Content-Based Video Search over 1 Million Videos with 1 Core in 1 Second

Shoou-I Yu[1], Lu Jiang[1], Zhongwen Xu[2], Yi Yang[2], Alexander G. Hauptmann[1]
[1]Language Technologies Institute, Carnegie Mellon University
[2]Centre for Quantum Computation and Intelligent Systems, University of Technology, Sydney
{iyu, lujiang, alex}@cs.cmu.edu, zhongwen.xu@student.uts.edu.au, yi.yang@uts.edu.au

ABSTRACT

Many content-based video search (CBVS) systems have been proposed to analyze the rapidly-increasing amount of user-generated videos on the Internet. Though the accuracy of CBVS systems have drastically improved, these high accuracy systems tend to be too inefficient for interactive search. Therefore, to strive for real-time web-scale CBVS, we perform a comprehensive study on the different components in a CBVS system to understand the trade-offs between accuracy and speed of each component. Directions investigated include exploring different low-level and semantics-based features, testing different compression factors and approximations during video search, and understanding the time v.s. accuracy trade-off of reranking. Extensive experiments on data sets consisting of more than 1,000 hours of video showed that through a combination of effective features, highly compressed representations, and one iteration of reranking, our proposed system can achieve an 10,000-fold speedup while retaining 80% accuracy of a state-of-the-art CBVS system. We further performed search over 1 million videos and demonstrated that our system can complete the search in 0.975 seconds with a single core, which potentially opens the door to interactive web-scale CBVS for the general public.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, Experimentation, Performance

Keywords

Content-Based Video Search; Multimedia Event Detection; Product Quantization; Reranking; Semantic Concept Detection

1. INTRODUCTION

Large-scale analysis of video data becomes ever more important as we see an unprecedented growth of user-generated videos on the Internet. To perform search, current video search engines mostly rely on user provided text metadata. However, text metadata is not a comprehensive representation of the video as an user cannot possibly annotate all pieces of information in a video. Also, it is often the case that the users do not provide any metadata at all, and these videos can never be retrieved by a metadata-based search engine. Therefore, we turn to content-based video analysis, which leverages the very rich content in the visual and audio channels of a video, for search. In the multimedia community, this task has been studied in the form of Multimedia Event Detection (MED) [20]. The task can be split into two main phases: feature extraction and search. The main goal of feature extraction is to extract discriminative feature representations from the video's raw visual and audio channels. Extracting effective, generalizable and efficient representations is a key challenge of this phase. During the search phase, which we refer to as Content-Based Video Search (CBVS), the queries are a set of example videos, and the CBVS system will utilize the previously extracted features to retrieve relevant videos. As the extracted feature representations are not text-based, traditional text-retrieval techniques are not directly applicable, and new search techniques need to be developed.

Much research have been proposed to improve the accuracy of content-based video analysis systems [19, 28, 30, 10, 25, 8, 23, 29, 2] and optimizing the speed of feature extraction [14, 16], but an important direction which is still under-explored is achieving real-time CBVS. For a CBVS system to be conveniently utilized by an user, the retrieval of large video collections needs to be done in real-time. However, near real-time content-based video search is a challenging task because, unlike text vectors which are high dimensional but very sparse, visual and audio features are all high dimensional dense vectors. Thus many state-of-the-art CBVS systems [30] still require a few minutes to perform retrieval, which is still too slow for interactive use. Though there are some work [17] which have found that compact semantics-based representations potentially achieve better performance than low-level features, these studies did not focus on utilizing such compact representations to optimize the speed of CBVS systems. Therefore, in this paper we focus on performing a comprehensive study on the different directions and methods to optimizing the speed of CBVS systems.

We introduce a number of aspects to optimize existing state-of-the-art CBVS systems, which include the set of features used, the learning algorithm, different compression and approximation configurations, and the number of reranking iterations. Comprehensive experiments to explore the speedup and accuracy trade-offs in each direction were performed, which resulted in a CBVS system that achieves a 10,000-fold speedup while retaining 80% Mean Average Precision[1] (MAP) of a top performing system [30] in the

[1]Official metric of TRECVID MED task 2014

ICMR'15, June 23–26, 2015, Shanghai, China.

http://dx.doi.org/10.1145/2671188.2749398.

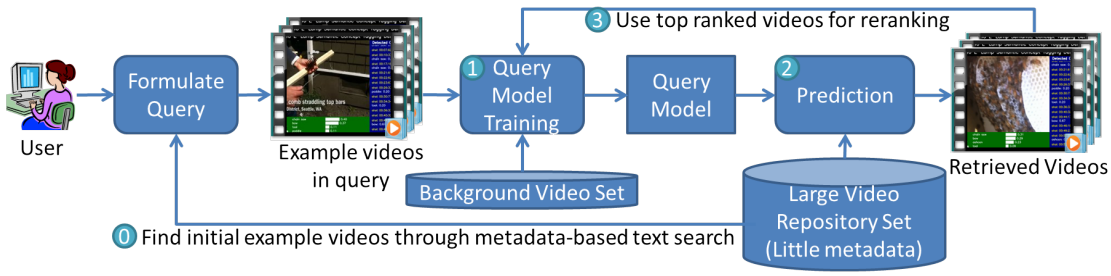


Figure 1: Pipeline of CBVS system.

TRECVID 2014 MED [20] task. In sum, the contributions of our paper are as follows.

1. A CBVS system which can search over 1 million videos with only 1 core in about 1 second is proposed. The proposed CBVS system provides over 10,000-fold speedup while retaining 80% of the MAP of a state-of-the-art system [30]. This potentially opens the door to interactive web-scale CBVS for the general public.
2. We comprehensively explore the speedup and accuracy trade-offs of many different components in state-of-the-art CBVS systems. These experiments may provide insight to the designing of future CBVS systems.

