Empowering Visual Categorization with the GPU

Koen E. A. van de Sande, *Student Member, IEEE,* Theo Gevers, *Member, IEEE,*
and Cees G. M. Snoek, *Member, IEEE*

***Abstract*—Visual categorization is important to manage large collections of digital images and video, where textual meta-data is often incomplete or simply unavailable. The bag-of-words model has become the most powerful method for visual categorization of images and video. Despite its high accuracy, a severe drawback of this model is its high computational cost. As the trend to increase computational power in newer CPU and GPU architectures is to increase their level of parallelism, exploiting this parallelism becomes an important direction to handle the computational cost of the bag-of-words approach. When optimizing a system based on the bag-of-words approach, the goal is to minimize the time it takes to process batches of images. Additionally, we also consider power usage as an evaluation metric.**

In this paper, we analyze the bag-of-words model for visual categorization in terms of computational cost and identify two major bottlenecks: the quantization step and the classification step. We address these two bottlenecks by proposing two efficient algorithms for quantization and classification by exploiting the GPU hardware and the CUDA parallel programming model. The algorithms are designed to (1) keep categorization accuracy intact, (2) decompose the problem and (3) give the same numerical results.

In the experiments on large scale datasets it is shown that, by using a parallel implementation on the Geforce GTX260 GPU, classifying unseen images is 4.8 times faster than a quad-core CPU version on the Core i7 920, while giving the exact same numerical results. In addition, we show how the algorithms can be generalized to other applications, such as text retrieval and video retrieval. Moreover, when the obtained speedup is used to process extra video frames in a video retrieval benchmark, the accuracy of visual categorization is improved by 29%.

I. Introduction

Visual categorization aims to determine whether objects or scene types are visually present in images or video segments. This is a useful prerequisite to manage large collections of digital images and video, where textual meta-data is often incomplete or simply unavailable [1]. Letting humans annotate such meta-data is expensive and infeasible for large datasets. While automatic visual categorization is not yet as accurate as a human annotation, it is a useful tool to manage large collections. The bag-of-words model [2] has become the most powerful method today for visual categorization [3–11]. The bag-of-words model computes image descriptors at specific points in the image. These descriptors are then quantized against a codebook of prototypical descriptors to obtain a fixed-length representation of an image. Although the bag-of-words model is a powerful mechanism for accurate visual categorization, a severe drawback is its high computational cost. Current state-of-the-art in visual categorization benchmarks such as TRECVID 2009 [12] require weeks of compute time on compute clusters to process 380 hours of video. However, even with weeks of compute time, most systems are still only able to process a limited subset of about 250,000 frames. In the future, more and more data needs to be processed as datasets continue to grow. To address the problem of computation, the two directions are *faster approximate methods* and *larger compute clusters*. Faster to compute descriptors (such as SURF [13, 14]) and indexing mechanisms (tree-based codebooks [15, 16]) have been developed. Another direction is to use large compute clusters with many CPUs [10, 11, 17] to solve the computational problem using brute force. However, both directions have their drawbacks. Faster methods will (1) suffer from reduced accuracy when they resort to increasingly coarse approximations and (2) suffer from increased complexity in the form of additional parameters and thresholds to control the approximations, all of which need to be hand-tuned. Brute force solutions based on compute clusters have the problem that (1) compute clusters are available in limited supply and (2) due to the complexities of resource scheduling and the large (network) communication overheads found in large distributed compute clusters, they are difficult to use efficiently.

Recently, another direction for acceleration has opened up: *computing on consumer graphics hardware*. Cornelis and Van Gool [18] have implemented SURF on the GPU (Graphics Processing Unit) and obtained an order of magnitude speedup compared to a CPU implementation. These GPU implementations [18, 19] build on the trend of increased parallelism. In recent years, the most important method for higher computational power in both CPUs and GPUs has been to increase parallelism: the number of processing units is increased, instead of the speed of the processing units. GPUs have been evolving faster than CPUs, with transistor counts doubling every few months. Whereas commodity CPUs currently have up to 4 cores, commodity GPUs have up to 30 cores at their disposal [20]. Together, the increased programmability and computational power of GPUs provides ample opportunities for acceleration of algorithms which can be parallelized [21]. However, note that the parallelization of an algorithm can be applied to CPU implementations as well. CPU implementations should be multi-threaded and SIMD-optimized to allow for a fair comparison to optimized GPU versions [22–24]. Compared to faster approximate methods, algorithms for the GPU do not need to approximate for speedups, if they are able to exploit the parallel nature of the GPU. Compared to compute clusters, the main advantages of the GPU are their wide availability and their potential to be

Manuscript received March 1, 2010; revised June 4, 2010.

The authors are with the Intelligent Systems Lab Amsterdam, University of Amsterdam, Science Park 904, 1098 XH Amsterdam, The Netherlands (e-mail: ksande@uva.nl).

more energy-efficient.

When optimizing a system based on the bag-of-words model, the goal is to minimize the time it takes to process batches of images. Individual components of the bag-of-words model, such as the point sampling strategy, descriptor computation and SVM model training, have been independently studied on the GPU before [18, 25, 26]. These studies accelerate specific algorithms with the GPU. However, it remains unclear whether those algorithms are the real bottlenecks in accurate visual categorization with the bag-of-words model. In our overview of related work on visual categorization with the GPU, we observe that quantization and classification have remained CPU-bound so far, despite being computationally very expensive.

Therefore, in this paper, the goal is to combine GPU hardware and a parallel programming model to accelerate the quantization and classification components of a visual categorization architecture. Two algorithms are proposed to accelerate these two components. We identify the following requirements to these algorithms:

1) The algorithms and their implementations should push the state-of-the-art in categorization **accuracy**.
2) Visual categorization must be **decomposable** into components to locate bottlenecks.
3) Given the same input, implementations of a component on various hardware architectures must give the **same output**[1].

Requirement 1 states that we are pursuing algorithms and implementations which will push the state-of-the-art in categorization accuracy, and therefore require high computational throughput. Requirement 2 implies that visual categorization can be decomposed into several steps, and the computational bottlenecks are located in specific parts. Requirement 3 allows CPU and GPU versions of the same visual categorization component to be interchanged in the system, because both versions will give the same output. Therefore, keeping the rest of the system the same, time measurements can be performed on these individual components.

Our contributions are (1) an analysis of the bottlenecks in accurate visual categorization systems and, to address these bottlenecks, (2) two GPU-accelerated algorithms, GPU vector quantization and GPU kernel value precomputation, which results in a substantial acceleration of the complete visual categorization pipeline.

This paper is organized as follows. In section II, an efficiency analysis of visual categorization based on the bag-of-words model is made. In section III, the GPU architecture and the GPU-accelerated versions of quantization and classification are discussed. In section IV, the experimental setup used to evaluate the accelerations is presented. In section V, results are shown and analyzed. In section VI, applications of the speedups in this paper besides visual categorization are discussed. Finally, in section VII, we conclude with an overview of the benefits of GPU acceleration for visual categorization.

[1]For practical purposes, small numeric deviations (less than 10^{-7}) in the output of a component are considered to be the same. We have verified that these deviations have not changed the accuracy of the complete visual categorization system.

II. Overview of Visual Categorization

The aim of this paper is to speed up state-of-the-art visual categorization systems using GPUs. In visual categorization [27], the visual presence of an object or scene of specified type is determined. In Figure 1, an overview of the components of a visual categorization system is shown. A trained visual categorization system takes an image as input and returns the likelihood that one or more visual categories are present in the image. Visual categorization systems break down into a number of common steps:

- *Image Feature Extraction*, which takes an image as input and outputs a fixed-length feature vector representing the image.
- *Category Model Learning*, learns one model per visual category by taking all vector representations of images from the train set and the category labels associated with those images.
- *Test Image Classification*, which takes vector representations of images from the test set and applies the visual category models to these images. The output of this step is a likelihood score for each image and each visual category.

