PRODUCT QUANTIZATION FOR VIDEO COPY DETECTION

[1]*Wei-Ta Chu* (朱威達), [1]*Tzu-Min Yang* (楊子旻), [2]*Jen-Yu Yu* (游人諭)

[1]Dept. of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan
E-mail: wtchu@cs.ccu.edu.tw, faad10@hotmail.com

[2]Information and Communication Research Labs,
Industrial Technology Research Institute, Hsinchu, Taiwan
E-mail: KevinYu@itri.org.tw

ABSTRACT

A novel indexing approach called product quantization is used to index high-dimensional feature vectors, and with this representation, video frames or video segments are effectively matched so that video copy detection is achieved. To accurately identify locations of video copy segments, the temporal Hough transform is utilized. To evaluate the proposed system, we first verify the superiority of product quantization over conventional vector quantization based on synthesized data, and then compare our work with other video copy detection methods based on the TRECVID 2010 dataset. Experimental results show our promising performance.

Keywords *Product Quantization; Video Copy Detection; Bag of Word Model*

1. INTRODUCTION

Calculating Euclidean distance between high-dimensional vectors is commonly used in many applications, such as object recognition, human detection, and content-based image retrieval. Calculating Euclidean distances seems fast. However, when numerous data are processed, calculating Euclidean distances for pairs of data may take a lot of time. Another issue is the curse of dimensionality. As dimension of vectors increases, using simple Euclidean distance is not sufficient to discriminate vectors of different characteristics. In this work, we attempt to adopt the idea of "product quantization" (PQ) [4] to reduce the time needed to calculate Euclidean distances between high-dimensional vectors.

Because PQ is highly correlated to conventional vector quantization (VQ), we first introduce VQ and then the kernel idea of PQ. For VQ, the vector space is first divided into a fixed number of subspaces, and each subspace is represented by the centroid of vectors in this subspace. The VQ process maps an original vector into one of these centroids. As compared to the original vector representation, the transformed vectors have fewer variations, and thus less memory is needed for data storage.

Given two high-dimensional vectors, we can firstly quantize these vectors to their corresponding centroids, and then the Euclidean distance between two high-dimensional vectors can be estimated. However, the difference between the estimated distance and the real one depends on quantization errors, which are highly related to the number of subspaces. In order to estimate accurate distance, finer subspaces are needed. Figure 1 illustrates this concept. The Euclidean distance between $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ is more accurately estimated by $\boldsymbol{c}_1'$ and $\boldsymbol{c}_2'$ than that by $\boldsymbol{c}_1$ and $\boldsymbol{c}_2$, where the former are centroids of the smaller six subspaces, and the latter are centroids of the larger three subspaces. However, dividing a high-dimensional vector space into a large number of subspaces is a troublesome issue. Because quantization amounts to assigning a vector to a centroid, we need to calculate Euclidean distance between vectors and centorids, which is prohibitive if the number of centroids is large. Moreover, memory used to store centroids composed of numerous floating-point values is often not affordable.

To tackle with the aforementioned issue, the idea of product quantization [4] was proposed. For PQ, a high-dimensional vector is firstly segmented into several low-dimensional subvectors, and the vector space constructed by corresponding subvectors is decomposed into low-dimensional subspaces. Finally, Cartesian product is used to combine these subspaces, and thus a large number of subspaces can be obtained to represent the original vector space. Complexity of quantization is significantly reduced. It also reduces the influence of the curse of dimensionality because it separately quantizes lower-dimentional subvectors into corresponding subspaces.

In this paper, we investigate how to utilize product quantization in video copy detection. The primary contributions of this paper are:

- We compare PQ with conventional VQ by experiments, and demonstrate the effectiveness of PQ.
- We propose a framework based on PQ to find video copies in a large video database.
- We devise a mechanism to combine different features and achieve the best performance.

The remainder of this paper is organized as follows. Related works are surveyed in Section 2. In Section 3, we introduce and analyze product quantization by experiments. In Section 4, we use product quantization for video copy detection. Experimental results are presented in Section 5. Finally, we conclude this paper in Section 6.

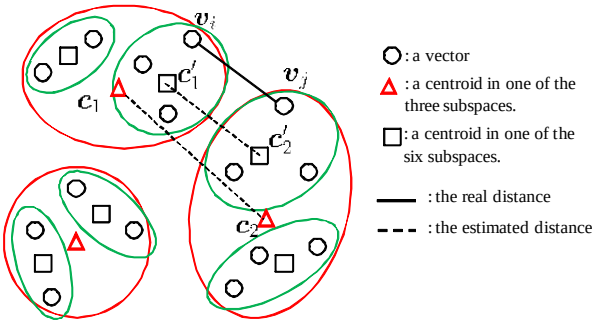

Figure 1. Difference between the estimated distance and the real one under different clustering settings.

