Learning Compact Binary Codes from Higher-Order Tensors via Free-Form Reshaping and Binarized Multilinear PCA

Haiping Lu
Department of Computer Science
Hong Kong Baptist University, China
Email: haiping@hkbu.edu.hk

Jianxin Wu
National Key Lab for Novel Software Technology
Nanjing University, China
Email: wujx2001@nju.edu.cn

and Yu Zhang
Bioinformatics Institute
A*STAR, Singapore
Email: zhangyu@bii.a-star.edu.sg

Abstract—**For big, high-dimensional dense features, it is important to learn compact binary codes or compress them for greater memory efficiency. This paper proposes a Binarized Multilinear PCA (BMP) method for this problem with Free-Form Reshaping (FFR) of such features to higher-order tensors, lifting the structure-modelling restriction in traditional tensor models. The reshaped tensors are transformed to a subspace using multilinear PCA. Then, we unsupervisedly select features and supervisedly binarize them with a minimum-classification-error scheme to get compact binary codes. We evaluate BMP on two scene recognition datasets against state-of-the-art algorithms. The FFR works well in experiments. With the same number of compression parameters (model size), BMP has much higher classification accuracy. To achieve the same accuracy or compression ratio, BMP has an order of magnitude smaller number of compression parameters. Thus, BMP has great potential in memory-sensitive applications such as mobile computing and big data analytics.**

I. Introduction

With the advances of sensor, networking, and storage technologies, we need to learn features, especially *compact* representations, from increasingly big data. For example, now computer vision researchers are dealing with millions of images daily, posing difficulties on both algorithmic and system sides. On the algorithmic side, a proper feature representation should bridge the semantic gap. The Fisher Vector (FV) representation [25] is a popular approach. It is simple to implement, and has applications in other domains, *e.g.*, action recognition in videos [33], [34]. However, FVs are usually *dense* and *high-dimensional*, and consume more memory than what ordinary systems can afford. For example, for a big dataset with millions of images, FV could easily generate raw features occupying several terabytes memory space [38].

One popular solution to obtain compact features is to *compress* high-dimensional dense features by converting them to binary codes, without sacrificing (much) classification accuracy. This can improve the efficiency greatly and has positive impact on *privacy* and *on-device* database storage [4]. One such approach is Product Quantization (PQ) [12] with segment-specific codebooks. Another is hashing-based methods for learning *similarity-preserving* binary codes. Traditional hashing-based methods [1], [20], [16], [8] are only suitable for low-dimensional features. The Bilinear Projection-based Binary Codes (BPBC) [7] is a *bilinear hashing* method that reduces the number of compression parameters using *natural* 2D (matrix) representations of dense features to improve scalability. Yu *et al.* [36] further proposed a Circulant Binary Embedding (CBE) scheme to improve the *time efficiency* of BPBC with similar memory efficiency. Nonetheless, existing feature compression methods have compression ratios only up to 256 and compression parameters can still take large memory space. They are inadequate for extra large scale data or domains requiring small memory footage, *e.g.*, the compact descriptors for visual search (CDVS) standard prefers memory usage to be below 128KB [5].

On the other hand, with the growth of big data, *tensor-based modelling and learning* [13] have received increasing attention recently [39], [9], [27], [22], [26]. The term tensor here refers to multidimensional arrays in mathematics as in [13], which is different from the meanings in physics. Third-order tensors are most commonly studied [29], [35], [23], [28], [2], [37]. Studies on tensors of order higher than three include fourth-order tensors [10], fifth-order tensors [24], and sixth-order tensors [11]. These tensors are formed following some natural structures so we seldom see side-by-side comparison of tensors of different orders, formed from the same data. *Can we model the same dense features as tensors of different orders? Do we always have to follow some natural structure?*

This paper is motivated by the questions above. Different from traditional tensor models that always follow natural structures, we model dense features with higher-order tensor representations using *Free-Form Reshaping* (FFR) and propose a tensor-based feature compression method, *Binarized Multilinear PCA* (BMP). We investigate the classification accuracy, the number of compression parameters (model size), and binary code size for BMP on tensors of different orders. In experimental evaluation on the FV dense features, we compress them up to a ratio of 2^{14} (or 32 bytes per image), with FFR and BMP showing *great memory efficiency* and *comparable accuracy*.

The three key contributions of this paper are:

- **Free-form reshaping.** We model dense features with

free-form reshaping to *higher-order tensors* (3D to 6D) for more compact representations and more effective compression, lifting the *structure-modelling restriction* in traditional tensor models, while achieving great memory efficiency and comparable accuracy.
- **Min-error binarization.** BMP learns binary codes through binarizing selected multilinear principal components of dense feature tensors with a novel *minimum-error-based binarization* scheme to improve classification accuracy.
- **Very-high compression ratio evaluation.** We evaluate BMP through the most "aggressive" feature compression studies with ratio up to 2^{14} and studies of the trade-offs between classification accuracy, number of compression parameters, and binary code size.

II. Preliminaries

In this section, we first briefly introduce some notations and operations related to tensor. Next, we review a popular dense visual feature representation: the Fisher Vector.

A. Notations and Basic Operations

We denote vectors by lowercase boldface letters, *e.g.*, $\mathbf{x}$; matrices by uppercase boldface letters, *e.g.*, $\mathbf{U}$; and tensors by calligraphic letters, *e.g.*, $\mathcal{A}$. Their elements are denoted with indices in parentheses. *Indices* are denoted by lowercase letters and span the range from 1 to the uppercase letter of the index whenever appropriate, *e.g.*, $n = 1, 2, \ldots, N$.

Tensor is a generalization of vector or matrix. The number of dimensions N of a tensor defines its *order*. We also refer to Nth-order tensors as ND tensors, *e.g.*, 3rd-order tensors as 3D tensors. An Nth-order tensor is denoted as $\mathcal{A} \in \mathbb{R}^{I_1 \times I_2 \times \cdots \times I_N}$, addressed by N indices i_n, $n = 1, \ldots, N$ (i_n addresses the n-mode). An n-mode matrix or vector is denoted as $\mathbf{A}^{(n)}$ or $\mathbf{a}^{(n)}$, respectively.

