Piecewise Hashing: A Deep Hashing Method for Large-Scale Fine-Grained Search

Yimu Wang[1], Xiu-Shen Wei[2(✉)], Bo Xue[1], and Lijun Zhang[1]

[1] State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
{wangym,xueb,zhanglj}@lamda.nju.edu.cn

[2] PCA Lab, Key Lab of Intelligent Perception and Systems for High-Dimensional Information of Ministry of Education, and Jiangsu Key Lab of Image and Video Understanding for Social Security, School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing, China
weixs@njust.edu.cn

Abstract. Fine-grained search task such as retrieving subordinate categories of birds, dogs or cars, has been an important but challenging problem in computer vision. Although many effective fine-grained search methods were developed, with the amount of data increasing, previous methods fail to handle the explosive fine-grained data with low storage cost and fast query speed. On the other side, since hashing sheds its light in large-scale image search for dramatically reducing the storage cost and achieving a constant or sub-linear time complexity, we leverage the power of hashing techniques to tackle this valuable yet challenging vision task, termed as *fine-grained hashing* in this paper. Specifically, our proposed method consists of two crucial modules, *i.e.*, the bilinear feature learning and the binary hash code learning. While the former encodes both local and global discriminative information of a fine-grained image, the latter drives the whole network to learn the final binary hash code to present that fine-grained image. Furthermore, we also introduce a novel multi-task hash training strategy, which can learn hash codes of different lengths simultaneously. It not only accelerates training procedures, but also significantly improves the fine-grained search accuracy. By conducting comprehensive experiments on diverse fine-grained datasets, we validate that the proposed method achieves superior performance over the competing baselines.

Keywords: Fine-grained image retrieval · Deep hashing · Multi-task learning · Large-scale methods

1 Introduction

As a fundamental and challenging problem in computer vision, fine-grained image analysis (FGIA) [26] has been an active research area for several decades.

Y. Wang—Is a student.

Y. Peng et al. (Eds.): PRCV 2020, LNCS 12306, pp. 432–444, 2020.
https://doi.org/10.1007/978-3-030-60639-8_36

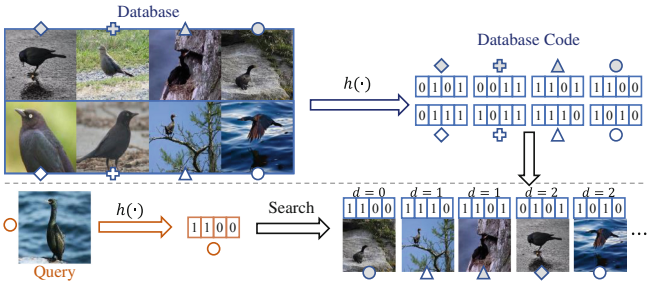

Fig. 1. An example of the *fine-grained hashing* problem. Its goal is to retrieve images belonging to multiple subordinate categories of a super-category (*i.e.*, birds). Each fine-grained image will be mapped into a binary code by a hashing function $h(\cdot)$. We return the images which are in the same variety as the query image since they have smaller hamming distance d. Here, different shapes of markers represent different subordinate categories.

One of the central tasks in FGIA is fine-grained image search [25,28], whose goal is to retrieve images belonging to multiple subordinate categories of a super-category (*e.g.*, different species of birds, different models of cars, or different kinds of clothes) and then return the images which are in the same variety as the query image, cf. Fig. 1. Due to small inter-class variations and large intra-class variations caused by the fine-grained nature, it is desirable to capture the discriminative parts of these fine-grained objects to form a powerful image representation.

During the booming of deep learning, recent years have witnessed effective progress of fine-grained search using deep learning techniques. Some trials [25] employed the pre-trained CNN models to unsupervisedly locate fine-grained objects and then obtain the deep features for image search. Later, to break through the limitation of unsupervised fine-grained search by pre-trained models, some works [29,30] tended to discovery novel loss functions under the supervised metric learning paradigm. Although these previous methods obtain good image search accuracy, they still cannot handle the *large-scale* fine-grained data with low storage cost and fast query speed, especially for the significant increment of data amount in the deep learning era. Moreover, with the rapidly explosive growth of vision data, more and more large-scale fine-grained datasets [3,8,9,24] are proposed recently. In consequence, the demand of handling large-scale data for fine-grained search methods has increased dramatically.

To deal with the large-scale data amount challenge, in machine learning, approximate nearest neighbor (ANN) search [1,2] has attracted much attention in recent years. Among ANN search methods, hashing [6,16,27] has been an active and representative subarea, which is able to map the data points to binary codes with hash functions by preserving the similarity in the original space of the data points. Thanks to the binary hash code representation, the storage cost for the large-scale data can be drastically reduced, and also the time complexity can be constant or sub-linear.

