Sparse Spatiotemporal Coding for Activity Recognition

Thomas Dean, Rich Washington and Greg Corrado

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-10-02
March 2010

Sparse Spatiotemporal Coding for Activity Recognition

Thomas Dean[1], Rich Washington[1], and Greg Corrado[2]

[1] Google Inc.
[2] Stanford University

Abstract. We present a new approach to learning sparse, spatiotemporal features and demonstrate the utility of the approach by applying the resulting sparse codes to the problem of activity recognition. Learning features that discriminate among human activities in video is difficult in part because the stable space-time events that reliably characterize the relevant motions are rare. To overcome this problem, we adopt a multi-stage approach to activity recognition. In the initial preprocessing stage, we first whiten and apply local contrast normalization to each frame of the video. We then apply an additional set of filters to identify and extract salient space-time volumes that exhibit smooth periodic motion. We collect a large corpus of these space-time volumes as training data for the unsupervised learning of a sparse, over-complete basis using a variant of the two-phase analysis-synthesis algorithm of Olshausen and Field [1997]. We treat the synthesis phase, which consists of reconstructing the input as a sparse — mostly zero coefficient – linear combination of basis vectors, as an L_1-regularized least-squares problem. We found most existing algorithms for solving the L_1-regularized least-squares problem to be slow and perform poorly at the task of learning codes that are spatially oriented and temporally diverse in terms of transformations and velocities. To reduce the time required in learning — and most importantly the time required for reconstruction in subsequent production use — we adapted existing algorithms to exploit potential parallelism through the use of readily-available SIMD hardware. To obtain better codes, we developed a new approach to learning sparse, spatiotemporal codes in which the number of basis vectors, their orientations, velocities and the size of their receptive fields change over the duration of unsupervised training. The algorithm starts with a relatively small, initial basis with minimal temporal extent. This initial basis is obtained through conventional sparse coding techniques and is expanded over time by recursively constructing a new basis consisting of basis vectors with larger temporal extent that proportionally conserve regions of previously trained weights. These proportionally conserved weights are combined with the result of adjusting newly added weights to represent a greater range of primitive motion features. The size of the current basis is determined probabilistically by sampling from existing basis vectors according to their activation on the training set. The resulting algorithm produces bases consisting of filters that are bandpass, spatially oriented and temporally diverse in terms of their transformations and velocities. We demonstrate the utility of our approach by using it to recognize human activity in video.

1 Introduction

This work focuses on learning sparse, over-complete spatiotemporal codes in the spirit of Olshausen and Field [1], Hyvärinen *et al* [2], and others. Our initial investigation into

this area was inspired by the work of Cadieu and Olshausen [3] on learning transformational invariants from the statistics of natural movies. We adopt a generative model similar to that of Olshausen and Field [1] and an alternating-optimization algorithm analogous to the analysis-synthesis model proposed by Mumford [4] and used by Olshausen and field in their *SPARSENET* implementation.

Following the trend in sparse coding, we substitute L_1-regularized least-squares algorithms for the conjugate gradient solver used in SPARSENET. In particular, we develop several variants of the one-at-a-time coordinate-wise descent algorithm of Friedman *et al* [5] and experiment with the feature-sign algorithm of Lee *et al* [6]. We also employ various methods for *shaping* the variance of the activations — the coefficients of the basis vectors in solutions to the least-squares problem — to ensure that all of the basis vectors are contributing.

Spatiotemporal bases that span more than a few frames of video have large numbers of weights and are slow to train. We present an algorithm that accelerates sparse coding by recursively constructing basis vectors, adjusting only a fraction of the weights at any given time. The resulting bases exhibit a wide range of orientations, scales and velocities, and outperform bases trained in a conventional manner.

We found it to be the case that interesting motion is quite rare even in videos selected for illustrating motion, and so we experimented with several interest-point operators to extract 3-D patches more likely to contain motion of the sort characteristic of human behavior. We use the space-time interest-point operator of Dollár *et al* [7] as a filter for extracting space-time volumes that exhibit periodic motion, and then use large collections of these volumes for learning sparse codes.

In this domain, processing speed is an issue for two reasons: First, each algorithm variant requires a different set of parameter values to learn codes that optimize its own performance on activity recognition. Exploring the space of algorithms and their parameters requires us to learn many thousands of models, a prohibitive process unless learning individual models can be accelerated. Of course, some of the parameter search can be carried in parallel — for example, running the same algorithm on a mesh over the parameter space — using a cluster of servers, but much of work is exploratory and involves sequential hypothesis testing. Second, our primary goal is not to learn models that merely visualize well, but to learn models that perform well on a task that necessarily involves very large data sets. To improve performance both in learning sparse codes and in applying them in practice, we adapt the L_1-regularized least-squares solver described by Raina *et al* [8] to exploit not only the parallelism inherent in the coordinate-descent algorithm as done in the orignal version, but also data parallelism, by rewriting the algorithm in vectorized style that allows us to leverage the power of existing linear algebra packages.

To evaluate our approach, we applied the resulting sparse codes to recognizing human activity. We adapted software developed by Piotr Dollár and performed the initial testing on his facial-expression dataset [7] and the Weizmann human-action dataset [9]. We then took codes trained on data from these two datasets and applied them to the KTH human-action dataset [10] achieving recognition error comparable to state-of-the-art methods. In leave-one-out experiments where we both trained and tested on the KTH

dataset, our approach exceeds the performance of state-of-the-art methods, including our own published results [11].

2 Preprocessing

We apply several preprocessing steps to the data prior to its use in learning sparse codes or inferring coefficients to reconstruct a 3-D patch as a sparse, linear combination of basis vectors. First, we apply a linear transform to each frame so that the pixels are uncorrelated (spatially) and their variances equal. This *whitening* step is used to remove correlations in the data that a learning algorithm would otherwise have to account for and typically are not of interest. Some neuroscientists believe the primate early visual system employs a gain-control mechanism whereby the response of each cell is normalized by the integrated activity of its neighbouring cells. Brady and Field [12] provide evidence suggesting that such a mechanism reduces cell-response variability both within and between scenes, as well as reducing the entropy of the response distribution resulting in a more efficient transfer of information. We therefore apply a method of local-contrast normalization which simulates this neural mechanism and thereby reduces undesirable variability in postprocessing. These two preprocessing steps, image whitening and local contrast normalization, provide a very rough approximation to the information being transferred by the optic tract leading from the retina and ultimately entering the striate cortex.

