The Thirty-Second AAAI Conference
on Artificial Intelligence (AAAI-18)

Tensorized Projection for High-Dimensional Binary Embedding

Weixiang Hong, Jingjing Meng, Junsong Yuan
School of Electrical and Electronic Engineering,
Nanyang Technological University, Singapore
{wxhong,jingjing.meng,JSYUAN}@ntu.edu.sg

Abstract

Embedding high-dimensional visual features (d-dimensional) to binary codes (b-dimensional) has shown advantages in various vision tasks such as object recognition and image retrieval. Meanwhile, recent works have demonstrated that to fully utilize the representation power of high-dimensional features, it is critical to encode them into long binary codes rather than short ones, *i.e.*, $b \sim \mathcal{O}(d)$ (Sánchez and Perronnin 2011). However, generating long binary codes involves large projection matrix and high-dimensional matrix-vector multiplication, thus is memory and computationally intensive. To tackle these problems, we propose Tensorized Projection (TP) to decompose the projection matrix using Tensor-Train (TT) format, which is a chain-like representation that allows to operate tensor in an efficient manner. As a result, TP can drastically reduce the computational complexity and memory cost. Moreover, by using the TT-format, TP can regulate the projection matrix against the risk of over-fitting, consequently, lead to better performance than using either dense projection matrix (like ITQ (Gong and Lazebnik 2011)) or sparse projection matrix (Xia et al. 2015). Experimental comparisons with state-of-the-art methods over various visual tasks demonstrate both the efficiency and performance advantages of our proposed TP, especially when generating high dimensional binary codes, *e.g.*, when $b \geq d$.

1. Introduction

Nearest neighbor (NN) search is a fundamental research topic in artificial intelligence (Shakhnarovich, Darrell, and Indyk 2006). The straightforward solution, linear scan, is memory intensive and computationally expensive in large-scale high-dimensional cases; hence approximate nearest neighbor (ANN) search is usually favored in practice.

Binary embedding (Gong and Lazebnik 2011; Yu et al. 2014; Wang et al. 2016), which aims at encoding high-dimensional feature vectors to compact binary codes, has recently arisen as an effective and efficient way for ANN search. By encoding high-dimensional features into binary codes, one can perform rapid ANN search because (1) operations with binary vectors (such as computing Hamming distance) are very fast thanks to hardware support, and (2) the entire dataset can fit in (fast) memory rather than slow memory or disk.

Representation learning using deep neural networks (DNN) (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016) has shown that DNN features are useful for various vision tasks such as object classification and image retrieval. Unlike traditional hand-crafted features like SIFT (Lowe 2004) and GIST (Oliva and Torralba 2001), these DNN features are usually of thousands of dimensions or even more. Meanwhile, although compact binary codes are preferred to save the storage, recent works have demonstrated that the long codes can bring superior performance to compact ones, especially when the feature vectors are of thousands of dimensions (Gong et al. 2013; Sánchez and Perronnin 2011; Xia et al. 2015; Yu et al. 2014). For example, the long binary codes of 4096 dimensions can achieve the mAP of 82% on DNN-4096 dataset (Xia et al. 2015), while the mAP of 256-dimensional binary codes is only 51%.

However, the large projection matrix for generating long binary codes can cause two challenges: (1) the high computational cost of high-dimensional matrix-vector multiplications, and (2) the risk of over-fitting. For the first challenge, it has been noticed that for high-dimensional input feature vector of dimensionality d, the length b of binary codes required to achieve reasonable accuracy is usually $\mathcal{O}(d)$ (Sánchez and Perronnin 2011; Gong et al. 2013; Yu et al. 2014). When d is large and $b \sim \mathcal{O}(d)$, the projection matrix $\mathrm{R} \in \mathbb{R}^{b \times d}$ could involve millions or even billions of parameters. Such a high cost is not favorable when we encode a big dataset of visual features, or when the computational resource is a concern, *e.g.*, on a mobile platform. For the second challenge, there have been efforts to address it by regulating the projection matrix and reducing the number of free parameters. Interestingly, such regularizations may also bring fast matrix-vector multiplication and improve the computational efficiency.

Representative works for high-dimensional binary embedding include Bilinear Projection(BP) (Gong et al. 2013), Sparse Projection(SP) (Xia et al. 2015), Circulant Binary Embedding(CBE) (Yu et al. 2014), Fast Orthogonal Projection (KBE) (Zhang et al. 2015) and Fried Binary Embedding (FBE) (Hong, Yuan, and Das Bhattacharjee 2017), *etc.* Unfortunately, these existing methods either achieve high accuracy but still suffer from the $\mathcal{O}(d^2)$ complexity like SP, or are promising in efficiency but at the cost of accuracy such as KBE, CBE and FBE. In detail, CBE and FBE cannot directly learn arbitrary shapes of projection matrix, several tricks such as zero padding, stacking and bits dropping are exploited by CBE and FBE to learn projection matrix of general shapes. These tricks introduce inconsistency between training and testing phases, thus, could lead to undesirable accuracy loss. Moreover, BP is not applicable to generating binary codes that are longer than the original feature vector.

To generate long binary codes efficiently while preserving accuracy, we propose a novel approach, Tensorized Projection (TP). The idea is to decompose the projection matrix using the Tensor-Train (TT) decomposition (Oseledets 2011), which is a chain-like format that enables efficient operation with tensors. The decomposition into TT-format can facilitate fast matrix-vector multiplications. Moreover, the ultimate projection matrix would have restricted freedom due to the inherent structure in TT-format. For instance, when encoding a 4096-dimensional feature vector into 4096-bit binary codes, our TP has only 1252 tunable parameters, which are only 0.1% of SP (Xia et al. 2015) and 0.01% of ITQ (dense matrix) (Gong and Lazebnik 2011). Restricted freedom is naturally against overfitting, and could lead to good generalization performance as shown in our experiments. Another side benefit of TP is that TT-format can be efficiently stored. As a result, the memory cost is also reduced. Extensive experiments show that our approach not only achieves competitive performance in compact-bit case but also outperforms state-of-the-art methods in long-bit scenario.

2. Related Work

A good review of binary embedding can be found in (Wang et al. 2017). Here we focus on several closely related works, including Iterative Quantization (ITQ) and its five variants. The Iterative Quantization is essential to the understanding of our work. Its five variants include Bilinear Projection (BP) (Gong et al. 2013), Circulant Binary Embedding (CBE) (Yu et al. 2014), Sparse Projection (SP) (Xia et al. 2015), Fast Orthogonal Projection (KBE) (Zhang et al. 2015) and Fried Binary Embedding (FBE) (Hong, Yuan, and Das Bhattacharjee 2017), which are state-of-the-art works for high-dimensional binary embedding.

