# Zero-Shot Learning to Index on Semantic Trees for Scalable Image Retrieval

Shichao Kan, Yi Cen, Yigang Cen, Mladenovic Vladimir, Yang Li, *Graduate Student Member, IEEE*, and Zhihai He, *Fellow, IEEE*

***Abstract*— In this study, we develop a new approach, called *zero-shot learning to index on semantic trees* (LTI-ST), for efficient image indexing and scalable image retrieval. Our method learns to model the inherent correlation structure between visual representations using a binary semantic tree from training images which can be effectively transferred to new test images from unknown classes. Based on predicted correlation structure, we construct an efficient indexing scheme for the whole test image set. Unlike existing image index methods, our proposed LTI-ST method has the following two unique characteristics. First, it does not need to analyze the test images in the query database to construct the index structure. Instead, it is directly predicted by a network learnt from the training set. This zero-shot capability is critical for flexible, distributed, and scalable implementation and deployment of the image indexing and retrieval services at large scales. Second, unlike the existing distance-based index methods, our index structure is learnt using the LTI-ST deep neural network with binary encoding and decoding on a hierarchical semantic tree. Our extensive experimental results on benchmark datasets and ablation studies demonstrate that the proposed LTI-ST method outperforms existing index methods by a large margin while providing the above new capabilities which are highly desirable in practice.**

***Index Terms*— Zero-shot, learning to index, semantic tree, scalable image retrieval, approximated nearest neighbor search.**

## I. Introduction

DURING the past decades, we have witnessed the explosive growth of images and videos on the web, social networking platforms, surveillance networks, cloud servers, and mobile devices. It has become an urgent task to develop highly efficient image indexing and scalable retrieval methods. Images are encoded into features and nearest neighbor (NN) are then used to find the retrieval result. NN search in the high-dimensional feature space on large-scale datasets is very challenging. Exhaustive NN search is computationally prohibitive. How to develop efficient index schemes to achieve fast and scalable NN search has recently emerged as an important research topic. A large body of approximate NN search algorithms, including binary embedding [1], hashing codes [2], image clustering [3], and codebook-based quantization [4]–[7], have been developed to address issues of search accuracy, computational complexity, and memory cost. Based on these algorithms, one can construct an index structure to accelerate NN search. An inverted index then stores the list of images that lie in the proximity of each codeword, where we expect a set of similar images are indexed into the same group [4], [7]. During query, only clusters of images indexed in high-ranking clusters are examined for detailed search. Those low-ranking clusters of images are considered irrelevant to the query and will be excluded from the detailed search process.

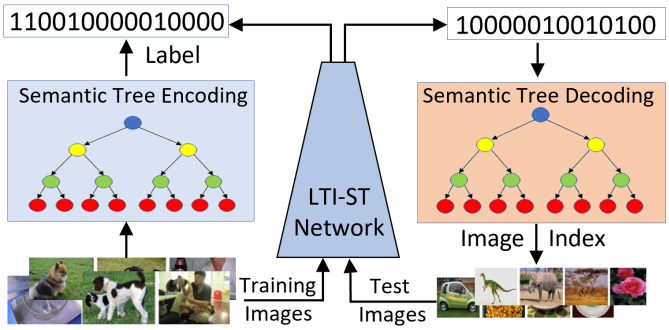



Fig. 1. Zero-shot learning to index on semantic trees (LTI-ST).

As pointed out in [8], most of existing index methods for approximate NN search [4], [7], [9] are based on distance between the original query sample and quantized codewords. This type of quantization process and distance-based index schemes will introduce information loss, especially for tree-based index methods [10], [11]. Although better quantization methods [6], [12], [13] can be used to alleviate this problem to some extent, the indexing and search complexity will increase dramatically. This information loss will result in significant performance degradation in image indexing and search. Recently, Chiu *et al.* [8] introduced a probability-based

Manuscript received March 20, 2020; revised July 26, 2020 and September 11, 2020; accepted October 17, 2020. Date of publication November 16, 2020; date of current version November 24, 2020. This work was supported in part by the National Key R&D Program of China under Grant 2019YFB2204200; in part by the National Natural Science Foundation of China under Grant 61872034, Grant 62011530042, and Grant 62062021; in part by the Beijing Municipal Natural Science Foundation under Grant 4202055; in part by the Natural Science Foundation of Guizhou Province under Grant [2019]1064; in part by the Science and Technology Program of Guangzhou under Grant 201804010271; and in part by the China Scholarship Council under Award 201907090007. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Husrev T. Sencar. *(Corresponding authors: Yigang Cen; Zhihai He.)*

Shichao Kan and Yigang Cen are with the Institute of Information Science, Beijing Jiaotong University, Beijing 100044, China, and also with the Beijing Key Laboratory of Advanced Information Science and Network Technology, Beijing 100044, China (e-mail: 16112062@bjtu.edu.cn; ygcen@bjtu.edu.cn).

Yi Cen is with the School of Information Engineering, Minzu University of China, Beijing 100081, China (e-mail: tsinge@muc.edu.cn).

Mladenovic Vladimir is with the Faculty of Technical Sciences, University of Kragujevac, 34000 Čačak, Serbia (e-mail: vladimir.mladenovic@ftn.kg.ac.rs).

Yang Li and Zhihai He are with the Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, MO 65211 USA (e-mail: yltb5@mail.missouri.edu; hezhi@missouri.edu).

Code project: https://github.com/kanshichao/LTI-ST

Digital Object Identifier 10.1109/TIP.2020.3036779

index scheme, called *Prob-RAW*, to address this issue and achieved significantly improved feature indexing and retrieval performance. This Prob-RAW method aims to learn neighborhood relationships embedded in the index space and nearest neighbor probabilities based on the query feedback. The clusters are ranked according to the NN probabilities instead of the Euclidean distances between features.

The Prob-RAW method in [8] represents one of the earliest efforts on developing probability-based or learning-based index schemes. However, we recognize that there are three important issues that have not been addressed. (1) *Depending on the testing dataset*. Both the distance-based index methods and the Prob-RAW method require the use of test images from the test database to construct the codebook or learn the index structure. This dependency has several major drawbacks. For example, when the test image database is updated with new images, these indexing schemes need to re-analyze the database to update the index structure. Can we develop an efficient index scheme which is learnt from a training set of images and directly applied to the test images? We refer to this new capability as *zero-shot learning to index*. This zero-shot capability is highly desirable in practice because it does not need to be updated with new images. It can be implemented in a distributed fashion and flexibly deployed at new service nodes. It requires the learning algorithm to capture the underlying structure of visual representation of images which can be effectively transferred from the training images to new test images. (2) *Constrained by the extracted image features*. Both the distance-based index methods and the Prob-RAW method learn the feature representation model and index model separately. Is it possible to learn them in an end-to-end manner? This end-to-end learning capability plays a crucial role in training a more flexible, more convenient, and better performance model. (3) *Non-scalable*. The Prob-RAW method learns to index the features onto a set of clusters. How to extend this learning to index scheme onto hierarchical trees to provide scalable image indexing? This tree-based indexing scheme is critical for efficient NN search when the number of clusters becomes large.

To address these important issues, in this work, we propose to develop a new method called *zero-shot learning to index on semantic trees* (LTI-ST) for highly efficient image indexing and scalable image retrieval. Our proposed LTI-ST method is motivated by the following observation: the fundamental objective of image indexing is to assign the same index to images of the same class. In other words, images which have strong semantic correlation with each other should be indexed into the same group. Therefore, the key task of image indexing is to characterize the correlation structure of images. As we know, images have a hierarchical correlation structure in the high-dimensional feature space, with the top layers showing significant difference between images and bottom layers showing subtle or fine-grained difference between neighboring images. Therefore, we propose to encode this correlation structure using a hierarchical tree, learn a network to map the input image into this code, predict this code for new test images using the network, and construct the correlation structure of test images in the feature space.

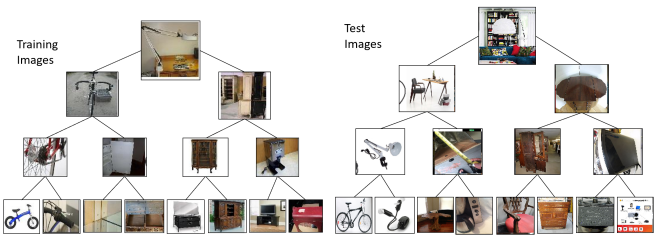



Fig. 2. Hierarchical correlation structure between images.

Specifically, as illustrated in Fig. 1, on the training set, with embedded learnt features, we construct a hierarchical tree of image clusters and encode each training image with a binary code mapped on this tree. This binary code will be used to generate labels for the training image. The image-label pairs will be used to train an LTI-ST deep neural network model. Once successfully trained, this LTI-ST network will be used to predict labels for each test image, which will be then decoded back onto the tree structure and construct the hierarchical clusters on test images and their indices. Our experimental results on benchmark datasets demonstrate that our new approach outperforms existing index methods while providing the zero-shot and scalable capabilities. Fig. 2 shows one example to demonstrate this idea. The left figure shows the hierarchical clustering of the training images and each image is encoded by its path information on this tree. The right figure shows the correlation structure of the new test images predicted by our LTI-ST network learnt from the training data.

The major contributions of this paper can be summarized as follows:

- We propose a learning-based image index scheme with embedded features based on binary encoding and decoding on semantic trees. With our framework, feature embedding and indexing model can be learnt in an end-to-end manner, which is different from other feature-based image index methods.
- Unlike existing index methods, our proposed LTI-ST index scheme is zero-shot and highly scalable, which does not require the analysis of test images in the query database, allowing fast, flexible, distributed, and scalable implementation and deployment of image indexing and retrieval services at large scales. Based on our method, there is no need for distance computation to obtain the index since it is directly predicted online by our deep neural network, and the predicted index value can be stored in the memory with the embedded features.
- Our extensive experimental results on benchmark datasets and ablation studies demonstrate that the proposed LTI-ST method outperforms existing index methods by a large margin while providing the above new capabilities which are highly desirable in practice.

The rest of this paper is organized as follows. Section II reviews the related work on image indexing. The proposed LTI-ST method is presented in Section III. Experimental results, performance comparison with state-of-the-art methods, and detailed ablation studies of our algorithm are provided in Section IV. Section V concludes the paper.

## II. Related Work

In the following, we review existing image indexing schemes related to our study, including codebook-based, tree-based, graph-based, and learning-based indexing schemes, and zero-shot learning methods.

### A. Codebook-Based Indexing Methods

Vector quantization is extensively used in codebook-based indexing methods, including the IVF (inverted file system) [7], [14], IVFADC (inverted file system with asymmetric distance computation) [9] and IMI (inverted multi index) [4], [15] methods developed in the literature. With codebook learning, they produce a set of representative centroid codewords in the high-dimensional feature space. Each feature to be indexed is quantized to the nearest codeword. These methods usually combine index and database compression.

Product quantization (PQ) [16] is one of the state-of-the-art methods for database compression. PQ is implemented by partitioning the feature vector of an image into several segments and encoding each segment with the corresponding codebook. Combining IVF and PQ, the IVFADC method [9] has achieved fast search in large-scale databases with billions of images. Babenko and Lempitsky proposed the Multi-D-ADC (multi-displacement-asymmetric distance computation) method [4] by combining IMI and PQ for fast retrieval in large-scale high-dimensional database. The performance of indexing in the Multi-D-ADC scheme has been further improved using global data rotation that minimizes correlations between subspaces [17]. Another major improvement has been achieved in [18] which introduces the Multi-LOPQ (multi-locally optimized product quantization) system that uses local PQ codebooks for displacement compression based on the IMI structure.

Embedding is another popular approach for database compression. Binary embedding [19]–[22] and feature embedding [23]–[26] are two major embedding methods. In general, the most compact data representation is provided by binary embedding. To realize binary embedding, the original data is first encoded into binary codes by a set of hash functions or learning methods, then the hamming distance between two binary codes can be calculated efficiently with hardware-supported machine instructions. Methods in [27]–[29] combine binary embedding and index schemes to achieve low storage and fast retrieval. Although, they can save time and memory, the retrieval performance is significantly degraded due to information loss.

