufficient enough to handle the amount of queries found in large-scale image retrieval datasets.

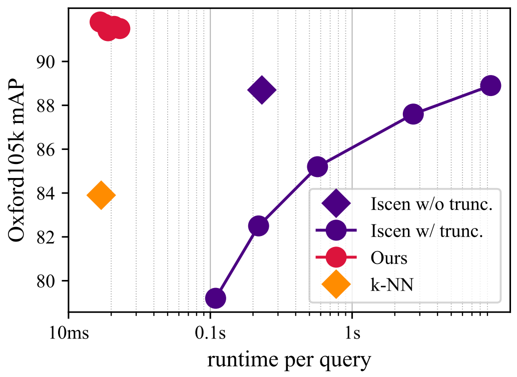


Figure 1: The efficiency vs. performance comparison between k-NN, Iscen's method (Iscen et al. 2017), and our proposed method on Oxford105k dataset with global image features. Our proposed method achieves better retrieval performance than diffusion by Iscen et al. with almost the same search speed ($\sim$20ms per query) as k-NN search. We show the results of varying truncation sizes L.

We notice that the main bottlenecks in speed of online diffusion processes come from the random walk and preparation steps. Inspired by the closed-form solution of diffusion, we find that the diffusion for each query can be converted to a linear combination of the pre-computed diffusion results of all database elements. Following this observation, our proposed method completely removes the random walk from the online stage. As a result, our work is able to improve the efficiency of the diffusion process by a factor of ten in a large-scale image retrieval setting.

In addition, the previous versions of diffusion utilize an early truncation that happens before the affinity matrix normalization process. In our proposed process, we propose to perform a late truncation after normalization, that results in significantly better performance. Fig. 1 summarizes the efficiency and performance of the aforementioned methods. The source code to replicate our experiments is available at https://github.com/fyang93/diffusion.

Related Works

Although originally developed for ranking on manifolds (Page et al. 1999; Zhou et al. 2004b; Donoser and Bischof 2013), diffusion was soon applied to classification (Zhou et al. 2004a), and image segmentation (Grady 2006). In the field of image retrieval, it is most frequently used as a re-ranking method (Iscen et al. 2017; Radenovic et al. 2018).

Query expansion, a common technique in image retrieval, can improve retrieval performance during query time. Average query expansion (AQE) (Chum et al. 2007; Iscen et al. 2017), a popular type of query expansion because of its simplicity, averages the features of the query's nearest neighbors to form a new query to run search again. When AQE is applied iteratively, the recomputation of the query is akin to traveling along the manifolds of the feature space. Although this traversal is similar to diffusion, AQE only utilizes the relationships between query and database images, but not between each of the database images with each other. With prior knowledge of the relationships between all of the database images, diffusion is thus better able to exploit the manifolds in the feature space than query expansion can.

In previous works of diffusion, the query is provided as a part of the database. However, in a real-world setting, queries are unavailable until they are issued by users. To tackle this issue without introducing any computational overhead, (Iscen et al. 2017) uses the short list of k-NN search results to form a sparse initial state vector, instead of using a one-hot vector as the initial state. As a consequence, queries are not included in the neighborhood graph. The downside to this is that the graph needs to be stored and loaded during the search stage for random walk, which is both memory and computationally inefficient. Since the previous methods were evaluated on the Oxford (Philbin et al. 2007) and Paris (Philbin et al. 2008) datasets, smaller datasets only containing 55 queries, the inefficiency of those methods did not have much impact on the total computation time. When these methods are used on large-scale datasets with many queries, the inefficiency during online search becomes magnified and intractable.

To tackle this inefficiency, past efforts have been made to scale diffusion up to handle larger datasets. (Dong, Moses, and Li 2011) proposed to accelerate the construction of the affinity matrix denoting the graph. Iscen et al. reported that Dong's method is orders of magnitude faster than exhaustive search with only limited decreases in performance (Iscen et al. 2017). Another approach to improve efficency is using approximate nearest neighbor search (ANN). Compared to constructing the graph by exhaustive k-NN search, ANN search is faster and provides comparable accuracy (Jegou, Douze, and Schmid 2011; Ge et al. 2014). Most recently, (Iscen et al. 2018a) approximated the affinity matrix by using a low-rank spectral decomposition to reduce the online computational cost. However, this method did not result in much improvement in terms of retrieval performance.

Preliminaries of Diffusion

There are two main approaches to conducting diffusion: through iterative updates or solving the closed form directly. Both Zhou et al. and Donoser et al. describe diffusion as a mechanism for spreading the query similarities over the manifolds (Zhou et al. 2004b; Donoser and Bischof 2013), while Iscen et al. utilizes the closed form theorem in (Zhou et al. 2004b) and proposed an efficient solution (Iscen et al. 2017). We mainly follow the steps from (Zhou et al. 2004a) and (Iscen et al. 2017) below.

Problem setup. For image retrieval tasks, we define a database as $\chi = \{\mathbf{x}_1, \dots, \mathbf{x}_n\} \subset \mathbb{R}^d$, where each $\mathbf{x}_i$ is a feature vector. Images can be represented by a global feature that corresponds to the entire image, or multiple regional features corresponding to different regions of the image. In the following equations, $\mathbf{x}_i$ can stand for either of these representations.

For most public datasets in the retrieval field, query and database images are both available. In our following example, queries are unseen to us until provided by users. We denote the query as $\mathcal{Q} = \{\mathbf{q}_1, \dots, \mathbf{q}_m\} \subset \mathbb{R}^d$, where $m = 1$ when the query is described by a global feature and $m > 1$ when it contains regional features.

