On-the-fly Visual Category Search in Web-scale Image Collections

D.Phil Thesis

Ken Chatfield
University College

Supervisor:
Professor Andrew Zisserman

September 2014

Ken Chatfield
University College

Doctor of Philosophy
Trinity Term 2014

On-the-fly Visual Category Search in Web-scale Image Collections

Abstract

This thesis tackles the problem of large-scale visual search for categories within large collections of images. Given a textual description of a visual category, such as 'car' or 'person', the objective is to retrieve images containing that category from the corpus quickly and accurately, and without the need for auxiliary meta-data or, crucially and in contrast to previous approaches, expensive pre-training.

The general approach to identifying different visual categories within a dataset is to train classifiers over features extracted from a set of training images. The performance of such classifiers relies heavily on sufficiently discriminative image representations, and many methods have been proposed which involve the aggregating of local appearance features into rich bag-of-words encodings. We begin by conducting a comprehensive evaluation of the latest such encodings, identifying best-of-breed practices for training powerful visual models using these representations. We also contrast these methods with the latest breed of Convolutional Network (ConvNet) based features, thus developing a state-of-the-art architecture for large-scale image classification.

Following this, we explore how a standard classification pipeline can be adapted for use in a real-time setting. One of the major issues, particularly with bag-of-words based methods, is the high dimensionality of the encodings, which causes ranking over large datasets to be prohibitively expensive. We therefore assess different methods for compressing such features, and further propose a novel cascade approach to ranking which both reduces ranking time and improves retrieval performance.

Finally, we explore the problem of training visual models on-the-fly, making use of visual data dynamically collected from the web to train classifiers on demand. On this basis, we develop a novel GPU architecture for on-the-fly visual category search which is capable of retrieving previously unknown categories over unannonated datasets of millions of images in just a few seconds.

This thesis is submitted to the Department of Engineering Science, University of Oxford, in fulfilment of the requirements for the degree of Doctor of Philosophy. This thesis is entirely my own work, and except where otherwise stated, describes my own research.

Ken Chatfield, University College

Contents

Chapter 1

Introduction

1.1 Objective

The objective of this thesis is to enable the fast, scalable and accurate retrieval of images purely on the basis of their *visual content*. This is in contrast to many existing approaches to image search which use a combination of a limited number of visual cues with text annotations and other metadata (*e.g.* Google Images, Bing, Flickr). Our aim is to work towards pure visual search systems that nonetheless operate with the same accuracy and, critically, speed as such existing text and multi-modal web 'image' search engines but with far more flexibility given that only visual data is used. The user should be able to progress from cold query to results in a matter of seconds. In other words, our objective is nothing less than to move towards visual search systems that can operate in a fully 'on-the-fly' manner.

Although there has been much work in recent years on scaling up search for visual *instances* (*e.g.* find all images of Westminster Cathedral), there has been less attention

given to the more general problem of searching for broader visual *categories* (*e.g.* find images of *all* cathedrals) particularly in the 'on-the-fly' setting and without involving some form of pre-training, and this is the particular problem we address here.

By making efficient pure visual search possible across unannotated datasets of millions of images, facilitating freeform queries in the same manner as existing text-based search engines, we hope that this work can contribute to making visual search as ubiquitous as the text-based search engines which we rely upon daily. Furthermore, we envisage that the ability to search for such broader visual categories in addition to instances, and in a similarly natural and unconstrained manner as Google and Bing, will form an important step in making visual search more widely applicable. In this thesis, we present a fully-operational prototype of a system based upon this idea.

1.2 Motivation

The web has come a long way since its early days, when information was predominantly textual and images and multimedia was seen as something new and exciting. In today's world, multimedia, and more specifically visual data, is increasingly the medium in which we choose to chronicle our lives and communicate with each other. At the time of writing, each day over 350M images are uploaded to the social network Facebook per day[1], 60M to Instagram[2], and nearly 16 years of video per day to the video sharing website YouTube[3]. With the increasing adoption of camera-equipped smartphones such as the iPhone this trend is only set to grow, with seemingly every couple of months a

[1] `http://mashable.com/2013/09/16/facebook-photo-uploads/`
[2] `http://instagram.com/press/`
[3] `https://www.youtube.com/yt/press/statistics.html`

new image or video sharing app being released and becoming popular, from Vine to Snapchat. In the business space too, TV and film production companies have begun to amass an increasingly large quantity of visual information in digital archives and repositories around the world.

Despite this, our ability to search and organise this increasing mountain of thousands of petabytes of visual data has not kept up. Most existing visual search systems operate solely on the basis of manually added textual annotations, which are often inadequate or sometimes missing completely. For the most part, the huge quantity of visual data stored in data centres around the world remains locked-off and incomprehensible to most computers. If the visual content of images and videos could be indexed and searched directly in an effective manner, the implications clearly would be significant.

Much progress has already been made in the indexing of large visual datasets for on-demand fast visual search within certain limited scenarios, for example with efficient retrieval of specific people and places making rapid progress over the past decade and progressing to the point of commercialisation in systems such as Google Goggles and Kooaba. However, the ability to make similar on-demand queries for more general and abstract concepts and categories over datasets of millions of images, without falling back on a limited set of pre-prepared queries and all within a reasonable timeframe, has remained elusive.

1.2.1 Applications

Throughout this work, our objective is to build up the tools to build an efficient and scalable system for large-scale visual category search. Here a 'category' can mean a collection of related objects (*e.g.* 'cathedral', 'train', 'dog') but also a collection of related scenes (*e.g.* 'forest', 'sunset', 'gothic architecture'). We envisage many applications for such a system, several of which are described below:

1. **Web attribute-based visual category search.** With the increasing volume of photos and videos available on the web, much of it with rudimentary annotation at best, there is a growing need for systems that can understand visual data without the aid of accompanying metadata. Existing visual search systems, from the web-wide Google Image search to the site-level search tools on sites such as YouTube and Flickr, rely heavily on textual metadata and link popularity, and particularly in the former case this has made the precision of such systems typically very high. Nonetheless, with the many millions of images now available on the web, such systems fail to provide an effective means to access the sheer diversity of the data which has been uploaded, and there is a long tail which is effectively rendered inaccessible by the computer's inability to really 'understand' the images it is searching.

   Scalable visual category search would go some way to making more of this information accessible, along with the tighter integration of visual cues. We envisage an attribute-based system whereby an appearance-based query can be iteratively refined (*e.g.* `car + red + coupe`) to allow visual queries to be specified with

increasing precision.

2. **Digital media archive search.** While television production companies are archiving more and more of their library footage in digital format, often lack of meta-data makes such footage inaccessible. When preparing a report or article, production teams often have a clear idea of the kind of material they want (*e.g.* when preparing a report on the autumn, they may envisage a road running through a glade of trees decked in autumnal leaves) but are often unable to then search using such high-level queries in the archive, ultimately leading them to rely on expensive professionally curated and annotated stock image and video collections. Scalable visual category search would allow archive footage to be used instead. We are collaborating with the BBC to provide visual category search functionality across their archive in just this way, further details of which can be found in the later chapters of this thesis.

3. **Automated organisation and search of personal photo collections.** Despite the huge increase in volume of photos in most people's personal photography collection which has resulted from the rise of the ever-present camera-equipped smartphone and driven by the popularity of such services as Flickr and Instagram, organising that growing amount of visual data, which for the most part lacks any meaningful meta-data short of location and time, is a challenge that remains very much unsolved. Scalable visual category search would allow the user to quickly explore their collection by visual keyword, finding all of their photos taken at the beach, with their cat, on a train, at a dinner party, or in the city.

In addition, there are numerous other more specific applications which would benefit from the techniques explored in this work, from facilitating search for creative work of a certain visual style on websites such as Adobe Behance[4] to the on-the-fly identification of objects in augmented reality applications, which are sure to become more popular with the growth of wearables such as Google Glass, all of which require real-time or close performance.

1.3 Challenges

Accurate and scalable large-scale visual search presents many challenges:

Robustness to changes in appearance of a fixed object. Even in the simplest case, where we wish to retrieve all images of a fixed object instance, its appearance may vary significantly between two images due to changing viewpoint and lighting, or it may be partially occluded by other objects or clipped by the edge of the image frame.

Robustness to changes within a visual category. In our case, we wish to address the more complex problem of visual category retrieval, which introduces problems of its own. Namely, the visual appearance of an object or scene belonging to a particular category may vary greatly. For example, cars come in many different shapes, sizes and colours, and a Ferrari looks quite different visually when compared to, for example, a Land Rover (Figure 1.1a). This is the problem of *intra-class variation*. On the other hand, two semantically very different categories may still be very similar in terms of

[4] `https://www.behance.net`

(a) Intra-class variation (b) Inter-class variation

Figure 1.1: Challenges of category retrieval

appearance, for example a goods train and a lorry (Figure 1.1b). This is the problem of *inter-class variation*. As a result, moving from instances to categories is a doubly challenging task, as it requires an additional level of *generalisation*.

Speed and Scalability. Our aim is to work towards a retrieval framework which is not only fast, but can operate at close to real-time speed ('on-the-fly') such that it is possible to retrieve visual categories from datasets of millions of images with the same ease and speed as issuing a query to Google search. This is an incredibly challenging task as it requires (i) efficient retrieval algorithms, which should scale at worst case linearly, preferably sub-linearly to allow application to an ever-expanding dataset size and (ii) a compact image representation, since all required data should fit inside the computer's RAM memory if we are to have a hope of meeting the near real-time operation requirement.

In addition, if our objective is to train visual category models on demand also, rather than relying on a fixed bank of pre-trained models, we must also collect the necessary training data and process it all within the time budget of a few seconds we have allocated ourselves. This requires a highly-tuned and efficient learning stage in addition to the fast retrieval stage described above.

1.4 Contributions

In this section, we list some of the key contributions made in this thesis.

1. Best-of-breed shallow encodings for image classification. Over the past five years, numerous novel image encodings have been proposed for the image classification task, many claiming to perform at state-of-the-art performance by their authors. However, given the many components of an image classification pipeline, the experimental setup used in each case was often so different as to make comparisons between different encodings challenging at best. By conducting a comprehensive evaluation of a large selection of recent shallow encoding methods for the image classification task using a fixed experimental framework and describing all experimental details, much of the confusion surrounding state-of-the-art shallow image encodings has been dissipated, and our software evaluation framework has been widely adopted within the community. Our work on shallow feature encodings is presented in Chapter 3.

2. State-of-the-art deep classification pipeline. With the recent re-emergence of Convolutional Networks (ConvNets) in the community for large-scale classification, much work has been undertaken recently to make ConvNet-based approaches more widely applicable by employing them as extractors of general purpose 'deep' features. However, details such as how the design of the network architecture, along with data augmentation and other learning heuristics, effects the performance of such features have not been extensively explored. By undertaking such a methodological evaluation of deep features we develop a pipeline which achieves current state-of-the-art

performance on the image classification task with a very simple architecture. This is presented in Chapter 4.

3. Live learning of visual models from the web. Although there have been efforts in the past to harness visual data collected from the web for training visual models, for the most part this has been in an offline manner, crawling websites for data to be post-processed using expensive machine learning algorithms. We demonstrate how it is possible to harness images downloaded from the latest versions of web image search engines such as Google Image search and Bing to train highly discriminative visual models on-the-fly using simple linear classifiers in combination with the latest state-of-the-art image representations. This idea is explored in Chapters 5 and 6.

4. Cascade approach for fast ranking. One of the major constraints which has presented the latest rich shallow image encodings being used at scale for online retrieval over large unannotated datasets has been their relatively high dimensionality, resulting in inordinately slow processing times when many such encodings must be processed simultaneously as is the case during ranking. We propose a novel cascade architecture for processing such features which both dramatically reduces the time required to rank large datasets whilst simultaneously *increasing* retrieval performance. The details are given in Chapter 5.

5. GPU architecture for on-the-fly visual search. Based on the idea of dynamically learning visual models from the web and harnessing the inherent amenability of ConvNet-based deep visual features for parallelisation using GPUs, we develop a

novel GPU-baed architecture which is capable of simultaneously learning new concepts on-the-fly whilst also using these models in a live manner to allow the close-to instantaneous return of ranked lists from the model which can then be incrementally refined as new training data is obtained. This architecture provides state-of-the-art retrieval performance, through its reliance on deep visual features, whilst providing live results over datasets of millions of images for previously unknown queries within seconds. Furthermore, it can be run entirely on commodity GPU hardware, allowing it to potentially be run entirely on a consumer-grade laptop for large-scale visual search of personal photography collections, for example. This system is presented in Chapter 6.

1.5 Publications

The work on evaluating shallow feature encodings in Chapter 3 was presented at BMVC 2011 [26], and was recipient of a best poster prize. As of spring 2015, the paper has been cited over 350 times. The deep feature evaluation in Chapter 4 was published at BMVC 2014 [27], and received the best academic paper prize. An early version of the on-the-fly retrieval work in Chapter 5 was published at ACCV 2012 [29] and also at the TRECVid workshops [3, 4]. The work on on-the-fly retrieval on GPU in Chapter 6 was presented at ACCV 2014 [28]. Finally, work from both Chapters 5 and 6 will be published in the journal IJMIR [25].

Chapter 2

Literature Review

In this chapter, we review some of the related work relevant to the development of an on-the-fly retrieval system. We start by reviewing early multimedia retrieval approaches to visual search in Section 2.1, before exploring the rapid progress made in the vision community in tackling the more restricted problem of *particular instance* search triggered by the introduction of powerful local appearance-based features in the late 90s in Section 2.2. Following this, we discuss how the same local features were adapted to tackle the more general problem of object *category* classification over large datasets in Section 2.3. Finally, we cover how attempts have been made to adapt such methods to large-scale object category retrieval in Section 2.4.

2.1 The Early Days: Content Based Image Retrieval

The origins of the first real attempts to deal with visual search lie in the early 90s. At that time, the computer vision community was still very much engaged with addressing

fundamental problems of object-level image understanding such as the correspondence problem and instance matching, and many of the algorithms being proposed were still far too computationally intensive to apply to datasets of any size. It was therefore left to researchers in the nascent discipline of multimedia retrieval to undertake the first pioneering work in visual search by focusing on how to capture the information contents of an image as efficiently as possible, consequently resulting in some very simple but nonetheless surprisingly effective image representations in the early days.

It was in 1993, the year of the introduction of the first graphical web browser Mosaic with all the implications that would have for the explosion of visual data to come, that heralded the release of the first commercial visual system in IBM QBIC [89], marking the birth of what would come to be known as the field of Content Based Image Retrieval (CBIR). Early approaches predominantly adopted a *query-by-example* metaphor, with images retrieved according to similarity to a template image, which served as the query. That similarity was measured on the basis of a number of global image statistics, such as colour histograms, geometric shape descriptors and texture descriptors.

What followed was a decade of research developing these global image representations, with better illumination-invariant colour spaces formulated [44, 47, 48, 119, 122] along with more descriptive shape and texture representations [83, 134] and a host of new web-based interfaces to facilitate the use of this new set of tools [31, 48, 96, 121]. The simplicity of the features generally used meant that the systems could generally operate at reasonably large-scale – for example, the Cortina system of Quack *et al.* [104] demonstrates real-time retrieval from a dataset of 10M+ images. Nonetheless, by the 2000s the limitations of visual search using such global image statistics was becoming

increasingly apparent. This was as a result of both: (i) the simplicity of the representation, which remained highly sensitive to changes in viewpoint, occlusion and layout within the image, and (ii) the inability of the system to generalise and understand the user intent when working from such low-level cues extracted from just a single image.

Some early attempts at resolving the second of these two problems resorted to strategies such as the crude segmentation of the query image [21]. An alternative approach that was explored was to shift from query-by-example to a learning-based approach, translating low-level features into higher-level entities. In the language of CBIR, these entities are called *concepts* and the process of generalising is described in terms of addressing the 'semantic gap' which exists between the low-level visual representation of an image and the higher-level abstractions provided by natural language (these 'concepts' are of a similar level of abstraction as the 'object categories' described in the vision literature). This is facilitated through the application of machine learning algorithms, often to attach text annotations to dataset images [10, 19] which can then be used as the basis of visual search. This has been the origin of the focus among CBIR researchers over the past decade on 'semantic retrieval', the broad motivations for which are described succinctly in [103], and has resulted in a return to a pragmatic, multi-modal approach to visual search in recent years. For a comprehensive overview of recent approaches to CBIR, refer to [33, 120].

Simultaneously, since the start of the century within the vision community there has been a rapid development of more powerful appearance-based visual features capable of addressing some of the short-comings of the early global representations used for CBIR, many of which have been subsequently adapted for use in a CBIR setting, as

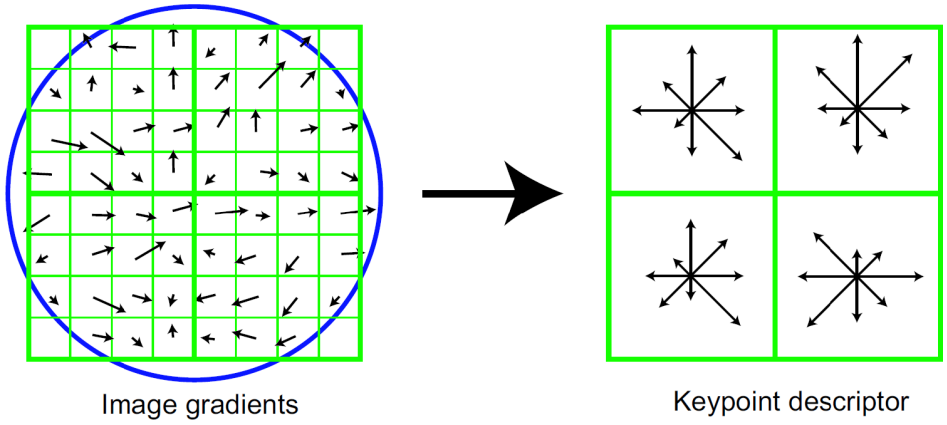


Figure 2.1: Illustration of the SIFT descriptor of Lowe [82]. Image gradients within a patch (left) are accumulated into a coarse 4×4 spatial grid (right, only a 2×2 grid is shown). A histogram of gradient orientations is formed in each grid cell. 8 orientation bins are used in each grid cell giving a descriptor of dimension $128 = (4 \times 4 \times 8)$.

well as for 'pure' visual search for categories as we will see. We thus now turn our attention to the problem of visual search from a vision perspective.

2.2 Going local: Large-scale Instance Retrieval

From around the end of the 90s, there was a revolution brewing in image representations in the Computer Vision community brought on by the introduction of local appearance-based features. In contrast to the global features common previously (*e.g.* colour histograms or texture features) which summarise the entirety of an image in the form of a single feature vector, the fundamental idea was to describe an image in terms of its local visual appearance extracted from a collection of smaller extracted sub-patches. This is based on the assumption that in the presence of occlusion, background clutter and changes in scale and camera viewpoint two images containing the same object may be very different on a macro-level, but share many similarities in appearance on a micro-level.

Given that raw pixel colour or intensity is clearly not robust to changes in image conditions, such as illumination and differences in scale and rotation, these patches must be processed in some way to produce a representation which is as far as possible invariant to such changes and yet still discriminative enough to distinguish between different patches. There are a variety of different hand-engineered local descriptors which were introduced at around this time that addressed this challenge [87,127] However, arguably the most successful has been the SIFT descriptor of Lowe [82]. This is computed for a local image patch by binning image gradients over a grid placed onto the patch (see Figure 2.1). Since Lowe's work, there have been many similar local features proposed based on the binning of local image gradients [12,20,32,86,123,142].

The locations at which local patches are extracted from an image can be determined in two ways – (1) using an interest point detector such as the difference of Gaussian method proposed by the SIFT paper [82] or other similar measures such as Harris corners [56] or the Laplacian [79] or (2) densely over a regularly spaced grid at multiple scales. Initially designed to address the correspondence problem, the selection of interest points was at the time the favoured approach in order to keep computation when working directly with the features tractable, and the explicit selection of 'interesting' features also proved sufficient and even desirable when they were soon adapted for the matching of instances across larger datasets (but we will see later how dense sampling is often the preferred strategy when such features are used for the more general problem of object *category* retrieval).

Given sets of local features extracted from all images in a collection and a query image, instance retrieval (*i.e.* the retrieval of a *particular* object *e.g.* Notre Dame

Cathedral in Paris) reduces to the efficient matching of the query feature set to the collection feature sets. Although typically hundreds of local features are extracted per image, and matching across them might at first glance seem to be a problem not amenable to scaling, a breakthrough was made in the seminal 'Video Google' work of Sivic and Zisserman [118] which adapted techniques from the field of text-retrieval to tackle this task in an efficient way. We describe the details of this approach in the next section.

From a visual search perspective, the 'Video Google' work was one of the most successful early attempts at large-scale image retrieval by *object*, and marked the beginning of a steady march of the computer vision community towards tackling the larger datasets which had been common in multimedia retrieval for some time. It was soon shown to scale to datasets of up to 1M+ images [100].

2.2.1 Representing an Image using Visual Words

The key insight of Sivic and Zisserman [118] was to draw a parallel between the unordered collection of discriminative local visual features extracted from an image and words in a text document. Similar to the image case, the important information in the case of a text document is local (the individual words) and there is a need to match across collections of words quickly and efficiently. An established technique for doing this in the text retrieval community is to represent documents as histograms of word occurrences, otherwise known as 'bag-of-words', which then allows quick matching on basis of histogram similarity.

In order to adapt this technique for instance retrieval, we require a set of canon-

ical 'visual words' to define our histogram bins. This is typically obtained by extracting local features from a set of training images which are then clustered in feature space to obtain a smaller set of K cluster centres. A typical approach is to use the k-means algorithm for this clustering which, given a set $\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$ of N training descriptors, seeks K vectors $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K \in \mathbb{R}^D$ and data-to-means assignments $\psi_1, \ldots, \psi_N \in \{1, \ldots, K\}$ such that the cumulative approximation error $\sum_{i=1}^{N} \|\mathbf{x}_i - \boldsymbol{\mu}_{\psi_i}\|^2$ is minimised.

Given the visual vocabulary, we can then encode our local feature by assigning it to its closest 'visual word' in the vocabulary, at which point it can be considered 'matched':

$$\phi(\mathbf{x}_i) = [q_1, ..., q_K]^\top, \quad q_k = \begin{cases} 1 & \text{if} \quad \mathrm{NN}(\mathbf{x}_i) = \boldsymbol{\mu}_k \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

Following this, the image representation is obtained by aggregating all the individual feature encodings into a single histogram:

$$\boldsymbol{\phi}_{\text{hist}} = \sum_{i=1}^{N} \phi(\mathbf{x}_i) \tag{2.2}$$

The output is a sparse *bag of visual words* vector of fixed cardinality, which can be efficiently ranked using the same techniques as developed over many years in the text-retrieval community for this task.

It should be noted that through the assignment of local features to canonical 'visual words' some information loss is incurred through quantisation error, and we explore

alternative encodings which mitigate against this in Section 2.3.2. Nonetheless, by using this simple strategy we are able to describe an image as a single vector again, avoiding both the computational and storage costs of having to process sets of local features whilst still capturing the overall statistics of image patterns on a local level.

2.3 Object Category Classification over Large Datasets

Parallel to the scaling up of instance search algorithms, there was also a move to address the broader problem of classifying object *categories* in images. For example, as shown in Figure 2.2, given a collection of images an example of the task might be to determine which images contain not just the Notre Dame cathedral in Paris, but *any* cathedral – the desired output of the system is a yes/no binary label indicating whether or not the object category 'cathedral' is present or not in each image.

In addition to being robust against changes in viewpoint, as is required for the simpler problem of instance retrieval, it is also necessary to be able to generalise each category to account for both intra-class and inter-class variation. The problem is essentially one of pattern classification and in the simplest case consists of identifying recurring localised and discriminative patterns which correlate with the existence or absence of particular object categories. For example, images containing cathedrals might be more likely to contain angular elements relating to the tops of the flying buttresses in gothic architecture.

Progress for the object category classification task been driven by a range of classification challenges of increasing difficulty and scale in the community in recent years,

from the PASCAL VOC challenge [37] (20 categories, 10,000 images) to, in more recent years, the ILSVRC challenge [13] (1,000 categories, 1.2M images) which is a curated subset of the full ImageNET dataset [34] (∼22,000 categories, 14M images). We explore the typical approach taken for this task in Section 2.3.1 which follows, which we will see often builds upon the method used for instance retrieval described in Section 2.2. Given the challenge of generalising over a wide variety of classes, particularly at scale, the focus of such challenges has predominantly been on achieving good classification performance on an increasing set of predetermined classes. The subsidiary problem of doing this at speed over novel classes, such as would be required for a real-time visual search system, has been consequently the subject of less attention, although we will see later in Section 2.4 how standard large-scale classification methods can be adapted to this task.

2.3.1 A Representative Pipeline

A representative architecture for object category classification is shown in Figure 2.2. The top branch is almost identical to the pipeline described for instance retrieval in Section 2.2, with locally extracted SIFT features providing the output of the feature extractor block and the 'bag-of-words' encoder described in Section 2.2.1 serving the role of the image encoder and pooler blocks. The main differences compared to the instance retrieval pipeline are that: (i) for the feature extractor stage, typically features are extracted densely and at multiple scales from the source images (rather than simply being extracted at interest points), and (ii) a learning stage has been added at the end of the pipeline, typically consisting of a discriminative classifier such as a support

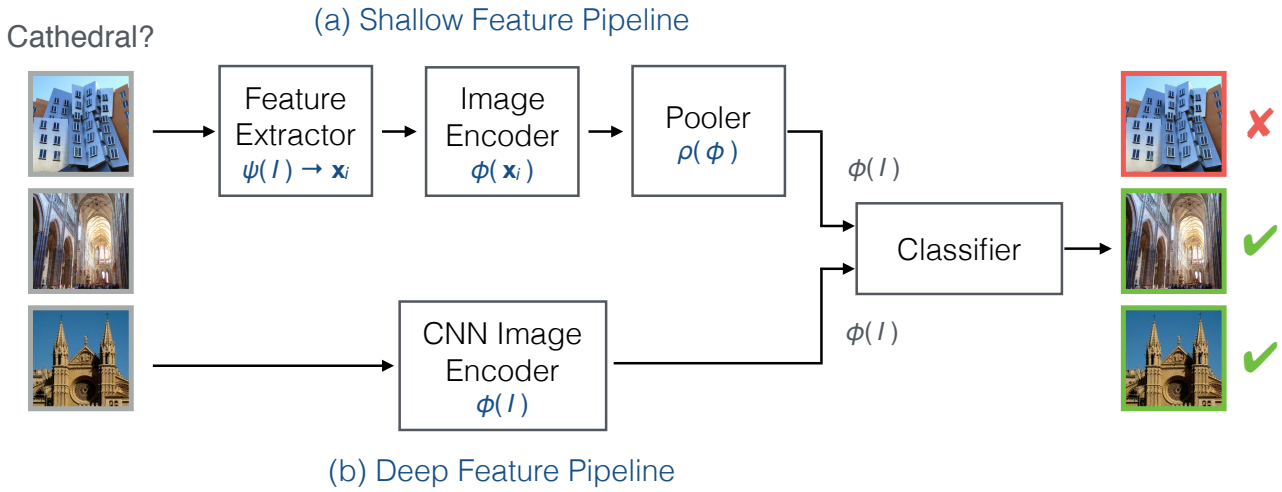


Figure 2.2: Object category classification pipeline

vector machine (SVM) (although other approaches are also possible [107]) which is used to train an object category model from multiple training images (rather than the query being specified by a single image and matched directly). This pipeline of hand-engineered feature extraction and encoding, followed by pooling into a compact vector representation and finally a learnt classifier stage has become over the past decade the established standard approach for the object category classification task. We explore this setup further in Section 2.3.2 below.

However, a new breed of methods has recently emerged based on Convolutional Neural Networks (CNNs). These methods can in fact replace the entirety of the pipeline, accepting as input the raw pixels of the test images and outputting a classification score, or alternatively can be used as a means of automatically learning low-level features which can be passed into our existing classification stage, as shown in the bottom branch of Figure 2.2. We explore this approach further in Section 2.3.3.

2.3.2 Shallow Representations

In this section we describe 'shallow' image representations for object category recognition, named as such simply to distinguish them from the 'deep' CNN based methods described in Section 2.3.3.

In order to generalise beyond a given set of training images and identify relevant class-specific characteristics whilst remaining robust to intra-class variations, early research established that when using simple 'bag-of-words' encoded local features (Section 2.2.1), the use of non-linear classifiers was highly beneficial to performance [46]. However, the use of kernel classifiers to provide this non-linearity rendered it very difficult to scale such systems to larger training sets. As a result, significant research effort was expended on improving the local feature encoding step to provide a richer encoding which incorporated a higher degree of non-linearity 'out of the box', and which could thus be used directly with much faster linear classifiers without modification.

More formally, the plain-vanilla 'bag-of-words' encoding, described in Section 2.2.1, can be seen as just one variation of a broader family of local feature encodings which perform non-linear aggregation in different ways:

$$\boldsymbol{\phi} = \text{pool}\left(\left\{\phi(\mathbf{x}_i)\right\}_{i=1}^{N}\right) \tag{2.3}$$

Where $\phi(\mathbf{x}_i)$ is the encoding of a local feature $\mathbf{x}_i$, N is the total number of local features, and pool is the function used to aggregate the features into a single vector representation, which is typically one of either sum (average) pooling: $\boldsymbol{\phi} = \frac{1}{N}\sum_{i=1}^{N}\phi(\mathbf{x}_i)$ or element-wise max pooling: $\boldsymbol{\phi} = \max_{i=1}^{N}\phi(\mathbf{x}_i)$.

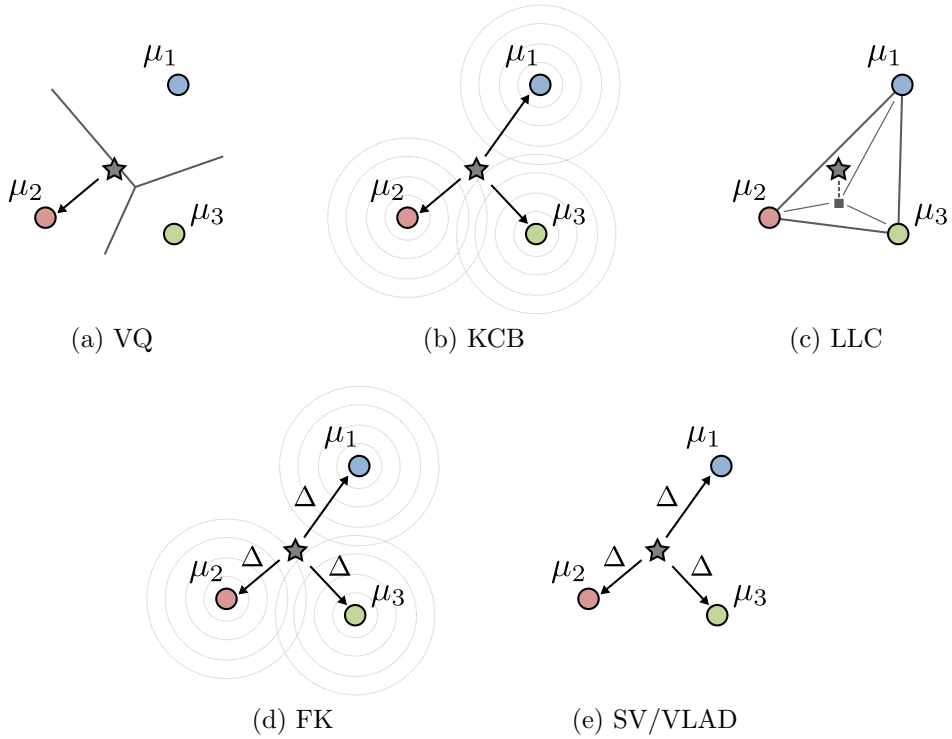
Regular 'bag-of-words', which we will refer to **histogram encoding** henceforth, is generally used with sum pooling, making it equivalent to a histogram of visual word frequencies within an image. Alternative methods replace the hard quantisation of features involved in this method with alternative encoding functions that retain more information about the original image features, thus mitigating the quantisation error incurred by the histogram encoding method. This has been done in two ways: (1) by expressing features as *combinations* of visual words and (2) by recording the *difference* between the features and the visual words.

We describe below two methods that use approach (1), namely *kernel codebook encoding* and *local linear coding*, followed by three methods which use approach (2), namely *Fisher vector encoding*, *super-vector encoding* and *vector of locally aggregated descriptors (VLAD)*. Refer to Figure 2.3 for a visualisation of the differences between these different encoding schemes.

Kernel codebook encoding [102, 129] is a variant of plain-vanilla 'bag-of-words' in which descriptors are assigned to visual words in a soft manner. More specifically, each descriptor is encoded as:

$$[\phi_{\text{kcb}}(\mathbf{x}_i)]_k = K(\mathbf{x}_i, \boldsymbol{\mu}_k) / \sum_{j=1}^{K} K(\mathbf{x}_i, \boldsymbol{\mu}_j) \tag{2.4}$$

Where $K(\mathbf{x}, \boldsymbol{\mu}) = \exp(-\frac{\gamma}{2}\|\mathbf{x} - \boldsymbol{\mu}\|^2)$ and γ is a free parameter determined using a validation set. In general, sum pooling is used with this method, to give a final encoding of size K-dimensions, identical to the standard hard-assigned *histogram encoding* method: $\boldsymbol{\phi} = \frac{1}{N}\sum_{i=1}^{N} \phi_{\text{kcb}}(\mathbf{x}_i)$.



(a) VQ (b) KCB (c) LLC

(d) FK (e) SV/VLAD

Figure 2.3: Different encoding methods. Histogram encoding (VQ) – assigns a novel feature (shown as a star) to the nearest visual word in feature space. Kernel codebook encoding (KCB) – features are mapped to multiple visual words in a soft manner in proportion to the Gaussian weighted distances to each. Locality constrained linear encoding (LLC) – features are mapped to multiple visual words in order to minimise the reconstruction error in feature space. Fisher encoding (FK) – captures the average first and second order *differences* between a feature and the centres of a GMM. Supervector encoding (SV) – captures the average zeroth and first order *differences* between a feature and a k-means trained vocabulary. Vector of locally aggregated descriptors (VLAD) – captures just the first order *differences* between a feature and a k-means trained vocabulary.

Locality-constrained linear (LLC) encoding [133] projects each image descriptor $\mathbf{x}_i$ down to the local linear subspace spanned by the $M \ll K$ visual words closest to $\mathbf{x}_i$. We once again start with a visual vocabulary learnt using k-means: $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$. The first step is to identify the indices of the M visual words $\boldsymbol{\mu}_k$ closest to $\mathbf{x}_i$ (measured by Euclidean distance). We denote the visual words themselves collectively as $\mathbf{B} = [\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K]$ and the subset of this matrix which contains the closest visual words to $\mathbf{x}_i$ as $\mathbf{B}_i = [\boldsymbol{\mu}_{\sigma_1}, \ldots, \boldsymbol{\mu}_{\sigma_M}]$.

To project $\mathbf{x}_i$ down to the span of the M closest visual words $\mathbf{B}_i$, we will use the approximation $\mathbf{x}_i \approx \mathbf{B}_i\boldsymbol{\alpha}$ where $\boldsymbol{\alpha}$ is obtained by solving the problem:

$$\boldsymbol{\alpha}^* = \underset{\mathbf{1}^\top\boldsymbol{\alpha}=1}{\operatorname{argmin}} \|\mathbf{x}_i - \mathbf{B}_i\boldsymbol{\alpha}\|^2 + \beta\|\boldsymbol{\alpha}\|^2 \tag{2.5}$$

Which is equivalent to solving the regularised least squares problem using the M closest visual words with the regularisation parameter β obtained by cross-validation. The norm of $\boldsymbol{\alpha}^*$ is fixed by the constraint $\mathbf{1}^\top\boldsymbol{\alpha} = 1$. The LLC encoding of the descriptor $\mathbf{x}_i$ is the K-dimensional vector $\phi_{\text{LLC}}(\mathbf{x}_i)$ of all zeroes except for the M components $[\phi_{\text{LLC}}(\mathbf{x}_i)]_{\sigma_m} = \boldsymbol{\alpha}_m$, $m = 1, \ldots, M$. In general, max-pooling is used with this method: $\boldsymbol{\phi} = \max_{i=1}^{N} \phi_{\text{LLC}}(\mathbf{x}_i)$. As with both histogram encoding and kernel codebook encoding, this gives an encoding of size K.

