Large-Scale Image Categorization with Explicit Data Embedding

Florent Perronnin, Jorge Sánchez and Yan Liu
Xerox Research Centre Europe (XRCE)
Firstname.Lastname@xrce.xerox.com

Abstract

Kernel machines rely on an implicit mapping of the data such that non-linear classification in the original space corresponds to linear classification in the new space. As kernel machines are difficult to scale to large training sets, it has been proposed to perform an explicit mapping of the data and to learn directly linear classifiers in the new space. In this paper, we consider the problem of learning the data and to learn directly linear classifiers in the new space. Since we know that there exist linear classifiers which perform well in this new space, it is extremely tempting to perform an explicit (possibly approximate) mapping of the data and learn linear classifiers directly in this new space.

One of the leading approaches to image categorization has been to describe images with bag-of-visual-words (BOV) histograms and to classify them using Support Vector Machine classifiers. We experiment with three approaches to BOV embedding: 1) kernel PCA (kPCA) [15], 2) a modified kPCA we propose for additive kernels and 3) random projections for shift-invariant kernels [14]. We report experiments on 3 datasets: Caltech101, VOC07 and ImageNet. An important conclusion is that simply square-rooting BOV vectors – which corresponds to an exact mapping for the Bhattacharyya kernel – already leads to large improvements, often quite close to the best results obtained with additive kernels. Another conclusion is that, although it is possible to go beyond additive kernels, the embedding comes at a much higher cost.

1. Introduction

Within the computer vision community, annotated data for training and evaluating image categorization algorithms has long been viewed as a scarce resource. This is because the traditional approach to building such datasets – querying image search engines such as Google Images (images.google.com) or photo sharing websites such as Flickr (www.flickr.com) and then post-processing the results manually – is a time consuming process. As a consequence, the datasets which have been (and are still currently) employed to benchmark image categorization algorithms are fairly modest in size. For instance, CalTech101 [7] and PASCAL VOC07 [5], two of the most popular benchmarks, contain less than 10k images each.

However, this perception is drastically changing as large quantities of quality labeled data are becoming available. The ImageNet dataset (www.image-net.org) is certainly the best example of this trend [4]. Leveraging the Amazon Mechanical Turk (AMT: www.mturk.com) for the manual post-processing part, a dataset of approximately 10M annotated images corresponding to almost 15K concepts was made available to the research community. As a consequence, the problem is shifting from “how to learn a category from a single image?” [7] to “how to handle these large quantities of data?” [20, 12, 10].

One of the leading approaches to image categorization has been to describe images with bag-of-visual-words (BOV) histograms and to classify them using Support Vector Machines (SVM) classifiers (see [3, 22, 19] for a few examples). While non-linear SVMs are perfectly suited to the small datasets which are typically employed, they do not scale well with the number N of training samples: between $O(N^2)$ and $O(N^3)$. This is to be contrasted with linear SVMs which can be learned in $O(N)$ using, for instance, a cutting-plane algorithm [9] or Stochastic Gradient Descent (SGD) [16]. It is even shown in [17] that, to achieve a target loss, the runtime of the SGD solver PEGASOS [16] decreases with the training set size (see also section 2 of [17] for additional background on the cost of training non-linear and linear SVMs). This is a very compelling argument for using linear SVMs. Yet, it has been repeatedly reported that non-linear SVMs outperform linear SVMs for image categorization (see e.g. [13, 20, 12, 19]).

However, non-linear kernel classifiers are linear-classifiers. Indeed, if K is a positive semi-definite (psd) kernel, there exists an implicit mapping $\varphi$ in a new (possibly infinite-dimensional) space such that $K(x,z) = \varphi(x)\varphi(z)$. Let $X = \{x_i, i = 1...N\}$ denote the set of training samples. The kernel classifier $f(z) = \sum_{i=1}^{N} a_i K(z, x_i) + b$ can be re-written as $f(z) = (\sum_{i=1}^{N} a_i \varphi(x_i))' \varphi(z) + b$ which is a linear classifier in the new space. Since we know that there exist linear classifiers which perform well in this new space, it is extremely tempting to perform an explicit (possibly approximate) mapping of the data and learn linear classifiers directly in this new space.