2. THE CBVS SYSTEM

Assume the following typical usage scenario for CBVS. The user performs an initial metadata-based text search to find a few videos of interest in the video repository. As the user would like to find more relevant videos, the user-selected videos of interest, which we refer to as the "example videos", can be provided to the CBVS system. The system will retrieve, based on video content, more videos related to the query and return them to the user. The videos retrieved by CBVS may or may not have metadata, which is the key advantage of CBVS systems over metadata-based text search.

Our proposed CBVS system, which follows the pipeline of the top-performing TRECVID MED systems [19, 30, 23], has three main components: query model training, prediction and reranking. The query model training step trains query models for each feature representation based on the example videos provided by the user. The prediction step takes the trained query models and performs prediction on an indexed repository set. Weighted fusion is performed to fuse the features from different modalities. Finally, the reranking step will enhance the initial retrieved results. The pipeline of our proposed CBVS system is shown in Figure 1.

To design a fast CBVS system, understanding the details of each component is necessary to find the sweet spot between speed and accuracy. Also, since we are allowed to perform offline preprocessing to index videos, the algorithms should be carefully designed such that most of the time-consuming computation is done in the offline phase, and only the necessary computation is done in the time-critical online phase. In the following sections, the details of each step are described and the implications of each design choice are discussed.

2.1 Fast Query Model Training

Query model training learns a model to discriminate relevant and irrelevant videos to the user query. To train the model, example videos are treated as positive samples and background videos are treated as negative samples. Support Vector Machines (SVM) [3] have been a popular choice for training query models [30, 22]. Furthermore, linear SVMs are very efficient during prediction, because prediction on an unseen instance only involves a single dot-product. In the following paragraphs, we will describe how to efficiently train query models for both linear and non-linear feature representations with linear SVMs.

It has been shown that SVMs with χ^2-kernels are very effective for bag-of-words features [22], which is an effective encoding method for low-level features. However, for SVMs, kernel-based prediction of an input vector is slow because prediction requires a dot product with all support vectors. Therefore, to still utilize kernel-based features but achieve fast prediction, the Explicit Feature Map [26] can effectively convert a kernel-based feature to a linear-based feature [22]. This results in substantial speedup with unsubstantial performance drop.

We now focus on efficient training of linear SVMs, which needs to be fast as the query models are trained on the fly. Linear SVMs can be trained at approximately $\mathcal{O}(nd)$ [13] time by solving the primal, or at approximately $\mathcal{O}(n^2d + n^3)$ [1] time by solving the dual, where n is the number of training vectors and d is the dimensions of a feature representation. There are two terms in the complexity formula of solving the SVM dual: pair-wise dot-product matrix computation time $\mathcal{O}(n^2d)$ and model training time $\mathcal{O}(n^3)$. The primal is in general faster to solve, but when d becomes very large, the dual can be faster to solve if the dot-product matrix can be precomputed offline. The online cost of solving the dual form becomes only $\mathcal{O}(n^3)$, which in practice is very fast. Experiments in Section 3.1 substantiates this claim.

For efficiently solving the dual form of SVMs, the pair-wise dot-product matrix needs to be precomputed offline. If we have n_c videos in the repository set, the entire dot-product matrix is of size $\mathcal{O}(n_c^2)$, which is prohibitively large as the repository set may have up to millions of videos. However, the training of SVMs only require the pair-wise dot-product of all vectors in the training set, which only includes the example videos and the background set, so precomputing the full matrix is not required. Let $C \in \mathbb{R}^{d \times n_c}$ be the feature matrix for all n_c videos in the search repository. Let $E \in \mathbb{R}^{d \times n_e}$ be the feature matrix for n_e example videos, which is a subset of C as the example videos are included in the search repository. Let $B \in \mathbb{R}^{d \times n_b}$ be the feature matrix for n_b background videos. We combine E and B to create the training feature matrix $F = [E, B] \in \mathbb{R}^{d \times (n_e + n_b)}$. Then the pair-wise dot-product matrix is as follows:

$$F^T F = \begin{bmatrix} E^T \\ B^T \end{bmatrix} \begin{bmatrix} E & B \end{bmatrix} = \begin{bmatrix} E^T E & E^T B \\ B^T E & B^T B \end{bmatrix}. \quad (1)$$

Since the background set is fixed and relatively small (several thousand videos), we can precompute $B^T B$ and store it in memory. To quickly acquire $E^T B$, we first precompute $C^T B$ offline, and during the online step only n_e rows need to be read from $C^T B$ to form $E^T B$ because E is a subset of C. $C^T B$ is stored on disk. Finally, the only part remaining is $E^T E$, which can be computed quickly on the fly as there will not be many example videos (tens of videos).

To summarize, the three steps taken to efficiently train a linear SVM is as follows. First, the pair-wise dot-product matrix for the query is assembled on the fly, which is fast because most of the matrix is already precomputed. Second, the SVM is trained by solving the dual, and a set of support vectors and their corresponding weights are acquired. Finally, the dual solution is converted to the primal solution, so that the final linear SVM model is only a d dimensional vector which supports fast prediction. This step consists of a weighted sum of all the support vectors' raw features, i.e. matrices E and B. This can be performed quickly if E and B are pre-loaded in main memory. In our actual implementation, B is pre-loaded, and E is read from disk on the fly, which is still fast since there are only a few example videos.

2.2 Fast Prediction

We describe how one could achieve fast prediction of linear SVM models over testing data. The prediction of a single instance requires a dot product with the model. A simple way to find top-ranked videos in the repository set is to exhaustively compute the dot-product for all videos. However, this is too inefficient as the repository size grows larger. As the dot product is highly related to the Euclidean distance [21], approximate nearest neighbor techniques can be used to save computation time. Many approximate nearest neighbor search methods have been proposed, such as KD-Tree-based Approximate Nearest Neighbors [18] and Locality Sensitive Hashing [5], but these methods tend to be too memory consuming [9]. Also, these methods tend to be not as effective when the number of dimensions of the vector space is significantly higher than the number of instances in the search set. Therefore, we focus on using Product Quantization (PQ) [9], which takes into account efficiency in both computation and memory. Given a d dimensional feature representation, PQ will split the high-dimensional space into s subspaces, where each subspace is of dimension $c = \frac{d}{s}$ assuming d is divisible by s. Then, the vector belonging to each of the subspace are quantized with a short code. The short codes from different subspaces are concatenated to form a compact representation of the original vector. The dot-product between two vectors can be efficiently estimated using this compact representation.

