

VISOR: Towards On-the-Fly Large-Scale Object Category Retrieval

Ken Chatfield and Andrew Zisserman

Department of Engineering Science, University of Oxford, United Kingdom
{ken,az}@robots.ox.ac.uk

Abstract. This paper addresses the problem of object category retrieval in large unannotated image datasets. Our aim is to enable both fast learning of an object category model, and fast retrieval over the dataset. With these elements we show that new visual concepts can be learnt on-the-fly, given a text description, and so images of that category can then be retrieved from the dataset in realtime.

To this end we compare state of the art encoding methods and introduce a novel cascade retrieval architecture, with a focus on achieving the best trade-off between three important performance measures for a realtime system of this kind, namely: (i) class accuracy, (ii) memory footprint, and (iii) speed.

We show that an on-the-fly system is possible and compare its performance (using noisy training images) to that of using carefully curated images. For this evaluation we use the VOC 2007 dataset together with 100k images from ImageNet to act as distractors.

1 Introduction

Over the last decade there have been great strides forwards in large scale image retrieval and classification. On the one hand methods have been developed for instance search [1–3] and near duplicate image search [4] which have been scaled to millions of images. These methods, which include Google Goggles, cast the problem as one of nearest neighbour matching to a query vector, with the entire dataset searchable in real time. On the other hand methods have been developed for image classification where the approach proceeds in two stages: first, a classifier is trained on a set of annotated data for a category; and, second, the dataset is ranked based on the classifier score [5–7].

However, as the size of datasets has increased, so it has become more evident that the expensive training of a proportionally increasing number of tailor-made classifiers, with all of the data annotation that implies, is not a sustainable approach. For example, for the ImageNet dataset [8] there are 14M images and 22k classes (concepts) to train for. This paper attempts to bridge these two classes of methods, and facilitate on-the-fly retrieval of object categories (rather than just object instances). We use as a basis for this work recent advances in both large-scale image classification [5–7] and large-scale retrieval [9, 8], with the key distinction being that we train object category models in realtime using the

web as a source of training images. This allows semantic queries to be made (e.g. ‘retrieve all images containing cars’) without limiting the queries to a bank of pre-trained classifiers. This is made possible by speeding up the training stage so that it can operate ‘on-the-fly’, training new concepts as they are needed.

The general methodology is as follows – we allow the user to specify a text query describing the object category of interest, and this is then used to query Google Image Search. The top N images returned (in our case $N = 150 \sim 200$) are then used as positive training images for the class, with an SVM trained against a predefined pool of negative training images (see Figure 1). Our recent work on on-the-fly person retrieval also makes use of Google Image search in a similar way but for a different purpose [10].

This is not the first time Google Image Search has been used in the literature as a source of images [11–15], but often the goal has been simply to re-rank the downloaded images to promote the target class. In contrast to this previous work, we take the next logical step and use the images to build new object class models on-the-fly. Although many of the problems with such an approach, as described in Schroff et al. [15], remain (in particular the problem of polysemy — e.g. ‘Jaguar’ the car versus ‘Jaguar’ the cat) the results of Google Image Search have improved significantly since these papers were published, presumably as a result of the incorporation of click-through data by Google, to the extent that for most queries the first 100 or so top ranking images are for the most part free of non-class images.

As in the case of large-scale instance retrieval [16], the two main considerations which are most critical when scaling up to web-scale datasets are: the time taken for the search itself, which tends to increase as the size of the dataset increases, and the memory usage per image, as this determines the overall memory footprint. Whilst in the case of instance (specific object) search, the search time is primarily determined by the generation of a ranked list of results, in the more general case of object category recognition we must also take into account the time required to train an object category model for each query.

In this paper, we make the following three contributions: (i) we compare a number of image representations (Section 2) and evaluate their suitability for use in large-scale object category retrieval, specifically considering their relative accuracy, memory requirements and computation speed (Section 5); (ii) we develop a method for fast ranking based on a cascade of classifiers, which results in a 50% increase in ranking speed with minimal performance penalty (Section 3); and (iii), we propose and demonstrate a method for on-the-fly training of object category classifiers using images sourced from the web (Section 6).

2 Image Representation – Review

There has been a large body of work in the area of large-scale image classification in recent years (e.g. see the PASCAL VOC workshops). The general trend has been to move towards using more sophisticated encoding schemes so that simple linear SVMs can be used. Chatfield *et al.* [17] give an overview of recent encoding approaches, and show that the Fisher Kernel, first proposed in [18], applied by

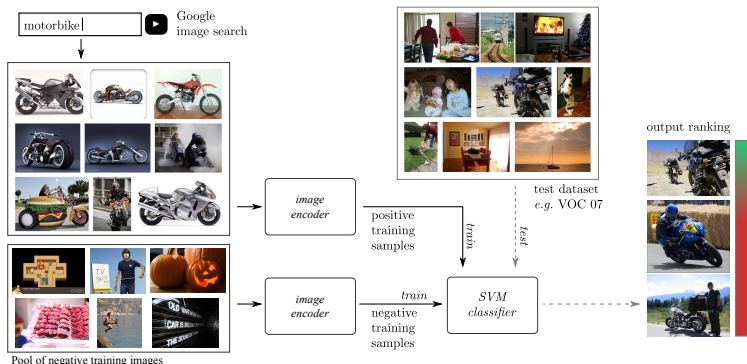


Fig. 1. Architecture of our proposed on-the-fly object category retrieval system. The user specifies a text query which is used to retrieve visual training data from Google Images. These images are encoded in real-time, and used to train a linear SVM against a pool of fixed negative training data (also sourced from Google Image search). The model from the SVM is then used to rank images in the target dataset and the results displayed to the user. The entire process from typing the query to obtaining the ranked images takes only a matter of seconds.