### B. Tree-Based Indexing Methods

Tree-based indexing is widely used for scalable and fast deployment at large scales [10], [11]. The kd-tree [30] is one of the well-known approximate NN search algorithms, which is constructed by splitting data on the dimension with the highest variance. Later, Arya *et al.* [31] proposed an "error bound" approximate NN search method by controlling the error rate based on a variation of the kd-tree. Beis *et al.* [32] proposed a "time bound" approximate NN search method by limiting the time spent on the search. In this way, the kd-tree search is stopped early after examining a fixed number of leaf nodes. The time-constrained approximation method has been found to produce better performance than the error-constrained approximate NN search. Randomized kd-tree is proposed in [11] to speed up approximate NN search.

Another class of tree index algorithms decompose the space using various clustering methods. These index algorithms include the hierarchical k-means tree [33], the anchors hierarchy [34], the vptree [35], the cover tree [10] and the spill-tree [36]. Nister and Stewenius [37] proposed the vocabulary tree, which is searched by accessing a single leaf of a hierarchical k-means tree. Leibe *et al.* [38] proposed a ball-tree data structure using a mixed partitional-agglomerative clustering algorithm. Schindler *et al.* [39] proposed a new method to search the hierarchical k-means tree.

Both the above tree-based dimension-decomposition methods and space-decomposition methods have the following two major drawbacks. (1) They need a considerable amount of memory to store the huge index structure. (2) They will reduce the accuracy of the query results, especially with deep trees. To address these two issues, Muja and Lowe [13] developed the priority search k-means tree algorithm for effective matching in high-dimensional spaces. Houle and Nett [12] proposed the rank cover tree, which uses the ordinal ranks of the query distance to prune data points. Liu *et al.* [6] introduced an aggregating tree using residual vector quantization (RVQ) encoding and beam search. In this paper, we propose to decompose the feature space on semantic trees and time-constrained approximation criterion is used during the query stage. Our indexing algorithm is learning-based and the accuracy is significantly improved when compared to the above distance-based indexing algorithms, and there is no need to store the codebook in the memory.

### C. Graph-Based Indexing Methods

Graph-based indexing schemes aim to build greedy rooting navigation on a group of datasets to realize fast NN retrieval. Generally, the greedy rooting is built on approximate k-nearest neighbor (k-NN) graphs [40]–[46], called *proximity graph*. For a given proximity graph, the search starts at an one point and iteratively traverses the graph. At each step of the traversal, the algorithm examines the distances from a query to the neighbors of a current base node, and then selects one of the neighbors as the next base node that minimizes the distance. Examples of proximity graph algorithms include the navigable small world (NSW) [43], the hierarchical navigable small world (HNSW) [44], the navigating spreading-out graph (NSG) [45] and the satellite system graph (SSG) [46] algorithms. Our proposed method is complementary to the proximity graph. We aim to group large-scale dataset and online data into semantic tree groups, and the graph-based indexing scheme can be applied to each group of the semantic tree to reduce the number of distance calculations for online retrieval.

### D. Learning-Based Indexing Methods

The above codebook-based and tree-based indexing methods are all based on distance. They suffer from information

loss during codebook matching or quantization. Recently, Chiu *et al.* [8] introduced one of the first indexing scheme, called *Prob-RAW*, based on learning. Unlike previous distance-based methods, this new index method aims to learn the neighborhood relationships embedded in the index space and nearest neighbor probabilities based on the query feedback. The clusters are ranked according to the NN probabilities instead of the Euclidean distances of features. Compared to this Prob-RAW method, our LTI-ST index method has the following unique and important characteristics: (1) it is zero-shot thus does not need to use images from the query database. The Prob-RAW method needs to use queries in the database to learn the NN probability nonlinear mapping. (2) Based on our method, models of feature embedding and index can be learnt simultaneously. The Prob-RAW method needs to extract features first and then learn the index model. (3) Our model is learnt on the semantic trees with binary encoding and decoding, which is scalable for large-scale image retrieval.

### *E. Zero-Shot Learning*

The general purpose of zero-shot learning (ZSL) [47] is to learn a model to recognize the test images from unseen classes. In other words, the train classes and test classes are totally different. ZSL has been widely studied in image classification [48]–[50], video classification [51], hashing [52], image retrieval [53], [54], etc. Usually, extra auxiliary supervision signals of unseen classes, such as semantic word embeddings of seen and unseen class names and explicit attribute information, are used. This work represents one of the first efforts to study zero-shot image indexing, where the indexing algorithm is learnt from the training classes with known labels and applied to query images from unseen classes. This zero-shot capability allows us to develop an index scheme which does not need to access the query database, which is however required in existing indexing methods. Similar with zero-shot learning on retrieval task that aims to learn a transferable feature embedding model based on instances belong to the training seen classes [53], we learn a transferable index model from the training classes.

## III. The Proposed LTI-ST Method

In this section, we present our proposed LTI-ST method.

### *A. Method Overview*

Fig. 3 provides an overview of the proposed LTI-ST method. We first use a pre-trained deep neural network, called *encoder network*, to encode the input image and extract its high-dimensional feature. The feature vector is then embedded into a compact feature space using a learnt feature embedding network to reduce its feature dimension and increase its discriminative power. Based on the embedded features of training images, we perform hierarchical $k$-means clustering of these features to construct a binary tree of codewords. Based on this binary codeword tree, we encode each class of training images with the same class label by identifying the best matching codeword at each layer of the tree for each

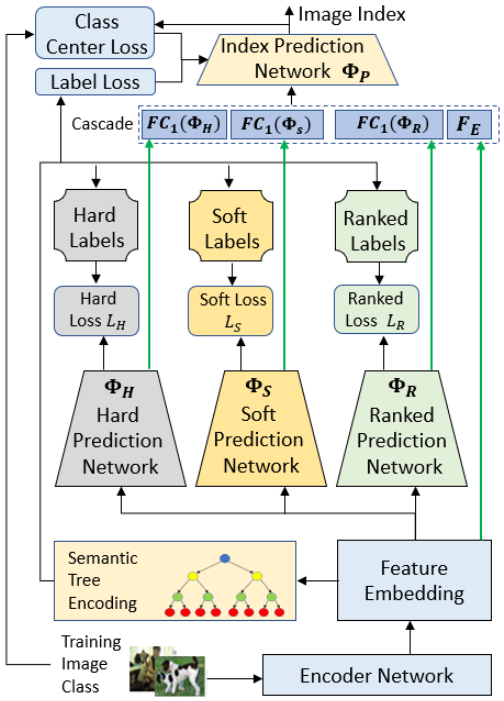



Fig. 3. The proposed learning to index on semantic tree (LTI-ST) framework (training).

image in the class. We then assemble histogram statistics of the best matching codewords for the whole class. This step of mapping the class of images with the same semantic label is referred to as *semantic tree encoding*.

To accurately characterize the correlation structures between images in the high-dimensional feature space, we study three different labeling schemes to convert this tree-based encoding into a label vector for the whole class of images: *hard*, *soft*, and *ranked* labeling schemes. From our experiments, we observe that these three labeling schemes are able to capture different aspects of the correlation structure between images. Based on these three labeling schemes, we will learn three different networks, *hard prediction network* $\Phi_H$, *soft prediction network* $\Phi_S$, and *ranked prediction network* $\Phi_R$ in Fig. 3, to predict these three types of labels. The corresponding loss functions are denoted by $L_H$, $L_S$, and $L_R$, respectively. In our proposed method, these three networks are jointly trained together in an end-to-end fashion. Specifically, we will concatenate the feature generated by these three networks (e.g., its *fc*-1 layer), denoted by $\mathbf{FC}_1(\Phi_H)$, $\mathbf{FC}_1(\Phi_S)$, and $\mathbf{FC}_1(\Phi_R)$, along with the embedded feature $\mathbf{F}_E$ that computed by the feature embedding network, to form a cascading feature for index prediction using the index prediction network $\Phi_P$. These three labels prediction networks, $\Phi_H$, $\Phi_S$, $\Phi_R$, and the index prediction network $\Phi_P$ will be jointly trained, which will be explained in more details in the following sections.

### *B. Semantic Tree Encoding and Sample Labeling*

The semantic tree encoding and labeling aim to generate the semantic tree labels for each class of training images. One important property of this semantic tree encoding is that images from semantically similar classes should have small

distance values between their codes. As illustrated in Fig. 4, we use GoogLeNet [55] or ResNet-50 [56] model that pre-trained on ImageNet and fine-tuned on our training set to encode each image into a feature vector. This GoogLeNet or ResNet-50 feature will be further embedded into a compact semantic feature so that image features of the same class are made closer to each other [25]. Based on this semantic feature, we perform hierarchical k-means clustering on the training image features to construct a binary codeword tree $\mathbf{T}$ with each codeword being the cluster center. Suppose that the tree $\mathbf{T}$ has $N_L + 1$ layers. At layer $l$, $0 \leq l \leq N_L$, we have a set of $2^l$ codewords:

$$\Omega^l = [\mathbf{c}_1^l, \mathbf{c}_2^l, \cdots, \mathbf{c}_{2^l}^l]. \tag{1}$$

Once this semantic tree is constructed, we are ready to map each class of training images onto this tree using the following semantic tree encoding scheme. Specifically, at layer $l$, we find the best matching codeword $\mathbf{c}_{k^*}^l$ from $\Omega_*^l$ for each image $x$ with feature embedding $\mathbf{e}(x)$:

$$\mathbf{c}_{k^*}^l = \arg \min_{\mathbf{c}_k^l \in \Omega_*^l} ||\mathbf{e}(x) - \mathbf{c}_k^l||_2. \tag{2}$$

Here, $\Omega_*^l$ is a subset of $\Omega^l$ whose parent nodes are the best matching codeword found in the previous layer. In this way, we can guarantee that the best matching codewords of all layers form a valid path on the tree [6]. We then define the binary vector $\mathbf{B}^l(x)$ with only 1 at position $k^*$ and zeros at other locations at the $l$-th layer. Let $\mathbf{G}_n$ be the $n$-th class of training images, $1 \leq n \leq N$. The semantic tree encoding vector for the whole class $\mathbf{G}_n$ is then defined as

$$\mathcal{Q}_n^l = \sum_{x \in \mathbf{G}_n} \mathbf{B}^l(x), \tag{3}$$

which is a histogram vector summarizing how many images from the class $\mathbf{G}_n$ are matched to each codeword at layer $l$, as illustrated by the histogram in Fig. 4. To minimize the impact of different class sizes, we perform the following normalization over all training classes:

$$\mathbf{Q}_n^l = \frac{\mathcal{Q}_n^l}{\sum_{i=1}^N \mathcal{Q}_i^l}, \tag{4}$$

Once the semantic tree encoding vector for each class of training images is obtained, we map it into three label vectors using the following three labels generation schemes. These three labeling schemes aim to capture different correlation structures between image samples. The **hard label** vector for $x$ at layer $l$, denoted by $\mathbf{h}^l(x)$, is a binary vector obtained by finding the maximum frequency number in $\mathbf{Q}_n^l$ and setting it to be 1 and the rest numbers to be zeros. The **soft label** vector for image $x$ at layer $l$, denoted by $\mathbf{s}^l(x)$, is obtained by normalizing the histogram vector $\mathbf{Q}_n^l$ into a probability vector. Unlike the hard and soft labels, the **ranked label** vector is generated by an image retrieval method. For each image $x$, we find the top $K$ best matches from the whole training set, denoted by $\Omega(x) = \{x_1', x_2', \cdots, x_K'\}$. We treat this set of retrieved images as one new class and follow Eq. (3) to compute its semantic tree encoding vector

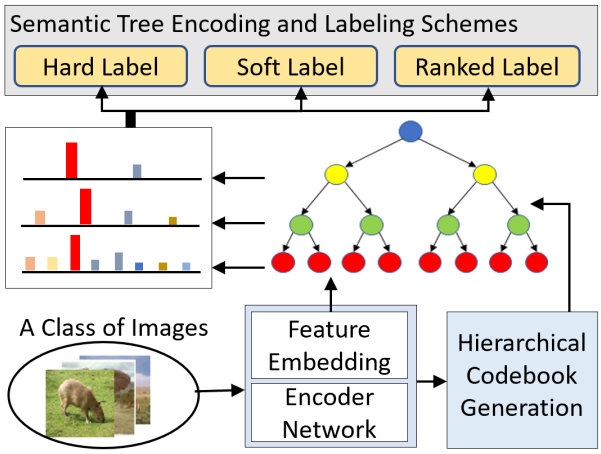



Fig. 4. Three different labeling schemes.

$$\mathcal{R}_x^l = \sum_{k=1}^K \mathbf{B}^l(x_k') \cdot w(k), \tag{5}$$

$w(k)$ is a weighting function. For example, in our experiments, we set $w(1)$ to be significantly larger than the remaining weights. Similar to Eq.(4), semantic tree normalization over all the training images is performed as follows:

$$\mathbf{R}_x^l = \frac{\mathcal{R}_x^l}{\sum_{i=1}^M \mathcal{R}_i^l}, \tag{6}$$

where $M$ is the number of training images. From our experimental results, we will see that these three labeling schemes capture different types of neighborhood relationship between the training images. Fusion of information or features generated by networks trained with these different labeling schemes will result in significantly improved indexing performance.