Graph construction. For simplicity, we consider an example where we handle only one query image $\mathcal{Q}$ and include it into the database. The entire set is defined as $\bar{\chi} = \{\mathbf{q}_1, \dots, \mathbf{q}_m, \mathbf{x}_1, \dots, \mathbf{x}_n\}$, and we denote i-th element in $\bar{\chi}$ as $\bar{\chi}_i$. In addition, a local constraint is adopted so that the graph only contains similarities between pairs of elements that are nearest neighbors to each other according to (Iscen et al. 2017). The affinity matrix is defined as $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{(n+m)\times(n+m)}$, where each element is obtained by

$$a_{ij} = \begin{cases} s(\bar{\chi}_i, \bar{\chi}_j) & i \neq j, \bar{\chi}_i \in \mathrm{NN}_k(\bar{\chi}_j), \bar{\chi}_j \in \mathrm{NN}_k(\bar{\chi}_i) \\ 0 & \text{otherwise} \end{cases}, \quad (1)$$

$\forall i, j \in \{1, \dots, n+m\}$, denoting $\mathrm{NN}_k(\mathbf{x})$ the k-NNs of $\mathbf{x}$. Since the similarity metric s is usually symmetric and positive, $\mathbf{A}$ is a symmetric matrix. Eq. (1) allows $\mathbf{A}$ to be sparse, providing memory and computational efficiency.

The degree matrix $\mathbf{D}$ is a diagonal matrix and each diagonal element is the corresponding row-wise sum of $\mathbf{A}$, *i.e.* the element d_{ii} in $\mathbf{D}$ is defined as $\sum_{j=1}^{n+m} a_{ij}$. It's later used to symmetrically normalize $\mathbf{A}$ into the stochastic matrix $\mathbf{S}$:

$$\mathbf{S} = \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}. \quad (2)$$

$\mathbf{S}$ is a variant of the typical transition matrix $\mathbf{D}^{-1}\mathbf{A}$, and both have the same eigenvalues and eigenvectors (Donoser and Bischof 2013).

Random walk. After the graph construction, the random walk is performed until it reaches a convergence state, resulting in final ranking scores for each of the images in the gallery. For the t-th step of random walk, the state is recorded in a vector $\mathbf{f}^t = [\mathbf{f}_q^{t\top}, \mathbf{f}_d^{t\top}]^\top \in \mathbb{R}^{n+m}$, where $\mathbf{f}_q^t \in \mathbb{R}^m, \mathbf{f}_d^t \in \mathbb{R}^n$. We set the initial state to be a m-hot vector where $\mathbf{f}_q^0 = \mathbf{1}_m, \mathbf{f}_d^0 = \mathbf{0}_n$. The random walk iterates the following step:

$$\mathbf{f}^{t+1} = \alpha \mathbf{S} \mathbf{f}^t + (1-\alpha)\mathbf{f}^0, \quad \alpha \in (0,1). \tag{3}$$

Essentially, there is a probability α to randomly walk from the current state $\mathbf{f}^t$ or $1-\alpha$ to restart from the initial state $\mathbf{f}^0$. Given the fact that $\alpha \in (0,1)$ and the abstract eigenvalues of $\mathbf{S}$ is no larger than 1 according to the Perron-Frobenius theorem, this iteration converges to a closed-form solution (Zhou et al. 2004b):

$$\mathbf{f}^* = (1-\alpha)(\mathbf{I} - \alpha\mathbf{S})^{-1}\mathbf{f}^0. \tag{4}$$

After convergence, the values in $\mathbf{f}^*$ contain the similarities of each database element to the query, which will be used as ranking scores for re-ranking.

Decomposition. The above steps incorporate the query into the graph during the diffusion process. Grady proposed to decompose queries from the above operations (Grady 2006), and his technique was recently followed by (Iscen et al. 2017).

Note, the closed-form solution $\mathbf{f}^* \in \mathbb{R}^{n+m}$ contains the ranking scores on both the query and database elements, but for the task of image retrieval, we only care about the ranking scores for database elements. This leads to the decomposition of the query and database ranking scores, so that the matrix $\mathbf{S}$ is split into 4 blocks

$$\mathbf{S} = \begin{bmatrix} \mathbf{S}_{qq} & \mathbf{S}_{qd} \\ \mathbf{S}_{dq} & \mathbf{S}_{dd} \end{bmatrix}, \tag{5}$$

where $\mathbf{S}_{qq} \in \mathbb{R}^{m\times m}, \mathbf{S}_{qd} \in \mathbb{R}^{m\times n}, \mathbf{S}_{dq} \in \mathbb{R}^{n\times m}$, and $\mathbf{S}_{dd} \in \mathbb{R}^{n\times n}$. The decomposed solution then becomes

$$\mathbf{f}_d^* = (1-\alpha)(\mathbf{I} - \alpha\mathbf{S}_{dd})^{-1}\mathbf{S}_{dq}\mathbf{f}_q^0 \in \mathbb{R}^n, \tag{6}$$

where $\mathbf{S}_{dd}$ can be viewed as the transition matrix for random walk on the database side, and $\mathbf{S}_{dq} = \mathbf{S}_{qd}^\top$ consists of normalized similarities between the query and its nearest neighbors. Subsequently, we can then obtain the cleaner form:

$$\mathbf{f}_d^* \propto \mathcal{L}_\alpha^{-1}\mathbf{y}, \tag{7}$$

where $\mathcal{L}_\alpha = \mathbf{I} - \alpha\mathbf{S}_{dd} \in \mathbb{R}^{n\times n}, \mathbf{y} = \mathbf{S}_{dq}\mathbf{f}_q^0 \in \mathbb{R}^n$. and we can ignore the constant $1-\alpha$ since scores are used for ranking.