Perronnin and Dance to image classification [19] and then extended in [20], provides the best performance over the PASCAL VOC 2007 classification task.

In the rest of this section we first review three image representations (these will be used in the comparison experiments of Section 5), and then describe the descriptor compression that is used to reduce the disk and memory requirements.

2.1 Image descriptors and encodings

Each image is first represented by a dense set of SIFT features [21] computed on a spatial grid over the image. At each grid node, SIFT is computed at four scales, defined by setting the width of the SIFT spatial bins to 4, 6, 8 and 10 pixels respectively. The rotation of the SIFT features is fixed to a constant value [22]. A total of N SIFT features are computed, $\mathbf{x}_1, \dots, \mathbf{x}_N$, with $N \sim 150,000$ per image in our case.

Histogram encoding otherwise known as *bag-of-words* provides the baseline encoding method in this paper. The construction of the encoding starts by learning a k -means visual vocabulary μ_1, \dots, μ_K . Given a set of SIFT features sampled from an image, let q_{ki} be the assignments of each descriptor \mathbf{x}_i to the corresponding visual word as given by $q_{ki} = \operatorname{argmin}_k \|\mathbf{x}_i - \mu_k\|^2$. The histogram encoding of the set of local descriptors is the non-negative vector $\mathbf{f}_{\text{hist}} \in \mathbb{R}^K$ such that $[\mathbf{f}_{\text{hist}}]_k = |\{i : q_{ki} = k\}|$. The descriptor is a K -dimensional vector.

Fisher encoding [20] captures the average first and second order differences between the SIFT features and the centres of a Gaussian Mixture Model (GMM), which can be thought of as a soft visual vocabulary. The construction of the encoding starts by learning a GMM model θ with K components. Let q_{ki} , $k = 1, \dots, K$, $i = 1, \dots, N$ be the soft assignments of the N SIFT features to the K Gaussian components. For each $k = 1, \dots, K$, define the vectors $\mathbf{u}_k =$

$\frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^N q_{ki} \Sigma_k^{-\frac{1}{2}}(\mathbf{x}_i - \mu_k)$ and $\mathbf{v}_k = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^N q_{ki} ((\mathbf{x}_i - \mu_k) \Sigma_k^{-1} (\mathbf{x}_i - \mu_k) - 1)$ then the Fisher encoding is given by: $\mathbf{f}_{\text{Fisher}} = [\mathbf{u}_1^\top, \mathbf{v}_1^\top, \dots, \mathbf{u}_K^\top, \mathbf{v}_K^\top]^\top$. The descriptor is a $2dK$ -dimensional vector, where d is the dimension of the SIFT feature (128). Note, in general, K which is here the number of mixture components, can be much smaller than the number of visual words used in histogram encoding, so the final descriptor vector need not be extremely high dimensional. This method has been shown to produce excellent performance both for the image classification task [20] and for large-scale retrieval [20, 7].

PiCoDes [23]¹ is an alternative encoding scheme which, unlike both histogram and Fisher encoding, was designed specifically to be compact. As a result, even without applying the compression methods described below in Section 2.2, a very low-dimensional image representation is obtained. The PiCoDes encoding is based on the classeme idea of Torresani *et al.* [24] where a pre-learned set of non-linear image classifiers are applied to a feature vector computed from the image (using vector quantized dense SIFT as above, together with other features). In PiCoDes, the thresholded output of the individual classifiers form the components of the image descriptor vector. Consequently, the PiCoDes encoding is a binary vector.

Compared to both histogram and Fisher encoding, which generally result in floating point image descriptors of thousands of dimensions, PiCoDes can be stored in as little as 128-bits. Here we use the PiCoDes-2048 encoding described in Bergamo *et al.* [23], which is a 2048-bit (256 byte) encoding.

Spatial pyramids are a standard way of introducing weak geometry into the image descriptor using spatial histograms [25, 26]. The idea is that the image is split into overlapping spatial regions of different sizes, and one encoding is computed for each spatial region. These encodings are then stacked to form the final image representation, resulting in an increase in the dimensionality of the descriptor from D to bD where b is the number of spatial bins.

In this paper, we use a spatial pyramid comprising eight regions for both the histogram and Fisher encodings. These regions are obtained by dividing the image in 1×1 , 3×1 (three horizontal stripes), and 2×2 (four quadrants) grids, and thus the resultant descriptor is of dimensionality $8D$.