Fisher encoding [99] makes the assumption that local features in an image are independent samples from a Gaussian Mixture Model (GMM) and is the first method we describe that records the *difference* between features and visual words in addition to just the raw assignments. We start by training our GMM, which acts as our visual

vocabulary:

$$p(\mathbf{x}|\boldsymbol{\theta}) = \sum_{k=1}^{K} p(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\pi_k, \quad p(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \frac{1}{\sqrt{(2\pi)^D \det \boldsymbol{\Sigma}_k}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k)}, \tag{2.6}$$

Where $\boldsymbol{\theta} = (\pi_1, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1, \ldots, \pi_K, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K)$ is the vector of parameters of the model, including the prior probability values $\pi_k \in \mathbb{R}_+$ (which sum to one), the means $\boldsymbol{\mu}_k \in \mathbb{R}^D$, and the positive definite covariance matrices $\boldsymbol{\Sigma}_k \in \mathbb{R}^{D\times D}$ of each Gaussian component. Here the covariance matrices are assumed to be *diagonal*, so that the GMM is fully specified by $(2D+1)K$ scalar parameters. The parameters can be learnt by expectation maximisation (EM, [85]) from a training set of descriptors $\mathbf{x}_1, \ldots, \mathbf{x}_N$. The GMM defines then the soft data-to-cluster assignments for a feature $\mathbf{x}_i$ to be encoded:

$$q_{ki} = \frac{p(\mathbf{x}_i|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)\pi_k}{\sum_{j=1}^{K} p(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\pi_j}, \quad k = 1, \ldots, K. \tag{2.7}$$

To compute the Fisher vector, we then compute the gradient of the log-likelihood of the features with respect to the GMM parameters, namely the mixture weights π_k, means $\boldsymbol{\mu}_k$, and diagonal covariances $\boldsymbol{\Sigma}_k$. In fact, Perronin et al. [99] suggest that the derivatives with respect to the mixture weights do not add much information and can be discarded, so we are left with the derivatives with respect to the means and covariances:

$$\mathbf{u}_k = \frac{1}{N\sqrt{\pi_k}} \sum_{i=1}^{N} q_{ki} \boldsymbol{\Sigma}_k^{-\frac{1}{2}} (\mathbf{x}_i - \boldsymbol{\mu}_k) \tag{2.8}$$

$$\mathbf{v}_k = \frac{1}{N\sqrt{2\pi_k}} \sum_{i=1}^{N} q_{ki} \left[(\mathbf{x}_i - \boldsymbol{\mu}_k) \boldsymbol{\Sigma}_k^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_k) - 1 \right]. \tag{2.9}$$

Computing the gradient with respect to each Gaussian mean (2.8) corresponds to the first moment of the image feature distribution, which is equivalent to the aggregation of the first order differences (measured by distance) of all features and all GMM centres, weighted by the posterior probability of each feature having been generated by a particular GMM centre. The second moment is captured in an analogous way.

Note that, since the covariance matrices $\boldsymbol{\Sigma}_k$ are assumed to be diagonal, computing these quantities is quite fast. The Fisher encoding of the set of local features is then given by the concatenation of $\mathbf{u}_k$ and $\mathbf{v}_k$ for all K components, giving an encoding of size $2DK$:

$$\boldsymbol{\phi}_{\mathrm{FV}} = [\mathbf{u}_1^\top, \mathbf{v}_1^\top, \ldots, \mathbf{u}_K^\top, \mathbf{v}_K^\top]^\top. \tag{2.10}$$

The final stacked feature vector can be considered to describe how the distribution of features of a particular image differs from that fitted to the features of all training images, thus encoding an image in terms of how it relates to the entire visual vocabulary in feature space rather than simply operating on the basis of assignments. It should be noted that in order to make SIFT features amenable to modelling using a diagonal covariance matrix, it is necessary to ensure that they are decorrelated. This can be achieved using, for example, Principal Component Analysis (PCA).

Super vector encoding [143] is similar to the Fisher encoding. There are two variants of this encoding, based on hard assignment to the nearest codeword or soft assignment to several near neighbours. For the hard super vector encoding, let $q_{ki} = 1$ if $\mathbf{x}_i$ is assigned to cluster k by k-means and 0 otherwise. [143] does not specify how q_{ki} are set in the soft assignment case. We defined q_{ki} to be essentially the same as for the Fisher encoding, setting $\mathbf{\Sigma}_k = \sigma^2 \cdot \mathbf{I}$ with σ being twice the mean distance between points and means within the k-means algorithm and $\mathbf{I}$ the identity matrix. Define:

$$p_k = \frac{1}{N} \sum_{i=1}^{N} q_{ki}, \quad s_k = s\sqrt{p_k}, \quad \mathbf{u}_k = \frac{1}{\sqrt{p_k}} \sum_{i=1}^{N} q_{ki}(\mathbf{x}_i - \boldsymbol{\mu}_k), \tag{2.11}$$

Where s is a constant chosen to balance s_k with $\mathbf{u}_k$ numerically. Then the super vector encoding is given by:

$$\boldsymbol{\phi}_{\mathrm{SV}} = [s_1, \mathbf{u}_1^\top, \ldots, s_K^\top, \mathbf{u}_K^\top]^\top. \tag{2.12}$$

This gives an encoding of size $K(D+1)$. Compared to the Fisher encoding, the super vector encoding: (1) considers only the first order differences $\mathbf{u}_k$ and not the second order differences between features and cluster centres; (2) adds the components s_k which represent the mass of each cluster; (3) normalises each cluster by the square root of the *posterior* probability $\sqrt{p_k}$ rather than of the *prior* probability $\sqrt{\pi_k}$.

Vector of locally aggregated descriptors (VLAD) [67] is also closely related to the Fisher encoding, but has a simpler formulation, aggregating first order differences only on a hard-assignment basis. Given a visual vocabulary learnt using k-means:

$\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$, we compute for each of K visual words:

$$\mathbf{u}_k = \sum_{\mathbf{x}_i \,:\, \mathrm{NN}(\mathbf{x}_i)=\boldsymbol{\mu}_k} \mathbf{x}_i - \boldsymbol{\mu}_i \tag{2.13}$$

Where $\mathrm{NN}(\mathbf{x}_\mathrm{i})$ indicates the closest visual word to $\mathbf{x}_i$ in feature space. The VLAD encoding is then given by the concatenation of $\mathbf{u}_k$ for all K components, giving an encoding of size DK or half that of the Fisher encoding:

$$\boldsymbol{\phi}_{\mathrm{VLAD}} = [\mathbf{u}_1^\top, \ldots, \mathbf{u}_K^\top]^\top. \tag{2.14}$$

In other words, each visual word corresponds to a D-dimensional 'slot' in the encoded vector, and each feature $\mathbf{x}_i$ is encoded by adding its displacement to its assigned visual word into the corresponding k-th slot. It was shown in [67] that VLAD can be considered to be a simplified, non-probabilistic version of the Fisher encoding, and as such is closely related to the super vector encoding (aside from just encoding the first order differences and not the zeroth as in the super vector encoding).

A full evaluation of all of these encoding methods is conducted in Chapter 3, along with an exploration of how to optimally set the various parameters of each method, and the interaction with the final classification stage of the pipeline.

2.3.3 Deep Representations

In this section we discuss 'deep' image representations, where by 'deep' we mean architectures which involve multiple layers of non-linear computation. Such architectures,

known as Deep Neural Networks (DNNs), have existed in some form or other since the late 60s, but it is only recently that it has been practical to apply them for vision tasks, due in part to the huge computational power they require to pre-train, and the large amount of training data required to make them effective.

The inspiration for DNNs lie in neuroscience and the human vision system, with Hubel and Wiesel [59] discovering in the early 60s that the structure of the mammalian visual cortex is hierarchical. They observed that each level of the hierarchy was formed by a series of cells sensitive to distinctive patterns within small sub-regions of the input space called a receptive field. However, they state: *'Patterns of light stimuli most effective in influencing units at one level may no longer be the most effective at the next'* or, in other words, each level of the hierarchy comprises of a distinct set of 'feature detectors' (or 'filters') which operate independently. Additionally, two basic cell types were identified: simple cells, which respond maximally to specific edge like patterns in their receptive fields ('filters'), and complex cells, which have larger receptive fields and are invariant to the exact location of the stimulus (which can be considered to undertake a form of 'pooling' operation).

Early DNNs such as the Neocognitron of Fukushima [43] operated in an unsupervised way and imitated this structure, with alternating 'filter' and 'pooling' layers. Given that the 'filters' employed at each level are all identical (on the basis that the human vision system observes an object in the same way regardless of where it is in the spatial field) the network they propose is actually an early example of a specialisation of a DNN known as a Convolutional Neural Network (CNN). The network is called 'convolutional' since applying the same set local filters densely across the spatial plane

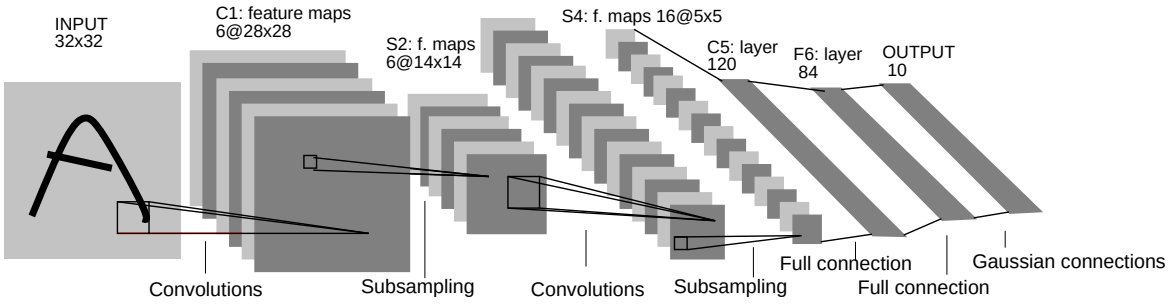


Figure 2.4: **Architecture of the LeNet-5 convolutional neural network.** Figure from LeCun *et al.*, 1998 [75]

can be seen as equivalent to the convolution operation. Following the application of the convolution, the resultant feature maps are passed through an activation function (*e.g.* hyperbolic tangent) to add a non-linearity. LeCun *et al.* [74, 75] developed this architecture further, using the back-propagation algorithm [109] to allow supervised training of the network.

LeCun's LeNet-5 network [75], shown in Figure 2.4, provided one of the first demonstrations of the successful application of CNNs to a supervised learning task, albeit one of limited scope, achieving very good performance on the MNIST digit recognition benchmark. However, until relatively recently their performance on more complex tasks such as natural image recognition was more limited primarily as a result of the computational complexity of training as well as the need to train on a large amount of data to avoid over-fitting. Two recent developments have finally permitted these obstacles to be overcome: (i) the advent of massively-parallel GPUs and (ii) the introduction of the first fully annotated large-scale image dataset in ImageNet [34].

With these two developments, Krizhevsky *et al.* [71] demonstrated that a CNN architecture could be applied to the problem of image classification with excellent performance, achieving not only state-of-the-art performance on the challenging ILSVRC-

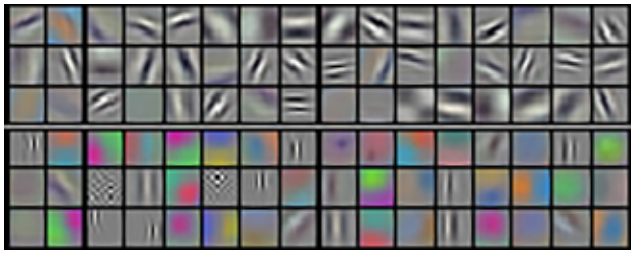


Figure 2.5: Visualisation of the 96 convolutional kernels of size $11 \times 11 \times 3$ learnt by the first layer of the CNN of Krizhevsky *et al.* [71] from images of the ILSVRC-2012 dataset.

12 benchmark [13] but outperforming all non-deep methods by a significant margin. To reduce over-fitting, they augmented the training set with jittered images and also adopted the 'drop-out' technique of Hinton *et al.* [58], which consists in randomly setting the activations in the final fully-connected layers to 0 (or 'dropping out') with 50% probability on each training sample. However, fundamentally the architecture was not so different to the one used to tackle the MNIST digit recognition benchmark more than a decade earlier.

2.3.3.1 CNNs as Feature Extractors

The input to the network of Krizhevsky *et al.* [71] is provided by the raw pixels of the training/test images and the output is a confidence score for the presence of each of the 1,000 ILSVRC classes. In this way, it can be seen to replace the entirety of the classification pipeline shown in Figure 2.2. However, an alternative way of looking at a convolutional neural network is a trained feature extractor, and this viewpoint is supported if we look at what each layer of the network is learning. The first layer's filters can be visualised directly as shown in Figure 2.5 (since they operate on the raw pixel values of the input image) and it can be seen that Gabor-like filters along with

blobs of colour appear to be being learnt. It is also possible to look at subsequent layers of the network by visualising activations, as shown in Zeiler & Fergus [139], and in line with what we would expect given the alternation of filtering and pooling layers in the network, the deeper layers appear to learn higher-level configuration of these base filters.

From an alternative viewpoint, the layers of a CNN can be seen to fall into a series of *non-linear filter* $\rightarrow$ *pool* steps, which are directly analogous to the same steps undertaken in the more conventional 'shallow' image classification architecture shown in Figure 2.2.

There, the analogy works at multiple levels. Firstly, the binning and gaussian weighting of gradients when computing the SIFT encoding of image patches can be considered to be a combined non-linear filter and pooling action. Following this, the construction of a 'bag-of-words' vector per feature relates to the 'non-linear filter' step, and the aggregation of those vectors into a single representation (either globally or by spatial region using the spatial pyramid kernel, as described in Section 3.1.3) relates to the 'pool' step. The only difference is that in a CNN architecture such a process is repeated many more times and each time in a similar manner, to extract progressively higher order features. This perhaps explains part of the performance edge that CNNs have exhibited compared to 'shallow' methods when trained over large enough datasets to support such multi-layered representations.

This being the case, [35] proposed cutting off the final layer(s) of a pre-trained network, and using the feature output as a general-purpose visual feature. This relies on the network having generalised enough to make these intermediate representations

discriminative when applied to images and datasets to which it was not exposed during training. Once features have been extracted from an intermediary layer in this way, they can be used in an exactly analogous way as our shallow image representations, and fed into a classical 'shallow' learning algorithm such as an SVM to learn discriminative classifiers across a different data (see Figure 2.2).

It turns out that using such a setup, we can achieve state-of-the-art performance over disjunct datasets also, and very quickly as it is much faster to train an SVM than an entirely new CNN (and requires much less data). We are effectively using the dataset used to train our original CNN as a 'feature generator', and offsetting the expensive feature learning and generalisation to a pre-processing stage and giving our SVM a much easier job to do.

A full evaluation of such 'deep' feature encodings is conducted in Chapter 4, along with a comparison to the 'shallow' methods described in the previous section.

2.4 The Missing Link: Large-scale Category Retrieval

In Section 2.3 we covered recent approaches for object category classification. However, in order to adapt such approaches so that they can be used efficiently and at scale for object category retrieval, there are two problems which must be addressed: (i) in general, the image representations used for state-of-the-art object category classification are often very high dimensional – for fast online ranking, we require both a way of ensuring that they can be stored in such a way as to fit into memory and an efficient way of ranking such codes (ii) learning of classifiers can also be expensive, and this also

needs to be optimised so that models can be learnt on-demand efficiently and without large sets of curated training data if the objective is to create a visual search system that does not rely on pre-trained models.

In order to sidestep the dual challenges of fast ranking and efficient learning, early vision approaches often instead took a simplified approach to the retrieval of categories, some examples of which are described further in Section 2.4.1. Following this, we explore some of the strategies that were used to address these issues in a more direct manner, and extend a full-pipeline approach to operate at scale in Section 2.4.2. Finally, in Section 2.4.3 we explore previous work on collecting training data in an automated fashion, in particular from the web, as this will form an important part of our on-the-fly visual search system.

2.4.1 Category Retrieval as Nearest Neighbour Search

The first real attempts to tackle the problem of category search at scale in the computer vision community generally took the approach of extending a strategy that had been very successful for instance search, that is recasting the problem as one of searching for nearest neighbours to a feature vector to avoid the need for explicit learning. Torralba *et al*. [124, 125] propose the use of the GIST descriptor [92], a global representation first developed for scene retrieval, to search for categories over a dataset of 79M images, all downsized to a scale of 32×32 pixels. Although still operating on the image-level rather than explicitly at the category level, both the use of GIST and downscaling is justified as: (i) the human visual system has been proven to be able to identify many object categories from very low resolution images, on the basis of just its 'gist'

and without pre-segmentation and, (ii) with a large enough dataset there should be enough images of any particular category which are visually very similar to a query image containing it that even a simple nearest neighbour search should return adequate results just through sheer volume of data. As might be expected, the proposed system performed adequately well more scene-like categories ('sunset', 'mountains' *etc.*) and also categories with a high level of occurrence ('person', 'computer monitor' *etc.*), but less well for more fine-grained queries.

2.4.2 Scaling Full Pipeline Approaches

When training models on demand in the context of a visual search system, often the training data available to us will be less plentiful and of lower quality than is the case when training offline from a pre-curated dataset. This has the implication that the image representation and learning architecture we use should be discriminative enough to produce good models even with this lower quality training data. In part, the more advanced encoding methods introduced recently and described in Section 2.3 provide part of the answer to this problem, as they often are discriminative enough that they can be used with fast linear classifiers whilst still producing performant models. However, one consequence is that such encodings tend to be high dimensional. For example, the Fisher encoding with the same experimental parameters as proposed in the original paper produces an encoding of $\sim 327,000$ dimensions. At such sizes, the features for a dataset of 1M images would take 1.3TB of memory to store, far in excess of what is common even in large server systems.

There have been two predominant approaches which have been proposed to deal

with this problem. The first is efficient compression algorithms to reduce the storage requirements of high-dimensional encodings such as the Fisher encoding (explored in Sections 2.4.2.1 and 2.4.2.2). The second is to add a second layer of encodings on top of the encodings, through the application of an attribute-based approach (explored in Section 2.4.2.3). In addition, there has been a further line of work on extracting mid-level features capturing higher-level concepts directly [41,78,117] of which the attribute-based approach can be considered one example. Whilst such representations can often be more compact, they often require far more training data to extract (*e.g.* deformable part models [41] require ground-truth annotations for not just object category bounding boxes, but also their parts), and so will not be explored in this thesis further.

2.4.2.1 Feature Compression using Binarisation

One effective strategy for compressing real-valued features is through their projection into Hamming space, or in other words down to a binary representation. This can be done through an embedding function $e : \mathbb{R}^D \rightarrow \mathbb{B}^M$ which maps the D-dimensional feature $\boldsymbol{\phi}_i$ to $\mathbf{b}_i = e(\boldsymbol{\phi}_i)$ in the M-dimensional Hamming space $\mathbb{B}^M$. The challenge is to design the embedding function such as to incur minimal information loss in the transformation.

In general, such binarisation through Hamming embedding approaches have been explored extensively for use in nearest neighbour search. The reason for this is that features in Hamming space can be compared using the Hamming distance (defined as the number of bits in two binary vectors which differ), and this can be computed very efficiently as modern CPUs have highly optimised instructions which can be used for

this purpose. A good embedding function e should fulfil the property:

$$\mathrm{NN}(\boldsymbol{\phi}_q) \approx \underset{\boldsymbol{\phi}_i=\boldsymbol{\phi}_1 \ldots \boldsymbol{\phi}_N}{\arg\min}\ e(\boldsymbol{\phi}_q)^\top e(\boldsymbol{\phi}_i) \tag{2.15}$$

Where $\boldsymbol{\phi}_q$ is the query feature. That is to say, the nearest neighbour(s) in Hamming distance is most of the time the closest also in the original (Euclidean) feature space. Andoni and Indyk [5] propose the use of Locality Sensitive Hashing (LSH) [23] using random projections to generate this embedding. This involves generating M random projections $\mathbf{p}_{j=1\ldots M} : \mathbb{R}^d \rightarrow \mathbb{R}^1$ to form the $D \times M$ matrix $\mathbf{P} = \left[\mathbf{p}_1 | \ldots | \mathbf{p}_M\right]$ and then thresholding by sign to obtain our binary representation: $\mathbf{b}_i = \mathrm{sgn}\left(\mathbf{P}^\top \boldsymbol{\phi}_i\right)$. It is shown by Andoni and Indyk [5] that the closer two features are in Euclidean space, the more probable they are to be assigned the same sign when multiplied by a random projection in this way, and so taking the combination of many different random projections causes the criterion in Equation 2.15 to be satisfied.

Jegou *et al.* [66] explore an alternative to LSH based on making an over-complete expansion $e : \mathbb{R}^D \rightarrow \mathbb{B}^M$ where $M > D$ obtained using a Parseval tight frame, followed again by thresholding on sign. In more detail, we first zero-centre our real-valued features $\boldsymbol{\phi}_i \rightarrow \boldsymbol{\psi}_i \in \mathbb{R}^D$, following which the binarisation is performed as:

$$\mathbf{b}_i = \mathrm{sgn}(\mathbf{U}\boldsymbol{\psi}_i) \tag{2.16}$$

Where $\mathbf{U}$ is defined as the Parseval tight frame, and is computed by keeping the first D columns of an orthogonal matrix, obtained from the QR-decomposition of a random $M \times M$ matrix. They show that for the case where $M > D$, the performance of this

approach exceeds that of random projection LSH. It should be noted that although the resultant binary feature $\mathbf{b}_i \in \mathbb{B}^M$ is of higher dimensionality than the original real-valued feature $\boldsymbol{\phi}_i \in \mathbb{R}^D$, in general a reduction in storage is still obtained as each dimension requires as little as one bit of memory, versus each dimension in the original real representation which require 32 bits (in IEEE single precision format, for example).

Finally, in the case of object category retrieval we are not concerned with the problem of nearest neighbour search, but rather given a linear model $\mathbf{w}$ (*e.g.* learnt using a linear SVM) and an image feature $\boldsymbol{\phi}_i$ we wish to quickly and efficiently compute a ranking score per-image by applying the model to the feature $\mathbf{w}.\boldsymbol{\phi}_i$. However, assuming both $\mathbf{w}$ and $\boldsymbol{\phi}_i$ are ℓ_2 normalised, it can be shown that Euclidean distance $\|\mathbf{w} - \boldsymbol{\phi}_i\|_2$ and the dot product $\mathbf{w}.\boldsymbol{\phi}_i$ are equivalent up to a scaling and a translation:

$$\begin{aligned} \|\mathbf{w} - \boldsymbol{\phi}_i\|_2 &= \mathbf{w}.\mathbf{w} + \boldsymbol{\phi}_i.\boldsymbol{\phi}_i - 2\mathbf{w}.\boldsymbol{\phi}_i \\ &= 2 - 2\mathbf{w}.\boldsymbol{\phi}_i \\ \mathbf{w}.\boldsymbol{\phi}_i &= (2 - \|\mathbf{w} - \boldsymbol{\phi}_i\|_2)/2 \end{aligned} \tag{2.17}$$

Therefore, given that the fast Hamming embedding can also be used to approximate distances in Euclidean space (although technically for only the N nearest neighbours) the resultant binary codes can be used to efficiently compute the output ranking from the application of a linear classifier $\mathbf{w}$ to a large dataset.

In addition, there has been some work to bypass the need to binarise real-valued features entirely by extracting binary features directly using local patch encodings such

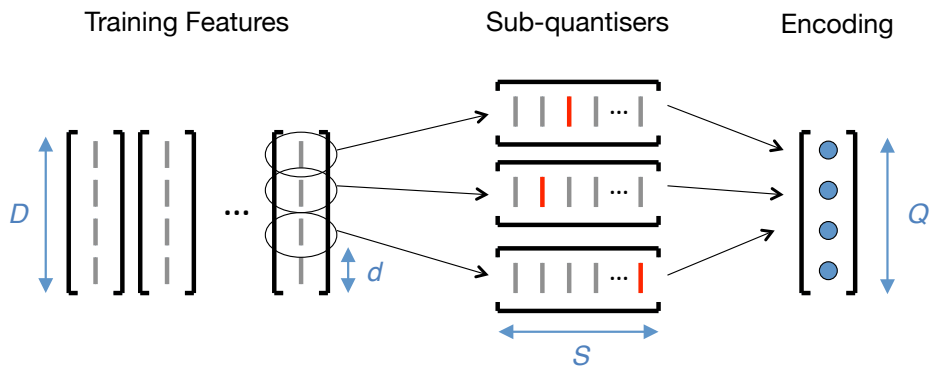


Figure 2.6: **Product Quantisation.** Input training features of dimension D are split into d dimensional sub-features, each of which is used to train a k-means codebooks with S centres. At encoding time, the feature to encode is again split into the same d dimensional sub-features, and each is encoded as its index in the corresponding codebook (or subquantiser) resulting in a Q dimensional output encoding of indexes. If $S \leq 256$ each index can be stored in a single byte which, compared to the 4 bytes required to store the original input feature in single precision floating point format, yields an additional compression of $4\times$. Therefore, the total resultant compression ratio is $1 : 4d$.

as LBP [91], BRISK [76] and similar [1,20,108]. These have mainly been used for local feature matching, but can also be encoded into a binary bag-of-words representation directly using the Hamming distance [141]. We do not explore these approaches further in this thesis, as the processing of binary features necessarily incurs some quantisation error, and so we reason that it is best to avoid feature binarisation until the latest stage possible and thus apply it once our full floating point image representation has been computed.

2.4.2.2 Feature Compression using Product Quantisation

Product quantisation (PQ) is another popular compression method which transforms a real-valued vector into a compressed vector of integer indexes. Originally proposed for the compression of local features by Jegou *et al.* [65], it has since also been applied to the compression of global image features [110]. PQ operates by splitting the original

feature vector into Q sub-vectors of dimension d, where $Q = D/d$. For each sub-vector, a codebook or *sub-quantiser* q of S representative 'codewords' $[\boldsymbol{\nu}_1, \ldots, \boldsymbol{\nu}_s]$ is learnt by clustering using k-means over a training set, in an analogous way to the 'bag-of-words' method described in Section 2.2.1. The PQ representation of the feature $\boldsymbol{\phi}_i$ is then obtained by storing the index of the nearest codeword for each sub-vector, using the matching sub-quantiser. Often the sub-quantisers are trained with *e.g.* $S = 256$ centres so that each group of d dimensions in the input vector translates to a single byte index in the output vector.

Using product quantisation, ranking for the object category retrieval problem (that is to say the computation of $\mathbf{w}.\boldsymbol{\phi}_i$) can be conducted very efficiently by precomputing a lookup table $\mathbf{W}^{LUT}$ of size $Q \times S$ where:

$$\mathbf{W}_{ij}^{LUT} = \mathbf{w}.\boldsymbol{\nu}_{ij} \tag{2.18}$$

2.4.2.3 Compact Codes using Attributes

An alternative approach to the compression methods outlined in the last two sections is to effectively add another layer of encoding on top of them to produce a second-level encoding containing higher-level information related to the task at hand (that is to say, object category classification) and in the process make them more compact. This is not dissimilar in spirit to adding another layer in the CNNs described in Section 2.3.3, but can be equally applied to 'shallow' representations.

One way we can do this is by expressing our image in terms of its relationship to a set of base classes or *attributes*. This idea was first introduced by Farhadi *et al.* [39],

with the proposition being that, for example, a 'goat' could be expressed in terms of a high score for the pre-trained classifiers for 'horn', 'head', 'leg' and 'fur', for example. This was further developed by Torresani *et al*. [126] who observed that the 'attributes' used to describe a category need not be semantically related to the category at all, and proposed the *classeme* feature, which encoded the output of a bank of pre-trained object classifiers for a selection of hand-curated classes. Given the objective of the work was a general purpose representation for object category search on the web, the classes were selected from a web-based concept ontology and training data collected automatically using the top 150 images returned from the Bing image search engine. Given that each dimension of the classeme feature essentially represents the output score of one of these pre-trained classifiers, they are very amenable to compression to a binary vector simply by thresholding on the sign of the output score: $\mathbf{b} = \text{sgn}(\boldsymbol{\phi}_{\text{cls}})$. The pre-training of the bank of abstract attribute classifiers is essentially a way to offload the learning of a powerful additional layer of non-linearity to training time.

Following Torresani's original paper, several improvements of this feature were proposed, including the PiCoDes feature [18] and meta-class [16] features. A good overview is provided in [17]. The motivating application for all of these features is very similar to our objective in this thesis, that is to say the efficient retrieval of novel object categories from large datasets, and as such all are also similarly compact binary representations (PiCoDes, for example, is of size 2048-bits per feature) in order to allow for fast ranking (and methods have further been proposed for even faster approximated ranking using sparse classifiers in [105]). We explore the performance of these features for on-the-fly category retrieval, comparing its performance to the compression of more

classical representations as described in the previous sections, in Chapter 5.

2.4.3 Training from the Web

As our aim is to develop a system for category search that allows visual data to be queried in as natural a manner as is possible for textual data today (through text search engines such as Google and Bing) the question arises as to what is the most natural manner to specify a visual query? The approach historically used by researchers in the vision community has been to facilitate query by template image [118, 125] or by selection from an increasingly growing bank of pre-trained classifiers [34]. However, what if the user doesn't have a query image to hand, or the category they wish to find doesn't exist in the classifier bank?

The approach that we aim to take in our category retrieval system, introduced in Chapter 5, is search by 'on-the-fly' training of an object model on the basis of a text query. That being the case, in this section we consider the final piece of the puzzle necessary for such an approach – that is the automated collection of training data, in particular from the web.

There have been a few attempts to automatically mine visual data from the web in the past decade. Berg and Forsyth [14] used a hidden topic model (LDA) to automatically cluster together images extracted from webpages obtained by text search for animals. Fergus *et al.* [42] take a similar generative semi-supervised approach, this time using the images returned from Google Image search and pLSA instead of LDA, as do Li and Li with their OPTIMOL system [77]. Liu *et al.* [80] take a different approach by collecting a large volume of extensively labelled data and then using images

tagged with a given label to search through unannotated photo collections by nearest-neighbour search, although the simplicity of the approach is also its weakness, as it relies on the presence of rich annotation data. Finally, Schroff *et al.* [114] return to the problem of crawling images from the open web, except this time adopting a discriminative approach based on linear SVM classifiers, showing this to perform better than the topic-model based classifiers explored in previous papers. In all cases, the focus has been on 'cleaning up' images downloaded from the web, and/or the architectures are intended to operate as background 'crawlers', informing many of the design choices made and rendering such systems too slow for real-time operation.

An idea first explored by Bergamo and Torresani [15] is that training images from web image search is normally quite different from those contained in any target dataset. The general problem of using training data which is sourced from a different distribution to the test data is termed domain adaptation, and there have been several recent papers on doing this in an unsupervised manner [9, 50, 51, 61]. However, the basis of Bergamo and Torresani's work is still the idea that image results sourced from the web need to be 'cleaned up' in some way. They thus formulate the domain adaptation problem in the following way: 'given a set of images of a class with strong labels, how do we select good additional images from a weakly labelled class which will help us with training whilst minimising the chance the picked images are false positives?'.

With the improvement in the quality of the web image search engines such as Google Image search and Bing for many queries however, as a result of click-through data and continuously tweaked ranking algorithms, many of the above approaches have also become obsolete to an extent. Despite this improvement, there is still no answer to

the question of polysemy (that is to say distinct visual concepts with the same textual description – *e.g.* Jaguar the cat *vs.* Jaguar the car) but for the most part the number of false positives and noise on the first few pages for a query on Google Image search is now much lower than it was before. Nonetheless, there has been relatively little attention paid to harnessing this recently improved source of training data directly to train models using state-of-the-art image representations as described in Section 2.3 above. The only other work that we are aware of is that of Kumar and Seitz [72], published only earlier this year, who also harness images from Google Image search to handle on-the-fly queries. However, their focus is on combining data from multiple different data sources (*e.g.* GPS, location, time in addition to visual data) and, in contrast to the work in this thesis, they use very simple visual features (colour histograms *etc.*) for the visual component of their system. Nonetheless, it would be interesting to combine the state-of-the-art visual features explored in this thesis with the multi-modal approach to queries described in Kumar and Seitz [72] in future work.

2.5 Datasets

In this section we introduce some of the datasets used to evaluate different approaches to object category retrieval and classification throughout this thesis. As mentioned in Section 2.3, there have been a number of classification 'challenge' datasets which have gained popularity within the community, along with long-established standard benchmarks, and for the most part we use these as the basis of our experiments, particularly in Chapters 3 and 4 which follow.

2.5.1 Caltech datasets

Caltech 101 [40] is both the oldest and the easiest of all datasets used in this thesis. It is included primarily to provide a baseline comparison to older methods, and as it remains a classic benchmark for classification performance in the community (albeit one which is rapidly becoming obsolete as focus shifts to larger and more challenging datasets). It comprises images of objects belonging to 102 categories (including one 'background' category) with around 40 to 800 images per category. Most categories have around 50 images with the object in question centred and aligned within each image in a consistent manner.

There are no clearly defined training/validation/test splits, nor a standardised testing methodology. However, it is common to split the dataset as follows:

- **Training set** of $N_{train} = 15$ *or* $N_{train} = 30$ randomly sampled images per class
- **Test set** of $N_{test} = \min(30,\ N_{class} - N_{train})$ randomly sampled images per class

We use a total of $N_{splits} = 3$ splits which are all pre-determined and sampled randomly in all of our experiments. Some classes have N_{class} number of training images less than required to sample 30 class instances for the test set, in which case all of the annotated images not assigned to the training set form the test set.

As per standard practice for this dataset, we present our results over the test set using our N_{splits} splits in terms of average classification accuracy (which is equivalent to the mean of the diagonal of the confusion matrix computed for the 102 classes, including the 'background' class). The mean of the average classification accuracy

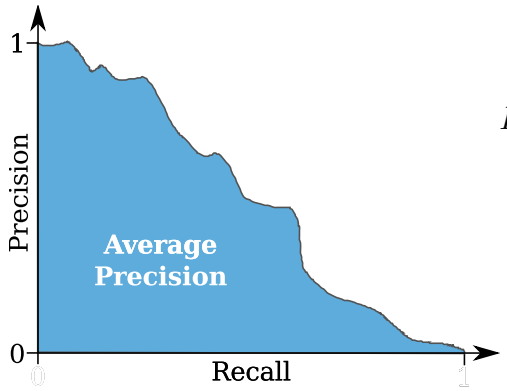
across splits is then taken, and bounds given on the variability across splits in terms of standard error.

Caltech-256 [54] is similar to the Caltech-101 dataset, but with annotations for 257 categories (including one 'background' category). The data is a little more challenging than Caltech-101 though, as the images are not aligned in the centre of the frame in the same way, although in general still comprise the category at hand quite often on a plain background making it still easier than the PASCAL VOC datasets described below. Evaluation is again by average classification accuracy over splits of 15 or 30 images, with 30 images per class assigned to the test set. For our experiments, we use $N_{splits} = 2$, with two randomly sampled splits which remain the same across experiments.

2.5.2 VOC PASCAL datasets

The VOC PASCAL datasets [37] are more challenging, comprising images downloaded from the photo sharing site Flickr for twenty different object classes. We primarily use the **VOC 2007** version of the dataset, as this was the last version to come with publicly available training, validation and test splits. Results are also provided for newer **VOC 2012** version of the dataset in Chapter 4, for which only the training and validation sets can be downloaded but an evaluation server is available for remote evaluation. VOC 2007 contains around 10k images in total, and VOC 2012 around 20k.