The n*-mode product* of a tensor $\mathcal{A}$ by a matrix $\mathbf{U} \in \mathbb{R}^{J_n \times I_n}$, denoted by $\mathcal{A} \times_n \mathbf{U}$, is a tensor with entries:

$$(\mathcal{A} \times_n \mathbf{U})(i_1, \ldots, i_{n-1}, j_n, i_{n+1}, \ldots, i_N) = \sum_{i_n} \mathcal{A}(i_1, \ldots, i_N) \cdot \mathbf{U}(j_n, i_n). \quad (1)$$

The n*-mode vectors* of $\mathcal{A}$ are the I_n-dimensional vectors obtained by varying i_n, keeping all the other indices fixed.

The *scalar product* of two same-size tensors $\mathcal{A}, \mathcal{B} \in \mathbb{R}^{I_1 \times \cdots \times I_N}$ is defined as:

$$\langle \mathcal{A}, \mathcal{B} \rangle = \sum_{i_1} \cdots \sum_{i_N} \mathcal{A}(i_1, \ldots, i_N) \cdot \mathcal{B}(i_1, \ldots, i_N). \quad (2)$$

The *Frobenius norm* of $\mathcal{A}$ is defined in scalar product as

$$\parallel \mathcal{A} \parallel_F = \sqrt{\langle \mathcal{A}, \mathcal{A} \rangle}. \quad (3)$$

B. Fisher Vector: Dense Visual Features

Here we discuss a typical example of dense features popular in computer vision: the Fisher Vector representation [25]. In this representation, an image is firstly defined as a collection of local descriptions. At a regular grid of locations, local descriptors such as SIFT are extracted from small local neighborhoods of each position. FV smartly reduces this collection of microscopic observations into a fixed-length, long vector.

Local descriptors are assumed to be generated from a GMM model with W diagonal covariance Gaussians, whose parameters are denoted as (π, μ, σ). A local descriptor $\mathbf{z} \in \mathbb{R}^H$ is then described by its gradient in the GMM model (gradient of the log-likelihood with respect to the mixing, mean, and covariance parameters). An image is then represented as the summation of gradients of all local descriptors.

We ignore the mixing parameter π, and set $H = 64$ and $W = 128$. To catch image variation, spatial pyramid matching [15] is used. Supposing $S = 8$ spatial regions are used, the FV has a length of $L = HS(W + W)$, or 131,072(=2^{17}) with typical choices of H, S, and W above. Usually, each real value takes 4 bytes to store. Thus, in training, the raw FV features for one hundred, one thousand, ten thousand, and one million training images will total 50MB, 500MB, 4.88GB, and 488GB, respectively, which can be too big to keep in memory. In testing, the stored model alone (using LIBLINEAR [6]) will use 5, 50, or 500MB memory for a task involving 10, 100 or 1,000 classes, respectively, which can be demanding for memory-constrained devices.

III. Proposed Feature Compression Method

We propose a *binarized multilinear PCA* for learning compact binary codes from higher-order tensors obtained by *free-form reshaping* of dense features, as shown in Fig. 1. BMP first performs multilinear PCA (MPCA) to transform tensor features into a subspace. Then, the top J features are selected to form a compact vector and a novel minimum-error-based binarization is performed to get binary codes. The BMP algorithm is summarized in Algorithm 1.

A. Free-Form Reshaping to Higher-Order Tensors

Dense features are usually represented as vectors. Only recently, Gong *et al.* [7] first studied their *natural second-order (matrix) representations* to improve memory efficiency. Inspired by [7], we hope to achieve *even greater memory efficiency* by representing dense features as *higher-order tensors*. However, traditional tensor-based methods model data by following some *natural structures* to capture *correlations* among neighboring elements [19], [17], which restricts the *representation flexibility*.

On the other hand, different from many other data modelled by tensors [19], [17], dense features such as Fisher Vectors have **very weak correlations**. A recent study in [38] shows that more than 99.9% of pairwise correlations between FV features are less than 0.2. Motivated by this study, we propose **free-form reshaping** to higher-order tensors to lift the *structure-modelling restriction* in traditional tensor-based

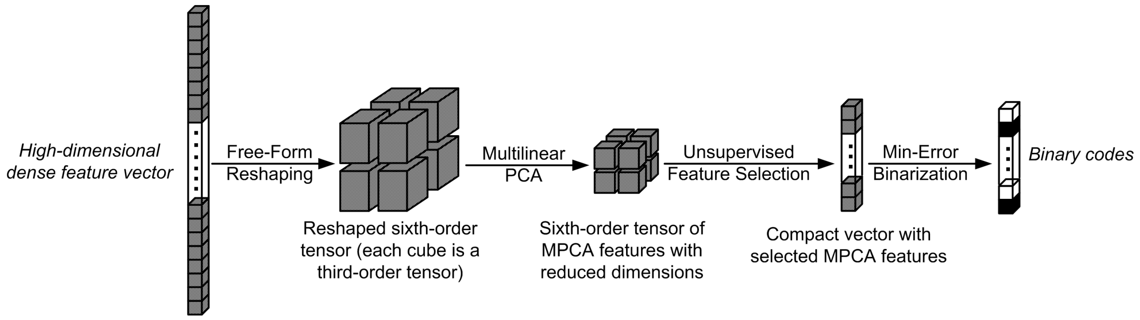

High-dimensional dense feature vector
Free-Form Reshaping
Reshaped sixth-order tensor (each cube is a third-order tensor)
Multilinear PCA
Sixth-order tensor of MPCA features with reduced dimensions
Unsupervised Feature Selection
Compact vector with selected MPCA features
Min-Error Binarization
Binary codes

Figure 1. The proposed *binarized multilinear PCA* with *free-form reshaping* (FFR) for dense feature compression. In the FFR shown, a high-dimensional dense feature vector is reshaped to a sixth-order tensor, which is visualized as stacking third-order tensors in 3D.

