DEEP SELF-TAUGHT GRAPH EMBEDDING HASHING WITH PSEUDO LABELS FOR IMAGE RETRIEVAL

Yu Liu[*], *Yangtao Wang*[*], *Jingkuan Song*[†], *Chan Guo*[*], *Ke Zhou*[*]✉ *and Zhili Xiao*[‡]

[*]Huazhong University of Science and Technology; [†]University of Electronic Science and Technology of China; [‡]Tencent

{liu_yu, ytwbruce, M201772906, k.zhou}@hust.edu.cn; jingkuan.song@gmail.com; tomxiao@tencent.com

ABSTRACT

It has always been a tricky task to generate image hashing function via deep learning without labels and allocate the relative distance between data through their features. Existing methods can complete this task and prevent the overfitting problem using shallow graph embedding technique. However, they only capture the first-order proximity. To address this problem, we design DSTGeH, a deep self-taught graph embedding hashing framework which learns hash function without labels for image retrieval. DSTGeH introduces deep graph embedding means to capture more complex topological relationships (the second-order proximity) on the graph and maps these relationships into pseudo labels, which enables an end-to-end hash model and helps recognize the samples outside the graph. We present the ablation studies and compare DSTGeH with the state-of-the-art label-free hashing algorithms. Extensive experiments show DSTGeH can achieve the best performances and produce an overwhelming advantage on multi-object datasets.

***Index Terms*—** Deep hashing, graph embedding, second-order proximity, image retrieval

1. INTRODUCTION

Hashing is widely used in large-scale retrieval owing to its efficiency for storage and matching [1, 2]. With the rapid development of deep learning, image hashing based on convolution neural network (CNN) [3] for feature extraction has become the mainstream of hashing methods. However, the dependence of label information in deep learning training restricts the practicability of deep hash algorithm, because elaborately collecting such labels is labor-intensive and no label set can cover the diversity of all classifications. To address this problem, unsupervised deep hashing methods are proposed. However, these algorithms, most of which are based on AutoEncoder (AE) [4], bring overfitting problems and thus suffer from severe performance degradation [5].

Another label-free method uses data features to construct a graph by means of graph embedding (GE). GE approaches in hashing focus on mapping the topological structure of a graph into Hamming space. GE takes the relationships between data into consideration, and map these relationships into compact binary expression, which solves the overfitting problem brought by AE algorithms. As a representative GE algorithm, Spectral hashing (SH) [6] utilizes Laplacian Eigenmaps (LE) algorithms [6] to executes minimum graph cut operation, which allocates the distance between data from a global perspective. However, these shallow algorithms can only capture the first-order proximity on the graph, thus severely reducing the accuracy of distance allocation.

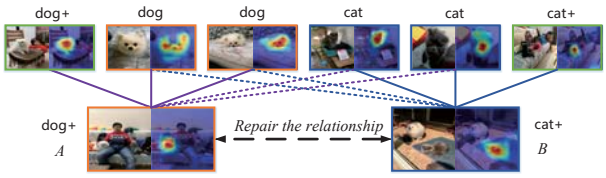


Fig. 1: The second-order proximity can construct (repair) the relationships between images (A and B) whose features are associated with those same or similar neighbors.

With GE technologies developing, researchers have paid more attention to the second-order (global) proximity mapping based on adjacency matrix [7, 8]. The second-order proximity can enhance the expression of relationships between data by exploring deeper topologies and repair the semantic bias. As shown in Fig. 1, there produces semantic bias if a pre-trained model that has only learned single-object image (cat or dog) is used to extract semantics of multi-object image (cat and dog), resulting that two semantically similar images are not approximately correlated. Fortunately, the second-order proximity is able to repair this relationship by associating the features of these two images with that of the same image by a similar distance. In addition, there is a drawback using the GE algorithms. Because the mapping must be completed on a graph, we have to reconstruct the graph repeatedly whenever a new data comes, which will result in unacceptable computational overhead. Considering that the best end-to-end model [9] still benefits from label information, we treat the output of GE as pseudo labels to learn an end-to-end hash model. This self-taught method combines the merits of transfer learning, which can not only reduce

✉Corresponding author: Ke Zhou. This work is supported by the Innovation Group Project of the National Natural Science Foundation of China No.61821003 and the National Natural Science Foundation of China No.61902135.

978-1-7281-1331-9/20/$31.00 ©2020 IEEE

the GE's dependence on uniform distribution of data but also avoid the overhead of reconstructing the graph for new data. Self-taught hashing (STH) [10] and DSTH [11], which are respectively shallow and deep self-taught GE hashing methods, have achieved good results, but they both neglect learning the second-order proximity and can no longer further promote the accuracy of hash codes.