A. Image Feature Extraction

Visual categorization systems which achieve state-of-the-art results on the PASCAL VOC benchmarks [4, 5, 7] use the bag-of-words model [2] as the underlying representation model. This model first extracts specific points in an image using a point sampling strategy. Over the area around these points, descriptors are computed which represent the local area. The bag-of-words model performs vector quantization of the descriptors in an image against a visual codebook. A descriptor is assigned to the codebook element which is closest in Euclidean space. Figure 1 gives an overview of the steps for the bag-of-words model in the image feature extraction blocks. In Table I, the computation times of different steps within the bag-of-words model are listed. For every step, multiple options are available. Next, we will discuss these options, their presence in related work and their computation times on the CPU and GPU.

1) Point Sampling Strategy: As a point sampling strategy, there are two commonly used techniques in state-of-the-art systems [5, 7]: dense sampling and salient point methods. Dense sampling samples points regularly over the image at fixed pixel intervals. As it does not depend on the image contents, it is a trivial operation to perform. Typically, around 10,000 points are sampled per image. Two examples of salient point methods are the Harris-Laplace salient point detector [29] and the Difference-of-Gaussians detector [28]. See Table I for computation times of these point sampling strategies. The Harris-Laplace detector uses the Harris corner detector to find scale-invariant interest points. It then selects a subset of these points for which the Laplacian-of-Gaussians reaches a maximum over scale. Using recursive Gaussian filters [30], the computation of Gaussian derivatives at multiple scale required for these steps is possible at a rate of multiple images per second: computational complexity of recursive

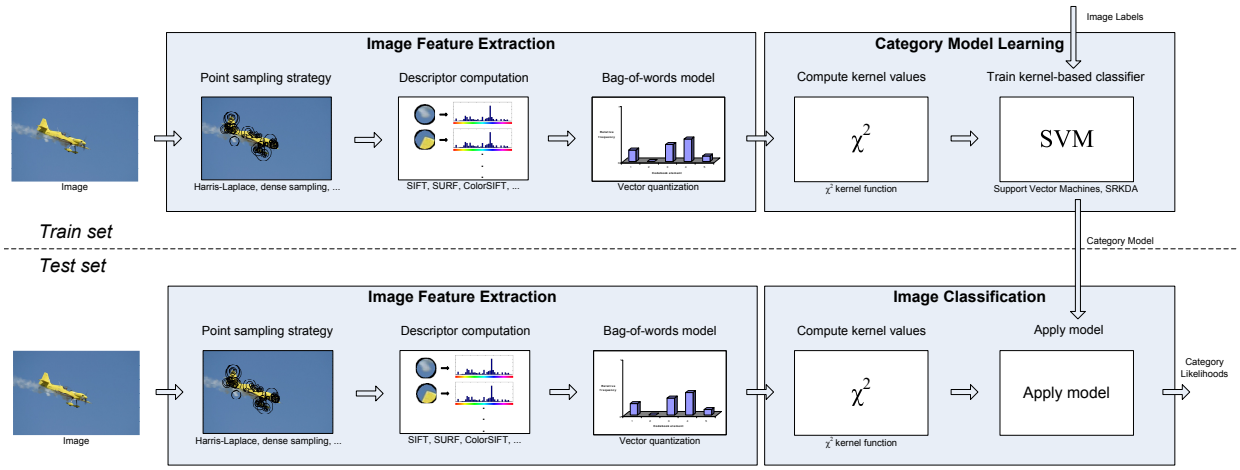

Fig. 1. The components of a state-of-the-art visual categorization system. For all images in both the train set and the test set, visual features are extracted in a number of steps. First, a point sampling method is applied to the image. Then, for every point a descriptor is computed over the area around the point. All the descriptors of an image are subsequently vector quantized against a codebook of prototypical descriptors. This results in a fixed-length feature vector representing the image. Next, the visual categorization system is trained based on the feature vectors of all training images and their category labels. To learn kernel-based classifiers, similarities between training images are needed. These similarities are computed using a kernel function. To apply a trained model to test images, the kernel function values are also needed. Given these values between a test image and the images in the train set, the category models are applied and category likelihoods are obtained.

TABLE I
IMAGE FEATURE EXTRACTION TIMINGS. COMPUTATION TIMES OF DIFFERENT STEPS WITHIN THE BAG-OF-WORDS MODEL WITH A SINGLE CPU CORE, FOUR CPU CORES AND ON THE GPU. FOR EVERY STEP, MULTIPLE CHOICES ARE AVAILABLE. CPU TIMES OBTAINED ON AMD OPTERON 250. GPU TIMES OBTAINED FROM THE LITERATURE. ONE OF THE CONTRIBUTIONS OF THIS PAPER IS SUBSTANTIALLY ACCELERATING THE VECTOR QUANTIZATION STEP USING THE GPU.

Image Feature Extraction	**Times (s)**				
	CPU (1 thread)	*CPU (4 threads)*		*GPU*	
1) Point Sampling Strategy					
• Dense Sampling	< 0.01	< 0.01		< 0.01	
• Difference-of-Gaussians	1.4	0.4	[28]	< 0.1	[18]
• Harris-Laplace	4.4	1.2	[29]	< 0.2	[30]
2) Descriptors					
• SIFT	1.4	0.4	[28]	< 0.1	[19]
• SURF	< 1.0	< 0.2	[13]	< 0.01	[18]
• ColorSIFT	4.0	1.3	[5]	< 0.3	[19]
3) Bag-of-Words					
• Tree-based Codebook	< 0.5	< 0.2	[15, 16]	< 0.01	[25]
• Vector Quantization	**4.1**	**1.1**	[2]	$<$ **0.2**	**this paper**

Gaussian filters is independent of the scale. As has been shown by Cornelis and Van Gool [18], running the Difference-of-Gaussians detector is possible in real-time, using a scale-space pyramid to limit computational complexity.

2) Descriptor Computation: To describe the area around the sampled points, the SIFT descriptor [28] and the SURF descriptor [13] are the most popular choices. Sinha *et al.* [19] compute SIFT descriptors at 10 frames per second for 640x480 images. Cornelis and Van Gool [18] compute SURF descriptors at 100 frames per second for 640x480 images. Both of these papers show that descriptor computation runs with excellent performance on the GPU, because one thread can be assigned per pixel or per descriptor, and thereby performing operations in parallel. The standard SIFT descriptor has a length of 128. Following Everingham *et al.* [4], color extensions of SIFT [5] would form a reasonable state-of-the-art baseline for future VOC challenges, due to their increased classification accuracy. ColorSIFT increases the descriptor length to 384 and the required computation time is also tripled.

3) Bag-of-Words: Vector quantization is computationally the most expensive part of the bag-of-words model. With n descriptors of length d in an image, the quantization against a codebook with m elements requires the full $(n \times m)$ distance matrix between all descriptors and codebook elements. For values which are common for visual categorization, $n = 10,000$, $d = 128$ and codebook size $m = 4,000$, a CPU implementation takes approximately 5 seconds per image, as the complexity is $O(ndm)$ per image. When d increases to 384, as is the case for ColorSIFT, the CPU implementation slows down to more than 10 seconds per image, which makes this a computational bottleneck.

One approach to address this bottleneck is to index using a tree-based codebook structure [14–16], instead of a standard codebook. A tree-based codebook replaces the comparison of each descriptor with all m codebook elements by a comparison against $\log(m)$ codebook elements. As a result, algorithmic complexity is reduced to $O(nd\log(m))$. Tree-based methods have been shown to run in real-time on the GPU [25]. However, for a tree-based codebook generally the accuracy is lower [14], especially for high-dimensional descriptors such as ColorSIFT. Therefore, tree-based codebooks conflict with our first requirement: it does not keep accuracy intact. The same argument applies to other indexing structures such as miniBOF (mini bag-of-features) [31]: accuracy is sacrificed in return for faster computation. Another drawback of tree-based codebooks and miniBOFs is that soft assignment [6, 32], *e.g.* , assigning weight to more than just the closest codebook element, requires the full distance matrix instead of only the closest elements. This soft assignment improves the classification accuracy for visual categorization by more than 5% on state-of-the-art systems [32]. Ruling out such

an important performance improvement again conflicts with requirement 1. Therefore, this paper studies how to accelerate the vector quantization step using normal codebooks on the GPU, as the same accelerations are then also applicable to soft assignment.

