Using Very Deep Autoencoders for Content-Based Image Retrieval

Alex Krizhevsky and Geoffrey E. Hinton

University of Toronto - Department of Computer Science
6 King's College Road, Toronto, M5S 3H5 - Canada

Abstract. We show how to learn many layers of features on color images and we use these features to initialize deep autoencoders. We then use the autoencoders to map images to short binary codes. Using semantic hashing [6], 28-bit codes can be used to retrieve images that are similar to a query image in a time that is independent of the size of the database. This extremely fast retrieval makes it possible to search using multiple different transformations of the query image. 256-bit binary codes allow much more accurate matching and can be used to prune the set of images found using the 28-bit codes.

1 Introduction

In this paper we use very deep autoencoders to map small color images to short binary codes. For content-based image retrieval, binary codes have many advantages compared with directly matching pixel intensities or matching real-valued codes. They are very cheap to store, and they are very fast to compare using bit-wise operations. If they are sufficiently short, *e.g.* 28 bits, they can be used as addresses in a "semantic" memory space in which similar addresses contain pointers to similar images. The codes found by learning a deep autoencoder tend to have this property. Images similar to a query image can then be found by flipping a few bits in the code and performing a memory access. The retrieval time for this "semantic hashing" method is completely independent of the size of the database: billions of images can be searched in a few milleseconds. This allows some invariances to be implemented by searching with many different transformations of the query image.

In [6], autoencoders initialised by learning Deep Belief Networks (DBNs) were used to obtain short binary codes for documents and it was shown that these codes could be used for semantic hashing. For documents, however, this may be unnecessary because individual words are very informative so inverted indexing works well. Pixels or low-level image features are much less semantically meaningful, so semantic hashing is much more appropriate for image retrieval. In [8], the authors used a DBN-initialized autoencoder to generate short binary codes, but they were not able to train the DBN on the raw image data so they used the 384-dimensional GIST descriptors of the color images. Unfortunately, the GIST descriptors act as a narrow bottleneck which occurs too early in the feature extraction process to allow a DBN to learn really good short codes. As a result, there has been no proper evaluation of binary codes produced by deep learning for image retrieval.

In [9], the authors introduced a new and very fast "spectral" method for generating binary codes from high-dimensional data and showed that these spectral codes are, in some cases, more useful for image retrieval than binary codes generated by autoencoders trained on the GIST descriptors. We demonstrate that spectral codes do not work as well as the codes produced by DBN-initialized autoencoders trained on the raw pixels.

2 How the codes are learned

DBNs are multilayer, stochastic generative models that are created by learning a stack of Restricted Boltzmann Machines (RBMs), each of which is trained by using the hidden activities of the previous RBM as its training data. Each time a new RBM is added to the stack, the new DBN has a better variational lower bound on the log probability of the data than the previous DBN, provided the new RBM is learned in the appropriate way [3].

We train on 1.6 million 32×32 color images that have been preprocessed by subtracting from each pixel its mean value over all images and then dividing by the standard deviation of all pixels over all images. The first RBM in the stack has 8192 binary hidden units and 3072 linear visible units with unit variance gaussian noise. All the remaining RBM's have N binary hidden units and $2N$ binary visible units. Details of how to train an RBM can be found in [1]. We use the standard contrastive divergence learning procedure which has four steps:

1. For each data-vector , $\mathbf{v}$, in a mini-batch, stochastically pick a binary state vector, $\mathbf{h}$ for the hidden units:

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_{i \in vis} v_i w_{ij}) \tag{1}$$

   where b_j is the bias, w_{ij}, is a weight, and $\sigma(x) = (1 + \exp(-x))^{-1}$.

2. Stochastically reconstruct each visible vector as $\mathbf{v}^{'}$ using the first equation for binary visible units and the second for linear visible units, where $N(\mu, V)$ is a Gaussian.

$$p(v_i^{'} = 1|\mathbf{h}) = \sigma(b_i + \sum_{j \in hid} h_j w_{ij}), \quad \text{or} \quad v_i^{'} = N(b_i + \sum_{j \in hid} h_j w_{ij}, 1) \tag{2}$$

3. Recompute the hidden states as $\mathbf{h}^{'}$ using Eq. 1 with $\mathbf{v}^{'}$ instead of $\mathbf{v}$.

4. Update the weights using $\Delta w_{ij} \propto \langle v_i h_j \rangle - \langle v_i^{'} h_j^{'} \rangle$ where the angle brackets denote averages over the mini-batch.

To reduce noise, we actually use the probabilities rather than the stochastic binary states in steps 2, 3, and 4, but it important to use stochastic binary hidden units in step 1 to avoid serious overfitting.

