Supplementary Materials: Predictable Discriminative Binary Codes

Anonymous ECCV submission

Paper ID 1467

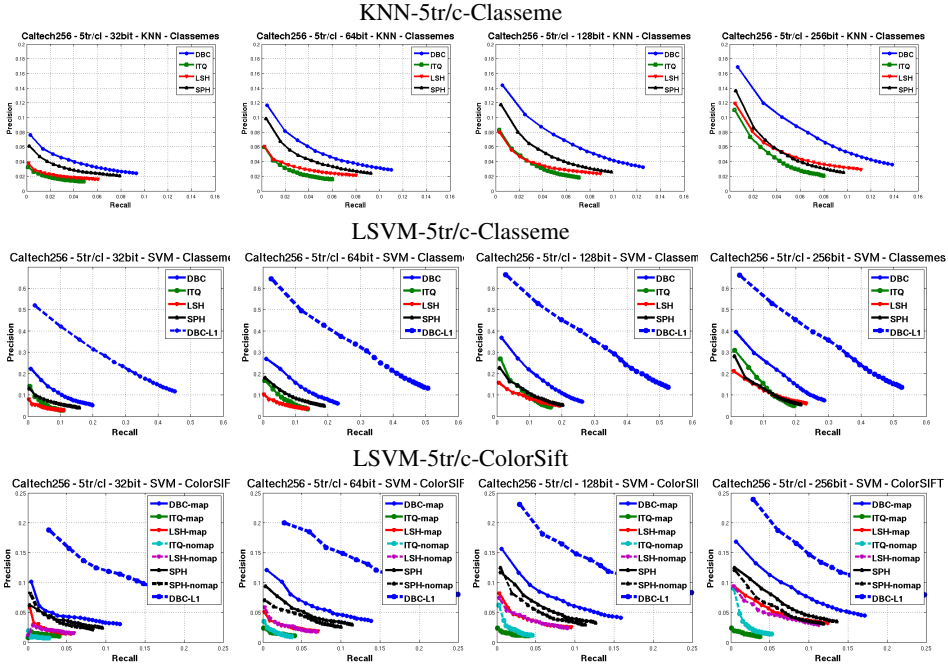

Fig. 1. This figure compares our discriminative binary codes(DBC, DBC-L1) with Locality Sensitive Hashing(LSH), Spectral Hashing(SpH), and supervised version of Iterative Quantization(ITQ) under several different settings: changing the length of the binary codes (32,64,128,256), changing the classifiers that takes binary codes as input and perform categorization (linear SVM or KNN), and changing the original features (Classeme, ColorSift). DBC consistently outperforms state of the art methods like SpH and ITQ by large margins. In these experiments we use Caltech256 with 5 training examples per category for learning both binary codes and the category classifiers. The test set contains 25 examples per category. The first row corresponds to using Classeme features with KNN. The second row is the same experiment but using linear SVM instead of KNN. Linear SVM on binary codes results in promising performances even with very few training examples. The third row is the same as the second row but using ColorSift features. DBC outperforms state of the art supervised and unsupervised methods.

This document provides additional results and experiments that support the methods presented in paper 1959.

1 Comparisons to the state of the art bit-code methods

To further compare our method with state of the art bit code methods, we perform series of extensive experiments. We compare our method (DBC, DBC_L1) with ITQ, SpH and LSH on Caltech256 while varying the code length (32,64,128,256), number of training examples per category(5,50), classifiers(SVM,KNN), features (Classeme, ColorSift), and the choice of using category specific codes(DBC-L1). Figure 1 fixes the number of training exampels per category to 5 and changes all other choices. Our method outperform state of the art bit code methods under all the settings.

