Kobe University and Muroran Institute of Technology at TRECVID 2012 Semantic Indexing Task

Kimiaki Shirahama
Muroran Institute of Technology
27-1, Mizumoto-cho, Muroran, 050-8585, Japan
shirahama@mmm.muroran-it.ac.jp

Kuniaki Uehara
Graduate School of System Informatics, Kobe University
1-1, Rokkodai, Nada, Kobe 657-8501, Japan
uehara@kobe-u.ac.jp

***Abstract*—This paper describes our method developed for TRECVID 2012 Semantic INdexing (SIN) Task. Our main research purpose is the development of a fast method, which can work on a single processor with no performance degradation. To this end, computationally expensive processes are re-formulated based on *matrix operation*. We re-formulate the Euclidian distance computation for the kernel value computation in an SVM, and the probability density computation of multivariate normal distributions for the GMM supervector representation. This enables accurate concept detection using a large number of training examples, and spatially-temporally dense features.**

The following four runs were submitted to SIN (light) task:

- *L_A_kobe_muro_l5_4:* **This is our baseline run using five features, 1. SIFT at Harris-Affine regions, 2. SIFT at Hessian-Affine regions, 3. Trajectory displacement, 4. HOG around trajectories, and 5. MFCC. Five SVMs built on these features are fused using the weighted linear combination approach. This run achieved the MAP** 0.320**.**
- *L_A_kobe_muro_l6_1:* **In addition to the above five features, this run uses the sixth feature,** *Spatio-Temporal-Dense RGB SIFT* **(***STD-RGB-SIFT***), consisting of SIFT descriptors sampled at every sixth pixel in every other frame. The extraction of this feature becomes feasible because of the significant speedup of the probability density computation. This run achieved the MAP** 0.348**.**
- *L_A_kobe_muro_l18_3:* **To cover the diversity of a concept's appearances, this run utilizes** *bagging* **where three SVMs are built on each of six features using different subsets of training examples. Such many SVMs can be built due to the fast kernel value computation. SVMs are fused using the weighted linear combination. This run achieved the MAP** 0.358**, which is the highest score among all the runs submitted to SIN (light) task.**
- *L_A_kobe_muro_r18_2:* **This run uses** *rough set theory* **to fuse SVMs in** *L_A_kobe_muro_l18_3*, **and achieved the MAP** 0.323**.**

The above results indicate the effectiveness of the spatially-temporally dense feature *STD-RGB-SIFT*. **In particular, the MAP** 0.302 **was achieved only using** *STD-RGB-SIFT*. **This is significantly higher than MAPs using the other single features. Also, the effectiveness of bagging can be seen from the above results.**

I. Introduction

Through our past participation in TRECVID experiments, we noticed that accurate concept detection requires both a large number of training examples and fine-grained features in both the spatial and temporal dimensions. Many training examples is necessary for covering the diversity of a concept's appearances. Naphade *et al.* reported that even though a concept has its own difficulty of detection, the detection performance is generally proportional to the logarithm of the number of available positive examples [1].

Regarding spatially fine-grained features, better performance can be achieved by sampling a larger number of spatial regions (and representing them with some region descriptor such as SIFT) [2]. The reason is that although several region detectors like Harris-Laplace and Difference-of-Gaussian (DoG) have been proposed, detected regions are useful for localizing the same instance of an object in different images, but are not necessarily useful for characterizing different instances of an object. Thus, rather than region detectors, exhaustively sampling many regions yields better performance. Regarding temporally fine-grained features, features extracted from multiple frames in an example yield much better performance than features extracted only from one keyframe [3], [4]. Features from multiple frames can characterize a concept's appearance, which may change due to object movement and camera motion.

We investigated the above issues on TRECVID 2011 SIN (light) task. The result is shown in Fig. 1 where each bar represents the MAP for 23 evaluated concepts. Three bars in the left side are obtained using SIFT descriptors at regions, detected by Harris-Laplace region detector. On the other hand, three bars in the right side are obtained using SIFT descriptors at regions, uniformly sampled at every sixth pixel in a video frame. For each side, the leftmost bar represents the MAP only using 750 positive examples for a concept, while the middle bar represents the MAP using all the available positive examples. This indicates the importance of a large number of training examples. In addition, the middle bar uses SIFT descriptors extracted only from one video frame in each example, the rightmost bar uses SIFT descriptors extracted from multiple video frames (one frame per second). This validates the effectiveness of temporally dense features. Furthermore, the comparison between the left and right sides indicates the effectiveness of spatially dense features.

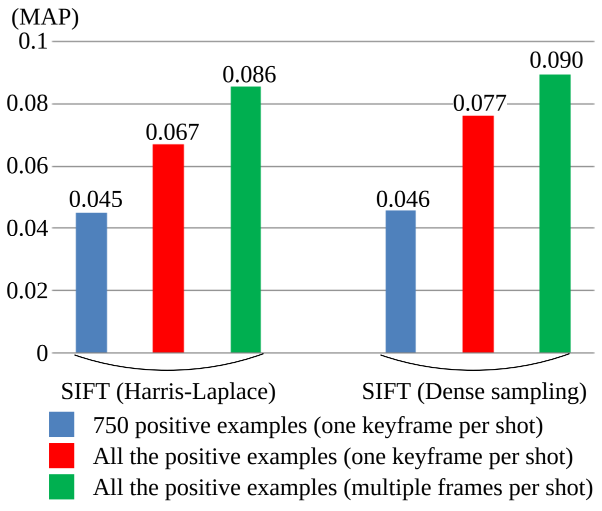

Figure 1. Result of our preliminary experiment about numbers of positive examples, and sampling approaches in the spatial and temporal dimensions.

However, processing many training examples and spatially-temporally dense features incurs intensive computational costs. In the former case, one most computationally-expensive process is the distance (dissimilarity) computation. This is necessary for the kernel value computation in an SVM, which is the most standard classification method for concept detection. Although many existing methods adopt computationally-inexpensive distance measures like histogram intersection [3] and inner product [5], the performance is generally degraded. When using spatially-temporally dense features, much computational cost is required to compute 'contributions' of numerous descriptors to the representation of an example. In the case of the bag-of-visual-words model, when one million descriptors are sampled from an example, it is needed to find the most similar visual word to each descriptor. Many existing methods restrict regions or video frames from which descriptors are sampled. But, this degrades the performance as indicated in Fig. 1.