2.2 Dimensionality reduction and compression

In order to facilitate fast ranking over large-scale datasets, it is desirable to store the encodings of all images in memory. However, as the number of images increases, it becomes less and less practical to store the full image encoding in core. For example, for the ILSVRC 2011 dataset, which comprises around 1.2M images, storing a bag-of-words encoding for each image with even a moderately small vocabulary size of 4,000 and using spatial pyramids would require 143GB of memory. When so much memory has to be visited during the ranking stage, this also begins to have an impact on the speed of retrieval, consequently some form of image descriptor compression is necessary.

¹ http://vlg.cs.dartmouth.edu/projects/vlg_extractor/vlg_extractor/

Product Quantization (PQ) was introduced in [9] as a compression method for SIFT features, and has since also been applied to the compression of image descriptors. Sánchez and Perronnin [7] provide a comprehensive review of compression methods, including product quantization, and show that it offers the best performance for a given compression ratio. Consequently, we apply product quantization to all our codes (aside from PiCoDes which is already compact) to reduce the storage requirements and make operation over large datasets feasible. For the histogram encoding method, a fairly aggressive compression ratio of up to 32 (reducing from 32K floats to 8K chars) is achievable without significant reduction in performance, with mAP over the VOC 2007 classification task (as described in Section 5), dropping from 52.63 for the full representation to 50.90. We adopt a more conservative $16\times$ compression ratio, resulting in a reduction of our code size from a 32K dimensional floating point representation to a 8K dimensional vector of bytes whilst incurring only a 0.07% drop in mAP.

The Fisher encoding representation results in a 327,680 dimensional code before compression, and as a result a more aggressive compression of eight dimensions per subquantizer is used giving a compression ratio of 32 and a reduction to a 40,960 dimensional vector of bytes. Also, an additional dimensionality reduction is applied to the SIFT features using PCA prior to encoding. This has been shown to actually improve the performance of the resultant representation [27]. For a vocabulary of 256 a reduction to 80-dimensions was found to be optimal (giving a further 37% reduction).

3 Speeding-up Retrieval

An image descriptor is computed (using one of the methods from the previous section) for all images in the target dataset. To retrieve images of a particular class (e.g. cars) from the target dataset, a linear SVM is first learnt for that class (the training data is described in the following section) and then the learnt weight vector is used to rank images by their distance from the decision boundary, *i.e.* if the image is represented by the vector \mathbf{x} and the learnt weight vector for the SVM is \mathbf{w} , then images are sorted on the value $\mathbf{w}^T\mathbf{x}$ (both \mathbf{x} and \mathbf{w} have unit Euclidean norm). In this section we describe a cascade method to compute the ranking more efficiently than exhaustive search.

If the learnt weight vector \mathbf{w} was sparse, then ranking images could be carried out efficiently using an inverted index in much the same way as large scale instance retrieval [28, 2, 1] – both operations are just scalar products between a vector and \mathbf{x} . In instance retrieval the vector used is the tf-idf BoW vector of the query image, whilst here it is the learnt weight vector \mathbf{w} . However, in our case the weight vector \mathbf{w} is not sparse, and other methods must be employed. A partial solution is offered by the use of product quantization, as the distance between all subquantizers and the \mathbf{w} vector can be pre-computed, reducing the scalar product to a series of LUT lookups [29]. However, ranking time still increases linearly with dataset size, limiting the scalability of the approach.

SVM cascade. Our solution to this problem is to propose a cascade of classifiers, with only the first level of the cascade being applied to the entire dataset.

Although the use of cascades is not uncommon in the object detection literature *e.g.* [30], as far as we are aware this is the first time such an approach has been applied to speed-up retrieval across collections of images. The initial level of the cascade operates on a simplified set of features (*e.g.* in the case of histogram encoding, encodings generated using a smaller visual vocabulary or without spatial binning). This is used to generate a shortlist of images with a higher probability of containing the class instance, and this shortlist is then re-ranked using the full representation.

4 Datasets and Evaluation Protocol

The following datasets are used to evaluate the performance of each method:

PASCAL VOC 2007 [31] consists of 9,963 images split into train, validation and test sets, and labelled with twenty object classes. Evaluation is performed over the 4,952 images of the test set using the standard VOC image classification protocol (where a ranking of the test images is generated for each of the twenty classes according to whether they contain that class or not).

PASCAL VOC 100k (VOC 2007 + distractors) is a combination of the 4,952 images in the test set of PASCAL VOC 2007 dataset with a randomly selected subset of 90,694 images from the ILSVRC 2011 dataset [8], which act as distractors, to form a new dataset containing around 100,000 images.

Evaluation protocol. The performance is computed using two standard measures: first, the Average Precision for each class (using the VOC supplied code), and second the precision@ K , i.e. the precision at K images, which gives an indication of the number of false positives in the first few pages of results in our retrieval system (with $K = 10, 50$) and thus the subjective ‘goodness’ of the results. In the retrieval context a good performance over the first few pages is often more critical than achieving high recall. For the VOC 100k evaluations those ILSVRC 2011 classes which intersect semantically with each VOC class name are ignored in the ranking list when evaluating over that class (*e.g.* for the VOC aeroplane class, all images from the plane, warplane, airliner and wing synsets from ILSVRC 2011 are ignored in the ranking list), otherwise those images would be counted falsely as negatives (as even though they are of the correct class, they are not in the positive set of the VOC test images for that class).