Several approaches have been proposed along this line.
Kernel Principal Component Analysis (kPCA) [15], which was successfully applied in [21], can be employed with any kernel. [14] proposed an embedding based on random projections (RP s) for shift-invariant kernels. Recently, [12] proposed an embedding for the intersection kernel. We note that the accuracy of an embedding approach depends directly on the accuracy of the kernel which is approximated: good kernels lead to good embeddings as verified in our experiments.

In this article we explore several approaches to performing explicit embedding on various kernels. More precisely, our contributions are the following ones:

- We show that square-rooting BOV vectors, which corresponds to an exact mapping in the case of the Bhattacharyya kernel, already leads to large improvements.
- To go beyond this simple strategy, we propose add-kPCA: a novel kPCA-based embedding for additive kernels. We demonstrate that it leads to additional improvements at a very affordable cost.
- We go one step further and experiment with embedding for non-additive kernels. We show that the fairly small gain in classification accuracy over additive kernels is obtained at the expense of a very significant increase of the computational cost.
- Using ImageNet data, we show experimentally that we continuously improve the classification accuracy using up to 10,000 training images per class. This clearly justifies the use of very large training sets.

The remainder of this article is organized as follows. In section 2, we discuss the choice of kernels for BOV histograms (i.e. multinomials). In section 3, we briefly review kPCA. In section 4, we show how to adapt this framework to additive kernels. In section 5, we review embedding with RPs for shift-invariant kernels. In section 6, we present experimental results on 3 datasets: Caltech101, PASCAL VOC07 and ImageNet. Due to space limitations, we only give a very brief introduction to the SGD solver we use in our experiments in Appendix A. The remainder of the relevant literature will be reviewed throughout the paper.

2. Kernels for BOV Histograms

2.1. Additive kernels

Additive kernels can be written as $K(x, z) = \sum_{d=1}^{D} k(x(d), z(d))$ where $x(d)$ denotes the $d$-th dimension of vector $x$. They have gained considerable popularity since [13] showed that the classification of a sample by an additive kernel SVM can be done approximately at a cost which is independent of the number of support vectors. This is because the classification function $f(z) = \sum_{i=1}^{N} a_i K(x_i, z) + b$ can be rewritten as $f(z) = \sum_{d=1}^{D} f_d(z(d)) + b$ with $f_d(z(d)) = \sum_{i=1}^{N} a_i k(z(d), x_i(d))$ and each of the $1$-D $f_d$ functions can be approximated by piecewise-constant or -linear functions. Among the most popular additive kernels are the intersection kernel $K_{int}(x, z) = \sum_{d=1}^{D} \min(x(d), z(d))$ and the $\chi^2$ kernel $K_{\chi^2}(x, z) = 2 \sum_{d=1}^{D} \frac{x(d)z(d)}{x(d) + z(d)}$, both of which are psd.

Another kernel for probability distributions is the Bhattacharyya kernel. While it has been successfully applied to continuous distributions [8, 6], it remains under-exploited in the case of BOV histograms. In the discrete case $K_{bha}(x, z) = \sum_{d=1}^{D} \sqrt{x(d)z(d)}$. $K_{bha}$ can be directly related to $K_{\chi^2}$. Indeed, we have the following bounds:

$$K_{\chi^2}(x, z) \leq K_{bha}(x, z) \leq \frac{1}{2} (1 + K_{\chi^2}(x, z))$$  (1)

and if $x \approx z$, $K_{bha}(x, z) \approx \frac{1}{2} (1 + K_{\chi^2}(x, z))$. Now, going back to our data embedding problem, we see that in the case of $K_{bha}$ the mapping is trivial: $\varphi(z) = \sqrt{z}$ (where the square-root of a vector should be understood as a term-by-term operation). Hence, in the case of $K_{bha}$ the mapping can be done exactly at (almost) zero cost, simply by square-rooting the BOV vectors.

2.2. Exponential kernels

If $K$ is a psd kernel, $\exp(\gamma K)$ with $\gamma > 0$ is guaranteed to be a psd kernel. Hence, all the previous kernels can be exponentiated: $K_{\exp} = \exp(\gamma (K_{int} - 1))$, $K_{\exp}^{\chi^2} = \exp(\gamma (K_{\chi^2} - 1))$ and $K_{\exp}^{bha} = \exp(\gamma (K_{bha} - 1))$. The $-1$ in the kernel is to ensure that the exponentiated kernel values are upper-bounded by 1. Also, it makes these kernels easier to relate to well-know kernels: $K_{int}^{\exp}$ is the Laplacian kernel, $K_{\exp}^{\chi^2}$ is a popular kernel used for instance in [22, 19], and $K_{\exp}^{bha}$ is the rbf kernel on the square-rooted vectors. The parameter $\gamma$ controls the neighborhood size around the training samples and therefore the non-linearity of the classifier. In all our experiments we set $\gamma$ to the inverse of the average non-exponentiated value [22].