2. RELATED WORK

Video copy detection refers to determine whether some videos in a database contain some content similar to the query video, while the targeted video differs from the query video by some video editing operations or visual transformation. Video copy detection is usually used for infringement detection and multimedia retrieval. With the development of Muscle benchmark [5] in year 2007, researchers have a common platform to compare their research results on video copy detection, and therefore it inspires many studies. Law-To et al. [6] gives a comparative study on video copy detection proposed in early years (~2007). Chiu et al. [7] transform video copy detection as a partial matching problem in a probabilistic model. They are devoted to develop a framework robust against spatial and temporal variations, and report relatively fewer experimental results. Yeh and Cheng [8] view video copy detection as a sequence matching problem. As large amounts of sequence matching should be performed, they propose a two-level filtration approach to accelerate the matching process. Wu et al. [3] propose the idea of representing videos by motion trajectories. The bag of word model is used to characterize basic trajectory elements. Finally, the watershed algorithm is used to find partial matching between the query video and videos in the database. Douze et al. [9] match individual frames and then verify their spatio-temporal consistency. Local feature indexing method is proposed to make video copy detection robust to video transformations and efficient in terms of memory usage and computation time. More recently, Douze et al. [10] further propose a compact yet discriminative video representation based on product quantization. Three levels of quantization are used to index feature descriptors. After measuring similarity between video frames, video copy segments are determined by a temporal alignment process.

Inspired by [10], we utilize product quantization to video copy detection. The difference between our work and [10] are threefold. First, we simplify the indexing structure proposed in [10] in order to speed up the detection process. Second, the effectiveness of product quantization is verified by experiments. Third, in addition to local features, we further consider spatial and color information to describe video frames, and devise a mechanism to select appropriate feature for video copy detection.

3. PRODUCT QUANTIZATION

In many multimedia applications, numerous feature vectors are extracted, and the feature dimension is often high. To efficiently represent these feature vectors, a usual manner is quantizing these vectors by a vector quantizer. However, constructing a vector quantizer takes much time, and we need huge memory space to store this quantizer if number of quantization intervals is large. Curse of dimensionality also influences performance of vector quantization. For example, a 960-dimensional GIST descriptor is usually used in scene recognition [1], object recognition, or image copy detection [2]. In [2], a codebook consisting of 20000 entries is constructed by clustering over 15 million GIST descriptors. Such kind of process has tremendous time cost, but unfortunately many researches demonstrate that satisfactory performance can be obtained only with large enough number of quantization intervals.

3.1 *Overview of Product Quantization*

Product quantization (PQ) [4] can efficiently address the aforementioned problems. Unlike VQ using the whole vector to learn a quantizer, PQ splits the original vector into many subvectors and quantizes each subvector by a corresponding quantizer. The set of quantizers that quantizes subvectors are called subquantizers, are combined together to form a product quantizer. Assume that a D-dimensional vector $\boldsymbol{x} = (x_1, x_2, ..., x_D)$ is equally split into p non-overlapping subvectors. The vector $\boldsymbol{x}$ with product quantization is then expressed as:

$$\overbrace{x_1, ..., x_m}^{sv_1(\boldsymbol{x})}, ..., \overbrace{x_{D-m+1}, ..., x_D}^{sv_p(\boldsymbol{x})} \xrightarrow{\text{Product Quantization}} sq_1(sv_1(\boldsymbol{x})), ..., sq_p(sv_p(\boldsymbol{x})), \quad (1)$$

where $m = D/p$ is the dimension of each subvector, $sv_j(\boldsymbol{x})$ is the jth subvector split from the vector $\boldsymbol{x}$, and sq_j is a subquantizer associated with the jth subvector.

To construct a product quantizer, we first collect a large number of D -dimensional feature vectors $\mathcal{X} = \{\boldsymbol{x}_i\}$ and then split each vector $\boldsymbol{x}_i$ into p subvectors ($\{sv_j(\boldsymbol{x}_i)\}$, $j = 1, 2, ..., p$). For the set of subvectors coming from the first range, i.e. $\mathcal{X}_1 = \{sv_j(\boldsymbol{x}_i)|j = 1, \forall i\}$, the k-means algorithm is applied to cluster $\mathcal{X}_1$ to construct the codebook C_1 for the first subquantizer sq_1. The same process is applied to other sets of subvectors separately, and codebooks $C_2, C_3, ..., C_p$ are obtained for $sq_2, sq_3, ..., sq_p$. The Cartesian product of "small" codebooks $C_1, C_2, C_3, ..., C_p$ for subvectors forms a "big" codebook C for the original vectors. The codebook C for product quantization can be expressed as:

$$C = C_1 \times C_2 \times \cdots \times C_p. \quad (2)$$

A codeword in C is formed by concatenating the p centroids from $C_1, C_2, C_3, ..., C_p$. Therefore, the total number of codewords K for product quantization is multiplication of the number of codewords in each subquantizer. This is given by

$$K = \prod_{j=1}^{p} k_j, \quad (3)$$

where k_j is the number of codewords in the j th subquantizer. From now on, in order to clearly distinguish between K and k_j , we use the phrase "number of clusters" to denote K , and the phrase "number of subclusters" to denote k_j.