Once the labels for image $x$ at tree layer $l$ is obtained, the labels of all layers in the tree are then concatenated together to form the final hard, soft, and ranked labels for image $x$: $\mathbf{h}(x) = [\mathbf{h}^0(x), \mathbf{h}^1(x), \cdots, \mathbf{h}^{N_L}(x)]$, $\mathbf{s}(x) = [\mathbf{s}^0(x), \mathbf{s}^1(x), \cdots, \mathbf{s}^{N_L}(x)]$, and $\mathbf{r}(x) = [\mathbf{r}^0(x), \mathbf{r}^1(x), \cdots, \mathbf{r}^{N_L}(x)]$. It should be noted that, for the soft and hard labels, the whole class of training images have the same label vector. But, for the ranked labels, images of the same class may have slightly different label vectors. In our experiments, we observed that the ranked labels often yielding better performance than the other two labeling schemes for zero-shot learning to index.

### C. Learning the LTI-ST Networks

As discussed in previous sections, our proposed method learns the feature embedding network and index network simultaneously. In this section, we first explain how to learn the feature embedding network, and then we explain how to learn these three labeling prediction networks and the index prediction network.

*1) Learning the Feature Embedding Network:* There are a number of methods developed in the literature for effective learning of feature embedding [23]–[26]. They map

TABLE I

THE DETAILED STRUCTURE OF ALL THE NETWORKS IN LTI-ST WITH THE ENCODER NETWORK OF GOOGLENET (G) OR RESNET-50 (R)

| Networks | Feature Embedding network | Hard prediction network | | | Soft prediction network | | | Ranked prediction network | | | Index prediction network | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Layers | $fc$ | $fc$-1 | $fc$-2 | $fc$-3 | $fc$-1 | $fc$-2 | $fc$-3 | $fc$-1 | $fc$-2 | $fc$-3 | $fc$-1 | $fc$-2 |
| Num of Input Nodes | G: 2048, R: 1024 | $d$ | 512 | 512 | $d$ | 512 | 512 | $d$ | 512 | 512 | $d$+1536 | 512 |
| Num of Output Nodes | $d$ | 512 | 512 | $2^{N_L+1}-2$ | 512 | 512 | $2^{N_L+1}-2$ | 512 | 512 | $2^{N_L+1}-2$ | 512 | $2^{N_L+1}-2$ |
| Losses | $L_E$: MSL [25] | $L_H$ | | | $L_S$ | | | $L_R$ | | | $L_P+\lambda L_C$ | |

high dimensional feature to compact low dimensional feature embedding, as well as learning a distance or similarity metric. Following previous works of feature embedding, in this work, we adopt the state-of-the-art multi-similarity loss (MSL) [25] as the $L_E$ loss to learn the feature embedding network with GoogLeNet and ResNet-50 encoders.

Specifically, given a mini-batch of $M$ images $\{x_i, y_i\}_{i=1}^M$, in which $x_i$ is the $i$-th image, $y_i$ is the corresponding class label. The output of the embedding network are image features, denoted as $\mathbf{e}(x_i)$. Suppose that the features and their corresponding labels in a mini-batch are $\{\mathbf{e}(x_k), y_k\}_{k=1}^M$. Then, the similarity matrix between features of the current mini-batch can be computed, denoted as $\mathbf{S} = \{s_{ik}\}$, $1 \leq i \leq M$, $1 \leq k \leq M$. Here, $s_{ik}$ is the cosine similarity between two features. Using $\mathbf{S}$, we determine the set of positive pairs $\mathcal{P}$ and the set of hard negative pairs $\mathcal{N}$ based on their similarity scores as follows:

$$\begin{aligned} \mathcal{P} &= \{(i,k)|y_i = y_k\} \\ \mathcal{N} &= \{(i,k)|y_i \neq y_k \text{ and } s_{ik} > \sigma\}, \end{aligned} \tag{7}$$

where $\sigma$ is a similarity threshold for hard negative pairs, which is set as 0.3 in our experiments. Once the sets of positive and hard negative pairs $\mathcal{P}$ and $\mathcal{N}$ are identified, we define the loss for each sample $x_i$ in the mini-batch as follows

$$\begin{aligned} \mathcal{L}_E^i = \frac{1}{\lambda_P} \log[1 + \sum_{(i,k)\in\mathcal{P}} (e^{-\lambda_P(s_{ik}-\delta)})] \\ + \frac{1}{\lambda_N} \log[1 + \sum_{(i,k)\in\mathcal{N}} (e^{\lambda_N(s_{ik}-\delta)})], \end{aligned} \tag{8}$$

where $\delta$ is a margin threshold. According to [25], $\delta$ is set to 0.5, $\lambda_P$ is set to 2 for positive pairs, and $\lambda_N$ is set to 40 for hard negative pairs. Then, the overall loss of the current mini-batch is given by

$$\mathcal{L}_E = \frac{1}{M} \sum_{i=1}^{M} \mathcal{L}_E^i. \tag{9}$$

For more details, please refer to [25] and our released code.

*2) Learning the Labels Prediction Networks:* These three labels generation network share the same network structure and training process. In the following, we take the hard labels generation network as an example to explain the training of these three labels generation networks.

Given a training set with $M$ images $\{x_i, y_i, \mathbf{t}_i\}_{i=1}^M$, in which $x_i$ is the $i$-th image, $y_i$ is the corresponding class label, and $\mathbf{t}_i$ is the hard semantic tree labels $\mathbf{h}(x_i)$. The hard labels prediction network $\Phi_H$ is a multiple layer perception network. The input to the network is the feature embedding $\mathbf{e}(x_i)$ of image $x_i$. Let $\mathbf{v}_i = \Phi_H(x_i)$ be the network output. The loss function for this network, called *hard loss*, is given by the following cross entropy

$$L_H = \Lambda[\Phi_H(x_i), \mathbf{h}(x_i)], \tag{10}$$

where $\Lambda[\cdot,\cdot]$ represents the cross entropy. Similarly, we can compute the losses of $L_S$ and $L_R$ for the soft prediction network and ranked prediction network.

*3) Learning the Index Prediction Network:* For the index prediction network, its output is the semantic tree labels generated from the encoding process described in the previous section. We can choose the hard, soft, or the ranked labels as the ground truth labels. The input to the index prediction network is the concatenation of four feature vectors: $\mathbf{FC}_1(\Phi_H)$ from the hard prediction network, $\mathbf{FC}_1(\Phi_S)$ from the soft prediction network, $\mathbf{FC}_1(\Phi_R)$ from the ranked prediction network, and $\mathbf{F}_E = \mathbf{e}(x)$ from the feature embedding network for image $x$. We use the output of the first fully connected layer $fc$-1 of the labels generation network as the feature. In our design, the index prediction network aggregates the features and information obtained from three drastically different learning paths to enhance its prediction power. According to our experiments, it is much more accurate and robust than each individual labels prediction network and is able to obtain more accurate and consistent indexing results within each class. This is very important for zero-shot learning to index cross different databases of images.

To train the index prediction network $\Phi_P$, we use the labels loss $L_P$ which is the cross entropy between the network prediction $\Phi_P(x_i)$ and the semantic tree labels. The semantic tree labels can be set as one of the hard, soft, and ranked labels. To further enhance its semantic prediction power such that images of the same class have similar prediction outputs, we also introduce the following class center loss

$$L_C = \sum_{i=1}^{N} ||\Phi_P(x_i) - \Phi_P^c(x_i)||_2, \tag{11}$$

where $\Phi_P^c(x_i)$ is the average of network outputs for all images $x_i$ from the same class. The three labels generation networks and the index prediction network are jointly trained.

*4) Overall Loss and Network Structure:* The overall loss function is formulated as:

$$\mathcal{L} = L_E + L_H + L_S + L_R + L_P + \lambda L_C, \tag{12}$$

where $\lambda$ is the weight of $L_C$. We can see that the loss functions for three labels prediction networks, $L_H$, $L_S$, and $L_R$ are used to provide intermediate regulation between the labels prediction networks and index prediction network. They are able to improve the performance of the whole end-to-end network learning to achieve the desired outcome.

The detailed structures of all the networks are summarized in Table I, where $d$ is the dimension of feature embedding,

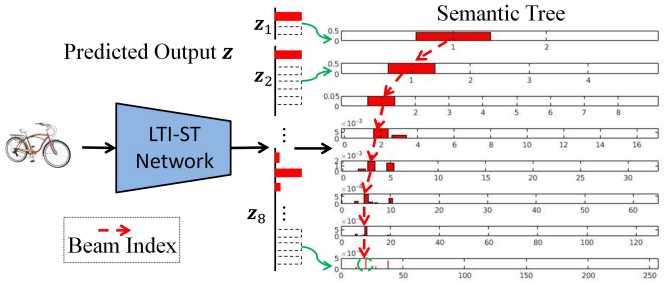



Fig. 5. Semantic tree decoding and indexing.

G represents GoogLeNet, and R represents ResNet-50. For the hard prediction network, soft prediction network, and ranked prediction network, there are one ReLU [57] activation layer and one dropout [58] layer (the drop ratio is 0.1) after $fc$-1 and $fc$-2 layers. For the index prediction network, there are one ReLU activation layer and one dropout layer (the drop ratio is 0.1) after the $fc$-1 layer.

### *D. Semantic Tree Decoding and Image Retrieval*

For scalable image retrieval with approximate NN search, we first need to group the test images in the test database into clusters organized by the index structure. In our proposed zero-shot LTI-ST method, we learn the LTI-ST network to directly predict the index of the input image, as discussed in the previous sections. This predicted outputs or labels have similar structure with the semantic tree encoding scheme. Our next step is to perform semantic tree decoding to map this predicted labels back onto a tree structure. This semantic tree decoding will achieve two important objectives. (1) The first objective is to construct the index structure of the test database and prepare it for image retrieval. Once the LTI-ST network is successfully learnt, it can be used to predict the tree-structured index for every image in the test database. With this index, all test images can be then organized and indexed. If new images need to be added into the database, we just run the LTI-ST network on these new images and add them to the corresponding clusters according to their indices. All existing images will not be touched and there is no need to update the whole database, which is however required in many existing methods [4], [8], [13]. (2) The second objective is to recover the tree index of the query image for hierarchical image retrieval. Given a query image, we first compute its index and identify the group or a subset of groups to be searched in the database, instead of searching the whole database whose complexity is prohibitively high [8].