In conclusion, in a state-of-the-art setup of the bag-of-words model, the most expensive part is the vector quantization step. Approximate methods are unable to satisfy our requirement to maintain accuracy.

B. Category Model Learning

To learn visual category models, supervised kernel-based learning algorithms such as Support Vector Machines (SVM) and Spectral Regression Kernel Discriminant Analysis [33] have shown good results [3, 5]. Key property of a kernel-based classifier is that it does not require the actual vector representation of the feature vector $\vec{F}$, but only a kernel function $k(\vec{F}, \vec{F'})$ which is related to the distance between the feature vectors. This is sometimes referred to as the 'kernel trick'. It has been shown experimentally [3] that the non-linear χ^2 kernel function is the best choice [5, 7] for accurate visual categorization.

When tuning the parameters of the classifier, the values of the kernel function are needed for every parameter setting. While typical implementations compute the values of this kernel function on-the-fly and only keep a cache of the most recent evaluations, it is more efficient to compute all values in advance and store them, because then the values can be re-used for every parameter setting and for every visual category. The total number of kernel values to be computed in advance is the number of pair-wise distances between all training images, *e.g.* , it is quadratic with respect to the number of images. The benefit of precomputing kernel values is illustrated in Table II.

The kernel-based SVM algorithm has been ported to the GPU by [26, 35]. In [35], specific optimizations are made in the GPU version such that only linear kernel functions are supported. For visual categorization, however, support for the more accurate non-linear χ^2 kernel function is needed to meet requirement 1. Catanzaro *et al*. [26] perform a selection of the training samples under consideration for SVM, resulting in a speedup of up to 35 times for training models. Further speedups are possible if this GPU-SVM implementation is combined with the precomputation of kernel values. The precomputation of kernel values itself has not been investigated yet. Therefore, in section III-C, we propose an algorithm to precompute the kernel values and investigate the speedup possibilities offered by precomputing these values.

Table II gives an overview of computation times on the PASCAL VOC 2008 dataset for different feature vector lengths, where the learning of visual category models is split into a precomputation of kernel values and the actual model learning. Because the ground truth labels of all images and their extracted features are needed before training can start, it is an inherently offline process. When multiple features are used, more than 90% of computation time is spent on precomputing the kernel values. This makes it the most expensive step in category model learning.

In conclusion, the learning of category models can be split into two steps, kernel value computation and classifier training. The classifier training has been accelerated with the GPU before, but the kernel value computation is the most expensive step. This paper will study how to accelerate the computation of the kernel values on the GPU.

C. Test Image Classification

To classify images from a test set, feature extraction first has to be applied to the images, similar to the train set. Therefore, speed-ups obtained in the image feature extraction stage are useful for both the train set and the test set. To apply the visual category models, pair-wise kernel values between the feature vectors of the train set and those of the test set are needed. The same precomputation strategy used in the learning stage is applicable here. When accelerating the computation of kernel values, this speedup will apply to both the training phase and the test phase of a visual categorization system. Timings in Table II illustrate that when processing images from the test set, again, the computation of kernel values takes up the most time.

In conclusion, the speedups obtained using GPU vector quantization and GPU precomputation of kernel values also directly apply to the classification of images/frames from the test set.

III. GPU Accelerated Categorization

We first discuss parallel programming with the GPU and the CPU (section III-A). Next, we discuss the GPU-accelerated versions of vector quantization (section III-B) and kernel value precomputation (section III-C). Both of these visual categorization steps take large numbers of vectors as input, and therefore are ideally suited for the data parallelism offered by the GPU.

A. Parallel Programming on the GPU and CPU

Over the years, there have been different approaches to programming generic algorithms on GPUs. Initially, algorithms needed to be formulated in terms of graphics primitives such as textures and vertices and written in specialized shader languages before they could run on the GPU. Through the availability of C-like parallel programming models such as CUDA [36] and OpenCL [37], the programmability of GPUs has increased. Since CUDA has the most mature software stack available at this moment, we use CUDA. The CUDA parallel programming model is explained in [38]. It is designed for writing scalable parallel code that runs across tens of thousands of concurrent threads and dozens of processor cores. Because the physical parallelism of current GPUs ranges up to 30 processor cores and over 30,000 threads, this is an essential property. The parallel models allows a programmer to write parallel programs that transparently and efficiently scale with this level of parallelism.

The model is also applicable to multicore CPUs, as has been shown for CUDA by Stratton *et al*. [39] and Diamos *et al*. [40, 41]. However, the code generated by their approaches is

TABLE II
COMPUTATION TIMES OF THE DIFFERENT STEPS IN VISUAL CATEGORIZATION. THE TIMES LISTED ARE FOR AN IMAGE DATASET (PASCAL VOC 2008), WHICH HAS A TRAINING SET OF SIZE 4332 AND TEST SET OF SIZE 4133. CLASSIFICATION TIMES ARE TOTALS FOR ALL 20 VISUAL CATEGORIES. CPU TIMES OBTAINED ON AMD OPTERON 250. THIS PAPER SUBSTANTIALLY ACCELERATES THE PRECOMPUTATION OF KERNEL VALUES (SHOWN IN BOLD) USING THE GPU.

Category Model Learning	**Times (s)**				
	CPU (1 thread)	*CPU (4 threads)*		*GPU*	
Category Model Learning (without precomputed)					
Parameter Tuning (length $\vec{F} = 4,000$)	$> 1,000,000$	$> 250,000$	[34]	$> 10,000$	[26]
Train Classifier (length $\vec{F} = 4,000$)	$> 100,000$	$> 25,000$	[34]	$> 1,000$	[26]
Category Model Learning (with precomputed)					
Precompute Kernel Values (length $\vec{F} = 4,000$)	**430**	**110**		**10**	**this paper**
Precompute Kernel Values (length $\vec{F} = 32,000$)	**3,400**	**900**		**64**	**this paper**
Precompute Kernel Values (length $\vec{F} = 320,000$)	**34,000**	**9,000**		**650**	**this paper**
Parameter Tuning	1,050	260	[34]	60	[26]
Train Classifier	240	60	[34]	10	[26]
Test Image Classification (with precomputed)					
Precompute Kernel Values (length $\vec{F} = 4,000$)	**430**	**110**		**10**	**this paper**
Apply Classifier	< 5	< 2	[34]	< 1	[26]

not yet as efficient as hand-written CPU code. On the CPU, programs can be parallellized by running multiple threads on different cores and by using SIMD instructions. SIMD instructions perform the same operation on multiple data elements at the same time, effectively allowing 2 to 4 floating point instructions to be executed at the same time on a single core. For additional information see [42]. Internally, the GPU uses SIMD as well: each of the 30 cores in the GTX275 can execute 8 floating point instructions at the same time [36].

B. Algorithm 1: GPU-Accelerated Vector Quantization

In section II-A, we have shown that vector quantization is computationally the most expensive step in image feature extraction. Therefore, in this section, the GPU implementation of vector quantization for an image with n descriptors against a codebook of m elements is proposed. The descriptor length is d. Quantization against a codebook requires the full $(n \times m)$ distance matrix between all descriptors and codebook elements. A descriptor is then assigned to the column which has the lowest distance in a row. By counting the number of minima occurring in each column, the vector quantized representation of the image is obtained. To be robust against changes in the number of descriptors in an image, these counts are divided by the number of descriptors n for the final feature vector.