Product quantization has three advantages:

- Significantly reducing the time required for learning a big quantizer: If there are K quantization levels, the complexity of conventional VQ is $O(KD)$, while complexity of PQ is $O(\max_{1\leq j\leq p} k_j D)$. Product quantization is expected to require much less time than vector quantization.
- Reducing the influence of the curse of dimensionality: If we directly use the high-dimensional vector to learn a quantizer, it's often impeded by the curse of dimensionality. Because product quantization splits a high-dimensional vector into distinct subvectors, dimensions of subvectors are reduced significantly. Therefore, the influence of curse of dimensionality greatly reduces.
- Reducing the memory space required for storing the codebook: If there are K quantization levels, memory space required to store the codebook is at most $\max_{1\leq j\leq p} k_j D$ floating points. The codebook of the same size for vector quantization needs to store KD floating points.

3.2 *Analysis of Product Quantization*

Product quantization can be analyzed from four aspects:

The way to split vectors: The way to split the original vector $\boldsymbol{x}$ into subvectors is not greatly restricted. Generally, the original vector is split such that any two subvectors are independent. On the other hand, consecutive components in the same subvector are usually correlated, and they are better quantized using the same subquantizer.

Subvector's dimension: Dimensions of subvectors are chosen to avoid the curse of dimensionality. In order to reduce influence of the curse of dimensionality, we need as much as possible to reduce subvectors' dimensions. For the ease of processing, it's better to have each subvector with the same dimension.

The number of subclusters in subquantizers: To generate a codebook for each subquantizer, it is not necessary that the number of subclusters for different subquantizers should be the same. The number of subclusters is determined by the importance of the subvector to the original vector, i.e. more important, more subclusters. From now on, we assume that each subvector has equal importance to the original vector. Therefore, we set the number of subclusters for different subquantizers as the same value k. Consequently, the total number of centroids for product quantization can be expressed as: (see also eqn. (3))

$$K = k^p. \quad (4)$$

The number of subquantizers: The number of subquantiers would make a significant impact on performance of product quantization. Assuming an extreme case in which $p = D$, each component of the original vector is quantized, and PQ would become scalar quantization which may completely lose characteristics of the original vector. On the contrary, in the case of $p = 1$, PQ is not different from VQ, and it has no improvement on performance.

3.3 *Performance Verification*

We verify the superiority of PQ over VQ in this section. We evaluate how different parameter settings affect performance. The se parameters are: the dimension of vectors (#dim), the number of vectors (#vector), and the number of clusters for PQ or VQ (K). The time for generating codebooks and the corresponding quantization errors are measured. In particular, quantization errors are average of the mean squared errors calculated between a vector and its corresponding centroid. All experiments were performed on a PC with 2.40GHZ CPU and 1.97GB RAM. The vectors used in experiments are random unit vectors. In the first experiment, we fix dimension of vectors, and evaluate how number of vectors and number of clusters affect PQ and VQ. In the second experiment, we fix number of clusters and number of vectors, and vary dimension of vectors. We set $p = 4$ and $D = 1000$ in the first experiment, and set $p = 4$, $K = 1296$, and the number of vector as 50000 in the second experiment.

Figure 2 illustrates time consumption for generating a codebook based on different numbers of vectors and different numbers of clusters. We found that PQ takes much less time than VQ to generate a codebook. For instance, when the number of vectors is 60000 and the

number of cluster is 1296, the time required for generating a codebook is about 100 seconds for PQ and about 20000 seconds by VQ. The time spent by VQ is several hundred times longer than that spent by PQ.

Table 1 lists average quantization errors caused by PQ and VQ under the corresponding settings of Figure 1. Quantization errors decrease as the number of clusters increases, and quantization errors generated by PQ and by VQ are similar. We know that as the number of vectors and the number of clusters increase, the time consumption increases as well. Therefore, we conclude that PQ is preferable.

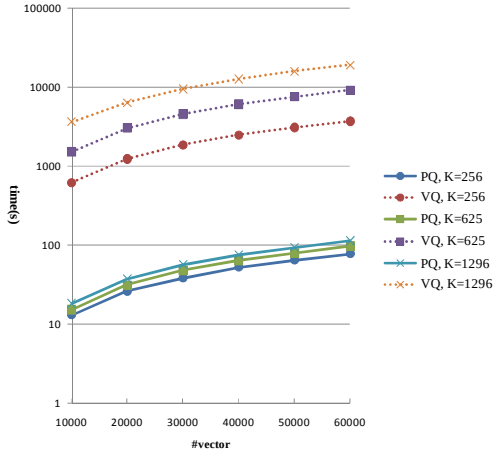

Figure 2. The time required for generating a codebook for PQ and VQ. The dimension of vectors is 1000, the number of vectors range from 10000 to 60000, and the number of clusters are 256, 625, and 1296, respectively.

Table 1. Average quantization errors under the corresponding settings of Figure 1.