5 Experiments

The experiments are divided into four steps. First, the performance of the image descriptors is evaluated over the VOC 2007 test data using VOC train+val for training. This scenario corresponds exactly to the standard VOC image classification task so that results are comparable with the published state of the art on this dataset.

Second, the 90k distractors are added to the test data (to form the VOC 100k dataset) and performance is evaluated. This determines how the performance deteriorates with the addition of distractors. We select the histogram encoding method only to compare at this stage since it is an order of magnitude quicker

to compute than the Fisher encoding and is also of lower dimensionality, thus being well suited to the fast online computation of codes for training images.

Up to this point evaluation has used exhaustive search. For the third set of experiments, exhaustive search is compared to the cascade approaches of Section 3. As well as measuring the retrieval accuracy, now the speed gain offered by this approach is evaluated. The perfect outcome would be that for the cascades there is no loss in retrieval accuracy (compared to exhaustive search) but a considerable increase in ranking speed.

Finally, for the fourth set of experiments, the training data is changed from VOC train+val to the Google downloaded images obtained using the class name. In this case the comparison is between training on a well curated training set (from VOC) and somewhat noisy images obtained from Google image search. For the positive training set, 200 images are downloaded from Google Image Search for each of the VOC classes. For the negative training set, a fixed pool of $\sim 1,000$ negative training images, sourced from Google using the query term ‘things’ and ‘photos’ is used. Tests are also performed using an expanded negative training set of 10,000 images from the ILSVRC 2011 dataset, disjoint from those images used as distractors and with all synsets related to the VOC classes removed.









The evaluation pipeline is based on the code of [17], with the VLFeat library [32] used to compute dense SIFT features using the `vl_phow` routine. A Hellinger kernel is used for histogram and Fisher encodings as described in [33].

5.1 Analysis

Experiment 1: PASCAL VOC 2007 encodings. The results are reported in Table 1. As expected, the Fisher Encoding offers the best performance over the dataset, with the simpler histogram encoding trailing behind. The cost is in computation time, with the Fisher encoding at 2 seconds per image taking more than double the time to compute when compared to the histogram encoding in our C++ implementation. The results are slightly lower than the state-of-the-art reported in [17] due to the application of product quantization, and also since the C parameter of the SVM has not been optimized here on a validation set (as a fixed value of $C = 10$ is used to allow for the on-the-fly nature of the system).

The PiCoDes-2048 encoding performs well given that it produces much more compact codes than the other encodings. Comparing with a reduced-dimension histogram encoding (row (c)), using a visual vocabulary of size 300 and a 256-bit PQ subquantizer for every 4-bits of the histogram output to give a code of size 75 bytes, it can be seen that the 256 byte PiCoDes encoding performs marginally better and is also slightly more consistent between classes, in some cases achieving a higher *precision@50* than even the higher-dimensional 4K+SPM histogram encoding (compare ‘bird’ and ‘potted plant’ in (b)-(d)). However, again the cost is in computation time, with the C implementation provided at [34] taking on average over 3 seconds per image to compute.

Experiment 2: adding distractors. As seen in Table 2, adding the distractors results in a substantial drop in performance as measured by the mean Average Precision (mAP). However, when considering the performance of the system as

Method	mean								
(a-1) FV 256-8	<i>prec@10</i>	92.5	100	100	100	60	100	100	100
(b-1) VQ 4K	<i>prec@10</i>	80.0	100	100	50	100	20	100	90
(c-1) VQ 300 (no SPM)	<i>prec@10</i>	56.5	100	70	30	90	20	50	90
(d-1) PiCoDeS-2048	<i>prec@10</i>	72.5	100	90	80	90	10	70	100
(a-2) FV 256-8	<i>prec@50</i>	85.1	96	94	72	100	46	96	100
(b-2) VQ 4K	<i>prec@50</i>	71.8	94	88	52	96	24	70	100
(c-2) VQ 300 (no SPM)	<i>prec@50</i>	47.2	86	62	28	86	10	36	76
(d-2) PiCoDeS-2048	<i>prec@50</i>	59.1	92	68	66	86	20	58	86
(a-3) FV 256-8	<i>mAP</i>	58.90	70.74	65.91	49.24	68.80	28.64	67.10	77.55
(b-3) VQ 4K	<i>mAP</i>	45.39	66.91	50.77	31.51	62.53	14.58	45.49	67.78
(c-3) VQ 300 (no SPM)	<i>mAP</i>	27.80	53.14	27.39	14.87	48.15	8.47	16.35	54.09
(d-3) PiCoDeS-2048	<i>mAP</i>	34.61	60.99	31.99	32.95	47.63	10.31	30.94	51.04





