As a direct consequence of it being downloaded from a real-world community photo collection, it is a more challenging dataset than Caltech-101, with each image often containing instances of multiple classes and with objects being in variable positions



$$Precision(r) = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}[..r]\}|}{r}$$

$$Recall(r) = \frac{|\{\text{relevant}\} \cap \{\text{retrieved}[..r]\}|}{|\{\text{relevant}\}|}$$

Figure 2.7: The relationship between precision-recall and average precision.

within the image and often partially occluded or clipped. Despite its limited size, with the provision of full annotations and with the classes themselves remaining very challenging the PASCAL datasets remain one of the most widely used benchmarks in the community.

Evaluation over the PASCAL VOC 2007 dataset is by average precision (AP). This is a standard evaluation metric for ranked results in the information retrieval community [84], and involves computing the precision and recall of a query defined as in Figure 2.7. Precision at N roughly indicates the proportion of relevant to retrieved images after the Nth result, and so is a particularly important metric in web-scale search engines where the 'goodness' of the first few pages is paramount to whether it is perceived as working correctly or not. Conversely, recall at N indicates the proportion of all relevant documents that have been returned after the Nth result. The average precision is computed by calculating the area under the precision-recall curve. For PASCAL VOC 2007, the AP is computed per class, and then the mean taken across classes to give mean average precision (mAP).

2.5.3 ILSVRC datasets

The ILSVRC dataset [13] is a partially annotated subset of the complete ImageNet dataset [34] comprising 1,000 categories and 1.2M images in the publicly available training set + a further 150,000 images in the validation and test sets. Categories are sourced from the WordNet ontology and thus there is a focus on fine-grained recognition, making the dataset more challenging than the PASCAL VOC datasets in terms of categories in addition to scale. Given the annotation for the dataset is only partial, evaluation is conducted by average top-5 and top-1 accuracy, where the top-N accuracy is a measure of the fraction of test images for which the correct label is not among the top N labels predicted as most probable. This dataset is used for evaluation in Chapter 4 and as distractor data in Chapter 5.

2.5.4 MIRFLICKR-1M dataset

The MIRFLICKR-1M dataset [60] comprises 1M unannotated images (aside from quite noisy image tags). The dataset represents a snapshot of images taken by popularity from the image sharing site Flickr, and thus is more representative of typical web-based consumer photography than ImageNet, which although also sourced from Flickr was collected through queries for often very specific terms from WordNet. Despite the lack of strong annotations, this dataset is useful for real-world evaluation and is used as both distractor data and in its own right in Chapter 6.

2.5.5 BBC News dataset

A dataset of 5M+ images sourced from video data provided by the BBC of News broadcasts is also used for evaluation in Chapter 6. This dataset comes with no annotations at all, but provides a large real-world collection of images with which to test our approach. The images themselves are sourced from all news programmes broadcast over all BBC channels from 6pm until midnight from 2007 to 2012. It consists of $10,132$ hours of footage from $17,401$ different programmes, resulting in the 5M+ keyframes.

Part I

Feature Evaluation

Chapter 3

Shallow Features for Image Classification

In this chapter we conduct a comprehensive evaluation of shallow encoding methods for object category classification and retrieval. Our starting point is the 'bag-of-words' classification pipeline shown in Figure 3.1. In general it comprises the following three steps: (i) extraction of local image features (*e.g.*, SIFT descriptors), (ii) encoding of the local features in an image descriptor (*e.g.* a histogram of the quantised local features), and (iii) classification of the image descriptor (*e.g.* by a support vector machine). As described in the introduction, over the past decade there has been a concerted effort to improve the second component of this pipeline, *i.e.* the encoding of local features into a global representation.

Despite the large volume of papers proposing new encoding methods for object category retrieval over the past decade [99,133,137,138,143], many of which have reported very good results on challenging benchmarks such as the PASCAL VOC classification

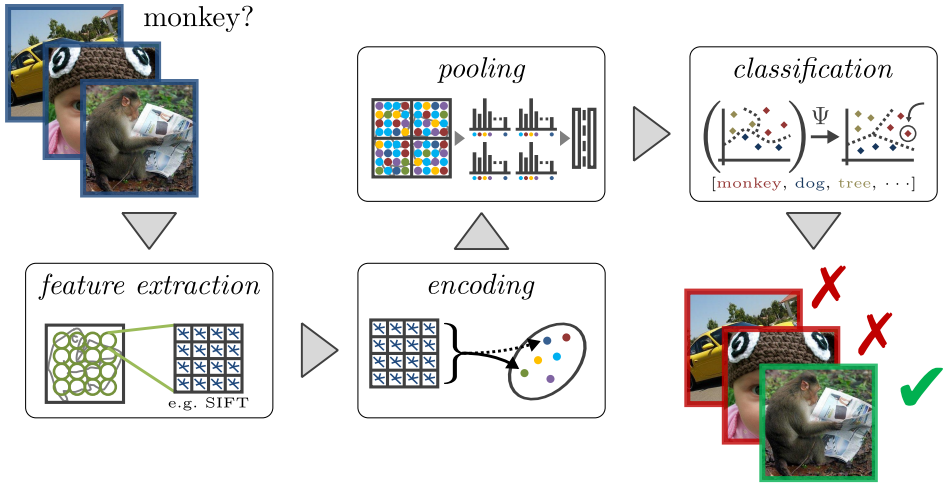


Figure 3.1: Bag-of-words object category recognition pipeline

challenge [37], often it is difficult to compare results between papers given that each evaluation often uses its own subtly different experimental setup, often the details of which are not fully disclosed. Unfortunately, the performance of object recognition methods depends very strongly on all the stages of the pipeline, and especially on the feature computation step, making comparison like-for-like even more challenging.

In this chapter, we consider the same five recent encoding methods as described in Chapter 1, namely: histogram encoding (or plain-vanilla bag-of-words), kernel codebook encoding, LLC, super vector encoding and the Fisher encoding. In order to asses each method in isolation from the details of feature extraction, learning *etc.*, we take the approach of fixing all other elements of the encoding pipeline. In this way, it is possible to analyse which aspects of the different constructions are important for performance and which are not, methodically changing one parameter of the experimental setup at a time. The overall picture that emerges cannot be inferred from the original publications alone. Given that in this thesis, one of our primary concerns is the speeding up of object category recognition systems to make visual search over object

categories tenable in real-time, we therefore pay particular attention to not only how these different methods compare in terms of classification accuracy, but also consider other metrics pertinent to their use in an on-the-fly system, namely memory consumption and speed, with a view to incorporating the most performant into a system for on-the-fly category search (described in Chapter 5).

We begin by considering in Section 3.1 various design choices that apply to the general construction of a bag-of-words based pipeline on a relatively high level. Following this, we present full details of our experimental setup along with results for all of the encoding methods in Section 3.2 before analysing their comparative performance in Section 3.3.

3.1 Pipeline Details

3.1.1 Feature Extraction

There are a variety of different local feature extractors which have enjoyed wide use in the community for object retrieval. We focus on SIFT features [82] due to their proven performance on the object category retrieval task in the PASCAL VOC classification challenges [37], having been used by the winners of the competition in some way in all years of the challenge from 2007–2012 (with more recent methods instead focusing on deep methods described in the next chapter which do not require explicit local feature extraction). Several extensions to SIFT are also possible, with several authors incorporating colour information for example [128]. However, none have really been shown to offer a significant performance boost over SIFT without incurring significant

extra computational cost. Other local feature extractors such as SURF [11] have been proposed, but as we will see, the time required for local feature extraction of even SIFT is generally far less than the time required for image encoding and so we stick with SIFT due to its relative ubiquity and for ease of comparison to other literature.

3.1.2 Dimensionality Reduction

It is common to apply some form of dimensionality reduction to the local features prior to encoding. Depending on how the dimensionality reduction is undertaken, this may have the positive effect of both reducing the storage requirements of the local features whilst also further removing noise incurred due to quantisation loss and other sources of low-variance noise. The simplest and most common approach is to employ unsupervised dimensionality reduction through Principal Component Analysis (PCA).

Principal Component Analysis can be defined as an orthogonal linear projection $\mathbf{W}_{PCA} \in \mathbb{R}_{N \times N}$ which transforms a set of features of dimensionality N onto a lower dimensional subspace known as the *principal subspace*. In this subspace, components are sorted in order of decreasing variance, such that the first component has the highest variance among all possible linear projections of the data, the second component has the second highest and so on. Each principal component also has the property of being orthogonal in the projected space to all others, and thus an important property of PCA is that it performs de-correlation, or in other words the covariance of the PCA-transformed data is diagonal (which has been shown to be important as a pre-processing step when training the GMMs of the Fisher encoding [99], for example, where in our

experiments we use PCA to de-correlate and down-project to 80 dimensions).

Given that the last principal components by definition have the smallest variance and, as such, might be less important in the data representation, the projection $\mathbf{W}_{PCA}$ can be used to reduce the dimensionality of our features to M by retaining just the first M rows of the projection *i.e.* using the matrix $\mathbf{W}_{PCA} \in \mathbb{R}_{M \times N}$ to down-project the data.

Whitening. Following the application of PCA, it is almost always important to apply a whitening stage, which normalises the resultant components of our lower dimensional feature to unit variance. We introduce the process here as it naturally follows on from our explanation of PCA. Whitening transforms the variances of all components post-PCA so that they are equal (as opposed to being in descending order as is the case with unwhitened PCA features, given the PCA transformation is a simple linear projection). The reason this is important is because when using unbalanced features, particularly when using a non-linear kernel, a dominant component containing little information can easily swamp out a weaker component containing more information. Even in the linear kernel case, where the cost function should be able to automatically learn the importance of even unbalanced components, such unbalanced components can otherwise cause issues with model regularisation.

The application of whitening is as simple as weighting each component of the projected features by the inverse square roots of the eigenvalues of the components *i.e.* $\mathbf{W}_{whitened} = \sqrt{\mathbf{D}^{-1}_{1\ldots M}}\mathbf{W}^{\top}_{PCA} \in \mathbb{R}_{M \times N}$ where $\mathbf{D}$ is a diagonal matrix of the top-M eigenvalues, and results in data with an identity covariance matrix.

Best Practice. We generally apply dimensionality reduction to the SIFT features or not following the methodology of the original publications for the encoding methods described in this chapter and the corresponding results presented in Section 3.2 (for example, dimensionality reduction is applied for the Fisher encoding, but not for the bag-of-words encoding, and there are particular reasons why dimensionality reduction may be critical to the performance of several specific methods which will be explored further below). However, we have found subsequent to conducting this work that reducing the dimensionality of SIFT features and whitening is almost always beneficial.

The level of dimensionality reduction applied must be decided according to the balance between the desire to remove noisy low-variance components from the local features and reduce memory consumption as much as possible with the requirement that not too much information is lost, as this will then negatively impact the discriminativeness of the dimensionality reduced features. Sanchez *et al.* [112] explore this balance further specifically for the Fisher encoding, *but we have found that a value of around 80-dimensions strikes the best balance in most cases.*

3.1.3 Pooling and Post-Processing

3.1.3.1 Incorporating Spatial Information

By moving from the representation of an image as a collection of local features to the bag-of-words representation, we necessarily lose some information about the absolute and relative spatial configuration of the original features. A standard way of introducing weak geometry and therefore counteracting this information loss is the use of

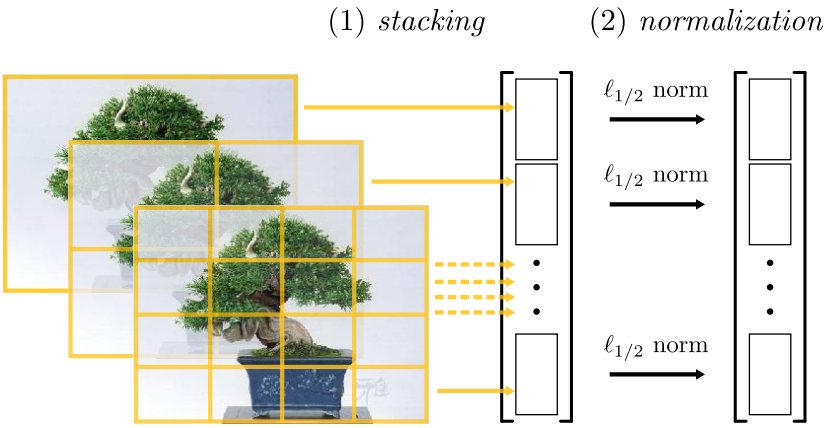


Figure 3.2: Spatial histogram generation. (1) The image is sub-divided into a collection of sub-regions of equal sizes. The encoding for each sub-region is computed and stacked. (2) The encoding for each sub-region is normalised to produce the final spatial histogram.

spatial histograms [53, 73]. Essentially, this entails the splitting of the image into a collection of sub-regions (*e.g.* quadrants or horizontal strips), computing the encoding for each of these sub-regions independently and then concatenating these encodings into a single larger bag-of-words descriptor of length KB where B is the number of spatial sub-regions (figure 3.2). The concept can be extended to any of the encodings described in this chapter. Note that each spatial region is normalised individually prior to stacking, using the same normalisation method as used to globally normalise the feature vectors in each case.

An alternative to pooling using spatial pyramids is instead to concatenate the spatial coordinates of each local feature within the image as additional feature dimensions as described in [111]. This idea is explored further in Chapter 4.

3.1.3.2 Pooling functions

When computing the encoding for each spatial region, the image features can be pooled in one of two ways: sum pooling, in which case the encodings of the local features in

a given region are combined additively, or max pooling, in which case each bin in the encoding is assigned a value equal to the maximum across local feature encodings in that region. In the original publications, max pooling is used for the LLC encoding, and sum pooling for all other methods, so we do the same in our experiments.

Best Practice. Once again, we use the pooling method specified in the original publications for each encoding method in the evaluation presented in this chapter. However, the properties of codes obtained using either sum or max pooling are quite different.

As shown in Murray and Perronnin [88], although sum pooling is more commonly applied a disadvantage is that frequently-occurring features will be more influential on the final representation than rarely-occurring ones. Max pooling, conversely, better captures this information and so often performs better for the object categorisation task, and having explored this further this seems to be the case for the histogram encoding, kernel codebook encoding and LLC methods covered in this chapter.

Unfortunately, max pooling cannot be applied directly to the Fisher or Super-vector encodings though, as it treats each feature dimension separately (whereas in reality, for both of these encodings, dimensions are highly correlated). For these representations either sum pooling must be used, or the generalised max pooling method proposed in [88].

3.1.3.3 Encoding Post-Processing and Feature Maps

For training with a support vector machine (SVM) the simplest approach is to use the encodings directly with a linear kernel $K(\mathbf{f}, \mathbf{g}) = \langle \mathbf{f}, \mathbf{g} \rangle$. While linear kernels are very efficient [69] during training as distances can be computed as a simple dot product, non-linear kernels can yield better classification accuracy, particularly when the underlying features are not discriminative enough for the task at hand to be linearly separable in feature space [140] (*i.e.* there is a balance between the non-linearity captured in the kernel, and that in the feature encoding itself). A class of kernels that are almost as efficient as the linear ones which can be used in these cases are the additive homogeneous kernels [131] $K(\mathbf{f}, \mathbf{g}) = \sum_{i=1}^{D} k(\mathbf{f}_i, \mathbf{g}_i)$, where k is itself a kernel on the non-negative reals. Examples of k include the Hellinger's (Bhattacharya's) kernel $k(f, g) = \sqrt{fg}$ and the χ^2 kernel $k(f, g) = 2fg/(f+g)$, both of which were designed for the task of histogram comparison, which is precisely the task at hand. While these kernels are normally defined for non-negative vectors (histograms), they can be extended to arbitrary vectors by setting $k'(f, g) = \text{sign}(fg)\, k(|f|, |g|)$.

The computational advantage of using additive kernels is that they can be represented as linear ones up to the computation of an efficient *feature map* – which can be defined as a transform to be applied to all elements of a feature to produce a new feature in the additive kernel space. For instance, for the Hellinger's kernel it suffices to consider the feature map defined by $[\Psi(\mathbf{f})]_i = \sqrt{\mathbf{f}_i}$, as in fact $K(\mathbf{f}, \mathbf{g}) = \langle \Psi(\mathbf{f}), \Psi(\mathbf{g}) \rangle = \sum_{i=1}^{D} \sqrt{\mathbf{f}_i}\sqrt{\mathbf{g}_i} = \sum_{i=1}^{D} \sqrt{\mathbf{f}_i \mathbf{g}_i}$. For the χ^2 and other kernels one can use the approximated feature maps introduced by [131], which are nearly as efficient. These feature maps can be applied directly to the feature vectors as part

of the feature computation stage, meaning that training can be undertaken using the linear kernel on the post-processed feature vectors (and thus it is possible to use highly optimised linear SVM implementations such as LIBLINEAR [38] whilst still enjoying the advantage of using a non-linear kernel).

Best Practice. Aside from the plain-vanilla histogram encoding method described in this chapter, many of the other methods have been found to work well even using the linear kernel – and indeed the ability to employ lighter-weight kernels is in part the motivation for the research effort that has been expended on developing more sophisticated encoding methods. Once again, in this chapter we focus on applying kernels following the approach outlined in the original papers. However, since the work undertaken here was completed, there has been some further exploration in the literature of the optimal post-processing regime for more sophisticated encodings and also of local features.

Perronnin *et al*. [99] explore further the use of the Hellinger kernel with the Fisher encoding, describing it as a special case of power normalisation of features. There it is motivated by the observation that: (1) the dot-product used in the linear kernel if the input features are ℓ^2 normalised is equivalent to the ℓ^2 distance, (2) the Fisher encoding is sparse, and the ℓ^2 distance is a poor measure of similarity on sparse vectors and (3) power normalisation offers a way to reduce the sparseness of the encoding, thus improving performance. An alternative interpretation is that power normalisation downplays the influence of features which occur frequently within a given image (or 'bursty' visual features) [98]. Our experience is that for all of the encodings described in

this chapter, the application of the Hellinger kernel boosts performance over standard object category recognition benchmarks such as PASCAL VOC when compared to the application of the plain-vanilla linear kernel.

Meanwhile, Arandjelović and Zisserman [6] have shown that similarly applying power normalisation (or more specifically the Hellinger kernel) to local SIFT features prior to encoding results in a further boost to performance. In this chapter, such modified SIFT features are not used further. However, they are used in all subsequent chapters.

3.1.3.4 Normalisation

SVMs usually work better if the data is properly normalised [131]. The appropriate normalisation for a linear SVM is ℓ^2 [99, 131] which we apply after all other post-processing of the feature vectors (application of feature maps *etc.*) One interpretation as to why ℓ^2 works best is that after normalisation the inner product corresponds to the cosine similarly (given that the cosine similarity is defined as $\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \|\mathbf{b}\| \cos\theta$ and $\|\mathbf{a}\| = \|\mathbf{b}\| = 1$ in this case).

3.1.4 Learning

All the experiments use a linear SVM on top of each encoding. The parameter C of the SVM (regularisation-loss trade off) is determined on a validation set (on the provided train and val split in the PASCAL VOC data and on a random split in Caltech-101).

3.2 Experiments

This section describes the experimental setup, including the parameters used for each of the encodings. In this chapter we evaluate over the PASCAL VOC 2007 dataset using mean Average Precision (mAP) and the Caltech-101 dataset using average classification accuracy. The standard experimental setup for these datasets described in Section 2.5 is employed.

3.2.1 Features, spatial binning and learning

Features. All SIFT features [81] are extracted densely across the image on a fixed grid, with a spatial stride of between two and five pixels and 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. Additionally, low contrast SIFT descriptors are detected by measuring the average of the gradient magnitude (in the descriptor support) and dropped when this magnitude is below a certain threshold (with our chosen threshold, about 5% of the SIFT features in the PASCAL dataset were dropped). To simplify reproducibility, the SIFT descriptors are computed by using the `vl_phow` command included in the publicly available VLFeat toolbox [130], version 0.9.13. This is an implementation very similar to the reference implementation of Lowe. The only difference is that we enable the `'fast'` option in the toolbox, which causes the extractor to use a piecewise-flat rather than Gaussian windowing function to accumulate the contributions of gradients in an individual feature (but then re-weight the individual bin later by the average of the Gaussian window over the spatial support

of that bin). This produces features very close to Lowe's original but is much faster, requiring well under a second on a VGA image.

Spatial binning. We bin our features spatially as described in Section 3.1.3. The spatial regions are obtained by dividing the image in 1×1, 3×1 (three horizontal stripes), and 2×2 (four quadrants) grids, for a total of 8 regions, for the PASCAL VOC data, and in 1×1, 2×2, and 4×4, for a total of 21 regions, for the Caltech-101 data. This is to mirror the spatial configurations common in the community for these datasets. However, we also found that the combination of horizontal strips with quadrants indeed provided the best performance on the PASCAL VOC dataset, and just quadrants on Caltech-101, presumably because the objects in the Caltech-101 dataset are generally centred and fill the frame in contrast to the more freeform nature of objects in the PASCAL VOC dataset.

Encoding post-processing and learning. The learning stage is provided by a linear SVM, with additive kernel maps used to provide a level of non-linearity in the case of some of the encoding methods (either the Hellinger's kernel or the χ^2 approximated feature map introduced by [131]). All features are ℓ^2 normalised before being passed into the SVM, as described in Section 3.1.3.

3.2.2 Encoding Methods

Baseline: histogram encoding. The baseline encoding is a histogram of visual words obtained with hard quantisation (for a description of this and all other encoding methods, refer to Section 2.3.2). For each dataset a single dictionary of visual words

is learned for all classes. The number of visual words varies between 600 and 8,000 for the Caltech-101 data and between 4,000 and 25,000 for the PASCAL VOC data, with a set of larger vocabulary sizes employed for the VOC dataset as this was found to be important to deal with the more challenging images it contained.

Kernel codebook encoding. This encoding is analogous to the histogram encoding, but uses soft quantization. It requires a single additional parameter γ which is determined on the data validation split of the data as for the other parameters. The codeword uncertainty method of [129] is used. For efficiency, each feature is encoded by considering only the top five nearest visual words.

LLC encoding. Similarly to the kernel codebook encoding, this encoding requires setting the number of neighbour visual words M considered for each encoded descriptor. This is again set to five. The parameter β in the computation of the projections is set to 10^{-4}.

Improved Fisher encoding. As suggested in [99], we used GMM with $K = 256$ components after reducing the dimensionality of the SIFT descriptors to 80 by using PCA (we found this to outperform slightly the reduction to 64-D suggested by [99]). PCA was found to improve the accuracy and decrease the memory footprint of this representation. Hellinger's kernel is also used with this encoding, which amounts to square rooting the features (up to rectification) and then ℓ^2-normalisnormalisinging the result. This was also the method used in [99] and is the main difference between the original Fisher encoding formulation and the 'improved' Fisher encoding.

Super vector encoding. Since [143] reports very similar performance for codebooks of size 1024 and 2048, we chose the former size to reduce computation and memory consumption, which still remained very high. In our experiments, PCA reduction to 80 dimensions degraded the performance considerably, and we therefore report the results for the original SIFT descriptors. The memory required for storing super-vectors is particularly high for Caltech-101 dataset (due to larger number of spatial bins – 21 versus 8, refer to Section 3.2.1); therefore, only the performance on PASCAL VOC 2007 is reported.

We used the soft-assignment variant of super vector encoding with 5 (approximate) nearest neighbours used to encode each descriptor. At least with our implementation of soft assignment, the method turned out to be very sensitive to the number of nearest neighbours, with performance for 20 nearest neighbours and 1 nearest neighbour (*i.e.*, hard-assignment) being much worse (-5% and -7% mAP respectively) than for 5 nearest neighbours. It is possible that since the encoding captures lower-order differences compared to the Fisher encoding (0th and 1st order rather than 1st and 2nd order in the case of Fisher encoding), as with LLC some degree of locality in encoding is beneficial explaining this drop when a larger number of nearest neighbours are used.

The accuracy achieved by the super vector encoding in our experiments (Table 3.1) is competitive with other methods (for example, it compares favourably to the 58.3% mAP reported in [99], albeit at a finer dense SIFT sampling of every 3 pixels compared to the 16 pixel spacing used in that paper, or the 59.3% mAP reported in [133]), yet still quite far from the performance (64% mAP) reported in [143]. However, as we learnt from personal communication with the authors, [143] used nontrivial modifications not

discussed in the paper to achieve those results (these include using LDA to compute the SVM kernel and second order information as in the Fisher encoding). According to the authors, the performance achieved with our implementation is representative of their method, given that we did not apply these additional modifications.

3.2.3 Sources of Variability and Statistical Significance

When undertaking a comparison of different methods, it is very important to ensure that we can say something about the confidence of our results, and the sources of variability that might cause the result for an experiment to vary. This is an area in which the vision community has often been lacking, but nonetheless is critical to allow us to judge whether a given difference in performance between two methods is statistically significant or not.

The predominant **source of variability** in the experiments in this chapter is the generation of codebooks, which has a stochastic element on the basis of the random initial codeword centres used to initialise the training of the codebook/gaussian mixture model. Although there are many other moving parts, all other components of the pipeline are deterministic as given a codebook and an image each method will always return the same encoding. In order to determine the effect of this on the results, three different codebooks were trained for both the histogram encoding method and for the fisher encoding methods, each time using different codeword centre initialisations over the training set of the PASCAL VOC 2007 dataset. Following this, evaluation over the test set was performed, and in each case the difference between the best and worst result in mAP was less than 0.05%. Given that the histogram encoding method in particular

is the method most likely to be susceptible to changes in the codebook (as it is least equipped to mitigate against quantisation error) this can be taken as a lower-bound on the confidence of our results for all methods in Table 3.1.

Quantifying the **statistical significance** of the results over the PASCAL VOC 2007 dataset is more challenging, particularly given that in the PASCAL VOC 2007 there is only a single test split, making calculating a confidence interval over the entirety of the test data challenging. Nonetheless, there are methods from statistics based on cross-validation which can be used to calculate whether the difference between a pair of results is statistically significant or not, with the normal approach usually involving testing whether the *p-value* given the pair is greater than a constant set before starting the experiments, normally 0.05.

One way of computing the p-value is through a bootstrapping process as described in [36], which involves splitting the test set into multiple evaluation splits with replacement. For a single method we can obtain a bootstrap estimate of the confidence interval for a methods score by running a large number of bootstrap replicates, sorting the resulting scores, and then returning the $\alpha/2$ and $1 - \alpha/2$ quantiles, where for example $\alpha = 0.05$ would yield a 95% confidence interval.

To compare two methods A and B, we first compute the difference in score for each method on each bootstrap sample and then use the percentile bootstrap to estimate a confidence interval, with a null hypothesis that A is equivalent to B (at the 1α level). The null hypothesis (*i.e.* that the results are equivalent) is rejected if zero is not contained in the confidence interval, leading to the conclusion that method A is statistically significantly better than method B, or vice versa, depending on the result.

Examples of confidence intervals computed in this way have been relatively recently been made available over the latest version of the PASCAL VOC 2012 dataset [37] on the evaluation server.[1] Unfortunately, when the experiments undertaken in the next section were undertaken, per-image results were not retained, and so the approach outlined above is not employed in this chapter. It is nonetheless worth bearing this limitation of the study in mind when assessing the analysis of the following section.

3.3 Analysis

The results of our experiments over PASCAL VOC 2007 are shown in Table 3.1. As expected, all extensions to the baseline histogram encoding method result in some improvement to the classification accuracy. This concurs with the fact that information is lost when a local descriptor is replaced with (assigned to) the nearest visual word. A detailed analysis follows; we will use letters in square brackets to indicate the corresponding rows of Table 3.1.

For small codebook sizes, the kernel codebook encoding with χ^2 kernel performs better than the newer LLC-encoding ([α] vs [s]), while the baseline encoding with χ^2 kernel performs almost as well [y] (the performance degrades dramatically with linear kernel [z]). For larger codebook sizes, LLC encoding performs better than the baseline and the kernel codebook ([f] vs [j] vs [k]), but the difference is not large. However, it is interesting to note that LLC-encoding using the linear kernel achieves comparable performance to the χ^2 kernel across different vocabulary sizes and outperforms the results using the Hellinger's kernel feature map ([f] vs [g] and [h]; [s] vs [t] and [u]).

[1] `http://host.robots.ox.ac.uk:8080` – accessed on 1st March 2015

Method				mAP								
(a) FK	Lin	ss3	256	**61.7**	79.0	67.4	51.9	70.9	30.8	72.2	79.9	61.4
(b) SV	Lin	ss3	1024	**58.2**	74.3	63.8	47.0	69.4	29.1	66.5	77.3	60.2
(c) LLC	Lin	ss2	25k	**57.6**	71.1	62.9	47.4	67.7	25.2	62.7	77.0	59.6
(d) LLC-F	Lin	ss2	25k	**59.3**	74.1	64.9	51.5	68.3	27.2	62.9	78.4	61.4
(e) VQ	Chi	ss2	25k	**56.1**	70.0	58.9	42.9	66.8	26.6	62.3	75.7	57.1
(f) LLC	Lin	ss3	25k	**57.3**	71.4	62.7	46.1	69.0	26.0	63.9	77.0	59.7
(g) LLC	Sqr	ss3	25k	**56.7**	71.2	61.8	42.7	68.2	25.9	62.3	76.4	59.3
(h) LLC	Chi	ss3	25k	**57.7**	72.4	62.2	47.3	68.9	25.8	64.0	77.3	59.8
(i) LLC-F	Lin	ss3	25k	**59.7**	74.2	65.4	51.2	69.7	28.7	64.4	78.5	63.0
(j) VQ	Chi	ss3	25k	**55.3**	70.1	59.2	44.1	66.3	26.8	60.9	75.6	55.4
(k) KCB	Chi	ss3	25k	**56.3**	70.8	60.6	44.5	66.5	27.0	62.1	76.3	57.6
(l) LLC	Lin	ss5	25k	**57.0**	69.8	61.6	46.7	68.3	25.7	63.8	76.3	59.8
(m) LLC-F	Lin	ss5	25k	**58.7**	73.4	62.9	50.2	67.9	27.9	64.4	77.9	62.4
(n) VQ	Chi	ss5	25k	**53.9**	68.7	57.1	41.2	64.5	25.2	61.1	74.1	53.2
(o) LLC	Lin	ss3	14k	**56.2**	70.7	59.7	44.8	67.2	26.0	61.0	76.3	58.5
(p) VQ	Chi	ss3	14k	**54.8**	69.1	58.6	41.3	66.3	26.5	61.5	75.4	55.8
(q) LLC	Lin	ss3	10k	**56.0**	69.7	60.4	44.2	67.8	24.7	61.8	75.4	57.7
(r) VQ	Chi	ss3	10k	**55.0**	69.6	58.0	42.9	65.8	23.5	61.1	75.9	55.6
(s) LLC	Lin	ss3	4k	**53.8**	69.8	57.6	42.0	66.5	22.4	55.6	72.8	57.0
(t) LLC	Sqr	ss3	4k	**52.0**	68.5	54.6	40.1	65.3	21.5	51.9	71.5	55.2
(u) LLC	Chi	ss3	4k	**53.5**	70.2	56.2	42.7	65.3	22.2	55.2	72.8	57.0
(v) LLC-1	Lin	ss3	4k	**36.1**	53.4	43.2	22.5	46.3	11.4	29.5	64.7	45.4
(w) LLC-F	Lin	ss3	4k	**55.9**	72.3	61.4	44.1	67.9	25.0	57.9	75.4	59.4
(x) VQ	Sqr	ss3	4k	**52.0**	67.3	55.2	36.6	64.4	21.9	56.3	72.9	52.1
(y) VQ	Chi	ss3	4k	**53.4**	68.7	57.0	39.9	64.6	22.0	58.8	73.9	53.8
(z) VQ	Lin	ss3	4k	**46.5**	60.6	48.8	32.8	58.5	16.3	50.4	68.4	46.0
(α) KCB	Chi	ss3	4k	**54.6**	69.8	59.2	42.0	64.9	23.9	59.0	75.0	54.6

Table 3.1: Classification results over PASCAL VOC 2007 dataset *(continued on next page)* **VQ** – baseline method; **FK** – Fisher kernel; **SV** – super vector coding; **KCB** – kernel codebook; **LLC** – locally-constrained linear coding; **LLC-F** – LLC encoding with original+left-right flipped training images; **LLC-1** – L1-normalized LLC encoding; **Lin/Sqr/Chi** – linear/hellinger/χ^2 kernel map; third column: SIFT sampling density; fourth column: visual vocabulary size

												
(a) FK	56.0	49.6	58.4	44.8	78.8	70.8	85.0	31.7	51.0	56.4	80.2	57.5
(b) SV	50.2	46.5	51.9	44.1	77.9	67.1	83.1	27.6	48.5	51.1	75.5	52.3
(c) LLC	54.24	45.3	51.6	44.2	77.5	67.1	83.3	27.6	45.7	53.6	76.0	52.3
(d) LLC-F	54.4	47.2	52.8	44.6	78.1	68.5	83.7	29.9	51.0	55.5	78.6	53.6
(e) VQ	53.8	41.4	51.6	42.6	76.3	65.1	83.0	26.7	43.5	52.2	74.3	50.9
(f) LLC	54.0	46.3	52.1	42.4	77.2	67.2	83.4	23.1	44.5	52.1	75.4	52.2
(g) LLC	53.9	45.6	49.6	42.9	75.5	65.9	83.0	27.1	43.8	52.3	74.7	51.9
(h) LLC	54.3	46.0	51.1	43.2	76.7	67.1	83.5	27.7	44.8	52.8	76.0	52.5
(i) LLC-F	55.31	49.52	53.0	44.6	78.5	68.9	84.2	28.4	50.5	55.2	77.6	54.2
(j) VQ	53.1	40.4	51.3	39.2	76.9	63.1	82.2	21.0	43.0	51.4	75.1	50.9
(k) KCB	54.3	41.6	51.9	41.1	76.4	66.1	83.0	23.9	43.7	51.3	75.1	51.4
(l) LLC	53.2	44.9	51.0	40.7	76.9	66.8	82.8	26.2	45.5	52.2	75.1	52.0
(m) LLC-F	54.1	47.3	52.3	42.9	77.7	68.4	83.3	27.4	50.1	54.4	77.4	51.7
(n) VQ	52.1	39.0	48.7	38.6	74.8	62.4	81.2	25.1	40.5	49.4	72.2	48.2
(o) LLC	52.6	44.1	49.2	38.7	75.9	66.6	82.3	27.2	44.6	52.0	74.6	51.6
(p) VQ	52.7	40.2	50.6	38.1	75.3	63.9	82.2	21.4	42.2	51.7	74.1	49.6
(q) LLC	52.9	45.1	49.8	39.9	75.6	65.3	82.0	26.0	43.2	51.2	74.8	52.6
(r) VQ	53.0	41.1	49.6	41.3	76.3	64.2	81.9	21.5	41.9	53.0	74.1	49.5
(s) LLC	51.7	42.8	46.1	39.5	74.1	62.0	80.9	24.5	38.8	49.4	71.2	51.0
(t) LLC	50.8	40.1	42.6	39.5	71.7	58.6	79.7	22.9	39.0	47.8	70.9	49.1
(u) LLC	52.13	41.51	43.9	40.7	73.2	60.6	81.1	24.3	38.9	48.6	72.0	51.0
(v) LLC-1	32.8	19.1	21.3	22.7	53.3	33.6	68.7	11.4	6.2	38.0	60.4	37.3
(w) LLC-F	53.0	43.9	47.7	39.9	76.0	62.8	81.9	25.3	45.2	51.7	74.1	52.7
(x) VQ	51.5	38.2	46.5	35.0	74.6	60.7	80.1	18.8	37.1	50.2	71.7	48.3
(y) VQ	52.40	38.57	49.2	36.9	75.6	61.6	81.6	20.5	40.1	50.9	73.4	49.2
(z) VQ	48.3	28.3	41.7	30.2	70.5	55.2	77.8	14.8	31.4	39.7	68.3	42.7
(α) KCB	52.5	40.5	50.5	38.0	76.0	63.7	82.5	22.3	43.1	51.0	74.0	49.7

Table 3.1: Classification results over PASCAL VOC 2007 dataset *(continued from previous page)*

		codebook size					
Method		256	600	1500	2000	4000	8000
(a) FK	Lin	77.8 ± 0.6	–	–	–	–	–
(b) LLC	Lin	–	73.1 ± 1.1	74.8 ± 0.7	75.8 ± 0.7	76.2 ± 0.6	77.0 ± 0.4
(c) LLC	Chi	–	72.3 ± 1.1	74.2 ± 0.6	75.2 ± 0.7	76.0 ± 0.6	76.6 ± 0.6
(d) VQ	Chi	–	72.7 ± 0.8	73.6 ± 0.5	73.9 ± 0.8	74.4 ± 1.0	74.2 ± 0.7
(e) KCB	Chi	–	73.4 ± 0.7	75.2 ± 0.6	75.5 ± 0.7	75.9 ± 0.6	75.9 ± 0.6

Table 3.2: Classification results over Caltech-101 dataset (30 training images)

This suggests that the linear kernel is sufficient to achieve good performance with the encoding, avoiding the computational expense of applying a non-linear kernel – even through a more efficient feature map.

