# Small Codes and Large Image Databases for Recognition

Antonio Torralba[1]

[1]CSAIL, MIT,
32 Vassar St.,
Cambridge, MA 02139
torralba@csail.mit.edu

Rob Fergus[2]

[2]Courant Institute, NYU,
715 Broadway,
New York, NY 10003
fergus@cs.nyu.edu

Yair Weiss[1,3]

[3]School of Computer Science,
Hebrew University,
91904, Jerusalem, Israel
yweiss@huji.ac.il

## Abstract

*The Internet contains billions of images, freely available online. Methods for efficiently searching this incredibly rich resource are vital for a large number of applications. These include object recognition [2], computer graphics [11, 27], personal photo collections, online image search tools.*

*In this paper, our goal is to develop efficient image search and scene matching techniques that are not only fast, but also require very little memory, enabling their use on standard hardware or even on handheld devices. Our approach uses recently developed machine learning techniques to convert the Gist descriptor (a real valued vector that describes orientation energies at different scales and orientations within an image) to a compact binary code, with a few hundred* **bits** *per image. Using our scheme, it is possible to perform real-time searches with millions from the Internet using a single large PC and obtain recognition results comparable to the full descriptor. Using our codes on high quality labeled images from the LabelMe database gives surprisingly powerful recognition results using simple nearest neighbor techniques.*

Recent interest in object recognition has yielded a wide range of approaches to describing the contents of an image. One important application for this technology is the visual search of large collections of images, such as those on the Internet or on people's home computers. Accordingly, a number of recognition papers have explored this area. Nister and Stewenius demonstrate the real-time specific object recognition using a database of 40,000 images [19]; Obdrzalek and Matas show sub-linear indexing time on the COIL dataset [20]. A common theme is the representation of the image as a collection of feature vectors and the use of efficient data structures to handle the large number of images.

These ideas are common to many approaches in the content based image retrieval (CBIR) community, although the emphasis on really large datasets means that the chosen image representation is often relatively simple, e.g. color [6], wavelets [29] or crude segmentations [4]. The Cortina system [22] demonstrates real-time retrieval from a 10 million image collection, using a combination of texture and edge histogram features. See Datta *et al.* for a survey of such methods [5].

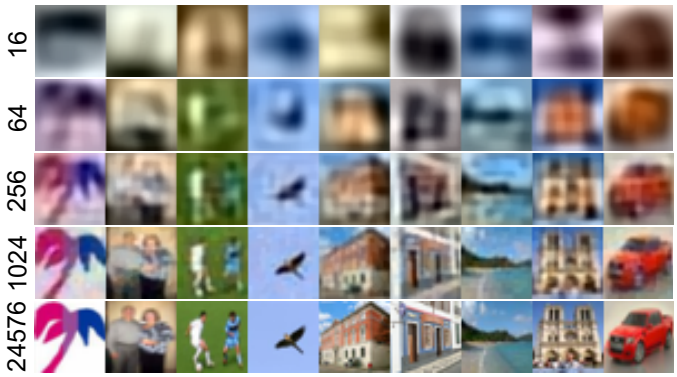


Figure 1. Short binary codes might be enough for recognition. This figure shows images reconstructed using an increasing number of bits and a compression algorithm similar to JPEG. The number on the left represents the number of bits used to compress each image. Reconstruction is done by adding a sparsity prior on image derivatives, which reduces typical JPEG artifacts. Many images are recognizable when compressed to have around 256-1024 bits.

Our approach is based on binary codes for representing images and their neighborhood structure. Such codes have received limited attention in both the vision and CBIR communities. Ghosh *et al.* [7] use them to find duplicate images in a database. Binary codes have also been used to represent the color of an image [13, 18]. Landre *et al.* [14] use color, texture and shape cues in a 32-bit vector to perform retrieval on a 10,000 image dataset. These approaches also use manually designed descriptors, which in view of the tiny capacity of each code, is likely to be highly suboptimal particularly when the database is large, a scenario not investigated by any of these papers.

Unlike CBIR we seek to recognize the objects present

in a novel image, not just retrieve relevant images from a database. We therefore want a fast way of finding images that are likely to contain the same objects as our novel image. Using the LabelMe database we show that the Gist descriptor, which represents orientation energy at different scales and orientations, is useful for this task. However, the descriptor is too high dimensional to use for fast querying of Internet-sized databases.

We were inspired by the results of Salakhutdinov and Hinton [25] who train compact binary codes to perform document retrieval. We believe binary codes are promising for three reasons. First, as shown by results on image compression (e.g. Figure 1) it is possible to represent images with a very small number of bits and still maintain the information needed for recognition. Second, scaling up to Internet-size databases requires doing the calculations in memory — desktop hard-drives are simply too slow. Fitting hundreds of millions of images into a few Gigabytes of memory means we have a budget of very few bytes per image. Third, as demonstrated convincingly in [25], short binary codes allow very fast querying in standard hardware, either using hash tables or efficient bit-count operations.