Iterative Quantization (ITQ) aims to find the hash codes such that the difference between the data and hash codes is minimized by viewing each bit as the quantization value along the corresponding dimension. It consists of two steps: (1) dimension reduction via PCA; (2) find the hash codes as well as an optimal rotation.

Bilinear Projections (BP) projects a data vector by two smaller matrices rather than a single large matrix, based on the assumption that the data vectors are formulated by reshaping matrices. This assumption of BP is valid for several traditional hand-crafted features like GIST (Oliva and Torralba 2001) and VLAD (Jegou et al. 2012), but not true for learned features such as those learned by deep neural network(DNN) (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016). Also, BP is not applicable to generate binary codes that are longer than the original feature vector.

Circulant Binary Embedding (CBE) imposes a circulant structure on the projection matrix for efficient matrix-vector multiplication. In virtue of the fast Fourier transform, the computational cost of FBE is only $\mathcal{O}(d\log d)$, much less than $\mathcal{O}(d^2)$ of the dense projection method.

Fried Binary Embedding (FBE) decomposes the projection matrix as a series of structured matrices and Hadamard matrices, which they referred as adaptive Fastfood Transform (Le, Sarlós, and Smola 2013; Yang et al. 2015). Similar to CBE, FBE accelerates binary encoding by fast Hadamard transform with a time complexity of $\mathcal{O}(d\log d)$. Although CBE and FBE show promising speedup, the accuracy and the number of tunable parameter of them are still not satisfactory enough as shown by our experiments. Moreover, CBE and FBE are not capable to learn arbitrary shapes of projection matrix directly.

Fast Orthogonal Projection (KBE) decomposes the projection matrix as a series of smaller orthogonal matrices. Similar to CBE, there exists a fast algorithm to compute Kronecker projection with a time complexity of $\mathcal{O}(d\log d)$, therefore, KBE has shown great advantages in terms of efficiency, compare with dense projection method (like ITQ).

Sparse Projection (SP) introduces a sparsity regularizer to achieve efficiency in encoding. They also show that there exist many redundant parameters in dense projection matrix. However, their method requires the percentage of non-zero elements to be around 10% for competitive performance, according to Figure 4 in (Xia et al. 2015), which can be still suffering in case that both *d* and *b* are very large.

3. TT-format

3.1. Definition

A t-dimensional tensor $\mathcal{R}$ is said to be in Tensor-Train format (TT-format) (Oseledets 2011), if every entry of it can be written as the following matrix product:

$$\mathcal{R}(i_1, i_2, \ldots, i_t) = \mathcal{G}_1[i_1]\mathcal{G}_2[i_2]\ldots\mathcal{G}_t[i_t] \qquad (1)$$

where $\mathcal{G}_k[i_k] \in \mathbb{R}^{r_{k-1}\times r_k}$ is a matrix sliced from a 3-dimensional tensor $\mathcal{G}_k \in \mathbb{R}^{r_{k-1}\times n_k\times r_k}$, n_k is the length of the k-th dimension of tensor $\mathcal{R}$. We refer such 3-dimensional tensors $\mathcal{G}_k$ as the cores. To ensure that the matrix product in Equation 1 is a scalar, $r_0 = r_t = 1$ has to be imposed. The sequence $\{r_k\}_{k=0}^{t}$ is called the TT-ranks, and its maximum is considered as the TT-rank of tensor $\mathcal{R}$: $r = max_{k=0,\ldots,t}r_k$. We use $r_{TT}(\mathcal{R})$ to denote the TT-rank of tensor $\mathcal{R}$.

3.2. TT-representations for vector and matrices

TT-format has demonstrated promising results in several recent machine learning topics (Novikov et al. 2014; 2015; Stoudenmire and Schwab 2016). In this work, we propose to integrate it into binary embedding by formulating the projection matrix as TT-format. However, the direct application of the TT-decomposition to a matrix (2-dimensional tensor) coincides with the low-rank matrix factorization and the direct TT-decomposition of a vector is equivalent to explicitly storing its elements. To work with large vectors and matrices efficiently, the TT-format for them is defined in a special manner (Dolgov and Savostyanov 2014).

Consider a vector $\mathrm{x} \in \mathbb{R}^d$, where $d = \prod_{k=1}^t d_k$. We can establish a bijection μ between the coordinate $l \in \{1, \dots, d\}$ of x and a t-dimensional vector-index $\mu(l) = \big(\mu_1(l), \dots, \mu_t(l)\big)$ of the corresponding tensor $\mathcal{X}$, where $\mu_k(l) \in \{1, \dots, d_k\}$. The element of tensor $\mathcal{X} \in \mathbb{R}^{d_1 \times d_2 \times \dots \times d_t}$ is then mapped by the corresponding vector elements: $\mathcal{X}(\mu(l)) = \mathrm{x}(l)$.

Now we introduce the TT-representation of a matrix $\mathrm{R} \in \mathbb{R}^{b \times d}$, where $b = \prod_{k=1}^t b_k$ and $d = \prod_{k=1}^t d_k$. Let bijections $\nu(s) = \big(\nu_1(s), \dots, \nu_t(s)\big)$ and $\mu(l) = \big(\mu_1(l), \dots, \mu_t(l)\big)$ map row and column indices s and l of the matrix R to the t-dimensional vector-indices whose k-th dimensions are of the length b_k and d_k respectively, $k = 1, \dots, t$. From the matrix R we can form a t-dimensional tensor $\mathcal{R}$ whose k-th dimension is of the length $b_k d_k$ and is indexed by the tuple $\big(\nu_k(s), \mu_k(l)\big)$. The tensor $\mathcal{R}$ can then be converted into the TT-format:

$$\begin{aligned} \mathrm{R}(s, l) &= \mathcal{R}\Big(\big(\nu_1(s), \mu_1(l)\big), \dots, \big(\nu_t(s), \mu_t(l)\big)\Big) \\ &= \mathcal{G}_1[\nu_1(s), \mu_1(l)] \dots \mathcal{G}_t[\nu_t(s), \mu_t(l)] \end{aligned} \tag{2}$$

where the tensors $\mathcal{G}_{k=1}^t$ serve as the cores with tuple $\big(\nu_k(s), \mu_k(l)\big)$ being the index. Throughout this paper, we use $\mathrm{R} \sim \mathcal{R}$ to denote such a transformable relationship between a matrix R and its corresponding tensor $\mathcal{R}$.