The PQ framework has a codebook learning, quantization, and prediction phase. In the codebook learning phase, a codebook for each small subspace is learned with K-Means clustering where $K = 256$. $K = 256$ is preferable because the cluster ID can be represented by a single byte. We define $D_i \in \mathbb{R}^{c \times 256},\ 1 \leq i \leq s$ as the codebook for the i-th subspace.

In the quantization phase, each input vector is split into s subspaces and quantized according to the corresponding codebooks. Specifically, let $v = [v_1^T, v_2^T, \ldots, v_s^T] \in \mathbb{R}^d$ be an input vector, where $v_i \in \mathbb{R}^c$ falls in subspace i. For each v_i, a nearest neighbor is found in D_i, and v_i is represented by the ID of the nearest neighbor: q_i, which ranges from 0 to 255 and takes up 1 byte. At the end, v is quantized with s bytes, i.e. $q = [q_1, q_2, \ldots, q_s] \in \{0, 1, \ldots, 255\}^s$. This is a compact representation in that we have succeeded in compressing a d dimensional floating point vector, which takes up at least $4d$ bytes assuming single-precision, to only s bytes. Therefore, the effective compression factor is $\frac{4d}{s} = 4c$.

In the prediction phase, given a query vector $w \in \mathbb{R}^d$ (the linear SVM model), we would like to quickly compute the dot-product over all quantized vectors (videos in repository set). The naïve way to compute the dot-product between two vectors requires d multiplications and additions, but PQ exploits the fact that the vectors in the repository set have been quantized to achieve fast dot-product computation. First, the query vector is also split according to the s subspaces, i.e. $w = [w_1^T, w_2^T, \ldots, w_s^T]$, where $w_i \in \mathbb{R}^c$. Then for each subspace, the dot-product between a subspace of the query vector (w_i) and all codewords in the corresponding codebook of the subspace (D_i) is computed and stored in a lookup table (LUT) Specifically, $L = [D_1^T w_1, D_2^T w_2, \ldots, D_s^T w_s] \in \mathbb{R}^{256 \times s}$. With the LUT, computing the dot-product between the query vector and a quantized vector consists of 1) for each subspace, look up the dot-product value given q_i (the quantized vector's cluster membership), and 2) sum up the dot-products from each subspace. For example, we would compute the dot-product between q and w as follows. Let $L(k, l)$ denote the element on the k-th row and l-th column of the LUT. Then the dot product is $\sum_{i=1}^{s} L(q_i + 1, i)$. $+1$ is required because cluster membership starts from 0. Therefore, with the LUT, the complexity of a dot-product drops from $\mathcal{O}(d)$ to $\mathcal{O}(s)$. We see that not only does PQ compress the feature representation, but also the prediction time is decreased.

In practice, codebook learning and quantization are performed offline, and the prediction step is performed online. To accelerate the prediction step, the codes for the videos in the search repository set are all loaded into memory. This is feasible because PQ significantly compresses the feature representations.

However, when the amount of videos to be searched becomes very big, spending $\mathcal{O}(s)$ per video may still be too long. [9] proposes to build an inverted index to prune out search candidates that are highly unlikely to be nearest neighbors. The codewords in the inverted index are acquired by quantizing with another coarse codebook learned in the d dimensional space. However, in [9], the largest d is only 960. This is very small compared to the dimensions of our video representations, which can go up to 100,000 dimensions. Therefore, this method may not be applicable in our scenario, and we tackle this problem from another direction. Instead of summing the dot-products for all s subspaces, we propose to only sum over a dynamically selected set of subspaces.

High Variance Subspaces First

We propose to speed up the dot-product computation process by only summing the dot-product for selected subspaces. The selected subspaces should be the subspaces which can enable us to quickly discriminate whether a video could be a top-ranked video or not. Motivated by KD-Tree-based Approximate Nearest Neighbors [18], which perform KD-tree splits according to dimensions having larger variance, we only select the subspaces which have high variance in the dot-product values. The intuition is that these subspaces provide more information in separating a relevant video from an irrelevant one. We name this method High Variance Subspaces First (HVSF).

The variance can be efficiently computed on the fly based on the LUT. For the i-th subspace, let $u_i(1), u_i(2), \ldots, u_i(256)$ correspond to the frequency of each codeword occurring in the repository set, which can be computed offline during the quantization step. The sum of all frequencies is n_c, i.e. $\sum_{j=1}^{256} u_i(j) = n_c$. We denote the average dot-product values of the classifier w for subspace i as $\mu_i = \sum_{j=1}^{256} (u_i(j) L(j, i)) / n_c$. Then the variance for subspace i is as follows: $\sigma_i^2 = \sum_{j=1}^{256} u_i(j) \left(L(j, i) - \mu_i\right)^2 / n_c$.

For the prediction phase of HVSF, there are three main steps. First, high variance subspaces are selected. Second, we perform summation of dot products only over the selected subspaces. Finally, to obtain a more accurate ranked list for the top-ranked videos, the top scoring videos (2500 in our experiments) are selected and their exact dot-product (summation over all subspaces) computed. In the following sections, HVSF hX means that only the top $\frac{100}{h}\%$ of the subspaces with higher variance were used for the dot-product summation.

Stages	Query Model Training	Prediction	Reranking
Stored in memory (shared by all queries)	1. Background set pair-wise dot-product matrix ($B^T B$) 2. Background set raw features (B)	1. Quantized features for large video repository set	(All loaded by previous steps)
Read from disk	1. Example video v.s. background set dot-product matrix ($E^T B$) 2. Example video raw features (E)	(None)	1. Top-ranked video v.s. background set dot-product matrix 2. Top-ranked video raw features
Compute on the fly	1. Compute example video pair-wise dot-product matrix ($E^T E$) 2. Assemble training set dot-product matrix ($F^T F$) 3. Train SVM model, convert SVM dual solution to primal solution 4. Quantize SVM model	1. Predict with quantized SVM model 2. Fuse ranked lists from all features	1. Compute top-ranked video pair-wise dot-product matrix 2. Assemble top-ranked video augmented training set dot-product matrix 3. Continue to Query Model Training step 3

Table 1: Overview of memory usage, disk usage and computational tasks of each stage in the online phase of the CBVS system.

