Computer Vision and Image Understanding 000 (2017) 1–11

Contents lists available at ScienceDirect

Computer Vision and Image Understanding

journal homepage: www.elsevier.com/locate/cviu

Large-scale image retrieval using transductive support vector machines

Hakan Cevikalp[a,*], Merve Elmas[a], Savas Ozkan[b]

[a] *Electrical and Electronics Engineering Department of Eskisehir Osmangazi University, Meselik 26480 Eskisehir, Turkey*
[b] *Turkiye Bilimsel ve Teknolojik Arastirma Grubu (TUBITAK) - UZAY, Ankara, Turkey*

ARTICLE INFO

Article history:
Received 21 February 2017
Revised 16 June 2017
Accepted 20 July 2017
Available online xxx

Keywords:
Image retrieval
Hashing
Transductive support vector machines
Semi-supervised learning
Ramp loss

ABSTRACT

In this paper, we propose a new method for large-scale image retrieval by using binary hierarchical trees and transductive support vector machines (TSVMs). We create multiple hierarchical trees based on the separability of the visual object classes, and TSVM classifier is used to find the hyperplane that best separates both the labeled and unlabeled data samples at each node of the binary hierarchical trees (BHTs). Then the separating hyperplanes returned by TSVM are used to create binary codes or to reduce the dimension. We propose a novel TSVM method that is more robust to the noisy labels by interchanging the classical Hinge loss with the robust Ramp loss. Stochastic gradient based solver is used to learn TSVM classifier to ensure that the method scales well with large-scale data sets. The proposed method significantly improves the Euclidean distance metric and achieves comparable results to the state-of-the-art on CIFAR10 and MNIST data sets, and significantly outperforms the state-of-the-art hashing methods on more challenging ImageCLEF 2013, NUS-WIDE, and CIFAR100 data sets.

1. Introduction

Large-scale image retrieval has recently attracted great attention due to the rapid growth of visual data produced by popularization of digital cameras, web-based services and social networks. Clearly, there is an emerging need for a complete visual search framework to retrieve relevant visual contents from such massive data. Despite the great research efforts, visual retrieval is still a challenging problem since web-scale visual search demands highly efficient and accurate retrieval methods. In general, image retrieval can be defined as follows: Given a query image, finding and representing (in an ordered manner) the images depicting the same scene or objects in large unordered image collections.

For large-scale image search, the most commonly used method is the hashing method that enables us to approximate the nearest neighbor search. Hashing methods convert each image feature vector in the database into a compact code (typically a binary code) and provide constant or sub-linear search time. Most of the hashing methods rely on Euclidean distances. However, due to the semantic gap between the low-level features and semantics, Euclidean distances in the feature space do not reflect the semantic similarities between the images. Furthermore, the state-of-the-art image visual features are typically high-dimensional vectors ranging from several thousands to millions. As pointed out in Cevikalp et al. (2008a), the performance of the nearest-neighbor techniques using the Euclidean distances in high-dimensional spaces is poor since sparse and irregular distributions of data samples tend to have many holes (regions that have few or no nearby samples from the same classes), so it is necessary to learn more discriminative distance metrics. Our experimental results at the end also verify these claims. Note that the authors in Torralba et al. (2008) observed that when the data set size grows, visual similarity approaches to the semantic similarity. This stems from the fact that as the number of samples is increased when the dimensionality of the input space is fixed, the distributions become denser and the holes are filled in mostly with the correct neighbors. That is why (Torralba et al., 2008) experiment with very low-dimensional pixel values or gist features and a large training set size (80 million tiny images). But the dimensionality of the state-of-the-art visual image representations is typically much higher, e.g., the dimensionality of fisher vectors (FV) is approximately 64K for moderate sized images, and the training set size must be much higher to fill in the holes with correct neighbors. Therefore, relying on Euclidean distances between image feature vectors for creating binary codes can be misleading. Another problem is that, the most learning algorithms work with only a limited amount of selected data samples while ignoring most of the training samples since these methods do not scale well with large training set size. Similarly, some methods use only labeled data and ignore any potential information conveyed in unlabeled data samples.

* Corresponding author.
E-mail addresses: hakan.cevikalp@gmail.com (H. Cevikalp), merveelmas1@gmail.com (M. Elmas), savas.ozkan@tubitak.gov.tr (S. Ozkan).

http://dx.doi.org/10.1016/j.cviu.2017.07.004
1077-3142/© 2017 Elsevier Inc. All rights reserved.

Related Work: Majority of the current popular hashing methods (Gionis et al., 1999; Gong et al., 2013b; Heo et al., 2012; Kulis and Darrell, 2009; Kulis and Grauman, 2009; Liu et al., 2011a; Raginsky and Lazebnik, 2009; Salakhutdinov and Hinton, 2009; Weiss et al., 2008) are unsupervised and they are built on the assumption that the similar images in the Euclidean space must have similar binary codes. Among these, Locality Sensitive Hashing (LSH) (Gionis et al., 1999) chooses random projections so that two closest image samples in the feature space fall into the same bucket with a high probability. To solve the challenging semantic gap problem, the most straightforward solution is to use label information to improve the distance metric. But, labeling all images in large image databases is too costly and difficult in practice. In contrast, relevant feedback given in terms of similar/dissimilar pairs is much easier to collect. Similarly, for most of the images on the web, some label tags can be collected at a more reasonable cost by using image file names or surrounding text. So, both semi-supervised and supervised hashing methods utilizing these types of information have been proposed (He et al., 2008; Hoi et al., 2008; Joly and Buisson, 2011; Liu et al., 2012; Mu et al., 2010; Norouzi and Fleet, 2011; Wang et al., 2010; Zhang et al., 2012). Majority of these methods (He et al., 2008; Liu et al., 2012; Norouzi and Fleet, 2011; Wang et al., 2010; Zhang et al., 2012) use label information during creating a similarity matrix, and then, projection directions that will preserve the similarities within the similarity matrix are found. Finally, these directions are used to produce binary codes. These methods cannot be applied directly to large-scale image datasets since they require computing and operating on a very large $n \times n$ sized similarity matrix, where n is the total number of image samples in the training (gallery) set. Generally, two procedures are followed to avoid this problem: In the first approach, only a small number of labeled data samples is used to learn binary codes and all unlabeled data samples are ignored. In the second procedure, some representative anchor points are created by random selection or clustering, and the similarity matrix of all data is approximated with much smaller sized similarity matrix of those anchor points. Both procedures are problematic in the sense that some potential information that may come from unlabeled data samples are ignored and propagation of supervised information from labeled samples to the neighboring unlabeled samples has not been taken into consideration.[1] Shi et al. (2016) introduced a new extension of the supervised hashing method proposed in Liu et al. (2012) by adding an additional regularization term to the optimization function and with some other minor changes to improve the model. Cakir and Sclaroff (2015) introduced an online supervised method for hashing. The methods (Hoi et al., 2008; Joly and Buisson, 2011; Mu et al., 2010) that are more related to ours use SVM based large margin classifiers to learn compact binary codes. Both Mu et al. (2010) and Hoi et al. (2008) use only labeled data since their methods require to operate on a $n \times n$ sized kernel matrix. Thus, unlabeled data are ignored again, and they do not contribute to the label propagation. The method proposed in Joly and Buisson (2011) does not need any supervision, and the authors randomly select some samples and randomly assign them positive and negative labels. Then they run SVM algorithm to find the hyperplanes separating these samples, and finally separating hyperplanes are used to produce binary codes.

More recently, deep neural networks and CNN (Convolutional Neural Networks) features have been used for image retrieval (Gong et al., 2013a; Lai et al., 2015; Lin et al., 2015; Xia et al., 2014; Zhang et al., 2015; 2016; Zhao et al., 2015). These methods typically follow the similar structure of classifier networks that use a stack of convolutional layers to produce discriminative features, but the last layers use different loss functions that are more suitable for retrieval applications. For example, Lai et al. (2015), Zhang et al. (2015) and Gordo et al. (2016) use a triplet ranking loss designed to characterize one image is more similar to the second image than the third one whereas some methods use other loss functions such as surrogate loss (Zhao et al., 2015), pair-wise ranking (Gong et al., 2013a; Liu et al., 2016), Wasserstein loss (Frogner et al., 2015), or weighted approximate ranking (Gong et al., 2013a). Zhang et al. (2016) introduced a new method for speeding up the training of deep neural networks for supervised hashing. Almost all these deep neural network methods are trained end-to-end fashion, which allows one to optimize the discriminative image features and hash functions simultaneously.

Our Contributions: Similar to the hashing methods using large-margin based classifiers, we also use SVM classifiers to learn binary codes, but in contrast to other methods, we incorporate the unlabeled data during learning process in a transductive learning setting. Since the labeled data can be noisy in large-scale image retrieval applications, we introduce a more robust transductive SVM (TSVM) method to the noise present in labels. We use stochastic gradient based solver instead of sequential minimal optimization (SMO) to solve the optimization problem, thus our method scales well with large-scale data (to the best of our knowledge, it is the second transductive method that can be used with more than a million data). We also adopt the method given in Cevikalp (2010) for learning class hierarchies based on graph cut and binary hierarchical trees to ensure the large margin between different class samples. As a final contribution, we developed a new multi-label image retrieval data set using a subset of ImageCLEF 2013 dataset.