We consider that one reason for the above intensive computational costs is to process examples or descriptors one by one. In this paper, we introduce two methods which utilize matrix operation to process a large number of examples or descriptors in batch. These methods do not conduct any approximation, but compute the exact solution in a much faster time than processing examples or descriptors one by one. The first method reformulates the Euclidian distance computation to compute distances among many examples in batch. This enables SVM training and testing to be about 37 times faster than computing distances one by one. The second method reformulates the probability density computation of many descriptors for multiple multivariate normal distributions. Thereby, we can efficiently compute the GMM supervector representation of an example for a spatially-temporally dense feature. This yields significant performance improvement.

It should be noted that several works address the parallelization of feature extraction and concept detection processes using GPU [6] and multiple processors [7]. However, although the parallelization speeds up a procedure, it occupies hardware resources. In other words, without abundant hardware resources (*i.e.*, many processors or many GPUs), it is impossible to simultaneously run multiple processes of the parallelized procedure, for instance, simultaneously extracting features from different examples, and simultaneously conducting the detection of multiple concepts. In addition, if the speedup by the parallelization is equal to the number of processors (*e.g.*, 10 times speedup with 10 processors) [7], this speedup is not so meaningful. The reason is that the same level of speedup can be easily obtained by simultaneously running multiple processes of the not-parallelized procedure. Thus, in this paper, we concentrate on the speedup of the procedure on a single processor.

II. Matrix Formulation

This section describes matrix formulations for the Euclidian distance computation and the probability density computation of multivariate normal distributions. These are used in our SIN method.

A. Euclidian Distance

Suppose that one example e is represented as a D-dimensional vector, that is, $e = [e^1, e^2, \cdots, e^D]$. The Euclidian distance between two examples $e_1 = [e_1^1, \cdots, e_1^D]$ and $e_2 = [e_2^1, \cdots, e_2^D]$ is computed as follows:

$$dist(e_1, e_2) = \sum_{d=1}^{D} (e_1^d - e_2^d)^2 \quad (1)$$

The naive approach for computing Euclidian distance among N examples, is to use the triple-nested loop. Two outer loops are used to set a pair of examples, and the most inner loop is used to compute equation (1). However, this is very slow when computing Euclidian distances among a large number of high-dimensional examples. Our SIN method requires to compute Euclidian distances among $30,000$ training examples, each of which is represented as a $16,384$-dimensional vector.

To speed up the Euclidian distance computation, we expand equation (1) as follows:

$$dist(e_1, e_2) = \sum_{d=1}^{D} (e_1^d)^2 - 2\sum_{d=1}^{D} e_1^d e_2^d + \sum_{d=1}^{D} (e_2^d)^2 \quad (2)$$

In the above equation, the Euclidian distance computation is divided into three parts, 1. sum of the squared value in each dimension of e_1, 2. inner product between e_1 and e_2, and 3. sum of the squared value in each dimension of e_2.

Based on the above expansion, we re-formulate the computation of Euclidian distances between a set of M examples and a set of N examples. Suppose the former and latter sets are represented by the $D \times M$ matrix EM and the $D \times N$ matrix EN. The $M \times N$ matrix, $D(EM, EN)$, which represents Euclidian distances between examples in EM and examples in EN, can be computed as follows:

$$
\begin{aligned}
& D(EM, EN) \\
&= \begin{bmatrix} dist(em_1, en_1) & \cdots & dist(em_1, en_N) \\ \vdots & & \vdots \\ dist(em_M, en_1) & \cdots & dist(em_M, en_N) \end{bmatrix} \\
&= \begin{bmatrix} \sum_{d=1}^{D}(em_1^d)^2 & \cdots & \sum_{d=1}^{D}(em_1^d)^2 \\ \vdots & & \vdots \\ \sum_{d=1}^{D}(em_M^d)^2 & \cdots & \sum_{d=1}^{D}(em_M^d)^2 \end{bmatrix} \\
&-2 \begin{bmatrix} em_1^1 & \cdots & em_1^D \\ \vdots & & \vdots \\ em_M^1 & \cdots & em_M^D \end{bmatrix} \begin{bmatrix} en_1^1 & \cdots & en_N^1 \\ \vdots & & \vdots \\ en_1^D & \cdots & en_N^D \end{bmatrix} \\
&+ \begin{bmatrix} \sum_{d=1}^{D}(en_1^d)^2 & \cdots & \sum_{d=1}^{D}(en_N^d)^2 \\ \vdots & & \vdots \\ \sum_{d=1}^{D}(en_1^d)^2 & \cdots & \sum_{d=1}^{D}(en_N^d)^2 \end{bmatrix} \qquad (3)
\end{aligned}
$$

where em_i $(1 \leq i \leq M)$ and en_j $(1 \leq j \leq N)$ represent the i-th example in EM and the j-th example in EN, respectively. Values of their d-th dimension are denoted by em_i^d and en_j^d. The first matrix in equation (3) is constructed by the following three steps: 1. take the square of each element in EM, 2. compute the row-wise summation of these squared values, and 3. create N copies of the 'transposed' row-wise summation. These steps can be easily implemented using general matrix manipulation libraries like MATLAB. The third matrix in equation (3) is constructed in a similar way to the first matrix, where the only difference is to create M copies of the row-wise summation in the third step. Finally, the second term in equation (3) is the inner product between EM and EN.

We present a preliminary experimental result showing the effectiveness of the above Euclidian distance computation. The second row in table I shows the computation time of the naive approach based on equation (1), and the third row shows the computation time of the batch approach based on matrix operation in equation (3). Note that Euclidian distances computed by both approaches are the same. The second and third columns in Table I represent the computation times required for computing Euclidian distances among $1,000$ and $5,000$ examples respectively, where each example is represented as a 16384-dimensional vector. As shown in Table I, the batch approach is surprisingly faster than the naive approach, and will be utilized in SVM training and testing in section III-C.