Recently, [19] reported on an object segmentation task that $K_{\exp}^{\chi^2}$ outperformed $K_{\chi^2}$. In section 6, we systematically compare the accuracy of the additive kernels with their exponential counterparts and confirm the superiority of exponential kernels.

3. Embedding with Kernel PCA

We just provide a brief introduction to embedding with kPCA. For more background on kPCA, please refer to [15]. Let $\{x_i, i = 1 \ldots M\}$ be a set of samples in $\mathbb{R}^D$ provided to learn the embedding. Given a kernel $K : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$, we seek a mapping $\varphi : \mathbb{R}^D \to \mathbb{R}^F$ such that $K(x_i, x_j) \approx \varphi(x_i)^\top \varphi(x_j)$, $\forall (i, j)$. A possible criterion to find an optimal
The cost of computing the kernel matrix $K$ is in $O(M^2D)$. The cost of the eigenvalue decomposition – typically based on a Cholesky decomposition – is in $O(M^3)$. The cost of embedding a new sample is in $O(M(D + E))$. In all our kPCA experiments, we set $M = E$.

4. Embedding for Additive Kernels: addkPCA

While kPCA can be applied as is to additive kernels, it does not exploit their additive structure. We now propose addkPCA: an efficient kPCA embedding for additive kernels.

4.1. Learning the embedding

As noted in [12], a possible way to find an embedding $\varphi$ for additive kernels is to find separate embeddings $\varphi_d$ for each dimension $d$ and then to concatenate them. Therefore, we propose to apply the kPCA framework separately in each dimension. Given an additive kernel $K(x,z) = \sum_{d=1}^{D} k_d(x(z), z(d))$ and a set of samples $\{x_i, i = 1 \ldots M\}$, the embedding is learned as follows:

1) For each dimension $d$:
   a) Compute the $M \times M$ kernel matrix $k_d$ with entries $k_d(x_i(d), x_j(d))$
   b) Compute the $F$ largest eigenvalues $\sigma_{d,1}^2, \ldots, \sigma_{d,F}^2$ of $k_d$ (e.g. $F = 10$) and their associated eigenvectors $\psi_{d,1}, \ldots, \psi_{d,F}$.

2) Sort the $D \times F$ eigenvalues $\sigma_{d,e}^2$ for $d = 1 \ldots D$ and $e = 1 \ldots F$ and keep the $E$ largest ones with their associated eigenvectors.

Although the function $k$ is the same for all dimensions, we learn different embedding functions $\varphi_{d,e}$ in each dimension because the distribution of samples is different in each dimension. Also, a variable number of functions $\varphi_{d,e}$ is selected automatically by the algorithm in each dimension $d$ (c.f. step 2)). This enables to put more emphasis on those dimensions which contain more energy.

The cost of computing the $D$ kernel matrices is in $O(M^2D)$ and the cost of the $D$ eigenvector decompositions is in $O(M^3D)$. The crucial difference between kPCA and addkPCA is that, in the first case, one approximates a kernel $k : \mathbb{R}^D \times \mathbb{R}^D \rightarrow \mathbb{R}$ while, in the second case, one approximates $D$ times a kernel $k : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. Since $K$ operates in a much higher dimensional space than $k$, the number $M$ of vectors necessary to learn a good approximation of $K$ is typically much larger than for $k$. In our experiments, we found that $M = 128$ was sufficient to learn addkPCA as opposed to several thousands for kPCA. Learning addkPCA on BOV histograms of dimensionality $D = 4,000$ with $M = 128$ takes less than 30s on a single CPU of a 3GHz Xeon machine.

We show in Figure 1 the first eigenfunctions in the case of $K_{\text{chi2}}$. An important question is: how many dimensions $E$ will typically need to obtain a good embedding? We show that $K_{\text{chi2}}$ can already be correctly approximated using a single eigenfunction. We will show in section 6 that with $E = 2D$ (i.e. by doubling the dimensionality of our representations) we can do as well as additive kernel SVMs.