Our method has great advantages over recent deep learning based and other hashing methods: First of all, we do not need many labeled data as in deep neural nets. Deep learning based methods have many parameters (in terms of millions, e.g. AlexNet (Krizhevsky et al., 2012) used in our experiments has 60 million parameters), and they require many labeled data to train a model. As a second limitation, how to handling multi-labeled data is still not a resolved problem for many hashing methods whereas the proposed method can handle multi-labeled data gracefully. Our method is also more robust to the noisy labels. In contrast, it is shown in Gordo et al. (2016) that deep neural net based hashing methods are very sensitive to the noisy labels. But, the major advantage of our proposed method compared to the deep neural net based hashing methods and other hashing methods is its ability to use unlabeled data. Preliminary version of this paper appeared in Cevikalp et al. (2016). This paper extends our previous work with (1) a more detailed analysis of the recent related work on image retrieval; (2) a more detailed description of the proposed methodology; (3) new experiments on new datasets such as CIFAR100 and newly developed ImageCLEF 2013; and (4) extended comparisons of the proposed method to some related supervised methods.

2. Method

Here we consider the scenario where we have many unlabeled images with some limited amount of labeled image data. As we mentioned earlier, labels can be gathered from image file names or nearby text on the web. But, we have to keep in mind that the labels can be very noisy and there might be more than one labels attached to an image, e.g., if an image contains people, car, buildings etc., all these tags can be used to label the image. We use TSVMs to create binary hash codes. It should be noted that SVM like large-margin based classifiers are widely used for this goal and it was shown that larger margin between class samples

[1] There have been attempts to propagate label information in graph-based metric learning methods, but these methods (Cevikalp et al., 2008b; Schölkopf et al., 2005) also do not scale well with large training set size.

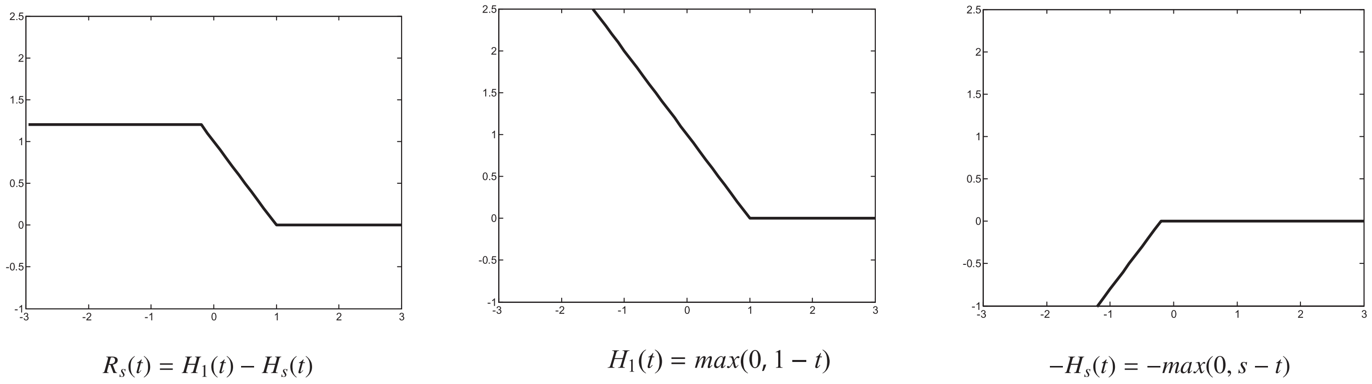


Fig. 1. The illustration of the Ramp loss function, $R_s(t) = H_1(t) - H_s(t)$, where $H_a(t) = max(0, a-t)$ is the classical Hinge loss. Here, we set $s = -0.20$.

yields to lower error rates in similarity search (Joly and Buisson, 2011; Mu et al., 2010). So, our goal is to find the best separating hyperplanes which will create balanced binary hash codes but at the same time they will yield to a large margin between the image samples of different classes.

In the proposed methodology, we first create image class hierarchies based on their visual content similarities and labels. To this end, we use binary hierarchical trees and Normalized Cuts clustering. This methodology is much better compared to the Wordnet based hierarchy used in Fergus et al. (2010) since it is created based on the separability of the visual classes. Then, we use TSVM to find the hyperplane that best separates the data samples (both labeled and unlabeled data) at each node of the binary hierarchical tree to make sure that the classes are split into two clusters with the largest margin possible. We first explain our novel TSVM algorithm and then describe how to create binary hierarchical trees and compact binary codes below.

2.1. Robust Transductive Support Vector Machines (RTSVMs)

Suppose that we are given a set of L labeled training samples $\mathcal{L} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_L, y_L)\}$, $\mathbf{x} \in \mathrm{IR}^d$, $y \in \{+1, -1\}$ and an unlabeled set of U samples $\mathcal{U} = \{\mathbf{x}_{L+1}, \ldots, \mathbf{x}_{L+U}\}$. Our goal is to find the best separating hyperplane characterized by $\theta = (\mathbf{w}, b)$, where $\mathbf{w}$ is the normal of the hyperplane and b is the bias term. We use separating hyperplanes to create binary codes and the sign of the following decision function defines the binary codes

$$f_\theta(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b. \tag{1}$$

The main idea of TSVM learning is to find an hyperplane that separates the labeled samples with a large margin at the same time ensures that the unlabeled samples will be as far as possible from the margin. So, both the labeled and unlabeled data play a dominant role for finding the separating hyperplane. To this end, Collobert et al. (2006) used the following optimization formulation

$$\begin{aligned}&\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} H_1(y_i(\mathbf{w}^\top\mathbf{x}_i + b))\\ &+C^*\sum_{i=L+1}^{L+U} SR_s(\mathbf{w}^\top\mathbf{x}_i + b)\\ &\text{s.t.} \frac{1}{U}\sum_{i=L+1}^{L+U}(\mathbf{w}^\top\mathbf{x}_i + b) = \frac{1}{L}\sum_{i=1}^{L} y_i.\end{aligned} \tag{2}$$

where the function $H_1(t) = max(0, 1-t)$ is the classical Hinge loss plotted in Fig. 1, and $C(C^*)$ is a user defined parameter that controls the weight of errors associated to the labeled (unlabeled) data samples, and SR_s is the symmetric Ramp loss defined as

$$SR_s(t) = R_s(t) + R_s(-t), \tag{3}$$

where $R_s(t) = min(1-s, max(0, 1-t))$ is the Ramp Loss function illustrated in Fig. 1. Here $-1 < s \le 0$ is a parameter that must be set by the user.

It should be noted that the loss functions for labeled and unlabeled data are not in the same range. For the symmetric Ramp loss used for unlabeled data, a sample can introduce at most a limited amount of cost value no matter of its position with respect to the margin in the input space (the loss can be maximum 0.8 when s is set to -0.2). However, there is no bound for the Hinge loss used for labeled samples, e.g., a single outlying point farther from the margin can yield to a large loss. Therefore, the labeled outlying points – the samples that are misclassified outside the margin – start to play a dominant role in determining the separating hyperplane. As we mentioned earlier, labels can be very noisy in image retrieval applications since the labels are mostly collected from the surrounding text, which aggravates the problem. To ameliorate this drawback, we interchange the convex Hinge loss with a more robust non-convex Ramp loss function. The Ramp loss also bounds the maximum amount of loss similar to the symmetric Ramp loss function and this helps to suppress the influence of misclassified examples. The superiority of the Ramp loss over the Hinge loss for supervised SVM training is well-proven and demonstrated in Ertekin et al. (2011), so we adopt it to the transductive learning here.

After these revisions, our robust TSVM method solves the following problem

$$\begin{aligned}&\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} R_s(y_i(\mathbf{w}^\top\mathbf{x}_i + b))\\ &+C^*\sum_{i=L+1}^{L+U} SR_s(\mathbf{w}^\top\mathbf{x}_i + b)\\ &\text{s.t.} \frac{1}{U}\sum_{i=L+1}^{L+U}(\mathbf{w}^\top\mathbf{x}_i + b) = \frac{1}{L}\sum_{i=1}^{L} y_i.\end{aligned} \tag{4}$$

By using the equations $R_s(t) = H_1(t) - H_s(t)$ and $SR_s(t) = R_s(t) + R_s(-t)$, the above cost function without the balancing constraint can be written as

$$J(\theta) = J_{convex}(\theta) + J_{concave}(\theta), \tag{5}$$

where

$$\begin{aligned}J_{convex}(\theta) = &\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} H_1(y_i(\mathbf{w}^\top\mathbf{x}_i + b))\\ &+C^*\sum_{i=L+1}^{L+2U} H_1(y_i(\mathbf{w}^\top\mathbf{x}_i + b)),\end{aligned} \tag{6}$$

and

$$J_{concave}(\theta) = -C\sum_{i=1}^{L} H_s(y_i(\mathbf{w}^\top\mathbf{x}_i+b)) - C^*\sum_{i=L+1}^{L+2U} H_s(y_i(\mathbf{w}^\top\mathbf{x}_i+b)). \quad (7)$$

The above cost function (5) is not convex but it can be decomposed into convex (6) and concave (7) parts, so we can apply the concave-convex procedure (CCCP) (Yullie and Rangarajan, 2002) to solve the problem. By employing CCCP, the minimization of $J(\theta)$ with respect to $\theta = (\mathbf{w}, b)$ can be achieved by iteratively updating the parameter θ by the following rule

$$\theta^{t+1} = \arg\min_{\theta}(J_{convex}(\theta) + J'_{concave}(\theta^t)\theta), \quad (8)$$

under the constraint $\frac{1}{U}\sum_{i=L+1}^{L+U}(\mathbf{w}^\top\mathbf{x}_i+b) = \frac{1}{L}\sum_{i=1}^{L} y_i$.