2.3 Fast Reranking

Unsupervised reranking of an initial ranked list, which is also known as pseudo-relevance feedback, has shown to be a promising method in improving CBVS results, increasing the MAP by at least 1% absolute in TRECVID MED 2014 [30]. Under the assumption that top-ranked videos in a retrieved ranked list are highly likely to be correct, the reranking algorithm can obtain more training data for the query model by treating the top-k-ranked videos as positive examples. These positive examples are added into the example video set, and a better query model could be trained with the newly added positive examples. However, we are not perfectly sure that the top-k-ranked videos are positives, so they should be treated with more scrutiny than the user-selected example videos. Also, the top k videos with slightly lower rank are less likely to be positives than the higher ranked videos. Therefore, [11] proposed to assign gradually decreasing weights to the loss incurred by the top-k-ranked videos in the SVM loss function. In our work, we simplified the reranking method proposed in [30, 11] and arrive at the following algorithm. Let $\mathcal{T}$ be the set of videos already in the training set, e.g. the background videos and the example videos. Let $\mathcal{U}$ be the set of top-k-ranked videos in the current ranked list. In each iteration, we iterate the following three steps: 1) train a SVM model using the samples in $\mathcal{T} \cup \mathcal{U}$; 2) fix the SVM model, and select some pseudo positive samples, calculate their weights, and put them in $\mathcal{U}$; 3) adjust threshold so that more samples will be added in $\mathcal{U}$ in the next iteration.

When we fix the SVM model, let $\text{rank}(v), v \in \mathcal{U}$ be the rank of testing video v in the ranked list. The weights of a video $v \in \mathcal{U}$ can be calculated by the following closed-form solution:

$$l_v = \begin{cases} 1 & \text{if rank}(v) \leq p \\ \frac{k-\text{rank}(v)+1}{k-p} & \text{otherwise} \end{cases}, \ \forall v \in \mathcal{U}, \qquad (2)$$

where p is the pivot. For videos ranked before rank p, their weight is 1. For videos ranked after the pivot, their weights decrease linearly. The weights of videos in the training set $\mathcal{T}$ are all 1. A similar mixture weighting scheme has been shown to be effective in [11].

When we fix the samples in $\mathcal{U}$, the modified linear SVM loss function, which re-weights the loss incurred by the videos in $\mathcal{U}$, is as follows.

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{v\in\mathcal{T}}\xi_v + C\sum_{v\in\mathcal{U}} l_v\xi_v$$
$$s.t.\ \forall v \in (\mathcal{T}\cup\mathcal{U}),\ y_v(\mathbf{w}^T x_v + b) \geq 1-\xi_v,\ \xi_v \geq 0, \qquad (3)$$

where $\mathbf{w}$ is the classifier, b is the bias, ξ_v is the slack for video v, C is the cost parameter, and x_v and y_v are the feature representations and label for video v respectively. $y_v = 1$ for all videos in $\mathcal{U}$. Equation 3 can be solved with slight modifications to the SVM solver [11]. Once a model is trained for each feature, we quantize the model with our PQ codebooks and run prediction only on the top 2500 videos to achieve faster reranking.

2.4 Memory and I/O Efficient CBVS System

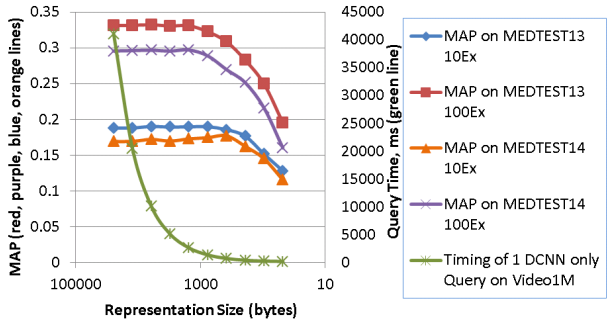
Careful design is required to implement the aforementioned 3 components in a memory and I/O efficient way. The key to achieve fast speed is try to offload as much computation as possible to the offline phase, and also try to load data that is shared across all queries into memory during the online phase. Our CBVS system takes these offline-computed data as input:

1. Product quantized feature representations: feature extraction and quantization are very time consuming and should definitely be done offline.
2. Background set pair-wise dot-product matrix ($B^T B$): as the background set is predefined and universal to all queries, the dot-product matrix of the background set itself can be precomputed for each feature representation.
3. Repository set v.s. background set pair-wise dot-product matrix ($C^T B$): as the background set and the repository set is predefined, the dot-product matrix can also be precomputed. This is for fast distance matrix building in model training and reranking, as the video examples and the videos used for reranking are from the repository set.

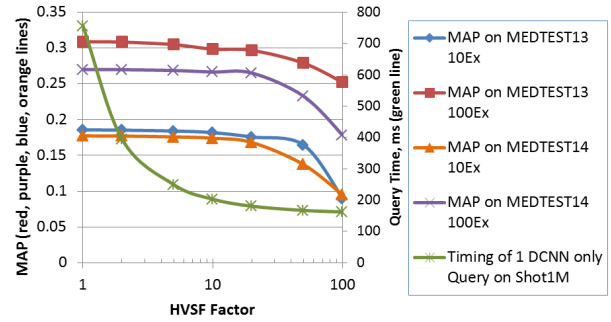
Table 1 summarizes the memory, disk access, and required computation of the online phase. The raw features of the videos in the training set are still necessary for model training, but as the raw features of the repository set is too large to fit in memory, the features of the positive examples are read from disk on the fly. However, this process is fast as there will not be many positive examples. On the other hand, the raw features of the background set, which is shared by all queries and only in the order of thousands of videos, are stored in memory. Other data that is shared across all queries, such as the background set distance matrix and quantized features for the prediction set, are all stored in memory. With this method, we are able to design an efficient CBVS system.