In view of above thoughts, to promote the precision and efficient of label-free image hashing method, we integrate the means of mining second-order proximity and propose a new deep self-taught graph embedding hashing (DSTGeH) framework with pseudo labels for image retrieval. Our method consists of pseudo label generating stage and hash function learning stage. At the first stage, we calculate feature adjacency matrix which reflects the second-order proximity of the corresponding node (feature), and we embed each row to hash code by AE. At the second stage, we adopt CNN to learn the hash function with hash codes as pseudo labels. The most essential advantage of DSTGeH is to deeply mine relationship and precisely allocate distance between samples because the second-order proximity repairs the semantic bias from the pre-trained model and the hash function is learnt by pseudo labels which express the difference of distance between diverse (categories) data. Extensive experiments show DSTGeH not only performs well on single-object datasets but also achieves state-of-the-art retrieval results on multi-object datasets.

2. DEEP SELF-TAUGHT HASHING WITH GRAPH EMBEDDING (DSTGEH)

The proposed deep self-taught graph embedding hashing (DSTGeH) framework is composed of pseudo label generating stage and hash function learning stage. In pseudo label generating stage, we take both the (sampled) image data and pre-trained data as input, then use the pre-trained model to extract features from these data. Next, different from previous works, we construct a graph according to the Cosine distance between these features as well as calculate its adjacency matrix (each element is represented by a Cosine distance). Because each row of the matrix reflects the second-order proximity of the corresponding node (feature), we take each row as input to learn hash code by AE. Our hash codes (used as pseudo labels in the next stage) which preserve the second-order proximity contains more accurate semantic expression of the topological information compared with previous works. In addition, we completely get rid of setting k-NN that highly relies on human experience, and no longer encounter the huge memory overhead problem caused by large Laplacian matrix. In the next hash function learning stage, we adopt CNN to learn the hash function by using the data in first stage as input and their corresponding pseudo labels. Note that the pseudo labels will guide the model to exactly extract features that express semantics based on different distances between diverse categories. The framework of our DSTGeH is shown in Fig. 2, and we introduce the details below.

2.1. Pseudo Label Generating Stage

In this stage, we generate hash codes (used as pseudo labels) with deep features and combine a GE method which captures both the first-order and second-order proximity. We adopt a deep model (i.e., GoogleNet Inception v4) pre-trained on ImageNet to extract features for both the (sampled) image data and pre-trained data (i.e., sampled ImageNet data) from the last pooling layer of the model. Given that, according to the pre-trained deep model, we acquire N z-dimensional feature vectors $\mathcal{FV} = \{v_1, v_2, \ldots, v_N\}$ where $v_i \in \mathbb{R}^z$ for $i = 1, 2, \ldots, N$, and then construct a graph $\mathcal{G} = (\mathcal{FV}, \mathcal{SE})$, where $\mathcal{FV}$ represents N nodes and $\mathcal{SE} = \{e_{i,j}\}_{i,j=1}^{N}$ $(i, j \in \mathbb{Z}^+)$ represents the edges. Each edge $e_{i,j}$ is associated with a weight $u_{i,j}$ defined as

$$u_{i,j} = \frac{\vec{v_i} \cdot \vec{v_j}}{\|v_i\| \cdot \|v_j\|}. \tag{1}$$

We obtain $\mathcal{G}$'s adjacency matrix U which consists of N elements $u_1, u_2, ...u_N$. Each element $u_i = \{u_{i,j}\}_{j=1}^{N}$ expresses the semantic distances from other nodes (images). Learning the latent representation of all data from the adjacency matrix can reflect the second-order proximity. With U, we use Stacked AutoEncoder (SAE) [4] model to embed this topological structure to preserve the second-proximity. SAE is a stack combination of multiple AEs to complete the encoding. The implementation process is that the training result of the top AE is used as input to the next AE, and this process is repeated until the output of previous AE is input to the bottom AE of the stack. The reason why we choose SAE is that compared with other unsupervised algorithms, it has better effect in learning from "hierarchical" structure features. In practice, we train the parameters by the style of SAE with reconstruction errors. Then we fine-tune the entire stacked network with three different loss functions, and complete the training to update the whole network by each iteration. Given the input u_i and M hidden layers, the output for each layer are shown as follows:

$$f_i^{(m)} = \begin{cases} \varphi(W^{(m)} u_i + b^{(m)}) & m = 1, \\ \varphi(W^{(m)} f_i^{(m-1)} + b^{(m)}) & m = 2, ..., M. \end{cases} \tag{2}$$

where $f_i^{(m)}$, $W^{(m)}$ and $b^{(m)}$ denote output, weight and bias of m-th layer respectively, and $\varphi = Tanh(\varrho\chi)$ is the activation function, where $\varrho \in (0, 1]$ and χ is an input for the activation function. When $m = M$, we can acquire g-dimensional $\widetilde{h_i}$ as the result of encoder. Next, we generate the output $\widehat{u_i}$ as the result of reconstruction space by reversing the above processing. The loss function $\mathcal{L}_{so}$ for second-order proximity of reconstruction error is shown as follows:

$$\mathcal{L}_{so} = \mu \sum_{i=1}^{N} \|u_i - \widehat{u_i}\|_2^2 + (1 - \mu)\mathcal{L}_{soh}, \tag{3}$$

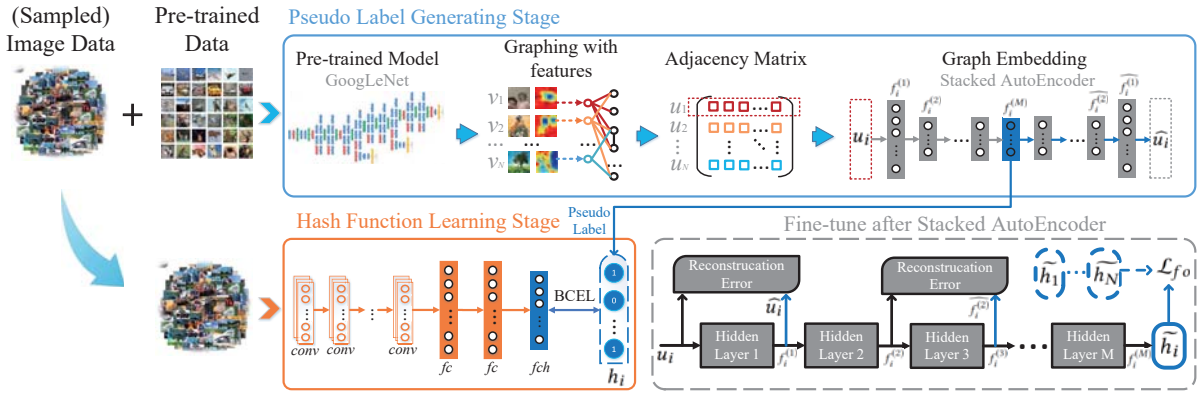


Fig. 2: DSTGeH framework. Our proposed method contains two stages: pseudo label generating stage (blue) and hash function learning stage (orange). We use Stacked AutoEncoder to map each row of adjacency matrix and then fine-tune the entire network with different loss functions (gray), where the reconstruction error represents learning the second-order proximity and $\mathcal{L}_{fo}$ (see Equation 5) represents learning the first-order proximity.

where $\mu \in (0, 1]$ and $\mathcal{L}_{soh}$ represents the sum of reconstruction loss of each hidden layer. Note that all nodes are fully connected according to their original distances, which avoids setting artificial threshold to determine the relationships between nodes. However, this may lead to overfitting (SAE may learn and map all nodes to the same data). To overcome this problem, we select part of hidden layers to learn the reconstruction error. Specifically, we only collect the reconstruction error every other hidden layer.

Given $\widehat{f_i^{(m)}}$ that represents the decoded data of m-th layer, $\mathcal{L}_{soh}$ is shown as follows:

$$\begin{aligned} \mathcal{L}_{soh} &= \sum_{i=1}^{N} \sum_{m=2\times t}^{M} \kappa_t \| f_i^{(m)} - \widehat{f_i^{(m)}} \|_2^2, \\ s.t. \quad & \frac{\kappa_t}{\kappa_{t+1}} = 2, \quad \sum \kappa_t = 1, \end{aligned} \tag{4}$$

where $t \in \mathbb{Z}^+$. The minimum of $\mathcal{L}_{so}$ can smoothly capture the data manifolds and preserve the similarity between data. In addition to using the adjacency matrix to learn the second-order proximity to obtain the distances between different categories data, we also hope to learn the first-order proximity of directly connected nodes (achieved by DSTH using LE) to ensure the classification accuracy of the same categories data. The loss function $\mathcal{L}_{fo}$ for this goal is defined as follows:

$$\mathcal{L}_{fo} = \sum_{i,j=1}^{N} u_{i,j} \| \widetilde{h_i} - \widetilde{h_j} \|_2^2. \tag{5}$$