Table I
COMPARISON IN COMPUTATIONAL TIMES BETWEEN THE NAIVE AND BATCH APPROACHES.

	$1,000$ examples	$5,000$ examples
Naive	200 (sec)	$5,027$ (sec)
Batch	0.5 (sec)	9.7 (sec)

B. Multivariate Normal Distribution

We extend the batch Euclidian distance computation to the batch probability density computation of multivariate normal distributions. A multivariate normal distribution $g_k(x)$ is defined as follows (k is the index to deal with multiple multivariate normal distributions later):

$$g_k(x) = N(x|\mu_k, \Sigma_k) = \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} e^{-\frac{1}{2}(x-\mu_k)\Sigma_k^{-1}(x-\mu_k)}, \qquad (4)$$

where x is a D-dimensional vector representing a descriptor (*e.g.*, a SIFT descriptor is a 128-dimensional vector). μ_k and Σ_k are the mean vector and the covariance matrix of $g_k(x)$, respectively. We assume that Σ_k is a diagonal matrix where D dimensions are independent from each other. This can be easily satisfied by applying in advance Principle Component Analysis (PCA) to descriptors [8], [9]. Under the above independence condition, $g_k(x)$ is simplified as follows:

$$g_k(x) = \frac{1}{\sqrt{(2\pi)^D \prod_{d=1}^{D} \sigma_k^d}} e^{-\frac{1}{2}\sum_{d=1}^{D} \frac{(x^d - \mu_k^d)^2}{\sigma_k^d}}, \qquad (5)$$

where x^d, μ_k^d and σ_k^d are respectively values of x, mean vector and variance vector (diagonal covariance matrix) in the d-th dimension $(1 \leq d \leq D)$. It should be noted that the exponential part (*i.e.*, $expo_k = \sum(x^d - \mu_k^d)^2/\sigma_k^d$) can be considered as a Euclidian distance, where the distance in each dimension is weighted by the inverse of the variance. Just like Euclidian distance, $expo_k$ can be rewritten as:

$$expo_k = \sum_{d=1}^{D} \frac{(x^d)^2}{\sigma_k^d} - 2\sum_{d=1}^{D} \frac{x^d \mu_k^d}{\sigma_k^d} + \sum_{d=1}^{D} \left(\frac{\mu_k^d}{\sqrt{\sigma_k^d}} \right)^2 \qquad (6)$$

Based on equation (6), we now consider the batch computation where exponential parts ($expo_k$) for K multivariate normal distributions are computed for a set of N descriptors. The input for this computation is the following three matrices, the $N \times D$ matrix X (set of N descriptors), the $K \times D$ matrix (set of K mean vectors) and the $K \times D$ matrix (set of K variance vectors). The batch computation outputs the $N \times K$ matrix $EXPO$, where each element at the i-th row and k-th column represents the exponential part $expo_{ik}$ of the i-th descriptor for the k-th multivariate normal distribution. The batch computation is conducted by the following matrix operation:

$$EXPO = \begin{bmatrix} expo_{11} & \cdots & expo_{1K} \\ \vdots & & \vdots \\ expo_{N1} & \cdots & expo_{NK} \end{bmatrix} \tag{7}$$

$$= \begin{bmatrix} \left(x_1^1\right)^2 & \cdots & \left(x_1^D\right)^2 \\ \vdots & & \vdots \\ \left(x_N^1\right)^2 & \cdots & \left(x_N^D\right)^2 \end{bmatrix} \begin{bmatrix} 1/\sigma_1^1 & \cdots & 1/\sigma_K^1 \\ \vdots & & \vdots \\ 1/\sigma_1^D & \cdots & 1/\sigma_K^D \end{bmatrix}$$
$$-2 \begin{bmatrix} x_1^1 & \cdots & x_1^D \\ \vdots & & \vdots \\ x_N^1 & \cdots & x_N^D \end{bmatrix} \begin{bmatrix} \mu_1^1/\sigma_1^1 & \cdots & \mu_K^1/\sigma_K^1 \\ \vdots & & \vdots \\ \mu_1^D/\sigma_1^D & \cdots & \mu_K^D/\sigma_K^D \end{bmatrix}$$
$$+ \begin{bmatrix} \sum_{d=1}^D \left(\mu_1^d/\sqrt{\sigma_1^d}\right)^2 & \cdots & \sum_{d=1}^D \left(\mu_K^d/\sqrt{\sigma_K^d}\right)^2 \\ \vdots & & \vdots \\ \sum_{d=1}^D \left(\mu_1^d/\sqrt{\sigma_1^d}\right)^2 & \cdots & \sum_{d=1}^D \left(\mu_K^d/\sqrt{\sigma_K^d}\right)^2 \end{bmatrix} \tag{8}$$

Each term in the above equation is an extension of the corresponding term in equation (6), in order to process N descriptors and K multivariate normal distributions in batch. It is clear that equation (8) can be easily computed using general matrix manipulation libraries. Once $EXPO$ is computed, probabilistic densities of N descriptors for K multivariate normal distributions are computed by the following three steps: 1. multiply each element in $EXPO$ by $-\frac{1}{2}$, 2. take the exponential of each element in the multiplied $EXPO$ (the resulting $N \times K$ matrix is denoted by $EXPO'$), and 3. multiply the k-th column in $EXPO'$ by the constant values of the k-th multivariate normal distribution (*i.e.*, $1/\sqrt{(2\pi)^D \prod \sigma_k^d}$). The above batch processing is used to extract the GMM supervector representation of an example in section III-B.

