Attribute Discovery via Predictable Discriminative Binary Codes

Mohammad Rastegari[†] Ali Farhadi[‡] David Forsyth[†]

[†]University of Illinois at Urbana Champagin [‡]Carnegi Mellon University
http://vision.ri.cmu.edu/projects/dbc/dbc.html

Abstract. We present images with binary codes in a way that balances discrimination and learnability of the codes. In our method, each image claims its own code in a way that maintains discrimination while being predictable from visual data. Category memberships are usually good proxies for visual similarity but should not be enforced as a hard constraint. Our method learns codes that maximize separability of categories unless there is strong visual evidence against it. Simple linear SVMs can achieve state-of-the-art results with our short codes. In fact, our method produces state-of-the-art results on Caltech256 with only 128-dimensional bit vectors and outperforms state of the art by using longer codes. We also evaluate our method on ImageNet and show that our method outperforms state-of-the-art binary code methods on this large scale dataset. Lastly, our codes can discover a discriminative set of attributes.

1 Introduction

This paper describes a method that represents images with binary codes. In training, we infer codes for training images, and learn classifiers to predict the codes; in testing, we apply those classifiers to a test image to produce a code. An established literature shows how such binary codes can be used for image retrieval (e.g via hashing) and for image classification (e.g, via multi-class classification).

Image retrieval emphasizes appearance similarity, and many similar looking objects belong to the same category. This means that images that look similar (resp. dissimilar) should have similar (resp. dissimilar) codes. We call this property **unsupervised similarity**. These kinds of appearance similarity are not particularly discriminative. To ensure discrimination one needs to produce same (resp. dissimilar) codes for members of same(resp. different) categories. We call this property **discriminative similarity**. Our code construction balances unsupervised similarity with discriminative similarity.

Our codes are learned from category information on a per-image basis, meaning that training images within the same category may have different codes. We see this as an important innovation. It is natural, because not all objects within a category share all properties. Furthermore, doing this allows us to balance the discriminative information in a particular bit with our ability to predict the bit. One might try to train a system to predict a fixed code for each image within a category; however, there is no evidence that one can predict these codes accurately from images. Each bit in our codes can be thought of as a visual attribute whose name is not known. Like attributes, our codes lead

to natural models of categories. Later in the process, we infer what is shared within each category by selecting the most informative bits per category.

Our experimental evaluations show that our codes can produce state of the art retrieval and classification performance results on Caltech256. We also perform extensive evaluations on ImageNet. Our bit codes behave like attributes, and we show that our method has discovered visual attributes (Figure 8). Like attributes, our bit codes can be used as features to recognize objects in categories not used for training the code (Figure 9). We show that learning codes on a per-image basis outperforms that of category-based codes. Finally, the space produced by our learned codes can model within category variations (Figure 10).

2 Related Works

Binary codes are attractive image representations for image search and retrieval, because they are easy to match, and the capacity of the space of very short binary codes is so large that all of the digital images in the world can be indexed with relatively short binary codes. 64-dimensional codes can index about 10^{19} images; 5 times the estimated number of bits created in 2002 and likely similar to the number of digital images in existence [1]. Unfortunately, it is not known how to perfectly encode visual information into the binary codes to enable efficient search and retrieval.

Binary codes have been usually used as hash keys where the hashing functions are learned to preserve some notion of similarity in the original feature space. Important examples include: locality sensitive hashing [2], where similar objects have high probability of collision; parameter sensitive hashing [3], where the hash code is adjusted to improve regression performance; kernelized locality sensitive hashing [4], which results in fast image search and retrieval; binary reconstructive embedding [5], which encourages distances between binary codes to mirror distances in the original image feature space; efficient retrieval [6]; and semantic hashing [7], which encourages distances between codes to mirror semantic similarities, approximated by category memberships. Semantic hashing methods can produce very efficient image search methods for collections of millions of images [8]. Semantic hashing methods use a restricted boltzman machine; extensions include stacking multiple restricted boltzmann machines [9]. Alternatively, Norouzi and Fleet [10] model the problem of supervised compact similarity-preserving binary code learning as a Latent SVM problem and defined a hashing-specific class of loss functions. None of these approaches would necessarily result in discriminative codes. In fact, Figure 5 shows that preserving patterns in the original feature space may hurt discrimination in both supervised and unsupervised methods. Our experiments demonstrate that our codes achieve significantly better performances compared to state of the art supervised and unsupervised binary code methods.