Therefore, in this paper, to alleviate the large-scale fine-grained search problem, we investigate a novel problem, *i.e.*, *fine-grained hashing*, as a step towards efficient and effective large-scale fine-grained image search. To realize fine-grained hashing, we propose an end-to-end trainable network which is inspired by a state-of-the-art fine-grained recognition backbone model, *i.e.*, bilinear pooling [14] and is also tailored for the fine-grained nature. Specifically, as shown in Fig. 2, our proposed method consists of a bilinear feature learning module and a hash code learning module. The former can encode the discriminative information of a fine-grained image by utilizing both global and local streams into the intermediate feature vector, while the latter plays the role to drive the whole network training and obtain the final binary hash codes.

The key novelty of our method is the "piecewise" and "synergistic" proposals. First, based on the outer product operation in bilinear pooling [14], the obtained feature can be viewed as a set of sub-vectors, and thus, each of which implicitly attends to one part of the image. We perform the part-level local stream mechanism upon these part-level sub-vectors "piece by piece" ("piecewise" proposal) to capture the discriminative cues for effectively representing fine-grained parts (*e.g.*, "tufted heads", "red-yellow stripe"), which favors the fine-grained nature. Additionally, a parallel global stream is also designed to obtain the global-level image feature. By aggregating both local (part-level) and global (object-level) information, the final image representation can be fed into the hash code learning module. Second, in order to accelerate the training process, we simultaneously train hash functions of different lengths in a novel multi-task hash training framework. Hash functions of different lengths share the convolutional layers with each other to learn feature representations (the "synergistic" proposal), while saving 70% training and inference time (with four functions) both theoretically and practically.

Empirical results on five fine-grained datasets, *i.e.*, *CUB* [21], *Aircraft* [17], *NABirds* [8], *VegFru* [9], and *Food101* [3] show that our piecewise hashing method significantly outperforms competing baselines, including the deep or non-deep hashing state-of-the-arts. Meanwhile, we perform our proposed method on a popular generic dataset, *i.e.*, *CIFAR-10* [11], to demonstrate that our method is able to achieve the best search accuracy when facing the generic images.

The main contributions of this paper are summarized as:

- We study the practical and challenging fine-grained hashing problem and propose an end-to-end trainable network with a novel multi-task hash training strategy to deal with the large-scale fine-grained search problem.
- We devise a novel piecewise hashing method consisting a bilinear feature learning module and a hash code learning module. The former resorts to the special structure of the bilinear CNN features to learn discriminative image features by leveraging both the global and local streams, while the latter drives the whole network training and returns the final fine-grained hash codes. Besides, the proposed multi-task strategy improves the efficiency and accuracy by synergistically learning the common layers across hash functions of different lengths.

– We conduct experiments on five fine-grained datasets and one generic dataset. Empirical results show that our proposed network outperforms previous hashing methods for both fine-grained images and generic images.

2 Related Work

2.1 Fine-Grained Image Search

Fine-grained image search has attracted increasing research attention in recent years, where the instances are within a subordinate category and different in slight patterns. In the literature, [28] is the first attempt to use handcraft features for fine-grained image search. Inspired by the power of deep learning, some unsupervised and supervised deep fine-grained image search methods have been proposed. Selective convolutional descriptor aggregation (SCDA) [25] is the first work on deep fine-grained image search, which directly discovers the discriminative parts in the images unsupervisedly. [29] defines the fine-grained search as a deep metric learning problem and tries to learn discriminative representations by designing specific loss functions. At the same time, with the rapid growth of fine-grained visual data, more and more large-scale fine-grained datasets have been proposed containing abundant labeled images, to name a few, *RPC* [24], *Cars* [5], *DA-Retail* [23] and *NABirds* [8], facilitating further research. Nevertheless, these previous fine-grained search work cannot handle large-scale fine-grained data, as they represented the images with *high-dimensional real-valued* vectors.

2.2 Hashing

Hashing is a widely used method for ANN search in large scale image retrieval with encouraging efficiency in both speed and storage. Existing hashing methods can be roughly categorized into unsupervised and supervised hashing. Unsupervised hashing methods [18] learn hash functions from unlabeled data, *e.g.*, LSH [6], SH [27] and ITQ [7]. Supervised hashing methods [22] attempt to leverage supervised information (*e.g.*, similarity matrix or label information) to improve the quality of hash codes, *e.g.*, KSH [16] and SDH [19]. Inspired by powerful feature representations learning with deep neural networks [20], the deep supervised hashing [10] adopting deep learning to generate high-level semantic features has been proposed. Deep Pairwise-Supervised Hashing (DPSH) [13] preserves relative similarity between image triplets straightly by integrating feature learning and hash functions in an end-to-end manner. Further, HashNet [4] tackles the data imbalance problem between similar and dissimilar pairs and alleviates this drawback by adjusting the weights of similar pairs. To additionally accelerate the training procedure, several asymmetric deep hashing methods are proposed, *i.e.*, Asymmetric Deep Supervised Hashing (ADSH) [10], which only learns the hash function for query points to alleviate time-consuming. However, previous hash methods were designed for generic images, which were not capable for *fine-grained* images search.