After some standard derivations given in Appendix A (available at http://mlcv.ogu.edu.tr/pdf/appendix.pdf), the resulting final robust TSVM method can be summarized as in Algorithm 1. It should be noted that the optimization problem that constitutes the core of the CCCP is convex. Instead of taking dual of this convex problem and solving it with a dual QP solver as in Collobert et al. (2006), we consider the primal problem and use SG algorithm given in Algorithm 2 to solve it. Thus, the proposed method scales well with large-scale data. To initialize the method, we use supervised linear SVM trained with labeled data samples only.

Algorithm 1 The Robust Transductive Support Vector Machines (RTSVM).

```
Initialize $\theta^0 = (\mathbf{w}^0, b^0)$, $t = 0$, $\epsilon_1 > 0$, $\epsilon_2 > 0$
Compute
$\beta_i^0 = y_i \frac{\partial J_{concave}(\theta)}{\partial f_\theta(\mathbf{x}_i)} = \begin{cases} C, & \text{if } y_i((\mathbf{w}^0)^\top\mathbf{x}_i+b^0) < s \text{ and } 1 \le i \le L \\ C^*, & \text{if } y_i((\mathbf{w}^0)^\top\mathbf{x}_i+b^0) < s \text{ and } L+1 \le i \le L+2U \\ 0, & \text{otherwise.} \end{cases}$

  while $\|\mathbf{w}_{t+1} - \mathbf{w}_t\| \ge \epsilon_1$ or $\|\boldsymbol{\beta}_{t+1} - \boldsymbol{\beta}_t\| \ge \epsilon_2$ do
    – Solve the following convex minimization problem by using SG algorithm given in Algorithm 2
    $\arg\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} H_1(y_i(\mathbf{w}^\top\mathbf{x}_i+b)) + C^*\sum_{i=L+1}^{L+2U} H_1(y_i(\mathbf{w}^\top\mathbf{x}_i+b)) + \sum_{i=1}^{L+2U}\beta_i^t y_i(\mathbf{w}^\top\mathbf{x}_i+b)$
    such that $\frac{1}{U}\sum_{i=1}^{L+U}(\mathbf{w}^\top\mathbf{x}_i+b) = \frac{1}{L}\sum_{i=1}^{L} y_i$ ;
    – Set $\mathbf{w}^{t+1} = \mathbf{w}$, $b^{t+1} = b$;
    – Compute
    $\beta_i^{t+1} = \begin{cases} C, & \text{if } y_i((\mathbf{w}^{t+1})^\top\mathbf{x}_i+b^{t+1}) < s \text{ and } 1 \le i \le L \\ C^*, & \text{if } y_i((\mathbf{w}^{t+1})^\top\mathbf{x}_i+b^{t+1}) < s \text{ and } L+1 \le i \le L+2U \\ 0, & \text{otherwise.} \end{cases}$
    – Set $t = t+1$;
  end while
```

The proposed TSVM method is fast and it can scale well with large scale data since it was proved in Shalev-Shwartz et al. (2007) that run-time of SG does not depend directly on the size of the training set. There are only two sets of parameters that must be set by the user in the proposed classifier: $C(C^*)$ error penalty terms for labeled (unlabeled) data and step length λ of SG algorithm. Fixing error penalty parameters is very straightforward for anyone who is familiar with linear SVMs. Since the data samples are linearly separable because of high-dimensionality of the image feature vectors, we must set $C(C^*)$ to moderate values larger than 1. We set $C = 20$ and $C^* = 5$ for Fisher vectors and $C = 10$ and $C^* = 2$ for CNN features for all experiments. For fixing step length, we test the method on a small validation data for $\lambda \in \{0.1, 0.01, 0.001, 0.0001\}$, and set it to the value that yields the best accuracy.

Algorithm 2 Stochastic Gradient Based Solver with Projection.

```
Initialize
$\mathbf{w}_1$, $b_1$, $T > 0$, $\lambda_0 > 0$, $\epsilon > 0$
Description:
  for $t \in 1, \ldots, T$ do
    $\lambda_t \leftarrow \lambda_0 / t$;
    for $i \in \text{randperm}(L+2U)$ do
      – Compute sub-gradients
      $\mathbf{g}_t = \begin{cases} -y_i C(C^*)\mathbf{x}_i + \beta_i y_i \mathbf{x}_i, & \text{if } y_i(\mathbf{w}_t^\top\mathbf{x}_i+b_t) \le 1 \\ \beta_i y_i \mathbf{x}_i, & y_i(\mathbf{w}_t^\top\mathbf{x}_i+b_t) > 1. \end{cases}$
      $h_t = \begin{cases} -y_i C(C^*) + \beta_i y_i, & \text{if } y_i(\mathbf{w}_t^\top\mathbf{x}_i+b_t) \le 1 \\ \beta_i y_i, & y_i(\mathbf{w}_t^\top\mathbf{x}_i+b_t) > 1. \end{cases}$
      – Update hyperplane parameters
      $\tilde{\mathbf{w}_t} \leftarrow \mathbf{w}_t - \frac{\lambda_t}{L+2U}(\mathbf{w}_t + \mathbf{g}_t)$
      $\tilde{b_t} \leftarrow b_t - \frac{\lambda_t}{L+2U} h_t$
      – Project parameters onto the feasible set imposed by the constraint
      $(\mathbf{w}_t, b_t) = \mathcal{P}(\tilde{\mathbf{w}_t}, \tilde{b_t})$
    end for
    if $(t > 2) \& (\|\mathbf{w}_t - \mathbf{w}_{t-1}\| < \epsilon)$, break
  end for
```

2.2. Building class hierarchies

To build class hierarchies, we adopt the method we introduced in Cevikalp (2010). This method uses only labeled data to create class hierarchies. Assume that we are given some classes and corresponding labeled samples for each class (these are created by random selection of samples from each class or random selection of classes to create more independent hash functions). We use a binary hierarchical tree (BHT) that divides the image classes into two groups until each group consists of only one image class. In this setup, the accuracy depends on the tree structure that creates well-balanced separable image class groups at each node of the tree. To this end, we use the Normalized Cuts (NCuts) algorithm of Shi and Malik (2000) to split image classes into two groups (called positive and negative groups). In our case, we must split image classes (not the individual image samples) into two groups. Therefore, we need to replace image data samples with image data classes. Next, we must define a distance metric to measure the similarities between classes. This must be compatible with our goal, which is grouping classes with the largest margin possible. Therefore, we approximate each class with a convex hull and use the convex hulls distances between pair-wise image classes to create a similarity matrix. Convex hull distance between two classes can be found by using quadratic programming as described in Bennett and Bredensteiner (2000). It should be noted that convex hulls are largely used to approximate classes, e.g., the linear SVM uses convex hull modeling and the margin between two classes is equivalent to the geometric distances between convex hulls of classes. Thus, this similarity measure is compatible with our margin maximization goal, and efficient SVM algorithms

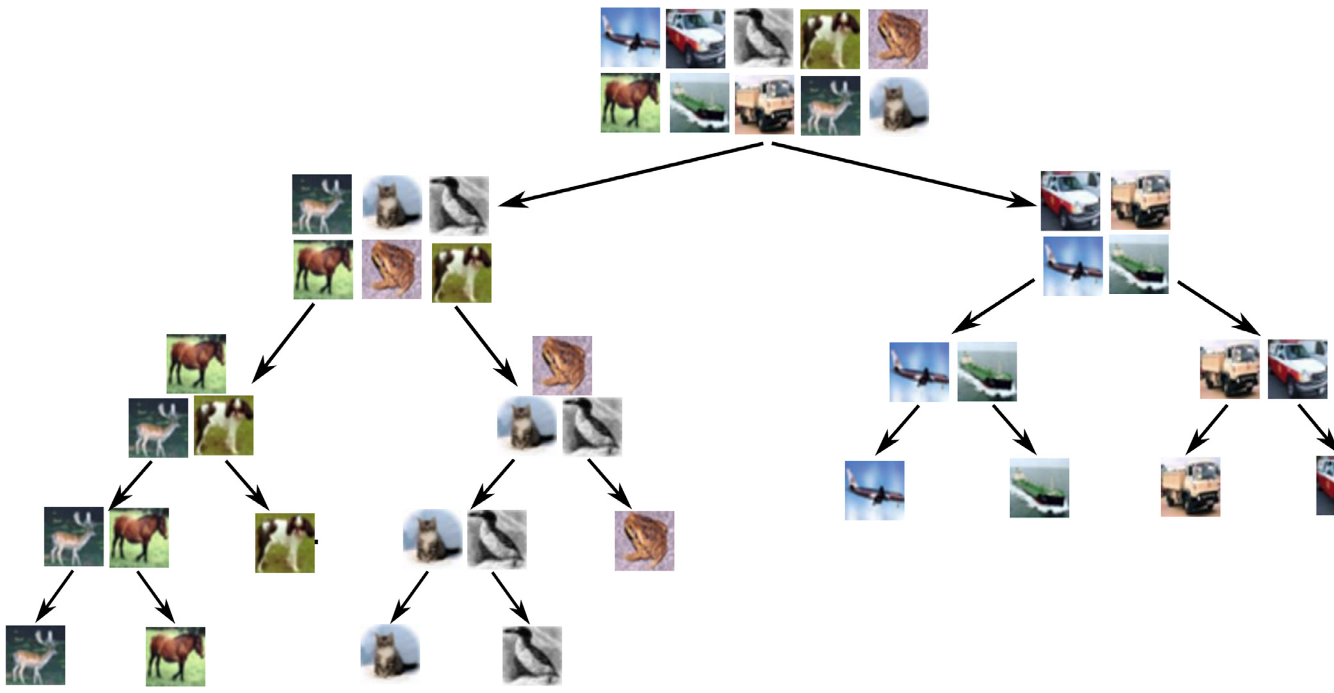


Fig. 2. Binary hierarchical tree obtained for CIFAR10 dataset using convex hull modeling of the classes. Each image represents an object class where it comes from.

can also be used to compute the distances between convex hulls. The distance is equivalent to $2/\|\mathbf{w}\|$, where $\mathbf{w}$ is the normal of the separating hyperplane returned by linear SVM classifier.