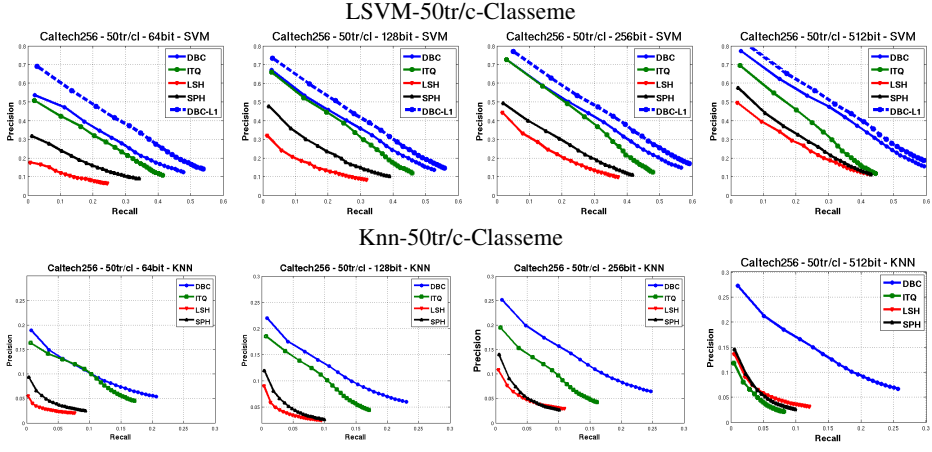

Fig. 2. This figure compares our discriminative binary codes(DBC, DBC-L1) with Locality Sensitive Hashing(LSH), Spectral Hashing(SpH), and Iterative Quantization(ITQ) under several different settings: changing the length of the binary codes (64,128,256,512), and changing the classifiers that takes binary codes as input and perform categorization (linear SVM or KNN). DBC consistently outperforms state of the art methods like SpH and ITQ by large margins. In these experiments we use Caltech 256 with 50 training examples per category for learning both binary codes and the category classifiers. The test set contains 25 images per category. The first row corresponds to using Classeme features with linear SVM. The second row is the same experiment but using linear KNN instead of linear SVM. DBC outperforms state of the art supervised and unsupervised methods.

Figure 2 shows the same comparisons but with 50 training examples per category.

We also perform comparisons on a large scale dataset: ImageNet ILSVRC2010 with 1000 categories and 20 training and 150 testing example per each category. Figure 3 shows that our method outperforms the state of the art bit code methods under different code length (64, 128,256,512).

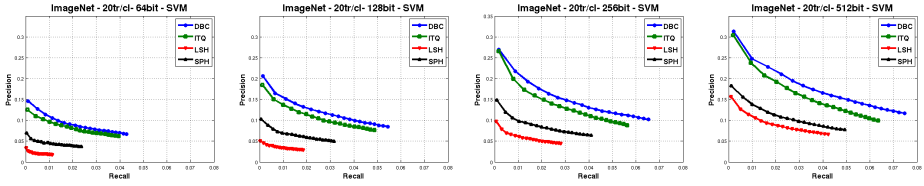

Fig. 3. We test Our method on a large scale dataset with 1000 categories of ImageNet. For this experiment we use classeme features and linear svm with 20 training and 150 testing examples. We change the number of bits in codes and compare with state of the art methods. DBC consistently outperforms state of the art binary code methods in a large scale setting.

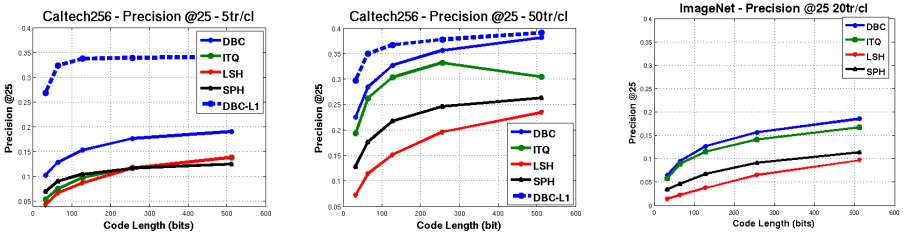

Fig. 4. Precision at 25 vs. number of bits in the code: The right and middle plots compare DBC, LSH, SpH and supervised ITQ on Caltech 256 using 5 and 50 training examples respectively. The test set contains 25 images per category. DBC consistently outperforms state of the art binary code models. The right plot is the same results on ImageNet dataset using 1000 categories and 20 examples per category for training. The test set contains 150 images per category. DBC outperforms LSH,ITQ and SpH in large scale setting with thousand categories.

We also compare our method with state of the art bit code method in precision at 25 versus the code length. Figure 4 compares our method with state of the art bit code methods on both Caltech256(5 and 50 training and 25 testing examples per class) and ImageNet. DBC and DBC-L1 consistently outperform state-of-the-art methods.