Our comparisons suggest that the Fisher encoding [a] and super-vector encoding [b] have an edge over other encoding methods, although for PASCAL the advantage is not as dramatic as portrayed in the respective papers (cf. *e.g.* [b] vs [j]). The Fisher encoding also is the best performing in the Caltech-101 experiment (Table 3.2). Thus it can be concluded that encoding the relative displacement between a descriptor and a codeword, as with both the Fisher encoding and super-vector encoding, successfully retains some extra information lost in the quantisation process.

Vocabulary size. The PASCAL experiments clearly demonstrate that larger vocabularies lead to higher accuracy. While the law of diminishing returns is clearly in place, it is most likely that the performance is not saturated even at 25K visual words ([s]-[q]-[o]-[f], [y]-[r]-[p]-[j]). A similar trend (although for a much smaller vocabulary size) is observed in Table 3.2. It should be noted that in the case of LLC encoding, even at a vocabulary size of 8,000 the performance achieved appears to be still increasing suggesting that further gains could be achieved by increasing the vocabulary size even further.

Sampling density/data augmentation. As has been observed in multiple previous works, the sampling density of the local descriptors matters, with denser sampling yielding higher accuracy ([f] vs [l]; [i] vs [m]; [j] vs [n]). The performance however saturates around the maximum sampling density that we considered here ([c] is better

than [f] but [d] is not better than [i]). Nonetheless, the performance can be further improved considerably via simple data augmentation tricks, such as adding left-right flipped versions of the training data to the training set ([c] vs [d], [f] vs [i], [l] vs [m]). Further data augmentation strategies are likely to improve the performance even more, and are indeed explored in the next chapter.

Size of encoding. Soft-assignment methods (LLC, kernel codebooks) gain considerably from large (*e.g.* 25,000) vocabularies resulting in a correspondingly large encoding size (number of words times number of spatial bins, *e.g.* 200K floats per image). Fisher kernel and super-vector encodings produce even larger vectors (*e.g.* with the encoding parameters suggested in [143], each VOC image is encoded with a 1.4-million dimensional dense vector). In our implementation, the size of the Fisher kernel encoding is smaller than super-vector encoding ($\sim$ 1.26MB vs $\sim$ 3.8MB) due to the smaller codebook employed and the use of PCA.

With our sampling density, neither of the encodings results in particularly sparse vectors: on PASCAL VOC, sparsity ranges from 30% non-zeros for the baseline encoding to close to 100% non-zeros for super-vector encoding and Fisher kernel encoding. Consequently, fitting the encoded training data in memory is not possible even for a relatively small datasets like PASCAL VOC 2007 and Caltech-101. In these cases computing the kernel matrix and solving the problem in the dual by using *e.g.* LIBSVM [22] reduces the memory footprint significantly, and this is the approach we use.

Speed of encoding. For the hard and soft-assignment based methods (LLC, kernel codebooks, the baseline) the encoding time is dominated by the approximate nearest

neighbour search, which increases sub-linearly yet considerably not only with the size of the vocabulary but also the number of nearest neighbours sought. We use the ANN algorithm described in [101] which has a reduced complexity per k-means iteration of $\mathcal{O}(Nlog(K))$ rather than $\mathcal{O}(NK)$ in the exact case, where N is the number of codewords and K is the number of nearest neighbours to search for. Nonetheless, given the scaling with K, the overhead of LLC and kernel codebook (KCB) encodings over the baseline are significant as a result of the additional time spent searching for K nearest neighbours per feature instead of just one. We also use a higher bound on the maximum number of comparisons per feature in our approximate nearest neighbour algorithm for the LLC and KCB encodings to maintain performance when searching for the greater number of nearest neighbours required by these methods (we use a maximum of 500 comparisons per feature for LLC and KCB, which in our experiments both use $K = 5$ nearest neighbours, compared to 25 comparisons for the baseline method, which requires just the first nearest neighbour) leading to an encoding time of 20 seconds per image for LLC and 30 seconds for KCB compared to 0.5 seconds for the baseline encoding. The time required to compute the approximate nearest neighbours comprises the majority of this – around 17 seconds for the 5-NN search for LLC and KCB compared to < 0.5 seconds for the 1-NN search in the case of the baseline encoding. The relative sluggishness of the KCB encoding can be explained by the fact our implementation is unoptimised MATLAB code compared to the C++ implementations used for the baseline and LLC encodings. All timings are for runs on a 3.07GHz Intel CPU using a vocabulary of 10K visual words.

Despite their size, super-vector encodings and Fisher kernel encodings can poten-

tially be faster (since they have to search neighbours/compute distances within a much smaller vocabulary). To achieve full speed, Fisher coding would probably benefit from some sparsification (*i.e.* $q_{ki} \approx 0$ should be set to 0), which we did not use. Super-vector encoding in our MATLAB implementation takes about 12 seconds per image and Fisher encoding about 9 seconds per image using a combined C++/MATLAB implementation.

It is noteworthy that for all encodings the cost of computing the SIFT descriptors is much smaller than the time required for any of the encodings (< 0.5 seconds per image).

3.4 Conclusion

In this chapter we have presented a detailed empirical evaluation of several recent encoding methods for bag-of-word models for object recognition. In some cases, our observations differed significantly from the one presented in the original publications, which emphasises the importance of controlling carefully all conditions when comparing different representations. To accompany the publication of this work, the source code used to produce all experiments was made publicly available [24].

Since the experiments outlined in this chapter were conducted, several developments to the encodings described within were proposed, such as the routine application of PCA whitening to dimensionality-reduced SIFT features prior to encoding using the Fisher vector [62], the application of the Hellinger kernel to SIFT features themselves [6], and indeed the routine application of the Hellinger kernel map as a

post-processing step for all the encoding methods described in this chapter as a way to reduce 'burstiness' in the feature representation [99]. These best practices are adopted for the remainder of experiments in this thesis. In the years that have followed, and possibly encouraged by the results disseminated by the published version of this chapter, the Fisher encoding and it's non-probabilistic variant VLAD [67] (covered in the next chapter) have become the most widely adopted encodings within the community for their balance of excellent performance and speed on the object category retrieval task.

However, much of the work on 'shallow' encoding methods for object category retrieval has been superseded in a way by the re-introduction of ConvNet based 'deep' methods by Krizhevsky *et al.* [71] in 2012. We turn our attention to such methods and how they compare to the encodings described here in the next chapter.

Chapter 4

Deep Features for Image Classification

In this chapter we conduct a comprehensive evaluation of deep image representations for object category classification and retrieval, naturally following on from our evaluation of 'shallow' encoding methods in the previous chapter and building on the best-practices both developed there and also within the community since the work of the previous chapter was undertaken.

The pipeline used with classical encoding methods such as those described in the previous chapter was relatively simple, with its power coming from finely-tuned local features and non-linear encoding functions, developed to optimally capture and summarise discriminative data from an image through human insight and engineering effort. Despite this, these handcrafted approaches are now substantially outperformed by the latest generation of *Convolutional Neural Networks* (CNNs) [74], reintroduced to the field over the past few years. These networks have a substantially more sophisti-

cated structure than standard representations, comprising several layers of non-linear feature extractors. Furthermore, while their structure is handcrafted, internally they contain a very large number of parameters which allows them to learn the appropriate 'feature extractor' from training data automatically. When applied to standard image classification and object detection benchmark datasets such as ImageNet ILSVRC [34] and PASCAL VOC [37] such networks have demonstrated excellent performance [35, 49, 93, 106, 115], surpassing the best published results for the 'shallow' methods described in the previous chapter.

Despite these impressive results, it remains unclear how different deep architectures compare to each other and to shallow computer vision methods such as the improved Fisher encoding (IFV). Most papers published at the time the work outlined in this chapter was undertaken did not test these representations extensively on a common ground, so a systematic evaluation of the effect of different design and implementation choices remained largely missing. As noted in Chapter 3, in our evaluation of shallow feature encodings, the performance of computer vision systems depends significantly on implementation details. For example, state-of-the-art methods such as [71] not only involve the use of a CNN, but also include other improvements such as the use of very large scale datasets, GPU computation, and data augmentation (also known as data jittering or virtual sampling). These improvements could also transfer to shallow representations such as the IFV, potentially explaining a part of the performance gap [95].

In this chapter we analyse and empirically clarify these issues, conducting a large set of rigorous experiments comparing shallow methods to deep methods, and analysing

which aspects of these new deep methods are important for performance. In many ways, just as the previous chapter was about the comparing and clarifying important details to get right when using state-of-the-art shallow encoding methods, so this chapter is about the same for deep encoding methods. Three scenarios are considered (Section 4.1, Section 4.2): shallow image representations, deep representations pre-trained on outside data, and deep representation pre-trained and then fine-tuned on the target dataset.

As part of our tests, we explore generally-applicable best practices that are nevertheless more often found in combination with CNNs [71] or, alternatively, with shallow encoders [26], porting them with mutual benefit. These are (Section 4.1): the use of *colour information*, feature normalisation, and, most importantly, the use of substantial data augmentation. We also develop scenario-specific best-practices further, continuing the exploration begun in Chapter 3 and also since in other papers [97], including dimensionality reduction for deep features. For example, we evaluate the effect of spatially extending SIFT descriptors with local feature coordinates rather than using spatial pyramids when encoding using IFV [111] and the effect of choosing different network structures, and in particular network sizes, for the CNN-based methods. Finally, by focusing on getting the details right, we were able to achieve performance competitive with the state of the art on the PASCAL VOC classification task [94, 135] at the time the work in this chapter was undertaken, using less additional training data than in other papers and significantly simpler techniques.

4.1 Scenarios

This section introduces the three types of image representation $\phi(\mathbf{I})$ considered in this chapter, describing them within the context of three different scenarios. Having outlined details specific to each, general methodologies which apply to all three scenarios are reviewed, such as data augmentation and feature normalisation, together with the linear classifier (trained with a standard hinge loss). In this evaluation, as in the previous chapter, we use both the PASCAL VOC 2007 dataset and the Caltech-101 dataset for evaluation using the standard experimental setup for these datasets as described in Section 2.5. In addition, we also present results over three additional datasets: the ILSVRC-2012 dataset, the PASCAL VOC 2012 dataset and the Caltech-256 dataset in order to provide a more comprehensive comparison to state-of-the-art and also demonstrate how the, particularly the CNN-based methods, perform over larger scale datasets also. These are also described in Section 2.5.

Scenario 1: Shallow representation (IFV) As a baseline shallow image representation, we use the same Fisher encoding features as described in Chapter 3. Here, we re-term them as the 'Improved Fisher Vector' (IFV), as the set of best-practices described in the previous chapter, including applying ℓ^2 and power normalisation, were re-termed as such in Perronnin *et al.* [99]. It was selected since it was the best-performing shallow feature encoding method considered in the previous chapter, and so provides 'best breed' shallow encoding performance for us to compare to the new CNN-based deep features.

Scenario 2: Deep representation (CNN) with pre-training The deep representations considered in this chapter are inspired by the success of the CNN of Krizhevsky *et al*. [71]. As shown in [35, 139], the vector of activities $\phi_{\text{CNN}}(\mathbf{I})$ of the penultimate layer of a deep CNN, learnt on a large dataset such as ImageNet [34], can be used as a powerful image descriptor applicable to other datasets. Numerous CNN architectures that improve the previous state of the art obtained using shallow representations have been proposed, but choosing the best one remains an open question. Many are inspired by [71]: DeCAF [35, 49], Caffe [68], Oquab *et al*. [93]. Others use larger networks with a smaller stride of the first convolutional layer: Zeiler and Fergus [139] and OverFeat [106, 115]. Other differences include the CNN pre-training protocols. Here we adopt a single learning framework and experiment with architectures of different complexity exploring their performance-speed trade-off.

Scenario 3: Deep representation (CNN) with pre-training and fine-tuning In Scenario 2 features are trained on one (large) dataset and applied to another (usually smaller). However, it was demonstrated [49] that fine-tuning a pre-trained CNN on the target data can significantly improve the performance. We consider this scenario separately from that of Scenario 2, as the image features become dataset-specific after the fine-tuning.

4.1.1 Commonalities

We now turn to what is in common across the scenarios.

Data augmentation Data augmentation is a method applicable to shallow and deep representations, but that has been so far mostly applied to the latter [71, 139]. By augmentation we mean perturbing an image $\mathbf{I}$ by transformations that leave the underlying class unchanged (*e.g.* cropping and flipping) in order to generate additional examples of the class. Augmentation can be applied at training time, at test time, or both. The augmented samples can either be taken as-is for use during training or can be combined to form a single feature, *e.g.* using sum/max-pooling or stacking.

Linear predictors Just as in Chapter 3, all representations are used to train a linear SVM for each class to be recognised. The parameter C in the SVM, trading-off regulariser and loss, is once again determined using an held-off validation subset of the data. Once again, we ensure that all feature vectors are ℓ^2 normalised before being fed into the SVM, for the reasons described in Section 3.1.3.

4.2 Details

This section gives the implementation details of the methods introduced in Sect. 4.1.

4.2.1 Improved Fisher Vector details

Our IFV representation uses a slightly improved setting compared to the features used in Chapter 3. Computation starts by upscaling the image $\mathbf{I}$ by a factor of 2 [111], followed by SIFT features extraction with a stride of 3 pixels at 7 different scales with $\sqrt{2}$ scale increments. These features are square-rooted as suggested by [6], and decorrelated and reduced in dimension from 128D to 80D using PCA. A GMM with

$K = 256$ components is learnt from features sampled from the training images. Hence the Fisher vector $\phi_{\text{FV}}(I)$ has dimension $2KD = 40,960$. Before use in classification, the vector is signed-square-rooted and ℓ^2-normalised (square rooting correspond to the Hellinger's kernel map [132]). Square-rooting is applied twice, once to the raw encodings, and once again after sum pooling and normalisation. In order to capture weak geometrical information, the IFV representation is used in a *spatial pyramid* [73]. As in Chapter 3, the image is divided into 1×1, 3×1, and 2×2 spatial subdivisions and corresponding IFVs are computed and stacked with an overall dimension of $8 \times 2KD = 327,680$ elements.

In addition to this standard formulation, we experiment with a few modifications. The first one is the use of *intra-normalisation* of the descriptor blocks, an idea recently proposed for the VLAD descriptor [7]. In this case, the ℓ^2 normalisation is applied to the individual sub-blocks $(\mathbf{u}_k, \mathbf{v}_k)$ of the vector $\phi_{\text{FV}}(\mathbf{I})$, which helps to alleviate the local feature burstiness [64]. In the case of the improved intra-normalised features, it was found that applying the square-rooting only once to the final encoding produced the best results (rather than to both the sub-blocks and the full encoding). We also provide results for the intra-normalised VLAD descriptor for the purposes of comparison, using a doubled vocabulary size of $K = 512$ centres as VLAD $\phi_{\text{VLAD}}(I)$ has dimension KD instead of $2KD$ for the Fisher encoding.

The second modification is the use of *spatially-extended local descriptors* [111] instead of a spatial pyramid. Here descriptors $\mathbf{x}_i$ are appended with their image location (x_i, y_i) before quantization with the GMM. Formally, $\mathbf{x}_i$ is extended, after PCA projection, with its normalised spatial coordinates: $[\mathbf{x}_i^\top, x_i/W - 0.5, y_i/H - 0.5]^\top$, where

$W \times H$ are the dimensions of the image. Since the GMM quantises both appearance and location, this allows for spatial information to be captured directly by the soft-quantization process. This method is significantly more memory-efficient than using a spatial pyramid. Specifically, the PCA-reduced SIFT features are spatially augmented by appending (x, y) yielding $D = 82$ dimensional descriptors pooled in a $2KD = 41,984$ dimensional IFV.

The third modification is the use of colour features in addition to SIFT descriptors. While colour information is used in CNNs [71] and by the original FV paper [99], it was not explored in our previous comparison [26]. We do so here by adopting the same Local Colour Statistics (LCS) features as used by [99]. LCS is computed by dividing an input patch into a 4×4 spatial grid (akin to SIFT), and computing the mean and variance of each of the *Lab* colour channels for each cell of the grid. The LCS dimensionality is thus $4 \times 4 \times 2 \times 3 = 96$. This is then encoded in a similar manner to SIFT. It should be noted that there may be other ways of incorporating colour information into the Fisher encoding (in particular, the LCS method used by [99] is not particularly invariant to illumination changes) but we stick with it here since this is the approach used by the original authors.

4.2.2 Convolutional neural networks details

The CNN-based features are based on three CNN architectures representative of the state of the art (shown in Table 4.1) each exploring a different accuracy/speed trade-off. To ensure a fair comparison between them, these networks are trained using the same training protocol and the same implementation, which we developed based on the

Arch.	conv1	conv2	conv3	conv4	conv5	full6	full7	full8
CNN-F	64x11x11 st. 4, pad 0 LRN, x2 pool	256x5x5 st. 1, pad 2 LRN, x2 pool	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 -	256x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-M	96x7x7 st. 2, pad 0 LRN, x2 pool	256x5x5 st. 2, pad 1 LRN, x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x2 pool	4096 drop- out	4096 drop- out	1000 soft- max
CNN-S	96x7x7 st. 2, pad 0 LRN, x3 pool	256x5x5 st. 1, pad 1 x2 pool	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 -	512x3x3 st. 1, pad 1 x3 pool	4096 drop- out	4096 drop- out	1000 soft- max

Table 4.1: **CNN architectures.** Each architecture contains 5 convolutional layers (conv 1–5) and three fully-connected layers (full 1–3). The details of each of the convolutional layers are given in three sub-rows: the first specifies the number of convolution filters and their receptive field size as "num x size x size"; the second indicates the convolution stride ("st.") and spatial padding ("pad"); the third indicates if Local Response Normalisation (LRN) [71] is applied, and the max-pooling downsampling factor. For full 1–3, we specify their dimensionality, which is the same for all three architectures. Full6 and full7 are regularised using dropout [71], while the last layer acts as a multi-way soft-max classifier. The activation function for all weight layers (except for full8) is the REctification Linear Unit (RELU) [71].

open-source Caffe framework [68].Full comparative results of the CNN architectures on which our networks are based are presented later in the results section in Table 4.3.

Our **Fast (CNN-F)** architecture is similar to the one used by Krizhevsky *et al*. [71]. It comprises 8 learnable layers, 5 of which are convolutional, and the last 3 are fully-connected. The input image size is 224×224. Fast processing is ensured by the 4 pixel stride in the first convolutional layer. The main differences between our architecture and that of [71] are the reduced number of convolutional layers and the dense connectivity between convolutional layers ([71] used sparse connections to enable training on two GPUs).

Our **Medium (CNN-M)** architecture is similar to the one used by Zeiler and Fergus [139]. It is characterised by the decreased stride and smaller receptive field of the first convolutional layer, which was shown to be beneficial on the ILSVRC dataset. At the same time, conv2 uses larger stride (2 instead of 1) to keep the computation time reasonable. The main difference between our net and that of [139] is we use less filters in the conv4 layer (512 *vs*. 1024).

Our **Slow (CNN-S)** architecture is related to the 'accurate' network from the OverFeat package [115]. It also uses 7×7 filters with stride 2 in conv1. Unlike CNN-M and [139], the stride in conv2 is smaller (1 pixel), but the max-pooling window in conv1 and conv5 is larger (3×3) to compensate for the increased spatial resolution. Compared to [115], we use 5 convolutional layers as in the previous architectures ([115] used 6), and less filters in conv5 (512 instead of 1024); we also incorporate an LRN layer after conv1 ([115] did not use contrast normalisation).

CNN training In general, our CNN training procedure follows that of [71], learning on ILSVRC-2012 using gradient descent with momentum. The hyper-parameters are the same as used by [71]: momentum 0.9; weight decay $5 \cdot 10^{-4}$; initial learning rate 10^{-2}, which is decreased by a factor of 10, when the validation error stops decreasing. The layers are initialised from a Gaussian distribution with a zero mean and variance equal to 10^{-2}. We also employ similar data augmentation in the form of random crops, horizontal flips, and RGB colour jittering. Test time crop sampling is discussed in Section 4.2.3; at training time, 224×224 crops are sampled randomly, rather than deterministically. Thus, the only notable difference to [71] is that the crops are taken from the whole training image $P \times 256, P \geq 256$, rather than its 256×256 centre. This results in a better coverage of images with a large aspect ratio. Training was performed on a single NVIDIA GTX Titan GPU and the training time varied from 5 days for CNN-F to 3 weeks for CNN-S.

CNN fine-tuning on the target dataset Given a pre-trained CNN, it can be fine-tuned on a target dataset by using it as a starting point for training. In our experiments, we fine-tuned CNN-S using VOC-2007, VOC-2012, or Caltech-101 as the target data. Fine-tuning was carried out using the same framework (and the same data augmentation), as we used for CNN training on ILSVRC. The last fully-connected layer (conv8) has output dimensionality equal to the number of classes (20 in the case of VOC datasets, 102 in the case of Caltech-101), so it could not be initialised with the corresponding layer of the ILSVRC net; instead, we initialised it from a Gaussian distribution (as used for CNN training above). Now we turn to dataset-specific fine-

tuning details.

VOC-2007 and VOC-2012. Considering that PASCAL VOC is a multi-label dataset (*i.e.* a single image might have multiple labels), we replaced the softmax regression loss with a more appropriate loss function, for which we considered two options: one-vs-rest classification hinge loss (the same loss as used in the SVM experiments) and ranking hinge loss. Both losses define constraints on the scores of positive ($\mathbf{I}_{pos}$) and negative ($\mathbf{I}_{neg}$) images for each class:

$$\mathbf{w}_c\phi(\mathbf{I}_{pos}) > 1 - \xi, \quad \mathbf{w}_c\phi(\mathbf{I}_{neg}) < -1 + \xi \qquad \textit{classification loss} \tag{4.1a}$$

$$\mathbf{w}_c\phi(\mathbf{I}_{pos}) > \mathbf{w}_c\phi(\mathbf{I}_{neg}) + 1 - \xi \qquad \textit{ranking loss} \tag{4.1b}$$

$\mathbf{w}_c$ is the c-th row of the last fully-connected layer, which can be seen as a linear classifier on deep features $\phi(\mathbf{I})$; ξ is a slack variable. We implemented both losses as the layers of our CNN training framework. Our fine-tuned networks are denoted as "CNN S TUNE-CLS" (for the classification loss) and "CNN S TUNE-RNK" (for the ranking loss). In the case of both VOC datasets, the training and validation subsets were combined to form a single training set. Given the smaller size of the training data when compared to ILSVRC-2012, we controlled for over-fitting by using lower initial learning rates for the fine-tuned hidden layers. The learning rate schedule for the last layer / hidden layers was: $10^{-2}/10^{-4} \rightarrow 10^{-3}/10^{-4} \rightarrow 10^{-4}/10^{-4} \rightarrow 10^{-5}/10^{-5}$, with the learning rate being advanced to the next values when the validation error stops decreasing as described in the previous section. The values of other hyper-parameters were kept the same as in ILSVRC-2012 training.

Caltech-101 dataset contains a single class label per image, so fine-tuning was performed using the softmax regression loss. Other settings (including the learning rate schedule) were the same as used for the VOC fine-tuning experiments.

Low-dimensional CNN feature training Our baseline networks (Table 4.1) have the same dimensionality of the last hidden layer (full7): 4096. This design choice is in accordance with the state-of-the-art architectures [71, 115, 139], and leads to a 4096-D dimensional image representation, which is already rather compact compared to IFV. At the same time, for large-scale image recognition applications, even lower dimensionality can be required. To investigate the dependency of the performance on the feature dimensionality, we further trained three modifications of the CNN-M network, with lower dimensional full7 layers of: 2048, 1024, and 128 dimensions respectively. The networks were learnt on ILSVRC-2012. To speed-up training, all layers aside from full7/full8 were set to those of the CNN-M net and a lower initial learning rate of 10^{-3} was used. The initial learning rate of full7/full8 was set to 10^{-2}.

4.2.3 Data augmentation details

We explore three data augmentation strategies. The first strategy is to use **no augmentation**. The IFV representation can take images of any size as input, and therefore in this case the image is processed untransformed. In contrast to IFV, however, CNNs require images to be transformed to a fixed size (224×224) even when no augmentation is used. Hence the image is downsized so that the smallest dimension is equal to 224 pixels and a 224×224 crop is extracted from the centre.[1] The second strategy is to use

[1]Extracting a 224×224 centre crop from a 256×256 image [71] resulted in worse performance.

flip augmentation, mirroring images about the y-axis producing two samples from each image. The third strategy, termed **C+F augmentation**, combines cropping and flipping. For CNN-based representations, the image is downsized so that the smallest dimension is equal to 256 pixels. Then 224×224 crops are extracted from the four corners and the centre of the image. Note that the crops are sampled from the whole image, rather than its 256×256 centre, as done by [71]. These crops are then flipped about the y-axis, producing 10 perturbed samples per input image. In the case of the IFV encoding, the same crops are extracted, but at the original image resolution.

4.3 Analysis

This section describes the experimental results, comparing different features and data augmentation schemes. The results are given in Table 4.2 for VOC 2007 and analysed next, starting from generally applicable methods such as augmentation and then discussing the specifics of each scenario. We then move onto other datasets and the state of the art in Section 4.3.

Data augmentation We experiment with no data augmentation (denoted *Image Aug=–* in Tab. 4.2), flip augmentation (*Image Aug=F*), and C+F augmentation (*Image Aug=C*). Augmented images are used as stand-alone samples (f), or by fusing the corresponding descriptors using sum (s) or max (m) pooling or stacking (t). So for example *Image Aug=(C) f s* in row [g] of Tab. 4.2 means that C+F augmentation is used to generate additional samples in training (f), and is combined with sum-pooling in testing (s).

Method	SPool	Image	Aug.		Dim	mAP						
(I) IFV BL	spm	–			327K	**61.69**	79.0	67.4	51.9	70.9	30.8	72.2
(II) DECAF	–	(C)	t	t	327K	**73.41**	87.4	79.3	84.1	78.4	42.3	73.7
(a) VLAD IN	spm	–			327K	**61.67**	80.7	67.4	54.6	72.9	32.8	71.4
(b) IFV	spm	–			327K	**63.66**	83.4	68.8	59.6	74.1	35.7	71.2
(c) IFV IN	spm	–			327K	**64.18**	82.1	69.7	59.7	75.2	35.7	71.3
(d) IFV	(x,y)	–			42K	**63.51**	83.2	69.4	60.6	73.9	36.3	68.6
(e) IFV IN	(x,y)	–			42K	**64.36**	83.1	70.4	62.4	75.2	37.1	69.1
(f) IFV IN	(x,y)	(F)	f	-	42K	**64.35**	83.1	70.5	62.3	75.4	37.1	69.1
(g) IFV IN	(x,y)	(C)	f	s	42K	**67.17**	85.5	71.6	64.6	77.2	39.0	70.8
(h) IFV IN	(x,y)	(C)	s	s	42K	**66.68**	84.9	70.1	64.7	76.3	39.2	69.8
(i) IFV IN 512	(x,y)	–			84K	**65.36**	84.1	70.4	65.0	76.7	37.2	71.3
(j) IFV IN 512	(x,y)	(C)	f	s	84K	**68.02**	85.9	71.8	67.1	77.1	38.8	72.3
(k) IFV IN COL 512	–	–			82K	**52.18**	69.5	52.1	47.5	64.0	24.6	49.8
(l) IFV IN 512 COL+	(x,y)	–			166K	**66.37**	82.9	70.1	67.0	77.0	36.1	70.0
(m) IFV IN 512 COL+	(x,y)	(C)	f	s	166K	**67.93**	85.1	70.5	67.5	77.4	35.7	71.2
(n) CNN F	–	(C)	f	s	4K	**77.38**	88.7	83.9	87.0	84.7	46.9	77.5
(o) CNN S	–	(C)	f	s	4K	**79.74**	90.7	85.7	88.9	86.6	50.5	80.1
(p) CNN M	–	–			4K	**76.97**	89.5	84.3	88.8	83.2	48.4	77.0
(q) CNN M	–	(C)	f	s	4K	**79.89**	91.7	85.4	89.5	86.6	51.6	79.3
(r) CNN M	–	(C)	f	m	4K	**79.50**	90.9	84.6	89.4	85.8	50.3	78.4
(s) CNN M	–	(C)	s	s	4K	**79.44**	91.4	85.2	89.1	86.1	52.1	78.0
(t) CNN M	–	(C)	t	t	4K	**78.77**	90.7	85.0	89.2	85.8	51.0	77.8
(u) CNN M	–	(C)	f	-	4K	**77.78**	90.5	84.3	88.8	84.5	47.9	78.0
(v) CNN M	–	(F)	f	-	4K	**76.99**	90.1	84.2	89.0	83.5	48.1	77.2
(w) CNN M GS	–	–			4K	**73.59**	87.4	80.8	82.4	82.1	44.5	73.5
(x) CNN M GS	–	(C)	f	s	4K	**77.00**	89.4	83.8	85.1	84.4	49.4	77.6
(y) CNN M 2048	–	(C)	f	s	2K	**80.10**	91.3	85.8	89.9	86.7	52.4	79.7
(z) CNN M 1024	–	(C)	f	s	1K	**79.91**	91.4	86.9	89.3	85.8	53.3	79.8
(α) CNN M 128	–	(C)	f	s	128	**78.60**	91.3	83.9	89.2	86.9	52.1	81.0
(β) IFV+CNN F	(x,y)	(C)	f	s	88K	**77.95**	89.6	83.1	87.1	84.5	48.0	79.4
(γ) IFV+CNN M 2048	(x,y)	(C)	f	s	86K	**80.14**	90.9	85.9	88.8	85.5	52.3	81.4
(δ) CNN S TUNE-RNK	–	(C)	f	s	4K	**82.42**	95.3	90.4	92.5	89.6	54.4	81.9

Table 4.2: VOC 2007 results *(continued overleaf)*. See Sect. 4.3 for details.

														
(I)	79.9	61.4	56.0	49.6	58.4	44.8	78.8	70.8	85.0	31.7	51.0	56.4	80.2	57.5
(II)	83.7	83.7	54.3	61.9	70.2	79.5	85.3	77.2	90.9	51.1	73.8	57.0	86.4	68.0
(a)	79.3	65.2	52.6	49.9	54.6	48.2	79.2	66.4	85.0	35.0	49.9	52.7	79.8	55.9
(b)	80.7	64.4	53.8	53.8	60.2	47.8	79.9	68.9	86.1	37.3	51.1	55.8	83.7	56.9
(c)	80.6	64.8	53.9	54.9	60.7	50.5	80.4	69.5	86.2	38.3	54.4	56.3	82.7	56.7
(d)	81.1	64.2	51.1	53.4	61.9	50.0	80.0	67.5	85.3	35.7	51.9	53.8	83.5	58.9
(e)	80.5	66.9	50.9	53.9	62.1	51.5	80.5	68.5	85.9	37.2	55.2	54.3	83.3	59.2
(f)	80.5	66.8	51.0	54.1	62.2	51.5	80.4	68.2	86.0	37.3	55.1	54.2	83.3	59.2
(g)	82.4	71.6	52.8	62.4	63.4	57.1	81.6	70.9	86.9	41.2	61.2	56.9	85.2	61.5
(h)	81.9	71.0	52.8	61.6	62.2	56.8	81.8	70.0	86.5	41.5	61.0	56.5	84.3	60.9
(i)	81.1	67.9	52.6	55.4	61.4	51.2	80.5	69.1	86.4	41.2	56.0	56.2	83.7	59.9
(j)	82.5	73.2	54.7	62.7	64.5	56.6	82.2	71.3	87.5	43.0	62.0	59.3	85.7	62.4
(k)	66.1	46.6	42.5	35.8	41.1	45.5	75.4	58.3	83.9	39.8	47.3	35.6	69.2	49.0
(l)	80.0	65.9	52.8	56.1	61.0	56.9	81.4	69.6	88.4	49.0	59.2	56.4	84.7	62.8
(m)	81.6	70.8	52.9	59.6	63.1	59.9	82.1	70.5	88.9	50.6	63.7	57.5	86.1	64.1
(n)	86.3	85.4	58.6	71.0	72.6	82.0	87.9	80.7	91.8	58.5	77.4	66.3	89.1	71.3
(o)	87.8	88.3	61.3	74.8	74.7	87.2	89.0	83.7	92.3	58.8	80.5	69.4	90.5	74.0
(p)	85.1	87.4	58.1	70.4	73.1	83.5	85.5	80.9	90.8	54.1	78.9	61.1	89.0	70.4
(q)	87.7	88.6	60.3	80.1	74.4	85.9	88.2	84.6	92.1	60.3	80.5	66.2	91.3	73.5
(r)	87.6	88.6	60.7	78.2	73.6	86.0	87.4	83.8	92.3	59.3	81.0	66.8	91.3	74.0
(s)	87.5	88.1	60.4	76.9	74.8	85.8	88.1	84.3	92.2	59.5	79.3	65.8	90.8	73.5
(t)	87.3	87.6	60.1	72.3	75.3	85.2	86.9	82.6	91.9	58.5	77.9	66.5	90.5	73.4
(u)	85.7	87.9	58.3	74.2	73.9	84.7	86.6	82.0	91.0	55.8	79.2	62.1	89.3	71.0
(v)	85.3	87.3	58.1	70.0	73.4	83.5	86.0	80.8	90.9	53.9	78.1	61.2	88.8	70.6
(w)	85.0	84.9	57.8	65.9	69.8	79.5	82.9	77.4	89.2	42.8	71.7	60.2	86.3	67.8
(x)	87.2	86.5	59.5	72.4	74.1	81.7	86.0	82.3	90.8	48.9	73.7	66.8	89.6	71.0
(y)	87.6	88.4	60.2	76.9	75.4	85.5	88.0	83.4	92.1	61.1	83.1	68.5	91.9	74.2
(z)	87.8	88.6	59.0	77.2	73.1	85.9	88.3	83.5	91.8	59.9	81.4	68.3	93.0	74.1
(α)	86.6	87.5	59.1	70.0	72.9	84.6	86.7	83.6	89.4	57.0	81.5	64.8	90.4	73.4
(β)	86.8	85.6	59.9	72.0	73.4	81.4	88.6	80.5	92.1	60.6	77.3	66.4	89.3	73.3
(γ)	87.7	88.4	61.2	76.9	76.6	84.9	89.1	82.9	92.4	61.9	80.9	68.7	91.5	75.1
(δ)	91.5	91.9	64.1	76.3	74.9	89.7	92.2	86.9	95.2	60.7	82.9	68.0	95.5	74.4

Table 4.2: VOC 2007 results *(continued from previous page)*

Augmentation consistently improves performance by $\sim 3\%$ for both IFV (*e.g.* [e] *vs.* [g]) and CNN (*e.g.* [p] *vs.* [q]). Using additional samples for training and sum-pooling for testing works best ([q]) followed by sum-pooling [s], max pooling [r], and stacking [t]. In terms of the choice of transformations, flipping improves only marginally ([p] *vs.* [v]), but using the more expensive C+F sampling improves, as seen, by about $2 \sim 3\%$ ([p] *vs.* [q]). We experimented with sampling more transformations, taking a higher density of crops from the centre of the image (*e.g.* sampling from a 4×4 or 5×5 grid of sub-crops evenly spaced across the image rather than a 3×3 one), but this did not result in any significant increase performance and so did not explore this avenue further.