#vector / #cluster	10000	20000	30000	40000	50000	60000
PQ, K=256	0.2480	0.2481	0.2483	0.2483	0.2483	0.2483
VQ, K=256	0.2407	0.2440	0.2451	0.2456	0.2460	0.2461
PQ, K=625	0.2477	0.2478	0.2479	0.2480	0.2480	0.2481
VQ, K=625	0.2304	0.2387	0.2414	0.2429	0.2436	0.2442
PQ, K=1296	0.2474	0.2476	0.2476	0.2476	0.2477	0.2478
VQ, K=1296	0.2121	0.2294	0.2350	0.2380	0.2398	0.2409

Figure 3 illustrates the time required for generating codebooks for different dimensions of vectors. When the dimension of vectors is 2000, the time required for generating the codebook is about 200 seconds for PQ and about 30000 seconds for VQ. The difference between the two techniques is by a factor of several hundreds. By jointly reading Table 2 that lists average quantization errors, we again realize that PQ spends much less time than VQ, and in the meanwhile quantization errors caused by PQ and by VQ are similar. From Figure 3, we see that vectors' dimension affects the time required to learn a quantizer. For instance, when the vectors' dimension is 500, the time required to generate the codebook is about 30 seconds for PQ. When the vectors' dimension is 2000, the time required to generate the codebook is about 200 seconds for PQ.

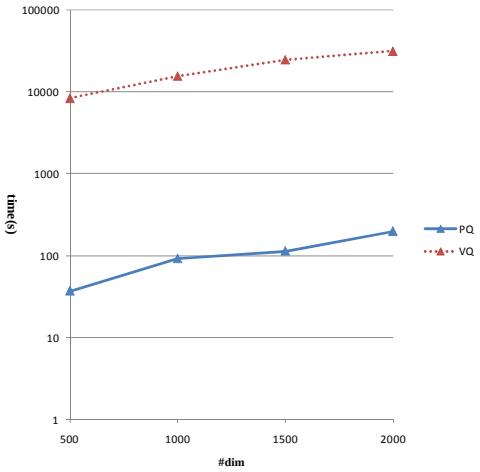

Figure 3. The time required for generating a codebook for PQ and VQ based on vectors of different dimensions. The dimensions of vectors range from 500 to 2000, the number of vectors is 50000, and the number of clusters is 1296.

Table 2. Average quantization errors under the corresponding settings of Figure 2.

#dim / Method	500	1000	1500	2000
PQ	0.2457	0.2477	0.2484	0.2488
VQ	0.2362	0.2398	0.2409	0.2415

4. VIDEO COPY DETECTION

Given a query video, which may be transformed or edited from some database videos, we want to tell whether there is a clip in a database video that has the same or similar content as the query video. In this work, we follow the settings of TRECVID 2010, which define eight video transformations including simulated camcording (T1), picture in picture (T2), insertions of pattern (T3), compression (T4), change of gamma (T5), decrease in quality such as blur and frame dropping (T6), post production such as crop, shift, and flip (T7), and randomly choosing and mixing three transformations (T8).

Figure 4 illustrates the proposed framework for video copy detection. For database videos, we uniformly sample a frame per second as a keyframe, and from each keyframe we extract two types of visual features describing keypoints and color/spatial information. We then use product quantization to index frame features for improving search efficiency. For the query video, we also select keyframes and extract frame features accordingly, but these features are not indexed by product quantization. In the frame search procedure, we estimate distances between keyframes in the query video and that in database videos, under the framework of product quantization. Based on the estimated distances, we determine keyframes from database videos that are similar to each query keyframe and in the meanwhile obtain similarity between matched keyframes. Finally,

we find a sequence of frames in the database videos that match with a sequence of frames in the query video by a temporal Hough transform approach.

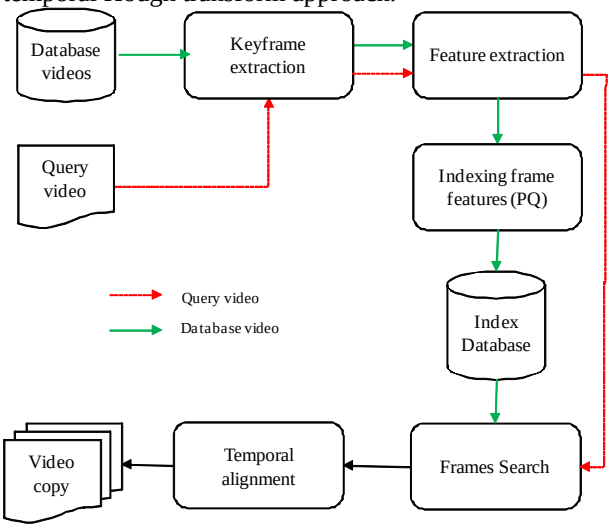

Figure 4. The proposed framework for video copy detection.

4.1 *Frame Features*

Two features are extracted to represent keyframes. First, we use the Hessian detector [12] to detect regions of interest and describe them by the 128-dimensional SIFT descriptors [13]. A keyframe is then represented by aggregating these descriptors based on a locality criterion. The second feature is color layout, which jointly represents color and spatial information.