The RBMs were initialized with very small random weights and zero biases and trained for 80 epochs using mini-batches of size 128. For the linear-binary RBM we used a learning rate of 0.001 on the average per-case gradient and for the binary-binary RBM's we used 0.01. We reduced the learning rate at the beginning of learning when the gradient can be big and also at the end of learning in order to minimize fluctuations in the final weights. We also used a momentum of 0.9 to speed learning (see [1] for details).

In the first RBM, most of hidden units learned to be high-frequency monotone Gabor-like filters that were balanced in the RGB channels and most of the remaining units became lower frequency filters that responded to color edges.

2.1 Fine-tuning the autoencoder

We initialized a 28-bit and a 256-bit autoencoder with the weights from two separately trained stacks of RBMs as described in [2] and fine-tuned them using backpropagation to minimise the root mean squared reconstruction error $\sqrt{\sum_i (v_i - \hat{v}_i)^2}$. In each autoencoder, the hidden layers halve in size until they reach the desired size, except that we use 28 instead of 32. To force the codes to be binary, we rounded the outputs of the logistic units in the central code layer to 1 or 0 during the forward pass but ignored this rounding during the backward pass. This is simpler and as good as the binarization method employed in [6], which adds noise to the code layer.

For the autoencoders, we used a learning rate of 10^{-6} for all layers and trained for 5 epochs. The weights only changed very slightly from the RBM weights, but the image reconstructions improved significantly. The entire training procedure for each autoencoder took about 2 days on an Nvidia GTX 285 GPU.

3 Results

We tested our models on two subsets of the 80 million tiny images dataset [7] that was produced by using various online image search engines to search for all the non-abstract English nouns and noun phrases in the WordNet lexical database. Each image in the dataset has a very noisy label, which is the word used to find it, but we did not use any labels for training.

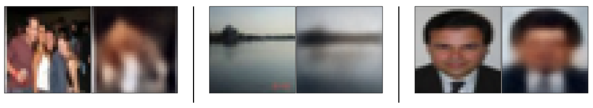
Figure 1: Reconstructions of test images from the 256-bit codes.

For a qualitative evaluation of the retrieval, we used an unlabeled subset of 1.6 million of the images for both training and retrieval. This subset contains only the first 30 images found for each non-abstract English noun. For a quantitative evaluation, we measured what fraction of the retrieved images were of the same

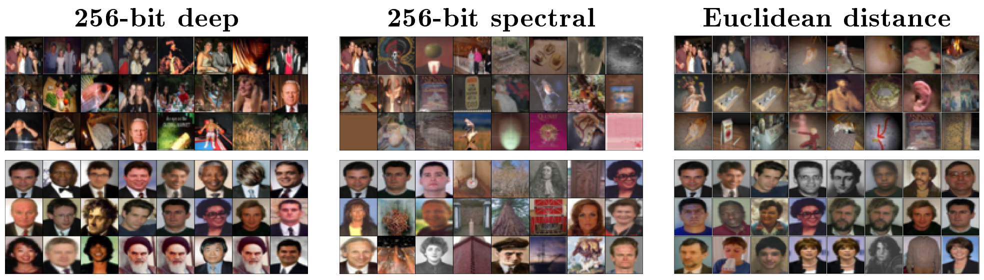

Figure 2: Retrieval results from the 1.6 million tiny images dataset using a full linear search with 256-bit deep codes, 256-bit spectral codes, and Euclidean distance. The top-left image in each block is the query image. The remaining images are the closest retrieved matches in scan-line order. The dataset contains some near-duplicate images.

class as the query image, averaged over 5,000 queries. We used exactly the same autoencoders, but used query images from the CIFAR-10 dataset [4], which is a carefully labeled subset of the 80 million tiny images, containing 60,000 images split equally between the ten classes: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. Each image in CIFAR-10 has been selected to contain one dominant object of the appropriate class and only 3% of the CIFAR-10 images are in the set of 1.6 million training images.

3.1 Retrieval results

Qualitatively, a full linear search of 1.6 million images using 256-bit deep codes produces better results than using Euclidean distance in pixel space and is about 1000 times faster. 256-bit spectral codes are much worse (see figure 2). Pruning the search by restricting it to images whose 28-bit deep code differs by 5 bits or less from the query image code only very slightly degrades the performance of the 256-bit deep codes.

Quantitatively, the ordering of the methods is the same, with 28-bit deep codes performing about as well as 256-bit spectral codes (see figure 3). The best performance is achieved by a more elaborate method described below that creates a candidate list by using many searches with many different 28-bit codes each of which corresponds to a transformed version of the query image.