The most expensive computational step in vector quantization is the calculation of the distance matrix. Typically, the Euclidean distance is employed:

$$||\vec{a}-\vec{b}|| = \sqrt{(a_1-b_1)^2 + (a_2-b_2)^2 + ... + (a_q-b_q)^2}. \quad (1)$$

This formula for the Euclidean distance can be directly implemented on the GPU using loops [43]. However, such a naive implementation is not very efficient, because the same result is obtained with fewer operations by simply vectorizing the Euclidean distance, which is a common trick [26]:

$$||\vec{a}-\vec{b}|| = \sqrt{||\vec{a}||^2 + ||\vec{b}||^2 - 2\vec{a}\cdot\vec{b}}. \quad (2)$$

The advantage of the vector form of the Euclidean distance is that it allows us to decompose the computation of a distance matrix between sets of vectors into several smaller steps which are faster to compute. In Algorithm 1, pseudo code is given for vector quantization using simple vectorization of the Euclidean distance. In the algorithm, A is the matrix with all image descriptors as rows, *e.g.* , a $(n \times d)$ matrix, B is the matrix with all codebook elements as rows, *e.g.* , a $(m \times d)$ matrix, $\vec{a_i}$ is the i^{th} row of A and $\vec{b_i}$ is the i^{th} row of B.

Algorithm 1 Vector Quantization with Simple Vectorized Euclidean Distance

1: **for** $i = 1$ to n **do**
2: lengthsA[i] $\leftarrow ||\vec{a_i}||^2$ {$\vec{a_i}$ *is the* i^{th} *row of* A}
3: **end for**
4: **for** $j = 1$ to m **do**
5: lengthsB[j] $\leftarrow ||\vec{b_j}||^2$ {$\vec{b_j}$ *is the* j^{th} *row of* B}
6: **end for**
7: $M \leftarrow$ MatrixMultiply(A, MatrixTranspose(B))
8: **for** $i = 1$ to n **do**
9: minDist $\leftarrow \infty$
10: lengthA $\leftarrow$ lengthsA[i]
11: **for** $j = 1$ to m **do**
12: $d \leftarrow$ lengthA + lengthsB[j]$-2M_{ij}$
13: **if** $d <$ minDist **then** minDist $\leftarrow d$, best $\leftarrow j$
14: **end for**
15: assignTo[i] $\leftarrow$ best
16: **end for**
17: **return** assignTo

We identify the following steps within Algorithm 1:

1) **Compute the squared vector lengths $||\vec{a}||^2$ for every row of A and $||\vec{b}||^2$ for every row of B (line 1-6).** We assign one GPU thread per vector and do a serial sum within each thread. To avoid numerical deviations due to the summing of many numbers with single precision floating point operations, we use Kahan summation [44].

Transposing the matrices A and B allows for faster (aligned) memory access. The CUDA SDK [45] contains an efficient implementation of matrix transpose for arbitrarily sized matrices. Transposing rectangular matrices on the GPU is faster than the CPU, because the GPU has a higher memory bandwidth.

2) **Compute the dot products $\vec{a} \cdot \vec{b}$ between all rows of A and B (line 7).** This operation can be performed by writing it as a matrix multiplication: AB^T contains all the dot products required for the full distance matrix. As matrix multiplications are the building block for many algorithms, highly optimized BLAS linear algebra libraries containing this operation exist for both the CPU and the GPU. An unvectorized implementation [43] is unable to take advantage of BLAS operations and is therefore less efficient.
3) **Sum the output of steps (1) and (2) to obtain the squared Euclidean distance (line 10-12).** Key insight when implementing this operation is that the vector lengths from step (1) are used multiple times and can be cached (line 10).
4) **For every descriptor i, find the codebook element j with the lowest distance (line 10-15).** The weight for a descriptor is then assigned to the codebook element corresponding to the column with the lowest distance.

The CPU implementation of vector quantization is able to use SSE instructions to execute floating point instructions on 4 single precision numbers at the same time. On a Core i7 920, the non-SSE version is 3.4 times slower. Our experiments use the SSE-optimized version only.

In conclusion, vector quantization involves computing the pair-wise Euclidean distances between n descriptors and m codebook elements. By simply vectorizing the computation of the Euclidean distance, the computation can be decomposed into steps which can be efficiently executed on the GPU.

C. Algorithm 2: GPU-Accelerated Kernel Value Precomputation

To compute kernel function values, we use the kernel function based on the χ^2 distance, which has shown the most accurate results in visual categorization (see section II-B). Our contribution is evaluating the χ^2 kernel function on the GPU efficiently, even for very large datasets which do not fit into memory. The χ^2 distance between feature vectors F and F' is:

$$dist_{\chi^2}(\vec{F}, \vec{F'}) = \frac{1}{2} \sum_{i=1}^{s} \frac{(\vec{F_i} - \vec{F_i'})^2}{\vec{F_i} + \vec{F_i'}}, \tag{3}$$

with s the size of the feature vectors. For notational convenience, $\frac{0}{0}$ is assumed to be equal to 0 iff $\vec{F_i} = \vec{F_i'} = 0$.

The kernel function based on this χ^2 distance then is:

$$k(\vec{F}, \vec{F'}) = e^{-\frac{1}{D} dist(\vec{F}, \vec{F'})}, \tag{4}$$

where D is an optional scalar to normalizes the distances [3]. Because the χ^2 distance is already constrained to lie between 0 and 1, this normalization is unnecessary and we therefore fix D to 1.

To use multiple input features, instead of relying on a single feature, the kernel function is extended in a weighted fashion for q features:

$$k(\{\vec{F}_{(1)}, ..., \vec{F}_{(q)}\}, \{\vec{F'}_{(1)}, ..., \vec{F'}_{(q)}\}) = e^{-\frac{1}{\sum_{j=1}^{q} w_j} \left(\sum_{j=1}^{q} w_j dist(\vec{F}_{(j)}, \vec{F'}_{(j)}) \right)}, \tag{5}$$

with w_j the weight of the j^{th} feature and $\vec{F}_{(j)}$ the j^{th} feature vector. An example of the use of multiple features with weights is the spatial pyramid [46, 47]. When using the spatial pyramid, additional features are extracted for specific parts of the image. For example, in a 2x2 subdivision of the image, feature vectors are extracted for each image quarter with a weight of $\frac{1}{4}$ for each quarter. Similarly, a 1x3 subdivision consisting of three horizontal bars, which introduces three new features (each with a weight of $\frac{1}{3}$). In this setting, the feature vector for the entire image has a weight of 1.

For vector quantization, discussed in the previous section, all input data and the resulting output fits into computer memory. For kernel value precomputation, memory usage is an important problem. For example, for a dataset with $50,000$ images, the input data is 12 GB and the output data is 19 GB. Therefore, special care must be taken when designing the implementation, to avoid holding all data in memory simultaneously. We divide the processing into evenly sized chunks. Each chunk corresponds to a square 1024x1024 subblock of the kernel matrix with all kernel function values, *i.e.* a chunk computes the kernel function values between 1024 vectors $\vec{F}$ and 1024 vectors $\vec{F'}$. The algorithm is given in pseudo code in Algorithm 2.