In working on video categorization involving large datasets where we were employing a previously-trained basis, we found the overhead of running the whitening and contrast normalization filters on video data to be prohibitively time consuming. We experimented with a number of simpler normalization techniques and ultimately settled on the strategy of extracting cuboids from the raw pixels and then normalizing the pixels in a sample of cuboids from each video to have zero mean and unit variance and truncating values greater than two-standard deviations from the mean. This operation added only a few milliseconds to the total time spent in sparse coding a sample of cuboids, and the new results presented in this paper all use this expeditious alternative; however, we continue to use whitening and local contrast normalization for learning bases.

We found it difficult to efficiently learn useful spatiotemporal codes from random 3-D patches extracted from video, and so we experimented with several interest-point operators to extract space time volumes likely to contain motion of the sort characteristic of human behavior. We use the space-time interest-point operator of Dollár *et al* [7] as a filter for extracting space-time volumes that exhibit periodic motion, and collect a large set of these volumes for learning sparse codes. Each video is first convolved with a 2-D Gaussian smoothing kernel applied along the spatial dimensions. The result is then convolved with a quadrature pair of 1-D Gabor filters applied temporally. A detector is tuned to respond whenever the variation in local image intensities contain periodic frequency components. The response function for the detector is defined as follows:

$$R = (I * g * h_1)^2 + (I * g * h_2)^2$$

where $g(x, y; \sigma)$ is the 2-D smoothing kernel, $\{h_1, h_2\}$ is the quadrature pair determined by $h_1(t; \tau, \omega) = -\cos(2\pi t\omega)e^{-t^2/\tau^2}$ and $h_2(t; \tau, \omega) = -\sin(2\pi t\omega)e^{-t^2/\tau^2}$.

Following [7], we set $\omega = 4/\tau$, and thus σ and τ correspond, respectively, to the spatial and temporal scale of the detector. A 3-D patch or *cuboid* is extracted at each local maxima of the filter response function using non-max-suppression to control for overlap.

We also experimented with an interest point detector developed by Laptev and Lindeberg [13]. Laptev and Lindeberg extend the idea of Harris-corner [14] detectors in the spatial domain to space-time interest points characterized by strong variation in both the spatial and temporal dimensions, *e.g.*, the abrupt change in velocity when a ball is kicked. They do so by expanding a linear scale-space representation of the image with a matrix of first-order spatial and temporal derivatives, and searching for regions with significant eigenvalues. The method is extended to be scale invariant by applying a normalized spatiotemporal Laplace operator. We found the Laptev and Lindeberg detector to be too restrictive for activity recognition and the Dollar *et al* detector to include most if not all of the points detected by Laptev and Lindeberg while excluding most irrelevant motion.

3 Sparse Coding

Olshausen and Field [15] present their method of learning a sparse basis to represent natural images as solving the following optimization problem:

$$B^* = \arg\min_B \langle \min_A \|X - AB\|_2^2 + \lambda S(A) \rangle \tag{1}$$

where X is an $M \times L$ data matrix consisting of M cuboids of L pixels each, B is an $N \times L$ matrix of N basis vectors, A is an $M \times N$ matrix of coefficients intended to reconstruct X as a linear combination of the basis vectors, $S(A)$ is a sparsity penalty, and λ is a constant that trades reconstruction error for sparsity. For many standard penalty functions, the objective function is convex in A if we hold B fixed and convex in B if we hold A fixed, and so solutions to Equation 1 are often solved using an iterative process in which each iteration consists of two steps: In the first step, we fix the coefficients and solve for the basis vectors subject to a set of linear constraints that control for the size of the basis weights. In the second step, we fix the basis vectors and solve for the coefficients in an attempt to reconstruct the input modulo some variant of weight penalty designed to encourage sparsity. Learning consists of alternating between these two steps until convergence or some performance threshold is achieved.

Mumford [4] describes this iterative process as an *analysis-synthesis* loop, which he conjectures plays an important role in early visual processing. In the case of learning a sparse code for natural images, the first step — solving for the basis vectors — constitutes a form of analysis which produces an explanation in the form of a generative model of the data; we call this the *analysis step*. The second step —- solving for the coefficients — corresponds to synthesizing the data from a given fixed basis; we call this the *synthesis step*.

Olshausen and Field [1] implemented a version of this analysis-synthesis loop which learns a sparse, over-complete code whose basis vectors correspond to filters that are bandpass and oriented and that resemble Gabor functions. The original Olshausen and Field work assumed a Cauchy prior on the coefficients, introduced a differentiable regularization term to implement this prior, and used a conjugate-gradient solver in the

synthesis step:

$$\text{minimize}_A \; J(A|B) = \|X - AB\|_2^2 + \lambda \sum_{i,j} \log(1 + A_{i,j}^2) \tag{2}$$

In the analysis step, they fix the coefficients and take one step of gradient descent toward solving the following minimization:

$$\text{minimize}_B \; J(B|A) = \|X - AB\|_2^2$$

Without imposing some constraint, this procedure will cause the basis weights to grow without bound. Olshausen and Field deal with this by adapting the L_2 norm of each basis vector independently so the coefficients are maintained at an appropriate level and the separate variances over the training data in the activation of basis vectors are approximately equal.[3] Their implementation of this algorithm is called *SPARSENET*.

3.1 L_1 Regularization

In the last few years, there has been a good deal of work in machine learning and statistics using L_1 regularization to induce sparsity. Since the L_1 term is not differentiable, much of the effort has gone into developing new algorithms for solving the corresponding optimization problem. We have experimented extensively with the method of one-at-a-time coordinate-wise descent [5] to solve for the coefficients in the synthesis step. Instantiations of this method are called *coordinate-descent* algorithms and solve the following alternative to the optimization in Equation 2:

$$\text{minimize}_A \; J(A|B) = \|X - AB\|_2^2 + \lambda \|A\|_1 \tag{3}$$

where $\|A\|_1 = \sum_{i,j} |A_{i,j}|$. In these experiments, we started with the basic SPARSENET algorithm and substituted a L_1-regularized coordinate-descent algorithm for the conjugate-gradient solver used by Olshausen and Field. We call the algorithm *LASSONET* in homage to SPARSENET and the acronym *LASSO* (for *Least Absolute Selection and Shrinkage Operator*), which has become a catchall term describing a class of methods for estimating least-squares parameters subject to an L_1 penalty. Convergence using coordinate descent is much faster than the original Olshausen and Field algorithm, but still relatively slow in working with large datasets and thousands of basis vectors.