												
(a-1) <i>prec@10</i>	100	90	100	70	100	100	100	80	100	80	100	70
(b-1) <i>prec@10</i>	90	60	50	60	100	100	100	30	70	100	100	80
(c-1) <i>prec@10</i>	70	10	20	10	90	60	90	20	50	60	100	40
(d-1) <i>prec@10</i>	90	70	60	50	100	80	100	50	40	50	90	50
(a-2) <i>prec@50</i>	84	72	82	70	100	96	100	62	72	78	98	88
(b-2) <i>prec@50</i>	82	44	60	46	100	92	100	26	62	60	96	62
(c-2) <i>prec@50</i>	66	20	14	18	66	48	94	12	18	40	76	38
(d-2) <i>prec@50</i>	60	42	46	36	90	68	90	26	36	32	80	48
(a-3) <i>mAP</i>	50.32	49.07	52.82	43.35	79.23	67.94	82.81	31.92	49.37	52.07	77.66	54.26
(b-3) <i>mAP</i>	44.34	29.80	31.86	28.69	70.98	53.07	74.06	13.12	36.59	39.53	66.58	35.27
(c-3) <i>mAP</i>	35.04	10.73	10.55	15.01	35.38	21.21	67.03	7.86	15.09	22.91	47.46	20.53
(d-3) <i>mAP</i>	33.94	24.80	28.19	23.20	56.14	33.82	62.44	12.79	21.09	21.75	54.39	24.84

Table 1. Performance over VOC 2007 dataset, using train+val sets as training data.

measured in terms of precision@ K images, although there is a drop, for many classes (*e.g.* ‘car’, ‘horse’, ‘motorbike’, ‘train’ in row (a)) still over 70% of the first 10 images are true positives, and over 50% at 50 images.

Experiment 3: cascade vs exhaustive search. Rows (b) and (c) in Table 2 show the results of our SVM cascade experiments. In both cases, we use a reduced histogram encoding based on a vocabulary of 300 visual words and no SPM to rank the results coarsely (the standalone performance of which is shown in Table 1 row (c)) followed by a re-ranking of the top Z images using our full histogram encoding where $Z = 10k$ in (b) and 1k in (c). For the first configuration, the drop in accuracy through the use of the cascade can be seen to be very small. The penalty for moving to a 1k shortlist is greater, but still in many cases the performance drop occurs further down the ranking list, with the precision@ K

Method	mean									
(a-1) VQ 4K	<i>prec@10</i>	40.0	60	40	0	60	0	50	70	20
(b-1) VQ 4K <i>10k casc.</i>	<i>prec@10</i>	39.5	60	40	0	60	0	40	70	20
(c-1) VQ 4K <i>1k casc.</i>	<i>prec@10</i>	38.0	40	50	0	60	0	40	70	30
(a-2) VQ 4K	<i>prec@50</i>	28.5	52	36	0	28	4	40	54	16
(b-2) VQ 4K <i>10k casc.</i>	<i>prec@50</i>	28.3	52	34	0	28	6	36	58	12
(c-2) VQ 4K <i>1k casc.</i>	<i>prec@50</i>	25.2	46	22	0	28	2	20	64	14
(a-3) VQ 4K	<i>mAP</i>	10.29	18.65	9.1	0.59	14.65	1.43	13.71	19.08	3.78
(b-3) VQ 4K <i>10k casc.</i>	<i>mAP</i>	9.54	17.56	7.22	0.42	14.42	1.34	10.73	19.69	2.71
(c-3) VQ 4K <i>1k casc.</i>	<i>mAP</i>	5.71	8.21	2.97	0.38	11.88	0.65	2.96	14.93	1.59





















													
(a-1) <i>prec@10</i>	50	10	40	10	100	70	50	0	30	40	90	10	
(b-1) <i>prec@10</i>	40	10	40	20	100	70	40	0	30	50	90	10	
(c-1) <i>prec@10</i>	50	10	20	10	100	60	40	10	30	40	90	10	
(a-2) <i>prec@50</i>	30	6	20	6	86	46	22	4	14	32	60	14	
(b-2) <i>prec@50</i>	30	8	20	6	88	44	22	6	12	30	60	14	
(c-2) <i>prec@50</i>	38	8	16	2	82	22	34	2	14	14	70	6	
(a-3) <i>mAP</i>	8.25	2.54	8.61	1.87	35.02	14.34	9.36	1.03	4.41	9.38	25.53	4.45	
(b-3) <i>mAP</i>	7.92	2.2	6.48	1.08	35.3	12.25	8.46	0.92	4.59	8.65	24.87	4.03	
(c-3) <i>mAP</i>	6.89	1.14	2.22	0.7	20.36	4.01	7.24	0.6	3.26	3.04	18.92	2.19	

Table 2. Performance over VOC 100k dataset, using train+val sets as training data.

measures experiencing less of a drop (e.g. ‘boat’, ‘car’, ‘horse’) despite only 1% of the dataset being passed through the full object model.

Experiment 4: Google training images. Finally, we switch to Google Image search as the source of our training data in Table 3. Again, taking the average across all classes the performance drops across the board when moving away from curated training data. However, what is interesting is that for several of the VOC classes there is very little drop in performance at all (e.g. compare ‘bicycle’, ‘bus’, ‘car’, ‘cow’ in row (b) of Table 3 to row (a) of Table 2) indicating that even with the compromises made with the negative training data (where a fixed pool of generic images are used) the quality of the positive data returned from Google is in some cases sufficient to train very good models. Adding extra negative training data from ILSVRC 2011 does, however, improve the results as seen in row (c). The relative performance of all of our methods, as measured by precision@50, can be seen in Figure 2.