Rotating the original feature space, then quantizing, is another approach. In spectral hashing [11], compact binary codes are calculated by thresholding a subset of eigenvectors of the laplacian of the affinity graph. Spectral hashing has been shown to outperform RBM and boosting SSC in [3]. Raginsky and Lazebnik [12] project the data to a low dimensional space and then use random quantizations, after [13]. Lin et al [14] use an iterative learning method to produce binary codes whose hamming distance corre-

lates to the similarities explained by the affinity matrix of the data in the original feature space; doing so requires long codes. Jégou et al. [15] jointly optimize for the search accuracy, search efficiency and memory requirements to obtain their binary codes. Gong and Lazebnik [16] iteratively minimize (ITQ) the quantization error of projecting examples from the original feature space to vertices of a binary hypercube. This method is capable of incorporating supervision by using CCA instead of PCA. ITQ has shown to produce state of the art results. Our experiments show that ITQ follows the patterns in the original feature space very well. This, however, may result in poor discrimination. We show that our binary codes consistently outperforms ITQ.

Many methods are unsupervised [11, 16, 12]. Some methods use a notion of similarity between labeled pairs of examples [10, 16, 14, 7]. Further, Wang et al. [17] get improvements over LSH and spectral hashing from a semi-supervised approach that minimizes the empirical loss over the labeled examples and maximizes the variance and independence of unlabeled examples. In either case, all methods assume that images that look "similar" should have the same label, but this is not always true for object categories.

Each bit in a binary code can be thought of as a **split** of the feature space into two half-spaces. Farhadi et al. [18, 19] construct thousands of random splits of the data, then pick the most predictable ones to generate random codes. Their splits are predictable and have high validation-set accuracy, but are not necessarily discriminative. Classeme features [20] are splits of data that produce state of the art results, but again there is no explicit discriminative component to the construction.

Alternatively, one could build codes out of object **attributes** [21] [22]. Such codes are easily interpreted semantically, but no explicit discriminative construction is yet known. Semantic attributes can also be discovered by selecting attributes that reduce the amount of confuision and are nameable[23, 24]. In terms of supervision, attributes are typically supervised or as recently shown they can be used in a semi-supervised fashion [25]. We do not use any supervision in terms of attributes. There is good evidence that random splits of data can produce informative bit strings [18, 20]. We differ from these constructions, because our method is explicitly discriminative. Furthermore, instead of learning bits independently, we learn bit vectors as a whole. Wang et al. [26] implicitly learns for discriminative codes by learning for hash functions that can sequentially correct the mistakes of the previous codes. We differ from them, because we explicitly optimize for discrimination in a max margin framework, we learn for the binary codes as a whole (not one by one), and we optimize jointly for discrimination and predictability.

3 Learning Discriminative Binary Codes

Our goal is to learn codes for each instance in the training set such that a) the codes can be reliably predicted from the visual data and b) if we represent each image with its learned codes then discrimination becomes easier. Our system consists of two main parts: learning binary codes for each instance and then performing search or classification in the space of binary codes. For learning our binary codes, we optimize for two criteria jointly: we want our codes to be as discriminative as they can, while maintain-

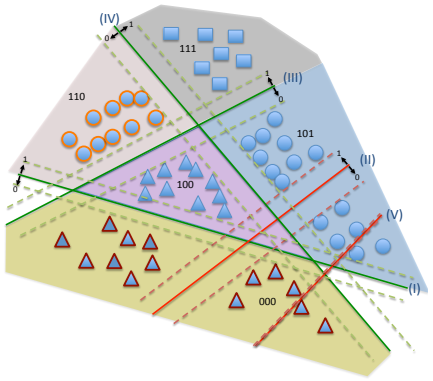

Fig. 1. Each bit in the code can be thought of as a hyperplane in the feature space. We learn arrangements of hyperplanes in a way that the resulting bit codes are discriminative and also all hyperplanes can be reliably predicted (enough margin). For example, the red hyperplanes (II,V) are not desirable because II is not informative(discriminative) and IV is not predictable (no margin). Our method allows the green hypeplanes (good ones) to sacrifice discrimination for predictability and vice versa. For example, our method allows the green hyperplane (I) to go through the triangle class because it has strong evidence that some of the triangles are very similar to circles.

ing predictability of the codes. The most discriminative codes (like assigning unique codes to examples of the same category) are extremely hard to predict from visual data. And the most predictable codes may contain very little information about categories resulting in poor discrimination. Our model balances between discrimination and predictability of the codes. In our view a code is discriminative if examples of different categories appear far away from each other and instances of the same category lie close by. However, we don't enforce these discriminative constraints as hard constraints but assign codes to each image in a way that the resulting codes have enough discriminative power and yet can be reliably predicted from images. Such a code allows for simple, efficient and very accurate classification and searches. For performing search and classification in the space of binary codes we use KNN and linear SVM. In Section 3, we demonstrate that KNN search in the space of our binary codes outperforms KNN on other state-of-the-art binary code spaces on Caltech256. We also show that linear SVM classifiers using our codes results in even higher accuracies with very few training examples per category.

Throughout this paper we use the term "splits" when we refer to bits of a code. Each bit can be visualized as a hyperplane that separates instances that have value 0 versus the ones that have value 1. Each bit of our codes is generated by checking which side of a hyperplane an instance lie. In [18], the splits are learned by randomly setting a subset of examples to positive and another subset to negative. Some of these splits can be reliably predicted from data. In [18], the splits are sorted based on how well they can be predicted from the data. The top K splits produce a K dimensional binary code. This procedure does not necessarily result in discriminative binary codes and the codes may need to be very long to ensure good performance. We believe the procedure to learn the splits and the procedure to find good binary codes should be learned jointly.