Perhaps the state-of-the-art method to obtain compact binary descriptors for querying a large database is Locality Sensitive Hashing (LSH), which finds nearest neighbors of points lying in a high dimensional Euclidean space in constant time. LSH does this by computing a hash function for a point by rounding a number of random projections of that point into $R^1$. Thus each random projection contributes a few bits (depending on the rounding function) to the descriptor of a point. Andoni and Indyk show that with high probability, points that are close in $R^n$ will have similar hash functions, and use this fact to efficiently find approximate nearest neighbors. LSH has been used successfully in a number of vision applications [26]. An alternative approach is to use kd-trees [16, 17] although LSH has been reported to work better in high dimensions [1].

Despite the success of LSH, it is important to realize that the theoretical guarantees are asymptotic - as the number of random projections grows. In our experience, when the number of bits is fixed and relatively small, LSH can perform quite poorly. The performance increases with more bits but given our desire to scale up to millions of images, it would be desirable to learn a compact code, rather than waiting for it to emerge from random projections.

In this paper, we leverage recent results in machine learning to learn compact binary codes that allow efficient retrieval. Specifically we explore how the Gist descriptor [21] can be reduced to a few hundred bits using a number of approaches including boosting, locality sensitive hashing and Hinton et al.'s restricted Boltzmann machine architecture [12]. We find that the learning approaches give superior performance compared to LSH and that using these codes it is possible to query databases with millions of images in a fraction of a second. When the retrieved images are annotated with high quality labels, simple nearest-neighbor techniques give surprisingly powerful recognition results.

## 1. Global image representations

Global image representations were developed in the framework of image retrieval (finding images that are *similar* to an input image) [5] and scene recognition (classifying an image as being a beach scene, a street, a living-room, etc.) [9, 21, 15]. The main characteristic of global image representations is that the scene is represented as a whole, rather than splitting it into its constituent objects. Such models correspond to the state of the art in scene recognition and context-based object recognition. Global image representations are based on computing statistics of low level features (oriented edges, vector quantized image patches, etc.) over fixed image regions or over large image segments [4].

In this paper we will use the scene representation proposed in [21] and we use the code available online. The image is first decomposed by a bank of multiscale oriented filters (tuned to 8 orientations and 4 scales). Then, the output magnitude of each filter is averaged over 16 non-overlapping windows arranged on a $4 \times 4$ grid. The resulting image representation is a $4 \times 8 \times 16 = 512$ dimensional vector. For smaller images ($32 \times 32$ pixels), we use 3 scales, resulting in $3 \times 8 \times 16 = 384$ dimensions. This representation can be thought of as using a single SIFT feature [17] to describe the entire image. Other techniques involve counting the number of occurrences of vector quantized SIFT features [3, 15] and textons.

Despite the simplicity of the representation, and the fact that they represent the full image rather than each object separately, these methods perform surprisingly well and can provide an initial guess of the scene identity, the objects present in the image and their spatial configuration. In this paper we will use global representation as a way of building very compact and efficient codes.

## 2. Learning binary codes

In this section we describe two learning approaches that generate binary codes. In the next section we will evaluate these approaches in the framework of recognition and segmentation. Our goal is to identify what is the minimal number of bits that we need to encode an image so that the nearest neighbor defined using a Hamming distance is also a semantically similar image.

We consider the following learning problem - given a database of images $\{x_i\}$ and a distance function $D(i, j)$ we seek a binary feature vector $y_i = f(x_i)$ that preserves the nearest neighbor relationships using a Hamming distance. Formally, for a point $x_i$, denoted by $N_{100}(x_i)$ the

indices of the 100 nearest neighbors of $x_i$ according to the distance function $D(i,j)$. Similarly, define $N_{100}(y_i)$ the set of indices of the 100 descriptors $y_j$ that are closest to $y_i$ in terms of Hamming distance. Ideally, we would like $N_{100}(x_i) = N_{100}(y_i)$ for all examples in our training set.

The two learning approaches are Boosting (introduced to this context by Shaknarovich and Darell [26]) and Restricted Boltzmann Machines (RBMs) introduced by Hinton and colleagues [12].

### 2.1. BoostSSC

Shaknarovich and Darrell [26] introduced Boosting similarity sensitive coding (BoostSSC) to learn an embedding of the original input space into a new space in which distances between images can be computed using a weighted hamming distance. In this section we describe the algorithm with some modifications so that it can be used with a Hamming distance.

In their approach, each image is represented by a binary vector with M bits $y_i = [h_1(x_i), h_2(x_i), ..., h_M(x_i)]$, so that the distance between two images is given by a weighted Hamming distance $D(i,j) = \sum_{n=1}^{M} \alpha_n |h_n(x_i) - h_n(x_j)|$. The weights $\alpha_i$ and the functions $h_n(x_i)$ that map the input vector $x_i$ into binary features are learned using Boosting.