Truncation. An optional truncation step is used on large datasets, where the dataset is truncated to a smaller subset before normalization and random walk. Iscen et al. conduct truncation with only global features representing entire images (Iscen et al. 2017). Given the global feature vector $\mathbf{q}$ of a new query ($m = 1, \mathbf{q} = \mathbf{q}_1$), the indexes $\mathcal{I} = \mathrm{NN}_L^{\mathrm{ID}}(\mathbf{q})$ of features corresponding to the top ranked images is retrieved by L-NN search, where L is a limiting constant that defines the maximum size of the subgraph (truncated graph). The affinity matrix denoting the subgraph is defined as $\hat{\mathbf{A}} \in \mathbb{R}^{L\times L}$, and each element $\hat{a}_{ij}$ in $\hat{\mathbf{A}}$ satisfies

$$\hat{a}_{ij} = \begin{cases} s(\chi_{\mathcal{I}_i}, \chi_{\mathcal{I}_j}) & \mathcal{I}_i \neq \mathcal{I}_j, \chi_{\mathcal{I}_i} \in \mathrm{NN}_k(\chi_{\mathcal{I}_j}), \chi_{\mathcal{I}_j} \in \mathrm{NN}_k(\chi_{\mathcal{I}_i}) \\ 0 & \text{otherwise} \end{cases}, \tag{8}$$

$\forall i,j \in \{1,\dots,L\}$, where $\mathcal{I}_i$ is the i-th index in the set $\mathcal{I}$. After truncation, $\hat{\mathbf{A}}$ is normalized into a stochastic matrix $\hat{\mathbf{S}}$ and then random walk is subsequently performed on $\hat{\mathbf{S}}$. We refer to the process of normalizing the truncated graph as subgraph normalization throughout the rest of the paper.

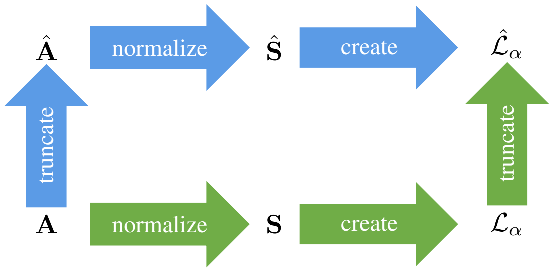


Figure 2: Comparison between our implementation (green) and previous works' (blue). Our method truncates late while previous methods truncate early during diffusion.

Proposed Method

We propose a remarkably fast diffusion achieving state-of-the-art retrieval performance. Iscen et al. reported that $\mathcal{L}_\alpha^{-1}$ is not sparse like $\mathcal{L}_\alpha$ making it less efficient to compute than using $\mathcal{L}_\alpha$ to solve $\mathcal{L}_\alpha \mathbf{f}_d^* \propto \mathbf{y}$ online (Iscen et al. 2017). Our method makes it possible to pre-compute and maintain a sparsified $\mathcal{L}_\alpha^{-1}$ offline to achieve better efficiency. Given a new query, its diffusion result can be obtained by linear combination according to Eq. (7). As a result, we achieve a substantial improvement in the online search speed. Moreover, we argue that the subgraph normalization that takes place in (Iscen et al. 2017) has negative effects on the retrieval performance. After the offline computation to obtain $\mathcal{L}_\alpha$ from the entire matrix $\mathbf{A}$, we apply slicing to $\mathcal{L}_\alpha$ to fetch the values on the corresponding rows and columns limited in the truncation subset.

In the following sections, we compare the time complexity between Iscen's method and our method to analyze the efficiency gains of our method.

Complexity analysis of online diffusion

In prior works, the entire process of diffusion is performed during runtime when queries are processed. A combination of global and regional features are also used, but for simplicity, we choose to analyze Iscen's online diffusion with only global features in this section.

Given the global feature $\mathbf{q}$ of a new query, the pipeline of online diffusion can be broken down to the following steps:

1. *Truncation*: the nearest neighbors $\mathrm{NN}_L(\mathbf{q}) \subset \chi$ of $\mathbf{q}$ is obtained by k-NN search for truncation

2. *Graph construction*: the truncated affinity matrix $\hat{\mathbf{A}}$ denoting subgraph is constructed for the subset $\mathrm{NN}_L(\mathbf{q})$ with a reciprocity check, then *subgraph normalization* is applied to form the matrix $\hat{\mathbf{S}}$ and $\hat{\mathcal{L}}_\alpha$ are created afterward
3. *Initialization*: the vector $\mathbf{y} = \mathbf{S}_{dq}\mathbf{f}_q^0$ contains the similarities between the query and its nearest neighbors, which is subsequently truncated to fit the size of $\hat{\mathcal{L}}_\alpha$
4. *Random walk*: the convergence state is solved by Eq. (7) to obtain the result of diffusion by conjugate gradient

The above steps include k-NN search, which is responsible for most of the time needed to process those steps. Recent k-NN search strategies are sufficiently efficient (Jegou, Douze, and Schmid 2011; Malkov and Yashunin 2016; Johnson, Douze, and Jégou 2017), so we do not discuss the complexity which is beyond the scope of this work. Constructing the affinity matrix denoting the subgraph also requires k-NN search, and the subgraph normalization according to Eq. (2) costs $\mathcal{O}(Lk)$ time. Here, Lk is the number of non-zero entries in the truncated affinity matrix, and k is the parameter for the number of nearest neighbors in k-NN search. In the random walk step, the result is approximated by the early-terminated conjugate gradient (CG) (Nocedal and Wright 2006; Iscen et al. 2017). Suppose we iterate CG for τ steps, its time complexity is $\mathcal{O}(Lk\tau)$.