Retrieval accuracy vs time and memory footprint. Table 4 summarises the time taken for each step in the computation pipeline, along with the speed gains achieved by employing the SVM cascade. It can be seen that a greater than 50% speed increase is achievable over our dataset of 100k images with negligible

Method	mean									
(a) VQ 4K + SPM	<i>prec@10</i>	58.5	70	100	80	30	10	90	100	70
over VOC 2007	<i>prec@50</i>	47.1	84	72	46	36	10	66	94	68
	<i>mAP</i>	27.36	47.59	34.24	22.06	18.04	5.65	30.75	58.33	35.73
(b) VQ 4K + SPM	<i>prec@10</i>	15.0	0	50	0	0	0	60	80	0
over VOC 100k	<i>prec@50</i>	12.1	8	36	0	2	0	38	66	4
	<i>mAP</i>	3.45	2.06	7.36	0.36	2.31	0.18	11.97	17.57	1.27
(c) VQ 4K + SPM	<i>prec@10</i>	24.5	20	80	0	0	0	70	70	30
over VOC 100k	<i>prec@50</i>	15.9	16	50	0	4	0	38	70	18
+extra negatives	<i>mAP</i>	4.68	3.26	12.47	0.41	2.40	0.31	11.71	18.37	3.40













												
(a) <i>prec@10</i>	20	70	70	40	60	70	70	0	80	30	100	10
<i>prec@50</i>	12	34	44	36	60	62	76	0	36	16	78	12
<i>mAP</i>	14.4	21.35	24.93	22.46	30.47	30.37	51.85	7.18	26.88	10.30	40.47	14.2
(b) <i>prec@10</i>	10	30	0	0	10	10	0	0	20	0	30	0
<i>prec@50</i>	2	14	6	0	10	8	0	0	14	4	30	0
<i>mAP</i>	0.52	3.58	1.57	0.99	2.19	2.97	1.82	0.31	2.49	0.59	8.23	0.55
(c) <i>prec@10</i>	10	20	10	0	10	40	0	0	50	30	50	0
<i>prec@50</i>	2	14	8	6	8	20	4	0	22	6	32	0
<i>mAP</i>	0.75	3.39	2.27	1.46	3.93	5.58	3.23	0.46	6.70	1.43	11.54	0.65

Table 3. Performance over both the VOC 2007 and VOC 100k datasets, using images from Google image search as training data.

impact on performance. We have found even greater gains are achievable when working with datasets larger than 1M+ images. For example, with 3M images, the ranking time is reduced from around 15 seconds to around 4-5 seconds using the cascade. Also, using PQ the footprint to store all features for 3M images in memory is only 23GB.

Summary: given the accuracy, memory and speed trade offs, the histogram or Fisher encoding are the methods of choice. We use these for the on-the-fly system described in the following section.

6 The On-the-fly System

Although we have presented experimental results for the VOC 2007 classes in the previous section, as described in the introduction the advantage of using Google Images as a source of training data is that the user is not limited to any fixed list of object categories. We have developed a web-based frontend which allows the user to specify arbitrary queries which are then learnt and applied to our target dataset ‘on-the-fly’.

	Time (s)	Memory saving (GB)	Perf. Penalty (prec@50)
Feature Computation	14.7		
Learning Linear SVM	6.49		
Ranking Original Features	1.41	119.2	
PQ (using LUT)	1.10	22%	7.44 93.8% 0.0
Cascade 10k	0.62	56%	7.51 93.7% -0.2
Cascade 1k	0.53	62%	7.51 93.7% -3.3

Table 4. Processing time for each stage of the system, with a comparison of the speed of different ranking methods.

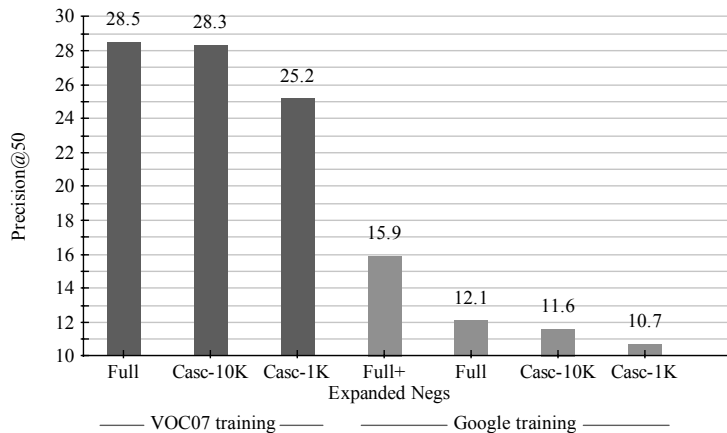


Fig. 2. Comparison of Precision@50 for the histogram encoding method, with and without use of a cascade and using images from the VOC 2007 train+val sets for training (left) or images sourced from Google Image search (right).