3.3. Fast matrix-vector multiplication

Matrix-vector multiplication is the fundamental operation in linear algebra. Given a matrix $\mathrm{R} \in \mathbb{R}^{b \times d}$ and a vector $\mathrm{x} \in \mathbb{R}^d$, their multiplication is written as following:

$$\mathrm{y} = \mathrm{R}\mathrm{x} \tag{3}$$

We assume that the matrix R is represented in TT-format as in Equation 2, with cores as $\mathcal{G}_{k=1}^t$ and TT-rank as r. Then Equation 3 can be expressed in the tensor form:

$$\mathcal{Y}(i_1, \dots, i_t) = \sum_{j_1, \dots, j_d} \mathcal{G}_1[i_1, j_1] \dots \mathcal{G}_t[i_t, j_t] \mathcal{X}(j_1, \dots, j_t) \tag{4}$$

Direct application of the TT-matrix-by-vector multiplication for Equation 4 yields the computational complexity $\mathcal{O}(tr^2 \hat{b} \max\{d, b\})$, where $\hat{b} = \max_k b_k$ (Oseledets 2011).

4. Tensorized Projection

A widely considered objective function for binary embedding involves minimizing the quantization loss between the data and hash codes, which is adopted in ITQ and its variants such as BP, FBE *etc*. Following (Wang et al. 2017; Xia et al. 2015), we put dimension reduction and optimal rotation of ITQ into one integrated objective:

$$\begin{aligned} \min_{\mathrm{C},\mathrm{R}} \quad & \|\mathrm{R}\mathrm{X} - \mathrm{C}\|_\mathrm{F}^2 \\ \text{s.t.} \quad & \mathrm{R}^\mathrm{T}\mathrm{R} = \mathrm{I}. \end{aligned} \tag{5}$$

where $\mathrm{X} \in \mathbb{R}^{d \times n}$ is the dataset, C is a b-by-n matrix containing only 1 and -1. The matrix $\mathrm{R} \in \mathbb{R}^{b \times d}$ serves for both dimension reduction and rotation. ITQ (Gong and Lazebnik 2011) solves Equation 5 via alternative update. After finding R, ITQ can produce binary codes using the hash function below:

$$\mathrm{c} = \mathrm{sgn}(\mathrm{R}\mathrm{x}), \tag{6}$$

where $\mathrm{x} \in \mathbb{R}^d$ denotes a data vector, and $\mathrm{sgn}(\cdot)$ is the sign function, which outputs 1 for positive numbers and -1 otherwise.

Although ITQ has shown promising results of binary embedding, its computational cost of the matrix-vector multiplication in Equation 6 is $\mathcal{O}(bd)$, which limits its application to high-dimensional binary embedding. To reduce the cost of calculating Rx, we decompose the projection matrix R using the TT-representation, *i.e.*,

$$\mathrm{R} \sim \mathcal{R} \tag{7}$$

By constraining the TT-rank of $\mathcal{R}$ to less than or equal to some desired threshold r_0, the number of tunable parameters can also be largely reduced. For example, for a projection matrix $\mathrm{R} \in \mathbb{R}^{4096 \times 4096}$, its free parameter will be reduced to 1252 when r_0 is 4. In summary, we attain our optimization objective by putting Equation 5 and 7 together:

$$\begin{aligned} \min_{\mathrm{C},\mathrm{R}} \quad & \|\mathrm{R}\mathrm{X} - \mathrm{C}\|_\mathrm{F}^2 \\ \text{s.t.} \quad & \mathrm{R}^\mathrm{T}\mathrm{R} = \mathrm{I}, \\ & \mathrm{R} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \le r_0. \end{aligned} \tag{8}$$

5. Optimization

Due to the involvement of TT-format, Equation 8 is a more challenging problem compared with Equation 5. Updating any core of $\mathcal{R}$ could cause the violation of the orthonormal constraint on R. To find a feasible solution, we adopt the variable-splitting and penalty techniques in optimization (Courant 1943; Wang et al. 2008). Specifically, we move the orthonormal constraint onto an auxiliary variable $\bar{\mathrm{R}}$ and meanwhile penalize the difference between $\bar{\mathrm{R}}\mathrm{X}$ and RX. As a result, we relax the problem in Equation 8 to the following form:

$$\begin{aligned} \min_{\mathrm{C},\mathrm{R},\bar{\mathrm{R}}} \quad & \|\bar{\mathrm{R}}\mathrm{X} - \mathrm{C}\|_\mathrm{F}^2 + \beta\|\bar{\mathrm{R}}\mathrm{X} - \mathrm{R}\mathrm{X}\|_\mathrm{F}^2 \\ \text{s.t.} \quad & \bar{\mathrm{R}}^\mathrm{T}\bar{\mathrm{R}} = \mathrm{I}, \\ & \mathrm{R} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \le r_0. \end{aligned} \tag{9}$$

where β is a penalty weight. Such a relaxation is similar to Half-Quadratic Splitting (Wang et al. 2008). By introducing an auxiliary variable, the original problem can be separated into feasible sub-problems, and the solution to Equation 9 will converge to that of Equation 8 when $\beta \to \infty$ (Wang et al. 2008). We solve Equation 9 in an alternating manner: update one variable with others fixed.

5.1. Update C

This sub-problem is equivalent to $\min_\mathrm{C} \|\bar{\mathrm{R}}\mathrm{X} - \mathrm{C}\|_\mathrm{F}^2 = \max_\mathrm{C} \sum_{i,j} (\bar{\mathrm{R}}\mathrm{X})_{ij} \mathrm{C}_{ij}$, where i, j are the indexes of matrix elements. Because $\mathrm{C}_{ij} \in \{-1, 1\}$, this problem can be easily solved by $\mathrm{C}_{ij} = \mathrm{sgn}((\bar{\mathrm{R}}\mathrm{X})_{ij})$, or simply

$$\mathrm{C} = \mathrm{sgn}(\bar{\mathrm{R}}\mathrm{X}). \tag{10}$$

5.2. Update $\bar{\mathrm{R}}$

With R fixed, the two terms in Equation 9 are both quadratic on $\bar{\mathrm{R}}$. By some derivations, the problem Equation 9 becomes:

$$\begin{aligned} \min_{\bar{\mathrm{R}}} \quad & \|\bar{\mathrm{R}}\mathrm{X} - \mathrm{Y}\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \bar{\mathrm{R}}^{\mathrm{T}}\bar{\mathrm{R}} = \mathrm{I}, \end{aligned} \tag{11}$$