From the above analysis, we can see that the cost to process each query is non-trivial and the most time-consuming steps are the subgraph construction and random walk. This motivates us to solve these inefficiencies.

From online to offline

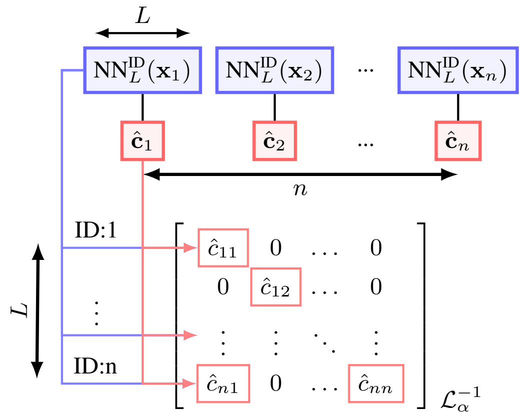
We propose a new form of diffusion that moves the *online* steps, processing the heavy computation steps during runtime, to *offline*, pre-computing those steps beforehand.

The right side of Eq. (7) can be considered as a linear combination of column vectors in $\mathcal{L}_\alpha^{-1}$ with weights in vector $\mathbf{y}$. In other words, the results of any new query is merely the linear combination of the columns of $\mathcal{L}_\alpha^{-1}$ with corresponding weights in $\mathbf{y}$. Unfortunately, the inverse of a large sparse matrix is hard to compute even though $\mathcal{L}_\alpha$ is positive-definite. Despite this difficulty, it is still possible to compute the approximate inverse by either global iteration or a column-oriented algorithm, two approaches summarized in (Saad 2003). Global iteration computes the inverse on the entirety of the matrix, whereas the column-oriented algorithm computes it one column at a time. Between the two, the column-oriented algorithm is more appealing for parallelism since it computes each column separately. It also allows us to apply truncation in computing each column to make a sparser structure. Therefore, we choose to adopt the column-oriented strategy in our proposed method.

To compute each column of $\mathcal{L}_\alpha^{-1}$, we first define a set of vectors $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$ to be the column vectors of an identity matrix $\mathbf{I} \in \mathbb{R}^{n\times n}$. Then, according to the closed-form solution in Eq. (7) we solve:

$$\mathcal{L}_\alpha \mathbf{c}_i = \mathbf{b}_i, \tag{9}$$

with conjugate gradient (CG) (Nocedal and Wright 2006), we obtain $\mathbf{c}_i$, the approximate i-th column vector in $\mathcal{L}_\alpha^{-1}$.



L
$\mathrm{NN}_L^{\mathrm{ID}}(\mathbf{x}_1)$
$\mathrm{NN}_L^{\mathrm{ID}}(\mathbf{x}_2)$
···
$\mathrm{NN}_L^{\mathrm{ID}}(\mathbf{x}_n)$
$\hat{\mathbf{c}}_1$
$\hat{\mathbf{c}}_2$
$\hat{\mathbf{c}}_n$
n
ID:1
ID:n
L
$\hat{c}_{11}$ 0 … 0
0 $\hat{c}_{12}$ … 0
$\hat{c}_{n1}$ 0 … $\hat{c}_{nn}$
$\mathcal{L}_\alpha^{-1}$

Figure 3: The data structure of sparsified matrix $\mathcal{L}_\alpha^{-1}$, where the values in i-th column compose the column vector $\hat{\mathbf{c}}_i$ and $\mathrm{NN}_L^{\mathrm{ID}}(\mathbf{x}_i)$ records the row indexes of each value in $\hat{\mathbf{c}}_i$.

Essentially, $\mathbf{c}_i$ can be viewed as the diffusion result of i-th database element $\mathbf{b}_i$. After the database-side diffusion, given a new query, the pipeline becomes

1. *Initialization*: prepare the initial state vector $\mathbf{y}$ for the query, where the indexes of non-zero entries are $\{i_1, \dots, i_h\}$, and their values are $\{v_1, \dots, v_h\}$
2. *Linear combination*: combine the corresponding columns $\{\mathbf{c}_{i_1}, \dots, \mathbf{c}_{i_h}\}$ from $\mathcal{L}_\alpha^{-1}$ with the values in step 1 to obtain the diffusion result: $\mathbf{f}_d^* \propto \sum_j v_j \mathbf{c}_{i_j}$

Note that the inverse matrix $\mathcal{L}_\alpha^{-1}$ obtained from the above procedure is a dense matrix, which is less memory efficient. We propose the sparsified version (Fig. 3) of inverse matrix $\mathcal{L}_\alpha^{-1}$ to provide better memory efficiency in the next section.

Database-side truncation

As we discussed, truncation is crucial for scaling up to large datasets. The time complexity of diffusion is deeply related to L, the size of subgraph after truncation. Thus, the process of random walk can be accelerated if the database is truncated to a smaller size. Despite increases in speed, Iscen et al. observed that truncation has a negative effect on the retrieval performance (Iscen et al. 2017).