Algorithm 1 Binarized Multilinear PCA (BMP)

```
1: Input: $M$ (reshaped) $N$th-order tensor samples $\{\mathcal{X}_m \in \mathbb{R}^{I_1\times\cdots\times I_N}, m = 1,\ldots,M\}$ with class labels $\mathbf{c} \in \mathbb{Z}^M$, the binary code length $J$.
2: Perform MPCA on $\{\mathcal{X}_m\}$ to get projection matrices $\{\mathbf{U}^{(n)}\}$.
3: Project $\{\mathcal{X}_m\}$ with $\{\mathbf{U}^{(n)}\}$ by Eqn. (5) to obtain $\{\mathcal{Y}_m\}$.
4: Select $J$ features from each tensor feature $\mathcal{Y}_m$ to form vector feature $\mathbf{y}_m$ capturing the largest scatter defined in Eqn. (9). Record the respective indices of selected features in $\{\Gamma_j\}$.
5: Binarize vector features $\{\mathbf{y}_m\}$ to obtain binary codes $\{\mathbf{b}_m\}$ with thresholds $\{T_j\}$ determined by Algorithm 2.
6: Output: The projection matrices $\{\mathbf{U}^{(n)}\}$, the indices of selected features $\{\Gamma_j\}$, and the thresholds $\{T_j\}$.
```

Table I
THE DIMENSIONS $\{I_n\}$ OF THE NTH-ORDER FFR TENSOR MODEL OF DENSE FEATURES IN $\mathbb{R}^{2^{17}}$ IN BMP.

N	Dimensions $I_1 \times I_2 \times \cdots \times I_N$
3	$64 \times 64 \times 32$
4	$32 \times 16 \times 16 \times 16$
5	$8 \times 8 \times 8 \times 16 \times 16$
6	$8 \times 8 \times 8 \times 8 \times 8 \times 4$

methods for *greater memory efficiency*. For example, we model typical Fisher Vectors in $\mathbb{R}^{131,072}$ with tensors of order $N = 3, 4, 5, 6$ as in Table I. We can view a 4th-order, 5th-order, or 6th-order tensor as a vector, matrix, or third-order tensor of 3D tensors, respectively, as depicted in Fig. 1. We will show in experiments that FFR-based models achieve accuracy comparable to traditional natural structure model, e.g., for BPBC [7].

We choose each mode dimension to be a power of two for simplicity, since the dense feature length in our experimental studies is a power of two (2^{17}). There can be other choices for the N dimensions. While reshaping into higher-order tensors, we prefer to have "balanced" dimensions in all N dimensions, which offers better memory efficiency and tends to give better accuracy as well in our studies.

B. Multilinear PCA Projection

MPCA is a multilinear extension of PCA to tensors [18]. It is closely related to the higher-order singular value decomposition (HOSVD) [3] and the Tucker decomposition [30].

Although MPCA has been mainly applied to third-order tensors, it is developed for learning features directly from general higher-order tensors. Consider M Nth-order tensors $\{\mathcal{X}_1, \ldots, \mathcal{X}_M \in \mathbb{R}^{I_1\times I_2\times\cdots I_N}\}$, MPCA seeks a multilinear tensor-to-tensor projection (TTP)

$$\{\mathbf{U}^{(n)} \in \mathbb{R}^{I_n\times J_n}, n = 1, 2, \cdots, N\}, \tag{4}$$

where $J_n \leq I_n$ in general, that maps the original tensor space $\mathbb{R}^{I_1} \bigotimes \cdots \bigotimes \mathbb{R}^{I_N}$ into a modewise-orthogonal tensor subspace $\mathbb{R}^{J_1} \bigotimes \cdots \bigotimes \mathbb{R}^{J_N}$:

$$\mathcal{Y}_m = \mathcal{X}_m \times_1 \mathbf{U}^{(1)^T} \times_2 \mathbf{U}^{(2)^T} \cdots \times_N \mathbf{U}^{(N)^T}, \tag{5}$$

where $m = 1, \ldots, M$ and $\mathcal{Y}_m \in \mathbb{R}^{J_1\times J_2\times\cdots\times J_N}$.[1]

Figure 2 demonstrates the TTP of a third-order tensor. It can be viewed as N projections, one in each mode by an n-mode projection matrix $\mathbf{U}^{(n)}$, as shown in Fig. 2(a). Figure 2(b) shows how the projection in the first (column) mode is done, which is a projection of 1-mode vectors (columns) by $\mathbf{U}^{(1)} \in \mathbb{R}^{I_1\times J_1}$ from $\mathbb{R}^{I_1}$ to $\mathbb{R}^{J_1}$ (see Eqn. 1):

$$\tilde{\mathcal{Y}}_m = \mathcal{X}_m \times_1 \mathbf{U}^{(1)^T}. \tag{6}$$

This can be done through taking inner products between each 1-mode vector and each column of $\mathbf{U}^{(1)}$ (*i.e.*, each row of $\mathbf{U}^{(1)^T}$), as shown in the figure.

MPCA solves for a TTP such that the total tensor scatter, defined through the Frobenius norm in Eqn. (3) as

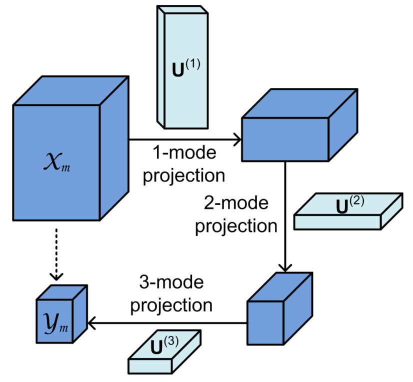
$$\Psi_{\mathcal{Y}} = \sum_{m=1}^{M} \parallel \mathcal{Y}_m - \bar{\mathcal{Y}} \parallel_F^2, \tag{7}$$

is maximized, where

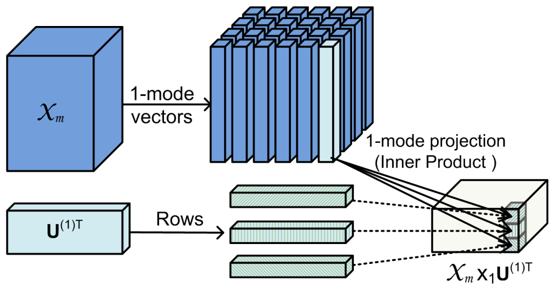
$$\bar{\mathcal{Y}} = \frac{1}{M}\sum_{m=1}^{M} \mathcal{Y}_m \tag{8}$$

is the mean projection. The subspace dimensions $\{J_n\}$ can be determined by specifying Q, the percentage of energy (defined as scatters) to be kept in each mode.

[1]Note that TTP is an alternative interpretation and extension of Tucker decomposition/HOSVD in dimensionality reduction [19].