3. EXPERIMENTS

We tested the accuracy and efficiency of our CBVS system by following the TRECVID MED data sets and evaluation protocol, which have been widely used to evaluate CBVS systems. Experiments were performed on MEDTEST13 and MEDTEST14, which are two standard data sets defined by the organizers of TRECVID MED. These two data sets contain around 33,000 unconstrained videos (~1200 hours of video). For both data sets, the data can be



(a) DCNN accuracy and timing with different PQ compressions factors.



(b) DCNN PQ 384 byte representation: accuracy and timing with varying HVSF factors.

Figure 2: Accuracy and speed trade-off analysis for the DCNN feature. MAP is computed on MEDTEST14 10Ex. Search time is computed on Video1M set. Much of the accuracy can still be retained under aggressive PQ compression and HVSF approximation.

split into 3 sets: the positive examples, the background set and the testing set. The positive examples contain videos corresponding to different events such as Birthday Party, Performing a Bike Trick and Rock-climbing. There are two different query settings defined by the organizers: the 10 exemplar (10Ex) and the 100 exemplar (100Ex) settings. In the 10Ex (100Ex) setting, 10 (100) positive example videos are provided to the system. The background set contains 4992 videos. The positive examples and the background set combined together form the training set. The testing set contains around 24,000 videos. Query models are trained on the training set and predicted on the testing set. There are a total of 20 events/queries each for the two data sets. The evaluation metric used is Mean Average Precision (MAP).

For scalability tests, we also created a Video1M set which contains one million videos. These videos were collected by sampling one million shots from the MEDTEST14 video collection. Shot-level labels were not provided by the organizers, so this set is only used for evaluating the efficiency our CBVS system.

All our experiments were run on a single Intel Xeon E5-2640 @ 2.50GHz core. A SSD RAID is utilized to enable fast accessing of on disk data during the training and reranking steps. We now present experiments on these data sets and explore the trade-off points of each component in our CBVS system.

3.1 Fast Query Model Training

The timing of training a SVM query model is measured. We utilize LIBSVM [3] for solving the dual and LIBLINEAR [7] for solving the primal. The results are shown in Table 2. Though solving primal form SVMs are in general faster, when the dimensions of the features become large, and also when the dot-product matrix

MEDTEST14 20 Events	Training Time (ms)		Speedup
	SVM Primal	SVM Dual	
10 Ex	11257 ± 615	115 ± 14	98x
100 Ex	11228 ± 716	362 ± 84	31x

Table 2: Training time for one SVM model with varying number of positives by solving either the primal or dual form on the Improved Dense Trajectories Fisher Vector feature (110592 dimensions). The dual form assumes that the dot-product matrix is already computed offline. Both models assume that the required data is loaded into memory. 4992 negatives are used for all models.

	Original Feature Dimensions	Subspace Dimensions	PQ Subspaces (Bytes in Representation)
DCNN	98304	256	384
MFCC	12288	64	192
Sports	512	2	256
YFCC	640	2	320
Total additions per prediction of video			1152
Total additions with 5x HVSF			230

Table 3: PQ parameters for the DMSY CBVS system. The total number of PQ subspaces (s) is the total number of additions required to compute the prediction score for a video.

can be precomputed, solving the dual form is significantly faster. Once the dual form solution is acquired, we still need to convert the solution to primal form, which is also fast as shown in Table 5.

3.2 Effective Feature Representations and PQ with HVSF

The accuracy and speed of the CBVS system relies heavily not only on the set of features utilized, but also on the compression and approximation of each feature representation. We explore the trade-offs of these design decisions in this section. There are two main kinds of feature representations: low-level features and semantics-based features. Low-level features are extracted directly from the visual or audio channels and do not contain semantic information. Effective low-level visual features include the Improved Dense Trajectories Fisher Vectors (IDT FV, 110592 dimensional) from [27, 4]. Effective low-level audio features include the MFCC (12288 dimensional) [30] features. Semantics-based features are features derived from the prediction of a set of semantic concept detectors. These detectors take low-level features as input and detect whether the semantic concept it was trained for exists or not in the video. Effective semantics-based features include those concept detectors trained on the Yahoo Flickr Creative Commons (YFCC, 640 dimensional) [24, 30] and Google Sports set (Sports, 512 dimensional) [15]. Both YFCC and Sports concept detectors are trained by a SVM-based self-paced learning pipeline [12] using IDT FV. On the other hand, the Deep Convolutional Neural Network (DCNN, 98304 dimensional) features from [30, 28] is actually a hybrid feature, including both low-level neural network features and semantics-based features corresponding to prediction scores of object detectors trained on ImageNet [6]. Explicit Feature Maps were applied to the DCNN and MFCC features. Note that the dimensions of our features have been slightly enlarged (by appending zeros) so that they are divisible by 128 for YFCC and Sports, and divisible by 2048 for IDT FV, DCNN and MFCC. This is done to conveniently perform PQ with high compression factors.

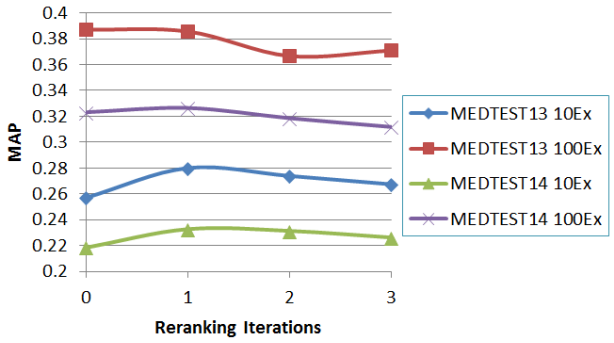


Figure 3: Reranking performance under different amounts of reranking iterations on the DMSY + HVSF 5X system.