III. Our SIN Method

Fig. 2 illustrates an overview of our SIN method. Our method is based on the method developed by Inoue *et al.* [8], [9]. First, given training examples for a concept, the method represents them using image, motion and audio features. It should be noted that, in this paper, a feature means a set of descriptors sampled from an example. In accordance with this, each example is represented using the GMM supervector representation which characterizes the distribution of descriptors. Then, for each feature, an SVM is constructed to distinguish test examples with the concept's presence from the other test examples. SVMs for all the features are fused into a classifier, in order to account characteristics of different features. For each test example, the fused classifier outputs an 'evaluation score' representing the likelihood of the concept's presence. Finally, $2,000$ test examples with the highest evaluation scores are returned as a detection result.

A. Features

The following six features are used in our SIN method:
1. SIFT-Har: A Scale-Invariant Feature Transform (SIFT) descriptor represents the edge shape of a local image region (patch), and is useful for characterizing characterizing a local shape of an object. *SIFT-Har* is defined as a set of 128-dimensional SIFT descriptors at regions, detected by applying *Harris-Affine region detector* [10] to every other frame in an example.
2. SIFT-Hes: This feature is defined as a set of SIFT descriptors at regions, detected by applying *Hessian-Affine region detector* to every other frame in an example [10]. The difference between *SIFT-Har* and *SIFT-Hes* is only in region detectors. This causes that compared to *SIFT-Har*, *SIFT-Hes* is more useful for characterizing a blob and ridge of an object.
3. Traj-Disp: Dense trajectories are extracted by tracking densely sampled points with optical flow fields [11]. Here, points are sampled at every fifth pixel in each video frame in an example. *Traj-Disp* is defined as a set of *displacement* descriptors, each is a 30-dimensional vector representing a sequence of displacements of a tracked point. This feature is useful for characterizing the shape of a local motion.
4. Traj-HOG: Traj-HOG is defined as a set of 96-dimensional vector HOG descriptors, each of which represents edges around a dense trajectory. Thus, *Traj-HOG* is useful for characterizing the appearance of a moving object in an example [11].
5. MFCC: Mel-Frequency Cepstral Coefficient (MFCC) represents the short-term power spectrum of an audio signal, using frequency bands which are defined based on the human auditory system's responses [12]. Based on [9], an *MFCC* descriptor in our SIN method consists of original *MFCC* coefficients, their deviation ($\Delta MFCC$), their acceleration ($\Delta\Delta MFCC$), and the deviation and acceleration of the audio energy (ΔE and $\Delta\Delta E$). In total, the *MFCC* descriptor is a 38-dimensional vector.
6. STD-RGB-SIFT: An *RGB-SIFT* descriptor is the combination of SIFT descriptors, each of which is independently extracted from one of RGB channels [13]. This is robust to changes in illumination, such as light intensity change and light color change. *STD-RGB-SIFT* is defined as a set of *RGB-SIFT* descriptors, sampled at every sixth pixel in every other frame in an example.

For each feature, PCA is applied to descriptors in order to reduce the computational complexity, and make dimensions independent from each other [8]. For each of features other than *Traj-Disp*, descriptors are projected into 32-dimensional vectors. Descriptors for *Traj-Disp* are projected into 30-dimensional vectors. Here, although the number of dimensions of an original displacement descriptor is the same to that of the projected one, the latter consists of mutually independent dimensions.

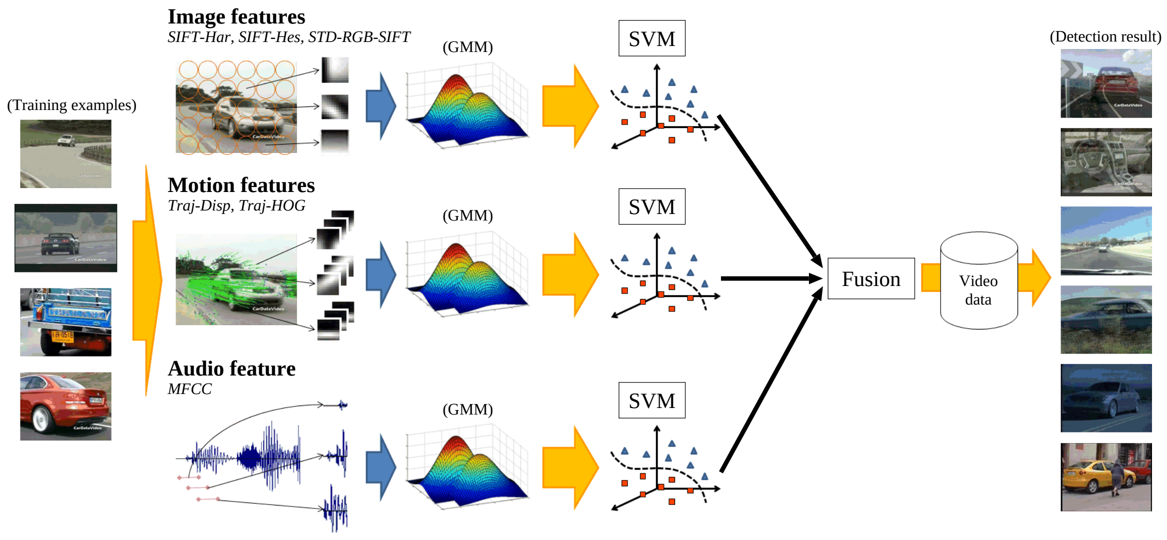

Figure 2. An overview of our SIN method.

B. GMM Supervector Representation

Based on [8], [9], for each feature, each example is represented by the *GMM supervector* which characterizes the distribution of descriptors in the example. The traditional Bag-of-Visual-Words (BoVW) model represents the distribution of descriptors using the pre-specified template (*i.e.*, visual words). Compared to this, the GMM supervector representation is more flexible where a GMM representing the distribution of descriptors is adaptively estimated for the example. The effectiveness of the GMM supervector representation is validated in [8], [9].

A GMM (Gaussian Mixture Model) is defined as the following probability density function:

$$p(x|\theta) = \sum_{k=1}^{K} w_k g_k(x), \tag{9}$$

where x is a descriptor, $g_k(x)$ is the k-th multivariate normal distribution (mixture component), and w_k is its mixture weight. The GMM is defined as the set of parameters, $\theta = \{w_k, \mu_k, \Sigma_k\}_{k=1}^{K}$, for each of K multivariate normal distributions.