Lee *et al* [6] describe a method for learning sparse codes that works by alternating between solving an L_1-regularized least-squares problem and an L_2-constrained least-squares problem. To solve the first they introduced the *feature-sign* algorithm based on the insight that once the signs of the coefficients are known, the problem reduces to a standard, unconstrained quadratic optimization problem, which can be solved efficiently. They *guess* the signs by performing line searches using a conjugate gradient solver. To solve the L_2-constrained least-squares problem, they reduce the number of optimization variables considerably by solving the Lagrange dual using Newton's method.

[3] A basis vector is said to be *activated* with respect to a given cuboid if the associated coefficient in the linear combination of basis vectors reconstructing the cuboid is non-zero.

We ran experiments substituting the feature-sign and Lagrange-dual algorithms for the analysis and synthesis steps in LASSONET. Solving for the optimal basis vectors B given fixed coefficients A reduces to minimizing $\|X - AB\|_F^2$ — where $\|.\|_F$ is the Frobenius norm — subject to the constraint that $\sum_{i=1}^{L} B_{i,j}^2 < c$ for all $1 \leq i, j \leq N$ assuming N basis vectors. This constraint on the basis vectors was introduced for numerical stability and to avoid degenerate solutions. In our experience, incorporating the constraint directly into the objective function, as in the Lagrange-dual algorithm, is more robust than the adaptive methods used in SPARSENET, and no additional normalization step is required to keep the activation variance approximately equal over all of the basis vectors.

3.2 Parallel Algorithms

Despite being very fast, we were disappointed with the reconstruction error that we were able to achieve using the feature-sign algorithm. In an attempt to address the speed versus accuracy tradeoff, we wrote a multi-threaded C++ version of coordinate descent that enabled us to gain some data parallelism — one thread per data vector up to some fixed number of threads — on machines with multiple cores. The C++ implementation averaged 1/3 the reconstruction error for the same degree of sparsity — approximately 5% non-zero coefficients was our target. Unfortunately, it took eight cores to match the speed of feature-sign.

Next, inspired by recent work of Raina *et al* [8] in developing parallel solvers to run on graphics processors. Their basic algorithm is a relatively straightforward implementation of coordinate descent with the addition of a simple-but-effective line search to select the gradient step. Raina *et al* implemented one version in straight C and a second version in C with Nvidia CUDA$^{\mathrm{TM}}$language extensions that compile to run on the Nvidia line of graphics processor (GPU) hardware.

We took the straight C version and rewrote it as vectorized Matlab code (see the listing in Appendix A), and thus were able to take advantage two sources of parallel hardware. First, we were able to leverage Intel's *Math Kernel Library* (MKL) which is an architecture-tuned library which includes the *Basic Linear Algebra Subroutines* (BLAS) and is available in Matlab running on Intel hardware. In addition to exploiting MKL's optimized use of the processor cache, MKL can take advantage of multiple-cores and the vector processing hardware available through the use of streaming SIMD extension (SSE) instructions. Second, we were able to experiment with a version of BLAS developed by Nvidia to exploit GPUs (CuBLAS) by using a trial copy of the Jacket Engine for Matlab developed by AccelerEyes$^{\mathrm{TM}}$. Jacket allows us write standard Matlab code that uses the native BLAS (MKL in the case of machines running Intel chips) if no acceleration hardware is available or CuBLAS if there is a GPU available available on the target machine. Ease of use coupled with quite remarkable improvements in performance make the Jacket system extremely attractive.

We also ported our vectorized Matlab code to C++ using the Eigen C++ template library (see the listing in Appendix B for comparison). Eigen produces single-threaded code which is ideal for our production environment, and is able achieve some performance gains by leveraging the streaming SIMD extension instructions (SSE) that run on AMD and Intel chips to exploit parallelism in elementary matrix and vector operations.

For our timing experiments, we used a Dell Precision T3400 workstation with two Intel 2.40 GHz Core 2 Quad Q6600 processors running Linux Ubuntu 8.0.4 and Matlab 7.8.0.347 (R2009a). The C++ implementations — the multi-threaded C++ version and the single-threaded Eigen code — were compiled with `gcc` 4.2.4. The GPU experiments used CUDA 2.2 running on an Nvidia GeForce GTX 280. Jacket 1.1.1 was used to run the Vectorized Matlab code on the GPU thereby leveraging CuBLAS. Here are the results averaged over multiple runs — there was little of no variation — for six implementations of three algorithms: Lee *et al*'s feature-sign written in Matlab, our multi-threaded C++ implementation of coordinate descent, and four implementations of Raina *et al*'s coordinate descent with its optimized gradient step. All runs were performed using 864 basis vectors, each vector with a spatial extent of 13×13 and a temporal extent of 7 frames, and 1024 cuboids. The size of the dataset was selected since we use a sample of 1024 cuboids extracted from the first thirty seconds of a video as the foundation for computing descriptors used to categorize videos in our experimental video-categorization work.

Implementation	Hardware	4 Cores	1 Core	MSE
Lee *et al*, 2007,	CPU	37.4s	40.7s	0.250
Multi-thread C++	CPU	60.0s	165.1s	0.086
Vectorized Matlab	CPU	36.7s	54.1s	0.088
Vectorized Matlab	GPU	—	3.8s	0.088
Raina *et al*, 2009	GPU	—	10.0s	0.088
Vectorized Eigen	CPU	—	58.7s	0.088

The multi-threaded C++ exploits data parallelism by spawning separate threads for each cuboid; it can be compiled with the `NTHREADS` constant set to limit the maximum number of threads, and hence cores, it can make use of. The Raina *et al* CUDA code exploits parallelism in the implementation of coordinate descent, but not data parallelism, it codes each cuboid in sequence. The vectorized Matlab implicitly allows us to exploit data parallelism and parallelism in the structure of coordinate descent. The degree to which it exploits either is entirely dependent on how well BLAS or, more generally, the underlying linear algebra package, is able to take advantage of parallelism in the elementary operations on matrices and vectors. The Raina *et al* CUDA implementation in coding for a particular cuboid terminates as soon as it cannot improve on the current reconstruction, and is guaranteed to reach an optimum. In contrast, the vectorized Matlab continues until it cannot improve on the reconstruction of *any* cuboid, and so rarely terminates before exhausting its maximum number of iterations (see Line 48 in the listing in Appendix A).