As illustrated in Fig. 5, the semantic tree decoding reconstructs the index structure of the input image in an iterative manner on the binary tree. Suppose that our task is to find the top $r$ clusters to be searched in the query database. As we can see from Section III B, the predicted labels $\mathbf{h}'(x)$, $\mathbf{s}'(x)$, or $\mathbf{r}'(x)$ have an embedded tree structure with the labels from all tree layers being concatenated together. Let

$$\mathbf{z}(x) = \mathbf{z}_0 \uplus \mathbf{z}_1 \uplus \cdots \uplus \mathbf{z}_{N_L} \tag{13}$$

be the predicted labels with the concatenated structure of tree layers for image $x$, where $\mathbf{z}_l = [p_1, p_2, \cdots, p_{2^l}]$ is the predicted label vector at layer $l$. For $1 \leq j \leq 2^l$, $p_j$ is the probability of image $x$ belongs to the $j$-th node at layer $l$. Let

$$K_l = \arg\max_{p_j \in \mathbf{z}_l} \{p_j, r\} \tag{14}$$

be the corresponding locations of top $r$ values in vector $\mathbf{z}_l$, if the number of values in $\mathbf{z}_l$ less than $r$, $K_l = [I_1, I_2, \cdots, I_{2^l}]$, $\{I_1, I_2, \cdots, I_{2^l}\}$ represents indices on $\mathbf{z}_l$. The indices in $K_l$ are the selected subgroup of indices and tree nodes at layer $l$ for image retrieval. These $|K_l|$ nodes have $2 \times |K_l|$ children in the next layer, where $|K_l|$ represents the length of $K_l$, and $|K_l| \leq r$. Within these children locations, we find the top $r$ values in vector $\mathbf{z}_{l+1}$, as follows

$$K_{l+1} = \arg \max_{j \in \{2\times K_l - 1, 2\times K_l\}, p_j \in \mathbf{z}_{l+1}} \{p_j, r\}, \tag{15}$$

where $\{2 \times K_l - 1, 2 \times K_l\}$ are children indices on $\mathbf{z}_{l+1}$ that deduced based on the found indices $K_l$ on $\mathbf{z}_l$. The top-$r$ semantic tree decoding operates in an iterative manner for all the tree layers based on Eq. (15), and this process is repeated until the last layer $N_L$ is reached and the corresponding $r$ nodes in the last layer indexed by $\{I_1, I_2, \cdots, I_r\}$ are the selected groups for image retrieval. The semantic tree decoding algorithm is summarized in **Algorithm** 1.

It should be noted that the semantic tree decoding algorithm is implemented for the test images by setting $r = 1$, and for the query image $x_q$ by setting $r \geq 1$. The retrieval algorithm is summarized in **Algorithm** 2.

In **Algorithm** 2, the Euclidean distance is used to compute the similarity in the third step, where $\mathbf{d} = [\mathfrak{d}_1, \mathfrak{d}_2, \cdots, \mathfrak{d}_{r \times n_q}]$

**Algorithm 1** Semantic Tree Decoding Algorithm

**Input:** Predicted output $\mathbf{z}(x)$ with the learnt LTI-ST model for image $x$, maximum number of visited groups $r$, number of layers $N_L$.
**Output:** Indices $\{K_1, K_2, \cdots, K_{N_L}\}$.
$K_0 = [1]$
**for** $l$ in range(0,$N_L - 1$) **do**
  Separate out $\mathbf{z}_{l+1}$ from $\mathbf{z}(x)$.
  Compute $K_{l+1}$ based on Eq.(15).
**end for**

**Algorithm 2** Scalable Image Retrieval Algorithm

**Input:** Query image $x_q$, maximum number of visited groups $r$, maximum number of retrieved results $n_q$.
**Output:** Retrieved images.
1) Generate the feature embedding $\mathbf{e}(x_q)$ and predict the labels probability $\mathbf{z}(x_q)$ for $x_q$ using the trained LTI-ST model.
2) Find $r$ candidate groups in the leaf nodes (last layer) based on $\mathbf{z}(x_q)$ with the **Algorithm** 1.
3) Retrieve $n_q$ results in each candidate group of the test set with $\mathbf{e}(x_q)$, and return $r \cdot n_q$ candidate retrieval results and corresponding Euclidean distances $\mathbf{d}$.
4) Compute indices of top-$n_q$ smallest distances by Eq.(16), and return $n_q$ retrieved images indexed from the candidate retrieval results by these indices.

and each $\mathfrak{d}_j$ represents one Euclidean distance between $\mathbf{e}(x_q)$ and one candidate retrieval result. These candidate distances are used to the last step to obtain the query results, where the indices of $n_q$ smallest distances based on $\mathbf{d}$ is defined as follows

$$K = \arg\min_{\mathfrak{d}_j \in \mathbf{d}} \{\mathfrak{d}_j, n_q\}. \tag{16}$$

Moreover, existing feature encoding methods, such as PQ [16], hashing [1], and in-group approximate NN retrieval methods (e.g., HNSW [44]), can be used at the third step to achieve fast image retrieval. In our experiments, in order to provide a fair comparison with the inverted index methods, exhaustive search with feature embedding is used for the in-group retrieval. Furthermore, we also present experimental results with the HNSW method to decrease the number of visited examples in each group.

### E. Complexity and Storage Analysis

The computational complexity of **Algorithm** 2 is mainly from the first and third steps. At the first step, the complexity of feature embedding computation is determined by the convolutional neural network. Suppose that the dimension of feature embedding is $d$, the number of nodes in the first layer of hard prediction network, soft prediction network, and ranked prediction network are $\tau_h$, $\tau_s$ and $\tau_r$, respectively. The index prediction network has $\kappa$ $fc$ layers and each layer has $\tau_p$ nodes. Then, the complexity in this step is approximately $O(d \times (\tau_h + \tau_s + \tau_r) + (\tau_h + \tau_s + \tau_r + d) \times \tau_p + (\kappa - 2) \times \tau_p^2 + \tau_p \times (2^{(N_L+1)} - 1))$. At the second step, the complexity for codebook and tree generation are $O(2^{N_l})$ and $O(2 \times r \times (N_l - \log(r)))$, respectively. Because there are only comparison operations, this step has very low complexity. At the third step, when exhaustive retrieval is used for in-group retrieval, the computational complexity is approximately $O(u_{I_1} + u_{I_2} + \cdots + u_{I_r}))$, where $u_{I_r}$ is the number of examples in group $I_r$. When the HNSW method is used for in-group retrieval, the complexity is approximately $O(\log(u_{I_1}) + \log(u_{I_2}) + \cdots + \log(u_{I_r}))$. At the fourth step, we only need to sort $r \times n_q$ distance values, which has very low complexity.

The memory required by **Algorithm** 2 is mainly for the feature embedding. If the dimension of feature embedding is $d$ and the data type is single float, then we need $4d$ bytes memory to store a feature embedding and 4 bytes memory to store an index value. The total memory consumption for one image is $4 \times (d + 1)$ bytes. Because feature embedding is learnt based on the training dataset and $d$ is flexible, thus we can control memory consumption by changing the value of $d$. In our work, 1GB memory can store $2^{(28-\log(d+1))}$ feature embeddings, e.g., if $d = 512$, a computer with 1GB memory can store more than half million feature embedding results.

## IV. Experimental Results

In this section, we present our experimental results on benchmark datasets, performance comparisons with state-of-the-art methods, and ablation studies of our algorithm.

### A. Datasets and Evaluation Metrics

To evaluate the effectiveness of the proposed framework, we conduct experiments on the benchmark datasets: Stanford Online Products dataset [59] and the ImageNet ILSVRC 2012 dataset [60].

The **Stanford Online Products (SOP)** dataset contains 22,634 categories with 120,053 images. Following the public partition rule, we split the first 11,318 classes with 59,551 images for training, and the remaining 11,316 classes with 60,502 images for retrieval test. In the test set, each image is also used as the query image. The **ImageNet ILSVRC 2012** dataset contains 1,000 classes with 1,281,167 training images and 50,000 validation images. In this work, the first 500 classes are used for training, and the remaining 500 classes are used for test.

We follow the standard protocol [3] to evaluate the performance of our algorithm using the following image index performance metrics: (1) the BCubed precision, and (2) F-measure [61]. Following other indexing methods [4], [8], we also adopt the visited list length v.s. Recall@1, where the number of returned query results $n_q$ in **Algorithm** 2 is set to be 1, to evaluate the retrieval performance.

For the BCubed precision and BCubed F-measure, let us denote $(y_i, y_j)$ and $(g_i, g_j)$ as the ground truth labels and group labels of image pair $(x_i, x_j)$, respectively, we first define the pairwise correctness as,

$$\delta(i, j) = \begin{cases} 1 & \text{if } y_i = y_j \text{ and } g_i = g_j \\ 0 & \text{otherwise.} \end{cases} \tag{17}$$

Then, the BCubed Precision and BCubed Recall are defined as:

$$\alpha = \mathbb{E}_i\{\mathbb{E}_{j:g_i=g_j}[\delta(i, j)]\}, \tag{18}$$

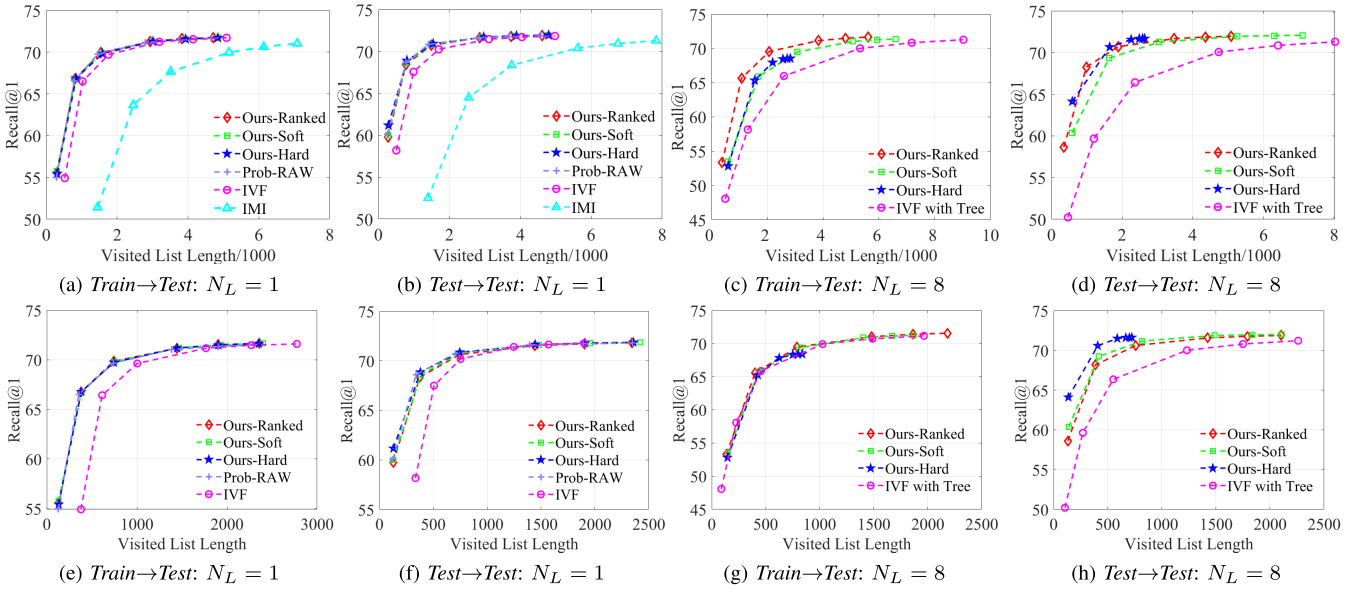
$$\beta = \mathbb{E}_i\{\mathbb{E}_{j:y_i=y_j}[\delta(i, j)]\}, \tag{19}$$

where $\mathbb{E}_i\{\cdot\}$ and $\mathbb{E}_j[\cdot]$ represent the arithmetic average over all pairs $(x_i, x_j)$. The BCubed F-measure is defined as: $\gamma = \frac{2\alpha \cdot \beta}{\alpha + \beta}$. The Recall@1 is computed as follows:

$$\text{Recall@1} = \mathbb{E}_i\{\xi_i\}, \tag{20}$$

where $\xi_i$ is the $i$-th query score. If the category label of the first example after sorting in ascending order (based on distance) is matched with the category label of the $i$-th query, $\xi_i = 1$, otherwise, $\xi_i = 0$. The performance metric of grouping measures the accuracy of whole image class, while the performance metric of visited list length v.s. Recall@1 measures the accuracy of first ranked example after retrieval. They are often jointly used for performance evaluations of indexing methods.