As illustrated in Figure 1 there are four stages to the on-the-fly pipeline. First, positive training images are obtained using Google image search starting from text. Second, a descriptor is computed for each of the positive images. Third, a linear SVM classifier is trained using these positive image descriptors and a pool of negative images (with pre-computed descriptors) to obtain a weight vector \mathbf{w} . Fourth, the data set is ranked by the classifier.

In the following we discuss these steps and describe how they are implemented efficiently on a multiple-core system.

Obtaining positive training images. For the positive training set, we attempt to download around 150-200 images from Google in parallel. Given the variability of the servers on which the images linked to from Google Image search are hosted, we do this by first requesting around 300 results from Google, and then impose a timeout of 100ms on the download time of each image, which ensures that no undue time is wasted retrieving images from slow servers. Features are then computed in real-time over multiple CPU cores (in our case 50) and stored in memory for training. When querying Google Image search for training data, we

ask it only to return ‘photos’, and this avoids many of the problems reported by [15] caused by the introduction of drawings or cartoons of object categories.

Learning the classifier. A key component of our method is the training in real-time of new object category models. However, as a result, if we are to make the system operate at an acceptable speed, we need to ensure this training is completed as quickly as possible. We achieve this by parallelising the SVM stage by splitting the negative training samples over multiple parallel SVM training runs, with all of the positive training images used in each case. Given that for the histogram encoding the image codes are all non-negative, we can then combine the resultant models by taking the maximum of all positive values in the \mathbf{w} vector (which relate to upweighting features common in the positive training instances) and taking the mean average of all negative values (relating to features common in the negative training instances). The combined model is then passed through a final SVM ‘polishing’ round before application. In this way, we have found we can achieve some improvement in training speeds with minimal adverse effect to performance. The SVM training is carried out using stochastic gradient descent [35].

Fast retrieval. After the model has been learnt, we apply the SVM cascade method described in Section 3 to speed up ranking, with a shortlist of 10k used for the VOC 100k dataset a ranking time of around 500ms is possible using this approach, with all features stored in under 10GB of memory.

Qualitative performance. Qualitatively, the system works for a very broad range of queries, with particularly good performance over ‘scene’ type queries such as ‘forests’, ‘underwater’ and ‘protest’ (see Figure 3). As seen in the results in Section 5, there are also many object categories the system works well on such as ‘car’ or ‘bus’. The overall on-the-fly system is very responsive – from typing the text query to receiving the ranked results is only a matter of a few seconds: ~ 6 seconds to train the classifier, and less than a second to rank 100k images.

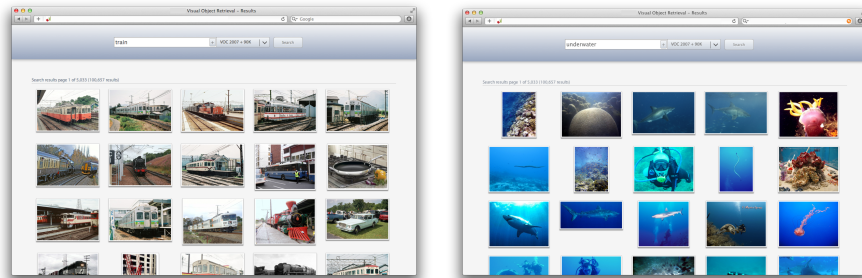


Fig. 3. The result of two queries using our live demo system. *Left:* example of VOC class ‘train’ *Right:* an example of a novel class ‘underwater’, showing how the ability to learn new models on-the-fly allows retrieval of a far broader range of images than possible with just a pre-trained bank of classifiers.

We have also run the system over a dataset of 3M images extracted from video archives over which it continues to perform well, with the ranking time increasing slightly to 4-5 seconds over 50 CPU cores.

The problem of polysemy remains, and a query for ‘apple’, for example, will not work well as the training data contains both ‘apple’ the fruit and images of products of ‘Apple’ the technology company. At present, we do not attempt to resolve this problem automatically, but instead provide the user with the option to select a set of positives and retrain the SVM – a form of relevance feedback.

7 Summary and extensions

We have investigated encodings and cascades in terms of their accuracy, speed and memory requirements with a view to their suitability for use within an on-the-fly category retrieval system, and have demonstrated that such a system is feasible. The on-the-fly system means that image datasets with no annotation can be searched for any suitable object category by ‘borrowing’ annotation information using Google image search.

There are many extensions of this work, and we conclude by mentioning a few. First, one could attempt to deal with the problem of visual polysemy (jaguar car, jaguar animal) in the Google training images by clustering images according to word sense. Second, attributes or modifiers can be added to the retrieval system. For example, adding filters on colours so that, e.g. only red cars are retrieved. Third, in this paper we have investigated category level retrieval, but we note that a similar on-the-fly learning method could be applied for instance search, e.g. if looking for a particular building, like Notre Dame, multiple image examples of the object/building could again be downloaded using a web based image search and issued as queries, an approach investigated further in our paper [36].

Acknowledgements

We are grateful for financial support from the EPSRC, ERC grant VisRec no. 228180, and EU Project FP7 AXES ICT-269980.