4.2. Embedding a new sample

We can embed a sample using the Nyström approximation:

$$\varphi_{d,e}(z(d)) = \frac{k(z(d), \cdot)\psi_{e,d}}{\sigma_{e,d}}$$

with $k(z(d), \cdot) = [k(z(d), x_1(d)), \ldots, k(z(d), x_M(d))]$. In this case, the embedding cost is in $O(M(D + E))$.

However, we can make the embedding cost independent of $M$ using the same trick as [13]. We split each of the $D$ dimensions into a set of $B$ bins (by regularly sampling the interval between the smallest and largest values). Let $c_{d,b}$ be the center of the $b$-th bin in dimension $d$. We pre-compute the values $\varphi_{d,e}(c_{d,b})$ for the $E$ dimensions of the embedding using Nyström. Given a new sample $z$, we compute for each dimension the bin index $(O(D))$ and then use a piecewise-constant or linear approximation of $\varphi_{d,e}(O(E))$.

4.3. Classification of a sample

We assume that we have learned a set of $C$ linear classifiers on the embedded samples: $f_c(z) = w_c^T \varphi(z) + b_c$. We
can compute \( f_c(\varphi(z)) \) in two different ways:

1. The simplest way is to perform the explicit embedding \( \varphi(z) \) \((O(D + E))\) and then do \( C \) dot-products \((O(CE))\).

2. Using -- again -- the trick of [13], we can rewrite the classifiers \( f_c \) as non-linear classifiers in the original space. \( f_c(z) = \sum_{d=1}^{D} f_{c,d}(z(d)) + b \) where each 1D function \( f_{c,d} \) can be approximated by binning the \( d \)-th dimension and then by using a piecewise-constant or -linear approximation. The cost of computing the bin indices in each dimension is in \( O(D) \) and the cost of the \( C \) classifications is in \( O(CE) \).

As already mentioned, to obtain top accuracy with our approach, we typically set \( E = 2D \) which may lead to think that the second option is cheaper. Actually, this is incorrect as the \( O() \) notation hides constants. For the first option, the computation of \( f_c(z) \) relies on dot-products which are very efficient operations while for the second option it relies on look-up table accesses which are much slower. In practice, using a C++ implementation and setting \( E = 2D \), we found that a standard dot-product was approximately 5 times faster than look-up tables. Using the SSE2\(^2\) implementation of the dot-product of [1], the speed-up was a factor of \( 15 \). Note that SSE2 instructions cannot speed-up look-up table accesses.

To learn the linear SVM for class \( c \) we use an SGD solver (c.f. Appendix A). At each iteration, the value \( f_c(x_i) \) is evaluated with a different training sample \( x_i \) (c.f. the term \( \delta_i \) in equation (12)). To train our linear classifiers, we also prefer the direct approach to computing \( f_c(x_i) \) (embedding + dot-product) over the approach based on look-up tables. This is much more efficient when the same sample is used multiple times by the SGD solver (5 times on average in our experiments).

### 4.4. Related work

Closest to our work is [12] which proposes an explicit embedding for the intersection kernel. Each dimension of the BOV vector is embedded into a \( B \)-dimensional space. Per dimension, only 2 values are non-zero. This leads to a \( BD \)-dimensional embedding with \( 2D \) non-zero values. As is the case of [12], addPCA is as accurate as the additive kernel SVM it approximates. The main advantages of the proposed approach with respect to [12] are the following ones:

- Our approach preserves data sparsity. Let \( z \) be a \( D \)-dimensional vector and let \( nz \) be the proportion of non-zero values in \( z \). According to equation (6), if \( z(d) = 0 \), then \( k(z(d),.) \) is the null vector (at least in the case of \( K_{\text{int}}, K_{\text{ch}2} \) or \( K_{\text{hha}} \)) and \( \varphi_{d,c}(z(d)) = 0 \). Hence, the proportion of non-zero values in \( \varphi(z) \) is approximately \( nz \) (this is only approximate because we keep a variable number of eigenvectors per dimension). This is to be contrasted with [12] where \( 2D \) values are non-zero (independently of \( nz \)).