Monitoring core usage, all of the four cores were pinned to the max running the multi-threaded C++ code with `NTHREADS` set to four. When you give Matlab all four cores, Intel's MKL version of BLAS is written to take advantage of multiple cores, but it doesn't pin them like the C++ code; all eight of the cores show some activity, but in the 50-75% range, and the performance gain isn't nearly what the C++ code managed. The vectorized Matlab code didn't take too much of a hit when we reduced the number of cores from four to one, which is a credit to Intel's MKL version of BLAS. The C++ code scales pretty much linearly with the number of cores.

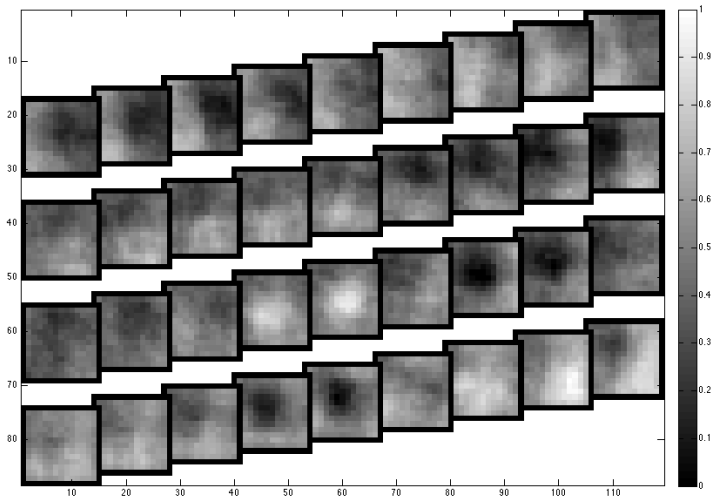

Fig. 1. Graphic depicting a sample of four of the 2048 basis vectors learned by LASSONET in a trial that performed well on our activity-recognition task.

The GTX provides a good deal more SIMD hardware to exploit, but Eigen can still achieve considerable advantage from the Intel vector units using the SSE instructions. The Eigen implementation is roughly equivalent to the vectorized Matlab code using MKL and the multi-threaded C++ running on four cores. For the exploratory experiments described in this paper, we used the vectorized Matlab running under Jacket and using CuBLAS. We chose a basis consisting of 864 vectors for our timing experiments; however, have been able to achieve high-accuracy on a human-activity recognition benchmark using as few as 256 vectors. The Eigen C++ implementation runs in about seven seconds on the same machine mentioned above using a basis consisting of 256 vectors with spatial extent 13×13 and temporal extent of 3 frames, and applied to dataset consisting of 1024 cuboids.

3.3 Recursive Sparse Coding

We have considerable experience with learning sparse spatiotemporal codes using different combinations of coding algorithm, interest-point operators for extracting space-time volumes, and basis vectors of varying spatial and temporal extent. If the basis vectors are too large, learning is slow and the resulting codes tend to generalize poorly; if they are too small, the codes tend to be too general and they discriminate poorly. We are not the first to generate sparse, spatiotemporal codes from video; van Hateren and Ruderman [16] apply independent components analysis to obtain codes resembling moving sinusoids windowed by Gaussian envelopes (Gabors), and Olshausen [17] was able to compute convolution codes that exhibit similar characteristics by applying matching-pursuit [18] in space and time. Olshausen [17] used a form of temporal smoothing to encourage the formation of spatiotemporal features that change gradually over time,

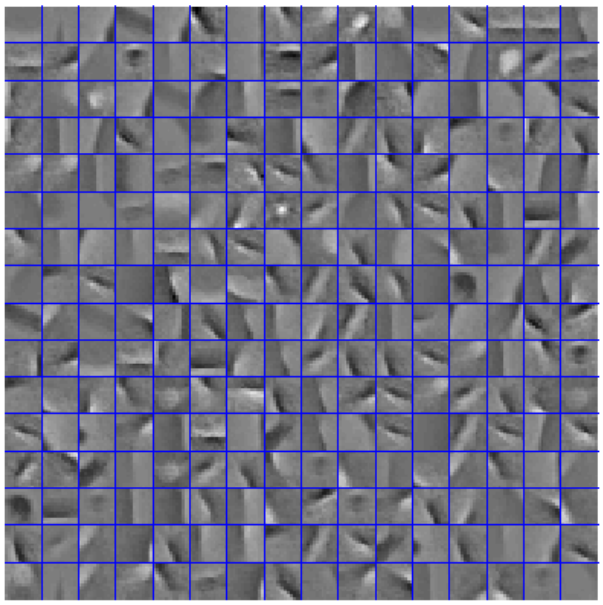

Fig. 2. The center frames of the receptive fields of 256 out of 2048 basis vectors learned by RECURSIVE_LASSONET. The basis vectors capture a wide range of oriented filters undergoing diverse transformations.

in a manner similar to Földiák's method for learning invariants from sequences [19] and Wiskott and Sejnowski's slow-feature analysis [20]. The best-performing codes obtained using LASSONET tend to look very different (see Figure 1) from those reported by Olshausen and van Hateren and Ruderman. Spectral analysis of the basis vectors reveals an entirely appropriate distribution of velocities but hardly any spatial orientation — the individual frames resemble symmetric Gaussians rather than Gabors.

Using our algorithms and training on random 3-D patches for a considerable duration, we were able to obtain sparse, spatiotemporal codes that appeared roughly similar to those reported by Olshausen and van Hateren and Ruderman; they did not, however, perform particularly well on the activity recognition task. We noticed, however, that when we applied any of the three algorithms to learning a (temporally) degenerate code consisting of a single frame trained using 3-D cuboids extracted using the Dollár *et al* filter, we observed the usual variety of oriented, bandpass filters, but with the difference that due to the max-suppression step in extracting interest points a Gaussian envelope fit to the filters tended to be centrally located in the frame (see Figure 2). This suggested a simple iterative training method that learns a basis one frame at a time.