- Vector of Locally Aggregated Descriptors (VLAD)

Given a set of SIFT descriptors $\{\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_N\}$ from a keyframe, we transform them into a vector of locally aggregated descriptors (VLAD) [14]. The steps of generating a VLAD are described as follows:

1) The k-means algorithm is used to cluster SIFT descriptors extracted from database videos into M groups. This process generates a codebook C with M centroids $\{\boldsymbol{c}_1, \boldsymbol{c}_2, ..., \boldsymbol{c}_M\}$.
2) Each SIFT descriptor $\boldsymbol{y}_i$ is assigned to its nearest centorid in the codebook C. The nearest centroid $\boldsymbol{c}_{i^*}$ is determined by
$$i^* = \arg\min_{1 \le j \le M} ||\boldsymbol{y}_i - \boldsymbol{c}_j||, \quad (5)$$
where $||\cdot||$ is the Euclidean distance.
3) For the SIFT descriptors that are in the same frame and are assigned to the same centroid, we sum the dimension-wise differences between them and their nearest centroids. The difference vector between a SIFT descriptor $\boldsymbol{y}_i$ and its nearest centroid $\boldsymbol{c}_{i^*}$ is calculated as
$$\boldsymbol{y}_i = (y_{i,1}, y_{i,2}, ..., y_{i,128}),$$
$$\boldsymbol{c}_{i^*} = (c_{i^*,1}, c_{i^*,2}, ..., c_{i^*,128}),$$
$$|\boldsymbol{y}_i - \boldsymbol{c}_{i^*}|_d = (y_{i,1} - c_{i^*,1}, y_{i,2} - c_{i^*,2}, ..., y_{i,128} - c_{i^*,128}),$$
where $|\boldsymbol{y}_i - \boldsymbol{c}_{i^*}|_d$ denotes the dimension-wise distance. By summing the dimension-wise distances between a centroid and the SIFT descriptors assigned to it, we obtain a vector
$$\boldsymbol{\mu}^{i^*} = \sum |\boldsymbol{y}_i - \boldsymbol{c}_{i^*}|. \quad (6)$$
Finally, a VLAD for a keyframe is obtained by concatenating $\boldsymbol{\mu}^1, \boldsymbol{\mu}^2, ..., \boldsymbol{\mu}^M$ in order.

The VLAD is L2-normalized into a unit vector. As the dimension of VLAD is M times the dimension of SIFT descriptors, the choice of M is significant to generating VLADs. We set M as 64 in this work according to the advice of [14].

- Color Layout

To extract color layout, we split each frame into $R_W \times R_H$ regions. An RGB color histogram is constructed for each region, and histograms from left-top to right-down regions are concatenated as a long vector, with L2 normalization. Because VLAD does not consider color information, we expect that the second feature provides different information to facilitate video copy detection.

4.2 *Indexing Database Videos*

Two levels of descriptions are used to represent a database video: the segment descriptor and the frame descriptor. Because number of keyframes extracted from database videos is huge, it's more efficient to consider a sequence of keyframes, called segment, in database videos. Each segment is described by a segment descriptor, which is constructed by product quantizing features in this segment and is represented as a tuple of quantization indices. Similarly, each keyframe in a segment is quantized by the same product quantizer, and is represented as a tuple of quantization indices to be a frame descriptor.

- Segment descriptor and segment feature

Suppose we have extracted T frame features $\{\boldsymbol{f}_1, \boldsymbol{f}_2, ..., \boldsymbol{f}_T\}$ from database videos, with the representation of VLAD or color layout. Based on these features, the codebook $C_{PQ} = C_1 \times C_2 \times \cdots \times C_p$ for product quantization is generated by the process described in Section 3.

A fixed-length segment $\mathcal{S}(t_s : t_e)$ in a database video includes keyframes in a short time period, where t_s and t_e denote the timestamps of the starting point and end point of this segment, respectively. In this work, we equally divide a database video into non-overlapping 4-second segments. To calculate the segment descriptor for $\mathcal{S}(t_s : t_e)$, the average frame descriptor is first calculated:
$$\bar{\boldsymbol{f}} = \frac{1}{t_e - t_s} \sum_{t=t_s}^{t_e} \boldsymbol{f}_t. \quad (7)$$
After that, we search the closest centroid from each subquantizer's codebook for a subvector of the average frame descriptor $\bar{\boldsymbol{f}}$ and simply use the indices of these closest centroids to represent the segment. The segment descriptor $\boldsymbol{s}_d(t_s : t_e)$ associated with $\mathcal{S}(t_s : t_e)$ is therefore obtained by

$$\boldsymbol{s}_d(t_s:t_e) = (idx_1, idx_2, ..., idx_p),$$
$$\text{and } idx_i = \langle sq_i(sv_i(\bar{\boldsymbol{f}})\rangle, \quad (8)$$

where $sv_i(\bar{\boldsymbol{f}})$ denotes the i th subvector of $\bar{\boldsymbol{f}}$, $sq_i(sv_i(\bar{\boldsymbol{f}}))$ means that the ith subquantizer quantizes $sv_i(\bar{\boldsymbol{f}})$, and the notation $\langle sq_i(sv_i(\bar{\boldsymbol{f}})\rangle$ denotes that quantization index of $sq_i(sv_i(\bar{\boldsymbol{f}}))$. Because we only use quantization indices to represent a segment, we elevate search efficiency and reduce the required storage space.

After finding the nearest centroid to each subvector of $\bar{\boldsymbol{f}}$, we define a segment feature vector $\boldsymbol{s}_f(t_s:t_e)$ as concatenation of these centroids, i.e. $\boldsymbol{s}_f(t_s:t_e) = (\boldsymbol{c}_{idx_1}, \boldsymbol{c}_{idx_2}, ..., \boldsymbol{c}_{idx_p})$, where $\boldsymbol{c}_{idx_i}$ denotes the centroid corresponding to index idx_i. Note that these centroids may be clustered from collections of VLAD or color layout features.