- The training cost of [12] is in \( O(NBD) \) with \( B \) ranging from 30 to 100. Our training cost is in \( O(ED) \) with \( E = 2D \) typically.

- [12] reports that their work makes “training and testing an additive classifier only a small (constant factor) slower than training a linear classifier”. The classification as well as the SGD training in [12] are based on look-up table accesses while our approach is based on dot-products. We explained in the previous subsection that, on modern processors, the main bottleneck is often not the execution of the instructions but the access to the data.

\(^1\)On modern processors, the main bottleneck is often not the execution of the instructions but the access to the data. To reduce the access time, processors perform prefetching, i.e., when accessing the data at a given position in RAM, they also retrieve the data at subsequent positions. Prefetching results in a very significant speed-up provided that data is accessed sequentially. This is the case when accessing dot-products and this is generally not the case when accessing look-up tables.

\(^2\)Modern processors, support single instruction with multiple data (SIMD) commands. SSE2 instructions perform multiple simple operations, such as additions or multiplications, in parallel.
that the second type of operation is much more efficient: up to 15 times faster (hardly a “small” constant) with a good implementation of a dot-product.

[20], which proposes an algorithm to learn intersection kernel SVMs, is closely related to [12] and to addkPCA. The advantages of our approach with respect to [20] are the same as the advantages with respect to [12]: we preserve sparsity and classification as well as training are more efficient.

5. Embedding for Shift-Invariant Kernels

We provide a brief introduction to embedding with random projections (RPs) for shift-invariant kernels. For more details, please refer to [14].

Let \( K : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R} \) be a kernel. It is shift-invariant if it can be written under the form \( K(x, z) = K(x - z) \). For instance \( K_{int} \) is shift-invariant since \( K_{int}(x, z) = 1 - \frac{1}{2} |x - z| \) where \(|.|\) is the L1 norm. Consequently \( K_{exp}^{\text{int}} \) is also shift-invariant. Similarly, \( K_{exp}^{\text{int}}(x, z) \) can be rewritten as an rbf kernel: \( K_{exp}^{\text{int}}(x, z) = K_{rbf}(\sqrt{x - z}) \). Therefore, \( K_{exp}^{\text{int}} \) is shift-invariant in the space of the square-rooted BOV vectors. In our experiments, we will focus on \( K_{exp}^{\text{int}} \).

\( K(x - z) \) is psd if and only if it is the Fourier transform of a non-negative measure \( p \) (Bochner’s theorem). If \( K \) is properly scaled, \( p \) is a probability distribution. For instance if \( K \) is the rbf kernel, then \( p \) is the Gaussian distribution. We have:

\[
K(x - z) = \int_{\omega \in \mathbb{R}^D} p(\omega) e^{j\omega^T(x - z)} d\omega \tag{7}
\]

As \( K \) and \( p \) are real, we can replace \( e^{j\omega^T(x - z)} \) by \( \cos(\omega^T(x - z)) \). Introducing the 2D vector \( \zeta_\omega(x) = \{\cos(\omega^T x) \sin(\omega^T x)\} \), we have:

\[
K(x - z) = E_\omega [\zeta_\omega(x)^T \zeta_\omega(z)] \tag{8}
\]

In practice, to compute an embedding \( \phi : \mathbb{R}^D \to \mathbb{R}^E \), one draws a set of \( E/2 \) iid Gaussian vectors \( \omega_1, \ldots, \omega_{E/2} \in \mathbb{R}^D \) and the mapping is \( \phi(x) = \frac{1}{\sqrt{D}} [\cos(\omega_1^T x), \ldots, \cos(\omega_{E/2}^T x), \sin(\omega_1^T x), \ldots, \sin(\omega_{E/2}^T x)] \).

As \( E \to \infty \), \( \phi(x)^T \phi(z) \to K(x - z) \).

We highlight that this approach does not require any learning. The cost of embedding a vector is in \( O(DE) \).