The function *RECURSIVE_LASSONET* implements such a frame-by-frame scheme as follows:

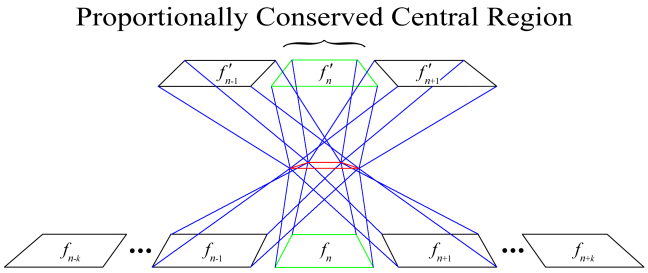

Fig. 3. RECURSIVE_LASSONET consists of learning a sequence of models with each successive model having a larger temporal extent and an increased number of basis vectors. In each recursive step, the central frames are conserved while the outer frame coefficients are altered by stochastic gradient descent.

- *base case* — we start with a basis B_1 consisting of $|B_1|$ basis vectors with a temporal extent of one that code for a single frame; we use just the center frames from a sample of cuboids for training — see graphic in Figure 3 for an illustration;
- *recursive step* — given the basis B_{n-1} which consists of $|B_{n-1}|$ vectors, we create a new basis with $|B_n| > |B_{n-1}|$ basis vectors; the default method uses a simple exponential growth model $|B_n| = |B_{n-1}|^G$, where $G > 1$ is the exponential growth factor;
- *temporal expansion* — each basis vector v in B_n is constructed from a randomly selected basis vector u in B_{n-1}; we increase the temporal extent of u to span $2n-1$ frames by adding two new frames to those of u with randomly initialized weights that *sandwich* the frames of u; the weights from u that reside within this sandwich are called the *conserved region* of v;
- *proportional weight conservation* — apply the chosen sparse-coding algorithm to adjust the new weights with the following twist; in the analysis step, hold the weights in the conserved region constant, but then prior to the synthesis step, rescale all of the basis vectors — including the weights in the conserved region — to have unit magnitude.

In a slightly more sophisticated version used to generate the models for the experiments described in Section 4.2, the proportionally conserved region of the expanded basis in the recursive step is selected by sampling with replacement from the previous basis in the sequence of models using an empirically derived proposal distribution. This proposal distribution was obtained by estimating the frequency with which basis vectors in the previous basis contribute to reconstructing the appropriate portions of cuboids that these basis vectors span both spatially and temporally. The coefficients obtained from fitting the current basis to a sample of training data are used as pseudo counts to generate this proposal distribution.

Unlike our earlier top performing bases, RECURSIVE_LASSONET produces bases in which the individual frames of the multi-frame basis vectors are oriented and appear Gabor like. Figure 4 summarizes the spectral analysis of the basis vectors of a typical result produced by RECURSIVE_LASSONET in terms of velocities and orientations.

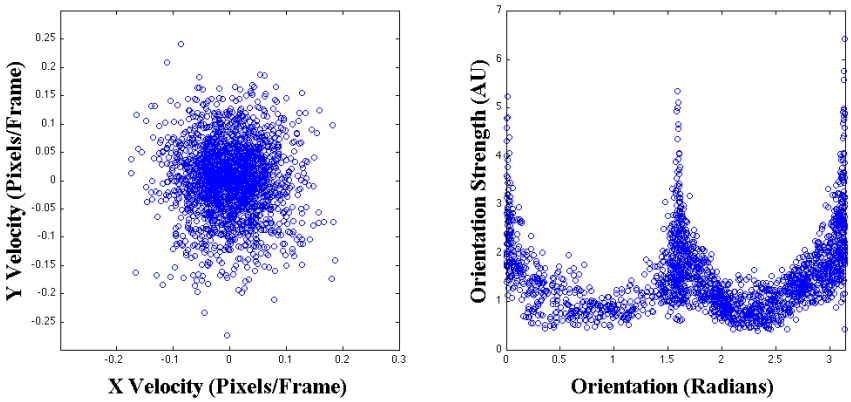

Fig. 4. The left plot shows the distribution of velocities obtained from a spectral analysis of the basis vectors. The plot on the right approximates the distribution of orientations.

4 Experiments

For exploratory experiments, we used the facial-expression dataset described by Dollár *et al* [7]. Our initial experiments in learning to recognize human activity were performed using the Weizmann human-action dataset described by Gorelick *et al* [9]. For comparing the performance of our features against other published results, we used the KTH human-activity dataset described by Schuldt *et al* [10]. Each dataset is divided into two or more disjoint subsets of video clips for testing, and each clip is assigned a *label* that characterizes its associated activity. The facial-expression dataset consists of 192 clips, 5 expressions and 4 subsets divided according to subject and lighting conditions. The Weizmann dataset consists of 93 clips and 10 activities divided into two subsets featuring different subjects. The KTH dataset consists of 599 clips of 25 subjects performing six activities (walking, jogging, running, boxing, hand-waving and hand-clapping) in four different scenarios (outdoors, outdoors with scale variation, outdoors with subjects wearing different clothes and indoors); the KTH dataset is divided into 25 subsets according to subject.

4.1 Methods

As prolog to testing, we learn a basis and then use it to generate a set of descriptors, one set of descriptors for each video clip. The following steps are performed in preparation for testing:

1. whiten and apply local contrast normalization to each frame of each video clip;
2. extract a sample of cuboids from each video clip using the method of Dollár *et al*;
3. learn a sparse-coding basis using RECURSIVE_LASSONET and the sampled cuboids as training data;
4. generate a descriptor for each cuboid corresponding to the coefficients inferred using the L_1-regularized least-squares solver used in the sparse-coding algorithms;

5. optionally, apply singular-value decomposition to a sample of descriptors to generate a set of principal components to reduce the dimensionality of the descriptors;

We use early stopping to avoid over fitting in learning the basis.