The implementation of $\mathcal{L}_{fo}$ refers to LE, which makes those similar nodes closer in the embedding space. Finally, in order to prevent the overfitting problem, we design the regularizer term $\mathcal{L}_r$ as follows:

$$\mathcal{L}_r = \frac{1}{2} \sum_{m=1}^{M} \| W^{(m)} \|_F^2 + \| \widehat{W^{(m)}} \|_F^2. \tag{6}$$

Based on above design, the terminal loss function $\mathcal{L}$ is:

$$\mathcal{L} = \alpha \mathcal{L}_{so} + \beta \mathcal{L}_{fo} + \gamma \mathcal{L}_r, \tag{7}$$

where α, β and γ are hyper-parameters that satisfy $\alpha + \beta + \gamma = 1$. When $\mathcal{L}$ reaches convergence, we convert the g-dimensional real-valued vector $\widetilde{h_1}, \widetilde{h_2}, ..., \widetilde{h_n}$ into binary codes according to the threshold. We set $\widetilde{h_i^p}$ to denote element of p-th ($p \in \mathbb{Z}^+$) bit of $\widetilde{h_i}$. The hash label as binary value of h_i^p is:

$$h_i^p = \begin{cases} 1 & \widetilde{h_i^p} \geqslant 0, \\ 0 & otherwise. \end{cases} \tag{8}$$

2.2. Hash Function Learning Stage

The primary work of this stage is to learn and get the hash function model based on pseudo labels obtained from the first step. We employ CNN to train the (sampled) image data in the first stage with hash labels. Because the pseudo label is a vector that contains multiple $\mathbb{1}$ element, this hash function learning process is actually a multi-label classification task. We adopt the Binary Cross Entropy Loss (BCEL) to implement multi-label approximating since it is a common means for this task.

The whole processing is shown in the orange frame of Fig. 2. Formally, we set a function $f_h : \mathbb{R}^I \rightarrow \mathbb{R}^O$, where I is the input set which has $n \in N$ images, O is the output set and x presents an input vector (image).

$$\begin{aligned} f_h^{(1)}(x) &= \sigma(W^{(1)} x + b^{(1)}), \\ f_h^{(q)}(x) &= \sigma(W^{(q)} f^{(q-1)}(x) + b^{(q)}), \end{aligned} \tag{9}$$

where $q = 2, 3, ..., Q$ (Q is number of CNN layers) and $\sigma(*)$ is Leakly ReLU with BN function. The Leakly ReLU $\Psi(*)$ is calculated as follows:

$$\Psi(x) = \begin{cases} x & x > 0, \\ \lambda x & x \leq 0. \end{cases} \tag{10}$$

where $\lambda \in [0, 1)$. In the last layer, we use a full connection (FC) layer with a $sigmoid$ function to fit the g-dimensional binary label, where $sigmoid(x) = \frac{1}{1+exp(-x)}$. Given that $\widetilde{H_i}$ denote the output vector for the i-th image, and the p-th value in $\widetilde{H_i}$ is $\widetilde{H_i^p}$, we use BCEL $\mathcal{L}_h$ to train the hash function. The $\mathcal{L}_h(h_i, \widetilde{H_i})$ is computed as:

$$\mathcal{L}_h(h_i, \widetilde{H_i}) = -\sum_{p=1}^{g} h_i^p log(\widetilde{H_i^p}) + (1 - h_i^p) log(1 - \widetilde{H_i^p}). \quad (11)$$

At last, for the i-th image, the p-th bit in hash code as binary value of H_i^p is:

$$H_i^p = \begin{cases} 1 & \widetilde{H_i^p} \geqslant 0.5, \\ 0 & otherwise. \end{cases} \quad (12)$$

3. EXPERIMENTS

In this section, we evaluate our framework and conduct extensive experiments. Our framework is open-sourced and publicly available on Github [1].

3.1. Datasets and Settings

CIFAR-10 [12] consists of 60,000 single object images which are labeled with 10 classes. There are 5,000 training images and 1,000 test images in each class.

MS-COCO [13] is a popular multiple object dataset for image recognition, segmentation and captioning, which contains 118,287 training images, 40,504 validation images and 40,775 test images, where each image is labeled by some of the 80 semantic concepts.

FLICKR25K [14] is a collection of 25,000 multi-label images belonging to 24 unique provided labels, and each image is annotated by 4.7 labels on average. We randomly select 2000 images as the test set. The remaining images are used as the retrieval images, where we randomly select 10,000 images as the training set.