This results in binary codes that are predictable and have built-in margins. This means that each code is associated with a cell in an arrangement of hyperplanes in the feature space (Figure 1). The family of such arrangements is rich, meaning that we can find good codes.

Assume that we are given a training set $\{x_i, y_i\}$ where $i \in \{1, 2, ..., N\}$, $x_i \in R^d$, and $y_i \in \{1, 2, ..., C\}$ and we plan to learn k splits, meaning that our final binary code is k-dimensional. To train each split $s \in \{1, 2, ..., k\}$ we have to learn for labels to give us positive and negative training examples l^s where $l^s \in \{-1, 1\}$, -1 for negative and 1 for positive training examples. For each instance i in the training set, we are learning for l_i^s indicating which side of the split s the i^{th} example should appear. When learning for these codes our goal is to learn for a set of labels for each instance in a way that those labels can be reliably predicted from visual data and the learned codes leave enough space between categories. The final binary codes are the actual predictions of each split classifier s trained with l^s as training labels. We call these predictions $b_i^s \in \{0, 1\}$. b_i^s indicates which side of the split s the i^{th} example actually lies. In other words, b_i^s is the prediction of a classifier that uses l_i^s as training labels; l is what we want and b is what we can actually predict. We can stack all the labels and the predictions from all the split classifiers to form the final binary vector for each instance i in the training set: $L_i = [l_i^1, l_i^2, ..., l_i^k]$, and $B_i = [b_i^1, b_i^2, ..., b_i^k]$.

We are looking for binary codes that (a) can be reliably predicted from the original features and (b) provide enough margins between examples from different categories. To do that we learn to allocate codes to instances by searching for the whole code that jointly optimizes these two criteria. We do not optimize bits one by one as we can not guarantee uncorrelated binary codes. Our formulation looks like a combination of max margin models for linear SVMs to satisfy (a) and pushing for large inter class and small intra class distances for (b). We achieve such codes by optimizing:

$$\min_{w,\xi,L,B} \frac{1}{2} \sum_{c\in\{1:C\}} \sum_{m,n\in c} d(B_m, B_n) + \gamma \sum_{s\in\{1:k\}} \|w^s\|^2 \tag{1}$$

$$+\lambda_1 \cdot \sum_{\substack{i\in\{1:N\}\\ s\in\{1:k\}}} \xi_i^s - \frac{\lambda_2}{2} \sum_{c'\in\{1:C\}} \sum_{\substack{c''\in\{1:C\}\\ c'\neq c'', q\in c''}} \sum_{p\in c'} d(B_p, B_q)$$

$$s.t.\ \ l_i^s(w^{s'} x_i) \geq 1 - \xi_i^s \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

$$b_i^s = (1 + sign(w^{s'} x_i))/2 \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

$$\xi_i^s > 0 \quad \forall i \in \{1:N\}, s \in \{1:k\}$$

where d can be any distance in the hamming space, $B_i = [b_i^1, b_i^2, ..., b_i^k]$, w^s is the weight vector corresponding to the s^{th} split, ξ_i^s is the slack variable corresponding to the s^{th} split and i^{th} example, C is the total number of categories, k is the number of splits, N is the total number of examples in the train set, l_i^s is the training label for the i^{th} example to train the s^{th} split, and b_i^s indicates the prediction results of i^{th} example using the split s.

Algorithm 1 Optimization

Input: $X = [x_1, ...x_N]$ (x_i is low-level feature vector for i^{th} image).
Output: B ($B_i = [b_i^1, b_i^2, ..., b_i^k]$ is binary code for i^{th} image).

1: Initialize B by: $B \leftarrow$ Projection of X on first k components of $PCA(X)$
2: Binarize B: $B \leftarrow (1 + sign(B))/2$
3: **repeat**
4: Optimizing for B in $\min_B \frac{1}{2} \sum_{c \in \{1:C\}} \sum_{m,n \in c} d(B_m, B_n) - \frac{\lambda_2}{2} \sum_{c' \in \{1:C\} p \in c'} \sum_{c'' \in \{1:C\} c' \neq c'', q \in c''} d(B_p, B_q)$ (see supplementary materials for details)
5: $l_i^s \leftarrow (2b_i^s - 1)\ \forall i\ i \in \{1:N\}, \forall s\ s \in \{1:k\}$
6: Train k linear-SVMs to update $w^s\ \forall s$, $s \in \{1:k\}$ using L as training labels (l_i^s : label for i^{th} image when training s^{th} split)
7: $b_i^s \leftarrow (1 + sign({w^s}^T x_i))/2\ \ \forall i, s\ i \in \{1:N\},\ s \in \{1:k\}$
8: **until** Convergence on optimization 1

This is an extremely hard optimization problem, but we may not need to find the global minimum to obtain good binary codes. "Good" local minima are capable of producing promising discriminative binary codes. To go down the objective function in the optimization 1 we use an iterative block coordinate descent method. In algorithm 1 we described our optimization steb-by-step.