- Frame descriptor and frame feature

With the similar concept, a frame descriptor is generated by product quantizing frame features into quantization indices. The main concept is that concatenating the nearest centroids of subvectors would generate an approximate vector for this frame. The frame descriptor representation $\boldsymbol{e}_t$ associated with the frame $\boldsymbol{f}_t$ is obtained by

$$\boldsymbol{e}_t = (idx_1, idx_2, ..., idx_p),$$
$$\text{and } idx_i = \langle sq_i(sv_i(\boldsymbol{f}_t)\rangle. \quad (9)$$

After finding the nearest centroid to each subvector of a frame, we modify a frame feature vector as concatenation of these centroids. Note that these centroids may be clustered from collections of VLAD or color layout features. The updated frame features are approximate to that described in Section 4.1. In the following, we use the modified frame features for distance calculation.

4.3 *Frame Search*

With the representation mentioned above, from database videos we search frames having content similar to that in query keyframes. For a query frame descriptor $\boldsymbol{q}$, we search similar frames from all frames in database videos as follows.

1) The n_a nearest centroids to each subvector $sv_i(\boldsymbol{q})$, $1 \le i \le p$, are first found. Let's denote the set of nearest centroids for $\boldsymbol{q}$ by $\{\mathcal{N}_1, \mathcal{N}_2, ..., \mathcal{N}_p\}$, where $\mathcal{N}_i = \{\langle c_1^i\rangle, \langle c_2^i\rangle, ..., \langle c_{n_a}^i\rangle\}$. Assume that a database video is divided into Z segments $\boldsymbol{s}_1, \boldsymbol{s}_2, ..., \boldsymbol{s}_Z$, and segment descriptor for each segment and frame descriptors for frames in each segment are generated by the process described in Section 4.2. For the jth segment, one nearest centroid is found for each subvector in $\boldsymbol{s}_j$, and the set of nearest centroids for this segment is denoted by $\mathbb{N}_j = \{\langle c_1^j\rangle, \langle c_2^j\rangle, ..., \langle c_p^j\rangle\}$. Note that quantization indices of these centroids are used in the following processes.
2) We compare the sets of nearest centroids of the query frame $\boldsymbol{q}$ with that of segment $\boldsymbol{s}_j$,
$m(\boldsymbol{q}, \boldsymbol{s}_j) = \sum_{i=1}^{p} |\mathcal{N}_i \bigcap \mathbb{N}_j|$, (10)
where $m(\boldsymbol{q}, \boldsymbol{s}_j)$ indicates the number of identical quantization indices between $\boldsymbol{q}$ and $\boldsymbol{s}_j$. If this value is larger than the value of n_c, this segment is claimed to be matched with $\boldsymbol{q}$.
After finding all matched segments, we then calculate the Euclidean distance (based on VLAD or color layout) between $\boldsymbol{q}$ and the matched segments $\{\boldsymbol{s}_j\}$. Only the segments that have the smallest n_f distances to $\boldsymbol{q}$ are selected for the final procedure.
3) We compare $\boldsymbol{q}$ with frames $\{\boldsymbol{f}_k\}$ in a selected segment based on the Euclidean distance calculated by frame features. The frames in $\{\boldsymbol{f}_k\}$ that have the n_r smallest distances to $\boldsymbol{q}$ are selected, and these frames are claimed to match with $\boldsymbol{q}$.

4.4 *Temporal Alignment*

After the process mentioned above, we would like to determine whether a segment of the query video matches with any segment of a database video. First, the frame-level similarity between a query keyframe $\boldsymbol{q}_i$ and a database keyframe $\boldsymbol{f}_t$ is defined as the inner product between them, if they are matched. Otherwise, similarity between them is set as 0. That is,

$$sim(\boldsymbol{q}_i, \boldsymbol{f}_t) = \begin{cases} \boldsymbol{q}_i \cdot \boldsymbol{f}_t, & \text{if } \boldsymbol{f}_t \text{is matched with } \boldsymbol{q}_i, \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

After defining frame-level similarity, we attempt to find a clip in a database video that has similar content to the query video. In our work, we exploit the temporal Hough transform [9] to evaluate sequence matching. The temporal Hough transform is a voting technique that accumulates frame-level similarities between frame matches with similar temporal shifts. Figure 5 illustrates the temporal Hough transform.

Assume that Q is the set of keyframes in the query video, and B is the set of keyframes in the database video r. The histogram value of temporal shift δ with respect to the database video r is

$$h(r, \delta) = \sum_{\boldsymbol{q}\in Q, \boldsymbol{b}\in B} sim(\boldsymbol{q}, \boldsymbol{b}), \quad (12)$$

in which the temporal shift $\delta = t_b - t_q$, where t_b and t_q are timestamps of the frames $\boldsymbol{b}$ and $\boldsymbol{q}$, respectively. The value $h(r, \delta) > 0$ if there is at least a frame match between the query video and the database video r. We then find the peak value of the histogram, which correspond to the most probable time shift δ^* between the query video and its corresponding video copy segment in the database video. The first frame in the database video with time shift δ^* indicates start of the video copy segment, and the last frame with time shift δ^* indicates end of the video copy segment. For each database video, we can find a segment similar to the query video, and only that have segments with the corresponding Hough histogram values larger than a

threshold are claimed to convey a video copy to the query video.