6. Experiments

We extract SIFT features [11] on regular grids at multiple scales. We train visual codebooks of 4,000 visual words. An image is described by the L1-normalized histogram of visual-word occurrences \( D = 4,000 \). To learn linear SVMs, we use the SGD code available at [1] (c.f. Appendix A). To learn non-linear SVMs, we use libsvm [2].

\[
\begin{array}{cccccc}
\text{Linear SVM [1]} & 47.1 (4.6) \\
\text{Non-Linear SVM [2]} & \hline
\text{Additive kernels} & \text{Exponential kernels} \\
K_{bha} & K_{chi} & K_{int} & K_{exp}^{bha} & K_{exp}^{chi} & K_{exp}^{int} \\
55.3 & 56.2 & 55.8 & 57.3 & 57.1 & 55.5 \\
(1.6) & (1.8) & (1.6) & (1.5) & (1.6) & (1.4)
\end{array}
\]

Table 1. Caltech101 baseline (in % recognition). The standard deviation is given between parentheses.

6.1. Small-scale experiments

For the small-scale experiments, we focus on accuracy, not on the training time. Indeed, training non-linear SVMs on small datasets is not an issue. Our goal is to verify to which extent the 3 approaches to explicit data embedding can approximate the non-linear kernel classifiers.

Datasets: We experimented with two standard benchmarks: Caltech101 [7] and PASCAL VOC07 [5]. Caltech101 contains 8,677 images of 101 classes. We use the standard protocol which consists in training with 30 images per class and testing with the remaining images. The accuracy is measured in terms of the number of images assigned to the correct class. The final score is the average over the 101 classes. We repeat the experiments 10 times with different training / testing folds. To tune the SVM regularization parameters, we trained the parameters on 25 images of the first training fold and validated the results on the last 5 images. These parameters were used for the 10 folds.

VOC07 contains 9,963 images of 20 classes. We used the standard protocol which consists in training on the train+val sets and testing on the test set. Classification accuracy is measured using Average Precision (AP). We report the average over the 20 classes. To tune the SVM regularization parameters, we trained them on the train set and used the val set for validation.

Baseline. We report baseline linear and non-linear SVM results in Tables 1 and 2 for Caltech101 and VOC07 respectively. As expected non-linear SVMs perform significantly better than linear ones, and exponential kernels somewhat better than their additive counterparts (with the exception of the intersection kernel on Caltech101). All additive kernels perform similarly and the same can be said about exponential kernels.

Embedding additive kernels. We start with linear classifiers on square-rooted BOV vectors (exact embedding for \( K_{bha} \)). We obtain a 54.4% accuracy (1.4% standard deviation) on Caltech101 and 49.0% AP on VOC07. Compared to the linear baseline, this corresponds to absolute increases of 7.3% and 4.9% respectively. On CalTech 101, we are only 1.8% away from the baseline \( K_{chi} \) SVM (and 1.4% away from \( K_{int} \) SVM). As a comparison, [12] reports on
### Table 3. Accuracy (and standard deviation) of the proposed addkPCA applied to \( K_{\chi^2} \). \( E \) is the dimensionality of the embedded vectors.

<table>
<thead>
<tr>
<th>( E )</th>
<th>CALTECH101</th>
<th>PASCAL VOC 07</th>
</tr>
</thead>
<tbody>
<tr>
<td>4k</td>
<td>54.6 (1.6)</td>
<td>49.9 (0.2)</td>
</tr>
<tr>
<td>8k</td>
<td>56.1 (1.2)</td>
<td>50.6 (0.1)</td>
</tr>
<tr>
<td>12k</td>
<td>56.3 (1.5)</td>
<td>50.7 (0.1)</td>
</tr>
</tbody>
</table>

### Table 4. kPCA accuracy (and standard deviation) with \( K_{\chi^2} \). \( E \) is the number of training samples used to learn the embedding and therefore the dimensionality of the embedded vectors. “Full” corresponds to using all training samples to learn the embedding (3,030 for Caltech101 and 5,011 for VOC07).

<table>
<thead>
<tr>
<th>( E )</th>
<th>CALTECH101</th>
<th>PASCAL VOC 07</th>
</tr>
</thead>
<tbody>
<tr>
<td>(in %)</td>
<td>(in %)</td>
<td></td>
</tr>
<tr>
<td>4k</td>
<td>54.1 (2.0)</td>
<td>46.7 (0.5)</td>
</tr>
<tr>
<td>8k</td>
<td>56.7 (1.4)</td>
<td>49.0 (0.3)</td>
</tr>
<tr>
<td>12k</td>
<td>57.5 (1.4)</td>
<td>50.0 (0.2)</td>
</tr>
<tr>
<td>16k</td>
<td>58.1 (1.3)</td>
<td>50.7 (0.2)</td>
</tr>
<tr>
<td>20k</td>
<td>58.4 (1.2)</td>
<td>51.0 (0.2)</td>
</tr>
</tbody>
</table>

### Table 5. Accuracy (and standard deviation) of the approach of [14] applied to \( K_{\chi^2} \). \( E \) is the dimensionality of the embedded vectors.