(a)



$\mathcal{X}_m$
1-mode vectors
1-mode projection (Inner Product)
$\mathbf{U}^{(1)T}$
Rows
$\mathcal{X}_m \times_1 \mathbf{U}^{(1)T}$

(b)

Figure 2. Tensor-to-tensor projection: (a) TTP of a 3rd-order tensor consists of 3 projection matrices, projecting n-mode ($n = 1, 2, 3$) vectors in each mode with an n-mode projection matrix, and (b) the projection in 1-mode (column mode) through inner products between 1-mode vectors (columns) and vectors (rows) from the 1-mode projection matrix.

C. Unsupervised Feature Selection

We can obtain more compact feature vectors $\{\mathbf{y}_m \in \mathbb{R}^J\}$ from $\{\mathcal{Y}_m\}$ through feature selection.[2] We choose an unsupervised feature selection strategy that selects the top J features with the highest scatters (energy) $E_{j_1 \ldots j_N}$ defined as

$$E_{j_1 \ldots j_N} = \sum_{m=1}^{M} \left[\mathcal{Y}_m(j_1, \ldots, j_N) - \bar{\mathcal{Y}}(j_1, \ldots, j_N)\right]^2. \quad (9)$$

The respective indices $(j_1, j_2, \ldots, j_N)$ of the J selected features are recorded in $\{\Gamma_j\}$.

D. Minimum-Error-Based Binarization

To obtain compact binary codes for memory and computational efficiency, we binarize the selected MPCA features $\{\mathbf{y}_m \in \mathbb{R}^J\}$ to produce binary codes $\{\mathbf{b}_m \in \mathbb{B}^J\}$, where $\mathbb{B} = \{0, 1\}$, by choosing a set of thresholds $\{T_j, j = 1, \ldots, J\}$.

One common strategy is to use the *sign function* for simple binarization (assuming zero-mean features), which may lead to limited performance (especially for a large number of classes C). Here, we propose a novel binarization scheme based on *minimizing (binary) classification errors* of two *neighboring classes* for better performance. Our motivation is that since the binarized features will have only two values, the best we can do is to minimize the classification error of a binary classification problem so we choose two neighboring classes to determine the threshold T_j for the j-th features. For convenience of discussion, we drop the feature index j below. Algorithm 2 summarizes the procedures.

[2]Similar to the success of deep learning [14], we found that an unsupervised selection at this stage outperforms a supervised selection.

Algorithm 2 Minimum-Error-Based Binarization

```
1:  Input: $M$ (scalar) features $\{y_m, m = 1, \ldots, M\}$ with respective
    class labels $\{c_m\}$, where $c_m \in \{1, \ldots, C\}$.
2:  Compute the mean $\mu_c$ and standard deviation $\sigma_c$ for class $c$.
3:  Sort the $C$ classes in ascending order so that $\mu_c \le \mu_d$ if $c \le d$.
4:  for $l = \lfloor C/2 \rfloor, \lfloor C/2 \rfloor + 1, \lfloor C/2 \rfloor - 1, \lfloor C/2 \rfloor + 2, \lfloor C/2 \rfloor - 2, \ldots$
    do
5:      $r \leftarrow l + 1$.
6:      if An optimal threshold $T_{opt}$ exists for $c_l$ and $c_r$ according to
        Eqn. (13) then
7:          Set $T = T_{opt}$ and break.
8:      end if
9:  end for
10: if $T$ is not found then
11:     Set $T = (\mu_l + \mu_r)/2$, where $l = \lfloor C/2 \rfloor$ and $r = l + 1$.
12: end if
13: Output: The binarization threshold $T$.
```

We model each class $c \in \{1, \ldots, C\}$ as Gaussian with mean μ_c and standard deviation σ_c:

$$p_c(y) = \frac{1}{\sigma_c \sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{y - \mu_c}{\sigma_c}\right)^2\right]. \quad (10)$$

Our objective is to find an optimal threshold in classifying two neighboring classes c_l (left class) and c_r (right class) with the minimum error, where $r = l + 1$. We first sort the class means $\{\mu_c\}$ in ascending order. The indices l and r refer to the sorted class indices. Then we search for an optimal binarization/classification threshold T_{opt} for two neighboring classes *starting from the middle*, i.e., $l = \lfloor C/2 \rfloor$, and continuing with $l = \lfloor C/2 \rfloor + 1$, $\lfloor C/2 \rfloor - 1$, $\lfloor C/2 \rfloor + 2$, $\lfloor C/2 \rfloor - 2$, and so on, where $\lfloor \cdot \rfloor$ is the floor function. The optimal threshold T_{opt} is the solution for which the *posteriors* of the two classes equal, i.e.,

$$P_l p_l(T) = P_r p_r(T), \quad (11)$$

where P_l and P_r are the prior probabilities of classes c_l and c_r, respectively, and $p_l(\cdot)$ and $p_r(\cdot)$ are the probability density function of classes c_l and c_r, respectively. In addition, T_{opt} should be between the two class means.

We estimate P_l and P_r as

$$P_l = \frac{N_l}{N_l + N_r} \text{ and } P_r = \frac{N_r}{N_l + N_r}, \quad (12)$$

where N_l and N_r are the number of training samples for classes c_l and c_r, respectively. After substituting Eqn. (12) and Eqn. (10) into Eqn. (11), taking logarithms and simplifying, we obtain T_{opt} as a solution of the following constrained quadratic equation

$$AT^2 + BT + C = 0, \text{ subject to } \mu_l \le T \le \mu_r \quad (13)$$

where

$$A = \sigma_l^2 - \sigma_r^2, \tag{14}$$

$$B = 2(\mu_l \sigma_r^2 - \mu_r \sigma_l^2), \tag{15}$$

$$C = \sigma_l^2 \mu_r^2 - \sigma_r^2 \mu_l^2 + 2\sigma_l^2 \sigma_r^2 \ln(\sigma_r N_l / \sigma_l N_r). \tag{16}$$

The search stops when a T_{opt} is found and we set $T = T_{opt}$. Otherwise, if $B^2 - 4AC < 0$ or the solution $T^* < \mu_l$ or $T^* > \mu_r$ for all neighboring classes, we set the threshold as $T = (\mu_l + \mu_r)/2$, where $l = \lfloor C/2 \rfloor$ and $r = l + 1$.