For the learning stage, positive examples are pairs of images $x_i, x_j$ so that $x_j$ is one of the N nearest neighbors of $x_i$ $j \in N(x_i)$. Negative examples are pairs of images that are not neighbors. In our implementation we use GentleBoost with regression stumps to minimize the exponential loss. In BoostSSC, each regression stump has the form:

$$f_n(x_i, x_j) = \alpha_n \left[ (e_n^T x_i > T_n) = (e_n^T x_j > T_n) \right] + \beta_n$$

At each iteration $n$ we select the parameters of $f_n$, the regression coefficients ($\alpha_n$, $\beta_n$), the stump parameters (where $e_k$ is a unit vector, so that $e_k^T x$ returns the $k$th component of $x$, and $T_n$ is a threshold), to minimize the square loss:

$$\sum_{k=1}^{K} w_n^k (z_k - f_n(x_i^k, x_j^k))^2$$

Where K is the number of training pairs, $z_k$ is the neighborhood label ($z_k = 1$ if the two images are neighbors and $z_k = -1$ otherwise), and $w_n^k$ is the weight for each training pair at iteration $n$ given by $w_n^k = \exp(-z_k \sum_{t=1}^{n-1} f_t(x_i^k, x_j^k))$.

As we want the final metric to be a Hamming distance, we restrict the class of weak learners so that all the weights are the same for all the features $\alpha_n = \alpha$ (the values of $\beta_n$ do not need to be constrained as they only contribute to final distance as a constant offset, independent of the input pair). This small modification is important as it allows for very efficient techniques for computing distances on very large datasets. The parameter $\alpha$ has an effect in the generalization of the final function. For our experiments, we set $\alpha = 0.1$. By using a larger value of $\alpha$ (closer to 1), the algorithm is only able to learn distances when very short codes are used (around 30 bits) and it starts over-fitting after that.

Once the learning stage is finished, every image can be compressed into M bits, where each bit is computed as $h_n(x_i) = e_n^T x_i > T_n$. The algorithm is simple to code, relatively fast to train, and it provides results competitive with more complex approaches as we will discuss in the next section. For our experiments, the vectors $x_i$ contain the Gist descriptors. For training, we use 150,000 training pairs ($80\%$ being negative examples).

### 2.2. Restricted Boltzmann Machines

The second algorithm is based on the dimensionality reduction framework of Salakhutdinov and Hinton [12], which uses multiple layers of restricted Boltzmann machines (RBMs). We first give a brief overview of RBM's, before describing how we apply them to our problem.

An RBM models an ensemble of binary vectors with a network of stochastic binary units arranged in two layers, one visible, one hidden. Units $\mathbf{v}$ in the visible layers are connected via a set of symmetric weights $W$ to units $\mathbf{h}$ in the hidden layer. The joint configuration of visible and hidden units has an energy:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i \in \text{visible}} b_i v_i - \sum_{j \in \text{hidden}} b_j h_j - \sum_{i,j} v_i h_i w_{ij} \quad (1)$$

where $v_i$ and $h_j$ are the binary states of visible and hidden units $i$ and $j$. $w_{ij}$ are the weights and $b_i$ and $b_j$ are bias terms, also model parameters. Using this energy function, a probability can be assigned to a binary vector at the visible units:

$$p(\mathbf{v}) = \sum_{\mathbf{h}} \frac{e^{-E(\mathbf{v},\mathbf{h})}}{\sum_{\mathbf{u},\mathbf{g}} e^{-E(\mathbf{u},\mathbf{g})}} \quad (2)$$

RBMs lack connections between units within a layer, hence the conditional distributions $p(\mathbf{h}|\mathbf{v})$ and $p(\mathbf{v}|\mathbf{h})$ have a convenient form, being products of Bernoulli distributions:

$$\begin{aligned} p(h_j = 1|\mathbf{v}) &= \sigma(b_j + \sum_i w_{ij} v_i) \\ p(v_i = 1|\mathbf{h}) &= \sigma(b_i + \sum_j w_{ij} h_j) \end{aligned} \quad (3)$$

where $\sigma(x) = 1/(1 + e^{-x})$, the logistic function. Using Eqn. 3, parameters $w_{ij}, b_i, b_j$ can be updated via a contrastive divergence sampling scheme (see [12] for details). This ensures that the training samples have a lower energy than nearby hallucinations, samples generated synthetically to act as negative examples.

Recently, Hinton and colleagues have demonstrated methods for stacking RBMs into multiple layers, creating "deep" networks which can capture high order correlations