We initialize by choosing B to form orthogonal codes that come from projections along PCA directions. In our experiments we find that this initialization yields promising results. The supplementary material describes the (minor) effects of other choices of initialization. We then initialize w^s to predict these codes. Notice that the w's are independent given a fixed B, so we can use an SVM.

We now proceed by iterating three steps in sequence. **First**, we update B for fixed w_i^s, ξ_i^s; this proposes an improvement in the codes that should achieve improved separation. This is an iterative procedure that is started at the current value of B. We use stochastic gradient descent (step 4) with an important optimization. Since B is binary, if b_i^s is 0 then the sum of differences is the number of 1s and vise versa. We can precompute number of 0s and 1s for each s^{th} element of B. This way, we decrease the complexity of computing sum of differences from $O(N^2K)$ to $O(NK)$. **Second**, we update L using B and then (Fixing L, B) we update w_i^s, ξ_i^s by training SVMs using L as training labels. This produces a set of SVM's to predict these improved codes. Each bit of B represents a labeling of instances that we want an SVM to reproduce. We can therefore compute optimal w_i^s and ξ_i^s with an SVM code. **Third**, we update the current value of B to reflect the codes that these SVM's actually predict; this biases the update of B in the direction of codes that can be predicted. While this optimization procedure doesn't guarantee descent in each iteration, we have found that we get descent in practice (Figure 3). This is most likely because the steps balance the goodness of the code (updated in the first step), with our ability to predict it (second, third steps). In our experiments, we did not tune the parameters; λ_1,γ are set to 1 and λ_2 is set to normalize for the size of categories. Figure 3 shows the behavior of the objective function and all the terms in equation 1 after each iteration.

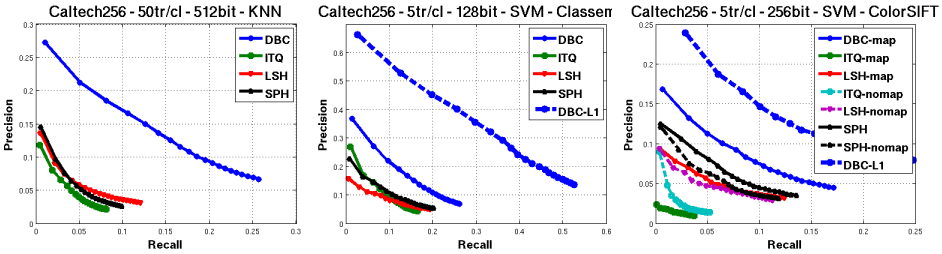

Fig. 2. We compare our binary codes(DBC) with Locality Sensitive Hashing(LSH), Spectral Hashing(SpH), and supervised version of Iterative Quantization(ITQ) under several different settings: changing the length of binary codes (32,64,128,256), classifiers (linear SVM or KNN), original features (Classeme, ColorSift) and also with L1 selection of category specific bits (DBC-L1). Our codes (DBC) consistently outperforms state of the art methods like SpH and ITQ by large margins. The test set contains 25 examples per category. Due to space limitations only very few of experimental settings can be showed in the paper. Please see supplementary material for all plots.

Once converged, optimization 1 provides us the weight vectors w^{s*} for split classifiers that tend to produce binary codes with built-in margins. We use w^{s*} to project the data to the space of the binary codes.

Using Codes: There are several ways to use the resulting binary codes. We evaluate our codes in a) using them as hash codes and performing KNN on the codes (called KNN in our experiments), b) using them as features and learning SVM classifiers for each category (called SVM), c) using the codes as features while accepting that these features might be redundant and using L1 regularized models to pick category specific codes (e.i. for each category we learn a L1-regularized SVM and pick the bits correspond to larger absolut weight value of the L1-SVM) and then learn normal SVM classifers using related bits.

4 Experimental Evaluations

Tasks: The main tasks of our experiments are in classification and category retrieval. We compare our method in several different settings with the state of the art bit-code-based methods. We also compare our method with state of the art classification techniques. Our bit learning algorithm results in interesting observations about the data like attribute discovery. Also, we qualitatively evaluate our method in retrieval and attribute discovery. Our method is also applicable to novel category recognition.

Datasets: We test our method on Caltech256 [27], and ImageNet [28] (ILSVRC2010). Both of these dataset have large number of categories (256 and 1000) with huge intra-class variations. Category retrieval on Caltech256 is a challenging task because the number of categories is much higher than typical experiments and also the intra-class variations are much higher than typical datasets like MNIST. There are around 30000 images belonging to 256 categories [27]. On average, there are about 120 images per category.