Algorithm 2 Compute Kernel Matrix Values with χ^2 Distance

```
1:  for every chunk of 1024 kernel matrix rows do
2:    for every chunk of 1024 kernel matrix columns do
3:      CurrentChunk ← 1024x1024 matrix with zeros
4:      for feature j = 1 to q do
5:        D ← dist_χ²(F_(j), F'_(j)) between 1024 vectors
          F_(j) and 1024 vectors F'_(j)
6:        CurrentChunk ← CurrentChunk + w_j D
7:      end for
8:      for all elements p of CurrentChunk do
9:        p ← e^(−1/(Σ_{j=1}^{q} w_j) p)
10:     end for
11:     Store CurrentChunk as part of the final kernel matrix
12:   end for
13: end for
```

To implement the $dist_{\chi^2}$ function in algorithm 2, we find that single precision is not accurate enough to sum many numbers. Therefore, we use double precision on the CPU with SSE instructions which can process 2 double precision numbers at the same time. Because double precision computations are 8 times slower than single precision on the GTX260, we use a Kahan summation [44] instead of switching to double precision on the GPU. For the CPU implementation, the additional operations of the Kahan summation are more expensive than switching to double precision.

IV. Experimental Setup

In this section, we discuss the setup of our experiments. In our first two experiments, we measure the speedup of our two contributions: GPU vector quantization and GPU kernel value precomputation. In the third experiment, instead of timing just the improved components, we measure the classification throughput of a complete visual categorization system. See Figure 1 for the pipeline of such a complete system. Software for the GPU-accelerated feature extraction will be released on our website[2], together with kernel value precomputation software.

A. Experiment 1: Vector Quantization Speed

We measure the relative speed of two vector quantization implementations: CPU and GPU versions of the vectorized approach from section III-B. The CPU implementation is SIMD-optimized. Measured times are the median of 25 runs; an initial warm-up run is discarded to exclude initialization effects. For the experiments, realistic data sizes are used, following the state-of-the-art [5]: a codebook of size $m = 4,000$; up to $20,000$ descriptors per image and descriptor lengths of $d = 128$ (SIFT) and $d = 384$ (ColorSIFT).

Because the compute power of CPU architectures still improves with every generation, we include two CPUs in our comparison of CPU and GPU, to show the rate of development in CPU compute speeds besides the increase in number of cores. Specifically, the single-core Opteron 250 (2.4GHz) from 2005 and the quad-core Core i7 920 (2.66GHz) from 2009 are included. For the quad-core Core i7, results for both a single-threaded and a multi-threaded CPU implementation are reported. These are compared to a Geforce GTX260 GPU (27 cores). Timing results are reported per frame; for a real dataset the times should be multiplied by the number of frames or images in the set.

B. Experiment 2: Kernel Value Precomputation Speed

To measure the speed of kernel value computation, we compare a CPU version and a GPU version based on the approach from section III-C. We evaluate these implementations on the same hardware as experiment 1.

To obtain timings results, we have chosen the large Mediamill Challenge training set of $30,993$ frames [48] with realistic feature vector lengths. Times required to precompute the kernel values are measured for different amounts of input features: from a single feature (total feature vector length $4,000$) up to 10 features (total feature vector length $128,000$). For a real system, the number of features might be even higher [5, 10].

C. Experiment 3: Visual Categorization Throughput

After accelerating two components of the categorization pipeline (see Figure 1) in the first two experiments, in this experiment, we measure the throughput of the complete system. The average time needed to classify a frame is referred to as the throughput of the system. For categorizing large datasets, the processing time required to push frames through the complete categorization pipeline is important, because this gives a good indication of the time needed to process the full dataset. For the throughput experiment, a comparison is made between the quad-core Core i7 920 CPU (2.66GHz) and the Gefore GTX260 GPU (27 cores).

[2]http://www.colordescriptors.com

V. Results

In this section, the results from the experiments listed in section IV are discussed. We will investigate the speed of vector quantization, the speed of precomputing kernel values and finally the throughput of a complete visual categorization system, with and without the GPU.

A. Experiment 1: Vector Quantization Speed

Figure 2 shows the vector quantization speeds for SIFT descriptors using different hardware platforms and implementations. From the results, it is shown that vector quantization on CPUs takes more time than on GPUs. The difference between the fastest single-threaded CPU and the fastest GPU is a factor of 13; both are using a vectorized implementation. If the CPU uses a multi-threaded implementation, the difference between the CPU and the GPU is a factor of 3.9. For a typical number of SIFT descriptors per frame, 10,000, this is the difference between $0.29s$ and $0.08s$ spent *per image* in vector quantization. In the ColorSIFT results, we see the same speedup: from $0.59s$ to $0.16s$. When processing datasets of thousands or even millions of images, this is an important acceleration.

An interesting observation, based on the single-threaded results, is that the CPU times can be used to roughly order them by release date. The single-core 2005 Opteron takes about 2.2 times longer than a single thread of a 2009 Core i7 920.

For the GPU, we obtain 212 GLOPS, which equals 0.65 instructions per clock cycle per core. This result includes the time it takes to transfer data between the CPU global memory and the GPU global memory. Without transfer times, performance would be 218 GLOPS. The optimized CUBLAS matrix multiplication used inside vector quantization achieves 0.74 instructions per cycle. The theoretical 875 GLOPS of the GPU is only reached when 2 instructions can be executed per clock cycle, which is possible for a specific combined add-multiply operation only. The computations use 70-80 GB/s out of a possible 117 GB/s GPU memory bandwidth.

For the Core i7 CPU, we obtain 43 GFLOPS out of a theoretical 100 GFLOPS for higher-clocked versions of this quad-core CPU architecture. For the Core i7 920, the theoretical maximum is about 80 GFLOPS. We observed (results not shown) that hyperthreading gives a speedup of at most 5 percent and sometimes decreases performance. Therefore, hyperthreading was disabled in our experiments. The CPU performance scales fairly well in terms of cores with the quad-core version being up to 3.4 times faster than the single-core version.

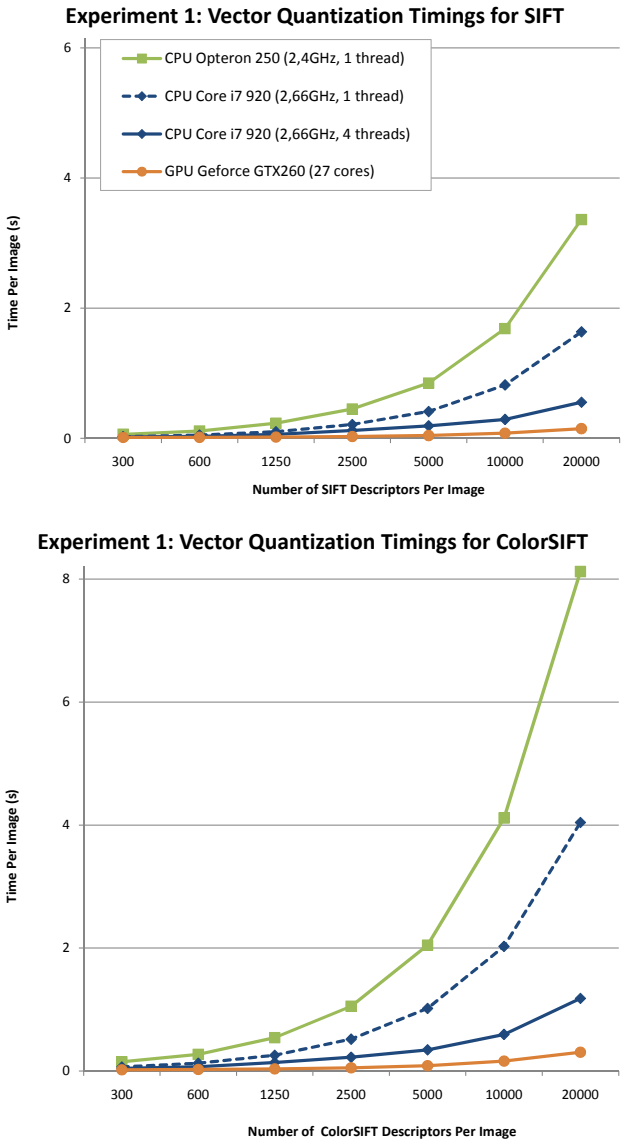

Fig. 2. Vector quantization speeds for a varying number of SIFT descriptors (top plot) or ColorSIFT descriptors (bottom plot). The difference between the multi-threaded CPU and the GPU is a factor of 3.9. The difference between the single-threaded CPU implementation and the GPU is a factor 13. The single-threaded results of the quad-core Core i7 CPU are shown as a dashed line, to indicate that it does not use all cores available.