Remark on why starting from the middle: Here we give a brief argument of preference on the neighboring classes in the middle assuming all neighboring class pairs are *equally-well-separated with no overlapping*. Suppose we have 20 classes, which means there are $\binom{20}{2} = 190$ pairs of classes. Note that after binarization by the proposed method, the within-class scatters are all zero so the classification performance is determined by the between-class scatters. By choosing a threshold in the middle, $\binom{10}{2} \times 2 = 90$ out of 190 class pairs (i.e., **47.37%**) have zero between-class scatters after binarization, which means they are indistinguishable after binarization. While by choosing a threshold at the head or tail, $\binom{19}{2} = 171$ out of 190 class pairs (i.e., **90%**) have zero between-class scatters after binarization. Thus, a threshold in the middle is preferred given equal, non-overlapping neighboring class separation.

E. Discussion

Parameter Settings: Since a feature selection procedure will follow MPCA, Q is set to 100 for simplicity. Assuming order N has been determined, the binary code length J is the only parameter to set. If the desired compression ratio is γ and the original dense feature is a FLOAT (4 bytes) type vector of $L \times 1$, then we need to set the code length $J = \frac{32L}{\gamma}$.

Computational Complexity: The most costly computations in BMP is the MPCA step. For $N \geq 3$, MPCA training has an approximate complexity of $O(N^2 ILM)$, assuming equal dimension I in each mode and $I^N = L$. Note that L and M are fixed for a given training set, and we have a smaller I for a larger N and vice versa. For example, when $L = 10^9$, $I = 1000$ for $N = 3$ while $I = 10$ for $N = 9$. MPCA has been shown to have good convergence and one iteration is often good enough when using full projection truncation to initialize.

Memory I/O Trade-Off: While we have chosen datasets with training samples fitting into the memory, MPCA does not require all training samples to be in the memory and can perform the analysis by reading one sample in at a time, at higher I/O cost [18].

Compression Parameters (Model Size): BMP compresses each tensor sample of size $\prod_{n=1}^{N} I_n (= L)$ to $J \times 1$. The compression parameters (the output of Algorithm 1) consist of the N matrices $\{\mathbf{U}^{(n)} \in \mathbb{R}^{I_n \times J_n}\}$, the J indices $\{\Gamma_j\}$ of the selected features, and the J thresholds $\{T_j\}$. Since we select only J features from the total $\prod_{n=1}^{N} J_n$ features, we only need to keep those columns in $\{\mathbf{U}^{(n)}\}$ corresponding to the selected features and the memory needed and computations in testing will often be lower than the full computations of all $\prod_{n=1}^{N} J_n$ features for a smaller J.

F. Differences with PQ and BPBC

PQ: PQ compresses dense features by dividing a long vector of length L into G segments of length $A = \frac{L}{G}$, and performing K-means for each segment, which is then replaced and approximated by its nearest centroid. Since we store all the GK centroids (*i.e.*, compression parameters) in all G segments, in order to represent a dense feature vector, we only need to store the G indices to respective nearest centroids. That is, we need to store the GK centroids ($4KL$ bytes) and a dense feature vector is reduced from $4L$ bytes to $\frac{G \log_2 K}{8}$ bytes, or a compression ratio of $\frac{32L}{G \log_2 K}$. For example, a setup with $K = 16$ and $L = 2^{17}$ will require 8MB of compression parameters to be stored.

BMP vs. PQ: The key differences are *dense feature representation* (higher-order tensor vs. vector), and *binary coding scheme* (minimum-error-based binarization vs. segment-specific codebooks).

BPBC: BPBC is a *bilinear hashing* method that uses a 2D (matrix) representation $\mathbf{X} \in \mathbb{R}^{I_1 \times I_2}$ of dense features $\mathbf{x} \in \mathbb{R}^L$, where $I_1 I_2 = L$ and $I_1 = H$. Left and right rotation matrices $\mathbf{R}_1 \in \mathbb{R}^{I_1 \times J_1}$ and $\mathbf{R}_2 \in \mathbb{R}^{I_2 \times J_2}$ are learned to maximize the similarity between a vectorization of the binary matrix $\mathrm{sgn}(\mathbf{R}_1^T \mathbf{X} \mathbf{R}_2)$ (*i.e.*, the compressed binary code) and $\mathbf{x}$, where $\mathrm{sgn}(\cdot)$ is the *sign function*. The binary code learned by BPBC requires $\frac{J_1 J_2}{8}$ bytes to store, resulting in a compression ratio of $\frac{32L}{J_1 J_2}$. Compression parameters (*i.e.*, $\mathbf{R}_1$ and $\mathbf{R}_2$) occupies $4(I_1 J_1 + I_2 J_2)$ bytes. As matrices are second-order tensors, it can be viewed as a *second-order* tensor method, which motivated our development of BMP.

BMP vs. BPBC: The key differences are *dense feature representation* (higher-order tensor vs. matrices, free-form reshaping vs. natural structure), *binarization* (minimum-error-based vs. sign function), and *learning process* (binarization separated from feature extraction vs. binarization embedded in feature extraction).

Furthermore, existing feature compression methods have only been evaluated with compression ratio ≤ 256 [32], [7], [25], [36], which may not be enough for large datasets and applications with stringent memory constraints [4]. In the next section, we carry out the most "aggressive" compression experiments, to the best of our knowledge, to study compression ratios ranging from 128 to 16,384.

IV. Experiments

BMP is motivated by the problem of dense feature compression in computer vision. Thus, we perform experimental evaluation on two popular scene recognition datasets, including both outdoor and indoor scenes. All experiments are done on Linux machines with Xeon X5650 (2.67GHz, 2-CPU/6-core) and 32GB RAM using MATLAB R2011a. We evaluate our method against two state-of-the-art feature compression methods: PQ, and BPBC (a *bilinear hashing* method for high-dimensional features).

A. Data Description

The **Scene 15** dataset contains scene images from 15 categories including outdoor and indoor scenes (e.g., mountain and tall building). There are 4,485 images, with image sizes varying around 300 × 250.[3] Following the common protocol of Lazebnik *et al.* [15], we use 100 images per category for training, and all the rest for testing. The train/test split is repeated 10 times and we report the average accuracy rate of this dataset.