2 Efficient Subgradient Descend

In optimization 1 after fixing w_i^s and ξ_i^s we optimize for B using subgradient descent method described in Algorithm 1. Minimizing for B can be implemented very efficiently because our codes are binary. To calculate the gradient at k^{th} element of B_i one should compute the sum of its differences with every other k^{th} elements in B which is $O(N^2K)$. But our B is binary; if B_i^k is 0 then the sum of difference is the number of 1s and vise versa. We can precompute number of 0s and 1s at each k^{th} elements which is $O(NK)$. (Lines 6,7,9 and 10 of Algorithm 1)

Algorithm 1 Efficient Subgradient Descent

Input: B is a binary matrix($B_i(k) : k^{th}$ bit of i^{th} code.).
Output: Updated b at last iteration.

$n \leftarrow 1$
$b \leftarrow$ A binary matrix $b_i^n(k) : k^{th}$ bit of i^{th} code at n^{th} iteration.
$\forall k, i \quad b_i^0(k) = 0$
$\forall k, i \quad b_i^1(k) = B_i(k)$
$N \leftarrow$ Number of Examples
while $d(b^n, b^{n-1}) < tol$ **do**
  for $k = 1 \rightarrow CodeLength$ **do**
    $C_0 \leftarrow \sum_{i=1}^{N} \neg b_i^n(k)$
    $C_1 \leftarrow \sum_{i=1}^{N} b_i^n(k)$
    for $c = 1 \rightarrow NumberOfCategories$ **do**
      $S_0^c \leftarrow \sum_{i \in c} \neg b_i^n(k)$
      $S_1^c \leftarrow \sum_{i \in c} b_i^n(k)$
      $D_0 \leftarrow C_0 - S_0^c$
      $D_1 \leftarrow C_1 - S_1^c$
      $g_0 \leftarrow S_1^c - \lambda D_1$
      $g_1 \leftarrow S_0^c - \lambda D_0$
      for $i \in c$ **do**
        if $b_i^n(k)$ **then**
          $b_i^{n+1}(k) \leftarrow \neg(g_0 > 0)$
        else
          $b_i^{n+1}(k) \leftarrow \neg(g_1 > 0)$
  $n \leftarrow n + 1$

3 Predictability Versus Discrimination

In this section, we evaluate our main intuition that codes should be both discriminative and predictable. We compare our formulation that jointly optimizes for discrimination and predictability with only optimizing for predictability and only optimizing for discrimination. Figure 5 shows that our joint optimization outperforms optimizing for each term independently. To only optimize for predictability, we adopt the method of [7,8]. We start with 1000 random splits and pick the ones that can be best predicted from training data. To only optimize for discrimination, we force all images that belongs to a category to have the same code.

4 Initialization Effects

We further investigate the effects of initialization in our optimization. To achieve a good local minimum, we believe that our codes should not be highly correlated. Initializing our codes along the orthogonal directions of PCA helps us to obtain good initialization. Figure 6 compares this initialization with random initialization. We performed this experiment on Caltech256 with 5 training and 25 testing examples per category with different code lengths.

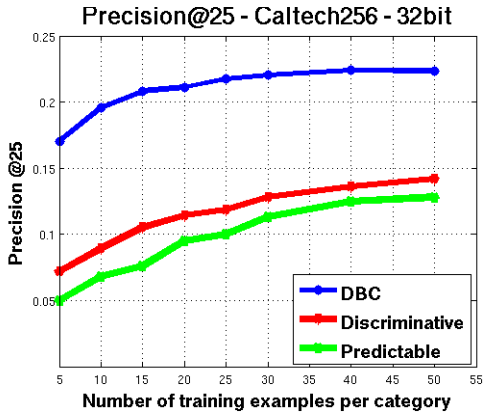

Fig. 5. Comparisons between jointly optimizing for predictability and discrimination and optimizing only for predictability and only for discrimination. We evaluate this comparison on Caltech256 using 32bit codes and change the number of training examples. There are 25 test examples for each category.

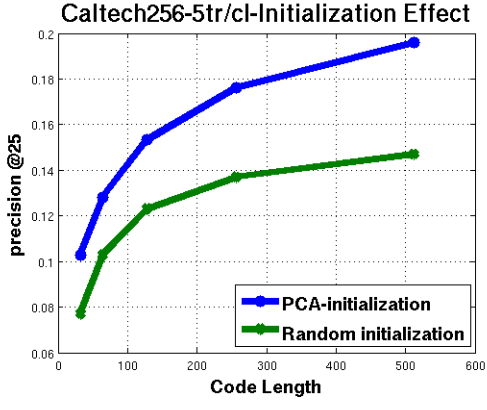

Fig. 6. Initialization: We compare the PCA-based initialization with random initialization on Caltech256 with 5 training and 25 testing examples per category. We vary the number of bits and evaluate precision at 25.