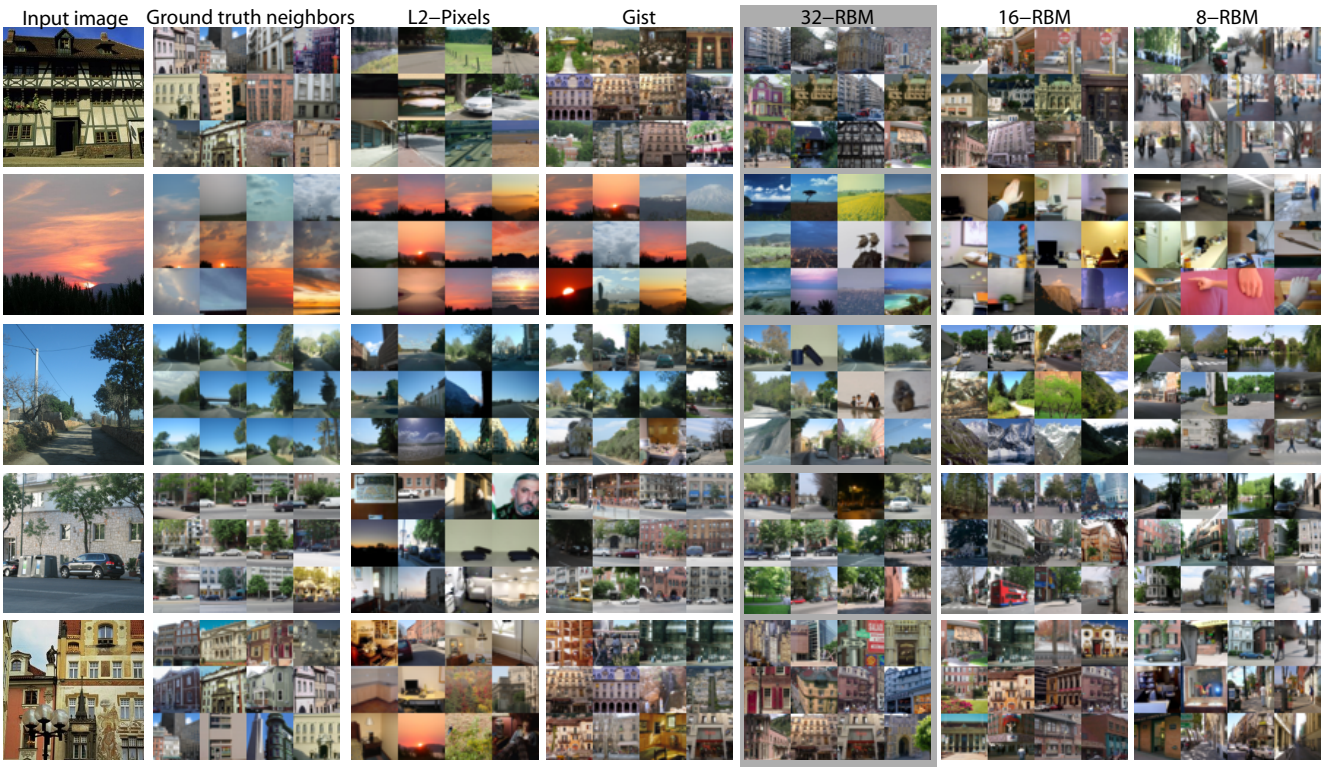


Figure 2. Each row shows the input image and the 12 nearest neighbors using, a) ground truth distance using the histograms of objects present on each image (see text), b) L2 distance using RGB values, c) L2 distance using Gist descriptors, d) Gist features compressed to 8 bits using an RBM and Hamming distance, e) 16 bits RBM and f) 32 bits RBM.

between the visible units at the bottom layer of the network. By choosing an architecture that progressively reduces the number of units in each layer, a high dimensional binary input vector can be mapped to a far smaller binary vector at the output. Thus each bit at the output maps through multiple layers of non-linearities to model the complicated subspace of the input data. Since the Gist descriptor values are not binary but real numbers, the first layer of visible units are modified to have a Gaussian distribution[1].

The deep network is trained into two stages: first, an unsupervised *pre-training* phase which sets the network weights to approximately the right neighborhood; second, a *fine-tuning* phase where the network has its weights moved to the local optimum by back-propagation on labeled data.

In pre-training, the network is trained from the visible input layer up to the output layer in a greedy fashion. Once the parameters of the first layer have converged using contrastive divergence, the activation probabilities (given in Eqn. 3) of the hidden layer are fixed and used as data for the layer above – the hidden units becoming the visible ones for the next layer up, and so on up to the top of the network.

In fine-tuning, we make all the units deterministic, retaining the weights and biases from pre-training and perform gradient descent on them using back-propagation. Our chosen objective function is Neighborhood Components Analysis (NCA) [8, 24]. This attempts to preserve the semantic neighborhood structure by maximizing the number of neighbors around each query that have the same class labels. Given $K$ labeled training cases $(\mathbf{x}^k, c^k)$, we define the probability that point $k$ is assigned the class of point $l$ as $p_{kl}$. The objective $O_{\text{NCA}}$ attempts to maximize the expected number of correctly classified points on the training data:

$$O_{\text{NCA}} = \sum_{k=1}^{K} \sum_{l:c^k=c^l} p_{kl}, p_{kl} = \frac{e^{-||f(\mathbf{x}^k|W)-f(\mathbf{x}^l|W)||^2}}{\sum_{m\neq l} e^{-||f(\mathbf{x}^m|W)-f(\mathbf{x}^l|W)||^2}}$$

where $f(\mathbf{x}|W)$ is the projection of the data point $\mathbf{x}$ by the multi-layered network with parameters $W$. This function can be minimized by taking derivatives of $O_{\text{NCA}}$ with respect to $W$ and using conjugate gradient descent.

Our chosen RBM architecture for experiments used four layers of hidden units, having sizes 512-512-256-$N$, $N$ being the desired size of the final code. However, for 8 and 16-bit codes, we set $N$=32 and added a fifth layer of 8 or 16 units respectively. The input to the model was a 384 or 512-dimensional Gist vector for the 12.9 million image dataset and the LabelMe dataset respectively. The model has a large number of parameters, for example in the case $N$=32, there are 663,552 ($512^2$+$512^2$+256·512+256·32)

[1]In Eqn. 3, $p(v_i = x|\mathbf{h})$ is modified to be a Gaussian with a mean determined by the hidden units, see [24].

