Fast Action Retrieval from Videos via Feature Disaggregation

Jie Qin[1]
qinjiebuaa@gmail.com
Li Liu[2]
li2.liu@northumbria.ac.uk
Mengyang Yu[2]
m.y.yu@ieee.org
Yunhong Wang[1]
yhwang@buaa.edu.cn
Ling Shao[2]
ling.shao@ieee.org

[1] Beijing Key Laboratory of Digital Media,
School of Computer Science and Engineering,
Beihang University, China

[2] Computer Vision and Artificial Intelligence Group,
Department of Computer Science and Digital Technologies,
Northumbria University, UK

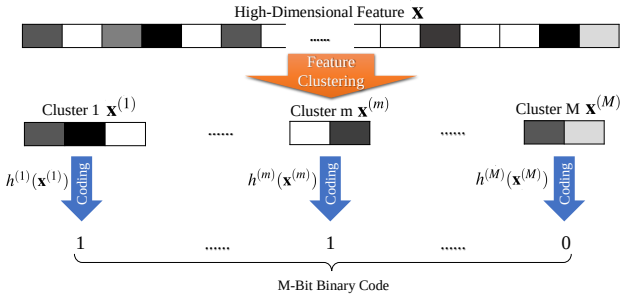


Figure 1: The overall framework of the proposed Disaggregation Hashing.

Motivation. Learning based hashing methods have been actively studied recently. A majority of the approaches [2, 3, 4, 5] have been specifically developed for image retrieval. However, due to the extra temporal information, videos are usually represented by much higher dimensional features compared with images, causing high computational complexity for conventional hashing schemes. Learning hash functions can become quite time-consuming and even intractable when dealing with very high-dimensional video data. Besides, a lot of hashing methods are based on linear projections (*e.g.*, random projection [4]). In terms of mapping the data from the very high-dimensional space to a reduced one, the memory requirements for storing the projection matrix and performing the mapping operation impose heavy burdens.

Contributions. A novel hashing scheme, namely Disaggregation Hashing (DH), is proposed for high-dimensional video data. Our main idea is to disaggregate the original high-dimensional features into several groups of low-dimensional ones, based on which independent hash functions are learned. Figure 1 shows the overall framework of our method.

Given a set of data points $\mathbf{x}$, our goal is to find a binary embedding function $H(\mathbf{x}) = \{-1,+1\}^M$, where M is the length of the code. Disaggregation Hashing learns different hash functions on different subspaces obtained by feature disaggregation. Hence, each bit of the entire code is learned independently, which enhances the scalability on high-dimensional video data and allows fast parallel computation as well. Furthermore, the proposed method only needs to store a D-dimensional projection vector and computing binary codes is of $O(D)$ complexity.

In terms of feature disaggregation or clustering, we show that by incorporating a special structure constraint into the projection matrix $\mathbf{W}$ of PCA, $\mathbf{W}$ can be regarded as the cluster indicator and the solution to feature clustering is identical to the one to PCA. Moreover, as mentioned in [1], k-means clustering has a similar formulation with PCA if we treat $\mathbf{W}$ as the cluster indicator. Therefore, feature clustering can be effectively addressed using k-means clustering along the feature dimensions.

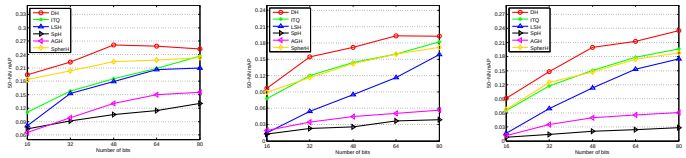
After feature disaggregation, the original feature space $\mathbb{R}^D$ is split into M subspaces $\mathbb{R}^{d_m}$, where $m = 1,...,M$, and $\sum_{m=1}^{M} d_m = D$. The overall hash function $H(\mathbf{x})$ consists of M independent functions $h^{(m)}(\mathbf{x}^{(m)})$, where $\mathbf{x}^{(m)}$ corresponds to the data points from subspace $\mathbb{R}^{d_m}$. Each $h^{(m)}$ maps $\mathbf{x}^{(m)}$ into one binary code $c^{(m)} \in \{-1,+1\}$ and the final code is a concatenation of M-bit codes $(c^{(1)},...,c^{(M)})$. Specifically, we define the above independent hash function as:

$$h^{(m)}(\mathbf{x}^{(m)}) = sgn(\mathbf{w}^{(m)}\mathbf{x}^{(m)} + b^{(m)}) \quad (1)$$

where '$sgn()$' returns '+1' if the argument is positive and '-1' otherwise, and $\mathbf{w}^{(m)} \in \mathbb{R}^{1\times d_m}$ is the projection vector. We aim at learning similarity-preserving codes that hold the minimal average Hamming distance between similar data points. Let $c_p^{(m)}$ denote one bit code for a data point $\mathbf{x}_p$, *i.e.*, $c_p^{(m)} = h^{(m)}(\mathbf{x}_p^{(m)})$, the objective function is defined as:

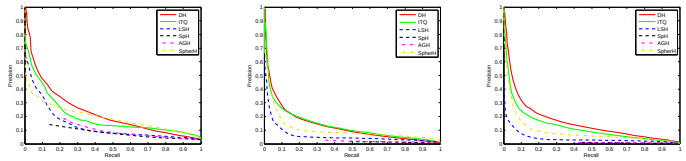
$$\min \sum_{p,q} ||c_p^{(m)} c_q^{(m)} - l_{pq}||^2 \quad (2)$$

where $l_{pq} = +1$ if $\mathbf{x}_p$ and $\mathbf{x}_q$ are similar points, and -1 otherwise. We find the solution to Eq. (2) by addressing a greedy optimization problem and obtain the approximate optimal parameters in Eq. (1).



(a) (b) (c)

Figure 2: Comparison with state-of-the-art methods in terms of mean Average Precision on (a) Hollywood2, (b) HMDB51 and (c) UCF101.



(a) (b) (c)

Figure 3: Comparison with state-of-the-art methods in terms of precision-recall curves@80 bits on (a) Hollywood2, (b) HMDB51 and (c) UCF101.

Results. The experiments are performed for action retrieval on three realistic action datasets, *i.e.*, Hollywood2, HMDB51 and UCF101, and follow the protocols widely used in [2, 3]. We compare our method with five state-of-the-art hashing algorithms. As shown in Figure 2 and 3, our hashing scheme consistently outperforms other methods on different datasets in terms of mean Average Precision and precision-recall curves. Table 1 illustrates the training and coding time of different methods on UCF101. Thanks to feature disaggregation, our method requires much lower time for learning hash functions than most of other methods and computing binary codes for test data points is also very fast.

[1] C. Ding and X. He. K-means clustering via principal component analysis. In *Proc. ICML*, 2004.

[2] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proc. CVPR*, 2011.

[3] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *Proc. CVPR*, 2012.

[4] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proc. STOC*, 1998.

[5] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proc. NIPS*, 2009.

Bits	Runtime (ms) of Training (Coding)					
	LSH	SpH	AGH	ITQ	SpherH	**DH**
16	(87.0)	4451.8(119.4)	819.0(44.9)	5570.0(82.0)	4774.7(245.0)	1703.5(2.8)
32	(103.9)	4061.8(213.7)	843.9(43.1)	6093.3(114.6)	6756.2(244.3)	1538.7(2.7)
48	(130.6)	4513.7(420.2)	913.3(47.0)	6948.6(140.2)	7809.5(296.7)	1820.2(2.3)
64	(154.3)	4774.9(600.0)	857.7(45.3)	7936.2(159.6)	10139.0(316.9)	2125.3(1.8)
80	(167.9)	5202.6(875.4)	886.4(46.6)	7960.9(190.6)	13720.2(339.9)	2467.7(1.7)

Table 1: Comparison of averaged runtime using different bits on UCF101.