Since enough descriptors for estimating GMM parameters cannot be obtained for an example with a short duration, these parameters are estimated using *Maximum A Posteriori* (MAP) adaptation [8], [9]. MAP adaptation consists of the following two steps:

1. Universal Background Model (UBM) construction: A UBM is a GMM which represents the distribution of descriptors in the general case, and serves as the prior distribution. In our implementation, for each feature, a UBM with $K = 512$ multivariate normal distributions is constructed using 1.5 million descriptors, randomly sampled from training examples. This UBM construction is conducted using Hidden Markov Model Toolkit (HTK) [12].

2. MAP adaptation: Using a maximum posteriori approach, parameters of the UBM are adapted so as to match the distribution of descriptors in an example. Let x_i be the i-th descriptor in the example ($1 \leq i \leq N$). Based on [8], [9], only the mean vector of each multivariate normal distribution is adapted as follows:

$$\hat{\mu_k} = \frac{\tau \bar{\mu_k} + \sum_{i=1}^{N} c_{ik} x_i}{\tau + \sum_{i=1}^{N} c_{ik}}, \text{ where } c_{ik} = \frac{\bar{w}_k \bar{g}_k(x_i)}{\sum_{k=1}^{K} \bar{w}_k \bar{g}_k(x_i)} \tag{10}$$

where the UBM is defined as the set of parameters $\bar{\theta} = \{\bar{w}_k, \bar{\mu}_k, \bar{\Sigma}_k\}_{k=1}^{K}$ (the k-th multivariate normal distribution is $\bar{g}_k$), and τ ($= 20$) is the pre-specified parameter. c_{ik} represents the 'occupation' probability of a descriptor x_i being associated with $\bar{g}_k$. Also, c_{ik} can be considered as the contribution of x_i to the adapted mean vector $\hat{\mu_k}$. Intuitively, if c_{ik} is small, the effect of x_i on $\hat{\mu_k}$ is small. Overall, equation (10) indicates that if many descriptors do not match with $\bar{g}_k$ (*i.e.*, their contributions are small), $\hat{\mu_k}$ remains close to the mean vector $\bar{\mu_k}$ of the UBM. In contrast, if many descriptors match with $\bar{g}_k$, $\hat{\mu_k}$ moves toward their mean.

In MAP adaptation, the biggest computational burden is to compute probability densities of each descriptor x_i for K multivariate normal distributions ($\bar{g}_k$). For example, when the UBM has $K = 512$ multivariate normal distributions and one million descriptors are sampled from an example, the number of probability density computations is 512 millions. Thus, we utilize the batch computation described in section II-B. Considering the memory limitation, for $K = 512$ multivariate normal distributions, the batch computation is conducted every $100,000$ descriptors. It is straightforward to incorporate this divisional computation into MAP adaptation, where summations in equation (10) (*i.e.*, $\sum c_{ik} x_i$ in the numerator and $\sum c_{ik}$ in the denominator) can be separately computed for every set of $100,000$ descriptors, and separate computation results are finally summed up.

Table II shows the efficiency of MAP adaptation using our batch computation. Each computational time in Table II includes the time of PCA for projecting SIFT descriptors into lower-dimensional descriptors, and the time of MAP adaptation using those projected descriptors. As shown in Table II, even for about 2.8 million descriptors, PCA and MAP adaptation can complete in about 76 seconds. This efficient MAP adaptation enables us to deal with descriptors which are sampled densely in both the spatial and temporal dimensions, like *RGB-SIFT* descriptors in *SDT-RGB-SIFT*.

Table II
EXAMPLES OF COMPUTATION TIMES REQUIRED FOR PCA AND MAP ADAPTATION USING OUR BATCH COMPUTATION.

# of descriptors	$95,411$	$589,251$	$2,785,939$
Computational time	3.8 (sec)	17.4 (sec)	75.9 (sec)

After MAP adaptation for an example ex, its GMM supervector representation $\phi(ex)$ is constructed as follows [8], [9]:

$$\phi(ex) = \begin{bmatrix} \tilde{\mu_1} \\ \vdots \\ \tilde{\mu_K} \end{bmatrix}, \text{ where } \tilde{\mu_k} = \sqrt{\bar{w}_k}(\bar{\Sigma}_k)^{-\frac{1}{2}}\hat{\mu}_k \quad (11)$$

Let be $\phi(ex_1)$ and $\phi(ex_2)$ as GMM supervectors for two examples ex_1 and ex_2, respectively. The GMM supervector representation is designed in such a way that, the Euclidian distance between $\phi(ex_1)$ and $\phi(ex_2)$ becomes the sum of squared Mahalanobis distances between mean vectors adopted for ex_1 and ex_2 (each squared distance is weighted by the corresponding mixture weight).

C. SVM Training and Test

For a concept, given training examples represented by GMM supervectors, we build an SVM which distinguishes test examples where the concept is present from the rest of test examples. In particular, the following RBF kernel is adopted in the SVM [14]:

$$\begin{aligned} K(\phi(ex_1), \phi(ex_2)) &= \exp(-\gamma ||\phi(ex_1) - \phi(ex_2)||^2) \quad (12) \\ &= \exp\big(-\gamma \; dist(\phi(ex_1), \phi(ex_2))\big), \end{aligned}$$

where γ is a pre-specified parameter (in our case, γ is set to the inverse of the average Euclidian distance among training examples). It should be noted that, once kernel values between each pair of examples are computed, SVM training and testing are relatively fast using general SVM solvers like "precomputed kernel in Libsvm library" [14]. As shown in equation (12), the exponential part of RBF kernel includes the Euclidian distance computation. Thus, the batch computation described in II-A is utilized to compute in batch kernel values among training examples (multiplying distances by γ and taking their exponentials are clear).