Contrary to Iscen's findings, we find that truncation by itself is not a detrimental practice. Rather, the order in which it is applied in relation to normalization is the important factor. We find that applying normalization after truncation is the reason for the decrease in retrieval performance. The subgraph after truncation contains incomplete manifolds and the later normalization raises the probabilities to transition to the nodes on such manifolds. This causes the random walk to be more likely to visit misleading nodes.

To tackle this issue, we normalize the complete matrix $\mathbf{A}$ to build $\mathcal{L}_\alpha$, and then apply late truncation (slicing) to $\mathcal{L}_\alpha$ directly, as shown in Fig. 2. Denoting the indexes of the nearest neighbors of a query as $\mathcal{I} = \mathrm{NN}_L^{\mathrm{ID}}(\mathbf{q})$, the truncation is applied by

$$\hat{l}_{ij} = l_{\mathcal{I}_i \mathcal{I}_j}, \quad \forall i,j \in \{1,\dots,L\}, \tag{10}$$

Algorithm 1 Online search

```
1: Input $\mathcal{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_m\} \leftarrow$ a new query
2: Output $\mathbf{f}_d^* \leftarrow$ a new array of $n$ zeros
3: for $i \leftarrow 1$ to $m$ do
4:     obtain $\text{NN}_k^{\text{SIM}}(\mathbf{q}_i), \text{NN}_k^{\text{ID}}(\mathbf{q}_i)$ by $k$-NN search
5:     for $j \leftarrow 1$ to $k$ do
6:         $\texttt{col\_id} \leftarrow \text{NN}_k^{\text{ID}}(\mathbf{q}_i)[j]$
7:         $\texttt{weight} \leftarrow \text{NN}_k^{\text{SIM}}(\mathbf{q}_i)[j]$
8:         $\texttt{row\_ids} \leftarrow \text{NN}_L^{\text{ID}}(\mathbf{x}_{\texttt{col\_id}})$ from sparsified $\mathcal{L}_\alpha^{-1}$
9:         $\mathbf{f}_d^*[\texttt{row\_ids}] \leftarrow \mathbf{f}_d^*[\texttt{row\_ids}] + \texttt{weight} * \hat{\mathbf{c}}_{\texttt{col\_id}}$
10: Aggregate scores in $\mathbf{f}_d^*$ to image level if needed
```

where l_{ij} is the element in $\mathcal{L}_\alpha$ while $\hat{l}_{ij}$ stands for the element in $\hat{\mathcal{L}}_\alpha$ after truncation. From our experiments, such a truncation can provide a significant performance boost.

In previous works, diffusion cannot be performed without the query because truncation process is applied under the guide of queries with a subsequent random walk. This forces diffusion in those works to be computed online. Our proposed method differs by instead using the elements in the database themselves as queries, thus moving the entire diffusion process offline. In addition, truncation is applied to the database elements as previous works applied it on queries.

As an example, we take a database element $\mathbf{x}_i$ and conduct L-NN search in χ to get the indexes $\mathcal{J} = \text{NN}_L^{\text{ID}}(\mathbf{x}_i)$ of the short list. Since $\mathcal{L}_\alpha$ can be computed and cached beforehand, we slice $\mathcal{L}_\alpha$ with $\mathcal{J}$ to avoid subgraph normalization. The one-hot vector $\mathbf{b}_i$ is also sliced to $\hat{\mathbf{b}}_i$ with the indexes $\mathcal{J}$. Usually the indexes $\text{NN}_L^{\text{ID}}(\mathbf{x}_i)$ are sorted by the similarities in a descending order. Thus, the index i of $\mathbf{x}_i$ is always at the top of $\text{NN}_L^{\text{ID}}(\mathbf{x}_i)$ since it is always most similar to itself. As a result, the truncated one-hot inital state vector $\hat{\mathbf{b}}_i = [1, 0, 0, \ldots]^\top \in \{0, 1\}^L$ is fixed regardless of i. Therefore, we use the same initial state vector for all database-side random walk.After the truncation on $\mathcal{L}_\alpha$ and $\mathbf{b}_i$, we obtain the truncated i-th column vector in $\mathcal{L}_\alpha^{-1}$ by $\hat{\mathcal{L}}_\alpha \hat{\mathbf{c}}_i = \hat{\mathbf{b}}_i$, where $\hat{\mathbf{c}}_i \in \mathbb{R}^L$. This sparsifies the matrix $\mathcal{L}_\alpha^{-1}$. Fig. 3 shows the structure of the sparsified $\mathcal{L}_\alpha^{-1}$. It costs $\mathcal{O}(Ln)$ memory to store the pre-computed $\mathcal{L}_\alpha^{-1}$, which is more than $\mathcal{O}(L^2)$, the memory usage of Iscen's method.

To summarize, the late truncation directly applied on $\mathcal{L}_\alpha$ removes the negative effects of early truncation, and is completely pre-computed offline. In addition, the truncated initial state vector is fixed, meaning there is no extra overhead cost to build it.

Online search

Once the sparsified inverse matrix is obtained, the online search results can be calculated by using Algorithm 1. We denote $\text{NN}_k^{\text{SIM}}(\mathbf{q}_i)$ as the similarities between the query feature $\mathbf{q}_i$ and its nearest neighbors in χ. The online search thus becomes very fast because it only includes the k-NN search and linear combination. Moreover, since we only need k columns in $\mathcal{L}_\alpha^{-1}$ for each query, our approach can merely cost $\mathcal{O}(Lk)$ RAM during runtime.