Indoor 67 is a challenging dataset consisting of 67 indoor categories, such as dental offices and malls. A pre-specified train/test split of 6,700 images exists for this dataset, following the original protocol of Quattoni and Torralba [21].[4] On average there are 80 training and 20 testing images per category. We report the classification accuracy using this split of training and testing images.

The high-dimensional dense features we use are Fisher Vectors, generated using the VLFeat package [31]. As aforementioned, the FV feature has a length $L = 2^{17}$ (131,072), with each element represented as a FLOAT type of 4 bytes. Thus, the feature length studied here is greater than the maximum feature length tested in [36] ($L_{max} = 51,200$) or [7] ($L_{max} = 105,000$). With this feature length, many feature compression methods, such as LSH [1] and ITQ [8], become infeasible on a commodity computer due to limited memory [7], [36].

B. Parameter Settings

We report results on eight compression ratios $\gamma = 2^{\alpha}$, for $\alpha = 7, \ldots, 14$. Thus, γ ranges from 128 to 16,384 so our maximum compression ratio is much higher than that considered in [36] ($\gamma_{max} = 256$) or [7] ($\gamma_{max} = 64$). Respective binary code lengths are from $J = 2^8$ (32 bytes, equivalent to the size of only 4 DOUBLE type or 8 FLOAT type numbers, corresponding to $\gamma = 2^{14}$) to $J = 2^{15}$ (4KB, corresponding to $\gamma = 2^7$).

BMP: We test from the 3rd- to 6th-order tensor models with tensor dimensions in Table I and compare the performance variations for different orders. BMP on Nth-order tensor representations is indicated as BMP$_{ND}$. To get various compression ratios, we set the code length $J = \frac{32L}{\gamma}$. We use the MPCA code with the default one iteration.[5]

BPBC: For BPBC, we test not only the 2D dense feature representation $I_1 \times I_2 = 64 \times 2048$ following *natural structure*, where $I_1 = H$ as suggested by Gong *et al.* [7], but also two additional *free-form reshaping* settings 128 × 1024 and 256 × 512, which are more memory-efficient. BPBC on $I_1 \times I_2$ representations is indicated as BPBC$_{I_1 \times I_2}$. The J_1 and J_2 values to get various compression ratios for 64 × 2048 are set as in Table II. For the other two settings 128 × 1024 and 256 × 512, we multiply the respective J_1 values in Table II by 2 and 4, and divide the respective J_2 values in Table II by 2 and 4, respectively. We use the BPBC code with suggested 3 iterations.[6]

PQ: For the PQ method, we use the standard settings with the (segment length, per-segment centroid number) values (A, K) in Table III to get the desired γs. PQ has only one parameter setting for a particular γ.

Classifier: We use LIBLINEAR [6] for classification with the same setting $c = 1$ and $B = 1$.

Table II
BPBC PARAMETERS TO GET VARIOUS COMPRESSION RATIO γ. A DENSE FV FEATURE IS RESHAPED INTO $I_1 \times I_2$ AND THE TWO ROTATION MATRICES ARE $\mathbf{R}_1 \in \mathbb{R}^{I_1 \times J_1}$ AND $\mathbf{R}_2 \in \mathbb{R}^{I_2 \times J_2}$. THIS TABLE INDICATES J_1 AND J_2 SETTINGS FOR $I_1 \times I_2 = 64 \times 2048$.

γ	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
J_1	32	32	16	16	16	16	8	4
J_2	1024	512	512	256	128	64	64	64

Table III
PQ PARAMETERS TO GET VARIOUS COMPRESSION RATIO γ, WHERE A IS THE SEGMENT LENGTH AND K IS THE NUMBER OF CENTROIDS FOR EACH SEGMENT.

γ	2^7	2^8	2^9	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
A	8	8	16	32	64	128	256	512
K	4	2	2	2	2	2	2	2

C. Results and Discussions

We first study the minimum-error-based binarization, and then examine BMP performance against BPBC and PQ in terms of classification accuracy, number of compression parameters (model size), and per-image binary code size.

Effectiveness of Min-Error-Based Binarization: Here we compare the proposed minimum-error-based binarization against the sign-function binarization. Figure 3 plots the classification accuracy of BMP$_{3D}$ and BMP$_{6D}$, in percentage (%), against the per-image binary code size in bytes in log scale for illustration. The proposed binarization method outperforms the sign-function binarization in most cases. The improvement is better for a larger N (higher-order tensor models, i.e., BMP$_{6D}$). In addition, it is more effective for the more challenging case of Indoor 67 with a larger $C = 67$ than Scene 15 with $C = 15$.

Accuracy vs. Number of Compression Parameters: Figure 4 depicts the classification performance of BMP, BPBC, and PQ against the number of compression parameters in log scale. We also include the results of using the dense FV features directly without compression as the "baseline" in the figure (on top). We observe that to achieve the same accuracy, the number of compression parameters of BMP is an order of magnitude smaller than those of BPBC and PQ, with higher-order BMP (*e.g.*, BMP$_{6D}$) gives smaller sizes than lower-order

[3]http://www-cvr.ai.uiuc.edu/ponce_grp/data/scene_categories/.
[4]Both the indoor scene images and their train/test split of images are available at http://web.mit.edu/torralba/www/indoor.html.
[5]Code available at http://www.dsp.utoronto.ca/~haiping/MSL.html.
[6]Code available at http://www.unc.edu/~yunchao/bpbc.htm

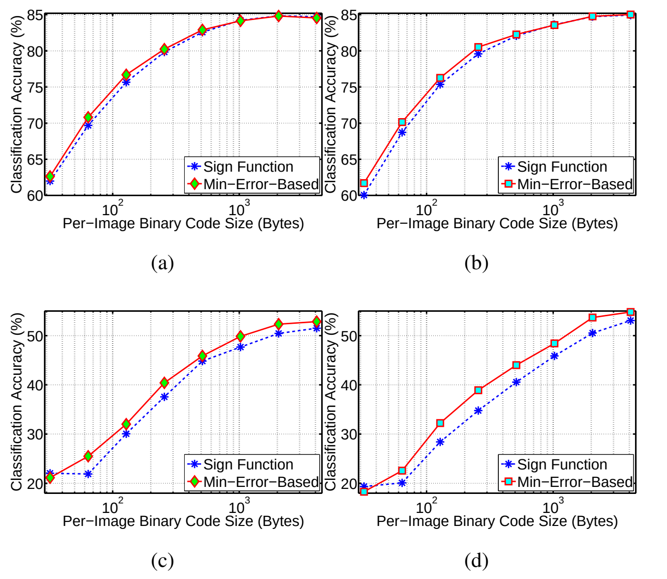


Figure 3. Comparison of BMP classification accuracies (%) with sign function and minimum-error-based binarization schemes for various per-image binary code sizes in bytes (log scale): (a) BMP_{3D} on Scene 15; (b) BMP_{6D} on Scene 15, (c) BMP_{3D} on Indoor 67; (d) BMP_{6D} on Indoor 67.

