Video Mining with Frequent Itemset Configurations

Till Quack[1], Vittorio Ferrari[2], and Luc Van Gool[1]

[1] ETH Zurich
[2] LEAR - INRIA Grenoble

Abstract. We present a method for mining frequently occurring objects and scenes from videos. Object candidates are detected by finding recurring spatial arrangements of affine covariant regions. Our mining method is based on the class of frequent itemset mining algorithms, which have proven their efficiency in other domains, but have not been applied to video mining before. In this work we show how to express vector-quantized features and their spatial relations as itemsets. Furthermore, a fast motion segmentation method is introduced as an attention filter for the mining algorithm. Results are shown on real world data consisting of music video clips.

1 Introduction

The goal of this work is to mine interesting objects and scenes from video data. In other words, to detect frequently occurring objects automatically. Mining such representative objects, actors, and scenes in video data is useful for many applications. For instance, they can serve as entry points for retrieval and browsing, or they can provide a basis for video summarization. Our approach to video data mining is based on the detection of recurring spatial arrangements of local features. These features are represented in quantized codebooks, which has been a recently popular and successful technique in object recognition, retrieval [14] and classification [10]. On top of this representation, we introduce a method to detect frequently re-occurring spatial configurations of codebook entries. Whereas isolated features still show weak links with semantic content, their co-occurrence has far greater meaning. Our approach relies on frequent itemset mining algorithms, which have been successfully applied to several other, large-scale data mining problems such as market basket analysis or query log analysis [1,2]. In our context, the concept of an item corresponds to a codebook entry. The input to the mining algorithm consists of subsets of feature-codebook entries for each video frame, encoded into "transactions", as they are known in the data mining literature [1]. We demonstrate how to incorporate spatial arrangement information in transactions and how to select the neighborhood defining the subset of image features included in a transaction. For scenes with significant motion, we define this neighborhood via motion segmentation. To this end, we also introduce a simple and very fast technique for motion segmentation on feature codebooks.

H. Sundaram et al. (Eds.): CIVR 2006, LNCS 4071, pp. 360–369, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Few works have dealt with the problem of mining objects composed of local features from video data. In this respect, the closest work to ours is by Sivic and Zisserman [5]. However, there are considerable differences. [5] starts by selecting subsets of quantized features. The neighborhoods for mining are always of fixed size (e.g. the 20 nearest neighbors). Each such neighborhood is expressed as a simple, orderless bag-of-words, represented as a sparse binary indicator vector. The actual mining proceeds by computing the dot-product between all pairs of neighborhoods and setting a threshold on the resulting number of codebook terms they have in common. While this definition of a neighborhood is similar in spirit to our transactions, we also include information about the localization of the feature within its neighborhood. Furthermore, the neighborhood itself is not of fixed size. For scenes containing significant motion, we can exploit our fast motion segmentation to restrict the neighborhood to features with similar motions, and hence more likely to belong to a single object. As another important difference, unlike [5] our approach does not require pairwise matching of bag-of-words indicator vectors, but it relies instead on a frequent itemset mining algorithm, which is a well studied technique in data mining. This brings the additional benefit of knowing *which* regions are common between neighborhoods, versus the dot-product technique only reporting *how many* they are. It also opens the doors to a large body of research on the efficient detection of frequent itemsets and many deduced mining methods.

To the best of our knowledge, no work has been published on frequent itemset mining of video data, and very little is reported on static image data. In [3] an extended association rule mining algorithm was used to mine spatial associations between five classes of texture-tiles in aerial images (forest, urban etc.). In [4] association rules were used to create a classifier for breast cancer detection from mammogram-images. Each mammogram was first cropped to contain the same fraction of the breast, and then described by photometric moments. Compared to our method, both works were only applied to static image data containing rather small variations.

The remainder of this paper is organized as follows. First the pre-processing steps (i.e. video shot detection, local feature extraction and clustering into appearance codebooks) are described in section 2. Section 3 introduces the concepts of our mining method. Section 4 describes the application of the mining method to video sequences. Finally, results are shown in section 5.

2 Shot Detection, Features and Appearance Codebooks

The main processing stages of our system (next sections) rely on the prior subdivision of the video into shots. We apply the shot partitioning algorithm [6], and pick four "keyframes" per second within each shot. As in [5], this results in a denser and more uniform sampling than when using the keyframes selected by [6]. In each keyframe we extract two types of affine covariant features (*regions*): Hessian-Affine [7] and MSER [8]. Affine covariant features are preferred over simpler scale-invariant ones, as they provide robustness against viewpoint

changes. Each normalized region is described with a SIFT-descriptor [9]. Next, a quantized codebook [10] (also called "visual vocabulary" [5]) is constructed by clustering the SIFT descriptors with the hierarchical-agglomerative technique described in [10]. In a typical video, this results in about 8000 appearance clusters for each feature type.