In this setting, the edges, w_{ij}, of the similarity matrix $\mathbf{W}$ is computed as

$$w_{ij} = \begin{cases} \exp(-d(H_i^{convex}, H_j^{convex})/t), & \text{if } i \neq j \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

where H_i^{convex} represents the convex hull of class i, $d(.)$ is the distance between convex hulls, and t is the width of the Gaussian kernel function that must be set by the user. Note that the size of the similarity matrix is $C \times C$ where C is the number of classes. Thus, it is a much smaller sized matrix compared to other methods (mentioned at Introduction) using individual image samples. Then, we cluster the image classes into two groups by solving the generalized eigenvalue problem

$$\mathbf{La} = \lambda \mathbf{Da}, \tag{10}$$

where $\mathbf{L} = \mathbf{D} - \mathbf{W}$ is the Laplacian matrix and $\mathbf{D}$ is a diagonal matrix whose entries are the column (or row) sums of $\mathbf{W}$. Finally, the components of the eigenvector $\mathbf{a}^*$ corresponding to the second smallest eigenvalue of (10) are thresholded to split image classes into two clusters, i.e.,

$$\begin{cases} y_i = -1, & \text{if } a_i^* \geq 0 \\ y_i = +1, & \text{if } a_i^* < 0 \end{cases} \tag{11}$$

Fig. 2 illustrates the hierarchy obtained for 10-classes of CIFAR10 dataset. At the top node, it successfully separates man-made vehicles (airplane, automobile, ship, and truck) from the animals (bird, cat, deer, dog, frog, and horse). It also successfully groups visually similar objects such as automobile-truck, deer-horse, and airplane-ship together. So, our method produces both well-separated and well-balanced groups of classes, which is crucial for successful balanced binary hash codes. It should be noted that this hierarchy is obtained automatically just by using the image feature samples and their labels, therefore the classes grouped together are visually similar but not necessarily semantically related. For example, cat class is more related to dog class semantically, but cat and bird classes are grouped together in the figure since the appearances of training examples of cat and bird classes are more similar in Cifar10 dataset. Similarly, the dog class is also grouped together with horse and deer classes based on the appearances. It should be noted that the groupings may change by changing training examples. Thus, the hierarchy obtained by using Wordnet in Fergus et al. (2010) does not guarantee the best separability for a given specific image data, which is crucial for the success of the proposed method.

As mentioned earlier, more than one label can be assigned to image samples, e.g., assume that an image sample contains both *people* and *car*. In such cases, we treat the groups with multiple labels as a new category and manually set the similarities between the related classes (*people* and *car* classes) to the maximum score 1. This makes sure that these related classes are grouped together at the upper nodes. By doing so, we postpone to separate these related classes by grouping them as similar classes. So, they appear at the lower nodes of the class hierarchy where we can do a finer separation between them. For separation we use TSVM as before.

There are also methods using one-against-rest (OAR) regime for creating class hierarchies. For example, Griffin and Perona (2008), Bengio et al. (2010), and Bergamo and Torresani (2014) first use OAR regime to train SVM classifiers and evaluate them on a validation set to create a confusion matrix. Then, this confusion matrix is treated as an affinity matrix and spectral clustering (which uses same principle for clustering as in NCuts) is used to partition the classes. However, OAR does not guarantee balanced and separable groups of classes since the separability of classes will decrease as the number of classes is increased when OAR regime is used for multi-class classification. This is illustrated in Fig. 3. All pair-wise classes are linearly separable in the figure, and our proposed method successfully groups classes A, B and C in one group and classes D, E and F in another group at the top node of the hierarchy by using pair-wise distances based on convex hull models (this is optimal grouping in terms of both margin and balance of the groups). However, if OAR regime is used, classes are not linearly separable anymore and class B will be confused with class D. In a similar manner, class A will be confused with

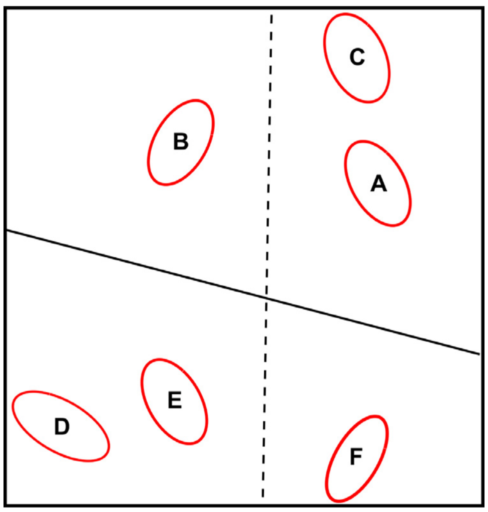


Fig. 3. Comparison of groupings based on OAO and OAR regimes. Using pair-wise distances between classes achieves the optimal grouping of classes in terms of margin (separation shown with the solid line), but OAR regime yields to a sub-optimal group (shown with the dashed line).

C and E will be confused with D. Therefore, the methods given in Griffin and Perona (2008), Bengio et al. (2010) and Bergamo and Torresani (2014) will group classes B, D and E in one group and classes A, C and F in another group, which is not optimal in terms of the largest margin between the groups. Another disadvantage of the methods given in Griffin and Perona (2008), Bengio et al. (2010) and Bergamo and Torresani (2014) is that the spectral clustering or NCuts requires a symmetric affinity matrix. Our proposed methodology already returns a symmetric affinity matrix since the distance between convex hulls of pair-wise classes is same. In contrast, the confusion matrix is not necessarily symmetric. This problem is avoided by summing the confusion matrix with its transpose and dividing by two, i.e., $\widetilde{\mathbf{C}} = (\mathbf{C} + \mathbf{C}^\top)/2$. As a result, the affinity matrix computed from a confusion matrix does not reflect the true similarities between classes. There are also some methods (Vural and Dy, 2004; Zhigang et al., 2005) that use class means to group classes. But, these methods are also not compatible with margin maximization goal, and they may not yield to balanced and separable class groups.

2.3. Creating binary hash codes

Once we split image classes into two groups at each node of the BHT, we run TSVM algorithm by using both labeled and randomly chosen unlabeled data to find the separating hyperplanes. Then these hyperplanes can be used in two ways to produce hash codes. In the first place, we can use the following rule to create hash codes directly

$$h_i(\mathbf{x}) = \begin{cases} 1 & \text{when } \mathbf{w}_i^\top \mathbf{x} + b_i \geq 0 \\ 0 & \text{when } \mathbf{w}_i^\top \mathbf{x} + b_i < 0 \end{cases} \tag{12}$$

where $\mathbf{w}_i$s are the returned hyperplane normals and b_is are the corresponding bias parameters. Each BHT produces $C-1$ hash functions where C is the number of classes used to build a single BHT. So, the total number of hash bits will be $C-1$ times the number of BHTs. As a second choice, we can use hyperplane normals to embed the data samples onto a more discriminative space and then use an Euclidean distance preserving hashing method or a quantizer (e.g., LSH or Product Quantization of Jegou et al., 2010) or any other successful hashing method in the embedded space (this can be seen as metric learning followed by using a hashing method that approximates the learned distance metric). In the experiments below, we used hyperplanes directly to create binary codes when the number of classes is small – smaller than 32 so that we can create 32 bits hash codes. But, for some datasets, the number of classes is larger than 32, therefore we used hyperplane normals to embed the data in such cases. When we use the hyperplanes to create binary hash codes directly, the bit size determines the number of trees. For example, for a 10-class dataset, we need at least 4 BHTs to create a 32-bit code since each BHT will return $C-1=9$ hyperplanes. To determine the number of BHTS that will be used for embedding, we simply trained 40 BHTs. Then, we set the final BHT number to the value for which the accuracy starts to saturate. Lastly, we use Hamming distance to find the distances between hash codes, but weighted Hamming distances using hierarchy or classifier margin can be also used for this goal.

We would like to point out that the diversity of BHTs is the most important factor for good results as in randomized forests. Therefore, we have to ensure that each BHT is quite different than the others. Therefore, we do not use all labeled data to train each BHT (this is similar to train weak classifiers instead of strong classifiers in random forests). For example, using many labeled data gave better accuracies for 32 bit hash codes at first, but then the improvement got easily saturated as the number of bits is increased. In contrast, using less labeled data yielded lower accuracies for 32 bits, but achieved the best accuracies as the number of bits is increased (this is the main reason why we obtained lower accuracy for 32 bit hash codes on some tested datasets).

Lastly, in the proposed method, we prefer BHT based hierarchy rather than the one using one-against-one (OAO) regime for two reasons: First, BHT provides an additional coarse information about the semantic similarity of the visual classes as demonstrated in Fig. 2 compared to OAO. Second, our proposed strategy used in BHT returns $C-1$ hyperplanes and consequently a single tree creates $C-1$ bit hash codes whereas a single tree using OAO regime will produce $C(C-1)/2$ bit hash codes. For example, for a dataset with 100 classes, a single BHT using the proposed strategy produces 99 bit hash codes, but a single BHT using OAO will produce 4950 bit hash codes. As we mentioned above, we have to create diverse BHTs for good accuracies. Therefore, it is not possible to create small hash codes with OAO regime.