For testing, we adopt the leave-one-out (LOO) protocol that was used in the evaluation of the other methods with which we compare ours. In each round of LOO evaluation on the KTH dataset, we train on 24 of the 25 subsets and test on the remaining one. In Dean *et al* [11] we used a bag-of-words approach similar to the one used by Dollár *et al* [7]. In this approach, we run ten trials, where each trial is composed of 25 rounds such that for each disjoint subset s_j of video clips we perform the following steps:

1. use k-means to cluster the descriptors in the complement $C_j = \cup_{i \neq j} s_i$ of s_j returning K centroids indexed $1, ..., K$, where K is the number of so-called *visual words*;
2. for each cuboid in each clip find the centroid closest to the cuboid's descriptor and return the centroid's index;
3. for each clip, construct a vector of length K whose kth component is the number of cuboids in the clip that map to the kth centroid; this vector (histogram) is used to represent the clip as a *bag of visual words*;
4. to label a clip c in s_j with associated histogram h_c use the label of the clip c' in C_j whose histogram $h_{c'}$ is nearest h_c according to the χ^2 distance metric;
5. the *recognition error* is just the number of incorrectly labeled clips divided by the number of clips in s_j;

This bag-of-words approach worked reasonably well on the KTH dataset, but it was not clear we were making the best use of the sparse codes we had worked so hard to generate. We eliminated optional dimensionality reduction step in constructing individual cuboid descriptors and the vector quantization steps in which we construct the dictionary of visual words and quantize each cuboid descriptor by computing its closest word. In the new approach, we don't have to run multiple trials because the process is completely deterministic. We run 25 rounds of LOO such that for each disjoint subset s_j of video clips we perform the following steps:

1. for each clip, suppose that we extract N cuboids and suppose that we have a basis consisting of M basis vectors; our sparse coding algorithm returns an $N \times M$ matrix of coefficients C such that C_{ij} is the jth coefficient used to reconstruct the ith cuboid — C is quite sparse, typically no more than 5% of the coefficients are non-zero;
2. set all of the non-zero coefficients in C to 1 and then sum the columns of C to obtain a $1 \times M$ vector d where d_k is the number of times the kth basis vector contributed to reconstructing one of the N cuboids; d is the descriptor that we use summarize the clip from which the cuboids were extracted;
3. as before $C_j = \cup_{i \neq j} s_i$ is the complement of s_j; to label a clip c in s_j with associated descriptor d_c use the label of the clip c' in C_j whose descriptor $d_{c'}$ is nearest d_c according to the χ^2 distance metric;
4. the *recognition error* is just the number of incorrectly labeled clips divided by the number of clips in s_j;

Our goal is not to find a better classifier, but rather to find better features for classification. For this reason, we use a simple nearest-neighbor classifier for all of our comparisons.

4.2 Results

The following table compares the reported performance of several approaches to activity recognition on the KTH dataset — including our earlier work — using the LOO protocol with our best model obtained by training with LASSONET and the feature-sign algorithm to compute descriptors:

Percentage Correct	**Box**	**Clap**	**Wave**	**Jog**	**Run**	**Walk**	**Mean**	**Notes**
This paper, 2009	79	86	93	78	82	99	85.73	1NN
Dean *et al*, 2009	81	80	86	69	89	81	81.0	1NN
Wang and Li, 2009	88	81	83	70	73	91	81.0	—
Niebles *et al*, 2008	98	86	93	53	88	82	83.33	LDA
Ke *et al*, 2007,	84	79	88	66	81	88	80.9	—
Dollár *et al*, 2005	—	—	—	—	—	—	80.17	SVM
Dollár *et al*, 2005	80	82	84	63	73	89	78.5	1NN
Ke *et al*, 2005,	81	36	44	69	56	92	62.97	
Schuldt *et al*, 2004	98	60	74	60	55	84	71.72	SVM

Our results are most appropriately compared with Dollár *et al* [7] and Wang and Li [21] since their work primarily concerns feature selection, and both our results and their results shown here are based on a 1-NN classifier. Schuldt *et al* [10] use the space-time interest points of Laptev and Lindeberg [13] and support-vector machines rather than 1-NN for classification. Dollár *et al* [7] use the same interest-point operator as we do — they introduced the operator in the referenced paper — and a descriptor based on histograms of gradients in x, y and t analogous to Lowe's [22] 2-D *Scale Invariant Feature Transform* (SIFT) descriptor. Wang and Li may gain some advantage from the fact that they crop the region of the video containing the person performing the activity. Niebles *et al* [23] also use Dollár's features, and their results are included as an example of how more sophisticated classifiers can leverage spatiotemporal features.

5 Discussion

It seems reasonable to conjecture that RECURSIVE_LASSONET works as well as it does in part for the same reasons that greedy layer-by-layer learning works in so-called deep networks [24, 25]. Weights trained early in the process continue to do a good job of reconstructing frames in the core of the basis vector, while gradient adjustments concentrate on the outermost frames added in the temporal expansion step. Vectors that don't serve to represent new aspects of the data can be weeded out by analyzing the degree to which they are activated during reconstruction.

Regarding the improvement in performance when we moved from the more complicated method of [11] involving additional steps for dimensionality reduction and vector

quantization, it would seem that in the case of the larger bases which required dimensionality reduction, the linear transformation erased the advantage of the original sparse code. And, in the case of smaller codes which didn't require dimensionality reduction, it seems likely the vector quantization step was unnecessary and only served to introduce an additional source of noise. In our experiments exploring possible alternatives to whitening and local contrast normalization as a preprocessing step in production, we found that we could obtain performance on a par with our earlier results by using the raw pixels — unsigned eight-bit integers — with no preprocessing and then summing the maximum coefficients from each reconstruction or sorting the the coefficients in descending order and taking the top K where $K = 4$ worked quite well in practice.

Nonlinear operations like taking the maximum or thresholding are often useful when working on noisy data [26]. The descriptor we finally adopted and responsible for the new results reported in this paper — computed in Step 2 — did not perform particularly well with the raw, unpreprocessed input. We experimented with faster, simpler forms of preprocessing and eventually settled on rescaling the raw pixels in each cuboid to be zero-mean and unit variance, and truncating any values more than two standard deviations from the mean to eliminate outliers. This simpler form of preprocessing can be performed in a couple of milliseconds, and, using the descriptor responsible for the results reported in this paper, we are able to achieve nearly 85% accuracy.

The generality of the features learned in our approach is evident from the fact we can achieve good human activity classification even when we learn codes from data sets of a very different character. Following Cadieu and Olshausen [3], we learned sparse codes using video from the BBC Motion Gallery — primarily clips of stampeding wildebeests and stalking leopards — and then applied these codes to achieve than approximately 84% accuracy on the KTH dataset of human behaviors. We also learned sparse codes using the considerably smaller Weizmann dataset and then used these codes to achieve a similar level of performance on the KTH dataset. See `http://www.cs.brown.edu/people/tld/publications/09/ijdem/` for a more detailed look at the several hundred experiments are summarized in this paper.