where $\mathrm{Y} = (\mathrm{C} + \beta\mathrm{RX})/(1+\beta)$. This problem is known as the orthogonal procrustes problem (Gower and Dijksterhuis 2004) and is widely studied in binary embedding (Gong and Lazebnik 2011; Gong et al. 2013; Xia et al. 2015). Equation 11 is minimized as following: first find the SVD of the matrix YX^{T} as $\mathrm{YX}^{\mathrm{T}} = \mathrm{U}\Sigma\mathrm{V}^{\mathrm{T}}$, then let

$$\bar{\mathrm{R}} = \mathrm{UV}^{\mathrm{T}}. \tag{12}$$

According to (Gower and Dijksterhuis 2004), the procrustes problem is solvable only if $b \geq d$. In case that $b < d$, $\bar{\mathrm{R}}^{\mathrm{T}}\bar{\mathrm{R}} = \mathrm{I}$ is no longer a valid constraint because $\mathrm{rank}(\bar{\mathrm{R}}^{\mathrm{T}}\bar{\mathrm{R}}) \leq \min(b, d)$ while $\mathrm{rank}(\mathrm{I})$ is d. To handle the case of $b < d$, we reduce X to b dimensions by $\mathrm{X}' = \mathrm{PX}$, where P is the b-by-d PCA projection matrix corresponding to the largest eigenvalues. Then we replace X with X' in the Equation 11 for solving a b-by-b matrix $\bar{\mathrm{R}}'$. Finally, $\bar{\mathrm{R}}$ is given by $\bar{\mathrm{R}}'\mathrm{P}$. We do not pre-project the data X by PCA in the main problem Equation 11, because the PCA projection matrix P is a dense matrix, using P as pre-processing will ruin all the merits introduced by TT-format.

5.3. Update R

Let $\mathrm{Z} = \bar{\mathrm{R}}\mathrm{X}$, the subproblem related to R is:

$$\begin{aligned} \min_{\mathrm{R}} \quad & ||\mathrm{RX} - \mathrm{Z}||_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \mathrm{R} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \leq r_0. \end{aligned} \tag{13}$$

Minimization of 2-norm is known as least square regression, whose solution is simply given by solving the corresponding linear system, thus, solving Equation 13 is equivalent to solve the following constrained linear system

$$\begin{aligned} & \mathrm{RX} = \mathrm{Z} \\ \text{s.t.} \quad & \mathrm{R} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \leq r_0. \end{aligned} \tag{14}$$

Let $\otimes$ denote Kronecker product, $\mathrm{vec}(\mathrm{X})$ denote the vectorization of the matrix X formed by stacking the columns of X into a single column vector. For any matrices $\mathrm{A}, \mathrm{X}, \mathrm{B}$ and C, we have

$$\begin{aligned} & (\mathrm{B}^{\mathrm{T}} \otimes \mathrm{A})\mathrm{vec}(\mathrm{X}) = \mathrm{vec}(\mathrm{AXB}) = \mathrm{vec}(\mathrm{C}) \\ & (\mathrm{A} \otimes \mathrm{B})^{\mathrm{T}} = \mathrm{A}^{\mathrm{T}} \otimes \mathrm{B}^{\mathrm{T}}. \end{aligned} \tag{15}$$

Moreover, if one can form the matrix products AB and XC, we can also perform

$$(\mathrm{A} \otimes \mathrm{X})(\mathrm{B} \otimes \mathrm{C}) = \mathrm{AB} \otimes \mathrm{XC}. \tag{16}$$

By applying these properties, Equation 14 can be transformed as a constrained least square problem

$$\begin{aligned} & (\mathrm{XX}^{\mathrm{T}} \otimes \mathrm{I_b})\mathrm{vec}(\mathrm{R}) = (\mathrm{X} \otimes \mathrm{I_b})\mathrm{vec}(\mathrm{Z}) \\ \text{s.t.} \quad & \mathrm{R} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \leq r_0. \end{aligned} \tag{17}$$

Although the left-hand side matrix $(\mathrm{XX}^{\mathrm{T}} \otimes \mathrm{I_b}) \in \mathbb{R}^{bd \times bd}$ is huge, it is a block-diagonal matrix, with all its blocks being the same. Thus we only need to store $\mathrm{XX}^{\mathrm{T}} \in \mathbb{R}^{d \times d}$, instead of explicitly storing $(\mathrm{XX}^{\mathrm{T}} \otimes \mathrm{I_b})$; Similarly, the right-hand vector $(\mathrm{X} \otimes \mathrm{I_b})\mathrm{vec}(\mathrm{Z}) \in \mathbb{R}^{bd}$ can be found out without explicitly computing $(\mathrm{X} \otimes \mathrm{I_b})$. Next, we adopt Alternating Least Square(ALS) methods (Dolgov and Savostyanov 2014) to find a locally optimal solution to Equation 17 by iteratively updating the cores $\mathcal{G}$ of the tensor $\mathcal{R}$.

Alternating Least Square (ALS) methods To simplify the presentation of ALS methods, we denote $\mathrm{Q} = (\mathrm{XX}^{\mathrm{T}} \otimes \mathrm{I_d}), \mathrm{r} = \mathrm{vec}(\mathrm{R})$ and $\mathrm{p} = (\mathrm{X} \otimes \mathrm{I_d})\mathrm{vec}(\mathrm{Z})$. Thus Equation 17 can be rewritten as:

$$\begin{aligned} & \mathrm{Qr} = \mathrm{p} \\ \text{s.t.} \quad & \mathrm{r} \sim \mathcal{R}, \text{ and } r_{TT}(\mathcal{R}) \leq r_0. \end{aligned} \tag{18}$$

Despite appearing as a constrained linear system, solving the cores $\mathcal{G}$ of the tensor $\mathcal{R}$ in Equation 18 is unfortunately a nonlinear optimization problem which can hardly be tackled unless an accurate initial guess is available (Rohwedder and Uschmajew 2013), even if we have known the shape $[n_1, n_2, \ldots, n_t]$ and TT-rank r_0 of $\mathcal{R}$. To make it tractable, we update the cores $\mathcal{G}$ one by one to attain an approximate solution.