As discussed in Section I, our proposed LTI-ST algorithm is zero-shot. It is able to learn the indexing scheme on the training set and can be applied directly to the test database of images. Existing methods need to analyze the test database to learn and construct the index structure. During performance evaluations, there are two different learning and test scenarios: (1) ***Train*** $\rightarrow$ ***Test*** where the index structure is learnt on the training set and evaluated on the test set, and (2) ***Test*** $\rightarrow$ ***Test***




(a) *Train*→*Test*: $N_L = 1$ (b) *Test*→*Test*: $N_L = 1$ (c) *Train*→*Test*: $N_L = 8$ (d) *Test*→*Test*: $N_L = 8$

(e) *Train*→*Test*: $N_L = 1$ (f) *Test*→*Test*: $N_L = 1$ (g) *Train*→*Test*: $N_L = 8$ (h) *Test*→*Test*: $N_L = 8$

Fig. 6. The comparison with state-of-the-art codebook-based ($N_L = 1$) and tree-based ($N_L = 8$) indexing methods on the **SOP** dataset (the group size is 256). Figs. 6a, 6b, 6c and 6d are results based on the exhaustive in-group retrieval method. Figs. 6e, 6f, 6g and 6h are results based on the HNSW in-group retrieval method.

where the index structure is learnt and also evaluated on the test set. We will compare our algorithm with state-of-the-art methods on both scenarios.

### *B. Implementation Details*

Results shown in this paper are based on the Caffe [62] and faiss [63] frameworks. The GoogLeNet [55] are ResNet-50 [56] are used as the encoder network in our LTI-ST. To generate compact feature embedding and accelerating training, the encoder network is first fine-tuned on the training set using the multi-similarity loss [25]. The number of iterations at the fine-tuning stage is 30,000 for feature embedding. As shown in Table I, the hard prediction network, soft prediction network, and ranked prediction network are three-layer perception (MLP) networks. The index prediction network is also an MLP and its number of layers is 2. During training, the batch size is set as 1000, the initial global learning rate is 0.001. If not specified, $d$ is set as 512. The GPU used in this study is GTX 1080 with 11G memory. The group accuracy on the SOP dataset is computed based on the F-measure. Because the number of images in each class is larger than the number of classes in ImageNet, and the recall groups for each class approximate the number of groups at the grouping stage, thus the group accuracy on the ImageNet is computed based on the BCubed precision in Table III.

### *C. Comparisons With the State-of-the-Art Methods*

For group accuracy evaluations, we compare our algorithm with (1) the distance-based index method IVF and (2) the state-of-the-art learning to index method Prob-RAW, Prob-QCS, and Prob-RAW+QCS [8] methods. For the retrieval performance evaluations, we compare our algorithm with (1) the codebook-based ($N_L = 1$) inverted indexing methods IVF and IMI in [4], (2) the Prob-RAW method [8], and (3) tree-based indexing method [13]. In the following, results marked with hard, soft and ranked are hard, soft and ranked labels predicted by the LTI-ST network.

*1) Results on the SOP Dataset:* Performance comparison of group results on the SOP dataset for two test scenarios *Train*→*Test* and *Test*→*Test* are shown in Table II. We can see that, on the SOP dataset, our method with different labeling schemes obtains the best group performance in both *Train*→*Test* and *Test*→*Test* scenarios. From the results of *Train*→*Test*, we can see that the group accuracy of hard labels, soft labels, and ranked labels are very close. When the number of groups is 256, the ranked labels obtain the best result, 0.78% and 1.26% higher than those by the Prob-RAW method for codebook ($N_L = 1$) and tree ($N_L = 8$), respectively. When the number of groups is 512, the soft labels and hard labels obtain the best results for codebook ($N_L = 1$) and tree ($N_L = 9$), outperforming the Prob-RAW method by 1.52% and 3.19%, respectively. These results show that the class-based labeling methods and the $r$-NN based labeling method are similar for zero-shot learning to index, and better than other indexing methods, especially for tree-based indexing ($N_L = 8$ and $N_L = 9$ in this experiments). From the results of *Test*→*Test*, we can see that the hard labels perform better than soft and ranked labels. This is because the index prediction network learnt based on the hard labels might be easily over-fitting in the scenario of *Test*→*Test*, thus obtain better group performance.

The comparison of retrieval results with the codebook-based ($N_L = 1$) indexing methods and tree-based ($N_L = 8$) indexing methods for scenarios *Train*→*Test* and *Test*→*Test* are shown in Fig. 6. Figs. 6a, 6b, 6c and 6d show results based on the exhaustive in-group retrieval method. Figs. 6e, 6f, 6g and 6h show results based on the HNSW in-group retrieval method.

From Fig. 6, we can see that our learning-based indexing method is much better than those distance-based indexing

TABLE II

THE COMPARISON OF GROUP ACCURACY(%) FOR DIFFERENT NUMBER OF GROUPS AND DIFFERENT NUMBER OF TREE LAYERS ON THE **SOP** DATASET FOR *Train→Test* AND *Test→Test* (∗ IS OUR RESULTS BASED ON THE RANKED LABELS AND THE MLP IS SIMILAR WITH [8].)

| Scenarios | **Train→Test (Zero-Shot)** | | | | **Test→Test** | | | |
|---|---|---|---|---|---|---|---|---|
| Number of Groups | 256 | | 512 | | 256 | | 512 | |
| Number of Tree Layers | 1 | 8 | 1 | 9 | 1 | 8 | 1 | 9 |
| IVF [4], [13] | 56.41 | 48.67 | 52.57 | 45.84 | 59.96 | 50.58 | 58.67 | 48.25 |
| Prob-RAW [8] | 56.76 | 53.20* | 52.97 | 50.46* | 62.21 | 57.82* | 60.78 | 55.61* |
| Prob-QCS [8] | 56.42 | 52.33* | 52.54 | 49.37* | 61.69 | 56.54* | 60.58 | 54.56* |
| Prob-RAW+QCS [8] | 56.66 | 52.66* | 52.84 | 49.55* | 62.28 | 57.35* | 60.95 | 54.87* |
| Index Prediction with Hard Labels | 57.32 | 53.87 | 54.15 | **53.25** | **68.24** | **70.33** | **68.21** | **68.01** |
| Index Prediction with Soft Labels | 57.40 | 54.16 | **54.59** | 53.23 | 64.00 | 63.57 | 62.64 | 59.70 |
| Index Prediction with Ranked Labels | **57.54** | **54.46** | 54.35 | 52.19 | 62.43 | 59.54 | 60.93 | 57.72 |

TABLE III

THE COMPARISON OF GROUP ACCURACY(%) FOR DIFFERENT NUMBER OF GROUPS AND DIFFERENT NUMBER OF TREE LAYERS ON THE **IMAGENET** FOR *Train→Test* AND *Test→Test* (∗ IS OUR RESULTS BASED ON THE RANKED LABELS AND THE MLP IS SIMILAR WITH [8].)

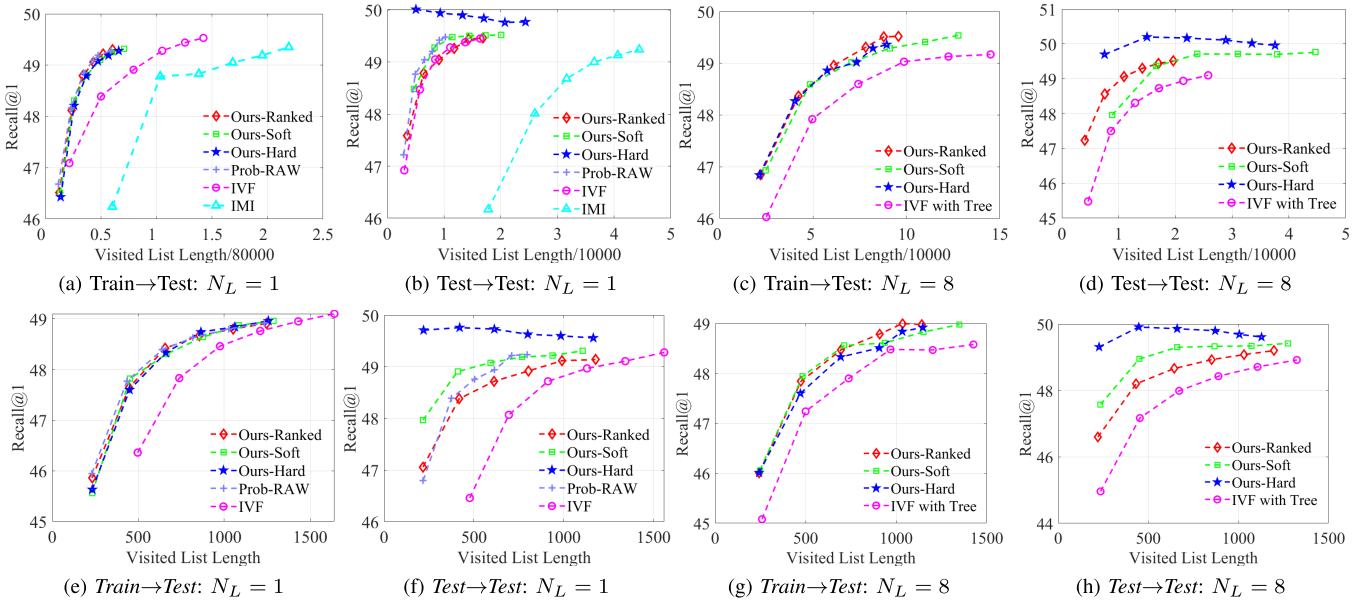
| Scenarios | **Train→Test (Zero-Shot)** | | | | **Test→Test** | | | |
|---|---|---|---|---|---|---|---|---|
| Number of Groups | 256 | | 512 | | 256 | | 512 | |
| Number of Tree Layers | 1 | 8 | 1 | 9 | 1 | 8 | 1 | 9 |
| IVF [4], [13] | **38.14** | 38.00 | 33.30 | 33.51 | 41.11 | 28.72 | 39.97 | 24.02 |
| Prob-RAW [8] | 36.48 | 40.61* | 31.93 | 39.47* | 44.06 | 36.52* | 43.26 | 32.76* |
| Prob-QCS [8] | 35.64 | 38.79* | 31.17 | 38.32* | 42.40 | 34.62* | 40.66 | 30.56* |
| Prob-RAW+QCS [8] | 36.16 | 40.13* | 31.30 | 39.29* | 43.44 | 35.42* | 41.60 | 31.96* |
| Index Prediction with Hard Labels | 37.80 | 40.71 | 33.45 | 41.34 | **64.14** | **63.24** | **51.15** | **53.48** |
| Index Prediction with Soft Labels | 37.69 | **42.03** | **33.74** | **44.46** | 53.45 | 50.48 | 49.09 | 46.36 |
| Index Prediction with Ranked Labels | 37.24 | 41.28 | 33.46 | 40.88 | 46.34 | 39.53 | 42.54 | 35.99 |

methods. From Figs. 6a, 6b, 6e and 6f, we can see that the retrieval performance of our method with hard labels, soft labels, ranked labels and the Prob-RAW method are very close to each other for codebook ($N_L = 1$). They all outperform the distance-based IVF and IMI methods. From Figs. 6c, 6d, 6g, 6h, we can see that the ranked labeling method obtains the best performance in the *Train→Test* scenario. This suggests that the ranked labels are much more efficient for zero-shot learning to index. However, the hard labels obtain the best performance for *Test→Test*, which indicates that the network supervised by hard labels is prone to over-fitting.