Evaluation metric: Similar to many other works [11], the retrieval performance is evaluated by mean Average Precision (mAP), Precision@Recall (PR) and F_1 score. On each of the three datasets, we calculate mAP@20% and PR with each category data, and we list their average values.

Parameter settings: We choose GoogLeNet inception V4 [15] (pre-trained on ImageNet) to extract features for matrix construction. We randomly select 50 images from each class in ImageNet to constitute 50,000 pre-trained data. Besides, we stipulate that (sampled) image data also consists of 50,000 images. Referring to the parameter settings in SDNE [7], we set $\alpha = 0.9$, $\beta = 0.099$ and $\gamma = 0.001$ in the loss function (see Equation (7)) of pseudo label generating. In addition, we set $\varrho = 0.2$ and $\mu = 0.7$ in Equation (3). Note that each row vector (including 100,000 elements) of the adjacency matrix is relatively large, so we arrange the row vector into matrix form ($10 \times 100 \times 100$). Meanwhile, in the hash function learning stage, we set $\lambda = 0.2$, and we apply ResNet-18 [16] with our customized full connection layer to complete the training.

[1] http://github.com/dl-ai/dstgeh.

Table 1: Performance (mAP) variances varying code length L and distance metric.

Dataset	Distance Metric	mAP@Hash Code Length L					
		16-bits	32-bits	48-bits	64-bits	96-bits	128-bits
CIFAR-10	Cosine	**0.2303**	**0.2818**	**0.3352**	**0.3408**	**0.3415**	**0.3447**
	Euclidean	0.2223	0.2702	0.3165	0.3252	0.3247	0.3269
MS-COCO	Cosine	0.3540	**0.3750**	**0.4102**	**0.4508**	**0.4703**	**0.4731**
	Euclidean	**0.3543**	0.3552	0.4008	0.4388	0.4607	0.4621

Table 2: Performance (PR) variance on 48-bits varying GE method, recall and loss function.

Dataset	GE Method (Loss Function)	Precision@Recall					F_1
		10%	20%	40%	60%	80%	
CIFAR-10	LE (BCEL)	0.3944	0.3302	0.2835	0.2397	0.2086	0.2828
	Word2Vec (BCEL)	0.5623	0.5037	0.4193	0.3025	0.2343	0.3260
	SDNE (BCEL)	**0.5919**	0.5578	0.4573	0.3604	0.2673	0.3487
	SDNE+SAE (BCEL)	0.5804	**0.5601**	**0.4848**	**0.3962**	**0.2885**	**0.3610**
	SDNE+SAE (MLML)	0.4311	0.3534	0.2942	0.2599	0.2217	0.2933
	SDNE+SAE (MLSML)	0.2319	0.2206	0.2032	0.1623	0.1303	0.1197
MS-COCO	LE (BCEL)	0.3120	0.2402	0.2001	0.1657	0.1304	0.2241
	Word2Vec (BCEL)	0.5509	0.5050	0.4001	0.3075	0.2265	0.3231
	SDNE (BCEL)	0.5932	0.5215	0.4188	0.3070	0.2173	0.3235
	SDNE+SAE (BCEL)	**0.6136**	**0.5223**	**0.4347**	**0.3552**	**0.2709**	**0.3458**
	SDNE+SAE (MLML)	0.3561	0.2437	0.2074	0.1763	0.1410	0.2322
	SDNE+SAE (MLSML)	0.2019	0.1882	0.1724	0.1592	0.1333	0.2098

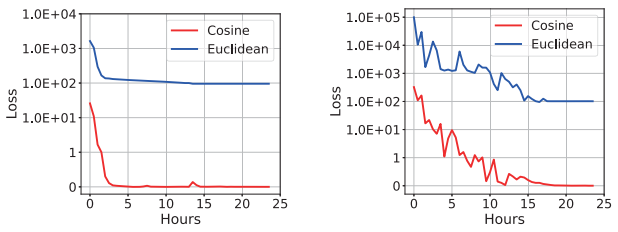
3.2. Ablation Study

In this part, we conduct ablation experiments to study the influence on our model with different components.

Influence of hash length L. We report the change of mAP with the increase of hash code length L from 16 to 128 on CIFAR-10 and MS-COCO in Table 1. We find that on CIFAR-10, mAP increases by 13.45% when L changes from 8 to 48, but it only increases by 0.71% when L changes from 48 to 128, while mAP shows a same trend on MS-COCO. Therefore, to balance the retrieval efficiency and effectiveness, we use $L = 48$ as the default setting.