References

1. Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A.: Object retrieval with large vocabularies and fast spatial matching. In: Proc. CVPR. (2007)
2. Nister, D., Stewenius, H.: Scalable recognition with a vocabulary tree. In: Proc. CVPR. (2006)
3. Sivic, J., Zisserman, A.: Video Google: A text retrieval approach to object matching in videos. In: Proc. ICCV. Volume 2. (2003) 1470–1477
4. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: Proc. CVPR. (2010)
5. Perronnin, F., Sanchez, J., Liu, Y.: Large-scale image categorization with explicit data embedding. In: Proc. CVPR. (2010)
6. Perronnin, F., Akata, Z., Harchaoui, Z., Schmid, C.: Towards good practice in large-scale learning for image classification. In: Proc. CVPR. (2012)
7. Sánchez, J., Perronnin, F.: High-dimensional signature compression for large-scale image classification. In: Proc. CVPR. (2011)
8. Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Proc. CVPR. (2009)

9. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. *IEEE PAMI* (2011)
10. Parkhi, O.M., Vedaldi, A., Zisserman, A.: On-the-fly specific person retrieval. In: *Intl. Workshop on Image Analysis for Multimedia Interactive Services, IEEE* (2012)
11. Berg, T.L., Forsyth, D.A.: Animals on the web. In: *Proc. CVPR.* (2006)
12. Fergus, R., Fei-Fei, L., Perona, P., Zisserman, A.: Learning object categories from Google’s image search. In: *Proc. ICCV.* (2005)
13. Li, J., Wang, G., Fei-Fei, L.: OPTIMOL: automatic Object Picture collecTion via Incremental MOdel Learning. In: *Proc. CVPR.* (2007)
14. Lin, W.H., Jin, R., Hauptmann, A.: Web Image Retrieval Re-Ranking with Relevance Model. In: *Proc. ICWI.* (2003)
15. Schroff, F., Criminisi, A., Zisserman, A.: Harvesting Image Databases from the Web. *IEEE PAMI* **33** (2011) 754–766
16. Jégou, H., Douze, M., Schmid, C.: Improving bag-of-features for large scale image search. *IJCV* **87** (2010) 316–336
17. Chatfield, K., Lempitsky, V., Vedaldi, A., Zisserman, A.: The devil is in the details: an evaluation of recent feature encoding methods. In: *Proc. BMVC.* (2011)
18. Jaakkola, T., Haussler, D.: Exploiting generative models in discriminative classifiers. In: *NIPS, MIT Press* (1998) 487–493
19. Perronnin, F., Dance, C.: Fisher kernels on visual vocabularies for image categorization. In: *Proc. CVPR.* (2007)
20. Perronnin, F., Sánchez, J., Mensink, T.: Improving the fisher kernel for large-scale image classification. In: *Proc. ECCV.* (2010)
21. Lowe, D.: Object recognition from local scale-invariant features. In: *Proc. ICCV.* (1999) 1150–1157
22. Bosch, A., Zisserman, A., Muñoz, X.: Scene classification using a hybrid generative/discriminative approach. *IEEE PAMI* **30** (2008)
23. Bergamo, A., Torresani, L., Fitzgibbon, A.: PiCoDes: Learning a compact code for novel-category recognition. In: *NIPS.* (2011) 2088–2096
24. Torresani, L., Szummer, M., Fitzgibbon, A.: Efficient object category recognition using classemes. In: *Proc. ECCV.* (2010) 776–789
25. Grauman, K., Darrel, T.: The pyramid match kernel: Discriminative classification with sets of image features. In: *Proc. ICCV.* (2005)
26. Lazebnik, S., Schmid, C., Ponce, J.: Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In: *Proc. CVPR.* (2006)
27. Jégou, H., Perronnin, F., Douze, M., Sánchez, J., Pérez, P., Schmid, C.: Aggregating local image descriptors into compact codes. *IEEE PAMI* (2011)
28. Jégou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: *Proc. ECCV.* (2008)
29. Rastegari, M., Fang, C., Torresani, L.: Scalable object-class retrieval with approximate and top-k ranking. In: *Proc. ICCV.* (2011)
30. Goto, K., Kidono, K., Kimura, Y., Naito, T.: Pedestrian detection and direction estimation by cascade detector with multi-classifiers utilizing feature interaction descriptor. In: *Proc. IEEE Symposium on Intelligent Vehicles.* (2011) 224–229
31. Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes (VOC) challenge. *IJCV* **88** (2010) 303–338
32. Vedaldi, A., Fulkerson, B.: VLFeat: An open and portable library of computer vision algorithms (2008) <http://www.vlfeat.org/>.
33. Vedaldi, A., Zisserman, A.: Efficient additive kernels via explicit feature maps. *IEEE PAMI* (2011)

34. Bergamo, A., Fang, C., Torresani, L.: VLG extractor software (2011)
35. Singer, Y., Srebro, N.: Pegasos: Primal estimated sub-gradient solver for SVM. In: Proc. ICML. (2007) 807–814
36. Arandjelović, R., Zisserman, A.: Multiple queries for large scale specific object retrieval. In: Proc. BMVC. (2012)