We use *index grouping* to reshape tensors into matrices and vectors, *i.e.*, combine two or more indices $i_1, \ldots, i_k$ into a single multi-index $\overline{i_1 \ldots i_k}$. Following (Rohwedder and Uschmajew 2013), we define interface matrices $\mathrm{G}^{\leq k} \in \mathbb{R}^{n_1 n_2 \ldots n_k \times r_0}$ and $\mathrm{G}^{<k} \in \mathbb{R}^{r_0 \times n_{k+1} \ldots n_t}$ as:

$$\begin{aligned} \mathrm{G}^{\leq k}(\overline{i_1 \ldots i_k}, r_0) &= \mathcal{G}_1(i_1) \ldots \mathcal{G}_k(i_k) \\ \mathrm{G}^{>k}(r_0, \overline{i_{k+1} \ldots i_d}) &= \mathcal{G}_{k+1}(i_{k+1}) \ldots \mathcal{G}_t(i_t) \end{aligned} \tag{19}$$

where $i_k \in \{1, \ldots, n_k\}$. We further define a matrix $\mathrm{G}_{\neq k} \in \mathbb{R}^{n_1 n_2 \ldots n_t \times r_0 n_k r_0}$ as:

$$\mathrm{G}_{\neq k} = \mathrm{G}^{<k} \otimes \mathrm{I}_{n_k} \otimes (\mathrm{G}^{>k})^{\mathrm{T}} \tag{20}$$

where I_{n_k} is the n_k-dimensional identity matrix.

To update the k-th core $\mathcal{G}_k$ with others fixed, we stretch all its entries into a vector $\mathrm{g_k} \in \mathbb{R}^{r_0 n_k r_0}$. According to (Dolgov and Savostyanov 2014), we have

$$\mathrm{r} = \mathrm{G}_{\neq k}\mathrm{g_k} \tag{21}$$

By substituting Equation 21 back to Equation 18, the update rule for the k-th core $\mathcal{G}_k$ can be easily found as:

$$\mathrm{g_k} = \left((\mathrm{QG}_{\neq \mathrm{k}})^{\mathrm{T}}(\mathrm{QG}_{\neq \mathrm{k}})\right)^{-1}\left(\mathrm{QG}_{\neq \mathrm{k}}\right)^{\mathrm{T}}\mathrm{p} \tag{22}$$

As shown in Figure 5.3 in (Kressner and Tobler 2011), the convergence of ALS algorithm is almost independent of the order of updating cores, and sequential update of cores can facilitate computational efficiency (Holtz, Rohwedder, and Schneider 2012), therefore, we sequentially update the cores from $\mathcal{G}_1$ to $\mathcal{G}_k$ in our implementation.

5.4. Implementations

To optimize our objective function in Equation 9, we iteratively update the three variable $\mathrm{C}, \bar{\mathrm{R}}$ and R. We initialize

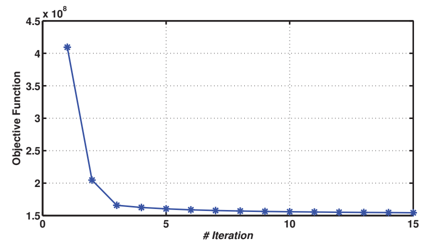


Figure 1: Convergence of our algorithm. The vertical axis represents the objective function value of Equation 9 and the horizontal axis corresponds to the number of iterations. The optimization of R is obtained on the training set of DNN-4096 with the number of cores as 6 and the TT-rank as 4.

$\bar{\mathrm{R}}$ as a random orthogonal matrix to satisfy the orthogonal constraint, consequently, R is initialized by rounding $\bar{\mathrm{R}}$ into TT-format with desired TT-rank r_0 using TT-SVD algorithm (Oseledets 2011). The training data is subtracted with its mean prior to learning. For the hyperparameter β in Equation 9, we experimentally find that a fixed $\beta = 1$ leads to competitive accuracy, and the accuracy is insensitive to the choice of β (we tried from 0.1 to 100). So we simply fix $\beta = 1$ for all experiments in this work.

Since our solution to each sub-problem can always reduce the objective function value in Equation 9, and the objective function value is lower-bounded (not smaller than 0), our algorithm is guaranteed to reach a convergence. In practice, it takes around 10 iterations to converge as shown in Figure 1.

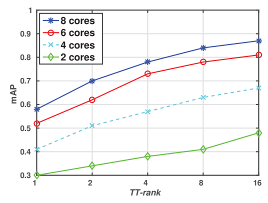
6. Experiments

To evaluate Tensorized Projection (TP), we conduct experiments on three tasks: approximate nearest neighbor (ANN) search, image retrieval, and image classification. For each task, we compare TP with several state-of-the-art methods for high-dimensional visual feature embedding. The compared methods are ITQ, BP, CBE, SP, KBE and FBE. All experiments are conducted using Matlab, while the evaluation of encoding time is implemented in C++ using a single thread. The machine we use is equipped with Intel Xeon CPUs E5-2630 (2.30GHz) and 96 GB memory.

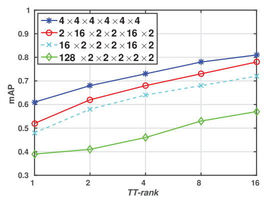
6.1. Approximate Nearest Neighbor Search

6.1.1. Experiments on DNN features Recent research advances have demonstrated the advantage of deep learning features as image representations (Krizhevsky, Sutskever, and Hinton 2012; He et al. 2016). We first conduct experiments on such features. We use the pre-trained AlexNet (Krizhevsky, Sutskever, and Hinton 2012) provided by Caffe (Jia et al. 2014) to extract DNN features for one million images in MIRFLICKR-1M dataset (Huiskes and Lew 2008). We extract 4096-dimensional outputs of the second fc layer as image features. We refer to this dataset as DNN-4096. We randomly pick 1,000 samples as queries. Note that each 4096-dimensional raw feature (real number) requires a storage of 16,384 bytes (131,072 bits).

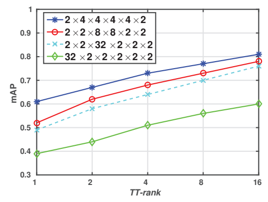
Following the protocol in (Xia et al. 2015), we measure the search quality using mean Average Precision (mAP),



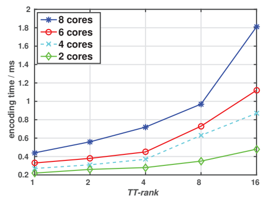
(a) mAP *vs.* TT-rank



(b) mAP *vs.* TT-rank



(c) mAP *vs.* TT-rank



(d) encoding time *vs.* TT-rank

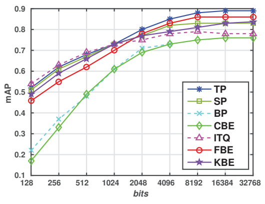
Figure 2: Comparison results on DNN-4096 dataset for the task of encoding 4096-dimensional feature to 1024-bit binary codes. (a) The change of mAP for different number of cores. (b) The change of mAP for different shapes of TP input tensor. (c) The change of mAP for different shapes of TP output tensor. (d) The change of encoding time for different number of cores, the detailed shapes of input/output tensor are shown in Table 1 below.