MAP per Condition	[30]	DMSY + 5X HVSF + 1R	Performance Difference
MEDTEST13 10Ex	0.313	0.280	89.4%
MEDTEST13 100Ex	0.464	0.386	83.1%
MEDTEST14 10Ex	0.285	0.233	81.6%
MEDTEST14 100Ex	0.419	0.326	77.9%
Time per query (ms)	558400 × 12	507.7	1319834%

Table 4: Performance comparison between our DMSY + 5X HVSF + 1R system and the state-of-the-art MED system [30]. The breakdown of our system's timing is shown in Table 5. 12 is multiplied to the timing of the 10Ex system from [30], because the system was run on a 12 core machine with 4 K-20 GPUs. This is still an underestimate as the GPU timings were not taken into account.

Compact representations for each video enable fast search, which we propose to achieve with PQ compressions and HVSF approximation. Figure 2 shows the accuracy and speed difference under different PQ compression and HVSF factors. Even under aggressive compression, the performance of DCNN only drops slightly, but the speed of search increased by 160x if we compare the search time of the PQ 24576 byte representation with the 384 byte representation + HVSF 5X (only 20% subspaces used). This shows the effectiveness of PQ compressions and HVSF approximation.

A comprehensive study on the affect of PQ compression on different features is performed. Figure 5 shows the MAP of different features under varying compression factors. A clear trend is that the MAP of IDT FV drops as less bytes are used in the feature representation. If around 1000 bytes are used to represent each video, the most effective features are DCNN, Sports and YFCC. This hints at the effectiveness of semantics-based features. Therefore, to strike a balance between accuracy and efficiency, we utilize the feature representations and compression parameters shown in Table 3. All feature representations have been compressed to take up around 300 bytes per video. The total amount of bytes to represent a video is around 1KB. We refer to this combination as the DMSY (DCNN, MFCC, Sports, YFCC) system. As learning fusion weights for each query may be slow, the fusion weights are universal for all queries and set empirically to be 2, 0.5, 1, 1 for the DCNN, MFCC, Sports and YFCC features respectively. This system will be the basis of further experiments. IDT FV has shown to perform poorly under high compression factors, so it was not used.

3.3 Reranking

Reranking experiments based on our DMSY system combined with 5X HVSF (DMSY + HVSF 5X) were performed. Three iterations of reranking were performed, where the top 2, 4, and 6 videos were used for reranking in each iteration respectively. The pivot value was selected by $p = \lfloor \frac{k}{3} \rfloor$, where k is the amount of top-ranked videos used for reranking. The results are shown in Figure 3. Results show that 1 iteration of reranking already provides most of the performance gain. More iterations may cause a performance drop. The main reason is that the MEDTEST13 and MEDTEST14 data sets only have around 20 positives in the testing set. Therefore, when the top 4 or top 6 videos are selected for reranking, the likelihood of selecting negative videos increases significantly. For a larger data set with more positives, this problem can potentially be alleviated.

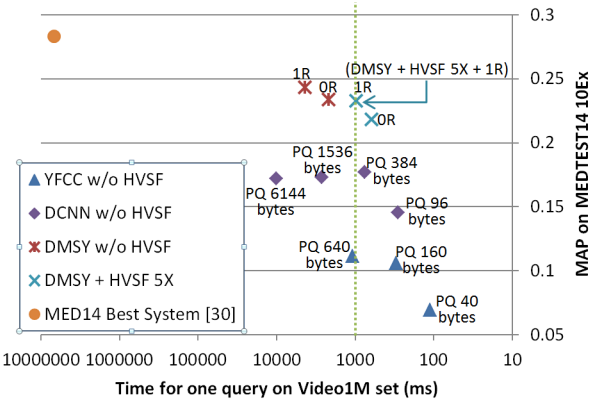
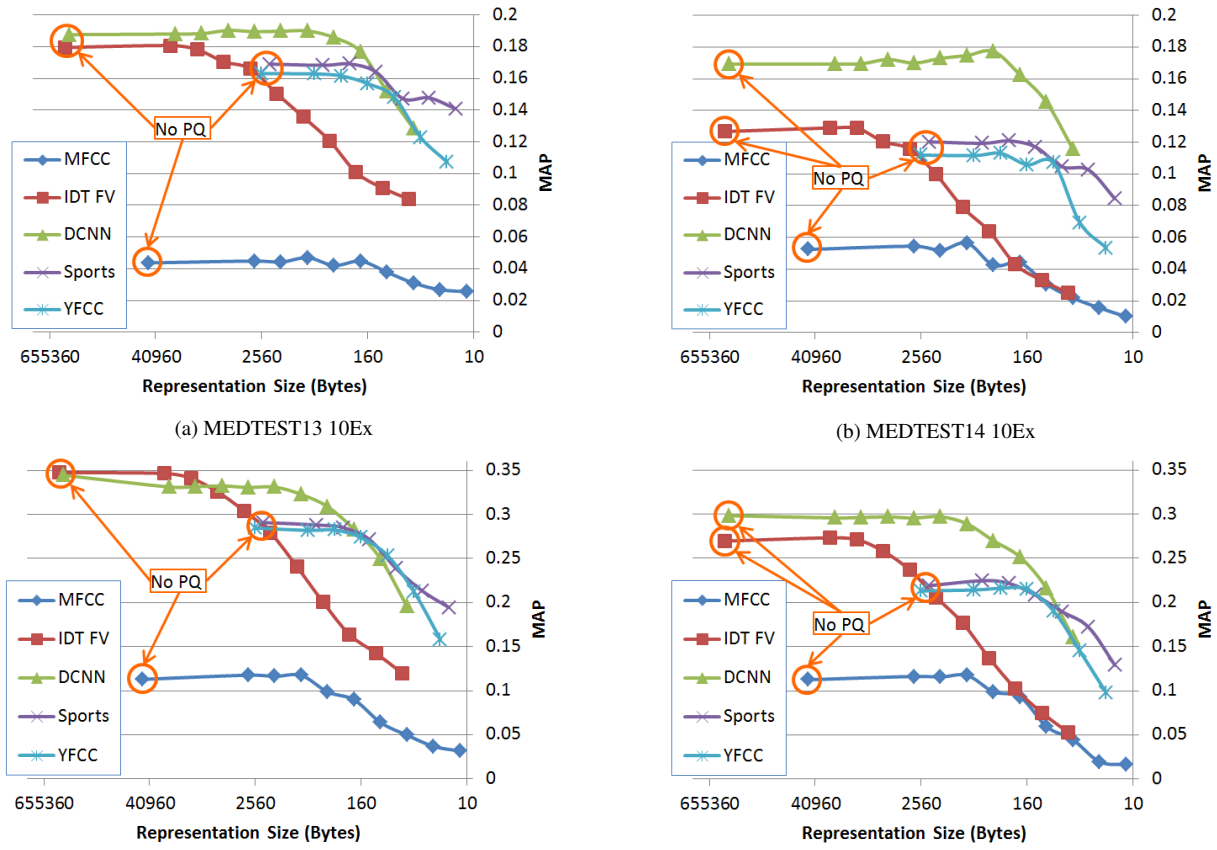