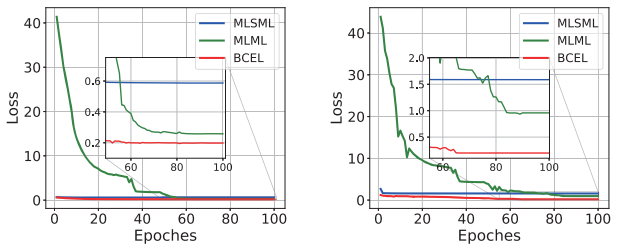
Influence of distance metric. We respectively demonstrate the effects of using Cosine distance and Euclidean distance on the performance of pseudo labels generating of our framework. Fig. 3 displays the convergence trend of above two methods, and we find the loss will converge to a much smaller value with a faster speed when using Cosine distance on CIFAR-10 and MS-COCO. As shown in Table 1 which lists the retrieval effects of two distance calculation ways on mAP, we can see Cosine distance brings higher mAP in most cases. Therefore, we adopt Cosine distance as the default setting to construct the adjacency matrix.

Influence of GE component. We compare GE components (LE, Word2Vec [17] and SDNE), and we give the results under conventional AE as well as SAE respectively when using SDNE as the component. In Table 2, we list the PR and F_1 score when applying 48-bits code length and Cosine distance metric. Obviously, SDNE leads to better performance than others, and this advantage becomes more obvious with the



(a) CIFAR-10 (b) MS-COCO

Fig. 3: GE convergence on CIFAR-10 and MS-COCO with different distance metrics in the first stage.



(a) CIFAR-10 (b) MS-COCO

Fig. 4: Hash function convergence on CIFAR-10 and MS-COCO with different loss functions in the second stage.

increase of recall rate. The F_1 scores respectively rises by 2.27% and 0.04% than that of Word2Vec. In addition, when we adopt SAE to run SDNE, our F_1 scores respectively rise by 1.23% and 2.23% than conventional AE. Therefore, we use "SDNE+SAE" to conduct the following experiments.

Influence of loss function. We compare BCEL with MultiLabelMarginLoss (MLML) and MultiLabelSoftMarginLoss (MLSML) which can fit multi-labels (the details for these loss functions are described in this url[2]). Fig. 4 presents the convergence trend of different loss functions, and BCEL converges to a smaller value with the fastest speed. We list the precision@recall and F_1 scores using above three loss functions. We find BCEL brings higher precision than others in most cases (and its F_1 score rises by 9.07% than MLML and 18.87% than MLSML), so BCEL becomes the best choice.

3.3. Comparison with the State-of-the-art Methods

In this part, we compare ours with the representative label-free or graph embedding hashing methods. These methods include PCA-ITQ (unsupervised quantization method) [18], DeepBit (unsupervised method using AE and quantization loss) [19], ARE (angle first-order proximity embedding) [20], DSTH (deep feature first-order proximity embedding) [11], ZSH (Word2Vec embedding with deep feature) [21], DHPL (unsupervised method with pseudo labels) [5], GraphBit (reinforcement learning method with graph) [1], DistillHash (unsupervised method with data pairs) [22]. On all datasets, we show the mAP at different code lengths and PR curve at 48 bits. It is worth noting that, to ensure the fairness, we will try our best to implement these algorithms without classification labels, which may result in different performances from the original one.

As shown in Fig. 5(a) and Fig. 5(d) which list the comparison results of mAP and PR on CIFAR-10, DSTGeH achieves the best performance in most case in terms of mAP and PR. Specially, it outperforms the runner up by 0.4%, 0.66%, 0.44% and 0.39% respectively (at 48, 64, 96 and 128 bits). Meanwhile, the precision is dominant when its recall rate reaches from 0.1 to 0.8, which is respectively higher than the runner up by 1.6%, 4%, 4.2%, 5% and 3% on 48 bits. This experiment shows DSTGeH can achieve a good result on single-object dataset, but it is not completely dominant compared with other candidates.

As shown in Fig. 5(b) and Fig. 5(e) which list the comparison results of mAP and PR on MSCOCO, DSTGeH achieves the best performance in all case in terms of mAP and PR. Specially, its mAP outperforms the runner up by 0.27%, 0.13%, 0.06%, 0.7%, 0.71% and 0.64% respectively. Meanwhile, the precision is respectively higher than the runner up by 1.3%, 1%, 1.9%, 1.2%, 0.9% and 1.6% on 48 bits in PR curve. This experiment shows DSTGeH can produce an overwhelming advantage on multi-object dataset.