Experiments

This section presents the experimental setup and investigates the computational efficiency as well as retrieval performance of our methods for image retrieval. For the efficiency evaluation, we use a single core of Intel Xeon 2.80GHz CPU.

Experimental setup

Datasets. We use the Oxford Buildings (Philbin et al. 2007) and Paris (Philbin et al. 2008) datasets in our experiments. The datasets are referred to as Oxford5k and Paris6k respectively in correspondence with the size of each dataset. Another set of 100k random images from Flicker (Philbin et al. 2008) are commonly used as distractors to enlarge the above datasets to Oxford105k and Paris106k. We measure the online computational time on the 55 queries of the datasets for Iscen's method (Iscen et al. 2017) and our proposed method. For evaluation, we adopt the standard mean average precision (mAP) as a performance measurement.

Features. We use the 512 and 1,024 dimensional global R-MAC descriptors (Tolias, Sicre, and Jégou 2015; Gordo et al. 2016) provided by Iscen et al. for fair comparison. We experiment on both global and regional features provided. For the Oxford and Paris datasets, there are 21 regional features per image on average.

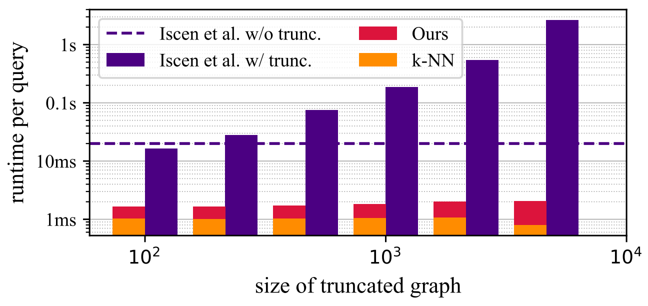
K-NN search. We conduct k-NN search by using the efficient FAISS toolkit [1], containing a CPU version and a faster GPU version (Johnson, Douze, and Jégou 2017), which allows us to deal with the larger Oxford105k and Paris106k datasets, especially for offline pre-computation.

Implementation details. We use the same graph construction parameters as in the previous work (Iscen et al. 2017). In particular, the parameter α to build $\mathcal{L}_\alpha$ is set to 0.99. For global features, 50 nearest neighbors of each database element are used for graph construction, and the initial state vector contains the similarities between the query and its 10 nearest neighbors. While for regional features, the corresponding numbers of nearest neighbors are set to 200, 200 respectively. Through our experiments, deviating from these parameters consistently resulted in worse performance.

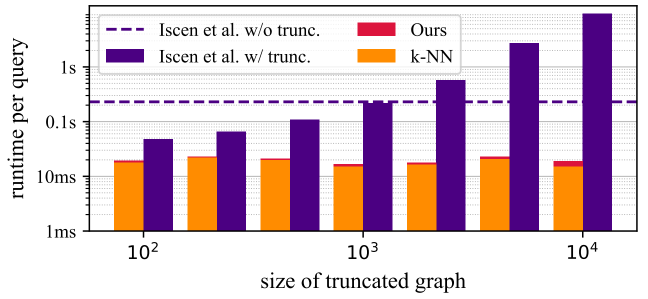
Runtime computational efficiency

We evaluate the computational efficiency on each query for k-NN search, Iscen's method and our proposed method. Each method is run 10 times and the average computational time is used for comparison. The results on Oxford5k and Oxford105k datasets are shown in Fig. 4. Since most of the computation in our proposed method is already done offline on database side, we observe that our method can be remarkably fast during online search, close to the speed of k-NN search. Fig. 4 shows that the average search time per query of our method with global features are ~2ms and ~10ms on Oxford5k and Oxford105k datasets respectively, and its extra computation over k-NN search is negligible. In contrast, Iscen's method is rather time-consuming since it has a lot of runtime processes. The search time per query for online

[1] https://github.com/facebookresearch/faiss



(a) Oxford5k



(b) Oxford105k

Figure 4: Online search time per query measured for k-NN search, Iscen's method and our proposed method respectively, using global features.

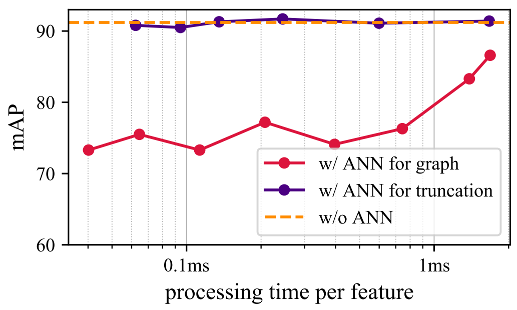


Figure 5: Speed/accuracy trade-off by using approximate nearest neighbor (ANN) search in truncation and graph construction.

diffusion without truncation is slower: ∼20ms on Oxford5k and ∼0.2s on Oxford105k. When truncation is applied during the offline diffusion process, the overhead to construct the graph during runtime causes it to be slow. We also observed a decrease in efficiency as the size of truncated graph grows as shown in Fig. 4.

Pre-computational efficiency

Now that we have shown the online search of our proposed approach is very efficient, we investigate the offline computational cost. The offline process mainly consists of two parts: the database vs. database k-NN search for truncation and random walk processes on each of the database element. Since exhaustive k-NN search on a large scale dataset is time-consuming, we utilize approximate nearest neighbor (ANN) search. Figure 5 shows the speed/accuracy trade-off when using ANN search for either truncation or graph construction. We adopt IVFADC (Jegou, Douze, and Schmid 2011) for ANN search using Faiss library (Johnson, Douze, and Jégou 2017), where the codebook size of coarse quantizer $\sqrt{n} \approx 316$, the number of subvectors $M = 128$ is used and the number of clusters to scan is varied. When compared to the exhaustive k-NN search, ANN is generally good at approximating the top results of the search but its ranking and scores can be out of order and imprecise. This makes it applicable to truncation which only requires a coarse set of the top results. Graph construction, however, would be negatively affected by even small differences in the scores of the search results. We therefore use ANN search only for truncation and use exhaustive k-NN search for graph construction.