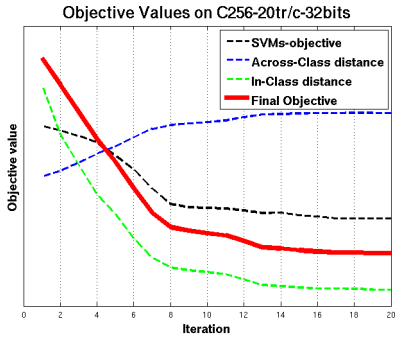

Fig. 3. Our optimization procedure finds descent directions in our challenging objective function. This figure shows that all terms in the objective function actually improves after each iteration.

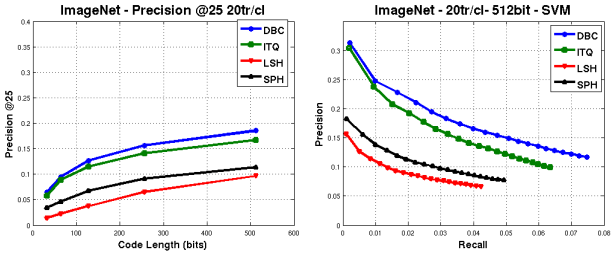

Fig. 4. Our method outperforms state of the art binary code methods (LSH, SpH, and supervised TTQ) on ImageNet(1000 categories, 20 per catgeory for training). Left plot compares precesion @25 vs. the code length. The test set contains 150 images per category. The right plot shows precision-recall curve for the same dataset using 512 dimensional codes. Our codes consistently outperforms all other methods.

Features: For experiments on Caltech256 we use two different sets of features that have been shown to produce state-of-the-art results on Caltech256: Classeme and ColorSift. We use Classeme features [20] because they have shown to outperform other features [20]. The Classeme features are of 2659 dimensionality. We also use ColorSift features as they show promising performances on classification tasks [29]. We use ColorSift bag-of-words features by building a 1000-word dictionary using ColorSift features provided by [30]. To make these features more discriminative we use homogeneous kernel map [31] on top of SIFT-BoW. The homogeneous kernels have shown to produce best results in many classification tasks [31]. Both of these features are among the most discriminative features. For ImageNet experiments we also use Classeme features.

Controls: To evaluate our method we perform series of extensive evaluations and comparisons. For our method, we change the following settings: the length of binary codes k (32, 64, 128, 256, 512), the number of training examples per category (5 0r 50), the original features (Classemes or ColorSift), the classifier (LSVM [32] or KNN), and the use of L1 selection of category specific bit strings. To compare with methods in the literature we compare our results with Locality Sensitive Hashing(LSH) as a standard baseline, with the supervised version of Iterative Quantization (ITQ) [16] as the best supervised method and Spectral Hashing (SpH)[11] as the state-of-the-art unsupervised

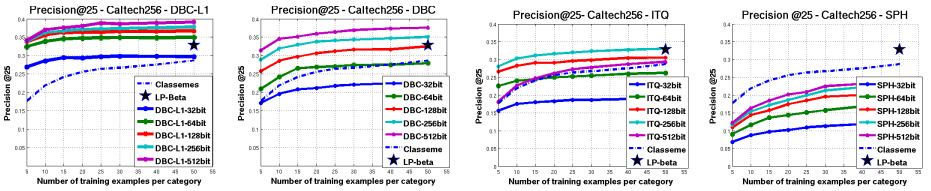

Fig. 5. Our method produces state of the art results on Caltech256. A linear SVM with only 128-bit code is as accurate as multiple kernel learning method of LPBeta (marked with a big star) that uses 13000 dimensional features. As we increase the size of the code we outperform the LPBeta method significantly. This figure compares our category specific codes (DBC-L1) , our codes without L1 selection (DBC), ITQ and SPH on precesion at 25 versus the number of training examples per category on caltech 256. One interesting observation is that the ITQ method does a great job in following the original features (Classeme) with 512 codes. This however hurts the performance as 128 and 256 dimensional codes outperforms the original features. This confirms our intuition that following the patterns in the original feature space does not necessarily result in good performance numbers.

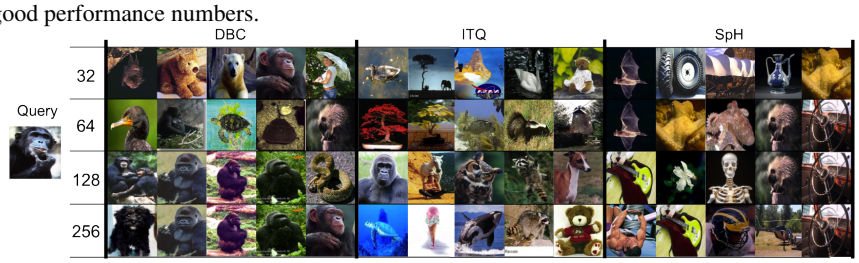

Fig. 6. This figure qualitatively compares the quality of retrieved images by our method comparing to that of ITQ and SpH. Each row corresponds to the top five images returned by three different methods: ours, ITQ and spectral hashing. This retrieval is done by first projecting the query image to the space of binary codes and then running KNN in that space. Notice how, even with relatively short codes(32 bits), our method recovers relevant objects. This menas that the discriminative training of the code has forced our code learning to focus on distinctive shared properties of categories. Our method consistently becomre more accurate as we increase the code size.