2.4. Training complexity

There are basically two computationally demanding steps of training phase of the proposed method: creating similarity matrix for BHTs and training a TSVM classifier at each node of BHTs. Only labeled data are used for construction of similarity matrix, and this requires solving $C(C-1)/2$ quadratic programming (QP) problems since we have to compute all possible pair-wise convex hull distances out of C classes. We use an efficient SVM solver (LIBOCAS - http://cmp.felk.cvut.cz/˜xfrancv/ocas/html/) to compute convex hull distances and the complexity is related to $O(dn)$ where d is the dimensionality and n is the number of training samples. It should be noted that we solve small binary optimization problems in this step and this is usually fast if the number of classes is not very high. The total number of QP problems grows quadratically with the number of classes, and hence, may become prohibitively expensive when C is very large. To solve TSVM problem at each node of BHTs, we use SG algorithm. It was shown in Shalev-Shwartz et al. (2007) that total run time complexity of SG algorithm is given by $O(f/(\lambda\epsilon))$, where f is a bound on the number of non-zero features, λ is the regularization parameter. Therefore, the run-time does not directly depend on the size of the training set. In practice, we scan through all training samples a few hundred times during SG algorithm, so complexity is again related to $O(dn)$ in the worst case. We solve $C-1$ TSVM problems for each BHT, and TSVM classifiers use more data at upper nodes of BHTs and less data at the lower nodes. As an important note, we should be kept in mind that BHTs can be learned in parallel since each BHT is learned independent of others.

3. Experiments

Here, we conduct image retrieval experiments on five datasets. We compared the proposed hashing method, TSVMH-BHT (Transductive Support Vector Machine Hashing using Binary Hierarchical Tree),[2] with both the supervised and unsupervised hashing methods: LSH (Gionis et al., 1999), PCA-RR (Principal Component Analysis - Random Rotations) (Gong et al., 2013b), PCA-ITQ (Principal Component Analysis - Iterative Quantization) (Gong et al., 2013b), SKLSH (Shift-Invariant Kernel Locality Sensitive Hashing) (Raginsky and Lazebnik, 2009), SH (Spectral Hashing) (Weiss et al., 2008), SHD (Spherical Hamming Distance) (Heo et al., 2012), SDH (Supervised Hashing) (Liu et al., 2012), and Supervised Trees of Lin et al. (2014). In addition to these hashing methods we also report the results obtained using PQ (Product Quantization) method of Jegou et al. (2010) as a baseline. For some datasets, we also give the best reported accuracies of recent hashing methods using deep neural networks. All accuracies are given in terms of mAP (mean average precision) scores which is the most common accuracy measure for image retrieval.

3.1. Experiments on cifar100 dataset

Cifar100 dataset (available at http://www.cs.toronto.edu/~kriz/cifar.html) includes 50K 32 × 32 small images of 100 classes. There are 500 samples per class in the training set, and the test set includes 10K samples. We used both Fisher vectors (FVs) and 4096-dimensional Convolutional Neural Network (CNN) features. We used a similar setup as in Sanchez et al. (2013) to extract FVs. More precisely, we extracted many descriptors per image from 12 × 12 patches on a regular grid every one pixel at 3 scales. The dimensionality of the tested descriptors is reduced to 80 by using PCA, and 128 components are used in Gaussian mixture model. To extract CNN features, all images are resized to 256 × 256 and we applied fine-tuning since direct training with available data yielded lower accuracies than fine-tuning. Pre-trained Caffe model for ILSVRC 2012 dataset is used to initialize the weights. We used 80% of the full training data for training and the remaining 20% for validation to train the CNN classifier with fine-tuning. The number of iterations is set to 180K. We trained 15 hierarchical trees for FVs and 17 trees for CNNs, and we randomly selected 300 labeled data for each class and used the remaining 200 samples as unlabeled data for each BHT.

Accuracies are given in Table 1. The mAP scores are obtained by using top 500 returned images as a function of code size. Euclidean distance in the original input space yields to 11.66% mAP for FVs and 51.50% for CNNs. Note that the number of classes is 100 so a single tree will yield to a 99-bit hash code. Therefore, we used the hyperplane normals returned by TSVM to embed the data onto 1485 (15 × 99)-dimensional space for FVs and onto 1683 (17 × 99)-dimensional space for CNNs. Then, we applied ITQ, SDH and Supervised Trees to the embedded data to create 32, 64, 128 and 256-bit hash codes (these methods are respectively denoted by BHDT+ITQ, BHDT+SDH and BHDT+Supervised Trees in Table 1). Embedding the data to this lower dimensional space improves the Euclidean distance accuracy to 17.26% mAP for FVs and 58.80% for CNNs, which clearly shows that the proposed method improves the Euclidean distance metric. The best accuracies are obtained by the hashing methods applied to the embedded data learned by using the proposed method for all cases. TSVMH-BHT+Supervised Trees method dominates the results for both FVs and CNNs. TSVMH-BHT+Supervised Trees improves the accuracies over Supervised Trees up to 15.78% for FVs and 2.95% for CNNs. The improvement brought by the proposed method in accuracy is more significant when FVs are used as features.

Table 1
mAP Scores (%) for Cifar100 dataset using FVs and CNNs.

FVs	32 bits	64 bits	128 bits	256 bits
TSVMH-BHT+SDH	13.44	17.00	19.48	21.46
TSVMH-BHT+Sup. Trees	**24.86**	**27.52**	**29.15**	**29.43**
TSVMH-BHT+ITQ	11.48	13.98	16.24	29.43
SDH	6.33	7.45	8.95	9.81
Supervised Trees	9.08	13.23	17.61	22.66
LSH	5.43	7.34	8.12	10.23
SH	5.55	7.24	8.86	10.07
SHD	3.83	4.57	4.85	4.95
SKLSH	1.82	2.21	2.92	3.76
PCA-ITQ	6.62	8.43	9.84	10.77
PCA-RR	6.50	8.24	9.52	10.46
CNN	**32 bits**	**64 bits**	**128 bits**	**256 bits**
TSVMH-BHT+SDH	51.23	58.58	61.13	63.53
TSVMH-BHT+Sup. Trees	**69.88**	**71.67**	**72.39**	**72.35**
TSVMH-BHT+ITQ	46.56	60.68	66.18	68.02
SDH	49.26	58.48	60.83	62.16
Supervised Trees	66.93	69.30	71.76	72.12
LSH	42.31	47.65	48.13	49.21
SH	28.86	39.55	45.10	48.54
SHD	30.64	40.67	42.93	42.91
SKLSH	9.40	12.48	26.18	33.75
PCA-ITQ	46.30	57.86	62.90	63.77
PCA-RR	42.42	51.48	56.94	57.99

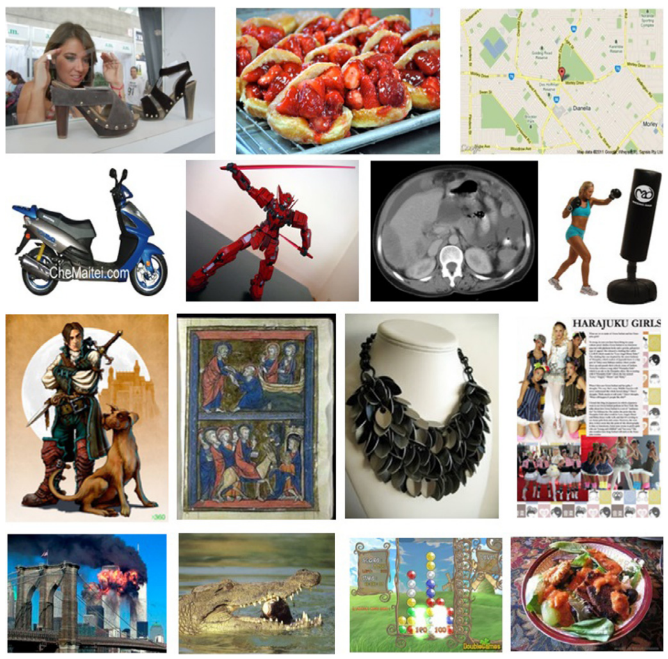


Fig. 4. Some image samples from ImageCLEF 2013 dataset.

3.2. Experiments on ImageCLEF 2013 Dataset

We created a new multi-label image retrieval dataset by using ImageCLEF 2013 dataset (http://imageclef.org/2013). This dataset includes approximately 26 million images (we could download 23,415,941 images), where there are very noisy tags for 250K images collected by using search engine information and the text surrounding the images. Some images are shown in Fig. 4. Only

[2] Our code is available at http://mlcv.ogu.edu.tr/softwarelsir.html.

a small amount of images (3000 images) are hand-labeled. The main goal of ImageCLEF challenge is to improve image annotation. In our study, to be able to use ImageCLEF dataset as an image retrieval data set, three graduate students went through noisy labels and corrected the noisy tags of 120,741 images in four months. These images included 1662 different labels (classes), and each image may contain multiple labels to represent the image content. In our experiments, we used the images associated to the most frequent 100 labels. Therefore, the final size of the training images is 76,138 and the test set size is 25816. In addition, we also added randomly selected 50K unlabeled images to the training set to train TSVM classifiers and other unsupervised methods.