5 Effects of neighborhood size in KNN

By increasing the neighborhood size in KNN (K) our method can still find right categories. This implies that our hash cells remain relatively pure as we increase the size of the neighborhood, confirming that our optimization manages to find predictable codes with enough margins between categories. Figure 7 plots the precision at 25 versus the code length by increasing neighborhood size K on Caltech256 with 50 training and 25 testing examples per each category.

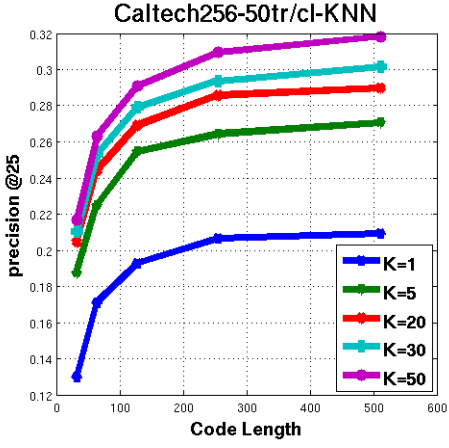

Fig. 7. Effects of neighborhood size in KNN: We plot the precision at 25 versus the code length. We run this experiments on Caltech256 with 50 training and 25 testing examples per each category. different colors corresponds to different Ks.

6 Attribute Discovery

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 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.

7 DBC Visualization

We show more qualitative results on visualizing our codes by projecting them into two dimensions using multi dimensional scaling. Figure 9 and Figure 10 illustrate 8 different visualizations. In each of the boxes we show 4 categories and for each category we

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 (from test set) for 5 different hyperplanes/bits (Each row). 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.

show 25 test images. These two figures exhibit an interesting balance between discrimination and learnability (visual similarities). 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.