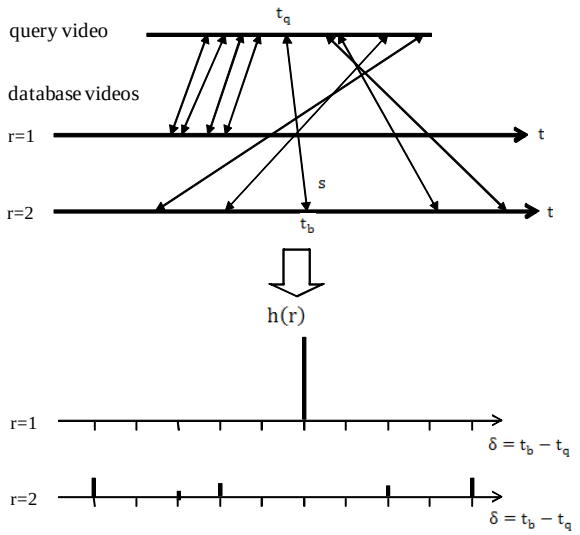

Figure 5. Top: frame matches between a query video and two database videos. Bottom: the corresponding histograms in the temporal Hough space. The number of frame matches between the query video and the 2nd database video is more than that between the query video and the 1st database video. However, high temporal shift consistency indicates that the first database video conveys a video copy segment to the query video.

Table 3. The values of parameters in this work.

Features	$M = 64$, $R_W = R_H = 4$
Descriptors	$p = 16$, $K = 16$
Frame search	$n_a = 16$, $n_c = 4$, $n_f = 2048$, $n_r = 512$

5. EXPERIMENTS

5.1 *Evaluation dataset and performance metric*

This work is evaluated based on the dataset in the content-based copy detection task of TRECVID 2010 [16]. There are 3167 videos, with totally 190 hours. Each query video is generated from a part of a database video, with sort of video transformations, and is arbitrarily concatenated with irrelevant video clips at the start or at the end. We randomly selected ten database videos to generate query videos, and each of them is applied with one of eight video transformations defined in TRECVID 2010. Therefore, there are totally eighty query videos in this evaluation. Values of parameters introduced in Section 3 and 4 are listed in Table 3.

Normalized detection cost ratio (NDCR) is used to measure performance, which integrates the cost of missing and false positives. Here, NDCR does not take how long the intersection between the detected segment and the ground truth into account. Therefore, a returned segment is considered a true positive if it overlaps with any region of the ground truth. NDCR is defined as

$$NDCR = P_{MISS} + \beta \times R_{FA}, \quad (13)$$

where $\beta = C_{FA}/(C_{MISS} \times R_t)$, and C_{MISS} and C_{FA} are the costs for a miss and a false alarm, and R_t is the priori target rate. The value P_{MISS} and R_{FA} are the conditional probabilities of a miss and a false alarm, respectively. These parameters are set according to the advice from TRECVID 2010. In TRECVID 2010, there are two profiles to compute NDCR: the "no false alarm" profile, and the "balanced" profile. The "no false alarm" profile heavily punishes false alarms, while the "balanced" profile does not. Many researches use the best NDCRs of the "balanced" profile to evaluate performance. The smaller NDCR is, the better the performance is.

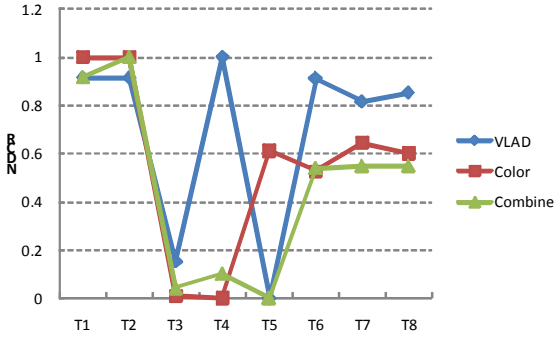

Figure 6. The best NDCRs of the "balanced" profile for (a) VLAD; (b) Color layout; and (c) Combination.

5.2 *Performance*

Figure 6 shows the best NDCRs for different features (VLAD and color layout) under eight different video transformations. From this figure we see that VLAD is particularly robust to gamma change (T5), which does not greatly alter local feature points. VLAD and color layout have the largest performance variations for T4 (strong reencoding). The reason is that color distribution does not change under strong encoding, while artifacts may cause many noisy keypoints for VLAD.

Two features have different strengths to resist different transformations. Therefore, we devise a mechanism to combine results obtained based on two features to achieve the best performance. From the detection results based on VLAD, a ratio is calculated:

$$R_{VLAD} = \frac{h(r_1^*, \delta_1^*)}{h(r_2^*, \delta_2^*)}, \quad (14)$$

where r_1^* is the database video that has the most similar content to the query, with time shift δ_1^*, and r_2^* is the database video that has the second most similar content to the query, with time shift δ_2^*. The ratio R_{VLAD} indicates that, relative to the second most similar video copy segment, how the most similar video copy segment corresponds to the query video. Similarly, we compute the ratio R_{CL} for color layout. For query video, we separately search video copies based on VLAD and color layout. The feature that draws larger ratios is used

to determine the copy detection result. Overall, performance of the combination mechanism is better than that of single features.