Cores	Input/Output Shape	TT-1	TT-2	TT-4	TT-8	TT-16
2	64×64	4096	8192	16384	32768	65536
	32×32					
4	$8 \times 8 \times 8 \times 8$	192	640	2304	8704	33792
	$4 \times 8 \times 8 \times 4$					
6	$4 \times 4 \times 4 \times 4 \times 4 \times 4$	80	288	1088	4224	16640
	$2 \times 4 \times 4 \times 4 \times 4 \times 2$					
8	$2 \times 2 \times 4 \times 4 \times 4 \times 4 \times 2 \times 2$	64	240	928	3072	12288
	$2 \times 2 \times 2 \times 4 \times 4 \times 2 \times 2 \times 2$					

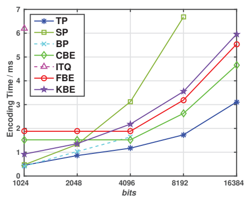
Table 1: The number of tunable parameters when encoding 4096-dimensional feature to 1024-bit binary codes using different cores and TT-rank. As comparison, the uncompressed projection matrix contains $4,194,304$ elements.

i.e., the mean area under the precision-recall curve. Given a query, we perform Hamming ranking, *i.e.*, samples in the dataset are ranked according to their Hamming distances to the query, based on their binary codes. The average distance to the 50th nearest neighbors of images in the dataset is used to define the true positive samples of a query.

Self-comparisons We first conduct ablation studies to investigate the effectiveness of various number of cores, input/output tensor shapes and TT-ranks. Consider the task of projecting 4096-dim feature to 1024-bit binary codes, Figure 2(a) shows the mAP of different number of cores, including $2, 4, 6, 8$ cores. The corresponding shapes of input and output tensor are listed in Table 1; For the case of 6 cores with the fixed shape of output tensor, Figure 2(b) shows the results for various shapes of input tensor; For the case of 6 cores with the fixed shape of input tensor, Figure 2(c) shows the results for various shapes of output tensor; Figure 2(d)



(a) mAP *vs.* bits



(b) encoding time *vs.* bits

Figure 3: Comparison results on DNN-4096 dataset. (a) The change of mAP w.r.t bits. (b) The change of encoding time w.r.t bits. For clarify, we only show the encoding time below 7ms. BP is unavailable for the longer codes $(b > d)$.

bits	2048	4096	8192	16384	32768
BP	6144	8192	–	–	–
CBE	8192	8192	16384	32768	65536
ITQ	8.3×10^6	1.6×10^7	3.2×10^7	6.4×10^7	1.2×10^8
SP	8.3×10^5	1.6×10^6	3.2×10^6	6.4×10^6	1.2×10^7
FBE	12288	12288	24576	49152	98304
KBE	88	96	104	112	120
TP (ours)	1120	1152	1216	1472	1728
Shape of TP Output	2×4×4×4×4×4	4×4×4×4×4×4	8×4×4×4×4×4	8×8×4×4×4×4	8×8×8×4×4×4

Table 2: The number of tunable parameters when encoding 4096-dimensional feature to binary codes. The last row contains the shapes of output tensor for various bits.

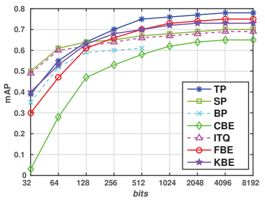
shows the encoding time for different number of cores, the detailed shapes of input/output tensor are shown in Table 1.

We have the following observations: (1) Figure 2(a) shows that more cores lead to larger model capability and higher accuracy, but the gains are diminishing; (2) Figure 2(b) and 2(c) illustrate that well-proportioned shapes of input/output tensor perform better than ill-proportioned ones; (3) Figure 2(d) suggests that more cores result in worse encoding efficiency, and the encoding time appears to increase quadratically with respect to TT-rank, which agrees with the symbolic computational complexity in Section 3.3.

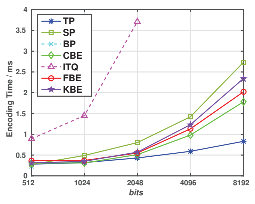
Besides, we notice that TT-1 with 2 cores and TT-8 with 6 cores in Table 1 have similar numbers of tunable parameters 4096 and 4224, while the mAP of TT-1 is almost 50% lower than that of TT-8. This phenomenon suggests that, with the same budget for tunable parameters, one should seek for more cores and higher rank to increase the model capacity, while the input/output shapes can be small. Based on these observations, we fix the TT-rank to 4 to balance the accuracy and efficiency in all following experiments. Since 8 cores does not perform much better then 6 cores, we reshape a 4096-dimensional input vector to the 6-dimensional tensor of the size $4 \times 4 \times 4 \times 4 \times 4 \times 4$.

Comparison with State-of-the-art In Figure 3(a), we show how mAP changes *w.r.t* code length b. The proposed TP achieves competitive mAP on the short codes, and outperforms other state-of-the-art methods on long codes, *i.e.*, bit lengths comparable to or longer than feature dimension.

As shown in Figure 3(b), the encoding time for computing binary codes does not increase linearly to b. The proposed TP takes the least encoding time among all state-of-the-art methods. The results also validate that longer codes in general achieve better performance than shorter ones (Sánchez and Perronnin 2011). We also compare the number of parameters of our proposed algorithm and those of the baselines in Table 2. Although KBE requires even fewer parameters than ours when producing binary codes of the same length, its mAP and encoding time are inferior to the proposed TP, as shown in Figure 3 and 4. The last row in Table 2 elaborates the shapes of output tensor for various bits.