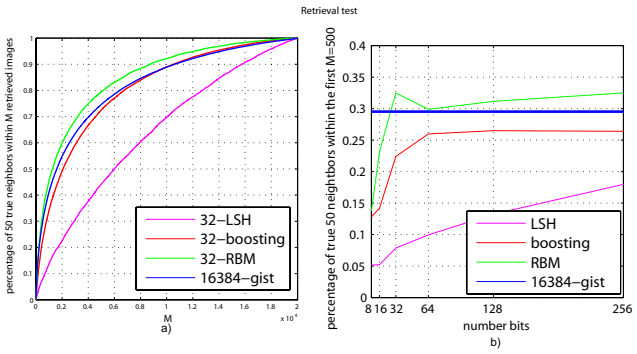


Figure 3. For a fixed number of true neighbors ($N = 50$), we plot the percentage of true nearest neighbors that are retrieved as a function of the total number of images retrieved $M$. True neighbors are defined in terms of object label histograms.

weights alone. Correspondingly, pre-training used Gist vectors from 70,000 images, 20,000 from the LabelMe training set, 50,000 from PeekaBoom images [2].

For back-propagation, in the case of LabelMe, we used 20 batches of 1000×1000 neighborhood regions, taking semantic labels from the LabelMe training data. Within each neighborhood region, the mean number of neighbors with the same class label was 100. For the Web images dataset, we computed the neighborhood labels to mimic the structure of Gist vectors, taking the 50 neighbors to have the same class label, with 100 batches of 500x500 neighborhood being used. For each batch, three iterations of conjugate gradient descent were performed, and a total of 20 epochs of training were used. The activation probabilities of the top hidden layer were binarized using a threshold set at the median value of each bit over the training data.

In testing, the Gist descriptors are computed for each query image and normalized in the same manner as the training data. In contrast to the training, evaluation of an RBM network is very fast (see Section 3.3 for timings), the activation probabilities being propagated up the network and binarized at the top layer, giving an $N$-bit binary code.

## 3. Experiments

How many bits do we need to represent images? In these experiments, our goal is to evaluate short binary codes that preserve "semantic" distance between scenes. Given an input image and a large database of annotated images, our goal is to find in this dataset images that are semantically similar. We use two datasets, one of 22,000 from LabelMe [23] and another of 12.9 million web images from [28].

[2]Further pre-training details: The Gist vectors were normalized to be zero mean, unit mean variance and split into 700 batches of size 100. The first layer of the RBM, having Gaussian visible units, was trained more gently than the others using 200 epochs of stochastic gradient descent with a learning rate of 0.001, weight decay of 0.00002 and momentum of 0.9. All other layers were trained with a learning rate of 0.1, using 50 epochs, other parameters being identical. The code was adapted from that accompanying [12].

### 3.1. LabelMe retrieval

In order to train the similarity measures we need to define what ground truth semantic similarity is. Our definition of semantic distance between two images is based on the histogram of object labels in the two images. For this we use the spatial pyramid matching [10, 15] over object labels. The spatial pyramid matching uses the histogram of object labels over image regions of different sizes, and the distance between images is computed using histogram intersection. This results in a simple similarity measure that takes into account the objects present in the image as well as their spatial organization: two images that have the same object labels in similar spatial locations are rated as closer than two images with the same objects but in different spatial locations, and this is rated closer than two images with different object classes.

As the LabelMe dataset contains many different labels describing the same objects, we collapsed the annotations using synonyms sets [23]. For instance, we group under the label "person", all the objects labeled as "pedestrian", "human", "woman", "man", etc.

Fig. 2 shows representative retrieval results and Fig. 3 provides a quantitative analysis of the retrieval performance on 2000 test images. Fig. 3(a) displays the percentage of the first true 50 nearest neighbors that are included in the retrieved set as a function of the number of the images retrieved ($M$). Fig. 3(b) shows a section of Fig. 3(a) for $M = 500$. The figures compare LSH, BoostSSC and RBMs. Fig. 3(b) shows the effect of increasing the number of bits. Top performance is reached with around 30 bits for RBMs, with the other methods requiring more bits. However, given enough bits, all the approaches converge to similar retrieval performance.

### 3.2. Web image dataset retrieval

As we increase the size of the dataset, we expect that longer codes will be required in order to find the nearest neighbors to one image. Here, we learn compact binary codes on a dataset of 12.9 million images from the web [28].

As we lack ground truth for semantic similarity in this dataset, in these experiments we have trained the RBM to reproduce the same neighborhood as the original Gist descriptors. Fig. 4 shows the overlap between the neighbors obtained with Gist and the neighbors obtained by computing Hamming distance using different bit length codes with RBMs and LSH. Fig. 5 shows examples of input images and the 12 nearest neighbors using different code lengths. There is a significant improvement in the semantic similarity of the neighbor images as we move from 30 bits to 256 bits per image.