Figure 4: MAP and timing scatter plot of different CBVS systems. MAP was computed under MEDTEST14 10Ex condition. Timings were computed from searching the Video1M set. 0R means no reranking. 1R means one round of reranking.

From our previous experiments, we have found that DMSY + HVSF 5X combined with 1 round of reranking (DMSY + HVSF 5X + 1R for short) achieves high performance. Table 4 compares DMSY + HVSF 5X + 1R with the MAP of a state-of-the-art system [30], which achieved top results in the TRECVID MED 2014 task. Results show that our DMSY + HVSF 5X + 1R system retains 80% of the performance which providing a 13,000 thousand times speedup. This demonstrates the effectiveness of our proposed compression, approximation and reranking schemes.

3.4 Searching 1 Million Videos

Armed with an effective and efficient DMSY + HVSF 5X + 1R system, we tackle the challenge of searching 1 million videos on 1 core in 1 second. Query model training was done under the MEDTEST14 10Ex condition. The search was performed on the Video1M set. Qualitative results are shown in Figure 6, and detailed timing results are shown in Table 5. Our system can complete the search in 0.975 seconds, which is one of the best systems in terms of trade-off between speed and accuracy as shown in Figure 4. Note that when HVSF 5X is combined with 1 iteration of reranking, we are able achieve nearly the same performance as the DMSY system without HVSF, while requiring less time. This indicates that fast reranking could perform error correction and compensate for some of the performance loss caused by approximations. In terms of timing, as shown in Table 5, not surprisingly the prediction phase of our system takes the longest time, but with 1 round of reranking, the query model training phase also takes up a significant portion of the time. In terms of memory usage, the dot-product matrices, raw features, and quantized features for prediction took 0.37GB, 2.09GB and 1.07GB respectively. The memory usage was less than 4GB and could potentially run on a laptop.



(a) MEDTEST13 10Ex

(b) MEDTEST14 10Ex

(c) MEDTEST13 100Ex

(d) MEDTEST14 100Ex

Figure 5: Performance of different feature representations under different PQ compression factors. The representation size corresponds to the amount of bytes required to encode one video. The performance and representation size of the original features (no PQ) are also shown in the orange circles.

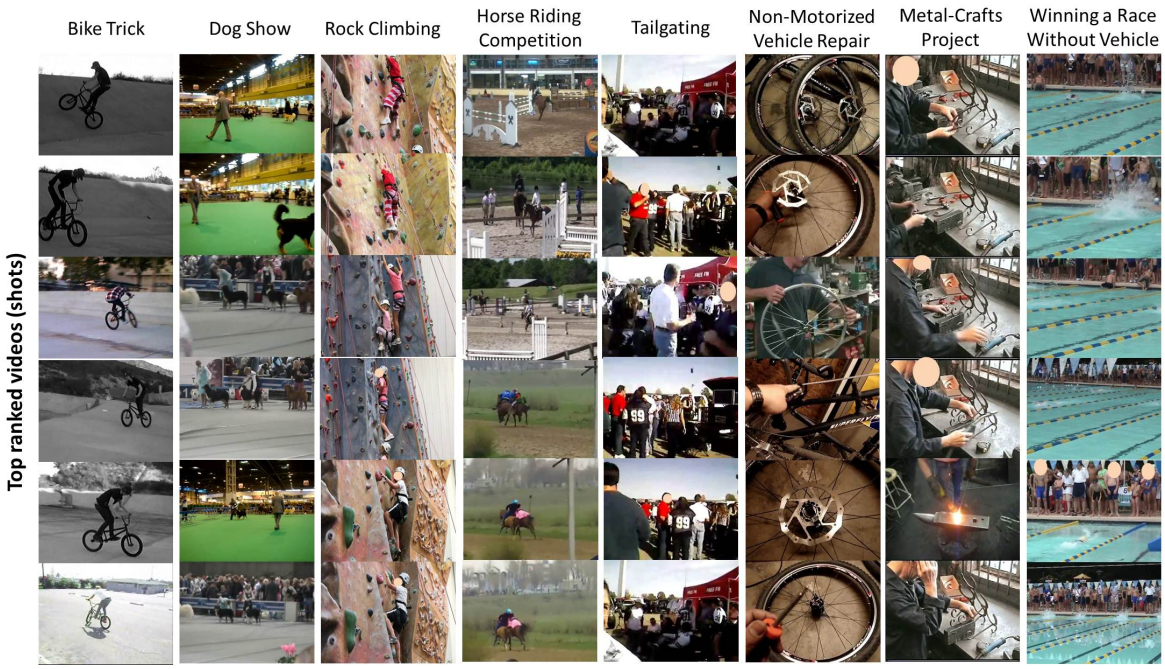


Figure 6: Qualitative results of retrieval by DMSY + HVSF 5X + 1R system on Video1M set. Due to space limitations only selected queries are shown.