In conclusion, sparse coding of cuboids extracted using Dollár *et al*'s interest point operator appears to be an effective and tractable solution to summarizing human activity in short videos. We have also applied unsupervised learning methods similar to those explored in Niebles *et al* [23] to categorize video taken from German television by van Hateren and Ruderman [16], and — at least qualitatively, which without some form of ground truth is about all you can say — the induced categorizations appear to be quite reasonable. Our next step is to apply these methods to more challenging video collections that feature a wider range of background, significant clutter and intermittent occlusion.

Acknowledgments Piotr Dollár's Computer Vision and Image Processing Toolkit was enormously useful, and we are also grateful to him for allowing us to use his code for activity recognition. Thanks to Charles Cadieu and Pierre Garrigues for their suggestion on solvers for sparse coding. Anand Madhavan and Rajat Raina working with Andrew Ng let us experiment with their GPU-based algorithms for sparse coding. Rajat provided timely feedback on an early draft of this paper and useful advice guiding our use of

graphics processors to speed learning. The folks at AccelerEyes, Gallagher Prior, James Malcolm and John Melanokos, were very helpful in assisting us in the use of their Jacket technology.

Appendix A: Matlab Implementation

```
function X = coord_descent(A, Y)

% Determine basis/data sizes:
[num_units, num_bases] = size(A);
[num_units, num_cases] = size(Y);
% Initialize the coefficients:
X = zeros(num_cases,num_bases);
% Specify maximum iterations:
max_iter = 128;
% Specify sparsity parameter:
gamma = 0.95000;
% Specify progress tolerance:
tolerance = 0.000001;
% Precompute static components:
AtA = A'* A;
YtA = Y' * A;
Pj = diag(AtA)';
% Number possible step sizes:
num_alphas = length(alphas);
% Add no-progress step size:
alphas_plus_zero = [alphas 0.0];
no_progress = num_alphas+1;

% Apply coordinate descent:
for iter = 1:max_iter
  % Compute the gradient vector:
  Y_minus_Ax_t_A = YtA - X * AtA;
  Qj = Y_minus_Ax_t_A + repmat(Pj,[r,1]) .* X;
  Xstar = ( Qj + sign( -Qj ) * gamma ) ./ repmat(Pj,[r,1]);
  % Zero out small coefficients:
  Xstar( abs( -Qj ) < gamma ) = 0;
  D = Xstar - X;
  % Prepare for the line search:
  DtAtA = D * AtA';
  av = sum(0.5 * DtAtA .* D, 2);
  bv = sum(- Y_minus_Ax_t_A .* D, 2);
  % Find the minimizing step size:
  minHx = gamma * norms(X,2);
  % Solve step equations in parallel:
  Hx = ones(num_alphas,num_cases);
  for k = 1:num_alphas
    Hx(k,:) = av * alphas(k) * alphas(k) + ...
      bv * alphas(k) + gamma * norms(X + alphas(k) * D, 2);
  end
  [Hx, I] = min(Hx, [], 1);
  I( ~( Hx' < minHx * ( 1 - tolerance ) ) ) = no_progress;
  % Terminate loop if no progress:
  if all( I == no_progress ); break; end
  % Apply the gradient step update:
  X = X + repmat(alphas_plus_zero(I)',[1,num_bases]) .* D;
end
```

Appendix B: Eigen Implementation

```
void coordinate_descent(MatrixXf A, MatrixXf Y, MatrixXi & X) {

  // Determine basis/data sizes:
  int num_units = A.rows();
  int num_bases = A.cols();
  int num_cases = Y.cols();

  // Initialize the coefficient matrix:
  X = MatrixXf::Zero(num_cases,num_bases);

  // Specify maximum iterations:
  int max_iter = 128;
  // Specify sparsity parameter:
  float gamma = 0.95000;
  // Specify progress tolerance:
  float tolerance = 0.000001;

  // Initialize the matrix AtA:
  MatrixXf AtA = A.transpose() * A;

  // Initialize vector of gradient steps:
  int num_alphas = 5;
  MatrixXf alphas(1,num_alphas);
  alphas << 1.0000, 0.3000, 0.1000, 0.0300, 0.0100;
  MatrixXf alphas_plus_zero(1,num_alphas + 1);
  alphas_plus_zero << alphas, 0.000;

  // Used to track the step indices:
  MatrixXi alphaIndex(1,num_cases);
  // Index indicating no progress:
  int no_progress = num_alphas;
  // Used to replicate alpha steps:
  MatrixXf alpha_replicator = MatrixXf::Ones(num_bases,1);

  // Vector of diagonal elements:
  MatrixXf P = (AtA.diagonal()).transpose();
  // Used to replicate diagonal elements:
  MatrixXf diagonal_replicator = MatrixXf::Ones(num_cases,1);
  // Replicate the diagonal elements:
  MatrixXf P_replicated = diagonal_replicator * P;

  // Initialize the Matrix YtA:
  MatrixXf YtA = Y.transpose() * A;

  // Intermediate matrix results:
  MatrixXf Y_minus_Ax_t_A(num_units,num_cases);
  MatrixXf Q(num_units,num_cases);
  MatrixXf delta(num_units, num_cases);
  MatrixXf DtAtA(num_units,num_cases);

  // Terms used in the line searches:
  MatrixXf av(num_cases,1);
  MatrixXf bv(num_cases,1);
```

```
54    MatrixXf minHx(num_cases,1);
55    MatrixXf Hx(num_cases,1);
56    MatrixXf Xn(num_cases,num_bases);
57
58    // Create vector of selected steps:
59    MatrixXf best_alphas(1, num_cases);
60
61    // Apply coordinate descent:
62    for (int iter = 0; iter < max_iter; ++iter) {
63      // Compute the gradient vector:
64      Y_minus_Ax_t_A = YtA - X * AtA;
65      Q = Y_minus_Ax_t_A + P_replicated.cwise() * X;
66      for (int i = 0; i < Q.rows(); ++i)
67        for (int j = 0; j < Q.cols(); ++j)
68          if (fabs(Q(i,j)) < gamma)
69            Q(i,j) = 0.0;
70          else if ( Q(i,j) < 0 )
71            Q(i,j) += gamma;
72          else
73            Q(i,j) -= gamma;
74      // Now compute the derivative:
75      delta = (Q.cwise() / P_replicated) - X;
76      // Compute terms for line search:
77      DtAtA = delta * AtA.transpose();
78      // Compute a = 0.5 * d' * AtA * d:
79      av = ((DtAtA.cwise() * delta) * 0.5).rowwise().sum();
80      // Compute b = - y_minus_Ax_t_A*d:
81      bv = (Y_minus_Ax_t_A.cwise() * delta * -1.0).rowwise().sum();
82      // Find the minimizing step size:
83      minHx = gamma * X.cwise().abs().rowwise().sum();
84      // Restore the best step indices:
85      alphaIndex.setConstant(no_progress);
86      // Carry out the line search:
87      for (int k = 0; k < num_alphas; ++k) {
88        float alpha = alphas(k);
89        Xn = X + delta * alpha;
90        Hx = av * alpha * alpha +
91                bv * alpha + Xn.cwise().abs().rowwise().sum() * gamma;
92        for (int i = 0; i < num_cases; ++i) {
93          if (Hx(i,0) < (minHx(i,0) * (1 - tolerance))) {
94            alphaIndex(0,i) = k;
95          }
96        }
97        minHx = minHx.cwise().min(Hx);
98      }
99      // Extract selected gradient steps:
100     for (int i = 0; i < num_cases; ++i) {
101       best_alphas(0,i) = alphas_plus_zero(alphaIndex(i));
102     }
103     // Apply the gradient step update:
104     X += (alpha_replicator * best_alphas).transpose().cwise() * delta;
105   }
106 }
```