### 3.3. Retrieval speed evaluation

We used two different algorithms for fast retrieval using the compact binary representation. The first is based

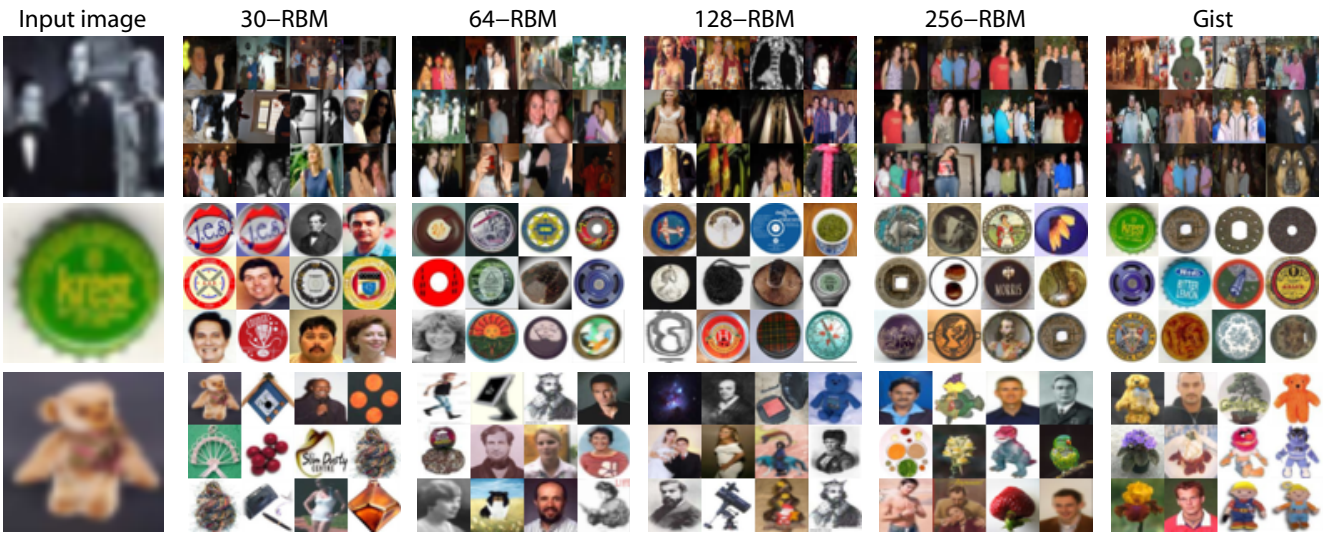


Figure 5. 12 nearest neighbors from a database of 12,900,000 images. As the number of bits increases the retrieved images are of similar visual quality as the Gist descriptor.

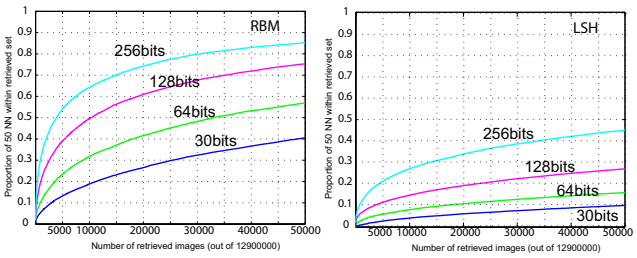


Figure 4. Comparison of retrieval results on the Web dataset (12.9 Million images) using different RBM encodings. The retrieval performance increases with the number of bits, but even for 256 bits we need to look at 4,000 images to find 25 of the 50 nearest neighbors obtained using Gist. Nevertheless, the performance is far better than LSH (right) which requires more than 50,000 images to retrieve 25 of the 50 nearest neighbors.

on hashing. The compact binary descriptor for each image becomes its hash key. Given a query descriptor, we enumerate all hash keys having up to $D$ bits different from the query. Any hash entries found are returned as neighbors, under our Hamming distance metric. Multiple images having the same hash key are stored in a linked list. The drawback to this scheme is the large memory requirements since an $N$-bit code requires a hash table of size $2^N$. Given the memory capacity of current PC's, this translates to a practical maximum of around $N$=30[3].

For codes longer than 30 bits, we use exhaustive search – for each query we calculate the Hamming distance to all images in the database. This would seem prohibitively slow for millions of images, but the Hamming distance can be calculated very quickly – it requires an xor followed by a bit-count operation.

[3]By using a 2nd (conventional) hash function in conjunction with the RBM hash function, it is possible to handle codes >30 bits but the volume of the Hamming ball quickly becomes prohibitive.

We compared our approach to kd-trees, a standard method for quick matching. Table 1 shows the time per image to find the closest 5 neighbors to a query point in both the LabelMe and Web images dataset using a variety of methods and input representations [4]. We also compared to the approximate-NN scheme used by Lowe [17] for matching SIFT vectors. We found this to be faster than ordinary kd-trees but still much slower than our learnt RBM codes and gave worse retrieval performance. Specifically, for the LabelMe dataset, retrieving 1000 neighbors using Lowe's kd-tree took 3ms/image and retrieved 17% of true neighbors, while RBMs took 6$\mu$s/image and retrieved 48% of true neighbors. Note that kd-trees cannot be applied to the Web dataset due to prohibitive memory requirements and even when they can be applied (for the LabelMe dataset), they are much slower than exhaustive search on the compact bit representation. Even for 12 million images, we can find exact nearest neighbors in Hamming space in a fraction of a second on a fast PC.