<table>
<thead>
<tr>
<th>( E )</th>
<th>CALTECH101</th>
<th>PASCAL VOC 07</th>
</tr>
</thead>
<tbody>
<tr>
<td>(in %)</td>
<td>(in %)</td>
<td></td>
</tr>
<tr>
<td>4k</td>
<td>46.7 (0.3)</td>
<td>54.1 (2.0)</td>
</tr>
<tr>
<td>8k</td>
<td>49.0 (0.3)</td>
<td>56.7 (1.4)</td>
</tr>
<tr>
<td>12k</td>
<td>50.0 (0.2)</td>
<td>57.5 (1.4)</td>
</tr>
<tr>
<td>16k</td>
<td>50.7 (0.2)</td>
<td>58.1 (1.3)</td>
</tr>
<tr>
<td>12k</td>
<td>51.0 (0.2)</td>
<td>58.4 (1.2)</td>
</tr>
</tbody>
</table>

We now turn to addkPCA. We used a subset of \( M = 128 \) samples to learn the embedding. We repeated the experiments 10 times with different subsets to learn the embedding and we report the average for \( K_{\chi^2} \) in Table 3. With \( E = D = 4k \), we get results which are slightly better than the simple square-rooting approach. With \( E = 2D = 8k \), we get results as good as those obtained with the best additive kernels.

**Embedding exponential kernels.** We focus on \( K_{\exp} \) since (i) it gives the best (or very close to the best) results and (ii) it is based on the dot-product between square-rooted vectors which is very fast to compute. We report experiments with kPCA and RPs.

For kPCA, we repeated the experiments 10 times using different subsets of the training data to learn the embedding. We report the accuracy in Table 4. On both datasets, kPCA does its best job when all samples are used to learn the embedding. This not surprising given the high-dimensionality of the data and the small training set sizes.

For RPs, we also repeated each experiment 10 times with different random matrices. Results are reported in Table 5 for various values \( E \) of the embedding dimensionality. On the easier Caltech101 dataset, the approximation is already very good for \( E = 2D = 8k \). However, on the more challenging VOC07 dataset, even with \( E = 5D = 20k \), we do not reach the accuracy of the baseline \( K_{\chi^2} \).

### 6.2. Large-scale experiments

We now evaluate how the various embedding techniques scale to large training sets. To train classifiers, we used a subset of ImageNet [4]. Our goal was to create a training set with the same 20 classes as PASCAL VOC 07. At the time when we downloaded the data, only 18 classes had at least one synset in ImageNet: aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, sheep, sofa, train and tvmonitor. We downloaded images from these 18 categories as well as their subcategories (i.e., children synsets). In our experiments, we trained classifiers with up to 10k images per class. Half of the classes have less than 10k images: bicycle (6,653 images), bus (5,224), cow (1,572), dining table (4,061), motorbike (3,779), sheep (9,702), sofa (7,474), train (9,982) and tvmonitor (1,202).

To avoid the bias which typically occurs when training classifiers and evaluating their accuracy on the same dataset, we used as test data the VOC07 test set. Training and testing datasets were collected separately reduces over-fitting effects and offers a much more realistic (and interesting) challenge. As is the case of VOC07, we use AP (averaged over the 18 classes) to measure classification accuracy.

Each experiment was repeated 5 times on 5 different training subsets (later referred to as folds) and we report the average value as well as the standard deviation. To tune the SVM regularization parameters, we trained the classi-
fiers on 50% of the training data of the first fold and used the second 50% of this fold as validation set. The same parameters were used for the 5 folds.

**Does more data help?** We first evaluate the influence of the training set size on the classification accuracy. This is an important question that remains largely unexplored. For example, [20, 12, 10] report experiments on large training sets but do not quantify the benefit of using the full training set as opposed to only a fraction. Results are reported in Figure 2 (left). As a comparison, the smallest training set (250 images per class = 4.5k images in total) is comparable in size to the train+val set of VOC 07. We do not report results with libsvm beyond 42k training images as training the 18 classifiers on 50% of this fold as validation set. The same parameters were used for the 5 folds.