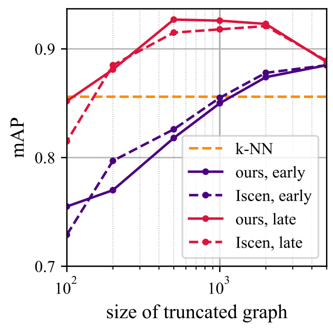
Since the k in k-NN search for graph construction is small compared to truncation, it can be efficiently processed even without using approximate search, especially by using Faiss on a GPU. As a result, the entire process of truncation and graph construction takes ∼1ms using a single GPU per image. Diffusion processes take ∼6 minutes to process the whole Oxford105k dataset using global features and a truncation size of 5,000 (3.4 ms per image). We measured using a single core of CPU, but these processes can be easily parallelized. Compared to the offline processing in (Iscen et al. 2018a) which takes a few hours, our method is much faster.

Influence of subgraph normalization

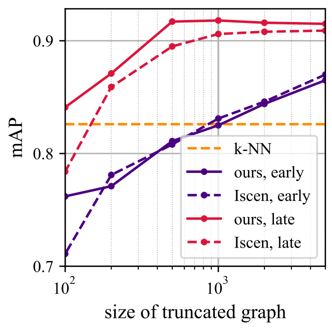
In (Iscen et al. 2017), truncation enables diffusion on large scale datasets but is described to be detrimental to retrieval performance. The authors claim that retrieval performance of diffusion grows as the percentage of the whole graph used grows, with a complete graph without any truncation having the best performance. However, we argue that the retrieval performance is mainly influenced by the early truncation leading to subgraph normalization. In order to avoid subgraph normalization, we first obtain the complete graph and apply late truncation (slicing) on the complete matrix $\mathcal{L}_\alpha$ in the process of diffusion. For comparison, we implement Iscen's method with and without subgraph normalization. We vary the truncation size L to observe the influence of subgraph normalization on the retrieval performance.

The experimental results with global features for the Oxford and Paris datasets are presented in Fig. 6. On the Oxford datasets, it is clear that the performance is significantly improved without subgraph normalization when the truncation size L is small, but its effectiveness on the Paris datasets is smaller. We observe that merely performing k-NN search on the Paris datasets already results in a high retrieval performance. This could mean that the Paris datasets have standard-shaped manifolds conducive to comparison by Euclidean distance, so it limits the benefits gained from diffusion.

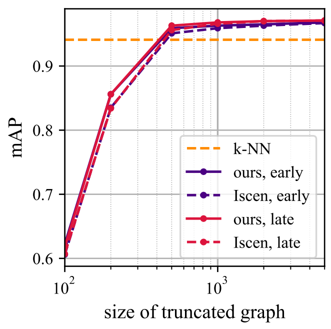
While previous works always encouraged using larger values of L for performance gains, our method can achieve state-of-the-art performance with smaller L values found through validation. As a bonus, a smaller truncation size L will lead to acceleration of the offline computation.



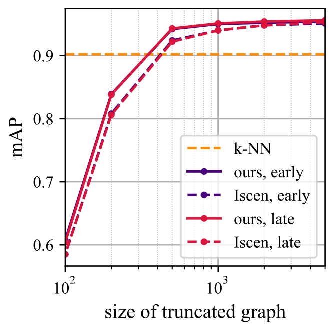
(a) Oxford5k



(b) Oxford105k



(c) Paris6k



(d) Paris106k

Figure 6: Retrieval performance (mAP) vs. the size of truncated graph L using early truncation and late truncation.

	Method	Feature	Global	Regional	Oxf5k	Oxf105k	Par6k	Par106k
w/o regional feature	k-NN search		✓		79.5	72.1	84.5	77.1
	k-NN + AQE (Chum et al. 2007)	R-MAC (VGG)	✓		85.4	79.7	88.4	83.5
	Iscen's diffusion (Iscen et al. 2017)		✓		85.7	82.7	94.1	92.5
	Proposed diffusion		✓		**89.7**	**86.8**	**94.7**	**92.9**
	k-NN search		✓		83.9	80.8	93.8	89.9
	k-NN + AQE (Chum et al. 2007)	R-MAC (ResNet)	✓		89.6	88.3	95.3	92.7
	Iscen's diffusion (Iscen et al. 2017)		✓		87.1	87.4	96.5	95.4
	Proposed diffusion		✓		**92.6**	**91.8**	**97.1**	**95.6**
w/ regional feature	R-match (Razavian et al. 2016)			✓	81.5	76.5	86.1	79.9
	R-match + AQE (Chum et al. 2007)			✓	83.6	78.6	87.0	81.0
	Iscen's diffusion (Iscen et al. 2017)	R-MAC (VGG)	✓	✓	93.2	90.3	**96.5**	92.6
	Proposed diffusion			✓	91.8	88.6	93.9	89.2
	Proposed diffusion w/ late fusion		✓	✓	**93.5**	**91.2**	96.1	**93.8**
	R-match (Razavian et al. 2016)			✓	88.1	85.7	94.9	91.3
	R-match + AQE (Chum et al. 2007)			✓	91.0	89.6	95.5	92.5
	Iscen's diffusion (Iscen et al. 2017)	R-MAC (ResNet)	✓	✓	95.8	94.2	96.9	95.3
	Proposed diffusion			✓	95.9	94.8	97.6	95.6
	Proposed diffusion w/ late fusion		✓	✓	**96.2**	**95.2**	**97.8**	**96.2**

Table 1: Performance comparison with the state of the art. We used R-MAC features extracted with VGG (Radenović, Tolias, and Chum 2016) and ResNet101 (Gordo et al. 2016).