### 3.4. Short Binary Codes for Recognition

From our previous definition of semantic similarity, two images are semantically similar if their segmentations and object labels are exchangeable. If the retrieval is successful, the output will be an object label for every pixel in the input image. For this to work, we need a large database of annotated images so that the database covers most object configurations. In the first set of these experiments we used a large collection of *labeled* images from the LabelMe dataset. In that database, the set of street scenes is represented particularly well. For each such image users have labeled pixels as belonging to different objects like cars, roads, tree, sky, pedestrians, etc.

[4]For the kd-tree, we used a variant known as a spill-tree, using code from [16].

| Dataset | LabelMe | Web |
|---|---|---|
| # images | $2\times10^4$ | $1.29\times10^7$ |
| Gist vector dim. | 512 | 384 |
| Method | Time (s) | Time (s) |
| Spill tree - Gist vector | 1.05 | - |
| Brute force - Gist vector | 0.38 | - |
| Brute force - 30 bit binary | $4.3\times10^{-4}$ | 0.146 |
| " - 30 bit binary, M/T | $2.7\times10^{-4}$ | 0.074 |
| Brute force - 256 bit binary | $1.4\times10^{-3}$ | 0.75 |
| " - 256 bit binary, M/T | $4.7\times10^{-4}$ | 0.23 |
| Hashing - 30 bit binary | $6\times10^{-6}$ | $6\times10^{-6}$ |

Table 1. Timings for different methods of finding the 5 nearest neighbors to query vectors from the LabelMe dataset (2nd column) and the web images dataset (3rd column). Rows 2 and 3 detail the size of the dataset and the dimensionality of Gist vectors. Using the standard Gist descriptor to represent each image results in slow matching since it must be performed in a high dimensional space. Efficient methods such as spill-trees offer no advantage over brute force in such settings. Note that for the web images dataset, the raw Gist vectors cannot fit into memory so timings cannot be computed. By contrast, our binary codes can be matched quickly by brute force search. Using multi-threading (M/T) on a quad-core processor offers significant performance gains. We also list the timing of our 30-bit hashing scheme, whose run time is independent of the database size, being approximately $10^5$ times faster that matching in the original Gist descriptor space.

Figure 6 shows some labeling results. For each test image we select the 50 nearest neighbors, then for each pixel we assign the object label that has more votes at that pixel location. The final performance corresponds to the percentage of pixels correctly labeled. Fig. 7 summarizes the results on 2000 test images. It is important to note that the performances are bounded by the dataset. If one image does not have another similar image in the dataset, then we can not provide a segmentation. The black line in Fig. 7(a),(b) represents the maximal performance of the labeling task achieved when we use the true neighbors. On average, 68% of the pixels are correctly labeled.

We also performed a larger scale experiment on the Web images dataset. 2000 test examples were selected at random and manually labeled as being one of six classes (person, location, plant, animal, device, junk). For each test image, we found the closest 500 images in the 12.9 million and used the text label from the neighbors to vote on the class label, using Wordnet voting in the manner of [28]. This gave a confidence score as to the presence/absence of each object class in the image. Fig. 8(a) shows the recall-precision curve for the person class, while Fig. 8(b) shows the relative performance codes for all 5 classes to the original Gist descriptor. This figure shows that in a recognition task, for such a large dataset, we need more than 30 bits. Around 256 bits, performances are comparable to those achieved with the uncompressed Gist descriptor.

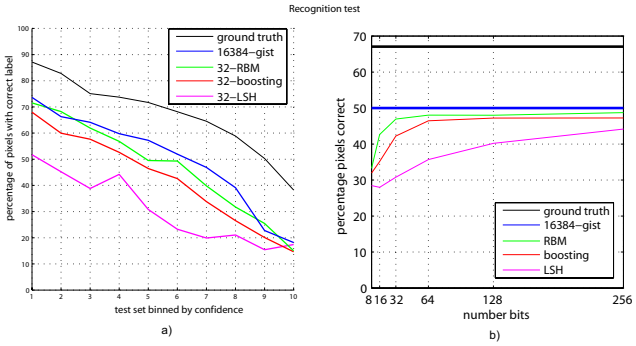


Figure 7. Pixel labeling experiment. a) Performance as a function of confidence (the agreement in votes provided by the retrieved images). Each bin contain 10% of the images from the test set. Note that for 20% of the images, around 70% of the pixels are correctly labeled. b) Average percentage of pixels correctly labeled as a function of the number of bits used for each code.

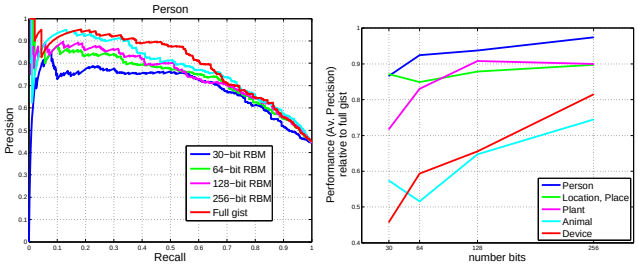


Figure 8. Recognition on Web images dataset. Left: recall-precision curves for the person category. Right: performance of binary codes (using average precision), relative to full Gist descriptor.