*2) Results on the ImageNet Dataset:* Table III shows the comparison of group performance on the ImageNet dataset for test scenarios of *Train→Test* and *Test→Test*. It can be seen that our method obtains the best group performance except the case of 256 groups with one layer in the *Train→Test* scenario. From the results of *Train→Test*, we can see that the soft labels obtain the best result when the number of groups is 256 and the number of layers is 8, outperforming the Prob-RAW method by 1.42%. When the number of groups is 512, soft labels also obtain the best results for both codebook ($N_L = 1$) and tree ($N_L = 9$), 0.44% higher than the IVF method for $N_L = 1$ and 4.99% higher than the Prob-RAW method for $N_L = 9$. For the *Test→Test* scenario, the results are similar to those in Table II.

The comparison of retrieval results for codebook-based ($N_L = 1$) indexing methods and tree-based ($N_L = 8$) indexing methods for scenarios *Train→Test* and *Test→Test* are shown in Fig. 7. Figs. 7a, 7b, 7c and 7d are results based on the exhaustive in-group retrieval. Figs. 7e, 7f, 7g and 7h are results based on the HNSW in-group retrieval. It can be seen that our LTI-ST indexing method performs much better than the distance-based indexing methods. From Figs. 7a and 7e, we can see that the retrieval performance of our method with hard, soft, and ranked labels, and the Prob-RAW method are very close to each other for the train→test scenario with $N_L = 1$. All of them are better than those of the distance-based IVF and IMI methods. From Figs. 7b, 7d, 7f and 7h, we can see that the hard labels obtain the best performance for *Test→Test*, the main reason is that the model is over-fitting with accurate labels information. From Figs. 7c and 7g, we can see that the ranked labels obtain the best retrieval performance for *Train→Test* with $N_L = 8$. The main reason is that the index prediction network learnt with ranked labels can obtain better generalization ability for *Train→Test* in our zero-shot setting.

*3) Retrieval Time Comparison on the SOP and ImageNet Datasets:* The running time depends on specific machines and implementation. Here, we implemented our method with the faiss [63] framework and Python. The retrieval time on the SOP and ImageNet datasets of our method with and without HNSW in-group retrieval is shown in Table IV. We can see that most of the running time is spent on the Python loop (loop of 60502 and 25000 for the SOP and ImageNet datasets, respectively). According to results in Table IV, considering both Recall@1 and time spent, we can see that the best performance is obtained by our index prediction with ranked labels for zero-shot setting on both the SOP and ImageNet datasets. This indicates that the ranked labels have better generalization ability for zero-shot setting. Comparing the retrieval time with exhaustive and HNSW in-group retrieval on the ImageNet dataset, we can see that the retrieval time can be greatly reduced by the HNSW in-group retrieval.




(a) Train→Test: $N_L = 1$ (b) Test→Test: $N_L = 1$ (c) Train→Test: $N_L = 8$ (d) Test→Test: $N_L = 8$

(e) *Train→Test*: $N_L = 1$ (f) *Test→Test*: $N_L = 1$ (g) *Train→Test*: $N_L = 8$ (h) *Test→Test*: $N_L = 8$

Fig. 7. The comparison with state-of-the-art codebook-based ($N_L = 1$) and tree-based ($N_L = 8$) indexing methods on the **ImageNet** dataset (the group size is 256). Figs. 7a, 7b, 7c, and 7d are results based on the exhaustive in-group retrieval method. Figs. 7e, 7f, 7g, and 7h are results based on the HNSW in-group retrieval method.

TABLE IV

THE COMPARISON OF RECALL@1 (%) V.S. RETRIEVAL TIME FOR DIFERENT METHODS ON THE SOP AND IMAGENET FOR **TRAIN→TEST** SCENARIO

| Datasets | **SOP** ($r = 12$, $d = 512$, $N_L = 8$) | | | | **ImageNet** ($r = 6$, $d = 512$, $N_L = 8$) | | | |
|---|---|---|---|---|---|---|---|---|
| In-group Retrieval Methods | Exhaustive | | HNSW | | Exhaustive | | HNSW | |
| Evaluations | Recall@1 ↑ | Time (ms) ↓ | Recall@1 ↑ | time (ms) ↓ | Recall@1 ↑ | time (ms) ↓ | Recall@1 ↑ | time (ms) ↓ |
| IVF with Tree [13] | 70.00 | 1.94 | 69.89 | 1.19 | 49.17 | 35.42 | 48.60 | 0.71 |
| Prob-RAW with Tree [8] | 71.03 | 1.78 | 71.00 | 1.40 | 49.40 | 24.26 | 48.91 | 0.44 |
| Index Prediction with Hard Labels | 68.42 | 0.99 | 68.31 | 0.55 | 49.36 | 22.43 | 48.91 | 0.38 |
| Index Prediction with Soft Labels | 71.00 | 1.95 | 70.88 | 1.17 | 49.54 | 26.52 | 49.02 | 0.52 |
| Index Prediction with Ranked Labels | 71.12 | 1.64 | 71.03 | 1.16 | 49.52 | 23.73 | 49.00 | 0.42 |

TABLE V

THE GROUP ACCURACY(%) ON THE SOP DATASET FOR CODEBOOK ($N_L = 1$) AND TREE ($N_L = 8$)

| Scenarios | **Train→Test:** $N_L = 1$ | | | **Test→Test:** $N_L = 1$ | | | **Train→Test:** $N_L = 8$ | | | **Test→Test:** $N_L = 8$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Loss Functions | Hard | Soft | Ranked | Hard | Soft | Ranked | Hard | Soft | Ranked | Hard | Soft | Ranked |
| Individual Loss | 56.07 | 56.87 | 56.76 | **68.24** | **64.00** | 62.21 | 52.28 | 53.62 | 53.20 | **70.33** | 62.74 | 57.82 |
| + Combined Loss | 56.37 | 56.76 | 56.93 | 65.70 | 63.39 | **62.43** | 53.27 | **54.16** | 53.52 | 69.69 | **63.57** | 59.42 |
| + Center Loss | **57.32** | **57.40** | **57.54** | 65.13 | 63.26 | 62.39 | **53.87** | 53.78 | **54.46** | 69.19 | 63.50 | **59.54** |

### *D. Ablation Studies*

We conduct ablation studies to evaluate the effectiveness of different components of our method.

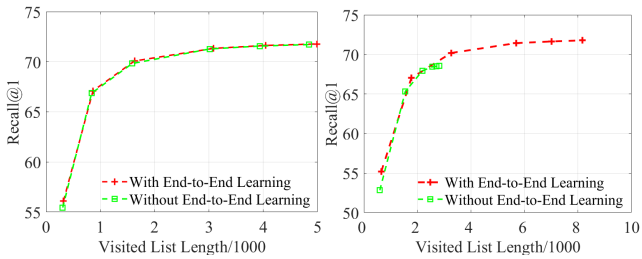
*1) The Effects of Different Loss Functions:* In our LTI-ST method, we have introduced three loss functions corresponding to three different labeling schemes, the combined loss of the concatenated feature, and the center loss. In the following experiments, we study the effects of different loss functions. We conduct experiments on the SOP dataset and fix the group size as 256. Table V shows the group accuracy of *Train→Test* and *Test→Test* for codebook ($N_L = 1$) and tree ($N_L = 8$). The third row shows the results when each individual labels loss function is used. In the fourth row, we show the combined loss. In the fifth row, we show the center loss. We can see that for the *Train→Test* scenario, each function contributes to the overall performance. The combination of them all yield the best performance. But, for the *Test→Test*, it achieves the best performance when the hard labels are used. From this ablation study, we can see that the proposed LTI-ST network and its design of loss functions are valid and effective for zero-shot learning to index.

TABLE VI

THE GROUP ACCURACY(%) ON THE SOP DATASET WITH AND WITHOUT END-TO-END TRAINING

| Scenario | **Train→Test (Zero-Shot)** | | | |
|---|---|---|---|---|
| Number of Groups | 256 | | 512 | |
| Number of Layers | 1 | 8 | 1 | 9 |
| Without End-to-End Learning | 57.32 | 53.87 | 54.15 | 53.25 |
| With End-to-End Learning | **58.09** | **56.15** | **55.11** | **55.77** |

*2) Performance of End-to-End Training:* In this work, we assume that the feature embedding network is pre-trained to accelerate the training process. We then use the embedded feature to learn and predict the index for the input image. In our experiments, for zero-shot learning to index in the *Train→Test* scenario, it is more efficient to perform the end-to-end training of the feature embedding network and the LTI-ST network. Using the hard labels on the index prediction




(a) Train→Test: $N_L = 1$ (b) Train→Test: $N_L = 8$

Fig. 8. The performance of visited list length v.s. Recall@1 on the SOP dataset (the group size is 256).

TABLE VII

THE GROUP ACCURACY(%) ON THE SOP DATASET WITH END-TO-END TRAINING BY SETTING DIFFERENT $\lambda$ (WEIGHT OF CENTER LOSS)

| Scenario | **Train→Test (Zero-Shot)** | | | |
|---|---|---|---|---|
| Number of Groups | 256 | | 512 | |
| Number of Layers | 1 | 8 | 1 | 9 |
| $\lambda = 0$ | 55.89 | 54.41 | 51.54 | 53.55 |
| $\lambda = 0.001$ | **58.09** | **56.15** | **55.11** | **55.77** |
| $\lambda = 0.01$ | 57.39 | 55.36 | 53.67 | 53.91 |

TABLE VIII

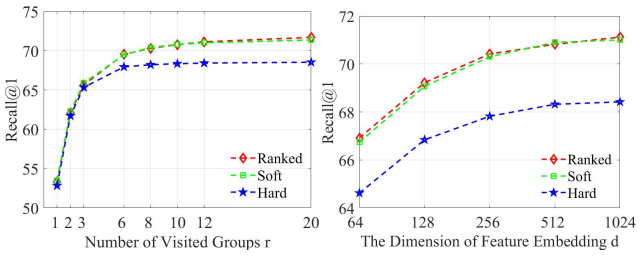
THE GROUP ACCURACY(%) ON THE SOP DATASET WITH END-TO-END TRAINING BY SETTING DIFFERENT SEMANTIC TREE LAYERS $N_L$

| Scenario | **Train→Test (Zero-Shot)** | | | |
|---|---|---|---|---|
| Semantic Tree Layers $N_L$ | 7 | 8 | 9 | 10 |
| Group Accuracy | 59.47 | 56.15 | 55.77 | 50.11 |

network, Table VI shows the group performance on the SOP dataset with and without end-to-end training. Fig. 8 shows the corresponding retrieval performance. We can see that, if the end-to-end training is applied, the performance of our current algorithm (currently without end-to-end training) can be further improved, outperforming existing methods by an even larger margin.

*3) The Impact of Hyperparameters:* The major hyperparameters of our method include the weight of center loss $\lambda$, the number of semantic tree layers $N_L$, the number of visited groups $r$ at the test stage, and the dimension of feature embedding $d$. In the following ablation studies, we evaluate the effect of $\lambda$ and $N_L$ on the SOP dataset with hard labels and end-to-end training. Results are shown in Tables VII and VIII, respectively. From Table VII, we can see that the best performance is obtained by setting $\lambda$ as 0.001, which was used in all previous experiments. From Table VIII, it can be seen that the group accuracy decreases with $N_L$. This is because there will be more information loss with quantization process in semantic tree encoding process and further reduces the accuracy of the encoded semantic tree labels. Another reason is that the semantic division between groups becomes more fine-grained when $N_L$ increases, which will make it more difficult for index prediction.

In the following experiments, we study the effects of $r$ and $d$ on the SOP dataset with three labeling schemes with zero-shot setting. Results are shown in Figs. 9a and 9b, respectively. In Fig. 9, we fix the group size as 256. From Fig. 9a, we can see that as $r$ increases, Recall@1 gradually increases, and tends to stabilize when $r$ is between 8 and 12. In Fig. 9b, we fix $r$ as 12. From Fig. 9b, we can see that the performance




(a) Train→Test with different $r$ (b) Train→Test with different $d$

Fig. 9. The performance with different $r$ and $d$ on the SOP dataset (the group size is 256).