Comparison to other methods

Table 1 compares our method with other competitive methods that use global and regional features. Testing on all datasets using global features, we observe up to a 5% increase in mAP performance compared to the previous state-of-the-art. Similarly, on diffusion with regional features, we achieve competitive or better performance. We note that Iscen et al. used global features to guide the truncation in their regional diffusion. For each query, they first apply k-NN search using the global features to obtain the closest images to that query. Subsequently, these results are used as a bounded set to perform regional diffusion. In contrast, we only use regional features in our regional diffusion for simplicity. To exploit global features, we apply a simple late fusion by computing a weighted mean of scores from regional and global diffusion, setting the weight for regional diffusion to 0.75. This further increases the performance (*proposed diffusion w/ late fusion* in Table 1), leading to better performance than Iscen et al. on all datasets.

Conclusion

In this paper, we propose a novel efficient diffusion to achieve fast retrieval during runtime with significant improvement in retrieval performance. We experimentally show that our approach has a similar efficiency to k-NN search, which is 10$\sim$ times faster than existing diffusion methods with global features. Moreover, our method achieves state-of-the-art performance on Oxford and Paris datasets. In conclusion, our method makes diffusion more practical for image retrieval on large-scale datasets, and has the potential to improve retrieval in other fields, such as text and video.

Acknowledgment

This study is supported by CREST (JPMJCR 1686).

References

Babenko, A.; Slesarev, A.; Chigorin, A.; and Lempitsky, V. 2014. Neural codes for image retrieval. In *ECCV*. Springer.

Chum, O.; Philbin, J.; Sivic, J.; Isard, M.; and Zisserman, A. 2007. Total recall: Automatic query expansion with a generative feature model for object retrieval. In *ICCV*.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *CVPR*, 248–255. IEEE.

Dong, W.; Moses, C.; and Li, K. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *WWW*.

Donoser, M., and Bischof, H. 2013. Diffusion processes for retrieval revisited. In *CVPR*.

Ge, T.; He, K.; Ke, Q.; and Sun, J. 2014. Optimized product quantization. *TPAMI* 36(4):744–755.

Gordo, A.; Almazán, J.; Revaud, J.; and Larlus, D. 2016. Deep image retrieval: Learning global representations for image search. In *ECCV*.

Grady, L. 2006. Random walks for image segmentation. *TPAMI* 28(11):1768–1783.

Iscen, A.; Tolias, G.; Avrithis, Y.; Furon, T.; and Chum, O. 2017. Efficient diffusion on region manifolds: Recovering small objects with compact cnn representations. In *CVPR*.

Iscen, A.; Avrithis, Y.; Tolias, G.; Furon, T.; and Chum, O. 2018a. Fast spectral ranking for similarity search. In *CVPR*.

Iscen, A.; Tolias, G.; Avrithis, Y.; and Chum, O. 2018b. Mining on manifolds: Metric learning without labels. In *CVPR*.

Jegou, H.; Douze, M.; and Schmid, C. 2011. Product quantization for nearest neighbor search. *TPAMI* 33(1):117–128.

Johnson, J.; Douze, M.; and Jégou, H. 2017. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*.

Malkov, Y. A., and Yashunin, D. A. 2016. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *arXiv preprint arXiv:1603.09320*.

Nocedal, J., and Wright, S. J. 2006. *Numerical optimization 2nd*. Springer.

Page, L.; Brin, S.; Motwani, R.; and Winograd, T. 1999. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab.

Philbin, J.; Chum, O.; Isard, M.; Sivic, J.; and Zisserman, A. 2007. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*.

Philbin, J.; Chum, O.; Isard, M.; Sivic, J.; and Zisserman, A. 2008. Lost in quantization: Improving particular object retrieval in large scale image databases.

Radenovic, F.; Iscen, A.; Tolias, G.; Avrithis, Y.; and Chum, O. 2018. Revisiting oxford and paris: Large-scale image retrieval benchmarking. In *CVPR*.

Radenović, F.; Tolias, G.; and Chum, O. 2016. Cnn image retrieval learns from bow: Unsupervised fine-tuning with hard examples. In *ECCV*.

Razavian, A. S.; Sullivan, J.; Carlsson, S.; and Maki, A. 2016. Visual instance retrieval with deep convolutional networks. *ITE MTA* 4(3):251–258.

Saad, Y. 2003. *Iterative methods for sparse linear systems*, volume 82. SIAM.

Tolias, G.; Sicre, R.; and Jégou, H. 2015. Particular object retrieval with integral max-pooling of cnn activations. In *ICLR*.

Zhou, D.; Bousquet, O.; Lal, T. N.; Weston, J.; and Schölkopf, B. 2004a. Learning with local and global consistency. In *NIPS*.

Zhou, D.; Weston, J.; Gretton, A.; Bousquet, O.; and Schölkopf, B. 2004b. Ranking on data manifolds. In *NIPS*.