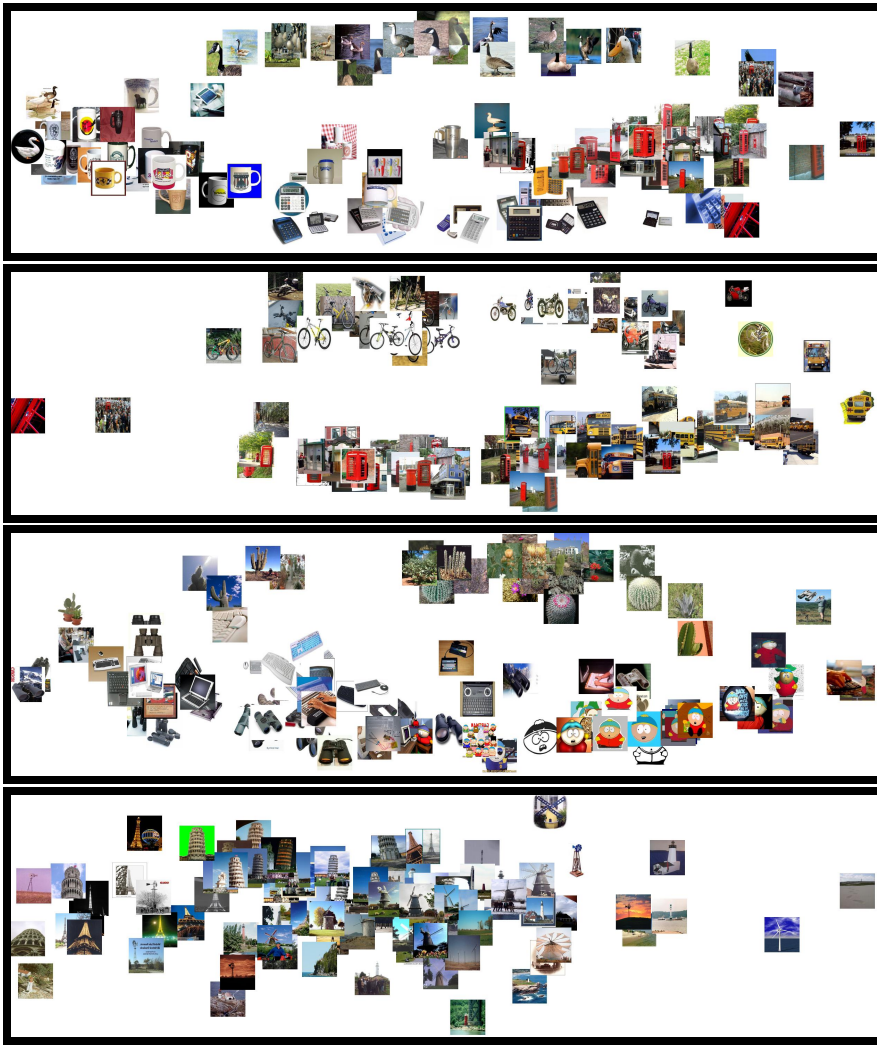

Fig. 9. Projection of the space of binary codes: We use multidimensional scaling and project our 64 dimensional codes into a two dimensional space. It is interesting to show that our method clearly balances between discrimination and learnability of the codes.

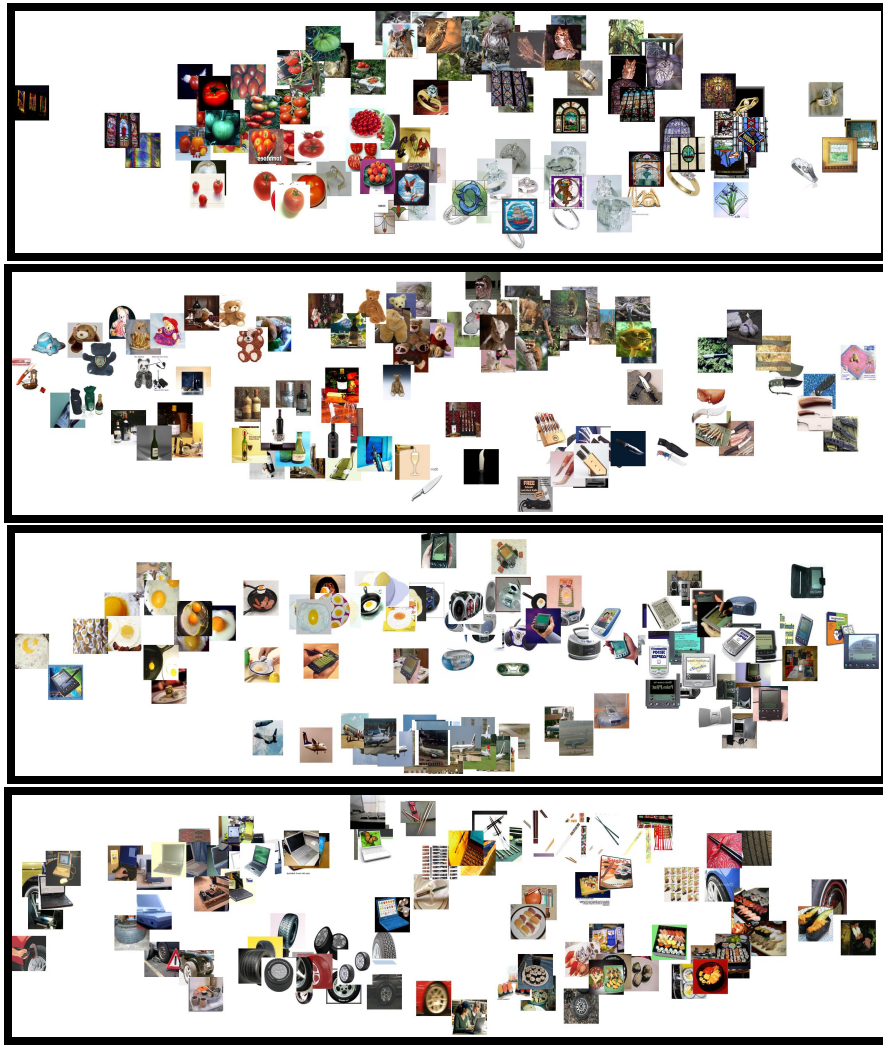

Fig. 10. Projection of the space of binary codes: We use multidimensional scaling and project our 64 dimensional codes into a two dimensional space. It is interesting to show that our method clearly balances between discrimination and learnability of the codes.