(a) mAP *vs.* bits



(b) encoding time *vs.* bits

Figure 4: Comparison results on GIST-960 dataset. (a) The change of mAP w.r.t bits. (b) The change of encoding time w.r.t bits. For clarify, we only show the encoding time below 4ms.

bits	32	128	512	2048	8192
Shape of TP Output	2×2×2×2×2	2×2×4×4×2	4×4×4×4×2	4×4×8×4×4	4×8×8×8×4

Table 3: The shapes of output tensor for GIST feature.

		mAP	Storage (relative)	Encoding time (ms)
raw deep feature (4096-d)		49.5%	1	-
1024 bits	BP	44.5%	1/128	**0.41**
	CBE	44.3%	1/128	1.52
	SP	44.9%	1/128	0.47
	FBE	45.6%	1/128	1.88
	KBE	45.0%	1/128	0.92
	TP (Ours)	**45.9%**	1/128	0.45
4096 bits	BP	46.4%	1/32	1.64
	CBE	46.6%	1/32	1.52
	SP	47.1%	1/32	3.12
	FBE	47.2%	1/32	1.88
	KBE	47.2%	1/32	2.18
	TP (Ours)	**47.6%**	1/32	**1.17**
8192 bits	CBE	47.8%	1/16	2.63
	SP	48.5%	1/16	6.68
	FBE	49.3%	1/16	3.19
	KBE	48.8%	1/16	3.55
	TP (Ours)	**49.3%**	1/16	**1.73**

Table 4: Image retrieval performance on Holidays+1M.

6.1.2. Experiments on traditional features To validate the effectiveness of TP, besides deep features, we also evaluate our method on a dataset of tranditional features, *i.e.*, GIST-960 (Jegou, Douze, and Schmid 2008) dataset, which contains one million 960-dimensional GIST features (Oliva and Torralba 2001) and $10,000$ queries. We reshape a 960-dimensional GIST feature to $3 \times 4 \times 4 \times 4 \times 5$, the shapes of output tensor for GIST feature are listed in Table 3.

As shown in Figure 4, the proposed TP still outperforms state-of-the-art methods on long codes in terms of both accuracy and encoding time. The experimental results on GIST features demonstrate again the potential of the TT-format to high-dimensional visual features.

6.2. Image Retrieval

We evaluate the performance of binary embedding for the image retrieval task on the Holidays + MIRFlickr-1M dataset (Jegou, Douze, and Schmid 2008). This dataset contains 1,419 images in 500 different scenes, with extra one million MIRFlickr-1M images as distracters. Another 500 query images are provided along with their ground truth neighbors under the same scene category. In light of the good performances with DNN features as image descriptors (Bhattacharjee et al. 2016; Meng et al. 2016; Tu et al. 2016; Yu, Meng, and Yuan 2017; Yu, Wang, and Yuan 2017), we represent each image by the 4096-dimensional response of the second fc layer of AlexNet. In this experiment, we use the same input/output shapes as elaborated in Section 6.1.

Following previous practices (Jegou et al. 2012), we treat image retrieval as an ANN search problem of the encoded features, while the ground truth neighbors are defined by scene labels. Given a query image, we perform Hamming ranking and evaluate mAP using the semantic ground truth.

Table 4 shows the results on the Holidays+1M dataset. As a baseline of using the raw features, the mAP of 4096-dim deep features is 49.5%. Our method can lead to the best mAP among all compared methods. In case of 8192 bits, our method has almost no degradation (49.4% mAP) compared with the use of the raw deep learning features. However, we do not observe better performance when using 16384 bits.

Table 4 also lists the storage required for the database. Without encoding, each 4096-dim raw feature requires 16384 bytes (131,072 bits), so the database of raw features requires around 16 GB memory. The binary encoding methods can significantly reduce this cost.

6.3. Image Classification

We further evaluate the binary codes as compact features for image classification on CIFAR-10 dataset (Krizhevsky 2009). We extract the 4096-dimensional responses of second fc layer in AlexNet as image features. We first fine-tune the pre-trained model provided by Caffe (Jia et al. 2014) on the training set of CIFAR-10, then we use the fine-tuned model to generate features for both training images and testing images. We then learn the hashing parameters on the features of CIFAR-10 training set. In this experiment, we use the same input/output shapes as elaborated in Section 6.1.

We use one-vs-rest linear SVM as classifier as we observe that one-vs-rest linear SVM achieves 82.6% classification accuracy on the raw feature, which is higher than that from the softmax layer (78.9%). We perform the comparisons with $1024 \sim 16384$ bits. We do not see significant performance gains when further increasing the bit length.

Table 5 shows the comparison result. With 4096 bits or more, the proposed TP performs better than other state-of-the-art methods with the same number of bits. Note that even when the number of bits is more than the input dimension 4096, these representations are still more compact than the original features, *e.g.*, 16,384 bits require only 1/8 storage cost of a raw 4096-dimensional feature of real numbers.

		Classification accuracy	Storage (relative)	Encoding time (ms)
raw deep feature (4096-d)		82.6%	1	-
1024 bits	BP	76.6%	1/128	**0.41**
	CBE	76.3%	1/128	0.85
	SP	77.8%	1/128	0.47
	FBE	**79.7%**	1/128	1.88
	KBE	78.3%	1/128	0.92
	TP(Ours)	79.3%	1/128	0.45
4096 bits	BP	77.5%	1/32	1.64
	CBE	77.4%	1/32	1.52
	SP	78.6%	1/32	3.12
	FBE	80.7%	1/32	1.88
	KBE	79.3%	1/32	2.18
	TP(Ours)	**81.2%**	1/32	**1.17**
8192 bits	CBE	78.1%	1/16	2.63
	SP	79.5%	1/16	6.68
	FBE	81.6%	1/16	3.19
	KBE	80.5%	1/16	3.55
	TP(Ours)	**82.0%**	1/16	**1.73**
16384 bits	CBE	78.6%	1/8	4.66
	SP	80.2%	1/8	13.4
	FBE	82.4%	1/8	5.54
	KBE	81.0%	1/8	5.96
	TP(Ours)	**82.6%**	1/8	**3.11**

Table 5: Classification accuracy on CIFAR-10 dataset.

6.4. Discussion

In the above experiments, we observe that the binary code length b required to achieve graceful degradation (compared with no encoding) is usually around $b \sim \mathcal{O}(d)$ for high-dimensional features, which justifies the rationale of using long binary codes for high-dimensional data. Short binary codes have considerable degradation in accuracy, thus in practice, it is desirable to have a feasible and accurate solution to high-dimensional binary embedding.

7. Conclusion

To compress high-dimensional visual features, we have proposed a novel binary embedding approach Tensorized Projection (TP). Compared with state-of-the-art methods, our proposed TP uses the fewest parameters of the projection matrix to achieve the fastest binary encoding of high-dimensional features. Moreover, due to the decrease of tunable parameters, the ultimate projection matrix would have restricted freedom, which could lead to good generalization performance. We evaluate our TP on three tasks, including approximate nearest neighbor search, image retrieval, and image classification. Experimental results validate the efficiency and accuracy of TP.