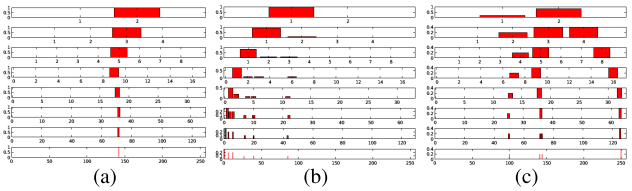


Fig. 10. Some random selected encoded semantic trees from the SOP training dataset (the group size is 256). (a)–(c) represent three semantic tree labels of three training classes.

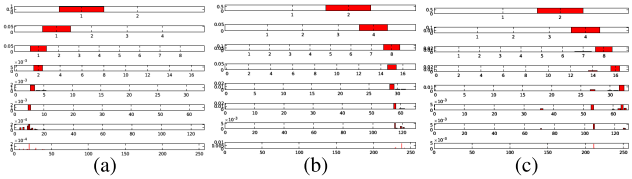


Fig. 11. Some random selected predicted semantic trees from the SOP test dataset (the group size is 256). (a)–(c) represent three predicted semantic trees of three test images.

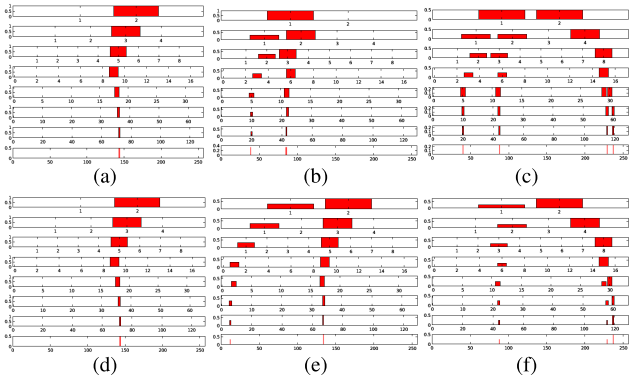
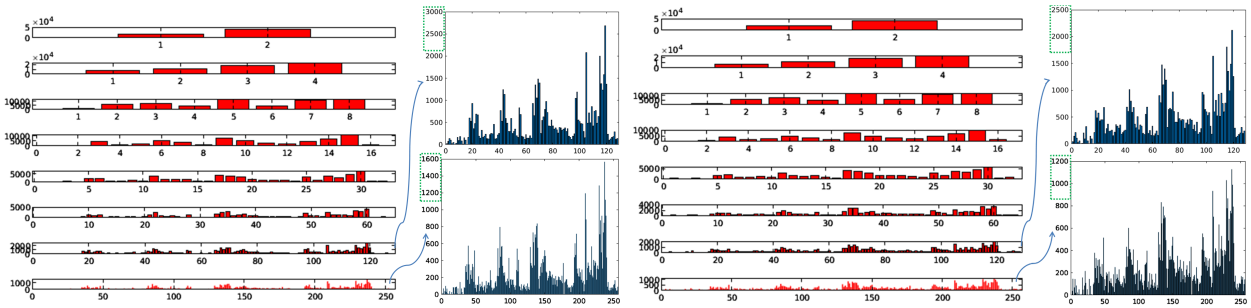


Fig. 12. Distributions of random selected four classes from the SOP test dataset (the group size is 256). Figs. 12a, 12b and 12c are the computed distributions based on the Euclidean distance. Figs. 12d, 12e and 12f are the predicted distributions based on the learnt LTI-ST model. Figs. 12a and 12d, 12b and 12e, 12c and 12f are pairs of distribution of same class, respectively.

of our method gradually increases with the increase of $d$, and saturates at $d = 512$.

*4) Impact of the Encoder Networks:* The encoder network is crucial for the index performance. Here, we conduct experiment on the SOP dataset with the ResNet-50 backbone network. By setting $d = 512$, $r = 12$, $N_L = 8$, and using the ranked labeling scheme to guide the learning of index prediction network, the Recall@1 for Train→Test is 75.4%, higher 4.4% than with the GoogLeNet backbone (71.0%). This experiment indicates that the performance of image retrieval can be greatly increased with a stronger encoder network.



(a) Euclidean distance-based index distribution (b) LTI-ST-based index distribution

Fig. 13. The index distributions of the SOP test set based on the Euclidean distance and LTI-ST methods, respectively.

TABLE IX
THE STANDARD DEVIATIONS OF LAST FOUR LAYERS FOR EUCLIDEAN DISTANCE AND LTI-ST INDEX DISTRIBUTIONS

| Layers | 5 | 6 | 7 | 8 |
|---|---|---|---|---|
| Euclidean Distance | 1541 | 857 | 457 | 239 |
| LTI-ST | 1361 | 737 | 387 | 205 |

*5) Semantic Tree Encoding:* As analyzed in Section III B, the semantic tree encoding should form valid paths on the tree. Fig. 10 shows some samples of encoded semantic trees. It can be seen that each semantic tree consists of one or more valid paths. As shown in Figs. 10a and 10b, the semantic trees have a main path, and most samples with the same class label are distributed on this path. However, as shown in Fig. 10c, there still exists some semantic trees that consist of more than one salient paths, which can decrease the labeling performance, especially for hard labeling scheme. This is why the index performance of hard labeling scheme is slightly worse than the soft labeling and ranked labeling schemes for zero-shot learning to index.

*6) Semantic Tree Decoding:* Samples of predicted semantic trees are shown in Fig. 11. It can be seen that each predicted semantic tree consists of a number of valid paths with a salient path. However, as shown in Fig. 11c, a salient path may not contain a truly salient node. Thus, we should set $r \geq \theta$ at the semantic tree decoding stage to ensure that an accurate index value can be obtained at each layer. In our experiments, we set the value of $\theta$ as 12 and 6 for the SOP and ImageNet datasets, respectively.

*7) Index Distribution:* The distribution of images on the leaf nodes affects the retrieval performance. More samples a leaf node contains, more time it will take to retrieve a candidate set from that node. If the image samples from one class are distributed to more leaf nodes, we need to visit more nodes to retrieve all the samples in that class. This work considers nearest neighbor search. Example pairs of distributions of random selected classes from the SOP test dataset are shown in Fig. 12. Figs. 12a, 12b and 12c are the distributions based on the Euclidean distance. Figs. 12d, 12e and 12f are distributions based on the learnt LTI-ST model. Figs. 12a and 12d, 12b and 12e, 12c and 12f are the pairs of distributions of same class, respectively.

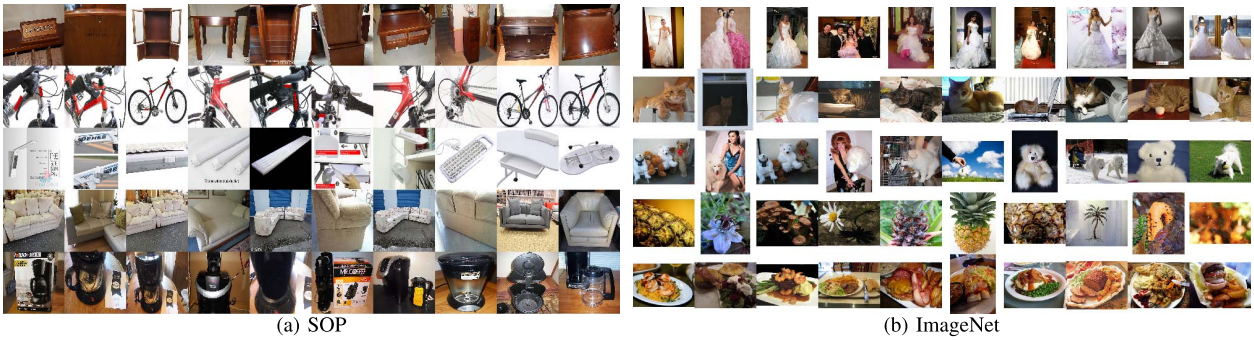
From Fig. 12, we can see that the distributions of LTI-ST-based index are more aggregated than the distributions of Euclidean distance-based indexing schemes, especially in Figs. 12c and 12f. This indicates that the index efficiency of our LTI-ST method is higher than those of the distance-based index methods for zero-shot index problem. From Figs. 12b and 12e, we can see that the predicted index by the learnt LTI-ST model are different from the Euclidean distance-based index.

Furthermore, we compute the index distribution of the SOP test set for Euclidean distance and LTI-ST, shown in Figs. 13a and 13b, respectively. From Fig. 13, we can see that the number of examples in the largest group by our LTI-ST model is smaller than those by the distance-based method. At the retrieval stage, the average number of visited list length of LTI-ST will be smaller. Also, we compute the standard deviations of last four layers for Euclidean distance (Fig. 13a) and LTI-ST (Fig. 13b) index distributions, results are shown in Table IX. We can see that the index distribution of LTI-ST has lower standard deviations than the Euclidean distance on these layers, which indicates that the index distribution of LTI-ST is more uniform than the Euclidean distance.

In Fig. 14, we show some indexed examples obtained by our LTI-ST method on the SOP and ImageNet test sets, respectively. Fig. 14a and Fig. 14b are randomly selected 8 groups from the SOP and ImageNet indexed results (the group size is 256). Images of each row are selected from corresponding group center. For example, images on the first row of the SOP and ImageNet datasets are brown wooden furniture and wedding gowns, respectively. From Fig. 14, we can see that images indexed to one group contain similar objects or similar scenes, thus share some similar semantic information.

### *E. Discussion*

In the above experiments, the proposed three different labeling schemes are used in the index prediction network. We have conducted experiments to evaluate their performance. From these results in Tables II, III and V, we can see that the index prediction with the hard labels achieves the best performance for the *Test* $\rightarrow$ *Test* scenario. However, for the zero-shot scenario (*Train* $\rightarrow$ *Test*), the best index prediction performance is achieved by the soft and ranked labels. According to Figs. 6c, 6g, 7c and 7g, the top-1 retrieval performance with the ranked labels is a slightly better than the soft labels. For non top-1 retrieval results, the soft labels show slightly better performance than the ranked labels. This is because the soft labels can better characterize the average similarity for groups of samples in a feature similarity-based image retrieval setting than the ranked labels.



(a) SOP (b) ImageNet

Fig. 14. Some examples indexed by the zero-shot LTI-ST method on the SOP and ImageNet test sets, respectively.

## V. Conclusion

In this paper, we have developed a zero-shot learning to index on semantic trees for scalable image retrieval. On the training set, using embedded learnt features, we constructed a hierarchical tree of image clusters and encoded each training image with a binary code being mapped on this tree. This binary code is then used to generate labels for the training images. The image-label pairs are used to train an LTI-ST model. After the LTI-ST network is trained, it can be used to predict the tree indexing structure for test images. Experimental results on the SOP and ImageNet datasets demonstrated that the proposed method outperforms the state-of-the-art index learning methods while providing zero-shot and scalable capabilities, which are highly desirable in practice. Future promising work could be focused on exploring the semantic tree information with ranked labels to build the model of unsupervised metric learning or representation learning with indexing.

## References


[1] B. Klein and L. Wolf, "End-To-End supervised product quantization for image search and retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 5041–5050.

[2] S. Li, Z. Chen, J. Lu, X. Li, and J. Zhou, "Neighborhood preserving hashing for scalable video retrieval," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2019, pp. 8212–8221.

[3] Z. Wang, L. Zheng, Y. Li, and S. Wang, "Linkage based face clustering via graph convolution network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 1117–1125.

[4] A. Babenko and V. Lempitsky, "The inverted multi-index," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 6, pp. 1247–1260, Jun. 2015.

[5] D. Baranchuk, A. Babenko, and Y. Malkov, "Revisiting the inverted indices for billion-scale approximate nearest neighbors," in *Proc. 15th Eur. Conf. Comput. Vis. (ECCV)*, Munich, Germany, Sep. 2018, pp. 209–224.

[6] S. Liu, J. Shao, and H. Lu, "Generalized residual vector quantization and aggregating tree for large scale search," *IEEE Trans. Multimedia*, vol. 19, no. 8, pp. 1785–1797, Aug. 2017.

[7] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *Proc. 9th IEEE Int. Conf. Comput. Vis.*, Nice, France, Oct. 2003, pp. 1470–1477.