4 Multiple semantic hashing

Semantic hashing retrieves objects in a time that is independent of the size of the database and an obvious question is whether this extreme speed can be traded for more accuracy by somehow using many different 28-bit coding schemes and combining their results. We now describe one way of doing this.

Codes obtained by looking at the entire image are good for capturing global structure, and hence for finding images that appear globally similar, but they are not invariant to transformations like translation of one object within the image which often have little effect on the semantic content. We can achieve invariance to translations of medium-sized objects by treating each image as a bag of patches and training an autoencoder on local patches of images. We obtained the patches by sampling the 1.6 million tiny images with the variable-resolution "retina" shown in Figure 4. This autoencoder had the following architecture: 336-1024-512-256-128-64-28 and was trained in the same way as before.

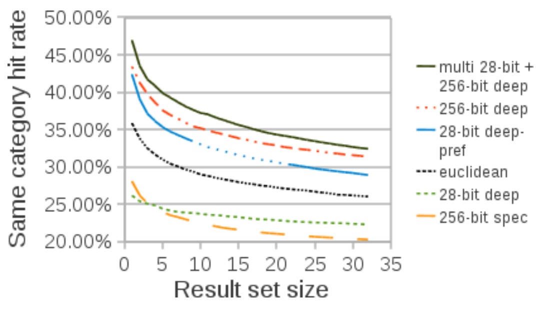

Figure 3: CIFAR-10 retrieval performance of the various methods tested.

4.1 Search

After some initial exploration of various scoring methods we found that the following procedure worked best.

First, construct a 2^{28}-element array (the semantic hashing table) that stores at index i the list of images that have image patches with code i (possibly with repetitions, as one image may have multiple patches with the same code).

Second, for each of the 81 patches of the query image, use the semantic hashing table to quickly find the set of images that have patches with codes no more than 3 bits away from the query patch code. Assign to each found image the score 2^{3-d}, where d is the hamming distance of the query patch code from the found image patch code. We explore the hamming ball around the query patch code in order of increasing distance, so each found image is given the maximal score it could get for a single patch. It does not get additional score if the same query patch matches some other part of the same image equally or less well. This avoids problems such as one query patch of sky matching a large number of other patches of sky in another image.

Third, sum the scores for each image over all of the 81 patches of the query image and return the images in order of descending score. Finally, combine the ranked list of images found in the second step with the ranked list of images found using 256-bit deep codes by adding the two ranks of each image and re-ordering them.

5 Future work

Deep autoencoders that are pre-trained as DBNs also work well for document retrieval [6], so it should be relatively easy to learn deep binary codes that are good for both reconstructing the image and reconstructing a label or bag of words representation of an image caption. This would encourage the deep codes to extract more semantic information from the images and it scales much better with the size of the database than methods with quadratic time complexity such as non-linear NCA [5] that was tried in [8].

6 Summary

Pre-training with RBM's makes it possible to learn very deep autoencoders that map similar images to similar binary codes. This allows the speed of hash-coding to be applied to approximate matching. This is particularly relevant for content-based image retrieval and we have shown that it works considerably better than a previous attempt [8]. The images returned by searching with 256-bit codes are qualitatively and quantitatively better than the near neighbors in pixel space and this advantage is preserved when using semantic hashing with 28-bit codes to eliminate nearly all of the database from the search.

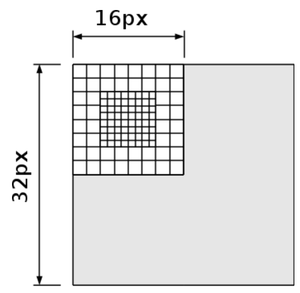

Figure 4: The variable-resolution "retina" used to sample the 32×32 images. The retina looks at at a 16×16 patch of the image, but contains only 112 pixels, because the pixels on the outside are sampled at lower resolution. Using a stride of 2 pixels, there are 81 distinct patches that can be extracted from a 32×32 image in this way.

References

[1] G. E. Hinton. A practical guide to training restricted boltzmann machines. Technical Report UTML2010-003, University of Toronto, 2010.

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

[3] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

[4] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto, 2009.

[5] R. Salakhutdinov and G. E. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. In *Proceedings of AI and Statistics*, 2007.

[6] R. Salakhutdinov and G. E. Hinton. Semantic hashing. In *Proceedings of the SIGIR Workshop on Information Retrieval and Applications of Graphical Models*, Amsterdam, 2007.

[7] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large database for non-parametric object and scene recognition. *IEEE PAMI*, 30 (11):1958–1970, November 2008.

[8] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proceedings of the IEEE Conf on Computer Vision and Pattern Recognition*, 2008.

[9] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proceedings of Neural Information Processing Systems*, 2008.