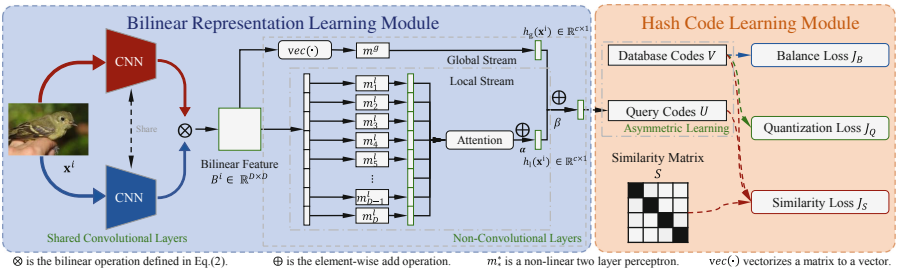

Fig. 2. Overview structure of our proposed piecewise hashing, which consists of two main modules, *i.e.*, bilinear representation learning and hash code learning. For bilinear representation learning, we develop both global (object-level) and local (part-level) stream to discover discriminative information for fine-grained objects by leveraging the bilinear features tailored for fine-grained images. Specifically, for better capturing different but subtle part features, we perform the part-level attention on the local stream. For hash code learning, three loss functions corresponding to three hash learning principles are employed, and the network can be trained in an end-to-end fashion with only image-level supervisions.

3 Proposed Method

In this section, we elaborately introduce our piecewise hashing method. Besides, for improving the efficiency, we also propose a simple yet effective multi-task training strategy to jointly learn hash functions of different code-lengths while sharing the convolutional layers of the bilinear representation learning module.

Vectors and scalars are bold lower case italic letters and lower case italic letters, such as $\boldsymbol{\alpha}$ and c. The i-th element of a vector $\boldsymbol{\alpha}$ and the i,j-th element of a matrix S are represented as $\boldsymbol{\alpha}_i$ and $S_{i,j}$, while the i-th column and j-th row of a matrix S are presented as $S_{*,i}$ and $S_{j,*}$. Assume that we have m training data points and n database points denoted as $X = \left\{\mathbf{x}^i\right\}_{i=1}^{m}$ and $Y = \left\{\mathbf{y}^j\right\}_{j=1}^{n}$. The pairwise supervised information is denoted as $S \in \{-1,+1\}^{m\times n}$ in training. If point $\mathbf{x}^i$ and point $\mathbf{y}^j$ are similar, $S_{ij} = +1$, otherwise $S_{ij} = -1$. Under this condition, the goal of supervised hashing is to learn binary hash codes $\boldsymbol{code} \in \{-1,+1\}^c$ for each point, and its corresponding hashing function $h(\cdot;\Theta)$, where c is the target length of binary code. Additionally, we use $U = \left\{\mathbf{u}^i\right\}_{i=1}^{m} \in \{-1,+1\}^{m\times c}$ and $V = \left\{\mathbf{v}^j\right\}_{j=1}^{n} \in \{-1,+1\}^{n\times c}$ to denote the learned binary hash codes for training points and database points. The hash codes have to preserve the similarity S between each point, which means the Hamming distance between $\mathbf{u}^i$ and $\mathbf{v}^j$ should be as small as possible if $S_{ij} = +1$.

3.1 Our Piecewise Hashing

Our model is shown in Fig. 2, which contains two modules: the bilinear representation learning module and the hash-code learning module. We will elaborate them as follows.

Bilinear Representation Learning Module. In this part, we introduce our network architecture in detail. For clarity, all the equations take one data point $\mathbf{x}^i$ as input.

Bilinear pooling (BCNN) [14] is an effective architecture tailored for the fine-grained task. BCNN represents an image as an outer product of features derived from two CNNs and discovers localized feature interactions. Inspired by BCNN, we acquire bilinear representations and further generate hash codes based on them. Specifically, we assume the outputs of two CNNs in BCNN are re-organized into $f_A(\mathbf{x}^i) \in \mathbb{R}^{D\times L}$ and $f_B(\mathbf{x}^i) \in \mathbb{R}^{D\times L}$, where D denotes the dimension of the outputs and L denotes the spatial locations. Then, we define the bilinear feature at location l as:

$$\text{bilinear}(l, \mathbf{x}^i, f_A, f_B) = f_A(l, \mathbf{x}^i) f_B(l, \mathbf{x}^i)^\top . \tag{1}$$

The sum pooling aggregates the bilinear combination of features across all locations in the image to obtain the global bilinear representation $B^i \in \mathbb{R}^{D\times D}$ as follows:

$$B^i = \sum\nolimits_{l=1}^{L} \text{bilinear}(l, \mathbf{x}^i, f_A, f_B) . \tag{2}$$