The training time includes the combined time of learning the embedding, embedding the training vectors and training the 18 classifiers. All the CPU times correspond to training on a single fold and were measured on a 3GHz Xeon machine using a single CPU. As a comparison, it takes less than 4h to compute SIFT features and extract BOV histograms from 140k images (approx. 100ms per image). While kPCA or RPs can outperform addkPCA, the benefit comes at a high cost: several hours of training as opposed to 2 min. Note that the cost of kPCA and RPs is largely dominated by the embedding, not by the linear SVM training.

Memory requirements should also be taken into account. For instance, with RPs, one has to set $E = 20D$ for instance, as the training data would not have fitted in the RAM of our 32GB machine. An important conclusion is that RPs for shift-invariant kernels seem better suited to problems where the input vectors are small dimensional as is the case of the datasets employed in [14] (127 dimensions max.).

**Training cost.** We report in Figure 2 (right) the accuracy as a function of the training time (i.e. for various sizes $E$ of the embedding space) using a training set of 140k images. “lin SVM” = linear SVM baseline. “sqrt + lin SVM” = linear SVM on square-rooted vectors. “add kPCA + lin SVM” = proposed addkPCA for $K_{exp}$. “rand + lin SVM” = random projections to approximate $K_{bh}$, “kPCA + lin SVM” = standard kPCA with $K_{exp}$, “sqrt + rbf SVM” = baseline kernel SVM with $K_{bh}$. For the 3 embedding techniques, we used on the left plot the best system, i.e. the one with the highest value $E$ on the right plot (and therefore the highest training cost).

![Figure 2. Left: classification accuracy as a function of the number of training images. Right: classification accuracy as a function of the training time (i.e. for various sizes $E$ of the embedding space) using a training set of 140k images. “lin SVM” = linear SVM baseline. “sqrt + lin SVM” = linear SVM on square-rooted vectors. “add kPCA + lin SVM” = proposed addkPCA for $K_{exp}$. “rand + lin SVM” = random projections to approximate $K_{bh}$, “kPCA + lin SVM” = standard kPCA with $K_{exp}$, “sqrt + rbf SVM” = baseline kernel SVM with $K_{bh}$.)
7. Conclusion

In this paper, we considered the problem of learning image classifiers with large training sets. We explored explicit embedding as an approach to scale non-linear SVMs and experimented with several techniques. We demonstrated that simply square-rooting BOV vectors, which corresponds to an exact embedding for $K_{bha}$, already leads to large improvements. We then proposed addkPCA: a simple and efficient embedding based on kPCA for additive kernels. We showed that it improved over the square-rooting at a very affordable cost. Finally, we experimented with kPCA and random projections to approximate exponential kernels and showed that it led to additional improvements but at a much higher cost this time.

This study focused on large training sets but did not address the problem of dealing with a large number of categories (e.g., thousands) which is a very challenging problem on its own.

A. SGD for linear SVMs

We give a brief overview of the SGD solver available at [1] inspired by [16]. Given a training set of labeled samples \( \{(x_i, y_i), i = 1, \ldots, N\} \), with \( y_i \in \{-1, +1\} \), we consider the problem of minimizing the regularized loss:

\[
E(w) = \frac{\lambda}{2} ||w||^2 + \frac{1}{N} \sum_{i=1}^{N} \ell(w' x_i, y_i).
\]

where \( \lambda \) is the regularization parameter. In the SVM case, \( \ell(w' x_i, y_i) = \max\{0, 1 - y_i w' x_i\} \). Assuming that at iteration \( t \) the SGD algorithm considers only sample \( x_i \), the loss \( (9) \) is approximated by:

\[
E_t(w) = \frac{\lambda}{2} ||w||^2 + \ell(w' x_i, y_i).
\]

\( w \) is updated as follows:

\[
w_{t+1} = w_t - \eta_t \nabla_{w=w_t} E_t(w) = (1 - \lambda \eta_t) w_t + \eta_t y_i x_i \delta_t
\]

with \( \delta_t = 1 \) if \( \ell(w' x_i, y_i) > 0 \) and \( \delta_t = 0 \) otherwise. The learning rate has the form \( \eta_t = \frac{1}{\lambda(t+t_0)} \) where \( t_0 \) is set heuristically in [1].

References