In our implementation of SVM training, considering the memory limitation, we divide training examples into sets of $5,000$ examples, and conduct the batch computation of kernel values for each set pair. In SVM testing, we repeat the batch computation of kernel values between $5,000$ test examples and $10,000$ training examples. Regarding the computational time, when $30,000$ training examples and $137,327$ test examples (in IACC.1.B) are used (each example is represented by $16,384$-dimensional GMM supervector for *SIFT-Har*), the above SVM training and testing complete in about two hours. Compared to this, SVM training and testing with the naive kernel value computation take more than one week. We will report the precise computational time of SVM training and testing at the workshop. Finally, we use 'probabilistic outputs' of an SVM, where each test example is assigned a probability of the concept's presence based on its distance to the decision boundary of the SVM [15]. Such probabilistic outputs are used in the fusion of SVMs on different features.

D. Fusion

In order to improve the concept detection performance, we fuse SVM testing results on different features. Our fusion approaches are classified into two types, *weighted linear combination* and *rough set theory*. Below, we present these approaches by associating them with our submitted runs.

1) Weighted Linear Combination: For a test example ex, this approach computes the fusion score $FS(ex)$ based on the following weighted linear combinations of SVM probabilistic outputs on different features [9]:

$$Score(ex) = \sum_{f \in F} \alpha_f \cdot prob_f(ex) \quad (13)$$

where F in $L\_A\_kobe\_muro\_l5\_4$ is the set of five features *1. SIFT-Har*, *2. SIFT-Hes*, *3. Traj-Disp*, *4. Traj-HOG* and *5. MFCC*, while F in $L\_A\_kobe\_muro\_l6\_1$ is the set of six features, including the above five features and *6. STD-RGB-SIFT*. In equation (13), $\alpha_f \geq 0$ is the weight for the feature f in F where $\sum_{f \in F} \alpha_f = 1$. In addition, $prob_f(ex)$ is the SVM probabilistic output for ex on f. Intuitively, when the SVM on f achieves more accurate concept detection, its weight α_f should become larger.

To obtain such a set of weights, we build an SVM on each feature using a half of training examples, and compute the set of weights using the other half of training examples. We employ *gradient-ascend* approach where the Average Precision (AP) is used to compute the gradient vector, which indicates the direction for improving the AP with the current set of weights. Note that SVMs for $prob_f(ex)$ are build using all the training examples. Building of these SVMs is separated from building of SVMs used in the above weight computation. In other words, we assume that weights obtained for SVMs built with a half of training examples are applicable for SVMs built with all the training examples. Although this weight computation has much room to be improved, our preliminary experiment showed that it can work in most cases, where the weighted fusion of SVMs on different features outperforms SVMs on single features.

$L\_A\_kobe\_muro\_l18\_3$ fuses the total 18 SVMs, where three SVMs are built on each of six features using different subsets of training examples. These subsets are constructed by randomly selecting three-quarter of training examples. This kind of fusion approach is called *bagging*. Our preliminary experiment showed that even when an SVM is built using a large number of training examples, its performance changes depending on used training examples. For this result, we consider that since a concept's appearances are significantly different depending on varied camera techniques and environments, one SVM on each feature is not enough for covering these diverse appearances of the concept. Thus, we adopt the above bagging approach.

To fuse 18 SVMs in $L\_A\_kobe\_muro\_l18\_3$, we use the following simple approach: The weight α_f obtained for the SVM on the feature f in $L\_A\_kobe\_muro\_l6\_1$ is divided by three, and this divided weight is used for three SVMs on f in $L\_A\_kobe\_muro\_l18\_3$. Since α_f in $L\_A\_kobe\_muro\_l6\_1$ can be considered as the effectiveness score of f, we consider that it can be equally applicable for three SVMs on f in $L\_A\_kobe\_muro\_l18\_3$.

2) Rough Set Theory: $L\_A\_kobe\_muro\_r18\_2$ uses *Rough Set Theory* (RST) to fuse the same set of 18 SVMs as $L\_A\_kobe\_muro\_l18\_3$. RST is a set-theoretic classification method for extracting 'rough' descriptions of a class from imprecise (or noisy) data [16]. Using RST, we aim to extract multiple classification rules which characterize different subsets of positive examples, so that test examples with diverse appearances of a concept can be covered.

Concept detection using RST is summarized as follows (see [17] for more detail): For each pair of the i-th positive example p_i ($1 \leq i \leq M$) and the j-th negative example n_j ($1 \leq j \leq N$), we first determine a set of SVMs, $SVM_{i,j}$, where each SVM can correctly classify p_i and n_j as positive and negative, respectively. In other words, p_i can be discriminated from n_j when at least one SVM in $SVM_{i,j}$ is used. To obtain $SVM_{i,j}$, positive and negative examples are ranked based on their probabilistic outputs of an SVM. If p_i and n_j are respectively ranked in and out of the top M-th position, they are regarded to be correctly classified by the SVM. Next, we compute the discernibility function df_i that represents sets of SVMs, required to discriminate p_i from all negative examples. This is achieved by using at least one SVM in $SVM_{i,j}$ for all negative examples. That is, df_i is computed by taking the disjunction of SVMs in $SVM_{i,j}$, and then taking the conjunction of such disjunctions for all negative examples:

$$df_i = \wedge\{\vee SVM_{i,j} |\ 1 \leq j \leq N\}, \tag{14}$$

df_i is simplified into the minimal disjunctive normal form, where each conjunctive term, called *reduct*, represents a minimal set of SVMs required to discriminate p_i from all negative examples. Such a reduct forms a rule: A test example is regarded to match the rule, if all SVMs in the reduct classify it as positive. Finally, the set of $2,000$ test examples which match the largest numbers of rules, is returned as the detection result.

IV. Experimental Results

We submitted the following four runs:
1. L_A_kobe_muro_l5_4: This is our baseline run where one SVM is built on each of *SIFT-Har*, *SIFT-Hes*, *Traj-Disp*, *Traj-HOG* and *MFCC*. SVMs are fused based on the weighted linear combination.
2. L_A_kobe_muro_l6_1: In addition to SVMs on five features in *L_A_kobe_muro_l5_4*, this run builds an SVM on *STD-RGB-SIFT*. These SVMs are fused based on the weighted linear combination.
3. L_A_kobe_muro_l18_3: This run adopts the bagging approach where three SVMs are built on each of six features. The total 18 SVMs are fused using the weighted linear combination.
4. L_A_kobe_muro_r18_2: This run fuses 18 SVMs in *L_A_kobe_muro_l18_3* using RST.