In conclusion, the speedup through parallelization obtained for vector quantization is an important acceleration when processing large image datasets. When combined with GPU versions of the other image feature extraction stages (see Table I), even the most expensive feature can still be extracted in less than 1 second per image.

B. Experiment 2: Kernel Value Precomputation Speed

Figure 3 shows the kernel value precomputation speeds on different hardware platforms. The difference between a single GTX260 and a single Opteron CPU is a factor 74! The difference between a single thread of the more recent Core i7 CPU and the GTX260 GPU is a factor 37. When all threads of the Core i7 are used, the difference is a factor 10. When using a bag-of-words model with features computed for four pyramid levels (1x1, 2x2, 3x3 and 4x4), *e.g.* , a total feature vector length of $120,000$, this is the difference between 1360 minutes and 142 minutes. Again, the GPU architecture results in a substantial acceleration.

The GPU achieves 349 GFLOPS including memory transfers between the CPU global memory and the GPU global memory, with 1.10 instructions per clock cycle per core. Excluding memory transfers the GPU achieves 357 GFLOPS. More importantly, the computation uses 85-97 GB/s out of a possible 117 GB/s bandwidth to the GPU memory, showing that the algorithm is both bandwidth-intensive and compute-intensive. The multi-threaded SIMD-optimized CPU version achieves 30 GFLOPS on the quad-core Core i7 920. However, as noted in Section III-C, the CPU version uses double precision for its computation, which limits the theoretical GFLOPS of the Core i7 920 to 40 GFLOPS, instead of 80 GFLOPS for single precision computations.

C. Experiment 3: Visual Categorization Throughput

For categorizing large datasets, the average amount of time required to classify a frame from start to finish is important. This is commonly referred to as the throughput of the system. As an example of a large real-world dataset, we again use the Mediamill Challenge [48]. See Table III for an overview of the throughput. To classify $12,914$ keyframes from the test set takes 40.3 minutes when using the GPU, equal to 5.3 frames per second. This includes the time it takes to load the frames, extract densely sampled SIFT features[3], perform vector quantization, compute kernel values and apply trained models. When looking at the feature extraction and kernel value computation separately, the feature extraction per frame achieved a throughput of 12.3 frames per second (17.5 minutes for all frames) and the kernel value precomputation with 30 993 training samples achieved 9.4 frames per second (22.8 minutes for all frames). Compared to the single-threaded CPU version, which takes 11.5 hours to process these frames and therefore runs at 0.31 frames per second, the speedup for the complete pipeline is 17x. The multi-threaded CPU version, running on a quad-core CPU, needs 3 hours 15 minutes to process all frames, and is 3.6x faster than the single-threaded CPU version. The GPU version is 4.8 times faster than the quad-core CPU.

VI. Other Applications

The speedups for vector quantization and computing kernel values obtained using GPU processing can be applied to other problems than visual categorization as well. In this section, we will discuss how it applies to the k-means clustering algorithm and to processing text with the bag-of-words model, and how the faster processing can be used to improve visual categorization accuracy.

[3]SIFT feature extraction is also performed on the GPU.

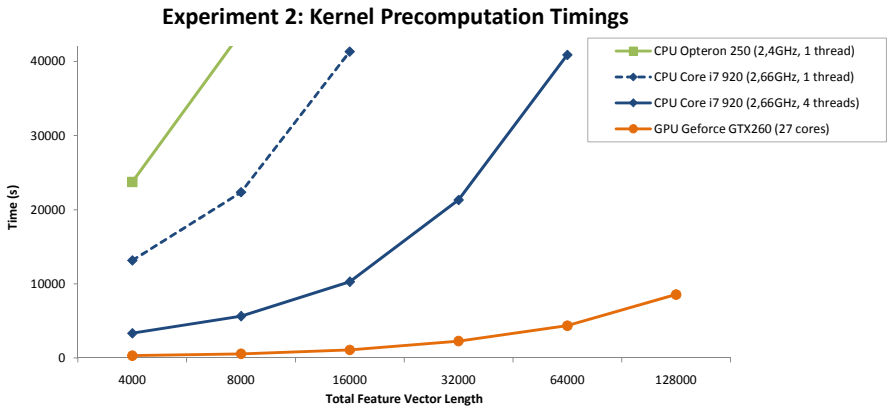

Fig. 3. Timings of kernel value precomputation on different hardware platforms for various total feature vector lengths. The difference between a GTX260 and a single-core Opteron CPU is a factor 74. The difference between the more recent Core i7 920 CPU utilizing 4 threads and the GPU is a factor 10. For reference, results of the Core i7 with only a single CPU thread are also shown (dashed line).

TABLE III
IN THIS TABLE, THE THROUGHPUT OF VISUAL CATEGORIZATION IS MEASURED USING THE MEDIAMILL CHALLENGE [48] DATASET. TIME MEASUREMENTS ARE FOR CLASSIFYING 12914 FRAMES, FRAMES PER SECOND (FPS) LISTINGS ARE THE AVERAGE TIME PER FRAME. THE SPEEDUP FOR THE GPU IS MEASURED AGAINST THE MULTI-THREADED CPU IMPLEMENTATION.

Visual Categorization Throughput

Operation	**CPU (1 thread)**		**CPU (4 threads)**			**GPU**		
	Time (min)	*Framerate*	*Time (min)*	*Framerate*	*Speedup*	*Time (min)*	*Framerate*	*Speedup*
Image Feature Extraction	99	2.2 fps	45.3	4.8 fps	2.2x	17.5	12.3 fps	2.6x
Compute Kernel Values/Apply Model	593	0.36 fps	150	1.4 fps	3.9x	22.8	9.4 fps	6.6x
Full Classification	692	0.31 fps	195.3	1.1 fps	3.6x	40.3	5.3 fps	4.8x

A. Application 1: k-means Clustering

The k-means clustering algorithm [49] is regularly used to construct the codebook used within a categorization pipeline. It is applicable to any real-valued set of data points and is one of the most common clustering algorithms in use. The k-means algorithm relies heavily on vector quantization. Once the set of k clusters has been initialized, all data points will be vector quantized against these k clusters. The data points are then assigned to the closest cluster, and the clusters are updated by computing the mean data value of all points assigned to that cluster. These steps are repeated until the clusters do not change anymore. Performing the vector quantization, *i.e.* finding the closest cluster for each data point, is the most expensive step in the k-means algorithm. When using the GPU vector quantization of experiment 1, a single iteration of the k-means algorithm took 3.4 seconds instead of 76 seconds, *i.e.* a speedup of 22.

B. Application 2: Bag-of-Words Model for Text Retrieval

The bag-of-words model as used in visual categorization is based on the original bag-of-words model as used for text. It results in the same kind of feature vectors with frequency counts of each 'codeword', where words are to be taken literally for text. Due to the large number of words possible, the feature vectors for documents can be very long. In the UCI datasets repository [50], there are several examples of textual bag-of-words datasets. The Enron e-mail collection, for example, contains almost $40,000$ documents which together contain $28,000$ unique words. The NYTimes news article collection contains $300,000$ documents with over $100,000$ unique words. The precomputation of kernel values from experiment 2 (to train a topic model based on annotations) and/or the computation of χ^2 distances (to *e.g.* cluster similar documents) can be directly applied to this text data, *i.e.* a speedup by a factor of 35.