method in producing binary codes. Our experimental evaluations demonstrate that our method consistently outperforms state-of-the-art methods under all the combinations of above settings. To evaluate our method on a large scale dataset we test it on ImageNet. We used 1000 category from ILSVRC2010 (ImageNet Challenge). For each category we randomly chose 20 examples for training and 150 examples for testing. Our results show that our codes also outperform state-of-the-art binary code results on this dataset.

Measurements: In case of SVM, we use the top k images to compute precision and recall values. Varying $k = [1 : 5 : 100]$ traces out the precision-recall curves. In case of using KNN, for each number of nearest neighbors we can compute a precision and recall. Varying this number makes a precision recall curve.

Results: There are four main categories of results. First, we compare our method with the state of the art bit-code methods on Caltech256 and ImageNet. We also show interesting qualitative results. Second, we compare our results with the state of the art

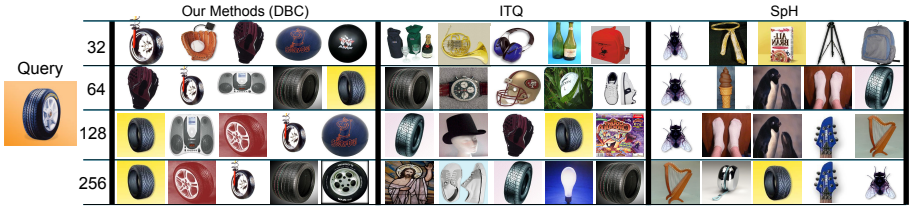

Fig. 7. This figure qualitatively compares the quality of retrieved images by our method comparing to that of ITQ and SpH. Each row corresponds to the top five images returned by three different methods: ours, ITQ and spectral hashing. This retrieval is done by first projecting the query image to the space of binary codes and then running KNN in that space. Notice how, even with relatively short codes(32 bits), our method recovers round objects. Our method is consistent in terms of returned images as we increase the code size. With 256 dimensional code our method returns 5 correct images.

method on Caltech256. Third, we compare our method on novel category recognition with the state of the art method of [33]. Fourth, we show qualitative results that reveal interesting properties of our method. We show promising attribute discovery results and also projections of the resulting bit code space.

Comparisons to the state of the art bit-code methods: Figure 2 compares our method (DBC, DBC-L1) with LSH,SpH, and supervised version of ITQ by varying the number of binary codes. We perform extensive evaluations on all combinations of different settings. Space does not allow showing all comparisons in all settings, please see supplementary material for all comparisons. The settings that we show here are: (from left to right on Figure 2) using KNN on 512-dimensional bit coses when 50 training examples per category are observed during training using Classeme features, using SVM on 128-dimensional bit codes when 5 training examples per category is observed during training using Classeme features, and using SVM on 256-dimensional bit codes when 5 training examples per category is observed during training using ColotSift features. In all possible settings, including these three, our method outperforms state of the art bit code methods. We also show that DBC-L1 performs better than DBC in all settings. The gap between the DBC and DBC-L1 increase as the number of bits decreases. The huge gap in the lower number of bits is due to the fact that in DBC-L1 we chose the bits to be specific at each category. In all of the experiments we use the same random selection of train and test set.

Our experiments show that as we increase the neighborhood size in KNN our method can still find the right categories (see supplementary materials). This implies that our hash cells remain pure as we increase the size of the neighborhood. This confirms that the optimization 1 managed to produce codes with enough margins. It is also worth noting that with such small training set per category linear SVM achieves excellent results using our codes.

In figure 5 we compare all the methods in terms of the precision at top-25 ranked images with different code length. We also compared our method with product quanti-