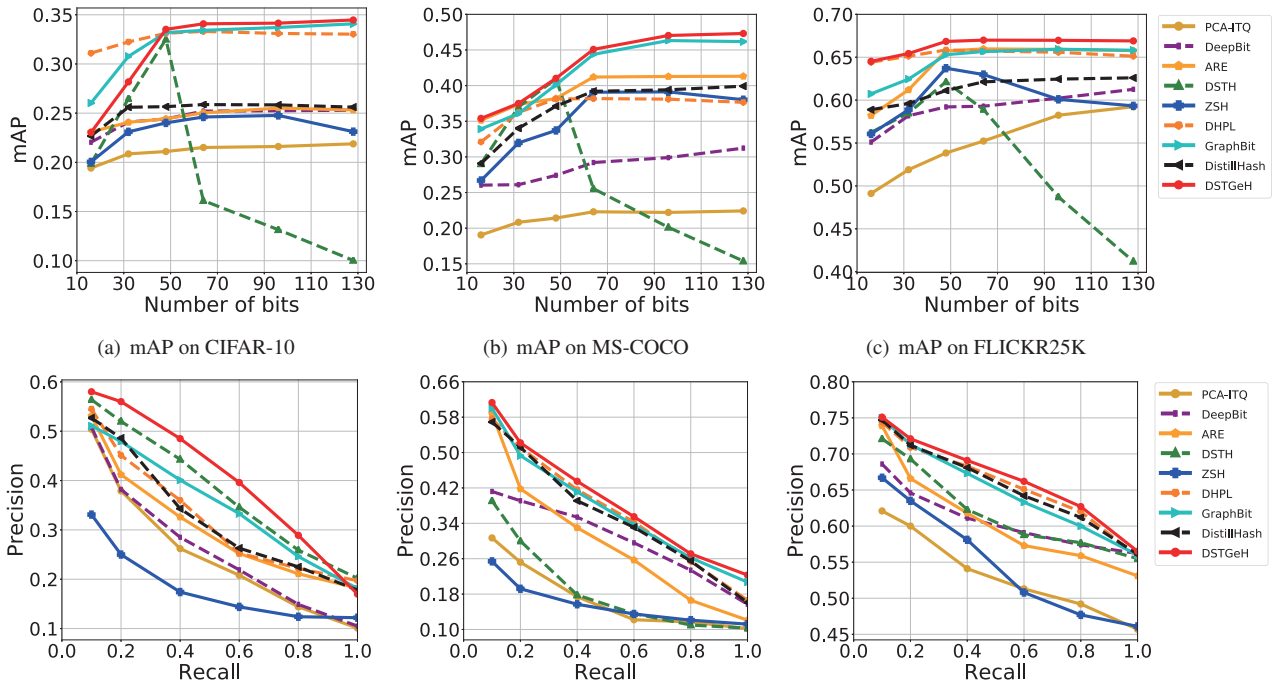
From above analysis, we find DSTGeH can bring a better retrieval result on multi-object MS-COCO than single-object CIFAR-10. Therefore, we choose FLlCKR25K which averagely owns more objects on each image to verify our observation. As shown in Fig. 5(c) and Fig. 5(f) which list the comparison results of mAP and PR on FLICKR25K, DSTGeH achieves the best performance in all case in terms of mAP and PR. Specially, its mAP outperforms the runner up by 0.11%, 0.31%, 0.99%, 0.103%, 0.104% and 0.109% respectively. Meanwhile, the precision is respectively higher than the runner up by 0.1%, 0.8%, 0.8%, 1.1%, 0.7% and 0.3% on 48 bits in PR curve. This experiment shows DSTGeH brings a stable and optimal effect on multi-object datasets.

According to above experiments, we are pleased and surprised to find DSTGeH can produce a good result on single-object dataset and an excellent result on multi-object datasets. We believe it is the second-proximity that can capture and build more reasonable semantic relationships between images. The reason why these relationships between multi-object images can produce more accurate retrieval result is that multi-object images own more rich semantic information and complex relationships between each other, and deep graph embedding can exactly express this complex topological structure and outperforms other algorithms. Both Fig. 5(c) and 5(f) verify the advantage of our method.