We again used fisher vectors (FVs) and 4096 dimensional CNN features to represent images. We used the same set up as in the previous experiment to extract FVs. To extract CNN features, we tried several things: First, we trained a CNN classifier network by using images with a single label following the same network structure described by Krizhevsky et al. (2012). Second, we directly used a pre-trained Caffe model trained for ILSVRC 2012 classification. CNN features obtained by both methods yield to lower accuracies than FVs. Therefore, we finally applied fine-tuning to the images with single labels, and a pre-trained Caffe model for ILSVRC 2012 dataset has been used to initialize the weights. We used 80% of the full training data for training and the remaining 20% as validation during fine-tuning. This procedure achieved better accuracies than FVs, so we report the results obtained by this setting. Since this is a multi-label dataset, two images are considered as a true match if they share at least one common label as in Lai et al. (2015), Xia et al. (2014), Zhang et al. (2016) and Liu et al. (2011b). It should be noted that the total training set size is large, and the supervised hashing methods and unsupervised hashing methods that build dense similarity (or kernel) matrix cannot be used directly. Therefore, randomly chosen anchor points are used in SDH (Liu et al., 2012). The default value for the number of anchor points is 300 for SDH, but we increased it to 500 for better accuracies. For the proposed method, we trained 35 hierarchical trees for FVs and 25 trees for CNNs and we randomly selected at most 1500 labeled examples per class and 50K unlabeled samples for each hierarchical tree. Test samples are not used as unlabeled data during training.

Results are given in Table 2. The mAP scores are obtained by using top 500 returned images as a function of code size. Euclidean distance in the original input space yields to 24.54% mAP for FVs and 36.26% for CNNs. Note that the number of classes is 100 so a single tree will yield to a 99-bit hash code. Thus, we used the hyperplane normals returned by TSVM to embed the data onto 3465 (35×99)-dimensional space for FVs and onto 2475-dimensional space for CNNs. Then, we applied ITQ, SDH and Supervised Trees to the embedded data to create 32, 64, 128 and 256-bit hash codes as in the previous experiment. The best accuracies are obtained by the hashing methods applied to the embedded data learned by using the proposed method for all cases except 256 bit hash codes when CNNs are used as visual features. For FVs, TSVMH-BHT+Supervised Trees method dominates the results whereas TSVMH-BHT+ITQ achieves the better results for CNNs. The improvement brought by the proposed method in accuracy is more significant when FVs are used as features as in the first experiment.

Table 2
mAP Scores (%) for ImageCLEF 2013 dataset using FVs and CNNs.

FVs	32 bits	64 bits	128 bits	256 bits
TSVMH-BHT+SDH	**24.46**	26.49	28.00	29.10
TSVMH-BHT+Sup. Trees	22.73	**28.28**	**32.97**	**37.27**
TSVMH-BHT+ITQ	20.96	22.63	24.14	25.10
SDH	23.21	25.09	26.50	27.33
Supervised Trees	19.70	23.32	28.66	33.53
LSH	16.67	18.23	19.45	20.54
SH	19.08	19.69	20.70	21.44
SHD	19.83	21.06	21.24	21.70
SKLSH	11.93	12.78	14.32	15.47
PCA-ITQ	20.57	22.01	22.93	23.68
PCA-RR	20.25	21.45	22.72	23.41
PQ	20.75	21.54	23.72	23.87
CNN	**32 bits**	**64 bits**	**128 bits**	**256 bits**
TSVMH-BHT+SDH	40.31	43.10	44.83	45.80
TSVMH-BHT+Sup. Trees	31.55	38.02	43.16	46.38
TSVMH-BHT+ITQ	**41.48**	**44.30**	**45.69**	46.09
SDH	40.21	42.18	44.74	45.10
Supervised Trees	32.45	40.64	44.40	**46.78**
LSH	36.42	38.67	40.43	41.20
SH	25.46	30.73	33.93	34.43
SHD	29.10	32.74	34.17	34.44
SKLSH	14.11	15.98	19.86	23.84
PCA-ITQ	40.94	43.92	45.29	45.79
PCA-RR	38.42	41.41	43.36	44.07
PQ	38.67	40.61	44.25	44.87

3.3. Experiments on NUS-WIDE dataset

The NUS-WIDE dataset has approximately 270K images collected from Flickr. Each image is annotated with one or multi labels in 81 semantic classes. To compare our results to the literature, we follow the setting in Lai et al. (2015), Xia et al. (2014), Zhang et al. (2016) and Liu et al. (2011b) and we use the images associated with the 21 most frequent labels. The number of final training images is 96,638 and the number of test images is 64704. Two images are considered as a true match if they share at least one common label as in Lai et al. (2015), Xia et al. (2014), Zhang et al. (2016) and Liu et al. (2011b). We used both FVs and CNN feature vectors for representing images. To obtain CNN features we first resized images to 256×256 as before and we trained a CNN classifier network by using the same network structure described by Krizhevsky et al. (2012). Note that this is a multi-label dataset, thus we selected images with non-overlapping unique labels to train the classifier network. But, results were very low compared to the ones obtained for FVs. We believe that the results were low mainly because the loss function is not designed for multi-label data. The noisy labels was another reason for the low accuracy. Note that there are recent attempts (Gong et al., 2013a) to design loss functions suitable for multi-label data but this topic is beyond the scope of our study. Thus, we used pre-trained Caffe model of ILSVRC 2012 to extract 4096-dimensional CNN features, and this procedure yielded better results than FVs.

Results are given in Table 3. We used at most 7000 labeled samples per class and 20K unlabeled samples to train each hierarchical tree. Test samples are not used as unlabeled samples in training phase. The separating hyperplanes returned by the TSVM classifiers are directly used to create hash codes. Using Euclidean distance with full features yields to 52.20% mAP for FVs and 69.65% for CNN features. As in the previous case, hashing codes obtained for CNN features yield to better accuracies. The proposed method achieves the best accuracies in all cases except for 32 bits and significantly improves the Euclidean distance metric for 64 bits and above. Our results are even better than the ones obtained by recent deep neural networks. To the best our knowledge, our results are the best published mAP scores for NUS-WIDE dataset. The performance difference is very significant for FVs. We believe that our better accuracies compared to the other state-of-the-art methods are due to using the robust Ramp loss for noisy labels.

3.4. Experiments on CIFAR10 dataset

Cifar10 dataset includes 60K 32×32 small images of 10 objects: airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. 50K samples are used as training and they are split

Table 3
mAP Scores (%) for NUS-WIDE dataset using FVs and CNN features.

FVs	32 bits	64 bits	128 bits	256 bits
TSVMH-BHT	53.97	**57.73**	**61.76**	**63.99**
SDH	**54.20**	56.41	58.14	58.93
Supervised Trees	46.14	51.87	57.13	60.89
LSH	30.25	36.27	39.50	43.20
SH	46.31	45.97	47.60	48.51
SHD	47.08	50.09	52.41	52.89
SKLSH	33.24	35.25	36.50	37.68
PCA-ITQ	50.41	51.95	52.63	53.17
PCA-RR	49.89	51.17	52.09	52.95
PQ	30.75	31.74	33.49	40.19
CNN	**32 bits**	**64 bits**	**128 bits**	**256 bits**
TSVMH-BHT	68.58	**72.90**	**74.64**	**76.21**
SDH	69.09	72.41	73.71	74.75
Supervised Trees	62.98	67.90	71.09	73.44
LSH	66.78	68.91	69.47	69.50
SH	58.03	60.08	61.31	64.41
SHD	61.60	64.23	64.83	64.66
SKLSH	39.65	43.37	47.47	53.65
PCA-ITQ	68.62	70.91	72.48	73.55
PCA-RR	65.82	68.27	70.57	71.60
PQ	**71.07**	71.93	72.20	72.17
Lai et al. (2015)	**71.30**	–	–	–
Xia et al. (2014)	62.90	–	–	–
Zhang et al. (2015)	62.64	63.82	–	–
Zhang et al. (2016)	≈52.0	≈52.5	≈54.0	–
	12 bits	**24 bits**	**36 bits**	**48 bits**
Liu et al. (2016)	54.80	55.10	55.80	56.20

Table 4
mAP Scores (%) for CIFAR 10 dataset using FVs and CNN features.

FVs	32 bits	64 bits	128 bits	256 bits
TSVMH-BHT	**46.74**	**51.37**	**53.94**	**55.10**
SDH	31.66	33.52	34.62	35.36
Supervised Trees	44.61	48.73	51.84	53.78
LSH	19.43	20.57	20.53	21.57
SH	19.60	20.43	21.44	21.86
SHD	19.47	21.91	24.09	25.77
SKLSH	11.10	11.38	11.93	12.68
PCA-ITQ	24.59	26.03	27.34	28.05
PCA-RR	23.07	24.53	25.83	36.76
PQ	24.30	23.80	22.90	22.00
CNN	**32 bits**	**64 bits**	**128 bits**	**256 bits**
TSVMH-BHT	79.97	81.89	82.45	82.79
SDH	83.05	83.59	83.77	83.83
Supervised Trees	**84.62**	**84.89**	**85.07**	**85.17**
LSH	73.98	74.87	75.56	76.40
SH	65.74	67.15	65.04	60.52
SHD	65.41	66.38	64.83	64.21
SKLSH	40.79	56.33	64.17	67.85
PCA-ITQ	80.78	81.27	81.86	82.05
PCA-RR	74.19	77.15	78.46	79.15
PQ	79.88	80.30	80.40	80.65
Lai et al. (2015)	55.80	–	–	–
Xia et al. (2014)	52.10	–	–	–
Zhang et al. (2015)	62.53	62.81	–	–
Zhang et al. (2016)	≈**86.0**	≈**86.0**	≈**86.0**	–
	12 bits	**24 bits**	**36 bits**	**48 bits**
Liu et al. (2016)	61.60	65.10	66.10	67.60

into 5 batches whereas the remaining 10K samples are used for testing. We used 16,384 dimensional fisher vectors (FVs) and 4096 dimensional CNN features as before. To extract CNN features, all images are first resized to 256 × 256 and then we used Caffe (Jia et al., 2014) implementation of the CNN described by Krizhevsky et al. (2012) by using the identical setting used for ILSVRC 2012 classification with the exception that the base learning rate was set to 0.001. We used 80% of the full training data for training and the remaining 20% as validation to train the CNN classifier. The number of iterations is set to 120K.