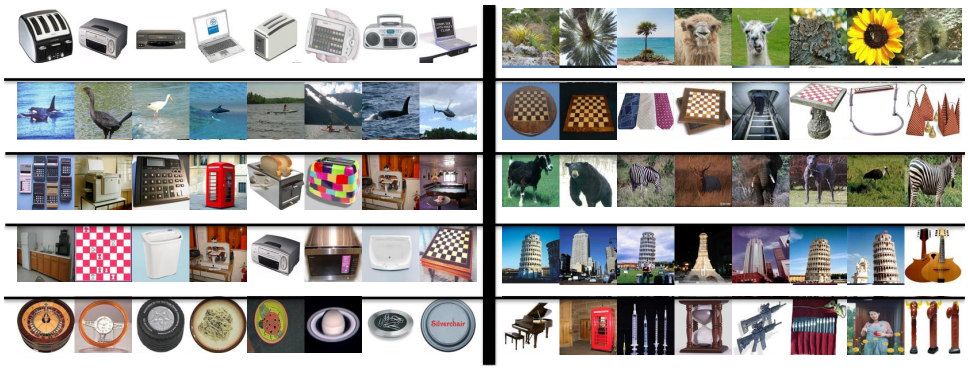

Fig. 8. Discovering attributes: Each bit corresponds to a hyperplane that group the data according to unknown notions of similarity. It is interesting to show what our bits have discovered. On two sides of the black bar we show 8 most confident images for 5 different hyperplanes/bits (Each row). Note that one can easily provide names for these attributes. For example, the bottom row corresponds to all round objects versus objects with straight vertical lines. The top row has silver, metalic and boxy objects on one side and natural images on the other side, the second row has water animals versus objects with checkerboard patterns. Discovered attributes are in the form of contrast: both sides have its own meaning. These attributes are compact representations of standard attributes that only explain one property. For more examples of discovered attributes please see supplementary material.

zation [34] for 5tr/cl and follow the same experimental setup. Product quantization got the precision of 0.04, 0.05, 0.064, 0.08, 0.09 for 32, 64, 128, 256, 512 bits repectively. Our method outperforms all the methods in all different code lengths. The Left plot in Figure 4 shows this comparison on ImageNet. For all other comparisons on Caltech256 please see the supplementary material. In these experiments, the test set contains 25 images per category . Figure 6 and 7 qualitatively compare our discriminative binary codes with ITQ and SpH in an image retrieval task. We show the top five retrieved images for the query image. It is interesting to see that even with 32 dimensional code our method is capable of extracting relevant properties. Our method is consistent in terms of returned images as one increases the code size.

Comparison to the state of the art models on Caltech256: Figure 5 compares our results with state-of-the art methods on Caltech256. We use the same features as the state-of-the-art method of LPBeta (The big star in the figure). With only 128 bits we can achieve the same results as the state of the art method of LPBeta that uses 13000 dimensional features. By increasing the number of bits our codes outperform the multiple kernel learning method of LPBeta. This shows that DBC can be significantly more discriminative than state-of-the-art features. We also compare with the classeme features. In this Figure we perform the same test with other binary code method. ITQ is doing a great job in getting close to the original features of Classeme by using 512 binary codes. However, it gets worse comparing to using 128 or 256 codes. This is mainly due to the fact that ITQ minimizes the quantization error of binarization and this does not necessarily result in better discrimination. Our method consistently gets better with more and more bits.

Novel Category Recognition: So far, we have shown that our codes are discriminative for categories they have been trained on. Similar to cross category generalization of attributes, we also evaluate our method on categories that have not been observed during training. For that, we learn the binary codes on 1000 categories of ImageNet with 20 examples per category and test our codes on Caltech256. We make sure that none of the 1000 categories intersect with 256 categories on Caltech. We adopt an experimental setting from [33] for which training data is available. Figure 9 shows that our method outperforms PiCodes [33], the state of the art novel category recognition method. We used the same low-level features as in [33].

Attribute Discovery: Binary codes can be though of as attributes. Our algorithm discovers attributes that can be named without much difficulty. Figure 8 shows some of the attributes discovered by our method. Each row shows 8 most confident examples for both sides of a hyperplane that corresponds to a bit in our code. Our learning procedure can discover attributes like is round, is boxy, is natural, has checkerboard pattern, and etc. More discovered attributes can be find in supplementary material. Our model learns strong contrasts that are discriminative. As a result of this each side of the discovered attributes has its own meaning. The discovered attributes are compact versions of standard attributes. Standard attributes describe only one property. But our discovered attributes are in the form of contrasts. For example, the first row contrasts boxy and silver objects against natural objects. If the bit that corresponds to the first row is 1 this means that the attribute boxy is 1 and if the bit is zero this means that the attribute natural is 1. It is also very interesting to look at the space of binary codes. To do that, we project our binary codes into a 2-dimensional space using multidimensional scaling. Figure 10 shows an interesting balance between discrimination and classification. In the projected space round things like wheels and coins are close together despite belonging to different categories. At the same time, round things are far away from horse and camel examples. Examples of the head of horse and camels are closer together than those to side views of horses and camels. Category memberships are suitable proxies for visual similarity but should not be enforced as hard constraints. Our model manages to balance between discrimination in terms of basic level categories and learnability of the codes from visual data.