## 4. Discussion

One of the lessons of modern search engines is that even very simple algorithms can give remarkable performances by utilizing data on an Internet scale. It is therefore very tempting to apply such an approach to object recognition. But any research in this direction immediately runs into daunting problems of computation — imagine trying to download 80 million images, to say nothing of doing experiments with such a huge database. Efficient schemes of representation and matching are needed.

In this paper we have presented such schemes. We have shown that using recent developments in machine learning, it is possible to learn compact binary codes for large databases (as few as 256 bits per image). With these codes, a database of 12.9 million images takes up less than 600 Megabytes of memory and can fit on a memory stick. Furthermore, we have shown that fast querying is possible on this database on a standard PC. We plan to make the database and the querying tools publically available and hope this will help push object recognition research towards the Internet-scale challenge.


Figure 6. This figure shows six example input images. For each image, we show the first 12 nearest neighbors when using ground truth semantic distance (see text), using 32bits RBM and the original Gist descriptor (which uses 16384 bits). Below each set of neighbors we show the LabelMe segmentations of each image. Those segmentations and their corresponding labels are used by a pixel-wise voting scheme to propose a segmentation and labeling of the input image. The resulting segmentation is shown below each input image. The number above the segmentation indicates the percentage of pixels correctly labeled. A more quantitative analysis is shown in Fig. 7.

## Acknowledgments

The authors would like to thank Geoff Hinton and Rus Salakhutdinov for making their RBM code available online. Funding for this research was provided by NSF Career award (IIS 0747120), NGA NEGI- 1582-04-0004, Shell Research and ONR-MURI Grant N00014- 06-1-0734.

## References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468, 2006.

[2] K. Barnard, P. Duygulu, N. de Freitas, D. Forsyth, D. Blei, and M. Jordan. Matching words and pictures. *JMLR*, 3:1107–1135, Feb 2003.

[3] A. Bosch, A. Zisserman, and X. Muoz. Representing shape with a spatial pyramid kernel. In *CIVR*, 2006.

[4] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: Image segmentation using expectation-maximization and its application to image querying. *PAMI*, 24(8):1026–1038, 2002.

[5] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Computing Surveys*, page to appear, 2008.

[6] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: the QBIC system. *IEEE Computer*, 28(9):23–32, 1995.

[7] P. Ghosh, B. Manjunath, and K. Ramakrishnan. A compact image signature for RTS-invariant image retrieval. In *IEE VIE*, Sep 2006.

[8] J. Goldberger, S. T. Roweis, R. R. Salakhutdinov, and G. E. Hinton. Neighborhood components analysis. In *NIPS*, 2004.

[9] M. M. Gorkani and R. W. Picard. Texture orientation for sorting photos at a glance. In *Intl. Conf. Pattern Recognition*, volume 1, pages 459–464, 1994.

[10] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *Proc. CVPR*, 2007.

[11] J. Hayes and A. Efros. Scene completion using millions of photographs. *SIGGRAPH*, 2007.

[12] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Nature*, 313(5786):504–507, July 2006.

[13] I. Kunttu, L. Lepisto, J. Rauhamaa, and A. Visa. Binary histogram in image classification for retrieval purposes. In *Intl. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision*, pages 269–273, 2003.

[14] J. Landré and F. Truchetet. Optimizing signal and image processing applications using intel libraries. In *Proc. of QCAV 2007*, page to appear, Le Creusot, France, May 2007. SPIE.

[15] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, pages 2169–2178, 2006.

[16] T. Liu, A. W. Moore, and A. Gray. Efficient exact kNN and non-parametric classification in high dimensions. In *NIPS*, 2004.

[17] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[18] M. Nascimento and V. Chitkara. Color-based image retrieval using binary signatures. In *ACM symposium on Applied computing*, pages 687–692, 2002.

[19] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, pages 2161–2168, 2006.

[20] S. Obdrzalek and J. Matas. Sub-linear indexing for large scale object recognition. In *BMVC*, 2005.

[21] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal in Computer Vision*, 42:145–175, 2001.

[22] T. Quack, U. Monich, L. Thiele, and B. Manjunath. Cortina: A system for large-scale, content-based web image retrieval. In *ACM Multimedia*, 2004.

[23] B. C. Russell, A. Torralba, K. P. Murphy, and W. T. Freeman. Labelme: a database and web-based tool for image annotation. Technical Report AIM-2005-025, MIT AI Lab Memo, September, 2005.

[24] R. R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *AISTATS*, 2007.

[25] R. R. Salakhutdinov and G. E. Hinton. Semantic hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*, 2007.

[26] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *Proc. ICCV*, 2003.

[27] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: Exploring photo collections in 3d. In *SIGGRAPH*, pages 835–846, 2006.

[28] A. Torralba, R. Fergus, and W. T. Freeman. Tiny images. Technical Report MIT-CSAIL-TR-2007-024, Computer Science and Artificial Intelligence Lab, Massachusetts Institute of Technology, 2007.

[29] J. Wang, G. Wiederhold, O. Firschein, and S. Wei. Content-based image indexing and searching using daubechies' wavelets. *Int. J. Digital Libraries*, 1:311–328, 1998.