Acknowledgements

This work is supported in part by Singapore Ministry of Education Academic Research Fund Tier 2 MOE2015-T2-2-114 and was carried out at the Rapid-Rich Object Search (ROSE) Lab in the Nanyang Technological University, Singapore. The ROSE Lab is supported by the National Research Foundation, Singapore, under its Interactive Digital Media (IDM) Strategic Research Programme. We gratefully acknowledge the support of NVAITC (NVIDIA AI Technology Centre) for their donation of a Tesla K80 and M60 GPU used for our research at the ROSE Lab.

The authors appreciate Sergey Dolgov, Dmitry Savostyanov and Ivan Oseledets for valuable discussions!

References

Bhattacharjee, S. D.; Yuan, J.; Hong, W.; and Ruan, X. 2016. Query adaptive instance search using object sketches. In *Proceedings of the 2016 ACM on Multimedia Conference*, 1306–1315. ACM.

Courant, R. 1943. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society* 49(1):1–23.

Dolgov, S. V., and Savostyanov, D. V. 2014. Alternating minimal energy methods for linear systems in higher dimensions. *SIAM Journal on Scientific Computing* 36(5).

Gong, Y., and Lazebnik, S. 2011. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 817–824. IEEE.

Gong, Y.; Kumar, S.; Rowley, H. A.; and Lazebnik, S. 2013. Learning binary codes for high-dimensional data using bilinear projections. In *CVPR*, 484–491.

Gower, J. C., and Dijksterhuis, G. B. 2004. *Procrustes problems*. Number 30. Oxford University Press.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR*, 770–778.

Holtz, S.; Rohwedder, T.; and Schneider, R. 2012. The alternating linear scheme for tensor optimization in the tensor train format. *SIAM Journal on Scientific Computing* 34(2):A683–A713.

Hong, W.; Yuan, J.; and Das Bhattacharjee, S. 2017. Fried binary embedding for high-dimensional visual features. In *CVPR*.

Huiskes, M. J., and Lew, M. S. 2008. The mir flickr retrieval evaluation. In *Proceedings of the 1st ACM international conference on Multimedia information retrieval*.

Jegou, H.; Perronnin, F.; Douze, M.; Sánchez, J.; Perez, P.; and Schmid, C. 2012. Aggregating local image descriptors into compact codes. *TPAMI*.

Jegou, H.; Douze, M.; and Schmid, C. 2008. Hamming embedding and weak geometric consistency for large scale image search. In *ECCV*, 304–317. Springer.

Jia, Y.; Shelhamer, E.; Donahue, J.; Karayev, S.; Long, J.; Girshick, R.; Guadarrama, S.; and Darrell, T. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, 675–678. ACM.

Kressner, D., and Tobler, C. 2011. Preconditioned low-rank methods for high-dimensional elliptic pde eigenvalue problems. *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.*

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *NIPS*, 1097–1105.

Krizhevsky, A. 2009. Learning multiple layers of features from tiny images.

Le, Q.; Sarlós, T.; and Smola, A. 2013. Fastfood: approximating kernel expansions in loglinear time. In *ICML*.

Lowe, D. G. 2004. Distinctive image features from scale-invariant keypoints. *IJCV*.

Meng, J.; Wang, H.; Yuan, J.; and Tan, Y.-P. 2016. From keyframes to key objects: Video summarization by representative object proposal selection. In *CVPR*, 1039–1048.

Novikov, A.; Rodomanov, A.; Osokin, A.; and Vetrov, D. 2014. Putting mrfs on a tensor train. In *ICML*, 811–819.

Novikov, A.; Podoprikhin, D.; Osokin, A.; and Vetrov, D. P. 2015. Tensorizing neural networks. In *NIPS*, 442–450.

Oliva, A., and Torralba, A. 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *IJCV*.

Oseledets, I. V. 2011. Tensor-train decomposition. *SIAM Journal on Scientific Computing*.

Rohwedder, T., and Uschmajew, A. 2013. On local convergence of alternating schemes for optimization of convex problems in the tensor train format. *SIAM Journal on Numerical Analysis*.

Sánchez, J., and Perronnin, F. 2011. High-dimensional signature compression for large-scale image classification. In *CVPR*, 1665–1672. IEEE.

Shakhnarovich, G.; Darrell, T.; and Indyk, P. 2006. Nearest-neighbor methods in learning and vision: Theory and practice.

Stoudenmire, E., and Schwab, D. J. 2016. Supervised learning with tensor networks. In *NIPS*, 4799–4807.

Tu, Z.; Cao, J.; Li, Y.; and Li, B. 2016. Msr-cnn: applying motion salient region based descriptors for action recognition. In *ICPR*, 3524–3529.

Wang, Y.; Yang, J.; Yin, W.; and Zhang, Y. 2008. A new alternating minimization algorithm for total variation image reconstruction. *SIAM Journal on Imaging Sciences* 1(3):248–272.

Wang, Z.; Duan, L.-Y.; Yuan, J.; Huang, T.; and Gao, W. 2016. To project more or to quantize more: minimizing reconstruction bias for learning compact binary codes. In *IJCAI*, 2181–2188. AAAI Press.

Wang, J.; Zhang, T.; Sebe, N.; Shen, H. T.; et al. 2017. A survey on learning to hash. *PAMI*.

Xia, Y.; He, K.; Kohli, P.; and Sun, J. 2015. Sparse projections for high-dimensional binary codes. In *CVPR*, 3332–3339.

Yang, Z.; Moczulski, M.; Denil, M.; de Freitas, N.; Smola, A.; Song, L.; and Wang, Z. 2015. Deep fried convnets. In *ICCV*, 1476–1483.

Yu, F.; Kumar, S.; Gong, Y.; and Chang, S.-F. 2014. Circulant binary embedding. In *ICML*, 946–954.

Yu, T.; Meng, J.; and Yuan, J. 2017. Is my object in this video? reconstruction-based object search in videos. In *IJCAI*.

Yu, T.; Wang, Z.; and Yuan, J. 2017. Compressive quantization for fast object instance search in videos. In *ICCV*.

Zhang, X.; Yu, F. X.; Guo, R.; Kumar, S.; Wang, S.; and Chang, S.-F. 2015. Fast orthogonal projection based on kronecker product. In *ICCV*.