C. Application 3: Multi-Frame Processing for Video Retrieval

The increased throughput for visual categorization has been instrumental in our participation in the visual categorization task of the TRECVID 2009 video retrieval benchmark [12]. This task has a test set with 280 hours of material in which 20 visual categories need to be identified. Instead of processing only the keyframes in the test set (97,150), the improved throughput made processing of up to 10 extra frames per shot feasible, for a total of 1 million frames. When looking at just the keyframe of a shot, there is a large chance that a visual category is not visible in that specific frame. By looking beyond the keyframes, more relevant frames can be identified and accuracy can be improved. See Figure 4 for an overview of accuracy results by including 1 to 10 additional frames. The likelihood a visual category occurs in a shot is estimated by either taking the maximum score of all frames in the shot or the average score. From the results, it is clear that taking the maximum score instead of the average gives better results. The accuracy gained by including more frames becomes smaller after 5 additional frames have been added, though the accuracy does increase. The relative improvement due to processing extra frames, while keeping all other components of the system the same, is 29%: from 0.175 to 0.226. This is in

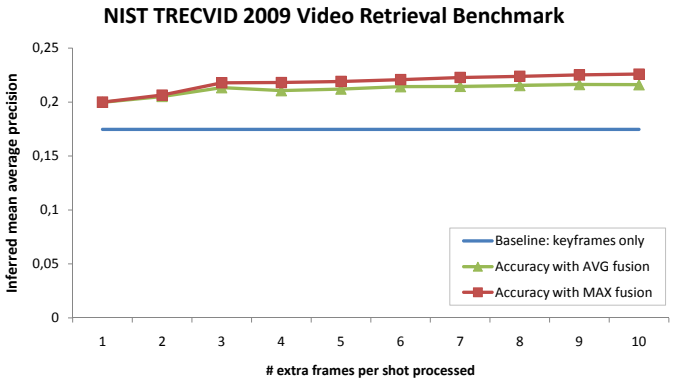

Fig. 4. The effect of multi-frame processing on the NIST TRECVID 2009 video retrieval benchmark [12], made possibly by the use of GPU computing. This task has a test set with 280 hours of material in which 20 visual categories need to be identified. The relative improvement due to processing extra frames is 29%. The baseline and all additional frame results use the same visual features and training procedures.

line with previous work in [51], where it was shown that processing additional frames will improve accuracy of visual categorization. In the official evaluation of the TRECVID 2009 visual categorization task, we obtained state-of-the-art results using the GPU and multi-frame processing: the system achieved the highest overall accuracy [10].

VII. Conclusions

This paper provides an efficiency analysis of a state-of-the-art visual categorization pipeline based on the bag-of-words model. In this analysis, two large bottlenecks were identified: the vector quantization step in the image feature extraction and the kernel value computation in the category classification. By using a vectorized GPU implementation of vector quantization, it is 3.9 times faster than when it is computed on a modern quad-core CPU. For the classification, we exploit the intrinsic property of kernel-based classifiers that only kernel values are needed. By precomputing these kernel values, the parameter tuning and model learning stages can reuse these values, instead of computing them on the fly for every visual category and parameter setting. Also, precomputing these kernel values on the GPU instead of a quad-core CPU accelerates it by a factor of 10. The latter GPU acceleration is applicable to both the learning phase and the training phase. The speedups obtained in the visual categorization pipeline are also applicable to other problems, such as text retrieval and video retrieval. Additionally, when the obtained speedup is used to process extra video frames in a video retrieval benchmark, the accuracy of visual categorization is improved by 29%.

Overall, by using a parallel implementation on the GPU, classifying unseen images is 17 times faster than a single-threaded CPU version, while giving the exact same results for visual categorization. Compared to a multi-threaded CPU implementation on a quad-core CPU, the GPU is 4.8 times faster.

References

[1] B. Huurnink, L. Hollink, W. van den Heuvel, and M. de Rijke, "Search behavior of media professionals at an audiovisual archive: A transaction log analysis," *Journal of the American Society for Information Science and Technology*, vol. 61, no. 6, pp. 1180–1197, June 2010.

[2] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *IEEE International Conference on Computer Vision*, 2003, pp. 1470–1477.

[3] J. Zhang, M. Marszałek, S. Lazebnik, and C. Schmid, "Local features and kernels for classification of texture and object categories: A comprehensive study," *International Journal of Computer Vision*, vol. 73, no. 2, pp. 213–238, 2007.

[4] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010.

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

[6] Y.-G. Jiang, J. Yang, C.-W. Ngo, and A. Hauptmann, "Representations of keypoint-based semantic concept detection: A comprehensive study," *IEEE Transactions on Multimedia*, vol. 12, no. 1, pp. 42–53, 2010.

[7] M. Marszałek, C. Schmid, H. Harzallah, and J. van de Weijer, "Learning object representations for visual object class recognition," 2007, Visual Recognition Challenge workshop, in conjunction with IEEE ICCV. [Online]. Available: http://lear.inrialpes.fr/pubs/2007/MSHV07

[8] S.-F. Chang, J. He, Y.-G. Jiang, E. E. Khoury, C.-W. Ngo, A. Yanagawa, and E. Zavesky, "Columbia university/VIREO-CityU/IRIT TRECVID2008 high-level feature extraction and interactive video search," in *Proceedings of the TRECVID Workshop*, 2008.

[9] A. Gaidon, M. Marszałek, and C. Schmid, "The PASCAL visual object classes challenge 2008 submission," INRIA-LEAR, Tech. Rep., 2008.

[10] C. G. M. Snoek, K. E. A. van de Sande, O. de Rooij, B. Huurnink, J. R. R. Uijlings, M. van Liempt, M. Bugalho, I. Trancoso, F. Yan, M. A. Tahir, K. Mikolajczyk, J. Kittler, M. de Rijke, J. M. Geusebroek, T. Gevers, M. Worring, D. C. Koelma, and A. W. M. Smeulders, "The MediaMill TRECVID 2009 semantic video search engine," in *Proceedings of the TRECVID Workshop*, 2009.

[11] D. Wang, X. Liu, L. Luo, J. Li, and B. Zhang, "Video diver: generic video indexing with diverse features," in *ACM International Workshop on Multimedia Information Retrieval*, 2007, pp. 61–70.

[12] A. F. Smeaton, P. Over, and W. Kraaij, "Evaluation campaigns and TRECVid," in *ACM International Workshop on Multimedia Information Retrieval*, 2006, pp. 321–330.

[13] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[14] J. R. R. Uijlings, A. W. M. Smeulders, and R. J. H. Scha, "Real-time bag-of-words, approximately," in *ACM International Conference on Image and Video Retrieval*, 2009.

[15] C.-C. Chang, Y.-C. Li, and J.-B. Yeh, "Fast codebook search algorithms based on tree-structured vector quantization," *Pattern Recognition Letters*, vol. 27, no. 10, pp. 1077–1086, 2006.

[16] F. Moosmann, B. Triggs, and F. Jurie, "Fast discriminative visual codebooks using randomized clustering forests," in *Neural Information Processing Systems*, 2006, pp. 985–992.

[17] F. J. Seinstra, J.-M. Geusebroek, D. Koelma, C. G. M. Snoek, M. Worring, and A. W. M. Smeulders, "High-performance distributed video content analysis with parallel-horus," *IEEE Multimedia*, vol. 14, no. 4, pp. 64–75, 2007.

[18] N. Cornelis and L. Van Gool, "Fast scale invariant feature detection and matching on programmable graphics hardware," in *IEEE Computer Vision and Pattern Recognition Workshops*, 2008.

[19] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, "Feature tracking and matching in video using programmable graphics hardware," *Machine Vision and Applications*, 2007.

[20] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," *IEEE Micro*, vol. 28, no. 2, pp. 39–55, 2008.

[21] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, "GPU computing," *Proceedings of the IEEE*, vol. 96, no. 5, pp. 879–899, 2008.

[22] R. Bordawekar, U. Bondhugula, and R. Rao, "Believe it or not! multi-core cpus can match gpu performance for flop-intensive application!" IBM Thomas J. Watson Research Center, Tech. Rep. IBM-RC24982, 2010.