After obtaining the bilinear representation B^i, the global stream and the local stream are paralleled to derive outputs. In the global stream, the D^2-dimension bilinear representation B^i is mapped to a c-dimension binary-like output. A straightforward solution to realize this mapping is employing a multi-layer perception (MLP) $m(\cdot)$. Therefore, we derive the global (object-level) binary-like output via an MLP $m^g(\cdot) : \mathbb{R}^{D^2} \to \mathbb{R}^c$ as:

$$h_g(\mathbf{x}^i) = m^g(\text{vec}(B^i)) , \tag{3}$$

where $\text{vec}(\cdot)$ vectorizes a $D \times D$ matrix to a $D^2 \times 1$ vector.

The key novelty of our method is "piecewise" in the local stream. Based on the outer product operation above, the bilinear representations can be viewed as a set of sub-vectors (pieces), and thus, each of which implicitly attends to a highly localized image feature (*e.g.*, "tufted heads" and "red-yellow stripe"). Hence, in the local stream, we apply the part-level attention ("piecewise") to emphasize the discriminative pieces and understate the pointless pieces over the bilinear representation. Specifically, we locally re-organize the bilinear representation B^i by column and map each column to a local binary-like output via an MLP. Then we derive the local (part-level) bilinear binary-like output from the convex combination of local binary-like outputs for each column as follows:

$$\begin{aligned} h_l(\mathbf{x}^i) &= \text{Attention}(\boldsymbol{\alpha}, B^i) = \sum\nolimits_{d=1}^{D} \boldsymbol{\alpha}_d m_d^l(B^i_{*,d}) \\ &\text{s.t.} \sum\nolimits_{d=1}^{D} \boldsymbol{\alpha}_d = 1 , \end{aligned} \tag{4}$$

where $\boldsymbol{\alpha} \in \mathbb{R}^{D\times 1}$ is a learnable parameter, and $m_d^l(\cdot) : \mathbb{R}^D \to \mathbb{R}^c$ represents the MLP for the d-th column.

The overall binary-like output $h_{soft}(\mathbf{x}^i;\Theta)$ is also the convex combination of the global bilinear binary-like output $h_g(\mathbf{x}^i)$ and the local bilinear binary-like output $h_l(\mathbf{x}^i)$. Discrete hash codes $h(\mathbf{x}_i;\Theta)$ are derived by employing $\text{sign}(\cdot)$ on the overall binary-like output as follows:

$$\begin{aligned} h(\mathbf{x}_i;\Theta) &= \text{sign}(h_{soft}(\mathbf{x}^i;\Theta)) = \text{sign}(\beta h_g(\mathbf{x}^i) + (1-\beta)h_l(\mathbf{x}^i)) \\ &\text{s.t. } 0 \le \beta \le 1\,, \end{aligned} \tag{5}$$

where β is a learnable parameter, and Θ is the parameters of the whole network including $\boldsymbol{\alpha}$ and β.

Hash-Code Learning Module. In this section, we design our objective function by three following principles including: preserving the similarity of data points in the original space, distributing the codes to uniformly fulfill the code space, and generating compact binary codes. In total, these principles correspond to three loss functions to drive the training procedure of the whole network.

To preserve the similarity S during the training procedure, the previous work [10,15] achieves it by minimizing the ℓ_2 loss between similarity S_{ij} and inner product of query-database binary code pairs $\mathbf{u}^i\mathbf{v}^{j\top}$. As all the hash codes are discrete, it is hard to complete the training of hash functions. Previous works always employ discrete function $\text{sign}(\cdot)$ to calculate the hash codes $\mathbf{u}_i$ and utilize $\text{sigmoid}(\cdot)$ or $\tanh(\cdot)$ functions to approximate $\text{sign}(\cdot)$. The common similarity loss they minimize is as:

$$\begin{aligned} J_{S0} &= \sum\nolimits_{i=1}^{m}\sum\nolimits_{j=1}^{n} \|\mathbf{u}^i\mathbf{v}^{j\top} - cS_{ij}\|^2 \\ &= \sum\nolimits_{i=1}^{m}\sum\nolimits_{j=1}^{n} \|\,\text{sign}(h_{soft}(\mathbf{x}^i;\Theta))\mathbf{v}^{j\top} - cS_{ij}\|^2 \\ &\approx \sum\nolimits_{i=1}^{m}\sum\nolimits_{j=1}^{n} \|\,\tanh(h_{soft}(\mathbf{x}^i;\Theta))\mathbf{v}^{j\top} - cS_{ij}\|^2\,. \end{aligned} \tag{6}$$