References

1. Olshausen, B.A., Field, D.J.: Sparse coding with an overcomplete basis set: A strategy employed by V1? Vision Research **37**(23) (1997) 3311–3325
2. Hyvärinen, A., Hurri, J., Väyrynen, J.: Bubbles: a unifying framework for low-level statistical properties of natural image sequences. Journal of the Optical Society of America **20**(7) (2003) 1237–1252
3. Cadieu, C., Olshausen, B.: Learning transformational invariants from time-varying natural images. In Schuurmans, D., Bengio, Y., eds.: Advances in Neural Information Processing Systems 21. MIT Press, Cambridge, MA (2008)
4. Mumford, D.: Neuronal architectures for pattern-theoretic problems. In: Large Scale Neuronal Theories of the Brain. MIT Press (1994) 125–152
5. Friedman, J., Hastie, T., Höfling, H., Tibshirani, R.: Pathwise coordinate optimization. Annals of Applied Statistics **1**(2) (2007) 302–332
6. Lee, H., Battle, A., Raina, R., Ng, A.Y.: Efficient sparse coding algorithms. In Schölkopf, B., Platt, J., Hofmann, T., eds.: Advances in Neural Information Processing Systems 19. MIT Press, Cambridge, MA (2007) 801–808
7. Dollár, P., Rabaud, V., Cottrell, G., Belongie, S.: Behavior recognition via sparse spatio-temporal features. In: Second Joint IEEE International Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance. (October 2005) 65
8. Raina, R., Madhavan, A., Ng, A.: Large-scale deep unsupervised learning using graphics processors. In: Proceedings of the 25th Annual International Conference on Machine Learning. (2009)
9. Gorelick, L., Blank, M., Shechtman, E., Irani, M., Basri, R.: Actions as space-time shapes. Transactions on Pattern Analysis and Machine Intelligence **29**(12) (December 2007) 2247–2253
10. Schuldt, C., Laptev, I., Caputo, B.: Recognizing human actions: A local SVM approach. In: Proceedings of the International Conference on Pattern Recognition, IEEE Computer Society (2004)
11. Dean, T., Corrado, G., Washington, R.: Recursive sparse, spatiotemporal coding. In: Proceedings of the Fifth IEEE International Workshop on Multimedia Information Processing and Retrieval. (December 2009)
12. Brady, N., Field, D.J.: Local contrast in natural images: normalisation and coding efficiency. Perception **29**(9) (2000) 1041–1055
13. Laptev, I., Lindeberg, T.: Space-time interest points. In: Proceedings of the ninth IEEE International Conference on Computer Vision. Volume 1. (2003) 432–439
14. Harris, C., Stephens, M.: A combined corner and edge detector. In: Alvey Vision Conference. (1988) 147–152
15. Olshausen, B.A., Field, D.J.: Natural image statistics and efficient coding. Computation in Neural Systems **7**(2) (1996) 333–339
16. van Hateren, J.H., Ruderman, D.L.: Independent component analysis of natural image sequences yields spatiotemporal filters similar to simple cells in primary visual cortex. Proceedings Royal Society London B **265** (1998) 2315–2320
17. Olshausen, B.: Learning sparse, overcomplete representations of time-varying natural images. In: Proceedings of the IEEE International Conference on Image Processing. Volume 1., IEEE Computer Society (September 2003) 41–44
18. Mallat, S., Zhang, Z.: Matching pursuits with time-frequency dictionaries. IEEE Transactions on Signal Processing **41** (1993) 3397–3415
19. Földiák, P.: Learning invariance from transformation sequences. Neural Computation **3** (1991) 194–200

20. Wiskott, L., Sejnowski, T.: Slow feature analysis: Unsupervised learning of invariances. Neural Computation **14**(4) (2002) 715–770
21. Wang, Z., Li, B.: Human activity encoding and recognition using low-level visual features. In: Proceedings of the 21th International Joint Conference on Artificial Intelligence. (2009)
22. Lowe, D.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2) (November 2004) 91–110
23. Niebles, J., Wang, H., Fei-Fei, L.: Unsupervised learning of human action categories using spatial-temporal words. International Journal of Computer Vision **79**(3) (2008) 299–318
24. Hinton, G., Salakhutdinov, R.: Reducing the dimensionality of data with neural networks. Science **313**(5786) (July 2006) 504–507
25. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in Neural Information Processing Systems 19. MIT Press, Cambridge, MA (2007) 153–160
26. Riesenhuber, M., Poggio, T.: Hierarchical models of object recognition in cortex. Nature Neuroscience **2**(11) (November 1999) 1019–1025