Colour Colour information can be added and subtracted in CNN and IFV. In IFV replacing SIFT with the colour descriptors of [99] (denoted *COL* in *Method*) yields significantly worse performance ([k] *vs.* [i]). However, when SIFT and colour descriptors are combined by stacking the corresponding IFVs (*COL+*) there is a small but significant improvement of around $\sim 1\%$ in the non-augmented case (*e.g.* [i] *vs.* [l]) but little impact in the augmented case (*e.g.* [j] *vs.* [m]). For CNNs, retraining the network after converting all the input images to grayscale (denoted *GS* in *Methods*) has a more significant impact, resulting in a performance drop of $\sim 3\%$ ([x] *vs.* [q], [w] *vs.* [p]).

Scenario 1: Shallow representation (IFV) The baseline IFV encoding using a spatial pyramid [b] performs slightly better than the results [I] taken from the previous chapter, primarily due to a larger number of spatial scales being used during SIFT

feature extraction, and the resultant SIFT features being square-rooted. It also has a slight edge over the VLAD representation ([a] *vs.* [b]). *Intra-normalisation*, denoted as *IN* in the *Method* column of the table, improves the performance by $\sim 1\%$ (*e.g.* [d] *vs.* [e]). More interestingly, switching from spatial pooling (denoted *spm* in the *SPool* column) to feature spatial augmentation (*SPool=(x,y)*) has either little effect on the performance or results in a marginal increase ([b] *vs.* [d], [c] *vs.* [e]), whilst resulting in a representation which is over $10\times$ smaller. We also experimented with augmenting with scale in addition to position as in [111] but observed no improvement. Finally, we investigate pushing the parameters of the representation setting $K = 512$ (rows [i]-[m]). Increasing the number of GMM centres in the model from $K = 256$ to 512 results in a further performance increase (e.g. [i] *vs.* [e]), but at the expense of higher-dimensional codes (125K dimensional).

Scenario 2: Deep representation (CNN) with pre-training CNN-based methods consistently outperform the shallow encodings, even after the improvements discussed above, by a large $\sim 10\%$ mAP margin ([j] *vs.* [q]). Our small architecture CNN-F, which is similar to DeCAF [35], performs significantly better than the latter ([II] *vs.* [t]), providing a solid baseline for our other architectures and methods to build upon. Both medium CNN-M [n] and slow CNN-S [q] outperform the fast CNN-F [n] by a significant $2 \sim 3\%$ margin. Since the accuracy of CNN-S and CNN-M is nearly the same, we focus on the latter as it is simpler and marginally ($\sim 25\%$) faster. Remarkably, these good networks work very well even with no augmentation [p]. Another advantage of CNNs compared to IFV is the small dimensionality of the output features,

although IFV can be compressed to an extent. We explored retraining the CNNs such that the final layer was of a lower dimensionality, and reducing from 4096 to 2048 actually resulted in a marginal performance boost ([y] *vs.* [q]). What is surprising is that we can reduce the output dimensionality further to 1024D [z] and even 128D [α] with only a drop of $\sim 2\%$ for codes that are $32\times$ smaller ($\sim 650\times$ smaller than our best performing IFV [j]). Note, ℓ^2-normalising the features accounted for up to $\sim 5\%$ of their performance over VOC 2007; it should be applied before input to the SVM and after pooling the augmented descriptors (where applicable).

Scenario 3: Deep representation (CNN) with pre-training and fine-tuning We fine-tuned our CNN-S architecture on VOC-2007 using the ranking hinge loss, and achieved a significant improvement: 2.7% ([δ] *vs.* [o]). This demonstrates that in spite of the small amount of VOC training data (5,011 images), fine-tuning is able to adjust the learnt deep representation to better suit the dataset in question.

Combinations For the CNN-M 2048 representation [y], stacking deep and shallow representations to form a higher-dimensional descriptor makes little difference ([y] *vs.* [γ]). For the weaker CNN-F it results in a small boost of $\sim 0.8\%$ ([n] *vs.* [β]).

Comparison with the state of the art In Table 4.3 we report our results on ILSVRC-2012, VOC-2007, VOC-2012, Caltech-101, and Caltech-256 datasets, and compare them to the state of the art at the time the experiments in this chapter were undertaken. First, we note that the ILSVRC error rates of our CNN-F, CNN-M, and CNN-S networks are better than those reported by [71], [139], and [115] for the

	ILSVRC-2012 (top-5 error)	**VOC-2007** (mAP)	**VOC-2012** (mAP)	**Caltech-101** (accuracy)	**Caltech-256** (accuracy)
(a) IFV IN 512	-	68.0	–	–	–
(b) CNN F	16.7	77.4	79.9	–	–
(c) CNN M	13.7	79.9	82.5	87.15 ± 0.80	77.03 ± 0.46
(d) CNN M 2048	13.5	80.1	82.4	86.64 ± 0.53	76.88 ± 0.35
(e) CNN S	**13.1**	79.7	82.9	87.76 ± 0.66	**77.61 ± 0.12**
(f) CNN S TUNE-CLS	**13.1**	-	83.0	**88.35 ± 0.56**	77.33 ± 0.56
(g) CNN S TUNE-RNK	**13.1**	**82.4**	**83.2**	–	–
(h) Zeiler & Fergus [139]	16.1	-	79.0	86.5 ± 0.5	74.2 ± 0.3
(i) Razavian *et al.* [106, 115]	14.7	77.2	–	–	–
(j) Oquab *et al.* [93]	18	77.7	78.7 (82.8*)	–	–
(k) Oquab *et al.* [94]	-	-	**86.3***	–	–
(l) Wei *et al.* [135]	-	81.5 (**85.2***)	81.7 (**90.3***)	–	–
(m) He *et al.* [57]	13.6	80.1	-	**91.4 ± 0.7**	–

Table 4.3: **Comparison with the state of the art** on ILSVRC2012, VOC2007, VOC2012, Caltech-101, and Caltech-256. Results marked with * were achieved using models pre-trained on the *extended* ILSVRC datasets (1512 classes in [93, 94], 2000 classes in [135]). All other results were achieved using CNNs pre-trained on ILSVRC-2012 (1000 classes).

related configurations. As these networks are very similar in architecture to those proposed in the above publications, it is likely that the small boost is due to small details which might seem insignificant at first, such as the sampling of image crops from the uncropped image plane (instead of the centre). When using our CNN features on other datasets, the relative performance generally follows the same pattern as on ILSVRC, where the nets are trained – the CNN-F architecture exhibits the worst performance, with CNN-M and CNN-S performing considerably better.

Further fine-tuning of CNN-S on the VOC datasets turns out to be beneficial; on VOC-2012, using the ranking loss is marginally better than the classification loss ([g] *vs.* [f]), which can be explained by the ranking-based VOC evaluation criterion. Fine-tuning on Caltech-101 also yields a small improvement, but no gain is observed

over Caltech-256.

Our CNN-S net is competitive with recent CNN-based approaches [57, 93, 94, 106, 135, 139] and on a number of datasets (VOC-2007, VOC-2012, Caltech-101, Caltech-256) and sets the state of the art on VOC-2007 and VOC-2012 across methods pre-trained solely on ILSVRC-2012 dataset. While the CNN-based methods of [94, 135] achieve better performance on VOC (86.3% and 90.3% respectively), they were trained using extended ILSVRC datasets, enriched with additional categories semantically close to the ones in VOC. Additionally, [135] used a significantly more complex classification pipeline, driven by bounding box proposals [30], pre-trained on ILSVRC-2013 detection dataset. Their best reported result on VOC-2012 (90.3%) was achieved by the late fusion with a complex hand-crafted method of [136]; without fusion, they get 84.2%. On Caltech-101, [57] achieves the state of the art using spatial pyramid pooling of conv5 layer features, while we used full7 layer features consistently across all datasets (for full7 features, they report 87.08%).

In addition to achieving performance comparable to the state of the art with a very simple approach (but powerful CNN-based features), with the modifications outlined in this chapter (primarily the use of data augmentation similar to the CNN-based methods) we are able to improve the performance of shallow IFV to 68.02% (Table 4.2, [j]).

Timings and dimensionality One of our best-performing CNN representations CNN-M-2048 [y] is $\sim 42\times$ more compact than the best performing IFV [j] (84K vs. 2K) and CNN-M features are also $\sim 50\times$ faster to compute ($\sim 120s$ vs. $\sim 2.4s$ per im-

age with augmentation enabled, over a single CPU core). Non-augmented CNN-M features [p] take around $0.3s$ per image, compared to $\sim 0.4s$ for CNN-S features and $\sim 0.13s$ for CNN-F features.

4.4 Conclusion

In this chapter we conducted a rigorous empirical evaluation of CNN-based methods for image classification, along with a comparison with more traditional shallow feature encoding methods. It was shown that the performance of shallow representations can be significantly improved by adopting data augmentation, typically used in deep learning. In spite of this improvement, deep architectures still outperform the shallow methods by a large margin, suggesting that pre-trained CNN-based features will gradually replace the older shallow methods for many applications. We have shown that the performance of deep representations on the ILSVRC dataset is a good indicator of their performance on other datasets, and that fine-tuning can further improve on already very strong results achieved using the combination of deep representations and a linear SVM.

Now that we have conducted a thorough evaluation of visual features for object category retrieval, both deep in this chapter and shallow in Chapter 3, along with analysing the details of a 'best of breed' learning pipeline, we now have all the tools necessary to start considering how to build an on-the-fly category retrieval system. We will switch focus to this in the following chapter.

Part II

Applications

Chapter 5

On-the-fly Category Search

In this chapter, we shift our attention to the problem of on-the-fly visual search for object categories over large-scale unannotated datasets. This differs from the scenarios outlined in the previous two chapters, as we are primarily concerned with undertaking retrieval in a real-time manner. The focus of our investigation will be twofold: (i) in the first part of this chapter (Sections 5.1–5.3), we will investigate how we can adapt the state-of-the-art shallow image encoding methods introduced in Chapter 3 for use within a online setting where speed is critical and (ii) in the second part (Section 5.4), we will consider how such an accelerated retrieval pipeline can be used for on-the-fly learning, thus allowing visual category models to be trained on demand, and present a fully-functional object category search system which is capable of operating in real-time over datasets of millions of images.

5.1 Object Category Retrieval at Scale

In this section we investigate how we can adapt the shallow feature encodings described in Chapter 3 to operate in real-time. We begin by reviewing the standard object category retrieval pipeline in Section 5.1.1, considering the computational expense of each stage in turn. We will see how the most critical step to optimise in such a pipeline is the ranking stage, in particular its memory consumption, and consider both which shallow encodings are best suited to real-time search, and how they can be made more compact to facilitate fast ranking in Section 5.1.2. Following this, we present a novel cascade approach to speed up ranking further, making it sub-linear in dataset size in Section 5.1.3. A comprehensive evaluation of the methods proposed and their cost in terms of both speed and memory is then presented in Section 5.2.

5.1.1 Accelerating the Retrieval Pipeline

The state-of-the-art shallow object category classification/retrieval architecture described in Chapter 3, while providing excellent retrieval performance over standard benchmarks, is not optimised for speed. In order to scale such an architecture to operate over datasets of millions of images in an on-the-fly manner, this is clearly one of the key issues which must be addressed.

The basic architecture of an object category retrieval pipeline is shown in Figure 5.1. It can be split into three distinct stages, namely (i) encoding of a collection of positive and negative training images, (ii) training of a visual model and (iii) ranking over the target dataset. Of the above, the computation time of the first and second stages

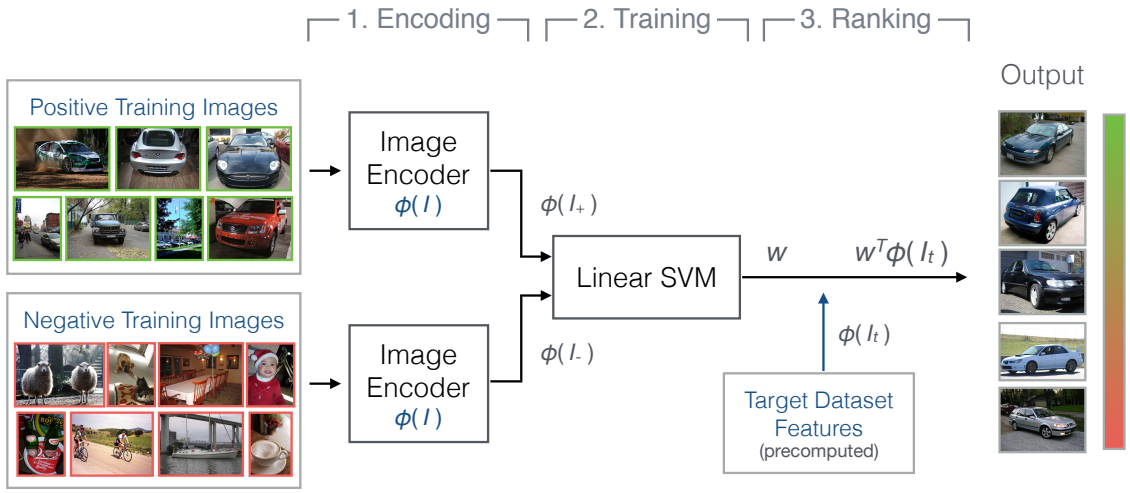


Figure 5.1: **Architecture of a typical object category retrieval pipeline.** Positive and negative training images can be either sourced from a separate training split of the target dataset, or from some other source such as Google Image search.

are fixed with training set size, so optimisation relies mainly on selecting the most computationally efficient encoding and learning algorithms. However, more critically, the ranking stage is also dependent on the size of our target data, tending to take longer the more images we have to rank. Given that our objective is to scale to datasets of millions of images, it is clear that this stage in particular must be made as efficient as possible, and so this is our focus.

In both cases, it is the choice of image representation, including any compression, has the largest impact on the total speed of the pipeline (given that learning using a linear SVM is already relatively fast) and so we explore the effect this choice has on both encoding and ranking times in the next section.

5.1.2 Faster Shallow Encodings

Starting with the encoding component of the retrieval pipeline (Figure 5.1), it is critical that computation of image codes is as fast as possible. Many of the encodings described

in Chapter 3 are already very quick to compute, but nonetheless there is some variation between methods, with for example methods based upon k-means vocabularies (histogram encoding, LLC, VLAD *etc.*) generally being faster than Fisher encoding, which uses a probabilistic GMM vocabulary. As a result, we focus on three different encodings in this chapter, namely: Fisher encoding (as it gave the best performance over standard benchmarks in Chapter 3), VLAD (as it also performs comparably to the Fisher encoding as shown in Chapter 4, and is faster to compute) and histogram encoding (as is simplest encoding formulation, and as a result also the fastest).

Although selecting an encoding that is fast to compute is sufficient to speed up the encoding stage, the ranking stage requires a little more attention. This is because although the Fisher encoding, VLAD and also the histogram encoding method are very powerful once computed, one disadvantage they have is that they are very high dimensional. This causes a particular problem where real-time operation is required, as in general we wish to fit the entirety of our target dataset into memory to ensure fast ranking (typically even the fastest SSDs generally have contiguous access speeds of 15 – 20$\times$ slower than RAM). Furthermore, it also makes the ranking process slower, since when evaluating the dot product of our linear model with the dataset $\mathbf{w}.\boldsymbol{\phi}$ the larger the encoding the more multiply operations are required (and the more data locations must be visited).

We therefore consider two strategies to reduce the memory footprint of our image representations: (i) the application of compression methods as described in Section 2.4.2, in particular Product Quantisation and (ii) the use of alternative attribute based binary codes, in particular PiCoDes [18]. Both of the resultant compact rep-

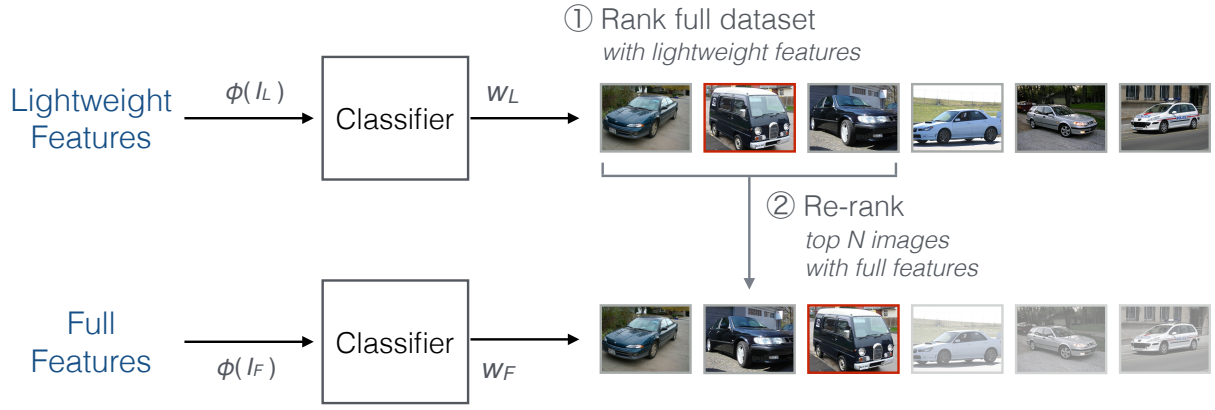


Figure 5.2: **Cascade Classifiers for Fast Ranking.** A set of both low-dimensional lightweight features and higher dimensional full features are computed for each query and used to train two separate visual models. Following this, the lightweight model is used to rank images from the entirety of the target dataset, with the full model then used to re-rank only the top N images from the coarse ranking.

resentations have the advantage that they can also be ranked very efficiently, using look-up tables in the case of the former and fast hamming distance computations in the case of the latter (refer to section 2.4.2 for further details).

5.1.3 Sub-linear Ranking using Cascades

In the case of instance retrieval, a common strategy to achieve sub-linear ranking is to use an inverted index [63, 90, 101]. However, this relies on both the query vector (the $\mathbf{w}$ vector from the linear SVM in our case) and the test vectors $\boldsymbol{\phi}$ being sparse, which is not the case with state-of-the-art shallow image encodings (*e.g.* typically a Fisher-encoded image is over 60% non-zeros). We therefore explore an alternative novel approach to reduce the ranking time so that it scales sub-linearly using a *cascade of classifiers*.

The basic idea is illustrated in Figure 5.2, and consists of computing two different sets of features for a given dataset. The first is a lightweight and simplified low-

dimensional representation (*e.g.* an encoding generated using a codebook with a lower number of centres, or without spatial pooling) which is used to rank the entirety of the target dataset. Following this, the top N images effectively act as a 'shortlist', and can then be re-ranked using a richer, high-dimensional representation (which would take too much time to apply to the entire dataset). Although the use of cascades is not uncommon in the object detection literature *e.g.* [52], 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.

5.2 Evaluating a Real-time System

Our objective is to assess both the shallow encoding and compression methods, along with our cascade approach for ranking, in a way that mirrors our intended use case as closely as possible, that is to say for real-time visual search. In such a setting, measuring the 'goodness' of the first few pages of retrieved results is critical. We therefore evaluate using precision @ K, which is a standard retrieval metric which can be used for this purpose, reflecting that the larger the proportion of true positives for a given object category at the top of a ranked list the better the perceived performance. In our experiments, we typically use $K = 100$.

5.2.1 Datasets

One difficulty of evaluating a large-scale object category retrieval system is the lack of large-scale datasets with sufficient annotation to assess retrieval performance fully.

The PASCAL VOC dataset [37] provides full annotation for a set of twenty common object classes, facilitating evaluation using common ranking performance measures such as mean average precision (mAP), but is much too small (∼10k images) to evaluate the performance of a real-world system operating at scale. Conversely, the ILSVRC dataset [13, 34] while being much larger (∼1M+ images) does not come loaded with complete annotation of *all* object categories in each image. This means that standard ranking metrics (*e.g.* precision, recall) cannot be computed without further annotation, and what annotation does exist is only really useful for object category *classification* metrics which are less sensitive to incomplete annotation (such as top-N classification error per image, customarily used for the ILSVRC challenge [13], which is a measure of the fraction of test images for which the correct label is not among the top N labels predicted as most probable), which do not accurately reflect the performance of an object category *retrieval* scenario.

As a result, in this chapter we use a custom combination of both of the above datasets, which aims to harness the existing complete annotation provided by the PASCAL VOC dataset with the scale of the ILSVRC dataset whilst reducing additional annotation as far as possible.

PASCAL VOC 2007 [37] is used as our base dataset, with assessment over all twenty of its classes. We use the provided train, validation and test splits.

ILSVRC 2011 [13] is used to augment the data from the PASCAL VOC 2007 test set as a source of distractor data. We remove all annotated ILSVRC synsets manually which clearly overlap with the PASCAL classes (*e.g.* for the PASCAL class

'car', synsets 'car mirror', 'cab, hack, taxi, taxicab', 'limousine, limo' *etc.* are removed). Following this, 500,000 images are sampled randomly to act as distractors and form a new dataset of ∼0.5M images.

Even with the removal of overlapping synsets from the ILSVRC dataset, due its lack of complete annotations it is still likely that instances of classes from the PASCAL class hierarchy will be present. Fortunately, as our chosen evaluation metric only requires us to evaluate using the top $K = 100$ images for each experiment, we take the approach of scanning each returned ranked list and removing any false negative images which surface from the dataset when using the images as distractors.

5.2.2 Experiments

This section describes the experimental setup, including dataset splits used and the parameters used for each of the encodings.

5.2.2.1 Scenarios

We run all experiments over two scenarios:

PASCAL VOC 2007 only — We use the train+val sets for training, and the test set for testing, which corresponds exactly to the standard VOC image classification task (as presented in Chapters 3 and 4). This scenario is primarily intended to provide a baseline benchmark at small scale of the various compression methods and representations, with the standard setup making the experiments comparable to other published work.

PASCAL VOC 2007 + distractors — Here we combine the 0.5M distractors from

the ILSVRC11 dataset as described in Section 5.2.1 with the PASCAL VOC dataset. Training is undertaken exactly as before (using just PASCAL VOC data) but the retrieval task becomes much more challenging, as in this scenario the positive images from the PASCAL VOC test set must be retrieved despite the addition of a large number of unrelated images.

5.2.2.2 Encodings

For both the Fisher encoding and VLAD, we use the same parameters as for the experiments in Chapter 3 along with spatial pyramid, which in both cases results in a encoding of size 327,680D. Although this is very high dimensional, post-compression the size becomes much more manageable, and they are intended as our 'gold standard' representations given their outstanding performance on standard benchmarks.

For PiCoDes, we compute both the 128D and 2048D variants described in the paper [13] using code from the author's website.[1] To compare this representation to PQ-compressed standard shallow encodings, we compute the histogram encoding, again with the same base parameters as in Chapter 3, but with a codebook size of 1,024 and 64 dimensions with spatial pyramids to match the dimension of the two PiCoDes variants after PQ-compression. We also explore a PCA-dimensionality reduced and whitened version of our VLAD features using the methodology described in Section 3.1.2. For this, full-sized VLAD encodings generated with a vocabulary of 512 centres and this time with no spatial pooling to give an output dimensionality of 40,060D are reduced using PCA to 512D to give a 128D PQ-compressed code to match that of the 128D

[1] `http://vlg.cs.dartmouth.edu/projects/vlg_extractor/vlg_extractor/`

PiCoDes variant.

For all experiments using product quantisation, the codes are compressed using a 256-bit PQ subquantiser for every 4-bits of the original uncompressed feature vector, resulting in a $4\times$ reduction in dimensionality and a $4 * 4 = 16\times$ reduction in storage requirements (moving from the single precision floats of the original feature vector to bytes in the compressed representation).

5.2.2.3 Cascade

We evaluate our cascade ranking approach only over the expanded PASCAL VOC + distractors scenario, and use the dimension-reduced variant of the VLAD representation for the first stage of the cascade and our full VLAD representation as the second stage of the cascade. A variety of values for the number of re-ranked images in the second stage N are investigated, from $N = 500$ up to $N = 10,000$.

5.3 Results and Analysis

The results of all experiments are presented in Table 5.1. We refer to this table in the sections which follow.

5.3.1 PASCAL VOC Experiments

Performance over the PASCAL VOC 2007 dataset is very much as expected given our experiments in Chapter 3. Fisher encoding ([a]) produces the best performance as measured by mean Prec @ 100 with 63.9% of results being true positives on average,

			Test data	**VOC 07**		**VOC 07 + ILSVRC11**		**VOC 07 + ILSVRC11**
			Train data	*VOC07*		*VOC07*		*Google*
Method	Dim.	Spatial Info	Compressed Size / bytes	Raw	Compr.	Raw	Compr.	Raw
(a) FK	256	spm (×8)	81,920	63.9	63.8	34.0	–	13.6 (25.8)
(b) VLAD	512	spm (×8)	81,920	61.7	61.6	29.1	–	10.9 (22.5)
(c) VLAD	512	none	10,240	58.6	58.5	23.8	–	8.3 (17.7)
(d) PiCoDes 2,048	–	–	2,048 (8)	47.1	46.9	–	–	–
(e) VQ	1,024	spm (×8)	2,048	51.4	50.7	12.9	–	2.6 (6.7)
(f) VQ	8,192	none	2,048	52.0	51.4	14.4	–	2.2 (3.3)
(g) PiCoDes 128	–	–	128 (0.5)	33.8	33.8	–	–	–
(h) VQ	64	spm (×8)	128	40.0	36.8	3.6	5.8	0.7
(i) VQ	512	none	128	42.7	39.9	3.2	6.6	0.3
(j) VLAD→512	512	none	128	**51.6**	**49.1**	**24.6**	23.9	0.1
(k) VLAD Casc. 500	[j] + [b]		82,048	–	–	–	34.7	–
(l) VLAD Casc. 1k	[j] + [b]		82,048	–	–	–	**35.9**	–
(m) VLAD Casc. 2k	[j] + [b]		82,048	–	–	–	35.8	–
(n) VLAD Casc. 5k	[j] + [b]		82,048	–	–	–	35.5	–
(o) VLAD Casc. 10k	[j] + [b]		82,048	–	–	–	34.4	–

Table 5.1: **Retrieval results for all experiments.** All figures are Mean Prec @ 100. For each method, both the performance with the original (uncompressed) features 'Raw' and PQ-compressed features 'Compr.' are given. In the final column, for the results using training images from Google, mean Prec @ 10 is also included in parenthesis. See the text for further details.

with VLAD ([b]) coming a close second at 61.7%. The per-class results for VLAD varied from 80.0% for 'aeroplane' down to 35.2% for 'potted plant'. Application of product quantisation compression to both the Fisher encoding, VLAD and histogram encoding resulted in a very small drop in Prec @ 100 of typically ∼0.1%, but this is offset by codes which are 16× more compact (which would result in a storage requirement reduction from 1.3TB to the slightly more reasonable 81GB for the features of a dataset of size 1M using the Fisher encoding in [a], for example).

Compression methods — Comparing these product-quantised codes to the binary codes produced by the PiCoDes encodings in ([d] and [g]) it can be seen that although that while both the 2048 and 128-dimension variants perform well, they are outperformed by even the histogram encoding given a fixed storage requirement budget ([d] *vs.* [e], [g] *vs.* [h]). Note that strictly speaking, each dimension of the PiCoDes representation can be stored in a single bit rather than a byte, resulting in a further 256× reduction in size, but it is unclear then how to apply a linear classification model $\mathbf{w}$ to the resultant codes, other than expensively expanding the binary codes to their float representations on-the-fly. Rastegari *et al.* [105] which use the related classeme representation for ranking also use linear classifiers, but it is not apparent they have a solution to this problem, and so we instead store the full 'float-ised' version of the PiCoDes features for fast processing instead.

5.3.2 PASCAL VOC + Distractors

Moving to the more difficult task of retrieving the small number of PASCAL VOC test images after the addition of 0.5M distractors, there is a corresponding drop in performance, with Prec @ 100 for the Fisher encoding ([a]) dropping to 34.0 and VLAD ([b]) to 29.1. The figures for Prec @ 10 of 57.3 and 48.6 respectively suggest that on average over half of returned images in the top 10 are still class instances, but in reality this does not capture the more subtle change which is that the difference in performance between 'easy' and 'more challenging' classes becomes more distinct, with the 'easier' classes still performing excellently, and the 'harder' classes less so. For example, the VLAD representation produces an average Prec @ 100 of 77.7% for the

'horse', 'train' and 'motorbike' classes whilst struggling with classes such as 'potted plant' and 'bottle'.

Different encoding methods — On this more challenging dataset, the performance gap between encoding methods also widens, with our compact histogram encoding method [e] achieving only 12.9% Prec @ 100. The performance gap between FK and VLAD ([a] *vs*. [b]) is less pronounced, and the returned rankings for many classes for these two methods is also very similar (an example is shown in Figure 5.3) which makes sense given how both of these methods encode similar information about the images.

Low dimensional encodings — The 512-dimension encodings ([h]–[j]) intended for use as the coarse ranker of our cascade ranking method also experience a drop, and one which is particularly significant for the histogram encoding method ([h] and [i]). It appears that at such low dimensions the quantisation error of the histogram encoding method causes the produced features to be not nearly discriminative enough to deal with the large volume of distractor data. Interestingly enough, this cannot be said for the dimensionality reduced VLAD encoding where the observed performance drop is much lower, from 29.1% to 24.6% ([b] *vs*. [j]). This is actually an improvement on the 23.8 achieved by the non-dimensionality reduced VLAD features without SPM ([c]) and suggests that the dimensionality reduced features still retain much of their discriminative power, consistent with the findings of Jegou *et al*. [67]. For this reason, we use these features as the first stage in our ranking cascade.

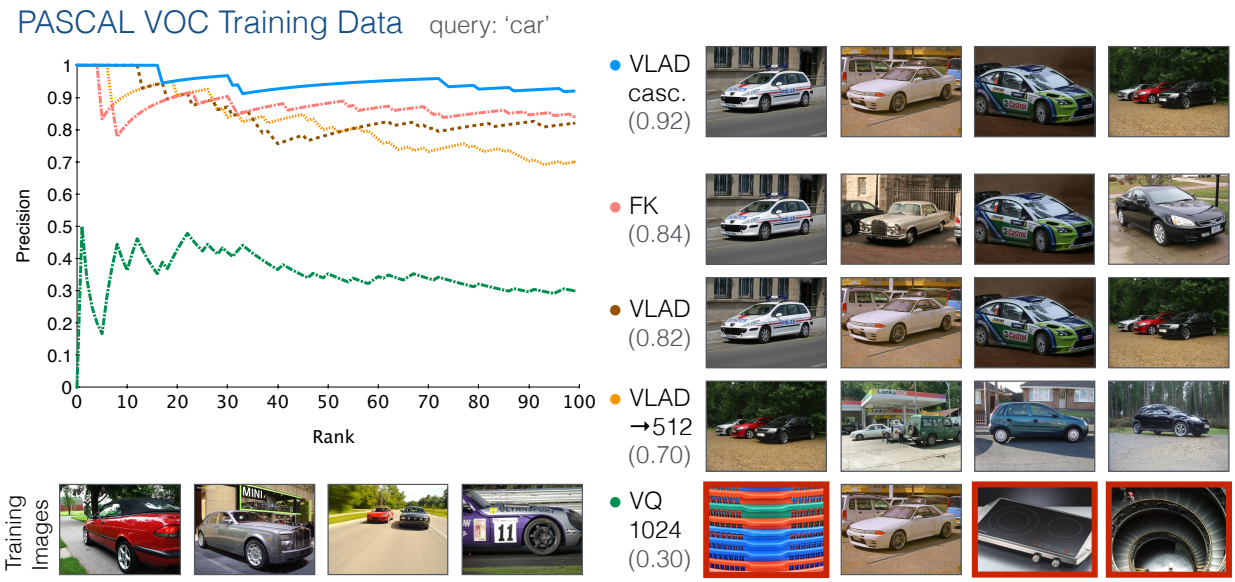


Figure 5.3: **Precision-rank curves and sample rankings for the query 'car' using training data from the PASCAL VOC dataset.** Results are shown for the combined PASCAL VOC + distractor data. A sample of the training images used to train the visual models are shown in the bottom left of the figure. False positives in the ranked lists are outlined in red.

5.3.2.1 Failure Modes

An example retrieval for the query 'car' is shown in Figure 5.3 along with the corresponding precision-rank curves for several methods. As described above, the performance of all of the Fisher encoding and VLAD-based methods is much better on the expanded VOC+distractors dataset than classic histogram encoding. However, even within this single query we can observe patterns in the failure mode that extend across all shallow encoding methods.

Putting to one side the result for the VLAD cascade method for now (which will be discussed in detail in the next section) the produced ranking for both the Fisher encoding and VLAD methods are very similar, with clear similarities to the training set. When moving to the 512D VLAD representation, the results remain very good,

but it can be seen that the top ranked images are more diverse. This seems to suggest that reducing the dimensionality (and also possibly by removing spatial information) from the encoding has actually had a positive benefit. Nonetheless, a result is that more false positives are also returned, particularly further down the ranked list (see the yellow curve in the plot), and the nature of the returned false positives, where they do occur, share similar characteristics, albeit in a more minor incarnation, to those returned by the low-dimensional histogram encoding method.

Turning now to the ranked results from this method, it can be seen immediately that even at the very top of the ranking the introduction of distractors has surfaced the encoding's inability to discriminate between cars and other images with similarly sharp edges such as the edge of baskets and mobile phones. This is consistent with both the lack of spatial pooling used with this method, and also quantisation error causing many similar looking edges being assigned to the same visual word, with the subtleties of the gradient patterns unique to a car being clumped together with less discriminative (and more common) patterns present in many other object categories.