Nevertheless, employing such non-linear functions would inevitably slow down or even affect the convergence of the network [12]. To ease such a problem, we directly optimize the real-valued network outputs instead of the approximation of $\mathbf{u}_i$, and impose a regularizer, *i.e.*, the quantization loss J_Q, on the real-valued network outputs to approach the desired discrete values. Particularly, we present the J_S and the regularizer J_Q as:

$$J_S = \sum\nolimits_{i=1}^{m}\sum\nolimits_{j=1}^{n} \|h_{soft}(\mathbf{x}^i;\Theta)\mathbf{v}^{j\top} - cS_{ij}\|^2\,, \tag{7}$$

$$J_Q = \sum\nolimits_{i=1}^{m} \|\mathbf{u}^i - h_{soft}(\mathbf{x}^i;\Theta)\|^2\,. \tag{8}$$

One of the advantages of hash is storage efficiency. This brings another goal to accomplish, that is, hash-codes points should uniformly fulfill the 2^c code space. Hence, for fully utilizing each bit of hash codes, we set a term J_B to make each bit of the hash codes be balanced on all the points. Ideally, if we sum up all the hash-codes, the results should be $\mathbf{0}$. This term can be presented as:

$$J_B = \sum\nolimits_{c}\left[\left|\sum\nolimits_{i=1}^{m}\mathbf{u}_c^i\right| + \left|\sum\nolimits_{j=1}^{n}\mathbf{v}_c^j\right|\right]\,. \tag{9}$$

The final objective function we minimize is as follows:

$$\begin{aligned} \min_{V,\Theta} J &= J_S + \lambda J_Q + \mu J_B \\ \text{s.t. } & V \in \{-1,+1\}^{n\times c}, \end{aligned} \tag{10}$$

where λ and μ are hyper-parameters, which are set to 200 and 0.1 in all the experiments.

3.2 Multi-task Hash Training Strategy

The another key novelty of our method is the "synergistic" proposal. Multi-task hash training is a simple but effective training strategy. In the previous deep hashing work [4,10], though the convolutional layers can be shared, they still separately train models of different code-lengths, while in shallow hash methods [6,16], they usually utilize the same features to train several hash functions of different lengths.

To reduce the training time and the redundancy among different hash functions, we propose a multi-task hash training strategy enabling us to learn hash functions of different code-lengths simultaneously. We split the network into the convolutional layers, and the non-convolutional layers, *i.e.*, the global and local stream, which are directly related to the code-length shown in Fig. 2. Specifically, hash functions of different code-lengths contain the non-convolutional layers, while sharing the convolutional layers with each other ("synergistic"). When we learn four hash functions of 12 bits, 24 bits, 32 bits, and 48 bits concurrently, the corresponding objective function becomes as:

$$\begin{aligned} \min_{V_{12},V_{24},V_{32},V_{48},\Theta} & J_{mul} = J_{12} + J_{24} + J_{32} + J_{48} \\ \text{s.t. } & V_{12} \in \{-1,+1\}^{n\times 12}, V_{24} \in \{-1,+1\}^{n\times 24}, \\ & V_{32} \in \{-1,+1\}^{n\times 32}, V_{48} \in \{-1,+1\}^{n\times 48}, \end{aligned} \tag{11}$$

where J_{12}, J_{24}, J_{32} and J_{48} are the objective functions where we set $c = 12, 24, 32, 48$ in Eq. (10). Note that, there is no hyper-parameter between these four terms, which reveals our multi-task learning strategy is not tricky.

Multi-task hash training strategy enables us to learn intermediate representations and hash functions of different code-lengths simultaneously, saving computation time and memory space. As the code length grows, the model would contain more parameters and then get prone to overfitting. This strategy can help overcome such problems by sharing parts of parameters on the whole network thus benefiting the training procedure. Additionally, the theoretical analyses in Sect. 3.4 prove that our strategy is efficient.

3.3 Learning Algorithm

In this section, we present an alternating learning algorithm to learn V and Θ of Eq. (10). The pseudo codes of our algorithm can be found in Appendix.

The parameters are learned alternatively, which means we update one parameter with another parameter fixed. The details are presented below.

Normally, we are only given the database points $Y = \{\mathbf{y}^j\}_{j=1}^n$ and the pairwise supervised information S between them, we can learn hash-codes and hash functions by sampling a subset or the whole set of Y as the query set X for training, *i.e.*, $X \subseteq Y$. To accelerate the training procedure, we construct a subset X of database Y instead of using the whole database. Hence, we only consider the item related to V in J_B of Eq. (10).

Learn Θ with V fixed. When V is fixed, we learn and update the parameter Θ of our neural network by back-propagation (BP) algorithm.