5 Discussions

In this paper we demonstrate that by balancing discrimination and learnability of the codes one can achieve small binary codes that outperform state of the art results. We do this by letting each image has its own code while jointly optimizing for discrimination and learnability of the codes. Our experimental evaluations show that when there are strong visual evidence against categorical membership constraint, accounting for violations actually improves the discrimination. The codes learned in this way can reveal interesting properties of the data. The space of projected bits reveals interesting groupings of objects. Our method is also capable of producing meaningful codes for retrieval and can discover attributes. Different applications may need different trade offs between discrimination and learnability of the codes. What remains is how to learn to balance them according to query tasks. Our software and supplementary material are publicly

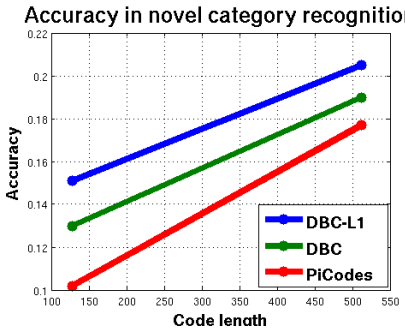

Fig. 9. Our codes can be used across the trainining categories (novel categories): we use 1000 categories of ImageNet to train our codes and use the codes to recognize objects in Caltech 256. The 1000 categories from ImageNet do not intersect with those of Caltech256. Our method outperforms state of the art methods in novel categories.

Fig. 10. Projection of the space of binary codes: We use multidimensional scaling and project our 64 dimensional codes into a two dimensional space. It is interesting to show that our method clearly balances between discrimination and learnability of the codes: round objects like wheel and coins appear close by while horses and camels are faraway. The head of the horse and the head of camels are close to each other and far way from side views of them. Supplementary material includes more examples of these projections.

available at `http://vision.ri.cmu.edu/projects/dbc/dbc.html`.
This work has been supported by the ONR-MURI grant N000141010934.

References

1. Lyman, P., Varian, H.R., Charles, P., Good, N., Jordan, L.L., Pal, J.: How much information? (2003)
2. A. Gionis, P.I., Motwani, R.: Similarity search in high dimensions via hashing. VLDB (1999)
3. Shakhnarovich, G., Viola, P.A., Darrell, T.: Fast pose estimation with parameter-sensitive hashing. In: ICCV. (2003)
4. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scalable image search. In: ICCV. (2009)
5. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. (2009)
6. Rastegari, M., Fang, C., Torresani, L.: Scalable object-class retrieval with approximate and top-k ranking. In: ICCV. (2011)

7. Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approx. Reasoning (2009)
8. Torralba, A., Fergus, R., , Weiss, Y.: Small codes and large image databases for recognition. In: CVPR. (2008)
9. Salakhutdinov, R., Hinton, G.: Learning a nonlinear embedding by preserving class neighbourhood structure. In: AISTATS. (2007)
10. Norouzi, M., Fleet, D.: Minimal loss hashing for compact binary codes. In: ICML. (2011)
11. Weiss, Y., Torralba, A.B., Fergus, R.: Spectral hashing. In: NIPS. (2008)
12. Raginsky, M., Lazebnik, S.: Locality sensitive binary codes from shift-invariant kernels. In: NIPS. (2009)
13. Rahimi, A., Recht, B.: Random features for large-scale kernel machines. In: NIPS. (2007)
14. sung Lin, R., Ross, D.A., Yagnik, J.: Spec hashing: Similarity preserving algorithm for entropy-based coding. In: CVPR. (2010)
15. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: CVPR. (2010)
16. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: CVPR. (2011)
17. Wang, J., Kumar, S., Chang, S.F.: Semi-Supervised Hashing for Scalable Image Retrieval. In: CVPR. (2010)
18. Farhadi, A., Forsyth, D.: Transfer learning in sign language. In: CVPR. (2007)
19. Farhadi, A., Tabrizi, M.K.: Learning to recognize activities from the wrong view point. In: ECCV. (2008)
20. Torresani, L., Szummer, M., Fitzgibbon, A.: Efficient object category recognition using classemes. In: ECCV. (2010)
21. Farhadi, A., Endres, I., Hoiem, D., Forsyth, D.: Describing objects by their attributes. In: CVPR. (2009)
22. Lampert, C.H., Nickisch, H., Harmeling, S.: Learning to detect unseen object classes by betweenclass attribute transfer. In: CVPR. (2009)
23. Parikh, D., Grauman, K.: Interactively building a discriminative vocabulary of nameable attributes. In: CVPR. (2011) 1681–1688
24. Duan, K., Parikh, D., Crandall, D., Grauman, K.: Discovering localized attributes for fine-grained recognition. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition. (2012)
25. Shrivastava, A., Singh, S., Gupta, A.: Constrained semi-supervised learning using attributes and comparative attributes. In: ECCV. (2012)
26. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: ICML. (2010)
27. Griffin, G., Holub, A., Perona, P.: The Caltech-256. Technical report (2007)
28. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR. (2009)
29. Burghouts, G.J., Geusebroek, J.M.: Performance evaluation of local colour invariants. CVIU (2009)
30. Gehler, P.V., Nowozin, S.: On feature combination for multiclass object classification. In: ICCV. (2009)
31. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. In: CVPR. (2010)
32. Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R., Lin, C.J.: LIBLINEAR: A library for large linear classification. JMLR (2008)
33. Bergamo, A., Torresani, L.: Picodes: Learning a compact code fornovel-category recognition. In: NIPS. (2011)
34. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33** (2011)