This pattern is observed consistently when working with lower-dimensional shallow encoding methods, particularly in classes which tend to contain a large volume of highly textured regions within the image. For example, The PASCAL 'sheep' class tends to do worse with lower-dimensional encodings, as with fewer visual words the grass texture in the background of many sheep images becomes indistinguishable from the fur of the sheep itself. Overcoming the issue of these 'bursty' images can be achieved by increasing the size of the codebook or complexity of the encoding (although as shown above, this when taking to the extreme can occasionally have the side-effect of reducing

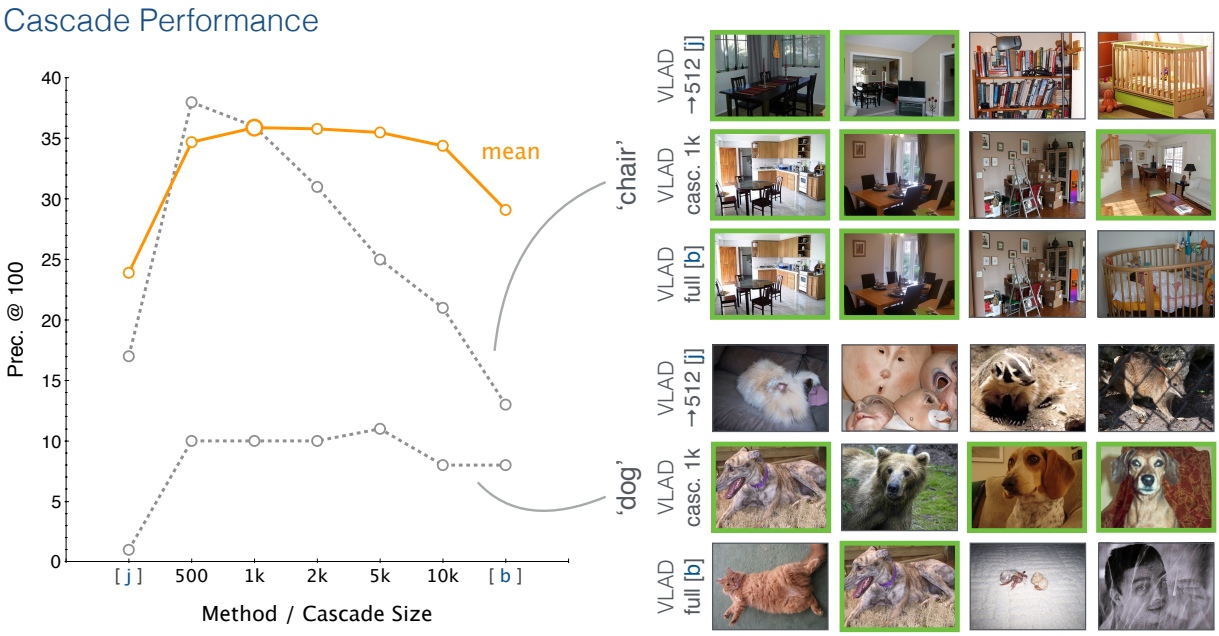


Figure 5.4: **Performance of the cascade ranking method.** The plot shows how the mean Prec @ 100 changes for different cascade sizes $N = 500, \ldots, 10\text{k}$, with the performance of the original feature representations also included, marked by their alphabetic row reference in Table 5.1. The plots and top four images returned for two sample queries 'chair' and 'dog' are also shown, with true positives outlined in green.

diversity) or, as we will see in Chapter 6, the use of alternative image representations that are more robust to this.

5.3.3 Cascade Ranking

We evaluate different versions of our cascade approach to ranking over the combined PASCAL VOC+distractors dataset, with the results presented in Table 5.1. A plot of the mean Prec @ 100 of each variant is shown in Figure 5.4 along with the performance of both the full VLAD features ([b]) and the lightweight VLAD features ([j]) used in the cascade. It is immediately obvious that far from reducing the precision of the produced ranking, the application of a cascade approach to ranking has actually *increased* the performance of the features compared to both the original constituent methods. Mean

performance peaks with a cascade size of $N = 1,000$, at which point the mean Prec @ 100 is 35.9% ([l]) compared to 29.1% for the full-dimensional VLAD representation ([b]) and 23.9% for the reduced dimension representation ([j]). In fact, all of our cascade methods exceed the performance of our best performing shallow method, the Fisher encoding, which only yields a mean Prec @ 100 of 34.0 ([a]), and that is despite a much reduced computation time and the advantages to ranking time that the cascade brings to the table.

Why is it that such a performance jump is observed? Looking at some sample rankings produced by the method and comparing it to the original ranked lists from both the full and lightweight features of the cascade gives us some clues. The rankings produced by the cascade with $N = 1,000$ are shown for two particularly challenging classes 'chair' and 'dog' in Figure 5.4.

Considering first the 'chair' example, it can be seen how both the full and lightweight features return images with chairs in as the top two results. However, the images returned by the lightweight features have a wider range of appearances, possibly for the reasons discussed in the previous section. This is also mirrored in the false positives returned, with a greater diversity in the false positives in the ranking of the lightweight features when compared to the full features, which seems to mainly get caught up on images of bookshelves. In other words, both for the returned true and false positives, both methods *succeed and fail in their own distinctive and different ways.* Sure enough, the combined ranking contains far fewer false positive bookshelf images whilst maintaining the 'good' top ranked images from the full feature ranking, which must also be captured in the top $N = 1,000$ of the coarse ranking.

The efficacy of the cascade approach is even clearer with the more challenging 'dog' class. In this case, Prec @ 100 is only 1% for the ranking produced by the reduced features, with the solitary returned dog ranking 6th (and thus not shown in Figure 5.4). The full features perform better, at 8%, but many of the returned dogs rank behind noisy false positive images. However, in the combined ranking, an improved Prec @ 100 of 10% is achieved, and 5 out of 6 top-ranked images are of dogs. This hints at the ability of the cascade approach to 'pull up' good images using our full feature representation once some of the noisy images which it is liable to get caught up on have been removed by our lightweight features in the first stage of the cascade.

A simple way of looking at these results is that the cascade is essentially providing a simple kind of feature fusion, controlling the excesses and mitigating against the limitations of both feature representations and producing an improved ranking by averaging out results. However, it is perhaps more appropriate to think of the cascade as an additional mechanism to overcome the quantisation error present in all shallow encoding methods through the use of multiple vocabularies and a hierarchical approach which allows the results to be iteratively refined. Through the use of multiple vocabularies, the tendency for shallow encodings to be fooled by 'bursty' images as described in the previous section as a result of assigning subtle repeating patterns to the same visual word is reduced. The use of a coarse ranker first, followed by the application of a 'refinement' feature also allows images to be pre-filtered initially on a higher level, before applying a finer-grained vocabulary which may be more susceptible to this effect. In this sense, the approach is related in spirit to the very early work of Nister and Stewenius [90] and their vocabulary tree. A further example ranking, comparing

				Time / s	Memory / GB
Feature Computation				0.382	–
Encoding	FK	[a]	327K	0.638	–
	VLAD	[b]	327K	0.134	–
	VQ	[e]	8K	0.133	–
Learning				∼1.0	
Ranking	FK/VLAD	[a]/[b]		1445	655
for 0.5M	FK/VLAD + PQ	[a]/[b]		19.2 ms	40.96
images	VQ	[e]		236 ms	16.38
	VQ + PQ	[e]		< 0.1 ms	1.02
	VLAD Casc. 10k	[l]		< 0.1 ms	41.02

Table 5.2: **Processing time and memory consumption of the system.** Timings are shown for all three components of the pipeline: encoding, learning and ranking. Memory requirements and ranking times are stated for a dataset of 0.5M images.

the cascade approach to other methods, is shown in Figure 5.3 in the previous section.

5.3.4 Memory and Computation Time

A summary of the memory requirements and processing time for each stage of the pipeline is shown in Table 5.2.

Encoding speed — Both VLAD and histogram encoding take around 130ms to compute, with the Fisher encoding taking over 5× as long at 640ms. Given the retrieval performance of VLAD and the Fisher encoding are similar, this motivates our choice of VLAD as our primary shallow representation for on-the-fly retrieval.

Ranking speed — The use of product quantisation, in addition to reducing the storage requirements of our features by 16× without significantly impacting their performance, also results in a massive increase in ranking speed of close to $8,000\times$ when using a

lookup table for computation of $\mathbf{w}.\boldsymbol{\phi}$. This puts on-the-fly ranking of datasets of 1M+ images using state-of-the-art high-dimensional shallow encodings in reach for the first time, as we will see in Section 5.4, with real-time ranking speeds with all overheads over a dataset of 5M images typically being a few seconds.

Our test size of 0.5M images is too small to really appreciate the speed benefit of our cascade approach to ranking, with results returned in under the precision of the CPU clock. However, over the larger dataset of 5M images considered in the next section, we observed a speed-up of at least $3\times$, taking ranking of a dataset of this size to sub-second levels, and this gain will of course improve the larger the dataset as the sizes of the cascade stages can be set independently of the dataset size, producing a cumulative speed-up of over $24,000\times$ when compared to the naive approach of ranking full-sized features.

5.4 On-the-fly Learning and Visual Search

Having accelerated the object category retrieval pipeline in the first part of this chapter by compressing state-of-the-art shallow features and employing a cascade approach to ranking, we now turn our attention to the generation of visual models to drive the search process. In the past, a common approach has been to train banks of pre-trained classifiers for a set of pre-determined queries [34], but it is our contention that such a methodology is neither sufficiently flexible nor scalable to really take visual search into the mainstream. Therefore, we instead explore training models on demand.

Inspired by the success of text search engines, we envisage a system where a user

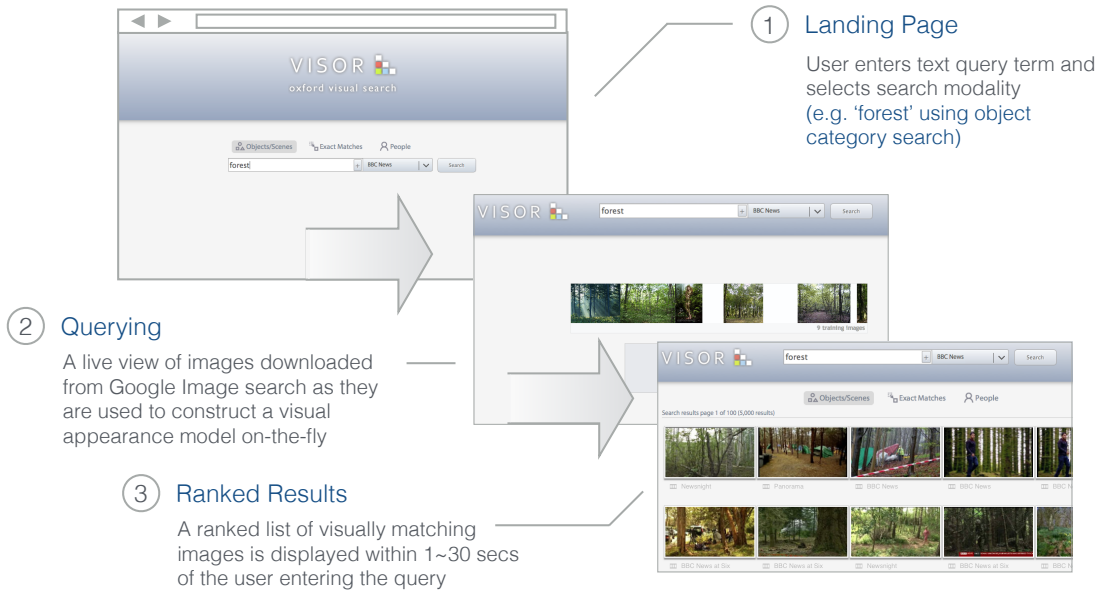


Figure 5.5: **Web-based on-the-fly demo system.** (1) the user enters a text query term and selects a search modality, (2) images are downloaded from Google Image search and used to train an appearance model on-the-fly, (3) ranked results over the target dataset are returned. A live demo is available online – see the text for details.

can input a text query describing a visual concept, and a visual model generated live in response which can then be used to query an unannotated dataset for images of that concept. We do this by learning from readily available images downloaded from the web, collected using standard image search engines (such as Google Image search). Although such platforms are still based on a combination of many different cues rather than just visual (*e.g.* clickthrough rate, surrounding text), over the past five years their precision has increased markedly, making them, in combination with state-of-the-art shallow image representations, a suitable candidate for bootstrapping the text query to visual data conversion process for the first time, as we will see. Screenshots of the prototype system we have developed based on this idea are shown in Figure 5.5. We go beyond simply using data from the web to pre-train models, as has been explored

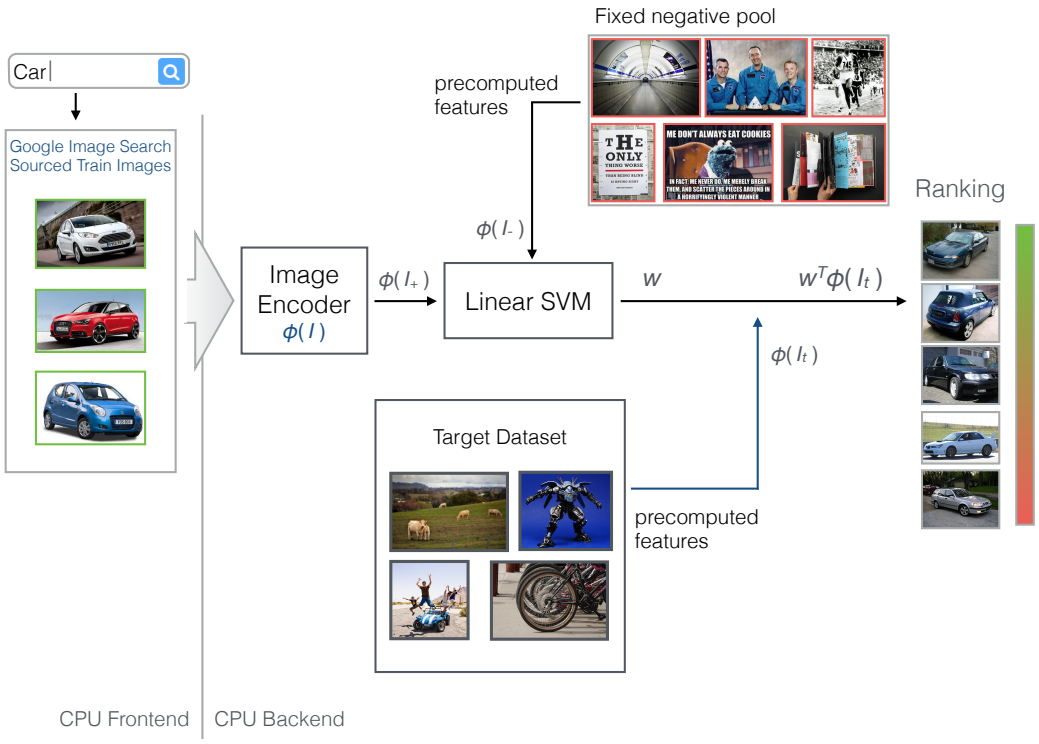


Figure 5.6: **Architecture of the on-the-fly object category retrieval pipeline.** Positive training images are downloaded on-the-fly from Google Image search by the frontend component, and are then fed to the backend for processing. In the backend, the images are encoded and after a fixed timeout of τ seconds all encoded images are fed to a Linear SVM for batch training (along with a fixed pool of negative features precomputed during the pre-processing phase). Finally, the linear SVM model $\mathbf{w}$ is applied to the precomputed features of the target dataset, and the resulting classification scores sorted in descending order to produce the output ranking.

by several authors in the past [42, 77] (see Section 2.4.3), to use such data to prepare visual models *on-the-fly* in response to the real-time queries of the end user.

5.4.1 An Architecture for Online Training

Our proposed architecture for on-the-fly learning is shown in Figure 5.6, and is split into a pre-processing and online stage:

Pre-processing — Visual features are extracted for all images in the target dataset,

along with those for a fixed pool of $\sim 16,000$ negative training images. The negative images are sourced from the web by issuing queries for a set of fixed 'negative' query terms[2] to both Google and Bing image search, and attempting to download the first $2,000$ results in each case, and for each class around $\sim 1,200$ positive training images are retrieved. Both the dataset features and those of the negative training pool are then stored in memory when the system is launched for speed of access.

On-line stage — Given a textual query, the corresponding top $K \sim 250$ images are retrieved from Google Image Search. Visual features are computed on-the-fly for each image as it is downloaded. These images are used as the positive visual training data for our object category, and used with the fixed pool of pre-computed negative training data to train a linear SVM in the normal way, fixing the SVM C parameter to $C = 0.25$. This can be used to rank the unannotated target dataset using the compressed features and efficient ranking methods described in the first half of this chapter.

5.4.2 Evaluating Models Trained from the Web

In order to evaluate the performance of models trained from the web, we adapt the second evaluation scenario described in Section 5.2.2. First positive training data is collected for each of the PASCAL VOC classes by issuing the class name as a query to Google Image search, retrieving around ~ 250 images in each case. This data is used along with the fixed pool of $16,000$ negative training images described in Section 5.4.1 to train a visual model for each class, which is then evaluated over the combined PAS-

[2]miscellanea, random selection, photo random selection, random objects, random things, nothing in particular, photos of stuff, random photos, random stuff, things

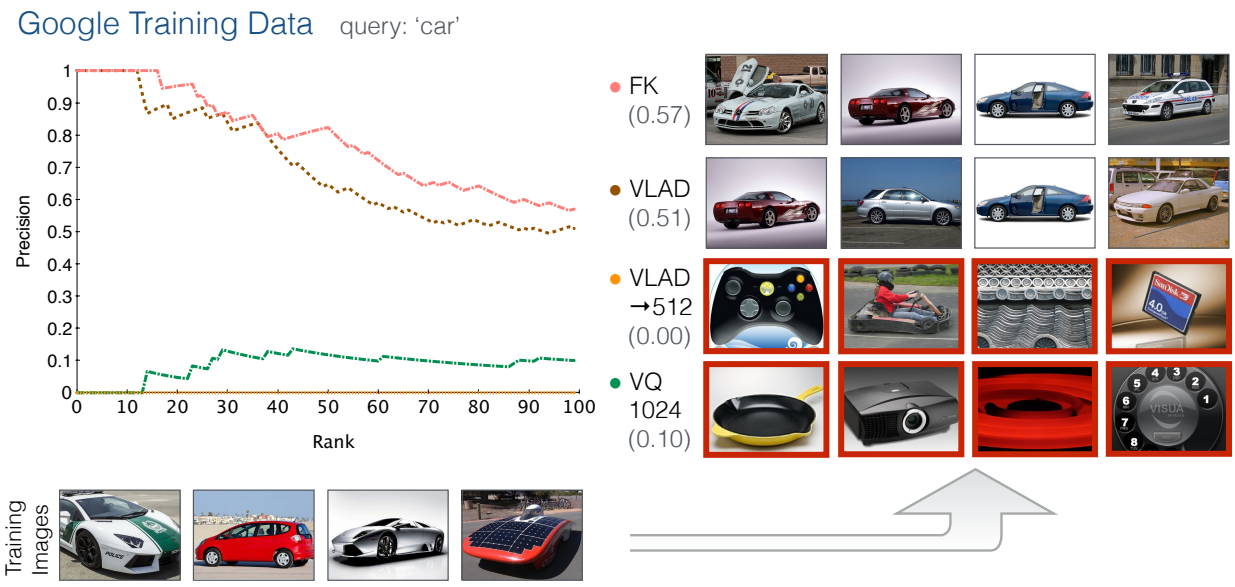


Figure 5.7: **Precision-rank curves and sample rankings for the query 'car' using training data from Google.** Results are shown for the combined PASCAL VOC + distractor data. A sample of the training images used to train the visual models are shown in the bottom left of the figure. False positives in the ranked lists are outlined in red.

CAL VOC + ILSVRC11 distractor dataset as before. The results of these experiments are reported in the final column of Table 5.1.

Switching to Google training data over this particular data does result in a drop in performance across the board when compared to the results trained on the VOC training set. However, what is interesting is that this manifests itself by the total failure of some classes, with many of the classes which performed well before still maintaining good performance even with the non-curated training data. An example is shown in Figure 5.7 for the query 'car'. Here we can also see that the style of the returned images is also different to those returned when using the VOC training data (refer to Figure 5.3), with the results using the Google training data tending to be more of a sideview of the vehicle with a plain background, reflecting a different distribution and focus in the Google training images. As can be seen from the precision-rank curves,

the results for the Fisher encoding ([a]) and VLAD ([b]) methods remain very good at the top of the ranking, with most of the performance drop observed towards the tail-end of the top 100 images.

The story is different for the lower dimensional encodings such as the dimensionality reduced VLAD ([j]) which fails completely in this new scenario. However, this appears to be caused by these lower dimensional representations being unable to sufficiently capture the difference in images caused by the domain shift to Google-sourced training images, rather than the models being no good, since the top of the returned rankings has plenty of false negative images of cars sourced from the ILSVRC11 dataset which were removed prior to evaluation following our annotation guidelines (refer to Section 5.2.2). More generally, the domain shift appears to cause a more or less significant loss of recall with our shallow features (reflected in the fact that more difficulty is experienced in pulling out the small number of true positive PASCAL images from the combined VOC + distractor dataset) but if there are enough class instances in the dataset, the observed performance drop can be much lower, as we will see in the next section.

5.4.3 Live Demo System

We have developed a working on-the-fly visual search system based on the architecture described in this chapter, screenshots of which are shown in Figure 5.5 and for which a live demo is available online.[3] The system currently demonstrates on-the-fly search over a video dataset of 5M+ images provided by the BBC of News broadcasts, further described in Section 2.5. Typically even over such a large dataset, results are returned

[3] `http://www.robots.ox.ac.uk/~vgg/research/on-the-fly/`

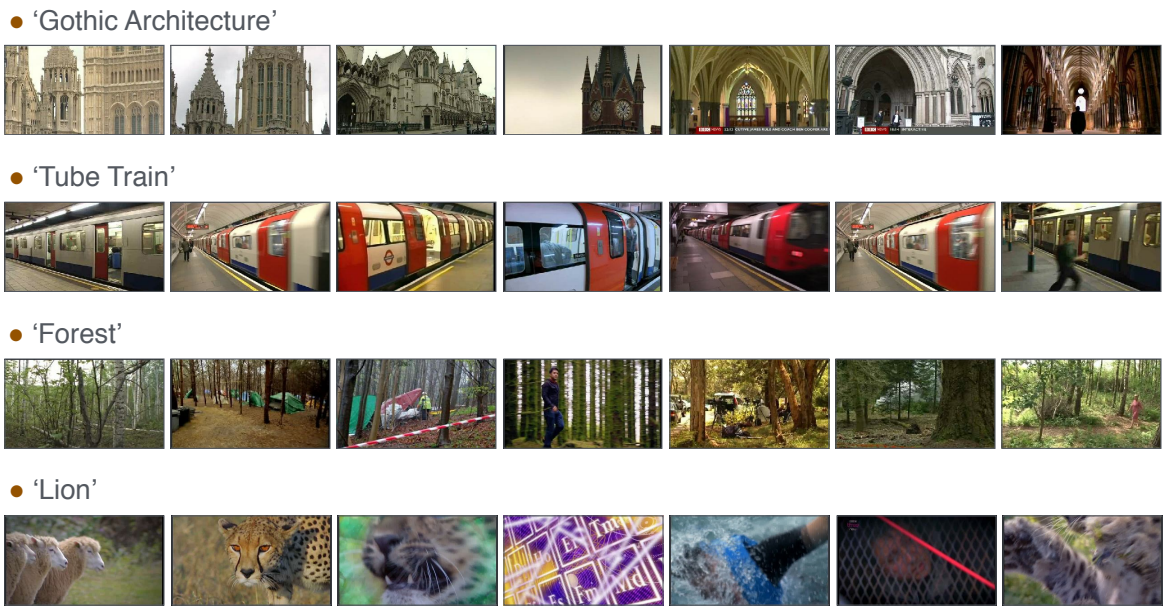


Figure 5.8: **Object Category Retrieval Results** over the BBC News dataset using images from Google Image search for training. All results use the PQ-compressed VLAD representation [b].

in around ~ 10 seconds from query input, most of which is spent downloading training data from the web and computing features for these images with training and ranking being much faster (around ~ 2 seconds combined).

5.4.3.1 Performance over Large-scale Data

Some example results over the BBC News data are shown in Figure 5.8. It can be seen that for a wide selection of queries, the system performs well. In particular, queries for manmade objects or scene types (*e.g.* 'gothic architecture' and 'tube train') return very good results. Many natural queries such as 'meadow' and 'forest' also work well. However, the tendency for the shallow features discussed in this chapter to catch on 'bursty' images as described in Section 5.3.2.1, and this is even more evident when searching for highly-textured objects such as 'lion' over the larger BBC News dataset,

with many of the returned images just being full of 'visual noise' and not containing lions. In Chapter 6, the next chapter, we will see how this can be overcome by using different image representations that are more robust to this.

5.5 Conclusion

In this chapter we have shown how the shallow image representations introduced in Chapter 3 can be adapted successfully for a visual category search system which is capable of operating over datasets of millions of images, in real-time and without any prior knowledge of the queries for which it will be used. We did this by (i) significantly speeding up the application of visual models to large datasets, by both applying efficient compression methods to state-of-the-art shallow feature encodings and proposing a novel cascade ranking approach which has the added benefit of *improving* retrieval performance and (ii) demonstrating how current web image search engines such as Google Image search and Bing can be used as a source of live training data, allowing visual models to be trained on-the-fly for use in such an accelerated pipeline.

In addition, we conducted an extensive evaluation of the strengths and limitations of shallow feature encodings for the large-scale retrieval task. It was shown how state-of-the-art image representations such as VLAD provide excellent performance even over large-scale datasets providing that issues of 'burstiness' are managed properly. In the next chapter, we will see how turning to deep image representations provides an alternative to this setup which is more robust to encoding quantisation loss whilst providing several other distinct advantages over the architecture proposed in this chapter.

Chapter 6

Scaling Category Search with ConvNets

In this chapter, we build upon the on-the-fly architecture developed in Chapter 5, demonstrating how through the use of Convolutional Networks (ConvNets) [75] with GPU training [71] it can be developed further, to operate faster, with better precision and at even larger scale.

The basis for the work in this chapter is the evidence given in Chapter 4 that the latest deep image representations significantly outperform shallow feature encodings on the image classification task. In addition to this, they also have the big advantage over state-of-the-art shallow feature encodings such as the Fisher encoding and VLAD of being much lower dimensional, even without compression. Nonetheless, we explore how they can be compacted further to produce incredibly low-dimensional and yet performant features, and obviating the need to make the same kind of performance/memory/speed trade offs that is necessary with shallow features. The compact

deep feature representations thus produced are introduced in Section 6.1, before we outline the evaluation setup which will be used to test these features for large-scale retrieval in Section 6.2 and present our experimental results and analysis.

However, our contributions go further than simply using ConvNet features in the same classical retrieval architecture as described in Chapter 5: using the amenability of deep features to be computed in a highly parallel manner on a GPU as a starting point, we develop an entirely new, GPU-based architecture which operates in a highly parallel and fundamentally iterative manner in Section 6.3. This facilitates the simultaneous computation of training features with the incremental training of a linear SVM at runtime, therefore allowing results to be returned from the system close to instantaneously. Alternatively, a time budget can be set so that the classifier, trained on the available images within the time limit, can be used to (re-)rank the dataset images at any stage of the process (for instance, every 0.5s). This is in strong contrast to the system proposed in the last chapter, where SVM training only begins once all training images have been downloaded and processed, and ranking follows after that.

What we are left with is a system that is able to go from cold-query to results even faster than the system proposed in Chapter 5, in a matter of second(s) over datasets of 1M+ images, and has the potential to either: (i) scale to even larger datasets of perhaps tens of millions to billions of images or (ii) operate in an entirely self-contained manner on commodity GPU hardware, to provide visual search at scale even on consumer-grade laptops and without the need for an expensive server, all whilst performing with improved precision compared to the first-generation system of the previous chapter.

Once again, we develop a working system as part of this work, and screenshots

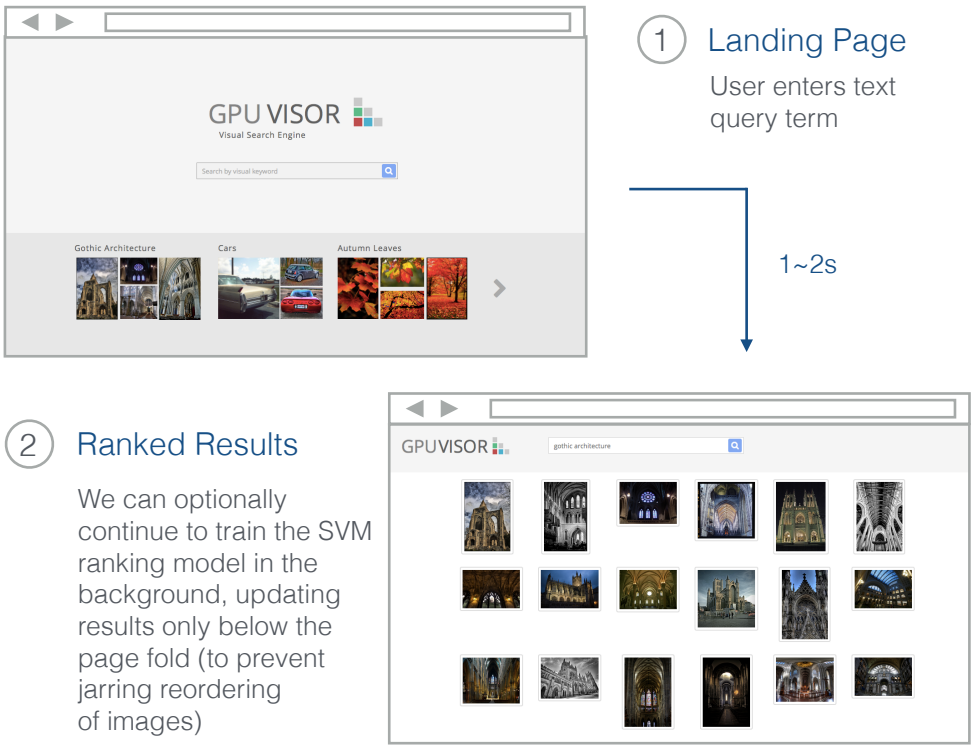


Figure 6.1: **Live on-the-fly system web frontend**. From entering a novel text query to viewing the results page, the entire process takes ∼1–2 seconds.

of our second-generation ConvNet based visual search system with a summary of its operation is shown in Figure 6.1.

6.1 Compact Deep Image Encodings

As our baseline deep features, we use the 2048-dimensional 'CNN M 2048' network as described in Chapter 4. Further implementation details of the baseline network can be found in Section 4.2.2, and we use the same software framework to compute the features. As mentioned in the introduction, the features produced by the 'CNN M 2048' network are already significantly lower-dimensional than the state-of-the-art shallow representations used in the last chapter. Nonetheless, in order to make them as

compact as possible for the ranking stage, we explore compressing them further using one of three methods:

Lower-dimensional ConvNet output layer — Given that the ConvNet features derive from a learnt representation with multiple levels, they also have the advantage that it is relatively easy to modify the network architecture to produce lower dimensional features without correspondingly reducing the bandwidth of the bulk of the encoding pipeline, as is the effect of reducing the codebook size in classical shallow representations, since a dimensionality reduction can be applied to just the final hidden layer leaving the remainder of the architecture as is (and keeping the trained weights from when the final layer was high-dimensional if computing the weights of the new lower-dimensional final layer using fine-tuning). As shown in Chapter 4, by re-training just this final layer, it is possible to obtain a lower-dimensional representation without a large performance hit over standard benchmarks. In order to evaluate such an approach in the retrieval setting, we consider the 'CNN M 128' network introduced in the same chapter. Comparing to the baseline 'CNN M 2048' network, this can be seen as discriminative dimensionality reduction by a factor of 16×.

Product quantisation — We proceed as in Chapter 5, using $Q = 4, 8$ dimensional sub-blocks for the PQ sub-quantisers, with each sub-quantiser using a codebook containing 256 codewords, producing compression by 16× and 32× respectively.

Binarisation — We explore the use of binarisation using Parseval tight frames, as described in Section 2.4.2.1, as an alternative to product quantisation. We up-project our 128-dimensional ConvNet features 'CNN M 128' to one of $D = 1024, 2048$ dimensions,

but given the resultant features are binary rather than floating point, this results in a compression of 2× and 4× respectively.

6.2 Evaluating Deep Image Encodings

Once we have our features, evaluation proceeds as before: a linear SVM is trained for all evaluation classes, and used to rank all images in the target dataset. Given the produced ranking, we evaluate using Prec @ $K = 100$.

6.2.1 Evaluation Data

As in Chapter 5, all experiments are run over a combination of one or more of the PASCAL VOC 2007 dataset and a second dataset (refer to Section 5.2). However, motivated by the potential use of our system consumer-grade hardware as well as servers, we replace ILSVRC11 with another dataset which better emulates the application of visual search to the task of retrieval from personal photo collections, either locally or on the web, namely the MIRFLICKR-1M dataset described in Section 2.5. This dataset has been confirmed to contain many images of the twenty PASCAL VOC classes.

All 1M images from the dataset are used, making a combined dataset of 1M+ images, or roughly double the size of the data used in Chapter 5. We also do not remove any classes from the dataset as we use the dataset for evaluation in its own right, as a archetypal consumer photography collection with unknown statistics. We use both datasets for three different evaluation scenarios, with each moving closer to modelling

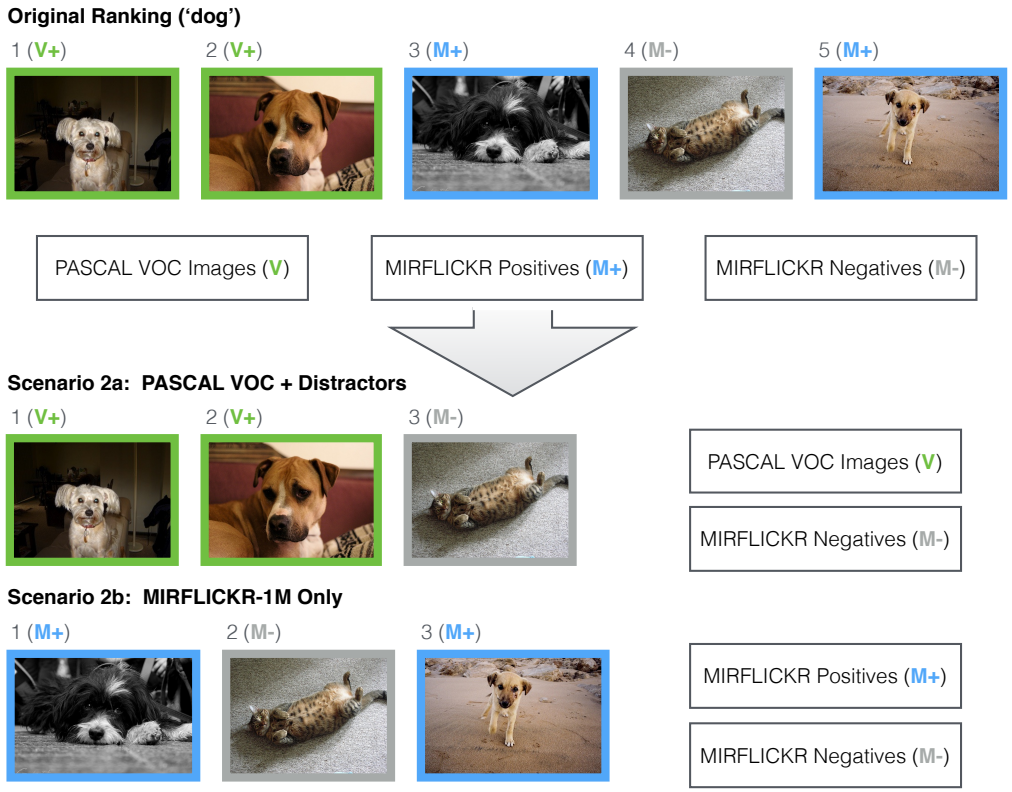


Figure 6.2: **Data subsets used for evaluation.** Using the example of the object category 'dog', the ranked lists used for evaluation in scenarios 2a+2b are compiled by combining the PASCAL VOC data with lazily annotated data from the MIRFLICKR-1M dataset.

the conditions experienced by a real-world large-scale object category retrieval system.

The scenarios are described in the next section.

6.2.2 Scenarios

6.2.2.1 Scenario 1: PASCAL VOC

We train models for seventeen of the twenty VOC object classes (excluding 'people', 'cats' and 'birds' for reasons described below in the section for scenario 2) using both the training and validation sets. Following this, a ranked list for each class is generated using images from the test set and Prec @ K evaluated.

6.2.2.2 Scenario 2: Large-scale Retrieval

Training is undertaken in the same manner as scenario 1, but during testing images are added from the MIRFLICKR-1M dataset. There are two sub-scenarios (using different subsets of the test data, summarised in Figure 6.2).