[8] C.-Y. Chiu, A. Prayoonwong, and Y.-C. Liao, "Learning to index for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 8, pp. 1942–1956, Aug. 2020.

[9] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Prague, Czech Republic, May 2011, pp. 861–864.

[10] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in *Proc. 23rd Int. Conf. Mach. Learn. (ICML)*, Pittsburgh, PA, USA, Jun. 2006, pp. 97–104.

[11] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Anchorage, AK, USA, Jun. 2008, pp. 24–26.

[12] M. E. Houle and M. Nett, "Rank-based similarity search: Reducing the dimensional dependence," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 1, pp. 136–150, Jan. 2015.

[13] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2227–2240, Nov. 2014.

[14] R. Liu, S. Wei, Y. Zhao, and Y. Yang, "Indexing of the CNN features for the large scale image search," *Multimedia Tools Appl.*, vol. 77, no. 24, pp. 32107–32131, Dec. 2018.

[15] S. Kan, L. Cen, X. Zheng, Y. Cen, Z. Zhu, and H. Wang, "A supervised learning to index model for approximate nearest neighbor image retrieval," *Signal Process., Image Commun.*, vol. 78, pp. 494–502, Oct. 2019.

[16] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011.

[17] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 4, pp. 744–755, Apr. 2014.

[18] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Columbus, OH, USA, Jun. 2014, pp. 2329–2336.

[19] L. Jin, X. Shu, K. Li, Z. Li, G.-J. Qi, and J. Tang, "Deep ordinal hashing with spatial attention," *IEEE Trans. Image Process.*, vol. 28, no. 5, pp. 2173–2186, May 2019.

[20] S. Zhang, J. Li, and B. Zhang, "Semantic cluster unary loss for efficient deep hashing," *IEEE Trans. Image Process.*, vol. 28, no. 6, pp. 2908–2920, Jun. 2019.

[21] C. Deng, E. Yang, T. Liu, J. Li, W. Liu, and D. Tao, "Unsupervised semantic-preserving adversarial hashing for image search," *IEEE Trans. Image Process.*, vol. 28, no. 8, pp. 4032–4044, Aug. 2019.

[22] C. Deng, E. Yang, T. Liu, and D. Tao, "Two-stream deep hashing with class-specific centers for supervised image search," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 2189–2201, Jun. 2020.

[23] M. Opitz, G. Waltner, H. Possegger, and H. Bischof, "Deep metric learning with BIER: Boosting independent embeddings robustly," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 276–290, Feb. 2020.

[24] S. Kan, Y. Cen, Z. He, Z. Zhang, L. Zhang, and Y. Wang, "Supervised deep feature embedding with handcrafted feature," *IEEE Trans. Image Process.*, vol. 28, no. 12, pp. 5809–5823, Dec. 2019.

[25] X. Wang, X. Han, W. Huang, D. Dong, and M. R. Scott, "Multi-similarity loss with general pair weighting for deep metric learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 5022–5030.

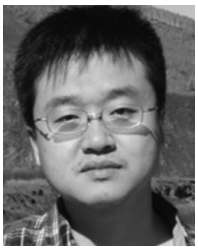
[26] X. Wang, H. Zhang, W. Huang, and M. R. Scott, "Cross-batch memory for embedding learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, Jun. 2020, pp. 6387–6396.

[27] M. Norouzi, A. Punjani, and D. J. Fleet, "Fast exact search in Hamming space with multi-index hashing," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 6, pp. 1107–1119, Jun. 2014.

[28] S. Eghbali, H. Ashtiani, and L. Tahvildari, "Online nearest neighbor search using Hamming weight trees," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 7, pp. 1729–1740, Jul. 2020.

[29] E.-J. Ong and M. Bober, "Improved Hamming distance search using variable length hashing," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 2000–2008.
[30] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.
[31] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, Nov. 1998.
[32] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, San Juan, Puerto Rico, Jun. 1997, pp. 1000–1006.
[33] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-Nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, no. 7, pp. 750–753, Jul. 1975.
[34] A. W. Moore, "The anchors hierarchy: Using the triangle inequality to survive high dimensional data," in *Proc. 16th Conf. Uncertainty Artif. Intell. (UAI)*. Stanford, CA, USA: Stanford Univ., Jun./Jul. 2000, pp. 397–405.
[35] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. 4th Annu. ACM/SIGACT-SIAM Symp. Discrete Algorithms*, Austin, TX, USA, Jan. 1993, pp. 311–321.
[36] T. Liu, A. W. Moore, A. G. Gray, and K. Yang, "An investigation of practical approximate nearest neighbor algorithms," in *Proc. Adv. Neural Inf. Process. Syst. (NIPS)*, Vancouver, BC, Canada, Dec. 2004, pp. 825–832.
[37] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, vol. 2, New York, NY, USA, Jun. 2006, pp. 2161–2168.
[38] B. Leibe, K. Mikolajczyk, and B. Schiele, "Efficient clustering and matching for object class recognition," in *Proc. Brit. Mach. Vis. Conf.*, Edinburgh, U.K., Sep. 2006, pp. 789–798.
[39] G. Schindler, M. A. Brown, and R. Szeliski, "City-scale location recognition," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Minneapolis, MN, USA, Jun. 2007, pp. 18–23.
[40] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo, "Fast neighborhood graph search using Cartesian concatenation," in *Multimedia Data Mining and Analytics—Disruptive Innovation*. Springer, 2015, pp. 397–417.
[41] J. Wang and S. Li, "Query-driven iterated neighborhood graph search for large scale indexing," in *Proc. 20th ACM Multimedia Conf. (MM)*, Nara, Japan: ACM, Oct./Nov. 2012, pp. 179–188.
[42] E. Chávez and E. S. Tellez, "Navigating $K$-nearest neighbor graphs to solve nearest neighbor searches," in *Proc. 2nd Mexico Conf. Pattern Recognit. Adv. Pattern Recognit. (MCPR)*, in Lecture Notes in Computer Science, vol. 6256. Puebla, Mexico: Springer, Sep. 2010, pp. 270–280.
[43] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov, "Approximate nearest neighbor algorithm based on navigable small world graphs," *Inf. Syst.*, vol. 45, pp. 61–68, Sep. 2014.
[44] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 4, pp. 824–836, Apr. 2020.
[45] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proc. VLDB Endowment*, vol. 12, no. 5, pp. 461–474, Jan. 2019.
[46] C. Fu, C. Wang, and D. Cai, "Satellite system graph: Towards the efficiency up-boundary of graph-based approximate nearest neighbor search," 2019, *arXiv:1907.06146*. [Online]. Available: http://arxiv.org/abs/1907.06146
[47] W. Wang, V. W. Zheng, H. Yu, and C. Miao, "A survey of zero-shot learning: Settings, methods, and applications," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, pp. 1–37, 2019.
[48] M. Kampffmeyer, Y. Chen, X. Liang, H. Wang, Y. Zhang, and E. P. Xing, "Rethinking knowledge graph propagation for zero-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 11487–11496.
[49] D. Huynh and E. Elhamifar, "A shared multi-attention framework for multi-label zero-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 8776–8786.
[50] K. Wei, M. Yang, H. Wang, C. Deng, and X. Liu, "Adversarial fine-grained composition learning for unseen attribute-object recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, Oct. 2019, pp. 3740–3748.
[51] B. Brattoli, J. Tighe, F. Zhdanov, P. Perona, and K. Chalupka, "Rethinking zero-shot video classification: End-to-end training for realistic applications," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 4613–4623.
[52] Y. Shen, L. Liu, F. Shen, and L. Shao, "Zero-shot sketch-image hashing," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Salt Lake City, UT, USA, Jun. 2018, pp. 3598–3607.
[53] B. Chen and W. Deng, "Hybrid-attention based decoupled metric learning for zero-shot image retrieval," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, Jun. 2019, pp. 2750–2759.
[54] X. Xu, M. Yang, Y. Yang, and H. Wang, "Progressive domain-independent feature decomposition network for zero-shot sketch-based image retrieval," in *Proc. 29th Int. Joint Conf. Artif. Intell.*, Jul. 2020, pp. 984–990.
[55] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9.
[56] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.
[57] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proc. 14th Int. Conf. Artif. Intell. Statist. (AISTATS)*, in JMLR Proceedings, vol. 15. Fort Lauderdale, FL, USA: JMLR.org, Apr. 2011, pp. 315–323.
[58] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[59] H. O. Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, Jun. 2016, pp. 4004–4012.
[60] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, Dec. 2015.
[61] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, "A comparison of extrinsic clustering evaluation metrics based on formal constraints," *Inf. Retr.*, vol. 12, no. 4, pp. 461–486, Aug. 2009.
[62] Y. Jia *et al.*, "Caffe: Convolutional architecture for fast feature embedding," in *Proc. ACM Int. Conf. Multimedia (MM)*, Orlando, FL, USA, 2014, pp. 675–678.
[63] J. Johnson, M. Douze, and H. Jegou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, early access, Jun. 7, 2019, doi: 10.1109/TBDATA.2019.2921572.

**Shichao Kan** received the B.E. and M.S. degrees from the School of Computer and Information Science, Beijing Jiaotong University, Beijing, China, in 2014 and 2016, respectively, where he is currently pursuing the Ph.D. degree. From 2019 to 2020, he was a Visiting Student Researcher with the Department of Computer Science, University of Missouri, Columbia, MO, USA. His research interests include large-scale image retrieval, object search, metric learning, and deep learning.

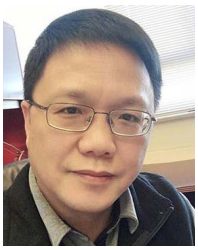


**Yi Cen** received the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2014. He is currently a Lecturer with the School of Information Engineering, Minzu University of China, Beijing. His research interests include compressed sensing, sparse representation, and low-rank matrix reconstruction.

**Yigang Cen** received the Ph.D. degree in control science engineering from the Huazhong University of Science Technology, Wuhan, China, in 2006. In 2006, he joined Signal Processing Center, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, as a Research Fellow. From 2014 to 2015, he was a Visiting Scholar with the Department of Computer Science, University of Missouri, Columbia, MO, USA. He is currently a Professor and a Supervisor of Doctoral Student with the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. His research interests include compressed sensing, sparse representation, low-rank matrix reconstruction, and wavelet construction theory.

**Mladenovic Vladimir** is currently an Associate Professor with the Faculty of Technical Sciences, University of Kragujevac, Čačak. His research interests include wireless communication and image processing.

**Yang Li** (Graduate Student Member, IEEE) is currently pursuing the Ph.D. degree with the Department of Electrical Engineering and Computer Science, University of Missouri, Columbia, MO, USA. His current research interests include semi-supervised learning, unsupervised learning, video compression, and metric learning.



**Zhihai He** (Fellow, IEEE) received the B.S. degree in mathematics from Beijing Normal University, Beijing, China, in 1994, the M.S. degree in mathematics from the Institute of Computational Mathematics, Chinese Academy of Sciences, Beijing, in 1997, and the Ph.D. degree in electrical engineering from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 2001.

In 2001, he joined Sarnoff Corporation, Princeton, NJ, USA, as a Member of Technical Staff. In 2003, he joined the Department of Electrical and Computer Engineering, University of Missouri, Columbia MO, USA, where he is currently a Tenured Full Professor. His current research interests include image/video processing and compression, wireless sensor networks, computer vision, and cyber-physical systems. He is also a member of the Visual Signal Processing and Communication Technical Committee of the IEEE Circuits and Systems Society. He also serves as a technical program committee member or a session chair of a number of international conferences. He was a recipient of the 2002 IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY Best Paper Award and the SPIE VCIP Young Investigator Award in 2004. He was the Co-Chair of the 2007 International Symposium on Multimedia over Wireless in Hawaii. He has served as an Associate Editor for the textscIEEE Transactions on Circuits and Systems for Video Technology (TCSVT), the textscIEEE Transactions on Multimedia (TMM), and the *Journal of Visual Communication and Image Representation*. He was also the Guest Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY (TCSVT) Special Issue on Video Surveillance.