For all methods, we used the full training data to create hash functions, but we use only the samples in each batch to find the Hamming distances from the test samples. So, the results are averages over the results of 5 trials obtained for each training batch. We used randomly chosen 600 labeled samples and 900 unlabeled samples from each class to train our method and the separating hyperplanes returned by the TSVM classifiers are directly used to create hash codes. Both the labeled and unlabeled images are chosen only from the training set. The default value for the number of anchor points is set to 500 for SDH.

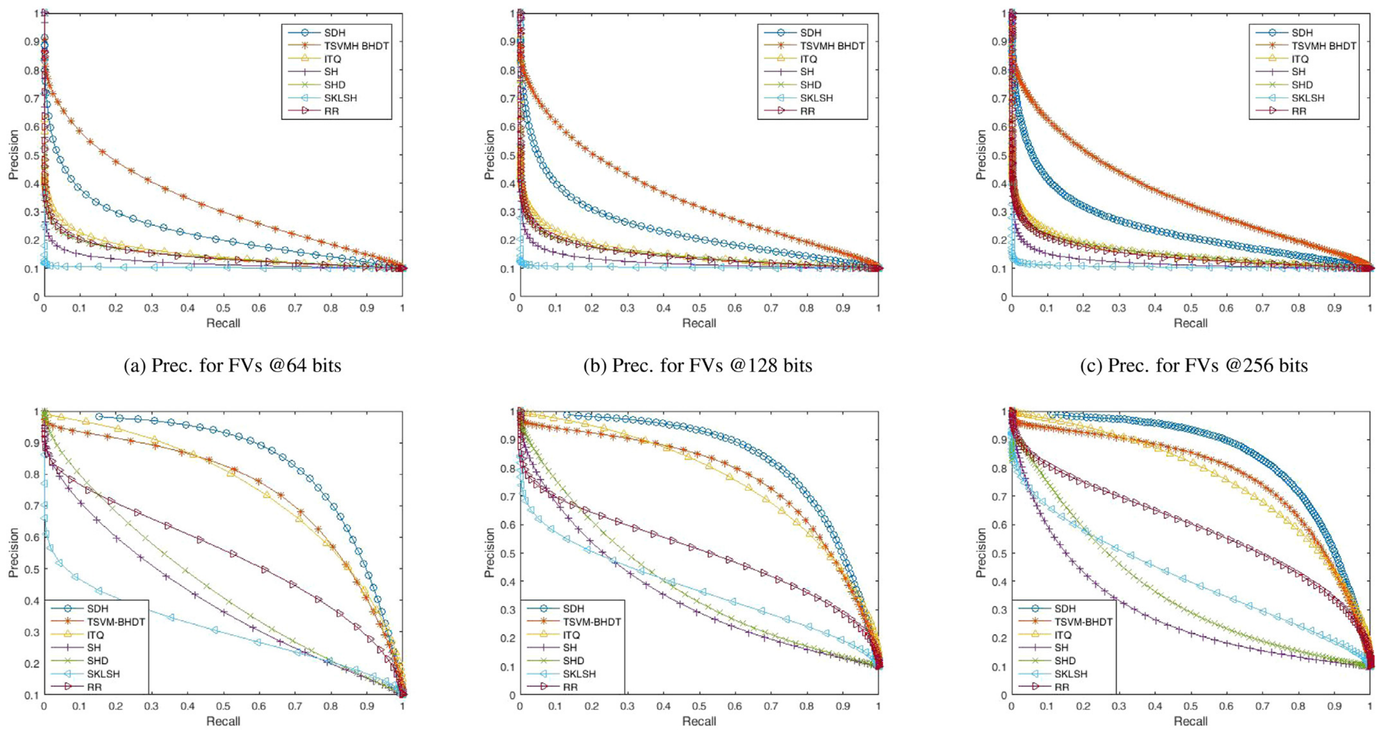
The mAP (mean Average Precision) scores using class labels as ground truth are given in Table 4 and Fig. 5 illustrates Precision curves obtained for different bit sizes. The mAP scores are obtained by using top 500 returned images as a function of code size as in Gong et al. (2013b). Gong et al. (2013b) also reports mAP scores using the Euclidean distances as ground truth, but this is wrong in our opinion since the performance of the Euclidean distance is very poor: Euclidean Distance in the original input space yields to 28.22% mAP for FVs and 76.90% for CNNs. As can be seen in Table 4, supervised hashing methods (TSVMH-BHT, SDH and Supervised Trees) always outperform unsupervised ones. The proposed method achieves the best accuracy for FVs whereas Supervised Trees produces the best results for CNNs. For FVs, the mAP score of the proposed method is approximately 20% better than the third best method SDH when 256 bits are used, which undoubtedly shows that the proposed method is better suited for high-dimensional visual image representations. Table 4 also shows some recently reported accuracies obtained using deep neural networks in the literature. It should be noted that the CNN features we extracted are different than the ones obtained by the methods given under PQ method in Table 4 since these methods are trained end-to-end fashion. Yet our proposed method outperforms all of them except the one given in Zhang et al. (2016). Supervised Trees beats our proposed method, but it should be noted that creating hash codes of a single test example in Supervised Trees takes a lot of time since the hash codes are not created by simple dot products as in the proposed method, e.g., it takes approximately 9 s to create a 32-bit hash code of a single test image for FVs with Supervised Trees, and it takes 3 s to create a hash code for CNNs. The unsupervised PCA-ITQ also works well. Lin et al. (2015) reports 89.4% mAP accuracy for Cifar10 dataset. We verified this result by using their pre-trained models. But, their training files show that they used test data as validation data to train the CNN network, which is a violation of a fair testing procedure. Thus, we omitted this result in our table.

As mentioned in Section 2.3, the diversity of BHTs is an important factor for good accuracies. When we used 800 labeled samples instead of 600, we obtained an accuracy of 48.7% which is higher than 46.7% for 32 bit hash codes. However, as the number of bits increased, we obtained accuracies of 50.3%, 52.1% and 52.7% for 64, 128 and 256 bit hash codes respectively. These results show that we need more diverse BHTs and correspondingly weaker classifiers for better accuracies.

3.5. Experiments on MNIST dataset

The MNIST[3] digit dataset consists of 70K hand-written digit samples, each of size 28 × 28 pixels. For this dataset, 60K samples are allocated for training and the remaining 10K samples are reserved for test. We use gray-scale values as visual features, thus dimensionality of the sample space is 784. We use randomly chosen 5000 labeled samples and 800 unlabeled samples from each class to train the proposed method. It should be noted that test samples are not used as unlabeled samples during training. Results

[3] available at http://yann.lecun.com/exdb/mnist/.



(a) Prec. for FVs @64 bits

(b) Prec. for FVs @128 bits

(c) Prec. for FVs @256 bits

(d) Prec. for CNN @64 bits

(e) Prec. for CNN @128 bits

(f) Prec. for CNN @256 bits

Fig. 5. Comparisons of the hashing methods on CIFAR10 dataset using labels as ground truth.

Table 5
mAP Scores (%) for MNIST Digit dataset.

Methods	32 bits	64 bits	128 bits	256 bits
TSVMH-BHT	87.68	89.15	89.07	89.13
SDH	81.29	85.37	86.10	86.26
Supervised Trees	**95.96**	**96.32**	**96.52**	**96.70**
LSH	72.15	74.40	79.30	82.43
SH	70.44	72.28	73.87	74.15
SHD	67.86	74.06	75.98	76.58
SKLSH	31.36	47.12	63.04	74.08
PCA-ITQ	79.55	83.37	85.50	86.23
PCA-RR	67.91	75.38	79.16	83.05
PQ	85.40	85.40	85.20	85.40

are given in Table 5. Euclidean Distance in the original input space yields to 85.95% mAP score which is quite satisfactory. Yet, our proposed method gives better accuracies than NN for all bit sizes. The best accuracies are obtained by Supervised Trees followed by the proposed method and the performance difference is quite significant. This shows that the decision boundaries are not linear, thus one needs to use kernel functions for better accuracies. Although Supervised Trees achieves the best accuracies, the testing time is very slow. For example, it takes respectively 3.0, 5.9, 10.9 and 15.0 s to build the hash code of a single digit image with the Supervised Trees whereas it takes around 4 ms to build a 256-bit hash code with the proposed method and the other tested hashing methods since these methods use a simple dot product to create the codes. PQ approximates NN using the original Euclidean space well and it achieves similar accuracies. However, due to the lack of semantic information, increasing code size does not effect the accuracy.

We also would like to point out that the Supervised Trees method is not suitable for very high-dimensional data as claimed in Lin et al. (2014). Note that, this method is proposed to achieve non-linearity in hashing and it is used to approximate nonlinear kernels such as high-order polynomial functions or Gaussian kernels. In high-dimensional datasets, e.g. data sets using FV representations, image classes are already separable with large margin and there is no need to use nonlinear kernels. As a result, Supervised Trees mostly failed to produce good results compared to the proposed method as demonstrated in our previous experiments. In contrast, Supervised Trees are more suitable for lower-dimensional data sets where the image classes cannot be separated with linear hyperplanes. This is also clear in these experiments where Supervised Trees significantly outperformed all other hashing methods.