Specifically, for each query point $\mathbf{x}^i$ in the query points, the gradient can be calculated as:

$$\frac{\partial J}{\partial \mathbf{z}^i} = \frac{\partial J_S}{\partial \mathbf{z}^i} + \lambda \frac{\partial J_Q}{\partial \mathbf{z}^i} + \mu \frac{\partial J_B}{\partial \mathbf{z}^i} = \sum\nolimits_{i=1}^{m} \left[\left(\mathbf{z}^i \mathbf{v}^{j\top} - cS_{ij} \right) \mathbf{v}^j + \lambda (\mathbf{u}^i - \mathbf{z}^i) \right], \quad (12)$$

where $\mathbf{z}^i = h_{soft}(\mathbf{x}^i, \Theta)$. Once we have the $\frac{\partial J}{\partial \mathbf{z}^i}$, we can compute $\frac{\partial J}{\partial \Theta}$ based on $\frac{\partial J}{\partial \mathbf{z}^i}$ using the chain rule and the back propagation algorithm to update Θ.

Learn V with Θ fixed. When Θ is fixed, we can easily reformulate the Eq. (10) as follows:

$$\begin{aligned} \min_V J(V) &= \operatorname{tr}\left(V \left[Z^\top V Z^\top - 2cZ^\top S - 2\lambda \bar{V}^\top \right] \right) + \mu \sum\nolimits_{k=1}^{c} |\mathbf{1} \cdot V_{*,k}| + \epsilon \\ &\text{s.t. } V \in \{-1, +1\}^{n \times c}, \end{aligned} \quad (13)$$

where $Z = \left[\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^m \right] \in [-1, +1]^{m \times c}$, $\mathbf{1} = [1, 1, \ldots, 1] \in \{1\}^{1 \times n}$, $\bar{V} = \{\bar{V}_j = \mathbf{I}(\mathbf{y}^j \in X)\mathbf{u}^j\}_{j=1}^{n \times 1} \in \mathbb{R}^{n \times c}$, $\mathbf{I}(\cdot)$ is the indicator function, and "ϵ" is a constant independent of V. For convenience, we let $Q = (Z^\top V Z^\top - 2cZ^\top S - 2\lambda \bar{V}^\top)^\top$. Then we can rewrite this problem as:

$$\begin{aligned} \min_V J(V) &= \operatorname{tr}\left(V Q^\top \right) + \mu \sum\nolimits_{k=1}^{c} |\mathbf{1} \cdot V_{*,k}| + \epsilon \\ &\text{s.t. } V \in \{-1, +1\}^{n \times c}. \end{aligned} \quad (14)$$

To ease this problem, we update one bit a time. We alternatively update one column of V with the other columns fixed. Hence, the optimal solution of this bit by bit problem is:

$$J(V_{*k}) = \begin{cases} = -\operatorname{sign}(Q_{*,k} + \mu \mathbf{1}^\top), \mathbf{1} \cdot V_{*,k} \geq 0 \\ = -\operatorname{sign}(Q_{*,k} - \mu \mathbf{1}^\top), \mathbf{1} \cdot V_{*,k} < 0 \end{cases}. \quad (15)$$

3.4 Out-of-Sample Extension and Model Analyses

After completing the learning procedure, hash-codes for all the database points can be easily generated. As for the point $\mathbf{x}_q$ in query points, we can use $\mathbf{u}^q = \operatorname{sign}(h_{soft}(\mathbf{x}^q, \Theta))$ to generate binary hash-codes. The total computational complexity for training our piecewise hashing is $O(n)$. For the training complexity, while the complexity of separate training without multi-task

is $O_4(n) \approx 4O(n)$, multi-task strategy accelerates model training by saving 70% time cost with the complexity of $O_{4multi-task}(n) \approx 1.2O(n)$. Thus, our proposal of simultaneously training hash functions of different lengths is theoretically efficient.

4 Experiments

In this section, we evaluate the performance of our proposed method on both fine-grained and generic datasets, and then compare with state-of-the-art approaches.

We evaluate the performance of our proposed method on six datasets, *i.e.*, CUB [21], Aircraft [17], NABirds [8], VegFru [9], Food101 [3] and CIFAR-10 [11]. For the above datasets, we follow the standard split proposed with these datasets, while two images will be treated as a ground-truth similar pair if they share the same label. We evaluate the retrieval performance by adopting two evaluation metrics: mean Average Precision (mAP) and Precision-recall Curves (PR Curves) based on lookup. Details are available in Appendix. All the data are reported with average values running five times. We compare our deep piecewise hashing method with several state-of-the-art hashing methods, including shallow methods, *i.e.*, LSH [6], ITQ [7], SH [27], SDH [19] and KSH [16], and deep supervised methods, *i.e.*, DSH [15], DPSH [13], HashNet [4], and ADSH [10]. For all deep hashing methods, we use raw images resized to 224×224 as inputs. For traditional shallow methods, we extract 4096-dimensional deep features by the VGG-16 model pre-trained with ImageNet to conduct fair comparisons. Besides, for all the state-of-the-art hashing methods, we prefer to employ the hyper-parameters introduced in their papers.

The mAP results on six datasets are presented in Table 1. Additional results of PR Curves and Top-5K mAP on all the datasets are available in Appendix.