Based on the best NDCRs of the "balanced" profile, we compare our results with that in [11]. Chu et al. used trajectories constructed by tracking SURF keypoints to describe object movement in videos. The bag of trajectory (BOT) model is then used to represent videos. This work also adopts local feature points and thus is suitable for performance comparison. Figure 7 shows comparison in terms of the best NDCRs of the "balanced" profile. Overall, we have obtained better performance than [11] for all transformation types. The method in [11] returns more false alarms and increases NDCR greatly. This experiment verifies that by integrating multiple features our work achieves promising performance.

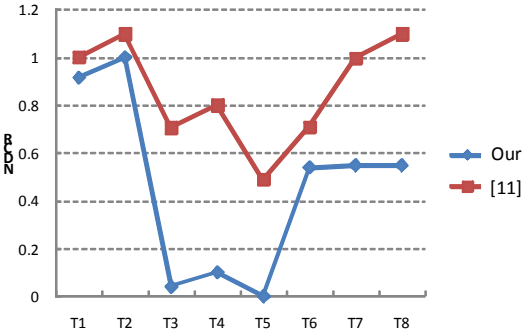

Figure 7. Performance comparison between our work and [11], based on the best NDCRs of the "balance" profile.

6. CONCLUSION

We have presented a video copy detection system based on product quantization. First, locally aggregated descriptors (VLAD) and color layout are extracted to represent video keyframes. They are high-dimensional feature vectors, and we use the idea of product quantization to efficiently index them. Pairs of keyframes (respectively from the query video and from a database video) that are quantized into similar subspaces are then found. Finally, the temporal Hough transform is utilized to find the most probably time shift between the query video and a database video. The experimental results verify that combining VLAD and color layout achieves promising performance. In the future, more experiments will be designed to emphasize the effectiveness of product quantization, and more robust features will be studied.

ACKNOWLEDGEMENT

This work was partially supported by the National Science Council of ROC under NSC 99-2221-E-194-036 and NSC 100-2221-E-194-061.

REFERENCES

[1] A. Oliva and A. Torralba, "Modeling the Shape of the Scene: A Holistic Representation of the Spatial Envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145-175, 2001.

[2] A. Torralba, R. Fergus, and Y. Weiss, "Small Codes and Large Databases for Recognition," *Proceeding of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[3] X. Wu, Y. Zhang, Y. Wu, J. Guo, and J. Li, "Invariant Visual Patterns for Video Copy Detection," *Proceedings of International Conference on Pattern Recognition*, 2008.

[4] H. Jegou, M. Douse, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 01, pp.117-128, 2011.

[5] J. Law-To, A. Joly, and N. Boujemaa. "Muscle-VCD-2007: A Live Benchmark for Video Copy Detection," http://www-rocq.inria.fr/imedia/civr-bench, 2007.

[6] J. Law-To, L. Chen, A. Joly, I. Laptev, O. Buisson, V. Gouet-Brunet, N. Boujemaa, and F. Stentiford, "Video Copy Detection: A Comparative Study," *Proceedings of ACM International Conference on Image and Video Retrieval*, pp. 371-378, 2007.

[7] C.-Y. Chiu, C.-S. Chen, and L.-F. Chien, "A Framework for Handling Spatiotemporal Variations in Video Copy Detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 3, pp. 412-417, 2008.

[8] M.-C. Yeh and K.-T. Cheng, "Video Copy Detection by Fast Sequence Matching," *Proceedings of ACM International Conference on Image and Video Retrieval*, 2009.

[9] M. Douze, H. Jegou, and C. Schmid, "An Image-based Approach to Video Copy Detection with Spatio-temporal Post-filtering," *IEEE Transactions on Multimedia*, vol. 12, no. 4, pp. 257-266, 2010.

[10] M. Douze, H. Jegou, C. Schmid, and P. Perez, "Compact Video Description for Copy Detection with Precise Temporal Alignment," *Proceedings of European conference on Computer Vision*, vol. 6311, pp. 522-535, 2010.

[11] W.-T. Chu, P.-C. Chuang, and J.-J. Yu, "Video Copy Detection Based on Bag of Trajectory and Two-Level Approximate Sequence Matching," *Proceedings of IPPR Conference on Computer Vision, Graphics, and Image Processing Conference*, 2010.

[12] K. Mikolajczyk and C. Schmid, "Scale and Affine Invariant Interest Point Detectors," *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63-86, 2004.

[13] D. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[14] H. Jegou, M. Douze, C. Schmid, and P. Perez, "Aggregating Local Descriptors into a Compact Image Representation," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pp.3304-3311, 2010.

[15] D. Nister and H. Stewenius, "Scalable Recognition with a Vocabulary Tree," *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp.2161-2168, 2006.

[16] TRECVID, http://www-nlpir.nist.gov/projects/tv2010/tv2010.html#ccd