Scenario 2a — We test using images from the PASCAL VOC test set (as in scenario 1) with the addition of the entirety of the MIRFLICKR-1M dataset. For each class, we remove all positive class occurrences in the ranked list which are retrieved from the MIRFLICKR-1M dataset using the lazy annotation described in Section 6.2.3, as the purpose of this scenario is to test how our features perform when attempting to retrieve a small, known number of class occurrences from a very large number of non-class 'distractor' images.[1]

Scenario 2b — This time we exclude all images from the PASCAL VOC dataset, and instead evaluate precision @ K solely over the MIRFLICKR-1M dataset, lazily annotating the retrieved ranked lists in each case as before. The purpose of this scenario is to test how our features perform over a real-world dataset with unknown statistics. In practice, it is an easier scenario than scenario 2a, since the MIRFLICKR-1M dataset contains many instances of all of the PASCAL VOC classes.

[1]The prevalence of the PASCAL VOC classes 'people', 'cats' and 'birds' in the MIRFLICKR-1M data explains why we exclude them, as restricting the annotation of these classes to reasonable levels proved to be impossible.

6.2.2.3 Scenario 3: Google Training.

Testing is the same as in scenario 2b, but instead of using PASCAL data for training, a query is issued to Google Image search for each of the PASCAL VOC classes, and the top $N \sim 250$ images are used in each case as training data. This is exactly the same data as used in Section 5.4.2. This scenario assesses the tolerance to training on images that differ from the VOC and MIRFLICKR-1M test images: the Google images may be noisy and typically contain the object in the centre. It also mirrors most closely a real-world on-the-fly object category retrieval setting, as the queries in practice do not need to be limited to the PASCAL VOC classes. There are again two sub-scenarios, with different data used for the negative training samples in each case:

Scenario 3a — The images downloaded from Google Image Search for all other classes, except for the current class, are used as negative training data (this mirrors the PASCAL VOC setup).

Scenario 3b — The same fixed pool of $\sim 16,000$ negative training images as described in Section 5.4.1 is used.

6.2.3 Dataset Ground Truth Preparation

As described in Section 6.2, we use a combination of the PASCAL VOC 2007 dataset and MIRFLICKR1M dataset for evaluation. MIRFLICKR1M does not come with any annotation, apart from noisy flickr image tags, and so we add our own annotations for the twenty PASCAL VOC classes.

Despite the dataset containing 1M images, we can get away with annotating far less

than this number given our chosen evaluation metric, precision @ K with $K = 100$, which only requires the ground truth for the first K items in the ranked list of each target class to compute. We therefore adopt a 'lazy' approach to annotation using our result ranked lists as a starting point.

The evaluation set (and thus the meaning of the 'first K images') is different for each scenario, as shown in Figure 6.2. Therefore, given any raw ranked list for class C (which is a combination of results from both the PASCAL VOC and MIRFLICKR1M datasets) it suffices to annotate images which fall within the following ranges:

- **For Scenario 2a** – the top K images from: the PASCAL VOC dataset combined with all images from MIRFLICKR1M annotated as negative for the target class C (these are 'distractors')

  ... *excluding* annotated positives for class C from MIRFLICKR1M.

- **For Scenario 2b/3** – the top K images from the MIRFLICKR1M dataset

  ... *excluding* all PASCAL VOC images.

The annotations we make for any particular method/scenario should be stored so that images do not need to be annotated more than once for different methods. We developed a web-based annotation tool to facilitate this process which allows positive annotations for a class to be shared across both methods and scenarios.

In total, 46,770 images from the MIRFLICKR1M dataset were annotated, with an average of $\sim 2,000$ annotations per class.

6.2.4 Results and Analysis

The results for all three experimental scenarios are presented in Table 6.1, along with details of storage requirements and computation time in Table 6.2. We discuss the results for each scenario below.

6.2.4.1 Scenario 1 (VOC Train/VOC Test)

Reflecting our findings in Section 5.3.1, the PASCAL VOC dataset does not pose any major challenges for any of our features. Our baseline shallow encoding, the Fisher encoding, produces an improved mean Prec @ 100 of 82.3% *vs.* the 63.9% recorded for the same experimental scenario in Table 5.1 in the last chapter, primarily due to the doubling of the codebook size to 512D. With this expanded encoding, even for the most challenging classes (*e.g.* 'potted plant') fairly good results are achieved, with the top 12 images being true positives (Prec @ 100 = 0.58) compared to the top 44 images being true positives in the case of our 2048-dimensional ConvNet features (Prec @ 100 = 0.83). Despite their far lower dimensionality, it is clear that the ConvNet-based features ([b], [d] in Table 6.1) have an edge over the Fisher representation ([a]).

6.2.4.2 Scenario 2a (VOC Train/VOC+distractors Test)

Adding 1M distractor images from the MIRFLICKR-1M dataset has a significant impact on the results, with the task now being to retrieve true positives that constitute less than $\sim 0.02\%$ of the dataset. This is a more challenging scenario, and under this setting the superior performance of the ConvNet-based features, when compared to the state-of-the-art shallow representation (IFV), is much clearer to see. Some sample

Scenario		VOC Only [1]	Large-scale Retr. [2a]	[2b]	Google Training [3a]	[3b]
(a) FK	512	82.3	29.3	80.5		
(b) CNN	2K	92.1	**55.4**	95.4	88.5	90.9
(c) CNN	2K PQ	90.7	55.1	96.4	88.2	91.9
(d) CNN	128	92.1	51.0	95.1	88.1	92.3
(e) CNN	128 noaug	88.8	45.4	93.1	87.1	91.1
(f) CNN	128 BIN 2K	91.5	**52.3**	94.0	89.6	
(g) CNN	128 BIN 1K	90.0	50.1	94.0	89.5	
(h) CNN	128 PQ	90.1	50.5	94.6	88.2	92.1
(i) CNN	128 PQ-8	88.8	47.4	93.1	87.7	91.1

Table 6.1: **Retrieval results for all image representations.** All figures are Mean Prec @ 100, with the labelled scenarios corresponding to the data splits described in Section 6.2.2.

	Dim	Compression			New Dim (bytes)	Storage / 1M ims.	Comp. Time / im (s)
(a)	83,968	–				312.8 GB	10.32
(b)	2048	–				7.63 GB	0.35 (0.061)
(c)	2048	PQ	4 dims/sq	(16×)	512	488 MB	+ 0.061
(d)	128	–				488 MB	0.34 (0.061)
(e)	128	noaug				488 MB	0.083 (**0.024**)
(f)	128	BIN	2048 bits	(2×)	256	244 MB	+ 0.38 ms
(g)	128	BIN	1024 bits	(4×)	128	122 MB	+ 0.22 ms
(h)	128	PQ	4 dims/sq	(16×)	32	30.5 MB	+ 3.9 ms
(i)	128	PQ	8 dims/sq	(32×)	16	**15.3 MB**	+ 2.0 ms

Table 6.2: **Dimensionality, storage requirements and computation time for all image representations.** The rows in this table correspond to those in Table 6.1. Timings for compression methods are specified as additional time added to the total feature encoding time, and those in parenthesis indicate GPU timings where applicable.

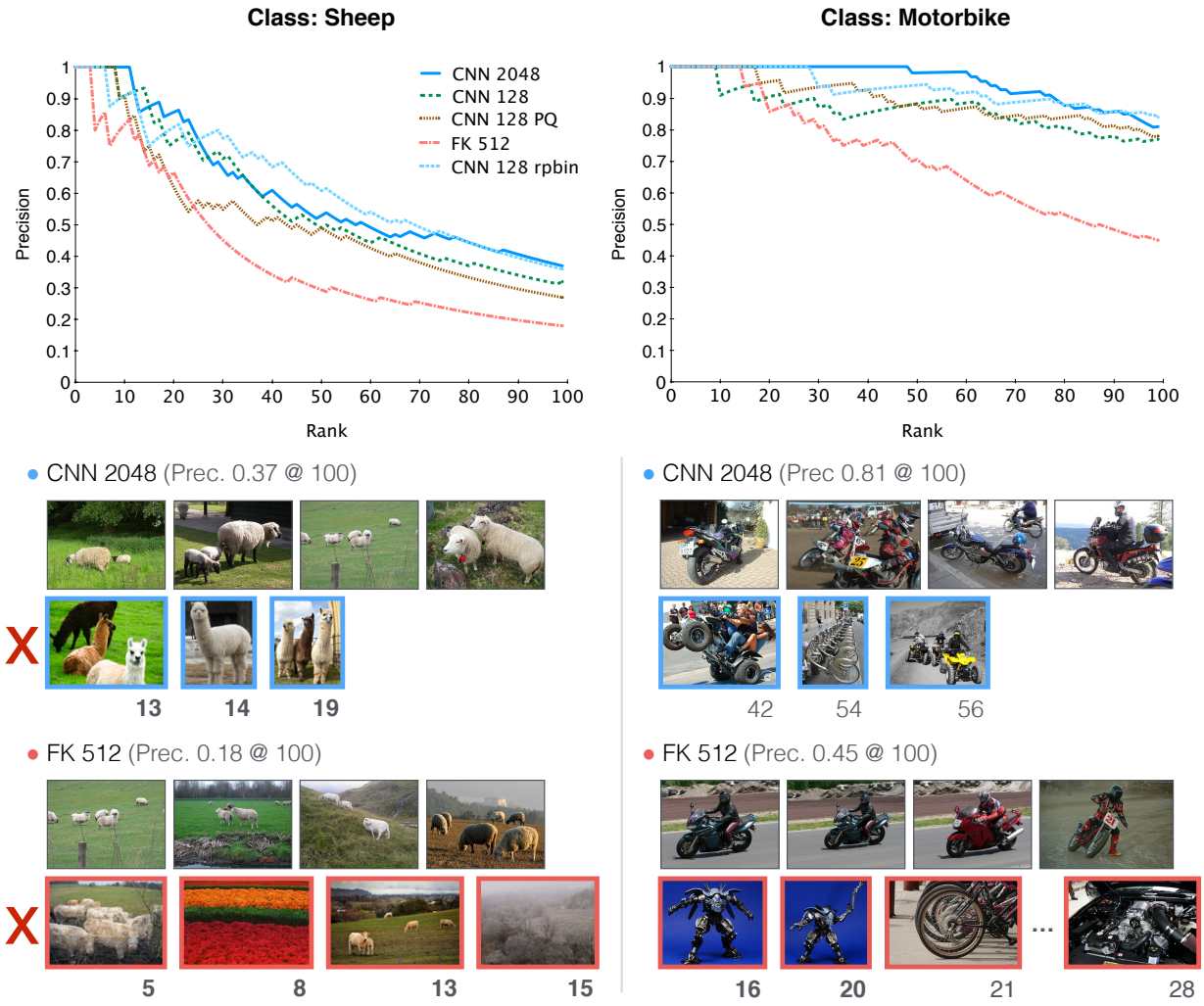


Figure 6.3: **Sample precision-rank curves and retrieved results** for two queries over the combined VOC+MIRFLICKR data (Scenario 2a). In the bottom half of the figure, the top row in each case shows the first few results returned for each method and the second shows the top retrieved false positives with their rank.

precision-rank curves for two queries, one particularly challenging ('sheep') and another less so ('motorbike') are shown in Figure 6.3. We can make the following observations:

IFV Performance. It can be seen that IFV ([b]) performs the worst of all methods, despite being much higher dimensional ($\sim 1000\times$) and taking much longer to compute ($\sim 200\times$) compared to our CNN-128 method ([d]). Nonetheless, even for challenging classes such as 'sheep' IFV manages to pull out a few true positives at the top of the ranked list. However, the relative performance drop with rank is much sharper than with the ConvNet-based methods.

Bursty Images and Diversity. As noted in Section 5.3.2.1, the mode of failure for shallow encodings for the large-scale retrieval problem tend to fall into a number of fixed patterns. One particularly prevalent phenomenon is for these encodings to get caught up on 'bursty' images, that is to say images which are highly textured or contain repeating patterns. This can be overcome to an extent through the use of larger codebook sizes, although this can have the undesired impact of reducing diversity as noted in our evaluation of the cascade ranking approach in Section 5.3.3. It is interesting therefore to consider the performance of our deep features with particular regard to these two properties:

Bursty Images — Comparing the top-ranked negatives for the FK-512 method ([a]) for 'sheep' to those of the CNN-2048 method ([b]), it can be seen that even with a very large codebook (at 512 codewords double the size used in Chapter 5 and producing encodings of size $655,360$D) IFV is still prone to mistakenly rank highly 'bursty' images

comprising repeating patterns or textures. This phenomenon is particularly evident for natural, outdoor scenes which explains why the performance of IFV is particularly low for the 'sheep', 'cow' and 'horses' classes. On the other hand, it appears that the ConvNet-based features are much more robust to such textured images, although the use of heavy PQ compression (*e.g.* the CNN-128-PQ-8 method [i]) starts to show some deterioration as a consequence of the retrieval of a smaller number of similarly 'bursty' images.

Diversity — The diversity of the retrieved results is also much greater for ConvNet-based representations than for IFV, indicating that the classifier is able to make better generalisations using these features. For example, as seen in Figure 6.3, whereas the top four retrieved results for the query 'motorbike' for the FK-512 method ([a]) all show a rider in a similar pose, on a racing bike on a race track, the top four retrieved results for the CNN-2048 method ([b]) depict a variety of different motorcycles (road, racing, off-road) from several different angles.

For the most part, compression of the ConvNet features does not appear to reduce their diversity appreciably, with the top-ranked results for all ConvNet methods, whether compressed or not, appearing to exhibit a similar diversity of results.

Compression. As mentioned above, the drop in performance in moving from ConvNet-based features to IFV is much greater than that incurred by any of the compression methods, and this seems to be strongly connected with the robustness of the ConvNet-based features, whether compressed or not, to the kind of 'bursty' textured images which IFV and other shallow methods are susceptible to. This is remarkable given

that comparing the size of the largest uncompressed ConvNet representation CNN-2048 ([b]) to the smallest compressed one, CNN-128-PQ-8 ([i]), there is a $\sim 512\times$ size difference. In the case of the CNN-128-BIN-2K method ([f]), the mPrec @ 100 actually increases marginally when compared to the non-compressed codes ([d]) which, when visually inspecting the rankings, again can be explained by the additional robustness brought by compression.

The binary representations ([f] & [g]), combined with a linear SVM, also exhibit competitive performance despite the reduced memory footprint. The ranking of such features can be significantly sped-up using hardware-accelerated Hamming distance computation, which, however, requires a different ranking model, which is left for future work. For its superior compression ratios and negligible impact on performance, product quantisation remains an obvious choice for the compression of ConvNet features, just as it was for shallow features. The fact that the ConvNet features are reasonably sparse, more so than typical shallow representations with the CNN-128 representation typically being over 60% zeros, is one reason why they are so amenable to compression, and it is possible that with compression methods geared specifically to capitalise on this sparsity even higher compression ratios could be achieved.

6.2.4.3 Scenario 2b (VOC Train/MIRFLICKR Test).

Given that the MIRFLICKR-1M dataset contains many instances of all of the PASCAL VOC classes, moving to testing solely on MIRFLICKR leads to a jump in performance of the results across all methods. Nonetheless, this scenario provides a closer representation of the performance of a real-world on-the-fly object category retrieval system,

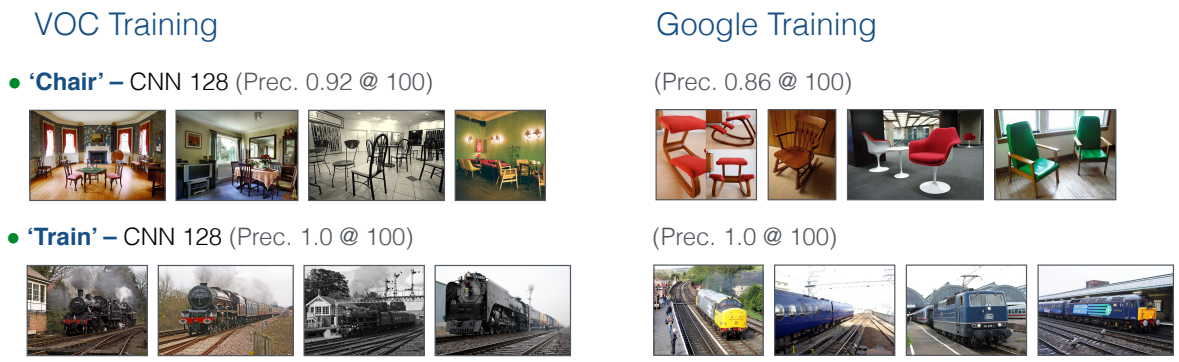


Figure 6.4: **Difference between retrieved results when training using VOC data and Google training data.** Results are shown over the MIRFLICKR-1M dataset (Scenarios 2b and 3b).

given that the image statistics of the MIRFLICKR-1M dataset are not known in advance.

6.2.4.4 Scenario 3a (Google Train/MIRFLICKR Test).

Switching to noisy training images from Google rather than the pre-curated PASCAL VOC training images as expected results in a small drop ($\sim 6\%$) across the board for all methods. However, the precision at the top of the ranking remains subjectively very good, and the performance impact is much less than observed in our tests for shallow representations in Chapter 5. Nonetheless, as also noted in the previous chapter and shown in Figure 6.4, the actual images returned from the dataset are very different, which reflects the differences in the training data sourced from Google Image search versus that from the curated dataset. For example, a query for 'chair' returns predominantly indoor scenes with regular dining-table chairs when using VOC training data, and more avant-garde, modern designs, generally centred in the frame when using Google training data.

6.2.4.5 Scenario 3b (Google Train + negative pool/MIRFLICKR Test).

In this scenario, we switch to using a fixed pool of negative data sourced from a set of 'negative' queries, and it can be seen how this improves the results by up to $\sim 5\%$. This may be a result of the larger negative training pool size ($\sim 16,000$ images *vs.* $\sim 4,000$ images when using queries for all other VOC classes to provide the negative data as we do in Scenario 3a). Given the assumed lack of coverage in the fixed negative image pool (as it is sourced by issuing queries for deliberately non-specific terms to facilitate its application to as broad a range of queries as possible), this suggests that to a certain extent lack of diversity can be made up for by using a larger number of negative training images.

6.3 ConvNet-based GPU On-the-fly Architecture

Having evaluated various image representations in Sect. 6.2.4, we now describe the architecture of the object category retrieval system, which fully exploits the advantages of ConvNet image representations. From the user experience point of view, the main requirement to our system is instant response: the first ranking of the repository images should be obtained immediately (in under a second), with a potential improvement over time. This dictates the following design choice: downloading the training images from the Internet should be carried out in parallel with training a model on the already downloaded images in the on-line fashion. As a result, at any point of time, the current model can be used to perform ranking of the dataset images.

For this approach to work, however, image representation should satisfy the follow-

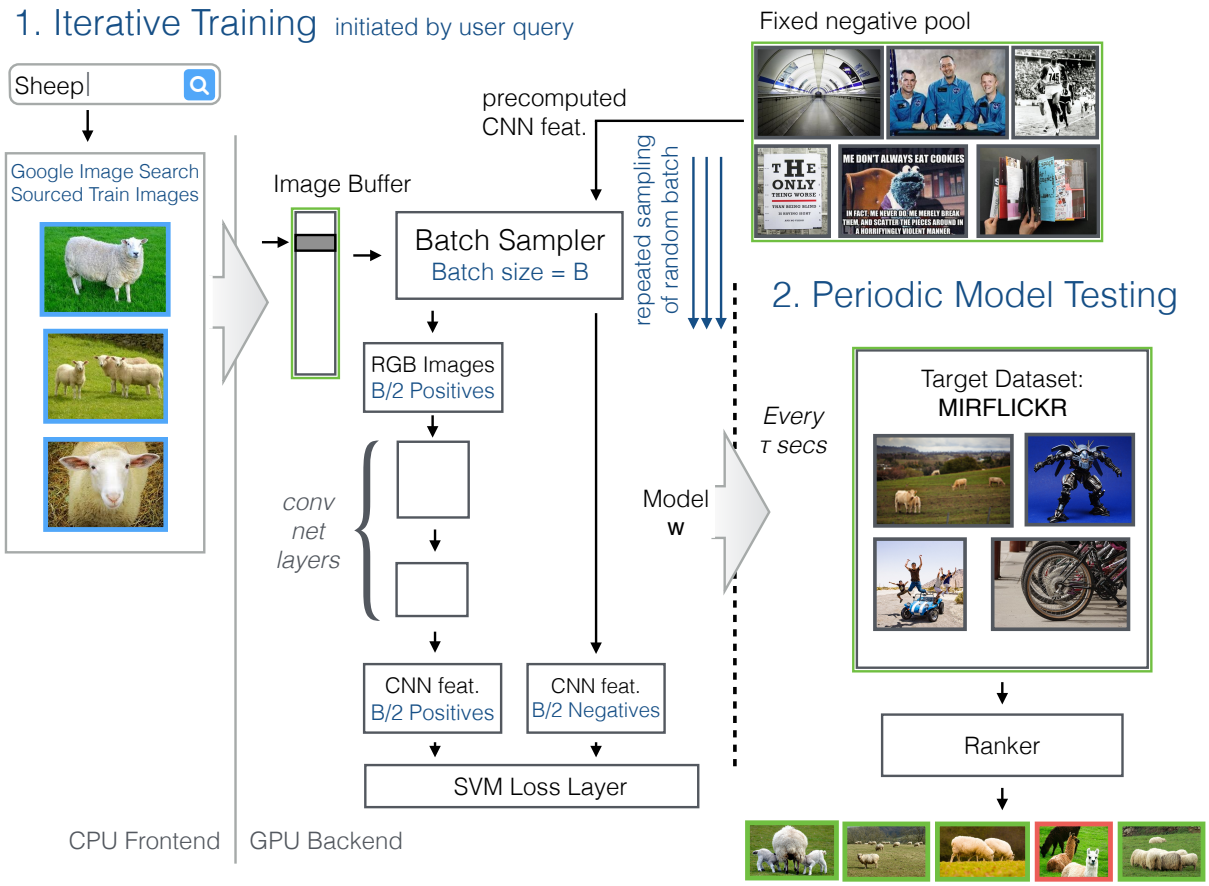


Figure 6.5: **Architecture of our on-the-fly object category retrieval system.** The entire framework aside from the image downloader is resident on the GPU, with data stored in GPU memory outlined in green. Its operation is split into two stages: (i) iterative training, as initiated by a user text query and (ii) periodic model testing to obtain a ranking over the target dataset (refer to the text for further details).

ing requirements: (i) highly discriminative, so that even a handful of training samples are sufficient to learn a linear ranking model; (ii) fast-to-compute, to maximise the amount of training data processed within the allocated time budget; (iii) low memory footprint, to allow for storing large-scale datasets in the main memory, and ranking them efficiently. As has been demonstrated in Sect. 6.2.4, a ConvNet image representation is a perfect match for these requirements. Indeed, pre-training on a large image collection (ImageNet) leads to highly discriminative representation, and even

a few training samples are sufficient for training an accurate linear model; ConvNet features can be computed very quickly on the highly-parallel GPU hardware; they have low dimensionality (even without PQ compression) and can be instantly scored using a linear model on the GPU.

Our on-the-fly architecture is illustrated in Fig. 6.5. It is divided into the CPU-based front-end (which controls the graphical user interface and downloads the training images from the Internet) and the GPU-based back-end, which continually trains the ranking model on the downloaded images and periodically applies it to the repository. The category retrieval is carried out as follows.

Off-line (pre-processing). To allow for fast processing, the ConvNet features for the target dataset images are pre-computed off-line, using the CNN-128 architecture. We also prepare the fixed negative image pool for *all queries* by issuing our negative pool queries (see Section 6.2.2) to both Bing and Google image search, and downloading the returned URLs. The negative image features are also pre-computed. The memory requirements for storing the pre-computed features are as follows: 488 MB for the MIRFLICKR-1M dataset and 78 MB for the pool of 16K negative features. It is thus feasible to permanently store the features of both negative and dataset images in the high-speed GPU memory even without compression of any kind (a consumer-grade NVIDIA GTX Titan GPU, used in our experiments, is equipped with 6GB RAM). As noted in Section 6.2, the ConvNet features can be compressed further by up to 16× using product quantization without significant degradation in performance, making datasets of up to 160M images storable in GPU memory, setting 1GB aside for storage

of the model (compared to 10M images without compression), and more if multiple GPUs are used. Many recent laptops are fitted with a GPU containing similar amounts of memory, making our system theoretically runnable on a single laptop. Furthermore, whilst storing the target repository on the GPU is preferable in terms of the ranking time, in the case of datasets of 1B+ images, it can be placed in the CPU memory, which typically has larger capacity.

On-line (CPU front-end). Given a textual query, provided by a user (*e.g.* in a browser window), the front-end starts by downloading relevant images, which will be used as positive samples for the queried category and fed to the GPU back-end. At regular time intervals, the front-end receives a ranked list of dataset images from the back-end, and displays them in the user interface.

On-line (GPU back-end). The GPU back-end runs in parallel with the front-end, and is responsible for both training the ranking model and applying it to the dataset. *Training* an ℓ_2-regularised linear SVM model is carried out using the mini-batch SGD with Pegasos updates [116]: at iteration t, the learning rate is $\frac{1}{\lambda t}$, where λ is the ℓ_2-norm regularisation constant, set to 1 in our experiments. Each batch contains an equal amount of positive and negative samples; the total batch size was set to $B = 32$ in our experiments. The training commences as soon as the first positive image has been downloaded and is received from the front-end, after which B random crops are taken each iteration from the pool of positive training images downloaded so far (so for the first iteration, B crops will be taken at random from a single positive training image downloaded, and then from the N_{iter} training images available on the second

iteration, and so on). Similarly, B of the pre-computed sub-crops are taken from all images in the negative image pool to provide the negative training data. The front-end in the meantime will continue downloading new images from the Internet, constantly increasing the size of the positive image pool and the diversity of the extracted crops. We note that while the positive image features need to be computed on-the-fly, this is very quick in the case of ConvNets. *Ranking* takes place using the current SVM model every τ seconds (we used $\tau \sim 0.18$). As mentioned above, the pre-computed dataset features are pre-stored on a GPU, so the scores for 1M images are computed in ≈ 0.01s. The 1M scores are then ranked (also on GPU, ≈ 0.002s) and the list of the top-ranked images is passed to the front-end to be displayed to the user. All components of the GPU back-end are implemented within the same framework, derived from Caffe [68].

6.3.1 System Performance

In order to evaluate the real-world performance of the system, we ran queries for several PASCAL VOC classes and tracked how the performance (measured in terms of Precision @ 100) evolved over time. To simulate the latency introduced by downloading images from the Internet, we limited the rate of positive images entering the network to 12 images/second (which is what we found to be a typical average real-world rate on our test system). These images were sampled randomly from the top-50 image URLs returned from Google Image search.

The results of these experiments for four classes are shown in Figure 6.6. Even for some of the most challenging PASCAL VOC classes 'sheep' and 'sofa', the performance converged to its final value in ~ 0.6 seconds, and as can be seen from the evolving rank-

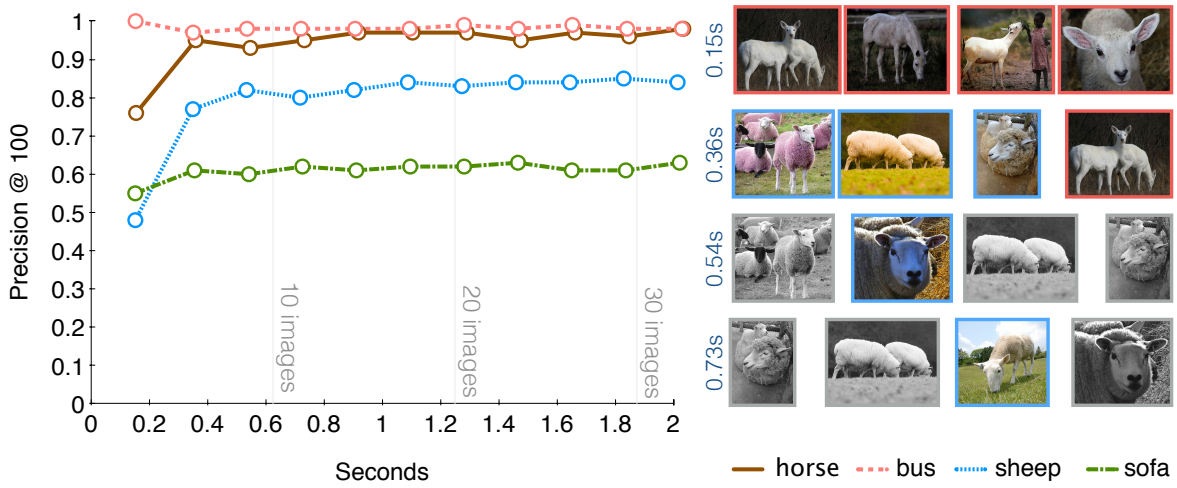


Figure 6.6: **Precision @ 100 against training time for four queries using our on-the-fly architecture.** The number of images in the dynamically expanding positive image training pool over time is also marked on the plot. The top-4 returned images for the 'sheep' query at the first four time-steps (up to 0.73s) is shown to the right. False positives are outlined in red, and new images in the top-4 at each time step are outlined in blue. Even for this moderately challenging query, the model settles in under a second.

ing at each time-step the ordering at the top of the ranking generally stabilises within a second, showing a good diversity of results. For easier classes such as 'aeroplane', convergence and stabilisation occurs even faster.

In real terms, this results in a typical query time for our on-the-fly architecture, from entering the text query to viewing the ranked retrieved images, of **1–2 seconds and often less** to complete convergence and stabilisation of results. However, one of the advantages of our proposed architecture is that it is adaptable to differing query complexity, and we can return good results early whilst still continuing to train in the background if necessary, exposing the classifier to an expanding pool of training data as it is downloaded from the web and updating the ranked list on-the-fly.

Figure 6.7: **Evolution of performance with increasing number of positive training images**. Results are presented for two of the queries presented in Section 6.3. (a) shows the performance (measured as Prec @ 100), (b) the number of new true positives entering the ranking (top 100) as the number of training images increases, (c) shows the minimum initial ranking position of any of those true positives, (d) shows the head of the ranked list after N=1..5 images have been added, and (e) shows the mean change in ranking position for all images in the top 100.

6.3.2 Impact of Training Image Count

In this section, we present further more detailed analysis of the changes that occur as more training images are fed into the network, to supplement those described above for the two most challenging classes 'sheep' and 'sofa'. The motivation is to determine the role of the size of the positive training image pool in the performance of the system. Note that to this end the experimental setup is slightly different to the previous section, as after inputting each training image into the system we waited for the output classifier to stabilise. We analyse the impact on each class in turn, referring to Figure 6.7.

Considering first the 'sheep' class, with only a single training image 70% of the final performance (as measured by precision @ 100) is reached, and the top of the ranked list contains many sheep. However, most of the highly ranked images are of horned sheep, suggestive of the bias introduced by training only on a single image. As the number of training images is increased to 2, the top-ranked images become much more diverse, with this translating into a further final small jump in performance as the third training image is fed into the network.

The 'sofa' class provides an example of how the architecture deals with a more challenging query, with a larger degree of intra-class appearance variation. In this case, a single training image clearly does not suffice, as the ranked list returned for a single training image has performance close to random, with no sofas retrieved. However, this very quickly changes as a second image is fed into the network, with 35 new true positives entering the top 100. Following this exposure, the top retrieved images are greatly improved, mostly being of sofas. Feeding five images into the network yields a further modest increase in diversity at the top of the ranked list.

In both cases, for this dataset any new true positives introduced to the top 100 after the introduction of the third or fourth training images have a very high initial position (~ 80) and the mean change in ranking position is very low (~ 1.5) suggesting that a coarse model can be trained with relatively few images, and improvements after this time predominantly effect the tail of the ranked list. This suggests that even when initially a very small number of training images are available, a user interface where the head of the ranked list is presented to the user almost immediately (trained on the small amount of training data which is available) whilst training continues in the background to refine the tail of the ranked results is possible. Such a restriction does not apply in our case, since as mentioned in Section 6.3.1 in general we have 30+ training images available to us within a few seconds of launching a query.

6.3.3 Novel On-the-fly Queries

Although experimental results have thus far only been presented for the PASCAL VOC classes, the advantage of an on-the-fly architecture is that no limitation is imposed on the object categories which can be queried for, as a new classifier can be trained on demand (in our case using Google Image search as a 'live' source of training data). We present some additional results of the on-the-fly system in Figure 6.8, using the same setup as in *Scenario 3b* and query terms disjunct from the twenty PASCAL VOC classes to test its performance for such novel on-the-fly queries. It can be seen that the architecture is very much generalisable to query terms outside of the PASCAL category hierarchy.

Some queries such as 'lion' were particularly challenging for shallow feature repre-

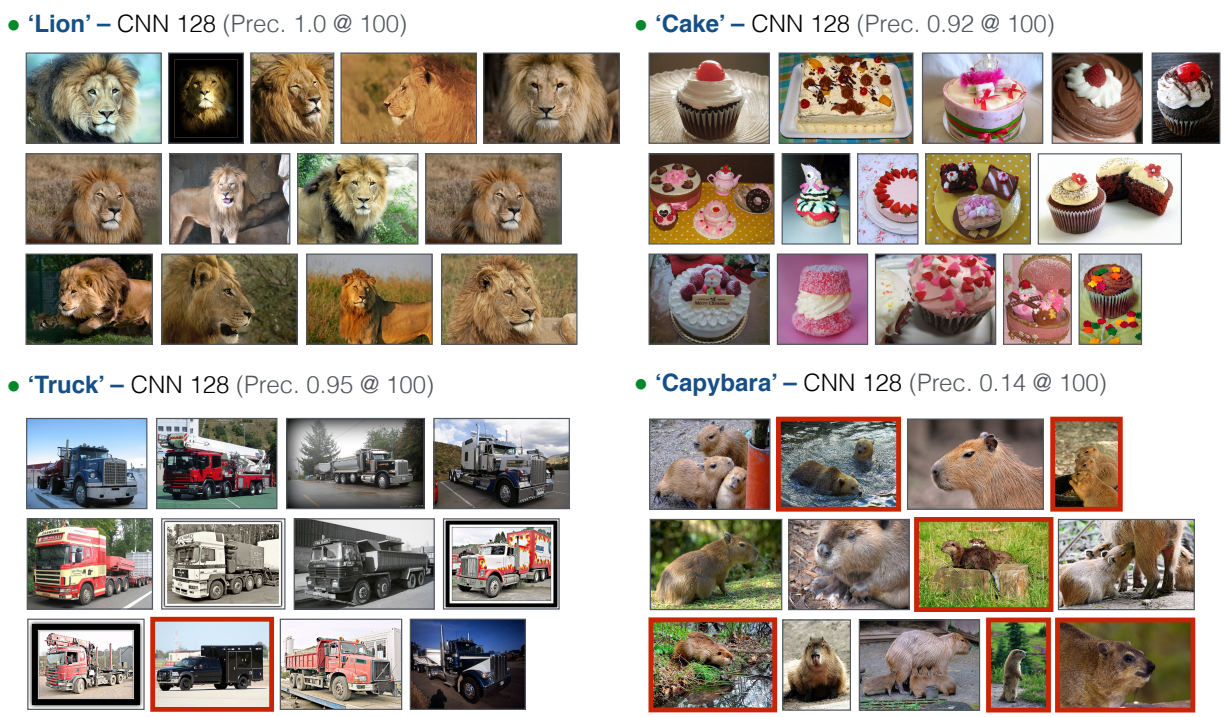


Figure 6.8: **Sample results for queries outside of the twenty PASCAL VOC classes.** False positives are outlined in red.

sentations such as Fisher Kernel, due to the repeating thick fur pattern and bushes present in many of the training images retrieving a large number of the bursty images described in Section 6.2.4. However, ConvNet-based features appear to be much more robust to this effect, with precision @ 100 of 1.0. The architecture is also capable of returning more abstract concepts such as 'cityscape' or 'forest' in addition to more concrete objects such as 'cake' and 'truck' (shown in the figure).

Finally, even when querying MIRFLICKR1M for the relatively obscure 'capybara' (Figure 6.8) which performs less well, the returned false positives all fit within a tight configuration of classes of very similar appearance ('otter', 'squirrel', 'meerkat') and, of course, the composition of the MIRFLICKR1M dataset is unknown, so it could be that there are very few images of 'capybara' in the dataset.