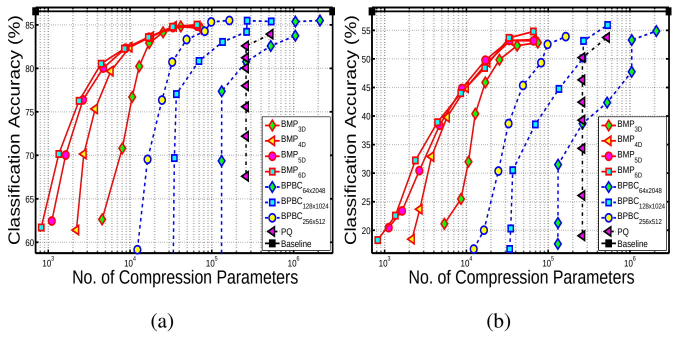


Figure 4. Classification accuracy (%) versus number of compression parameters (log scale): (a) Scene 15; (b) Indoor 67.

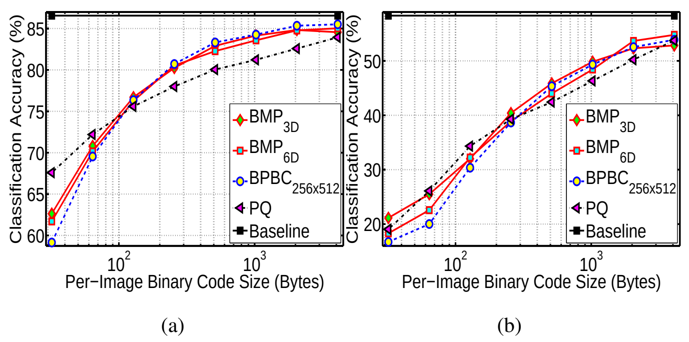


Figure 5. Classification accuracy (%) versus per-image binary code size in bytes (log scale), starting from 32 bytes on the left: (a) Scene 15; (b) Indoor 67.

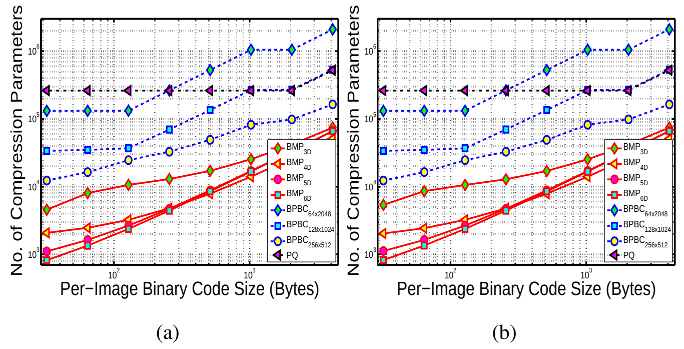


Figure 6. Number of compression parameters (log scale) versus per-image binary code size in bytes (log scale), starting from 32 bytes on the left: (a) Scene 15; (b) Indoor 67.

BMP (*e.g.*, BMP_{3D}). On the other hand, if we consider the classification accuracy with the same number of compression parameters, BMP gives much higher accuracy than BPBC and PQ. Thus, BMP has big advantages in *memory-sensitive applications*.

For BPBC, the *FFR* versions ($BPBC_{256\times512}$ and $BPBC_{128\times1024}$) achieved comparable accuracy as the original $BPBC_{64\times2048}$ following *natural structure*, while they need much smaller number of compression parameters.

Even with the highest compression $\gamma = 2^{14}$, the compression parameters of PQ and the original BPBC still take about 1MB and 500KB of memory, respectively, both exceeding the 128KB preferred by the *CDVS standard* [5]. In comparison, the compression parameters of $BPBC_{256\times512}$ (with FFR) and BMP_{6D} only take about **50KB** and **3KB** of memory for $\gamma = 2^{14}$, respectively.

Accuracy vs. Binary Code Size: Figure 5 shows the classification accuracy against the per-image binary code size in bytes in log scale. For better clarity, we only show BPBC with the most memory-efficient (FFR) setting of 256×512, and the 3rd- and 6th-order BMPs here (together with PQ and baseline). The results with other settings of BPBC and BMP do not deviate much from the respective results for BPBC and BMP in the figure. We can observe that without considering the number of compression parameters, for the same per-image binary code size, BMP (with both settings) outperforms BPBC for small code sizes (high compression ratios). For large code sizes (low compression ratios), BMP is slightly inferior to BPBC in general. For medium code sizes, BMP and BPBC have comparable accuracy. BMP outperforms PQ mainly in the mid-to-large code size range.

Another observation is that for large code sizes, BMP performance may drop slightly (e.g., BMP_{3D} on Scene 15), when more features are used. Therefore, some additional features are not useful for classification and an additional feature selection step could improve performance further.

Number of Compression Parameters vs. Binary Code Size: In the last comparison, we plot the number of compression parameters against per-image binary code size in Fig. 6. The observations here are similar to those in Fig. 4. BMP gives an order of magnitude smaller number of compression parameters for the same per-image binary code size.

V. Conclusion

We presented a memory-efficient feature compression method for learning compact binary codes from high-dimensional dense features, named as *Binarized Multilinear PCA* (BMP). BMP models dense features such as Fisher Vectors with *free-form reshaping* to higher-order tensors to lift the *structure-modelling restriction* in traditional tensor models. The BMP algorithm consists of multilinear PCA for a subspace transformation on reshaped tensors, an unsupervised feature selection to produce compact feature vectors, and a novel *minimum-error-based binarization* scheme to get compact binary codes for classification.