For all the above runs, an SVM is built using positive examples collected by the collaborative annotation effort [18]. On the other hand, negative examples are collected as shots which are randomly selected from training videos. Since negative examples in the collaborative annotation effort are collected based on the active learning approach, their distribution is biased toward the decision boundary of an SVM. Our preliminary experiment showed that rather than such biased negative examples, randomly selected negative examples lead to more accurate performance. In addition, the number of negative examples is determined in such a way that the total number of positive and negative examples becomes $30,000$[1]. Finally, we use Matlab engine [19] to call the batch computations based on matrix operation in C++ programs.

Fig. 3 shows the ranking of all runs submitted to TRECVID 2012 SIN (light) task, where each bar represents the MAP of one run. As can be seen from Fig. 3, *L_A_kobe_muro_l18_3* achieved the highest MAP (0.358) among all the submitted 91 runs. The other runs are

[1]An SVM in the bagging approach is built using the following training examples: three-quarter of positive examples are randomly selected, and negative examples are then randomly selected, so that the total number of positive and negative examples becomes $30,000$.

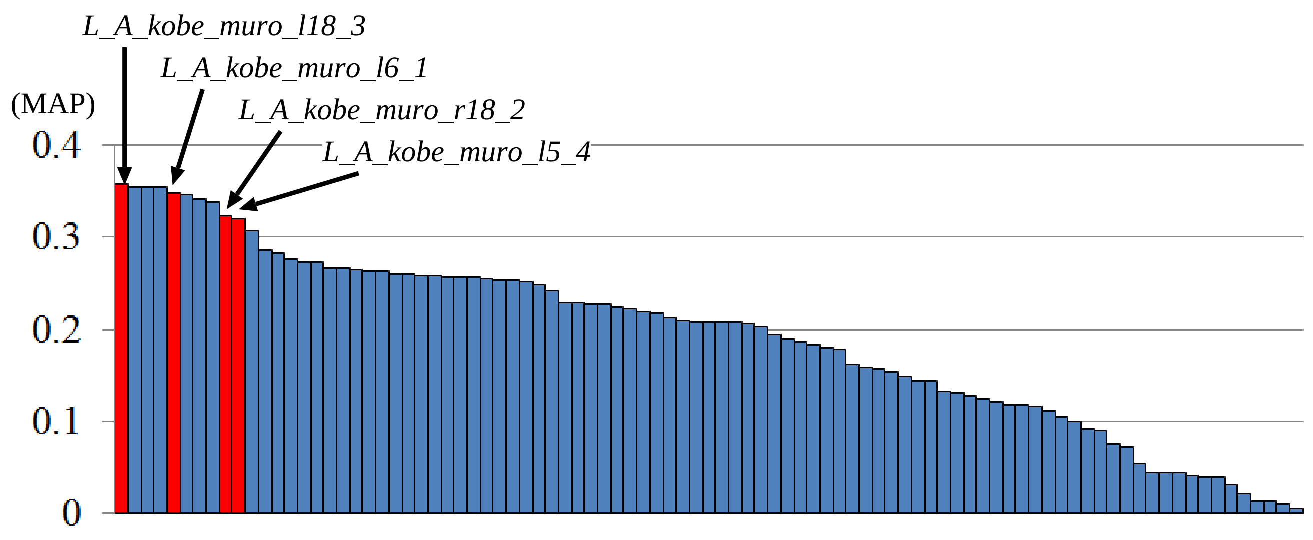

Figure 3. Ranking of all runs submitted to TRECVID 2012 SIN (light) task.

also ranked at top positions. Below, we closely investigate our submitted runs. First, the comparison between the MAP of *L_A_kobe_muro_l6_1* (0.348) and that of *L_A_kobe_muro_l5_4* (0.320), indicates the effectiveness of *STD-RGB-SIFT* (about 9% improvement). With respect to this, Fig. 4 shows the MAP achieved by the SVM built on each of six features in *L_A_kobe_muro_l6_1*. As shown in Fig. 4, the SVM on *STD-RGB-SIFT* significantly outperforms SVMs on the other single features. This means that spatially-temporally dense features are very useful for accurate concept detection. In addition, it may be interesting that the second highest MAP in Fig. 4 is achieved by the SVM on *Traj-HOG*, and is significantly larger than the third and fourth MAPs by SVMs on *SIFT-Har* and *SIFT-Hes*. Recall that *Traj-HOG* is a set of HOG descriptors around trajectories, obtained by tracking 'densely' sampled points in an example. On the other hand, *SIFT-Har* and *SIFT-Hes* are defined by SIFT descriptors at regions, detected by Harris-Affine and Hessian-Affine region detectors, respectively. The high MAP score on *Traj-HOG* may also indicate the effectiveness of spatially-temporally dense features.

In Fig. 3, the comparison between *L_A_kobe_muro_l18_3* and *L_A_kobe_muro_l6_1* indicates the effectiveness of the bagging approach, although the improvement is relatively small (about 3% improvement). One main reason is the simplicity of our current fusion method for *L_A_kobe_muro_l18_3*, as described in section III-D. A further improvement can be achieved when using a more sophisticated fusion method. Regarding *L_A_kobe_muro_l18_3*, our main argument is not the fusion method, but is fast SVM training and testing based on the batch computation of kernel values, so that multiple SVMs can be built on each feature.