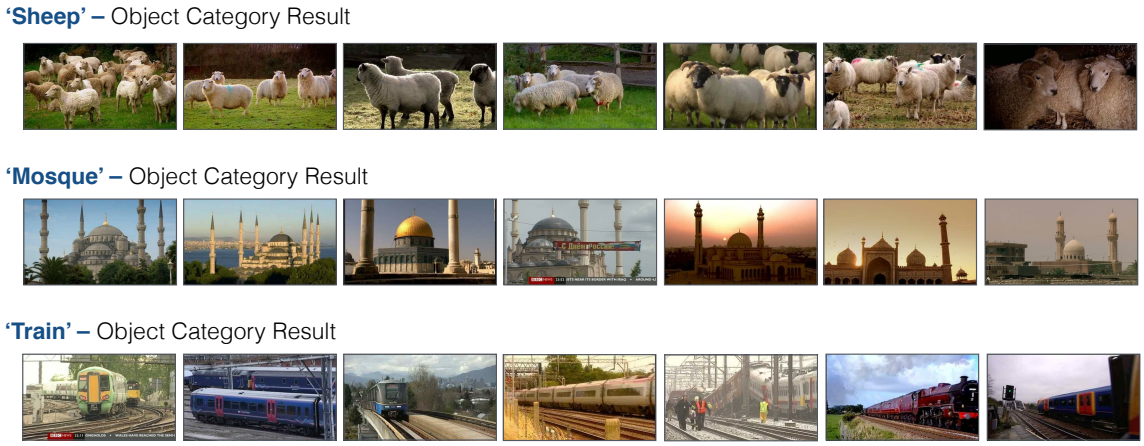


Figure 6.9: **Object Category Retrieval Results** over the BBC News dataset using images from Google Image search for training.

6.3.4 Performance over BBC Data

Some sample ranking results over the BBC News dataset described in Section 2.5 are shown in Figure 6.9. This dataset is larger still than the MIRFLICKR-1M dataset by a factor of 5, comprising 5M+ images. Compared to the live demo system described in Chapter 5 running over the same data, the extra power of the ConvNet-based features make a noticeable difference to the qualitative retrieval performance of the system, with an even wider range of queries than before now working well, marking a positive step towards a general purpose visual search system with real usefulness. In particular, classes such as 'sheep' (shown in the figure) or 'lion', which typically comprise images with a noisy, heavily textured background, generally performed quite badly with the previous generation shallow encoding-based system, but the ConvNet features are much more robust to such images.

6.4 Conclusion

In this chapter we have presented a system for on-the-fly object category retrieval, which builds upon the recent advances in deep convolutional image representations. We demonstrated how such representations can be efficiently compressed and used in a novel incremental learning architecture, capable of retrieval across datasets of 1M+ images within seconds and running entirely on a single GPU. For larger datasets the CPU, or multiple GPU cards, could be employed for ranking once the classifier has been learnt on the GPU. Along with further investigation of how the diversity of the ranked results changes over time, this is the subject of future work.

Chapter 7

Extensions

In this chapter we explore possible extensions to the on-the-fly visual search systems outlined in Chapters 5 and 6.

7.1 Query Refinement by Colour

One possible extension to our on-the-fly architecture is the incorporation of live refinement by visual attributes. As described in the introduction, this provides a very natural way of thinking about visual search, and allows the user to narrow down their search by adding additional descriptive labels ('attributes') which do not define a new query in and of themselves but rather serve as modifiers to refine the original query. One of the simplest attributes is colour, and we performed some initial experiments to investigate the effectiveness of query refinement by colour using ranked lists returned from the on-the-fly retrieval system from Chapter 6 as the starting point. Some sample colour attribute refinements for two different queries 'car', and 'bottle' are shown in

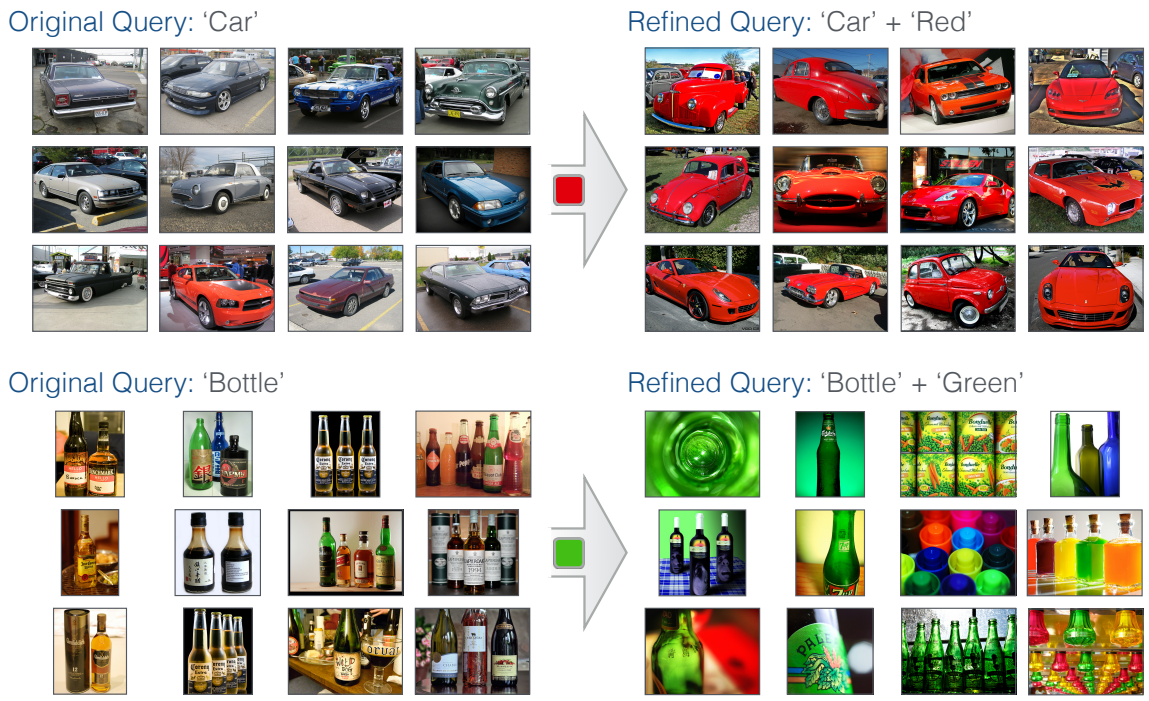


Figure 7.1: **Query refinement by colour attribute.** The original ranked list is produced using the CNN-M-2048 method from Chapter 6. By re-ranking by global colour histogram, specific subsets of the returned images can be interactively explored.

Figure 7.1, and demonstrate how such attributes can be used to retrieve simple and visually intuitive specialisations of the query quickly and effectively.

Despite the effectiveness of these colour attributes, the approach used is very simple: during the offline stage for each image in the dataset, a colour histogram ϕ_{colour} is computed by assigning every pixel in the image to its closest match in a pre-prepared palette of $C = 54$ cardinal colours, which are sampled evenly from the colour wheel such that neighbouring histogram bins are close in colour space. Following this, the histogram representation of each image is smoothed using a fixed Gaussian kernel. At runtime, the user selects a query colour from the cardinal palette which is then used to re-rank the top $N = 10,000$ images from the ranked list by sorting in descending order of Euclidean distance in CIELab colour space between a query histogram containing

only the query colour and all smoothed target image colour histograms. One of the downsides of this basic approach is that the feature is entirely global so, as can be seen in the figure, occasionally images with a large area of the query colour in the background may equally be re-ranked highly. This could be resolved by either centre-weighting the contribution of the image pixels to the colour histogram, or by using more advanced (and expensive) techniques such as running a lightweight object detector over the image to start, and then only accounting for colour in areas where objects of interest are present. Further development of such attributes is left for future work.

7.2 Query Fusion using Extreme Value Theorem

A parallel line of thought to the refinement of queries by attributes is the combination of different distinct queries into a single ranked list. This could be useful if, for instance, adapting a motivating example given in the introduction, the user wished to search for all images containing a road running through a glade of trees decked in autumnal leaves. Although a search for 'autumn road' might return adequate results, an alternative way of formulating the query might be as a combination of 'autumn leaves' and 'road'.

The difficulty in such a scenario is that fundamentally the ranking scores of the classifiers employed in Chapters 5 and 6 are unnormalised, and so cannot be compared across queries. One possible approach to overcoming this problem is to make use of statistics in the form of the Extreme Value Theorem (EVT) to translate the raw scores into a relevance score which *can* be compared across queries, an idea explored by Scheirer *et al.* [113] for normalisation of different attribute scores, and also by

Furon and Jégou [45] for the task of image detection. The basic idea is to model the *non-class* images in a retrieved set of ranking scores as occurrences of random variables which are i.i.d, *i.e.* distributed according to an unknown but unique law. Under such an assumption, top-ranking false positives, *i.e.* those non-class images which have a high score, can be modelled as extreme values in a theoretically determined family of distributions. Furon and Jégou [45] show how the parameters of this distribution can be estimated by alternately removing 'outliers' which are taken to be true positive images and then fitting the distribution to the remaining scores. Once this distribution has been learnt it can be used to translate each raw scores to a probability that it deviates from the random image distribution, which should be more consistent across queries.

We undertook some preliminary experiments using EVT score normalisation over the PASCAL + VOC distractors dataset from Chapter 5. Using the baseline VLAD features from that chapter, we computed rankings for all PASCAL VOC classes (corresponding to the second experimental scenario in the chapter) and then normalised the scores for each class. Following this, we evaluated the Global Average Precision (GAP) for both the normalised and unnormalised scores, similar to in Furon and Jégou [45]. This is done by interleaving results from all classes sorted by descending order of score and computing the precision-recall curve globally, treating as positives all PASCAL class instances correctly retrieved in the original rankings, and can be used as a measure of the comparability of scores between different classes. Over the combined PASCAL + VOC distractors dataset, we observed an increase in GAP from around $\sim$14% when using unnormalised scores to $\sim$27% when using normalised scores, suggesting that the method does indeed make scores more comparable. Nonetheless, closer inspection of

the ranked lists suggests that the effect is quite subtle, and works well when combining two classes which performed well to start with, but less so when combining a class that retrieves good results with one that performs less well.

This suggests that in order to normalise scores effectively using such techniques, we first need some measure of how 'good' the ranking model is to start with. One way of obtaining such a measure is to employ classification methods with stronger introspective capacity, such as Gaussian Process classifiers (GPC), an idea explored by Grimmett *et al*. [55]. These provide not only a classification score, but also a bounds on classification *confidence*. Unfortunately, such techniques do not scale well with high-dimensional features, making the training time prohibitive when used with the classic shallow encodings explored in Chapter 3. However, given the much lower dimensionality of deep features their application may now be possible. This could also be explored in future work.

Chapter 8

Conclusion

In this chapter, we summarise the contributions of the work in this thesis, and discuss promising directions for future research.

8.1 Contributions

In this thesis we have proposed methods for accurate, efficient and scalable visual search for object categories in large web-scale datasets. We started by evaluating and building on state-of-the-art image representations for image classification and finished by architecting a system which is capable of retrieving visual categories without any prior knowledge from datasets of millions of images within seconds. This paves the way for truly practical realtime visual search for a multitude of applications, from search within personal photo collections to entire libraries of digital archive footage. The journey we have taken is summarised as follows:

In Chapter 3 we performed a comprehensive evaluation of shallow image encodings

for image classification, comparing on a level footing for the first time the multitude of different bag-of-words encoding methods being proposed at the time the work was undertaken. Through this evaluation, a variety of hitherto neglected but important details were brought to the fore such as the effect of different normalisation, pooling strategies and feature maps on the performance of the pipeline. By fixing these details, a clearer picture of how different encoding schemes compared also emerged, and consequently cleared up a lot of confusion about these different methods. As a result, our paper based on the contents of this chapter has been widely cited in the community.

In Chapter 4 we compared the latest breed of Convolutional Network based methods in a similar manner. From this work, we demonstrated how it is possible to architect a highly performant classification pipeline using relatively simple methods providing that, once again, we get the details right. In this way, the best performing methods from the chapter set the state-of-the-art across several standard classification benchmarks compared to other published methods pre-trained on the same data, and was comparable even to methods using far more complicated architectures and techniques. We also demonstrated how fine-tuning over the target dataset could be used to boost the performance of our deep features, experimentally validated the importance of colour to both deep and shallow approaches, and showed how many of the techniques used to boost the performance of ConvNet-based approaches, such as data augmentation, could also be applied to shallow encoding methods such as the Fisher encoding with similarly positive effect. Finally, we showed how through reduction in the dimensionality of the last hidden layer in our networks, we could further reduce the already low-dimensional deep features to obtain incredibly compact representations with only a very minor drop

in performance, making them particularly amenable for use in the on-the-fly retrieval setting.

In Chapter 5 we showed how shallow image encodings could be used as the basis of a large-scale system for visual search over unannotated datasets. By making use of visual training data collected from the web in an on-demand manner, models can be learnt on-the-fly and at speed by using fast linear classifiers which capitalise on the descriptive power of state-of-the-art bag-of-words encodings. We also explored how such encodings could be made more compact to speed up ranking over datasets of millions of images, and conducted an in-depth evaluation of their characteristics and failure modes when used in a large-scale retrieval setting. Finally, we proposed a novel cascade approach which not only enabled us to speed up the ranking stage further, but also resulted in a boost in retrieval performance providing important insights into how shallow encoding methods operate.

In Chapter 6 we explored the application of deep features to the on-the-fly pipeline, and harnessing the amenability of such features to GPU computation proposed a novel, highly parallel GPU-based architecture for visual search. Using this architecture it is possible to both generate models and apply them to rank large datasets simultaneously, with the incremental training of a linear classification stage undertaken as additional training data sourced from the web becomes available. As a result of both this accelerated architecture and the state-of-the-art retrieval performance provided by ConvNet-based features, accurate retrieval for visual categories of which the system has no prior knowledge can be undertaken across unannotated datasets of millions of images within seconds. We further demonstrated the system over a dataset of 5M+ images sourced

from six years of BBC News broadcasts.

8.2 Outlook

The following perspectives for extension of the work presented in this thesis seem worth investigating:

Attribute-based search. One idea mentioned in the introduction, but only briefly touched upon in this thesis, is that of attribute-based refinement of visual search queries. We explore query refinement by colour in Section 7.1, but a more developed system might provide the ability to iteratively extend a query by setting constraints on shape or other higher-level appearance characteristics, such as whether the images to be retrieved were taken during the day or at night, for example. Another direction might be to explore refinement of results by visual style, an idea explored in Karayev *et al.* [70].

Object-level retrieval. In this thesis we have treated visual search as a retrieval problem. That is to say, given an image we are simply concerned whether it contains the query category or not. A related problem is that of detection – or the task of not only determining the presence or absence of a category, but also its precise location within the image frame. Although there are very many object 'categories' which our present system is capable of handling for which the notion of a precise location is ill-defined or meaningless (*e.g.* scene type or more abstract categories such as 'cityscape' or 'gothic architecture') for those categories where a distinct location is defined, taking

advantage of this additional information during either learning or ranking may help to improve retrieval performance further. One promising initial direction may be the investigation of combination of fast retrieval techniques with the 'objectness' measures of Alexe *et al*. [2], or the fast detection algorithm proposed by Aytar and Zisserman [8]. In addition, given the recent growth in wearable computing, the ability to both localise and identify things in the real-world at an object level is likely to become increasingly important as an end unto itself, possibly in combination with augmented reality applications or as a first step to querying a large-scale visual search system similar to the one we have developed.

Mobile retrieval. With the computational power of the latest generation of smartphones such as the iPhone 5S and 6 bringing them into the realm of close-to desktop performance for the first time. At the same time, as shown in Chapter 6, with the introduction of powerful ConvNet-based approaches to feature computation it is possible to develop visual search systems which are capable of running and great speed and retrieval performance on, at best, commodity hardware. Connecting these two trends, we believe the future scope for deploying powerful vision algorithms across a large number previously inconceivable platforms, such as mobile, is significant and only beginning to be explored.

In this thesis we have explored how common approaches for large-scale image classification can be adapted for the problem of real-time category search in web-scale datasets. Since beginning this work, and with the recent re-introduction into the community of powerful ConvNet-based methods, the potential for the application of computer vision

techniques to create truly practical visual search systems has only grown, offering both great challenges and opportunities for further work, both in the research community and beyond.

Bibliography

[1] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast retina keypoint. In *Proc. CVPR*, pages 510–517, 2012. 39

[2] B. Alexe, T. Deselaers, and V. Ferrari. Measuring the objectness of image windows. *IEEE PAMI*, 34(11):2189–2202, Nov 2012. 163

[3] R. Aly, R. Arandjelović, K. Chatfield, M. Douze, B. Fernando, Z. Harchaoui, K. McGuinness, N. E. O'Conner, D. Oneata, O. M. Parkhi, D. Potapov, J. Revaud, C. Schmid, J. Schwenninger, D. Scott, T. Tuytelaars, J. Verbeek, H. Wang, and A. Zisserman. The AXES submissions at trecvid 2013. In *TREC 2013 Video Retrieval Evaluation*, 2013. 10

[4] R. Aly, K. McGuinness, S. Chen, N. E. O'Connor, K. Chatfield, O. M. Parkhi, R. Arandjelović, A. Zisserman, B. Fernando, T. Tuytelaars, J. Schwenninger, D. Oneata, M. Douze, J. Revaud, D. Potapov, H. Wang, Z. Harchaoui, J. Verbeek, and C. Schmid. AXES at TRECVID 2012: KIS, INS, and MED. In *TREC 2012 Video Retrieval Evaluation*, 2012. 10

[5] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Comm. ACM*, 2008. 37

[6] R. Arandjelović and A. Zisserman. Three things everyone should know to improve object retrieval. In *Proc. CVPR*, 2012. 61, 74, 81

[7] R. Arandjelović and A. Zisserman. All about VLAD. In *Proc. CVPR*, 2013. 82

[8] Y. Aytar and A. Zisserman. Immediate, scalable object category detection. In *Proc. CVPR*, 2014. 163

[9] M. Baktashmotlagh, M.T. Harandi, B.C. Lovell, and M. Salzmann. Unsupervised domain adaptation by domain invariant projection. In *Proc. ICCV*, 2013. 43

[10] K. Barnard and D. Forsyth. Learning the semantics of words and pictures. In *Proc. ICCV*, volume 2, pages 408–415, 2001. 13

[11] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. *CVIU*, 110(3):346–359, 2008. 54

[12] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded up robust features. In *Proc. ECCV*, May 2006. 15

[13] A. Berg, J. Deng, and L. Fei-Fei. Large scale visual recognition challenge (ILSVRC), 2010. 19, 31, 48, 105, 107

[14] T. L. Berg and D. A. Forsyth. Animals on the web. In *Proc. CVPR*, 2006. 42

[15] A. Bergamo and L. Torresani. Exploiting weakly-labeled web images to improve object classification: a domain adaptation approach. In *NIPS*, 2010. 43

[16] A. Bergamo and L. Torresani. Meta-class features for large-scale object categorization on a budget. *Proc. CVPR*, 2012. 41

[17] A. Bergamo and L. Torresani. Classemes and other classifier-based features for efficient object categorization. *IEEE PAMI*, 36(10):1988–2001, Oct 2014. 41

[18] A. Bergamo, L. Torresani, and A. Fitzgibbon. PiCoDes: Learning a compact code for novel-category recognition. In *NIPS*, pages 2088–2096, 2011. 41, 102

[19] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. *JMLR*, 3:993–1022, Jan 2003. 13

[20] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary robust independent elementary features. In *Proc. ECCV*, 2010. 15, 39

[21] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: image segmentation using expectation-maximization and its application to image querying. *IEEE PAMI*, 24(8):1026–1038, Aug 2002. 13

[22] C. C. Chang and C. J. Lin. *LIBSVM: a library for support vector machines*, 2001. 72

[23] M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. 37

[24] K. Chatfield. Encoding Methods Evaluation Toolkit. `http://www.robots.ox.ac.uk/~vgg/software/enceval_toolkit/`, 2011. 74

[25] K. Chatfield, R. Arandjelović, O. Parkhi, and A. Zisserman. On-the-fly learning for visual search of large-scale image and video datasets. *International Journal of Multimedia Information Retrieval*, 2014. 10

[26] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *Proc. BMVC.*, 2011. 10, 78, 83

[27] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *Proc. BMVC.*, 2014. 10

[28] K. Chatfield, K. Simonyan, and A. Zisserman. Efficient on-the-fly category retrieval using ConvNets and GPUs. In *Proc. ACCV*, Lecture Notes in Computer Science. Springer, 2014. 10

[29] K. Chatfield and A. Zisserman. VISOR: Towards on-the-fly large-scale object category retrieval. In *Proc. ACCV*, Lecture Notes in Computer Science. Springer, 2012. 10

[30] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. H. S. Torr. BING: Binarized normed gradients for objectness estimation at 300fps. In *Proc. CVPR*, 2014. 96

[31] I. J. Cox, M. Miller, T. Minka, T. Papathomas, and P. N. Yianilos. The bayesian image retrieval system, PicHunter: Theory, implementation and psychophysical experiments. *IEEE Transactions on Image Processing*, 2000. 12

[32] N. Dalal and B Triggs. Histogram of Oriented Gradients for Human Detection. In *Proc. CVPR*, volume 2, pages 886–893, 2005. 15

[33] R. Datta, D. Joshi, J. Li, and J. Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2), 2008. 13

[34] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proc. CVPR*, 2009. 19, 30, 42, 48, 77, 80, 105, 118

[35] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. *CoRR*, abs/1310.1531, 2013. 32, 77, 80, 93

[36] M. Everingham, A. S. M. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *IJCV*, 111(1):98–136, 2015. 67

[37] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes (VOC) challenge. *IJCV*, 88(2):303–338, 2010. 19, 46, 52, 53, 68, 77, 105

[38] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *JMLR*, 9:1871–1874, 2008. 60

[39] A. Farhadi, I. Endres, D. Hoiem, and D. Forsyth. Describing objects by their attributes. In *Proc. CVPR*, pages 1778–1785, 2009. 40

[40] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. In *IEEE CVPR Workshop of Generative Model Based Vision*, 2004. 45

[41] P. Felzenszwalb, D. Mcallester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *Proc. CVPR*, 2008. 36

[42] R. Fergus, L. Fei-Fei, P. Perona, and A. Zisserman. Learning object categories from Google's image search. In *Proc. ICCV*, 2005. 42, 120

[43] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 1980. 29

[44] B. V. Funt and G. D. Finlayson. Color constant color indexing. *IEEE PAMI*, 17(5):522–529, May 1995. 12

[45] T. Furon and H. Jégou. Using extreme value theory for image detection. Technical report, Feb 2013. 157

[46] P. V. Gehler and S. Nowozin. On feature combination for multiclass object classification. In *Proc. ICCV*, pages 221–228, 2009. 21

[47] T. Gevers and A. W. M. Smeulders. Content-based image retrieval by viewpoint-invariant color indexing. *Image and vision computing*, 17:475–488, 1999. 12

[48] T. Gevers and A. W. M. Smeulders. PicToSeek: combining color and shape invariant features for image retrieval. *IEEE Transactions on Image Processing*, 9(1):102–119, Jan 2000. 12

[49] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proc. CVPR*, 2014. 77, 80

[50] B. Gong, Y. Shi, F. Sha, and K. Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *Proc. CVPR*, 2012. 43

[51] R. Gopalan, Ruonan Li, and R. Chellappa. Domain adaptation for object recognition: An unsupervised approach. In *Proc. ICCV*, 2011. 43

[52] K. Goto, K. Kidono, Y. Kimura, and T. Naito. Pedestrian detection and direction estimation by cascade detector with multi-classifiers utilizing feature interaction descriptor. In *Proc. IEEE Symposium on Intelligent Vehicles*, pages 224 –229, 2011. 104

[53] K. Grauman and T. Darrel. The pyramid match kernel: Discriminative classification with sets of image features. In *Proc. ICCV*, 2005. 57

[54] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. Technical Report 7694, California Institute of Technology, 2007. 46

[55] H. Grimmett, R. Paul, R. Triebel, and I. Posner. Knowing when we don't know: Introspective classification for mission-critical decision making. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4531–4538, 2013. 158

[56] C. G. Harris and M. Stephens. A combined corner and edge detector. In *Proc. Alvey Vision Conf.*, pages 147–151, 1988. 15

[57] K. He, A. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *Proc. ECCV*, 2014. 95, 96

[58] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. 31

[59] D. H. Hubel and T. N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160:106–154, 1962. 29

[60] M. J. Huiskes, B. Thomee, and M. S. Lew. New trends and ideas in visual concept detection: The mir flickr retrieval evaluation initiative. In *MIR '10: Proceedings of the 2010 ACM International Conference on Multimedia Information Retrieval*, pages 527–536, 2010. 48

[61] V. Jain and E. Learned-Miller. Online domain adaptation of a pre-trained cascade of classifiers. In *Proc. CVPR*, 2011. 43

[62] H. Jégou and O. Chum. Negative evidences and co-occurrences in image retrieval: the benefit of PCA and whitening. In *Proc. ECCV*, 2012. 74

[63] H. Jégou, M. Douze, and C. Schmid. Hamming embedding and weak geometric consistency for large scale image search. In *Proc. ECCV*, pages 304–317, 2008. 103

[64] H. Jégou, M. Douze, and C. Schmid. On the burstiness of visual elements. In *Proc. CVPR*, Jun 2009. 82

[65] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE PAMI*, 2011. 39

[66] H. Jégou, T. Furon, and J.-J. Fuchs. Anti-sparse coding for approximate nearest neighbor search. In *Proc. ICASSP*, pages 2029–2032, 2012. 37

[67] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid. Aggregating local images descriptors into compact codes. *IEEE PAMI*, 2012. 27, 28, 75, 111

[68] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`, 2013. 80, 85, 146

[69] T. Joachims. Training linear SVMs in linear time. In *Proc. KDD*, pages 217–226, 2006. 59

[70] S. Karayev, M. Trentacoste, H. Han, A. Agarwala, T. Darrell, A. Hertzmann, and H. Winnemoeller. Recognizing image style. In *Proc. BMVC.*, 2014. 162

[71] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, pages 1106–1114, 2012. 30, 31, 75, 77, 78, 80, 81, 83, 84, 85, 86, 88, 89, 94, 126

[72] N. Kumar and S. Seitz. Photo recall: Using the Internet to label your photos. In *2nd Workshop on Web-scale Vision and Social Media (VSM) at CVPR 2014*, June 2014. 44

[73] S. Lazebnik, C. Schmid, and J Ponce. Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories. In *Proc. CVPR*, 2006. 57, 82

[74] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989. 30, 76

[75] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 30, 126

[76] S. Leutenegger, M. Chli, and R. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Proc. ICCV*, pages 2548–2555, 2011. 39

[77] L. Li and F. Li. OPTIMOL: Automatic online picture collection via incremental model learning. *IJCV*, 88(2):147–168, 2010. 42, 120

[78] L. Li, H. Su, L. Fei-fei, and E. P. Xing. Object bank: A high-level image representation for scene classification & semantic feature sparsification. In *NIPS*. 2010. 36

[79] T. Lindeberg. Feature detection with automatic scale selection. *IJCV*, 30(2):77–116, 1998. 15

[80] Y. Liu, D. Xu, I. W. Tsang, and J. Luo. Using large-scale web data to facilitate textual query based retrieval of consumer photos. In *Proceedings of the 17th ACM International Conference on Multimedia*, MM '09, pages 55–64, 2009. 42

[81] D. Lowe. Object recognition from local scale-invariant features. In *Proc. ICCV*, pages 1150–1157, Sep 1999. 62

[82] D. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004. 14, 15, 53

[83] B. S. Manjunath and W. Y. Ma. Texture features for browsing and retrieval of image data. *pami*, 18(8):837–842, 1996. 12

[84] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. 47

[85] G. McLachlan and D. Peel. *Finite Mixture Models*. Wiley, 2000. 25

[86] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE PAMI*, 2004. 15

[87] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *IJCV*, 65(1/2):43–72, 2005. 15

[88] N. Murray and F. Perronnin. Generalized max pooling. In *Proc. CVPR*, pages 2473–2480, 2014. 58

[89] C.W. Niblack, R. Barber, W. Equitz, M. D. Flickner, E. H. Glasman, D. Petkovic, P. Yanker, C. Faloutsos, and G. Taubin. QBIC project: querying images by content, using color, texture, and shape, 1993. 12

[90] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Proc. CVPR*, pages 2161–2168, 2006. 103, 116

[91] T. Ojala, M. Pietikainen, and T. Maenpaa. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE PAMI*, 24(7), 2002. 39

[92] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *IJCV*, 2001. 34

[93] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and Transferring Mid-Level Image Representations using Convolutional Neural Networks. In *Proc. CVPR*, 2014. 77, 80, 95, 96

[94] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Weakly supervised object recognition with convolutional neural networks. Technical Report HAL-01015140, INRIA, 2014. 78, 95, 96

[95] M. Paulin, J. Revaud, Z. Harchaoui, F. Perronnin, and C. Schmid. Transformation Pursuit for Image Classification. In *Proc. CVPR*, 2014. 77

[96] A. Pentland, R. W. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *IJCV*, 18:233–254, 1996. 12

[97] F. Perronnin, Z. Akata, Z. Harchaoui, and C. Schmid. Towards good practice in large-scale learning for image classification. In *Proc. CVPR*, pages 3482–3489, 2012. 78

[98] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier. Large-scale image retrieval with compressed fisher vectors. In *Proc. CVPR*, 2010. 60

[99] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *Proc. ECCV*, 2010. 24, 25, 51, 54, 60, 61, 64, 65, 75, 79, 83, 92

[100] J. Philbin. *Scalable Object Retrieval in Very Large Image Collections*. PhD thesis, University of Oxford, 2010. 16

[101] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *Proc. CVPR*, 2007. 73, 103

[102] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Lost in quantization: Improving particular object retrieval in large scale image databases. In *Proc. CVPR*, 2008. 22

[103] R. W. Picard. Digital libraries: Meeting place for high-level and low-level vision. In *Proc. ACCV*, 1995. 13

[104] T. Quack, U. Monich, L. Thiele, and B. S. Manjunath. Cortina: A system for large-scale, content-based web image retrieval. In *Proc. ACMM*, Oct 2004. 12

[105] M. Rastegari, C. Fang, and L. Torresani. Scalable object-class retrieval with approximate and top-k ranking. In *Proc. ICCV*, 2011. 41, 110

[106] A. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. *CoRR*, abs/1403.6382, 2014. 77, 80, 95, 96

[107] Fritz M. Rematas, K. and T. Tuytelaars. The pooled NBNN kernel: Beyond Image-to-Class and Image-to-Image. In *Proc. ACCV*, Lecture Notes in Computer Science. Springer, 2012. 20

[108] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski. Orb: An efficient alternative to SIFT or SURF. In *Proc. ICCV*, pages 2564–2571, 2011. 39

[109] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986. 30

[110] J. Sánchez and F. Perronnin. High-dimensional signature compression for large-scale image classification. In *Proc. CVPR*, 2011. 39

[111] J. Sánchez, F. Perronnin, and T. Emídio de Campos. Modeling the spatial layout of images beyond spatial pyramids. *Pattern Recognition Letters*, 33(16):2216–2223, 2012. 57, 78, 81, 82, 93

[112] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *IJCV*, 105(3):222–245, 2013. 56

[113] W. J. Scheirer, N. Kumar, P. N. Belhumeur, and T. E. Boult. Multi-attribute spaces: Calibration for attribute fusion and similarity search. In *Proc. CVPR*, June 2012. 156

[114] F. Schroff, A. Criminisi, and A. Zisserman. Harvesting Image Databases from the Web. *IEEE PAMI*, 33(4):754–766, Apr 2011. 43

[115] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *Proc. ICLR*, 2014. 77, 80, 85, 88, 94, 95

[116] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated subgradient SOlver for SVM. In *Proc. ICML*, volume 227, 2007. 145

[117] S. Singh, A. Gupta, and A. A. Efros. Unsupervised discovery of mid-level discriminative patches. In *Proc. ECCV*, 2012. 36

[118] J. Sivic and A. Zisserman. Video Google: A text retrieval approach to object matching in videos. In *Proc. ICCV*, volume 2, pages 1470–1477, 2003. 16, 42

[119] D. Slater and G. Healey. The illumination-invariant recognition of 3D objects using local color invariants. *IEEE PAMI*, 18(2):206–210, Feb 1996. 12

[120] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of early years. *IEEE PAMI*, 22(12):1349–1380, 2000. 13

[121] A. R. Smith and J. F. Blinn. Blue screen matting. In *Proc. ACM SIGGRAPH*, pages 259–268, 1996. 12

[122] M. J. Swain and D. H. Ballard. Color indexing. *IJCV*, 7(1):11–32, Nov 1991. 12

[123] E. Tola, V. Lepetit, and P. Fua. A fast local descriptor for dense matching. In *Proc. CVPR*, 2008. 15

[124] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: a large dataset for non-parametric object and scene recognition. *IEEE PAMI*, 2008. 34

[125] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large databases for recognition. In *Proc. CVPR*, 2008. 34, 42

[126] L. Torresani, M. Szummer, and A. Fitzgibbon. Efficient object category recognition using classemes. In *Proc. ECCV*, pages 776–789, sep 2010. 41

[127] T. Tuytelaars and L. Van Gool. Matching widely separated views based on affine invariant regions. *IJCV*, 59(1):61–85, 2004. 15

[128] K. E. A. van de Sande, T. Gevers, and C. G. M. Snoek. Evaluating color descriptors for object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9):1582–1596, 2010. 53

[129] J. C. van Gemert, J. M. Geusebroek, C. J. Veenman, and A. W. M. Smeulders. Kernel codebooks for scene categorization. In *Proc. ECCV*, 2008. 22, 64

[130] A. Vedaldi and B. B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008. 62

[131] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *Proc. CVPR*, 2010. 59, 61, 63

[132] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. *IEEE PAMI*, 2011. 82

[133] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. In *Proc. CVPR*, 2010. 24, 51, 65

[134] J. Z. Wang, G. Wiederhold, O. Firschein, and S. Xin Wei. Content-based image indexing and searching using Daubechies' wavelets. *International Journal on Digital Libraries*, 1(4):311–328, 1998. 12

[135] Y. Wei, W. Xia, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan. CNN: Single-label to multi-label. *CoRR*, abs/1406.5726, 2014. 78, 95, 96

[136] S. Yan, J. Dong, Q. Chen, Z. Song, Y. Pan, W. Xia, H. Zhongyang, Y. Hua, and S. Shen. Generalized hierarchical matching for subcategory aware object classification. In *The PASCAL Visual Object Classes Challenge Workshop*, 2012. 96

[137] J. Yang, K. Yu, and T. Huang. Supervised translation-invariant sparse coding. In *Proc. CVPR*, 2010. 51

[138] K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In *NIPS*, 2009. 51

[139] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013. 32, 80, 81, 85, 88, 94, 95, 96

[140] J. Zhang, M. Marszałek, S. Lazebnik, and Schmid C. Local features and kernels for classification of texture and object categories: a comprehensive study. *IJCV*, 2007. 59

[141] Y. Zhang, C. Zhu, S. Bres, and L. Chen. Encoding local binary descriptors by bag-of-features with hamming distance for visual object categorization. In *Advances in Information Retrieval*, Lecture Notes in Computer Science. 2013. 39

[142] Y.-T. Zheng, M. Zhao, Y. Song, H. Adam, U. Buddemeier, A. Bissacco, F. Brucher, T.-S. Chua, and H. Neven. Tour the world: building a web-scale landmark recognition engine. In *Proc. CVPR*, 2009. 15

[143] X. Zhou, K. Yu, T. Zhang, and T. S. Huang. Image classification using super-vector coding of local image descriptors. In *Proc. ECCV*, 2010. 27, 51, 65, 72