We evaluated BMP on two scene recognition datasets against state-of-the-art feature compression algorithms with very-high compression ratios studied. The results show that BMP achieves *high memory efficiency with comparable accuracy*. Furthermore, higher-order BMP enjoys higher memory efficiency. Therefore, BMP has great potential in learning binary codes from dense features in memory-constrained domains.

References

[1] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388, 2002. 1, 6

[2] S. Chen, M. R. Lyu, I. King, and Z. Xu. Exact and stable recovery of pairwise interaction tensors. In *Advances in Neural Information Processing Systems 26*, pages 1691–1699, 2013. 1

[3] L. De Lathauwer, B. De Moor, and J. Vandewalle. A multilinear singular value decomposition. *SIAM Journal of Matrix Analysis and Applications*, 21(4):1253–1278, 2000. 3

[4] L. Duan, J. Lin, J. Chen, T. Huang, and W. Gao. Compact descriptors for visual search. *IEEE Multimedia*, 21(3):30–40, 2014. 1, 5

[5] L.-Y. Duan, F. Gao, J. Chen, J. Lin, and T. Huang. Compact descriptors for mobile visual search and MPEG CDVS standardization. In *IEEE International Symposium on Circuits and Systems*, pages 885–888, 2013. 1, 7

[6] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008. 2, 6

[7] Y. Gong, S. Kumar, H. A. Rowley, and S. Lazebnik. Learning binary codes for high-dimensional data using bilinear projections. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 484–491, 2013. 1, 2, 3, 5, 6

[8] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011. 1, 6

[9] B. Hutchinson, L. Deng, and D. Yu. Tensor deep stacking networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1944–1957, 2013. 1

[10] M. Ishteva, H. Park, and L. Song. Unfolding latent tree structures using 4th order tensors. In *Proc. International Conference on Machine Learning*, pages 316–324, 2013. 1

[11] M. Ishteva, L. Song, H. Park, A. Parikh, and E. Xing. Hierarchical tensor decomposition of latent tree graphical models. In *Proc. International Conference on Machine Learning*, pages 334–342, 2013. 1

[12] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, 2011. 1

[13] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM Review*, 51(3):455–500, 2009. 1

[14] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25*, pages 1106–1114, 2012. 4

[15] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages II:2169–2178, 2006. 2, 6

[16] W. Liu, J. Wang, Y. Mu, S. Kumar, and S.-F. Chang. Compact hyperplane hashing with bilinear functions. In *Proc. International Conference on Machine Learning*, pages 17–24, 2012. 1

[17] H. Lu, K. N. Plataniotis, and A. Venetsanopoulos. *Multilinear Subspace Learning: Dimensionality Reduction of Multidimensional Data*. CRC Press, 2013. 2

[18] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. MPCA: Multilinear principal component analysis of tensor objects. *IEEE Transactions on Neural Networks*, 19(1):18–39, 2008. 3, 5

[19] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos. A survey of multilinear subspace learning for tensor data. *Pattern Recognition*, 44(7):1540–1551, 2011. 2, 3

[20] M. Norouzi and D. M. Blei. Minimal loss hashing for compact binary codes. In *Proc. International Conference on Machine Learning*, pages 353–360, 2011. 1

[21] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 413–420, 2009. 6

[22] M. Rogers, L. Li, and S. Russell. Multilinear dynamical systems for tensor time series. In *Advances in Neural Information Processing Systems*, pages 2634–2642, 2013. 1

[23] B. Romera-Paredes, H. Aung, N. Bianchi-Berthouze, and M. Pontil. Multilinear multitask learning. In *Proc. International Conference on Machine Learning*, pages 1444–1452, 2013. 1

[24] B. Romera-Paredes and M. Pontil. A new convex relaxation for tensor completion. In *Advances in Neural Information Processing Systems 26*, pages 2967–2975, 2013. 1

[25] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the Fisher Vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013. 1, 2, 5

[26] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in Neural Information Processing Systems 26*, pages 926–934, 2013. 1

[27] Y. Tang, R. Salakhutdinov, and G. Hinton. Tensor analyzers. In *Proc. International Conference on Machine Learning*, pages 616–623, 2013. 1

[28] R. Tomioka and T. Suzuki. Convex tensor decomposition via structured Schatten norm regularization. In *Advances in Neural Information Processing Systems 26*, pages 1331–1339, 2013. 1

[29] R. Tomioka, T. Suzuki, K. Hayashi, and H. Kashima. Statistical performance of convex tensor decomposition. In *Advances in Neural Information Processing Systems 24*, pages 972–980, 2011. 1

[30] L. R. Tucker. Some mathematical notes on three-mode factor analysis. *Psychometrika*, 31(3):279–311, 1966. 3

[31] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/, 2008. 6

[32] A. Vedaldi and A. Zisserman. Sparse kernel approximations for efficient classification and detection. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2320–2327, 2012. 5

[33] H. Wang and C. Schmid. Action recognition with improved trajectories. In *IEEE International Conference on Computer Vision*, pages 3551–3558, 2013. 1

[34] J. Wu, Y. Zhang, and W. Lin. Towards good practices for action video encoding. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 2577–2584, 2014. 1

[35] K. Yoshii, R. Tomioka, D. Mochihashi, and M. Goto. Infinite positive semidefinite tensor factorization for source separation of mixture signals. In *Proc. International Conference on Machine Learning*, pages 576–584, 2013. 1

[36] F. X. Yu, S. Kumar, Y. Gong, and S.-F. Chang. Circulant binary embedding. In *Proc. International Conference on Machine Learning*, pages 946–954, 2014. 1, 5, 6

[37] X. Zhang, D. Wang, Z. Zhou, and Y. Ma. Simultaneous rectification and alignment via robust recovery of low-rank tensors. In *Advances in Neural Information Processing Systems 26*, pages 1637–1645, 2013. 1

[38] Y. Zhang, J. Wu, and J. Cai. Compact representation for image classification: To choose or to compress? In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 907–914, 2014. 1, 2

[39] Q. Zhao, C. F. Caiafa, D. P. Mandic, L. Zhang, T. Ball, A. Schulze-bonhage, and A. S. Cichocki. Multilinear subspace regression: An orthogonal tensor decomposition approach. In *Advances in Neural Information Processing Systems 24*, pages 1269–1277, 2011. 1