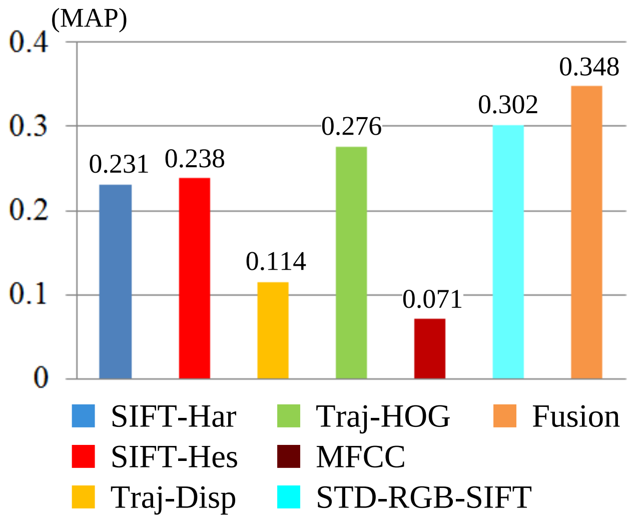

Figure 4. Performance comparison among SVMs on single features in *L_A_kobe_muro_l6_1*.

Finally, the MAP of *L_A_kobe_muro_r18_2* (0.323) is smaller than *L_A_kobe_muro_l18_3*. This indicates the ineffectiveness of RST. Specifically, many rules extracted by RST are not very useful, so several test examples where a concept is clearly absent are included in the detection result. Thus, we plan to improve the current classification method where each test example is classified using majority voting of matched rules. One promising approach is weighted majority voting where the weight of a rule is determined by the cross-validation.

V. Conclusion and Future Works

In this paper, we introduced our SIN method which takes advantage of matrix operation for the batch computation of Euclidian distances among many examples, and the batch computation of probabilistic densities of many descriptors for multiple multivariate normal distributions. The former enables fast SVM training and testing, and the latter enables MAP adaptation using a huge number of descriptors, sampled densely in both the spatial and temporal dimensions (*i.e.*, spatially-temporally dense features). Owing to these, the run *L_A_kobe_muro_l18_3* achieved the highest MAP among all the 91 runs submitted to TRECVID 2012 SIN (light) task.

Although MAP adaptation became fast based on the batch computation of probability densities, descriptor sampling (extraction) is currently very slow. Because of this, even using 50 processors, it took about one month to extract *STD-RGB-SIFT*'s GMM supervectors of all examples. Actually, we planned to incorporate another spatially-temporally dense feature based on *Opponent SIFT* descriptors [13] into our SIN method. However, this could not finish until the submission deadline. Thus, our important future work is to develop a method that can efficiently conduct spatially-temporally dense sampling of descriptors. To this end, we plan to hash regions in video frames using *locality sensitive hashing* [20]. Thereby, for a region which is very similar to a hashed region, descriptor computation can be skipped. In other words, the descriptor of the region can be approximated as the descriptor of the hashed region, with a very small approximation error.

References

[1] Naphade M. and Smith J., "On the detection of semantic concepts at trecvid," in *Proc. of ACM Multimedia 2004*, 2004, pp. 660–667.

[2] Nowak E., Jurie F. and Triggs B., "Sampling strategies for bag-of-features image classification," in *Proc. of ECCV 2006*, 2010, pp. 490–503.

[3] Snoek C. et al., "The mediamill trecvid 2010 semantic video search engine," in *Proc. of TRECVID 2010*, 2010.

[4] Snoek C., Worring M., Geusebroek J., Koelma D. and Seinstra F., "On the surplus value of semantic video analysis beyond the key frame," in *Proc. of ICME 2005*, 2010, pp. 386–389.

[5] Söberg M., Ishikawa S., Koskela M., Laaksonen J. and Oja E., "Picsom experiments in trecvid 2011," in *Proc. of TRECVID 2011*, 2011.

[6] Sande K., Gevers T. and Snoek C., "Empowering visual categorization with the gpu," *IEEE Transactions on Multimedia*, vol. 13, no. 1, pp. 60–70, 2011.

[7] Chu C. et al., "Map-reduce for machine learning on multicore," in *Proc. of NIPS 2006*, 2006, pp. 281–288.

[8] Inoue N. and Shinoda K., "A Fast MAP Adaptation Technique for GMM-Supervector-based Video Semantic Indexing Systems," in *Proc. of ACM Multimedia 2011*, 2011, pp. 1357–1360.

[9] Inoue N., Kamishima Y., Wada T., Shinoda K. and Sato S., "TokyoTech+Canon at TRECVID 2011," in *Proc. of TRECVID 2011*, 2011.

[10] Mikolajczyk K. et al., "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, no. 1-2, pp. 43–72, 2005.

[11] Wang H., Klä A., Schmid C. and Cheng-Lin L., "Action Recognition by Dense Trajectories," in *Proc. of CVPR 2011*, 2011, pp. 3169–3176.

[12] Young S. et al., *The HTK Book (for HTK Version 3.4)*. Cambridge University Engineering Department, 2009. [Online]. Available: http://htk.eng.cam.ac.uk/

[13] Sande K., Gevers T. and Snoek C., "Evaluating color descriptors for object and scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1582–1596, 2010.

[14] Chang C. and Lin C., "Libsvm : A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 27, pp. 1–27, 2011.

[15] Liu H., Lin C. and Weng R., "A note on platt's probabilistic outputs for support vector machines," *Machine Learning*, vol. 68, no. 3, pp. 267–276, 2007.

[16] Komorowski J., Øhrn A. and Skowron A., "The rosetta rough set software system," in *Handbook of Data Mining and Knowledge Discovery, W. Klösgen and J. Zytkow (eds.)*. Oxford University Press, 2002, ch. D.2.3.

[17] Shirahama K., Matsuoka Y. and Uehara K., "Event retrieval in video archives using rough set theory and partially supervised learning," *Multimedia Tools and Applications*, vol. 57, no. 1, pp. 59–75, 2012.

[18] Ayache S. and Quénot G., "Video corpus annotation using active learning," in *Proc. of ECIR 2008*, 2008, pp. 187–198.

[19] Mathworks, *Call MATLAB Engine*, http://www.mathworks.com/help/matlab/calling-matlab-engine-from-c-c-and-fortran-programs.html.

[20] Datar M., Immorlica N., Indyk P. and Mirrokni V., "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. of SCG 2004*, 2004, pp. 253–262.