Timing (ms)	First Pass			1 Iteration of Reranking				Total
	SVM training	Convert Dual Sol. to Primal	Prediction	Load Top Ranked Video Data	SVM training	Convert Dual Sol. to Primal	Prediction	
MEDTEST14 10Ex	136.1 ± 24.6	87.9 ± 9.1	17.3 ± 1.3	4.9 ± 4.8	136 ± 33.5	87.5 ± 13	10.4 ± 0.7	507.7 ± 73.1
Video1M 10Ex	128.9 ± 21.3	83 ± 5	380.3 ± 5.7	20.6 ± 9.8	127.7 ± 25.3	85.1 ± 5.7	104.9 ± 17.1	975.3 ± 63.4

Table 5: Timing breakdown for DMSY + HVSF 5X + 1R system for one MEDTEST14 10Ex and Video1M 10Ex query.

4. CONCLUSIONS

We have proposed a system which can search 1 million videos with 1 core in less than 1 second while retaining 80% of the performance of a state-of-the-art CBVS system. This potentially opens the door to content-based video search on web-scale video repositories. The speed gains in our system mainly come from three aspects. First, we perform approximations in locations which will not cause significant change in performance, such as increasing the PQ compression factor and only computing the dot-products for PQ subspaces with higher variance. Second, fast reranking can perform some error correction and compensate for some of the performance loss caused by approximations. Finally, our proposed system relies on 3 semantics-based features, which enabled us to significantly lower the amount of bytes required to represent each video. This may suggest that for the next challenge: searching 1 billion videos, even more compact semantics-based representations might be a promising solution.

Acknowledgments

This work was partially supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Interior National Business Center contract number D11PC20068. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. Disclaimer: The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoI/NBC, or the U.S. Government. This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by National Science Foundation grant number OCI-1053575. Specifically, it used the Blacklight system at the Pittsburgh Supercomputing Center (PSC).

5. REFERENCES

[1] A. Bordes, S. Ertekin, J. Weston, and L. Bottou. Fast kernel classifiers with online and active learning. In *JMLR*, 2005.

[2] E. F. Can and R. Manmatha. Modeling concept dependencies for event detection. In *ICMR*, 2014.

[3] C.-C. Chang and C.-J. Lin. LIBSVM: A library for support vector machines. In *ACM Transactions on Intelligent Systems and Technology*, 2011.

[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In *BMVC*, 2011.

[5] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20-th annual symposium on Computational geometry*, 2004.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[7] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. In *JMLR*, 2008.

[8] A. Habibian, T. Mensink, and C. G. M. Snoek. Videostory: A new multimedia embedding for few-example recognition and translation of events. In *ACM MM*, 2014.

[9] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. In *IEEE TPAMI*, 2011.

[10] L. Jiang, A. G. Hauptmann, and G. Xiang. Leveraging high-level and low-level features for multimedia event detection. In *ACM MM*, 2012.

[11] L. Jiang, D. Meng, T. Mitamura, and A. G. Hauptmann. Easy samples first: Self-paced reranking for zero-example multimedia search. In *ACM MM*, 2014.

[12] L. Jiang, D. Meng, S.-I. Yu, Z. Lan, S. Shan, and A. G. Hauptmann. Self-paced learning with diversity. In *NIPS*, 2014.

[13] T. Joachims. Training linear svms in linear time. In *KDD*, 2006.

[14] V. Kantorov and I. Laptev. Efficient feature extraction, encoding and classification for action recognition. In *CVPR*, 2014.

[15] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[16] Z.-Z. Lan, Y. Yang, N. Ballas, S.-I. Yu, and A. Hauptmann. Resource constrained multimedia event detection. In *MultiMedia Modeling*, 2014.

[17] M. Mazloom, E. Gavves, K. van de Sande, and C. Snoek. Searching informative concept banks for video event detection. In *ICMR*, 2013.

[18] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *Proceedings of the International Conference on Computer Vision Theory and Applications*, 2009.

[19] P. Natarajan, S. Wu, S. Vitaladevuni, X. Zhuang, S. Tsakalidis, U. Park, and R. Prasad. Multimodal feature fusion for robust event detection in web videos. In *CVPR*, 2012.

[20] P. Over, G. Awad, M. Michel, J. Fiscus, G. Sanders, W. Kraaij, A. F. Smeaton, and G. Quénot. TRECVID 2014 – an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *TRECVID*, 2014.

[21] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *NIPS*, 2014.

[22] M. Sjöberg, M. Koskela, S. Ishikawa, and J. Laaksonen. Large-scale visual concept detection with explicit kernel maps and power mean svm. In *ICMR*, 2013.

[23] A. Tamrakar, S. Ali, Q. Yu, J. Liu, O. Javed, A. Divakaran, H. Cheng, and H. Sawhney. Evaluation of low-level features and their combinations for complex event detection in open source videos. In *CVPR*, 2012.

[24] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. The new data and new challenges in multimedia research. *arXiv preprint arXiv:1503.01817*, 2015.

[25] W. Tong, Y. Yang, L. Jiang, S.-I. Yu, Z. Lan, Z. Ma, W. Sze, E. Younessian, and A. Hauptmann. E-lamp: integration of innovative ideas for multimedia event detection. In *Machine Vision and Applications*, 2014.

[26] A. Vedaldi and A. Zisserman. Efficient additive kernels via explicit feature maps. In *IEEE TPAMI*, 2012.

[27] H. Wang and C. Schmid. Action recognition with improved trajectories. In *ICCV*, 2013.

[28] Z. Xu, Y. Yang, and A. G. Hauptmann. A discriminative CNN video representation for event detection. In *CVPR*, 2015.

[29] Y. Yang, Z. Ma, Z. Xu, S. Yan, and A. G. Hauptmann. How related exemplars help complex event detection in web videos? In *ICCV*, 2013.

[30] S.-I. Yu, L. Jiang, Z. Xu, Z. Lan, et al. Informedia@ TRECVID 2014 MED and MER. In *TRECVID Video Retrieval Evaluation Workshops*, 2014.