Search Accuracy on the Fine-Grained Datasets: Our proposed method with the multi-task learning strategy outperforms other hashing methods across different code-lengths on fine-grained datasets. Specifically, the mAP of our piecewise hashing obtains relative improvements over the next-best state-of-the-art methods of 14.36%, 37.52%, 32.82%, and 24.30% on CUB. We notice that similar improvements are achieved on other fine-grained datasets.

Search Accuracy on the Generic Dataset: Moreover, as shown in Table 1, our piecewise hashing still outperforms other hashing methods on the generic dataset, *i.e.*, *CIFAR-10*, obtaining significant increment of 1.90%, 4.53%, 2.85%, and 2.57% for different lengths of hash codes, respectively.

Table 1. Comparisons of mAP w.r.t. different number of bits on six datasets, *CIFAR-10*, *CUB*, *Aircraft*, *NABirds*, *VegFru* and *Food101*. Best in bold.

Method	Backbone	*CIFAR-10*				*CUB*				*Aircraft*			
		12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
LSH	–	0.1162	0.1215	0.1224	0.1244	0.0152	0.0235	0.0288	0.0415	0.0169	0.0219	0.0238	0.0282
SH	–	0.1316	0.1289	0.1287	0.1274	0.0666	0.0809	0.0893	0.1048	0.0328	0.0385	0.0404	0.0428
ITQ	–	0.1544	0.1607	0.1630	0.1656	0.0855	0.1196	0.1376	0.1549	0.0438	0.0528	0.0582	0.0605
KSH	–	0.2353	0.2563	0.2669	0.2763	0.1125	0.1502	0.1722	0.1954	0.0557	0.0738	0.0814	0.0892
SDH	–	0.1746	0.2140	0.2115	0.2362	0.0964	0.1442	0.1491	0.1827	0.0489	0.0636	0.0690	0.0765
DPSH	ResNet50	0.6872	0.7024	0.7281	0.7437	0.0685	0.0885	0.1008	0.1148	0.0874	0.1087	0.1354	0.1394
DSH	ResNet50	0.7230	0.7644	0.7746	0.7920	0.1360	0.1899	0.2237	0.2744	0.0814	0.1066	0.1221	0.1445
HashNet	ResNet50	0.7261	0.7614	0.7858	0.7950	0.1203	0.1777	0.1993	0.2213	0.1491	0.1775	0.1942	0.2032
ADSH	ResNet50	0.6599	0.7413	0.7590	0.7672	0.0209	0.1002	0.2997	0.4535	0.0924	0.2314	0.3204	0.4278
Ours	VGG-16	**0.7451**	**0.8097**	**0.8143**	**0.8207**	**0.2796**	**0.5651**	**0.6279**	**0.6956**	**0.4392**	**0.5662**	**0.5997**	**0.6296**
Method	Backbone	*NABirds*				*VegFru*				*Food101*			
		12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits	12 bits	24 bits	32 bits	48 bits
LSH	–	0.0064	0.0096	0.0132	0.0201	0.0077	0.0117	0.0147	0.0207	0.0158	0.0199	0.0221	0.0279
SH	–	0.0258	0.0437	0.0470	0.0660	0.0258	0.0437	0.0549	0.0660	0.0410	0.0480	0.0494	0.0532
ITQ	–	0.0351	0.0591	0.0668	0.0782	0.0306	0.0569	0.0711	0.0866	0.0559	0.0748	0.0831	0.0939
KSH	–	0.0396	0.0645	0.0768	0.0915	0.0353	0.0006	0.0923	0.1096	0.0804	0.0954	0.1040	0.1099
SDH	–	0.0327	0.0637	0.0814	0.0945	0.0384	0.0659	0.0868	0.1085	0.0719	0.1048	0.1167	0.1295
DPSH	ResNet50	0.0159	0.0225	0.0241	0.0380	0.0375	0.0541	0.0731	0.0931	0.0795	0.1059	0.1370	0.2025
DSH	ResNet50	0.0139	0.0225	0.0304	0.0392	0.0537	0.0786	0.0970	0.1119	0.1225	0.2392	0.2643	0.2961
HashNet	ResNet50	0.0157	0.0242	0.0276	0.0351	0.0726	0.1157	0.1284	0.1568	0.2186	0.3222	0.3515	0.4109
ADSH	ResNet50	0.0124	0.0998	0.1782	0.3041	0.0838	0.2460	0.3679	0.5285	0.0296	0.0499	0.1605	0.4835
Ours	VGG-16	**0.0940**	**0.2619**	**0.3419**	**0.4093**	**0.2974**	**0.5525**	**0.6058**	**0.6674**	**0.4177**	**0.5896**	**0.6284**	**0.6666**

5 Conclusion

In this paper, we presented a piecewise hashing method for the novel fine-grained hashing task. One of the key contributions was the local stream with piecewise part-level attention on bilinear representations to capture the discriminative cues for effectively representing fine-grained parts. Besides, our proposed multi-task training strategy can decrease the training and inference time while concurrently learned several hash functions and improving the search accuracy. Experimental results on diverse fine-grained datasets and the generic dataset showed the superiority of our method. In the future, we would like to explore novel fine-grained hashing methods under the unsupervised setting.