4. Conclusion

In this study, we discussed the fact that the Euclidean distances in the high-dimensional feature spaces can be misleading, thus hashing methods approximating the Euclidean distances may perform poorly. To counter this, we proposed a hashing method that does both metric learning and fast image search. To this end, we used binary hierarchical trees and TSVM classifier. We proposed a more robust TSVM method which is designed for especially image retrieval applications. Using TSVM is extremely important here since it also exploits the unlabeled data that is neglected by many related hashing and distance metric learning methods.

We tested the proposed method on five image retrieval datasets. The results with high-dimensional FV features were particularly promising: Our method significantly outperformed all other tested hashing methods with FVs with a few exceptions. We also obtained state-of-the-art results using CNN features on noisy labeled NUS-WIDE dataset which shows the importance of using our robust Ramp loss function. In addition, we compared the proposed method to the recently published hashing methods using deep neural networks. These methods emphasize the importance of simultaneous learning of image features and binary codes, yet the results prove that this issue has not been resolved yet since majority of these deep neural networks methods yield very low accuracies compared to our method which learns hash codes independently from the pre-computed CNN features.

Lastly, we can directly use hyperplane normals returned by TSVM classifiers to create binary hash codes when the number of classes is not very large. If the number of classes is large, we can use hyperplane normals to embed data onto a lower-dimensional space that reflects semantic relations, and a different hashing method can be applied to the embedded data. We tested ITQ, SDH and Supervised Trees methods on embedded data obtained by the proposed method, and Supervised Trees typically produced the best results.

Acknowledgment

This work has been supported by the Scientific and Technological Research Council of Turkey (TUBİTAK) under Grant number TUBITAK-113E118.

References

Bengio, S., Weston, J., Grangier, D., 2010. Label embedding trees for large multi-class tasks. NIPS.
Bennett, K.P., Bredensteiner, E.J., 2000. Duality and geometry in svm classifiers. In: International Conference on Machine Learning.
Bergamo, A., Torresani, L., 2014. Classemes and other classifier-based features for efficient object categorization. IEEE Trans. Pattern Anal. Mach. Intell. 36, 1988–2001.
Cakir, F., Sclaroff, S., 2015. Adaptive hashing for fast similarity search. In: International Conference on Computer Vision.
Cevikalp, H., 2010. New clustering algorithms for the support vector machine based hierarchical classification. Pattern Recognit. Lett. 31, 1285–1291.
Cevikalp, H., Elmas, M., Ozkan, S., 2016. Towards category based large-scale image retrieval using transductive support vector machines. In: European Conference on Computer Vision Workshops.
Cevikalp, H., Triggs, B., Polikar, R., 2008. Nearest hyperdisk methods for high-dimensional classification. ICML.
Cevikalp, H., Verbeek, J., Jurie, F., Klaser, A., 2008. Semi-supervised dimensionality reduction using pairwise equivalence constraints. In: International Conference on Computer Vision Theory and Applications.
Collobert, R., Sinz, F., Weston, J., Bottou, L., 2006. Large scale transductive svms. J. Mach. Learn. Res. 7, 1687–1712.
Ertekin, S., Bottou, L., Giles, C.L., 2011. Nonconvex online support vector machines. IEEE Trans. Pattern Anal. Mach. Intell. 33, 368–381.
Fergus, R., Bernal, H., Weiss, Y., Torralba, A., 2010. Semantic label sharing for learning with many categories. ECCV.
Frogner, C., Zhang, C., Mobahi, H., Araya-Polo, M., Poggio, T., 2015. Learning with a wasserstein loss. NIPS.
Gionis, A., Indyk, P., Motwani, R., 1999. Similarity search in high dimensions via hashing. In: International Conference on Very Large Databases.
Gong, Y., Jia, Y., Leung, T., Toshev, A., Ioffe, S., 2013. Deep convolutional ranking for multilabel image annotation. arXiv:1312.4894.
Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F., 2013. Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans. PAMI 35, 2916–2929.
Gordo, A., Almazan, J., Revaud, J., Larlus, D., 2016. Deep image retrieval: learning global representations for image search. In: European Conference on Computer Vision.
Griffin, G., Perona, P., 2008. Learning and using taxonomies for fast visual categorization. CVPR.
He, X., Cai, D., Han, J., 2008. Learning a maximum margin subspace for image retrieval. IEEE Trans. Knowl. Data Eng. 20, 189–201.
Heo, J.-P., Lee, Y., He, J., Chang, S.-F., Yoon, S.-E., 2012. Spherical hashing. CVPR.
Hoi, S.C.H., Jin, R., Zhu, J., Lyu, M.R., 2008. Semi-supervised svm batch mode active learning for image retrieval. CVPR.
Jegou, H., Douze, M., Schmid, C., 2010. Product quantization for nearest neighbor search. IEEE Trans. PAMI 33, 117–128.
Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T., 2014. Caffe: convolutional architecture for fast feature embedding. arXiv:1408.5093 arXiv preprint.
Joly, A., Buisson, O., 2011. Random maximum margin hashing. CVPR.
Krizhevsky, A., Sutskever, I., Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. NIPS.
Kulis, B., Darrell, T., 2009. Learning to hash with binary reconstructive embeddings. NIPS.
Kulis, B., Grauman, K., 2009. Kernelized locality-sensitive hashing for scalable image search. ICCV.
Lai, H., Pan, Y., Yan, S., 2015. Simultaneous feature learning and hash coding with deep neural networks. CVPR.
Lin, G., Shen, C., Shi, Q., van den Hengel, A., Suter, D., 2014. Fast supervised hashing with decision trees for high-dimensional data. CVPR.
Lin, K., Yang, H.-F., Hsiao, J.-H., Chen, C.-S., 2015. Deep learning of binary hash codes for fast image retrieval. In: IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW).
Liu, H., Wang, R., Shan, S., Chen, X., 2016. Deep supervised hashing for fast image retrieval. CVPR.
Liu, W., Wang, J., Ji, R., Jiang, Y.-G., Chang, S.-F., 2012. Supervised hashing with kernels. CVPR.
Liu, W., Wang, J., Kumar, S., Chang, S.-F., 2011. Hashing with graphs. ICML.
Liu, W., Wang, J., Kumar, S., Chang, S.-F., 2011. Hashing with graphs. In: International Conference on Machine Learning.
Mu, Y., Shen, J., Yan, S., 2010. Weakly-supervised hashing in kernel space. CVPR.
Norouzi, M., Fleet, D.J., 2011. Minimal loss hashing for compact binary codes. ICML.
Raginsky, M., Lazebnik, S., 2009. Locality-sensitive binary codes from shift-invariant kernels. ICCV.
Salakhutdinov, R., Hinton, G., 2009. Semantic hashing. Int. J. Approximate Reasoning 50 (12), 969–978.
Sanchez, J., Perronnin, F., Mensink, T., Verbeek, J., 2013. Image classification with the fisher vector: theory and practice. Int. J. Comput. Vis. 34, 1704–1716.
Schölkopf, B., Zhou, D., Huang, J., 2005. Learning from labeled and unlabeled data on a directed graphs. ICML.
Shalev-Shwartz, S., Singer, Y., Srebro, N., 2007. Pegasos: Primal estimated subgradient solver for svm. ICML.
Shi, J., Malik, J., 2000. Normalized cuts and image segmentation. IEEE Trans. PAMI 22, 888–905.
Shi, X., Xing, F., Cai, J., Zhang, Z., Xie, Y., Yang, L., 2016. Kernel-based supervised discrete hashing for image retrieval. In: European Conference on Computer Vision.
Torralba, A., Fergus, R., Freeman, W.T., 2008. 80 million tiny images: a large data set for nonparametric object and scene recognition. IEEE Trans. PAMI 30, 1958–1970.
Vural, V., Dy, J.G., 2004. A hierarchical method for multi-class support vector machines. ICML.
Wang, J., Kumar, S., Chang, S.F., 2010. Semi-supervised hashing for scalable image retrieval. CVPR.
Weiss, Y., Torralba, A., Fergus, R., 2008. Spectral hashing. NIPS.
Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S., 2014. Supervised hashing for image retrieval via image representation learning. In: Proceedings of the 28 AAAI Conference on Artificial Intelligence.
Yullie, A.L., Rangarajan, A., 2002. The concave-convex procedure (cccp). Neural Information Processing Systems.
Zhang, L., Wang, L., Lin, W., 2012. Semi-supervised biased maximum margin analysis for interactive image retrieval. IEEE Trans. Image Process. 21, 2294–2308.
Zhang, R., L.Lin, Zhang, R., Zuo, W., Zhang, L., 2015. Bit-scalable deep hashing with regularized similarity learning for image retrieval and person re-identification. IEEE Trans. Image Process. 24, 4766–4779.
Zhang, Z., Chen, Y., Saligrama, V., 2016. Efficient training of very deep neural networks for supervised hashing. CVPR.
Zhao, F., Huang, Y., Wang, L., Tan, T., 2015. Deep semantic ranking based hashing for multi-label image retrieval. CVPR.
Zhigang, L., Wenzhong, S., Qianqing, Q., Xiaowen, L., Donghu, X., 2005. Hierachical support vector machines. Geoscience and Remote Sensing Symposium.