4. CONCLUSION

In this paper, we propose a deep self-taught graph embedding hashing (DSTGeH) framework for image retrieval. DSTGeH integrates the second-order proximity to learn both the local and global structure information between the input images

[2]https://pytorch.org/docs/master/search.html?q=loss&check_keywords=yes&area=default



(a) mAP on CIFAR-10 (b) mAP on MS-COCO (c) mAP on FLICKR25K

(d) PR on CIFAR-10 (e) PR on MS-COCO (f) PR on FLICKR25K

Fig. 5: mAP with different code lengths and 48-bits PR curve on CIFAR-10, MS-COCO and FLICKR25K.

using deep GE means. Meanwhile, it applies pseudo labels to achieve self-taught processing which helps model mapping new data without graph reconstruction. Extensive experiments manifest DSTGeH achieves the best performances on most evaluation metrics and produce an overwhelming advantage on multi-object datasets.

5. REFERENCES

[1] Yueqi Duan, Ziwei Wang, Jiwen Lu, Xudong Lin, and Jie Zhou, "Graphbit: Bitwise interaction mining via deep reinforcement learning," in *CVPR*, 2018, pp. 8270–8279.

[2] Jingkuan Song, Hervé Jégou, Cees Snoek, Qi Tian, and Nicu Sebe, "Guest editorial: Large-scale multimedia data retrieval, classification, and understanding," *TMM*, vol. 19, no. 9, pp. 1965–1967, 2017.

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1106–1114.

[4] Xiaofeng Yuan, Biao Huang, Yalin Wang, Chunhua Yang, and Weihua Gui, "Deep learning-based feature representation and its application for soft sensor modeling with variable-wise weighted SAE," *IEEE Trans. Industrial Informatics*, vol. 14, no. 7, pp. 3235–3243, 2018.

[5] Haofeng Zhang, Li Liu, Yang Long, and Ling Shao, "Unsupervised deep hashing with pseudo labels for scalable image retrieval," *TIP*, vol. 27, no. 4, pp. 1626–1638, 2018.

[6] Yair Weiss, Antonio Torralba, and Robert Fergus, "Spectral hashing," in *NIPS*, 2008, pp. 1753–1760.

[7] Daixin Wang, Peng Cui, and Wenwu Zhu, "Structural deep network embedding," in *KDD*. ACM, 2016, pp. 1225–1234.

[8] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji, "Large-scale learnable graph convolutional networks," in *KDD*, 2018, pp. 1416–1424.

[9] Tan Yu, Junsong Yuan, Chen Fang, and Hailin Jin, "Product quantization network for fast image retrieval," in *ECCV*, 2018, pp. 191–206.

[10] Dell Zhang, Jun Wang, Deng Cai, and Jinsong Lu, "Self-taught hashing for fast similarity search," in *SIGIR*, 2010, pp. 18–25.

[11] Liu Yu, Jingkuan Song, Zhou Ke, Lingyu Yan, Liu Li, Fuhao Zou, and Shao Ling, "Deep self-taught hashing for image retrieval," *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2229–2241, 2019.

[12] Alex Krizhevsky, Geoffrey Hinton, et al., "Learning multiple layers of features from tiny images," Tech. Rep., Citeseer, 2009.

[13] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick, "Microsoft COCO: common objects in context," in *ECCV*, 2014, pp. 740–755.

[14] Cheng Ma, Zhixiang Chen, Jiwen Lu, and Jie Zhou, "Rank-consistency multi-label deep hashing," in *ICME*, 2018, pp. 1–6.

[15] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017, pp. 4278–4284.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[17] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean, "Distributed representations of words and phrases and their compositionality," in *NIPS*, 2013, pp. 3111–3119.

[18] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin, "Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval," *TPAMI*, vol. 35, no. 12, pp. 2916–2929, 2013.

[19] Kevin Lin, Jiwen Lu, Chu-Song Chen, and Jie Zhou, "Learning compact binary descriptors with unsupervised deep neural networks," in *CVPR*, 2016, pp. 1183–1192.

[20] Mengqiu Hu, Yang Yang, Fumin Shen, Ning Xie, and Heng Tao Shen, "Hashing with angular reconstructive embeddings," *TIP*, vol. 27, no. 2, pp. 545–555, 2018.

[21] Yang Yang, Yadan Luo, Weilun Chen, Fumin Shen, Jie Shao, and Heng Tao Shen, "Zero-shot hashing via transferring supervised knowledge," in *MM*, 2016, pp. 1286–1295.

[22] Erkun Yang, Tongliang Liu, Cheng Deng, Wei Liu, and Dacheng Tao, "Distillhash: Unsupervised deep hashing by distilling data pairs," in *CVPR*, 2019, pp. 2946–2955.