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. Commun. ACM **51**(1), 117 (2008)
2. Andoni, A., Razenshteyn, I.: Optimal data-dependent hashing for approximate near neighbors. In: STOC, pp. 793–801 (2015)
3. Bossard, L., Guillaumin, M., Van Gool, L.: Food-101 – mining discriminative components with random forests. In: Fleet, D., Pajdla, T., Schiele, B., Tuytelaars, T. (eds.) ECCV 2014. LNCS, vol. 8694, pp. 446–461. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10599-4_29

4. Cao, Z., Long, M., Wang, J., Yu, P.S.: HashNet: deep learning to hash by continuation. In: ICCV, pp. 5608–5617 (2017)
5. Gebru, T., Krause, J., Wang, Y., Chen, D., Deng, J., Fei-Fei, L.: Fine-grained car detection for visual census estimation. In: AAAI, pp. 4502–4508 (2017)
6. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: VLDB, pp. 518–529 (1999)
7. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. TPAMI **35**(12), 2916–2929 (2012)
8. Horn, G.V., Branson, S., Farrell, R., Haber, S.: Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In: CVPR, pp. 595–604 (2015)
9. Hou, S., Feng, Y., Wang, Z.: VegFru: a domain-specific dataset for fine-grained visual categorization. In: ICCV, pp. 541–549 (2017)
10. Jiang, Q.Y., Li, W.J.: Asymmetric deep supervised hashing. In: AAAI, pp. 3342–3349 (2018)
11. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Technical report, Citeseer (2009)
12. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: NeurIPS, pp. 1097–1105 (2012)
13. Li, W.J., Wang, S., Kang, W.C.: Feature learning based deep supervised hashing with pairwise labels. arXiv preprint arXiv:1511.03855 (2015)
14. Lin, T.Y., RoyChowdhury, A., Maji, S.: Bilinear convolutional neural networks for fine-grained visual recognition. TPAMI **40**(6), 1309–1322 (2018)
15. Liu, H., Wang, R., Shan, S., Chen, X.: Deep supervised hashing for fast image retrieval. In: CVPR, pp. 2064–2072 (2016)
16. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: CVPR, pp. 2074–2081 (2012)
17. Maji, S., Kannala, J., Rahtu, E., Blaschko, M., Vedaldi, A.: Fine-grained visual classification of aircraft. Technical report (2013)
18. Mu, Y., Wright, J., Chang, S.-F.: Accelerated large scale optimization by concomitant hashing. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012. LNCS, vol. 7572, pp. 414–427. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33718-5_30
19. Shen, F., Shen, C., Liu, W., Tao Shen, H.: Supervised discrete hashing. In: CVPR, pp. 37–45 (2015)
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
21. Wah, C., Branson, S., Welinder, P., Pietro Perona, S.B.: The Caltech-UCSD Birds-200-2011 Dataset. Technical report CNS-TR-2011-001, California Institute of Technology (2011)
22. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: CVPR, pp. 3424–3431 (2010)
23. Wang, Y., Song, R., Wei, X.S., Zhang, L.: An adversarial domain adaptation network for cross-domain fine-grained recognition. In: WACV, pp. 1228–1236 (2020)
24. Wei, X.S., Cui, Q., Yang, L., Wang, P., Liu, L.: RPC: a large-scale retail product checkout dataset. arXiv preprint arXiv:1901.07249 (2019)
25. Wei, X.S., Luo, J.H., Wu, J., Zhou, Z.H.: Selective convolutional descriptor aggregation for fine-grained image retrieval. TIP **26**(6), 2868–2881 (2017)
26. Wei, X.S., Wu, J., Cui, Q.: Deep learning for fine-grained image analysis: a survey. arXiv preprint arXiv:1907.03069 (2019)

27. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NeurIPS, pp. 1753–1760 (2009)
28. Xie, L., Wang, J., Zhang, B., Tian, Q.: Fine-grained image search. TMM **17**(5), 636–647 (2015)
29. Zheng, X., Ji, R., Sun, X., Wu, Y., Huang, F., Yang, Y.: Centralized ranking loss with weakly supervised localization for fine-grained object retrieval. In: IJCAI, pp. 1226–1233 (2018)
30. Zheng, X., Ji, R., Sun, X., Zhang, B., Wu, Y., Huang, F.: Towards optimal fine grained retrieval via decorrelated centralized loss with normalize-scale layer. In: AAAI, pp. 9291–9298 (2019)