[23] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey, "Debunking the 100x GPU vs. CPU myth: an evaluation

of throughput computing on CPU and GPU," *SIGARCH Computer Architecture News*, vol. 38, no. 3, pp. 451–460, 2010.
[24] R. Vuduc, A. Chandramowlishwaran, J. W. Choi, M. E. Guney, and A. Shringarpure, "On the limits of GPU acceleration," in *USENIX Workshop on Hot Topics in Parallelism*, 2010.
[25] T. Sharp, "Implementing decision trees and forests on a GPU," in *IEEE European Conference on Computer Vision*, 2008, pp. 595–608.
[26] B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in *International conference on Machine learning*, 2008, pp. 104–111.
[27] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys*, vol. 40, no. 2, pp. 1–60, 2008.
[28] D. G. Lowe, "Distinctive image features from scale-invariant keypoints." *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.
[29] K. Mikolajczyk and *et al.* , "A comparison of affine region detectors," *International Journal of Computer Vision*, vol. 65, no. 1-2, pp. 43–72, 2005.
[30] J.-M. Geusebroek, A. W. M. Smeulders, and J. van de Weijer, "Fast anisotropic gauss filtering," *IEEE Transactions on Image Processing*, vol. 12, no. 8, pp. 938–943, 2003.
[31] H. Jégou, M. Douze, and C. Schmid, "Packing bag-of-features," in *IEEE International Conference on Computer Vision*, 2009.
[32] J. C. van Gemert, C. J. Veenman, A. W. M. Smeulders, and J.-M. Geusebroek, "Visual word ambiguity," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 7, pp. 1271–1283, 2010.
[33] D. Cai, X. He, and J. Han, "Efficient kernel discriminant analysis via spectral regression," in *IEEE International Conference on Data Mining*, 2007, pp. 427–432.
[34] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.
[35] T.-N. Do, V.-H. Nguyen, and F. Poulet, "Speed up SVM algorithm for massive classification tasks," in *Advanced Data Mining and Applications*, 2008, pp. 147–157.
[36] M. Garland, S. L. Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, and V. Volkov, "Parallel computing experiences with CUDA," *IEEE Micro*, vol. 28, no. 4, pp. 13–27, 2008.
[37] KhronosGroup, *OpenCL website*, 2010, available at http://www.khronos.org/opencl/.
[38] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *Queue*, vol. 6, no. 2, pp. 40–53, 2008.
[39] J. Stratton, S. Stone, and W. mei Hwu, "MCUDA: An efficient implementation of CUDA kernels for multi-core CPUs," in *Workshop on Languages and Compilers for Parallel Computing*, 2008.
[40] G. Diamos, A. Kerr, and M. Kesavan, "Translating GPU binaries to tiered SIMD architectures with ocelot," Center for Experimental Research in Computer Systems, Tech. Rep., 2009.
[41] G. Diamos, "The design and implementation of ocelot's dynamic binary translator from PTX to multi-core x86," Center for Experimental Research in Computer Systems, Tech. Rep., 2009.
[42] M. Hassaballah, S. Omran, and Y. B. Mahdy, "A review of simd multimedia extensions and their usage in scientific and engineering applications," *Computer Journal*, vol. 51, no. 6, pp. 630–649, 2008.
[43] D. Chang, N. A. Jones, D. Li, and M. Ouyang, "Compute pairwise euclidean distances of data points with GPUs," in *Intelligent Systems and Control*, 2008, pp. 278–283.
[44] W. Kahan, "Pracniques: further remarks on reducing truncation errors," *Communications of the ACM*, vol. 8, no. 1, p. 40, 1965.
[45] Nvidia, *CUDA Programming Guide*, 2010, available at http://www.nvidia.com/CUDA.
[46] K. Grauman and T. Darrell, "The pyramid match kernel: Efficient learning with sets of features," *Journal of Machine Learning Research*, vol. 8, pp. 725–760, 2007.
[47] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories." in *IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 2169–2178.
[48] C. G. M. Snoek, M. Worring, J. C. van Gemert, J.-M. Geusebroek, and A. W. M. Smeulders, "The challenge problem for automated detection of 101 semantic concepts in multimedia," in *ACM International Conference on Multimedia*, 2006, pp. 421–430.
[49] A. R. Webb, *Statistical Pattern Recognition, 2nd Edition*. John Wiley & Sons, 2002.
[50] A. Asuncion and D. Newman, "UCI machine learning repository," 2007. [Online]. Available: http://archive.ics.uci.edu/ml
[51] C. G. M. Snoek, M. Worring, J.-M. Geusebroek, D. C. Koelma, and F. J. Seinstra, "On the surplus value of semantic video analysis beyond the key frame," in *IEEE International Conference on Multimedia & Expo*, 2005.

Koen van de Sande Koen E.A. van de Sande received a BSc in Computer Science (2004), a BSc in Artificial Intelligence (2004) and a MSc in Computer Science (2007) from the University of Amsterdam, The Netherlands. Currently, he is pursuing the PhD degree at the University of Amsterdam. His research interests include computer vision, visual categorization, (color) image processing, statistical pattern recognition and large-scale benchmark evaluations. He is a co-organizer of the annual VideOlympics. He is a student member of the IEEE.

Theo Gevers Theo Gevers is an Associate Professor of Computer Science at the University of Amsterdam, The Netherlands and a (part-time) full Professor at the Computer Vision Center (UAB), Barcelona, Spain. At the University of Amsterdam he is a teaching director of the MSc of Artificial Intelligence. He currently holds a VICI-award (for excellent researchers) from the Dutch Organisation for Scientific Research. His main research interests are in the fundamentals of content-based image retrieval, colour image processing and computer vision specifically in the theoretical foundation of geometric and photometric invariants. He is an associate editor for the IEEE Transactions on Image Processing. He is co-chair of the Internet Imaging Conference (SPIE 2005, 2006), co-organizer of the First International Workshop on Image Databases and Multi Media Search (1996), the International Conference on Visual Information Systems (1999, 2005), the Conference on Multimedia & Expo (ICME, 2005), and the European Conference on Colour in Graphics, Imaging, and Vision (CGIV, 2012). He is guest editor of the special issue on content-based image retrieval for the International Journal of Computer Vision (IJCV 2004) and the special issue on Colour for Image Indexing and Retrieval for the journal of Computer Vision and Image Understanding (CVIU 2004). He has published over 100 papers on colour image processing, image retrieval and computer vision. He is program committee member of a number of conferences, and an invited speaker at major conferences. He is a lecturer of post-doctoral courses given at various major conferences (CVPR, ICPR, SPIE, CGIV). He is member of the IEEE.

Cees Snoek Cees G.M. Snoek received the MSc degree in business information systems (2005) and the PhD degree in computer science (2005), both from the University of Amsterdam, where he is currently a senior researcher in the Intelligent Systems Lab. He was a visiting scientist at Carnegie Mellon University in 2003 and a Fulbright Junior Scholar at UC Berkeley in 2010-2011. His research interests include visual categorization, statistical pattern recognition, social media retrieval, and large-scale benchmark evaluations, especially when applied in combination for video search. He has published more than 90 refereed book chapters, journal, and conference papers in these fields and serves on the program committees of several conferences. He is the lead researcher of the award-winning MediaMill Semantic Video Search Engine, which is a consistent top performer in the yearly NIST TRECVID evaluations. He is a co-initiator and co-organizer of the annual VideOlympics, co-chair of the SPIE Multimedia Content Access conference, the Multimedia Grand Challenge at ACM Multimedia 2010 and a lecturer of postdoctoral courses given at international conferences and summer schools. He received a young talent (VENI) grant from the Dutch Organization for Scientific Research in 2008 and a Fulbright visiting scholar grant in 2010. He is a member of the IEEE.