We apply the 'stop-list' method known from text-retrieval and [5] as a final polishing: very frequent and very rare visual words are removed from the codebook (the 5% most and 5% least frequent). Note that the following processing stages use only the spatial location of features and their assigned appearance-codebook id's. The appearance descriptors are no longer needed.

3 Our Mining Approach

Our goal is to find frequent spatial configurations of visual words in video scenes. For the time being, let us consider a configuration to be just an unordered set of visual words. We add spatial relationships later, in section 3.2. For a codebook of size d there are 2^d possible subsets of visual words. For each of our two feature types we have a codebook with about 8000 words, which means d is typically > 10000, resulting in an immense search space. Hence we need a mining method capable of dealing with such a large dataset and to return frequently occurring word combinations. Frequent itemset mining methods are a good choice, as they have solved analogous problems in market basket like data [1,2]. Here we briefly summarize the terminology and methodology of frequent itemset mining.

3.1 Frequent Itemset Mining

Let $I = \{i_1 \dots i_p\}$ be a set of p *items*. Let A be a subset of I with l items,i.e. $A \subseteq I, |A| = l$. Then we call A a *l-itemset*. A transaction is an itemset $T \subseteq I$ with a transaction identifier $tid(T)$. A transaction database D is a set of transactions with unique identifiers $D = \{T_1 \dots T_n\}$. We say that a transaction T *supports* an itemset A, if $A \subseteq T$. We can now define the support of an itemset A in the transactions-database D as follows:

Definition 1. *The support of an itemset $A \in D$ is*

$$support(A) = \frac{|\{T \in D | A \subseteq T\}|}{|D|} \in [0, 1] \tag{1}$$

An itemset A is called *frequent* in D if $support(A) \geq s$ where s is a threshold for the minimal support defined by the user.

Frequent itemsets are subject to the monotonicity property: all l-subsets of frequent $(l+1)$-sets are also frequent. The well known APriori algorithm [1] takes advantage of the monotonicity property and allows us to find frequent itemsets very quickly.

3.2 Incorporating Spatial Information

In our context, the items correspond to visual words. In the simplest case, a transaction would be created for each visual word, and would consist of an orderless bag of all other words within some image neighborhood. In order to include also spatial information (i.e. locations of visual words) in the mining process, we further adapt the concept of an item to our problem. The key idea is to encode spatial information directly in the items. In each image we create transactions from the neighborhood around a limited subset of selected words $\{v_c\}$. These words must appear in at least f_{min} and at most in f_{max} frames. This is motivated by the notion that neighbourhoods containing a very infrequent word would create infrequent itemsets, neighbourhoods around extremely frequent word have a high probability of being part of clutter. Each v_c must also have a matching word in the previous frame, if both frames are from the same shot. Typically, with these restrictions, about 1/4 of the regions in a frame are selected.

For each v_c we create a transaction which contains the surrounding k nearest words together with their rough spatial arrangement. The neighbourhood around v_c is divided into B sections. In all experiments we use $B = 4$ sections. Each section covers $90\,^\circ$ plus an overlap $o = 5\,^\circ$ with its neighboring sections, to be robust against small rotations. We label the sections $\{tl, tr, bl, br\}$ (for "top-left", "top-right", etc.), and append to each visual word the label of the section it lies in. In the example in figure 1, the transaction created for v_c is $T = \{tl55, tl9, tr923, br79, br23, bl23, bl9\}$. In the following, we refer to the selected words $\{v_c\}$ as *central* words. Although the approach only accomodates for small rotations, in most videos objects rarely appear in substantially different orientations. Rotations of the neighborhood stemming from perspective transformations are safely accomodated by the overlap o. Although augmenting the items in this fashion increases their total number by a factor B, no changes to the frequent itemset mining algorithm itself are necessary. Besides, thanks to the careful selection of the central visual words v_c, we reduce the number of transactions and thus the runtime of the algorithm.

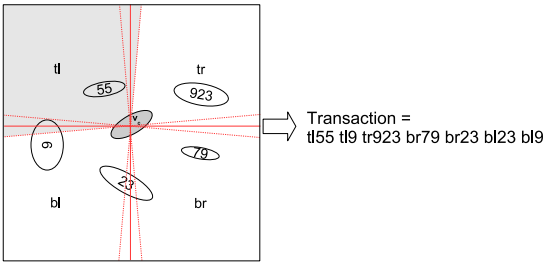


Fig. 1. Creating transaction from a neighborhood. The area around a central visual word v_c is divided into sections. Each section is labeled (tl, tr, bl, br) and the label is appended to the visual word ids.

3.3 Exploiting Motion

Shots containing significant motion[1] allow us to further increase the degree of specificity of transactions: if we had a rough segmentation of the scene into object candidates, we could restrict the neighborhood for a transaction to the segmented area for each candidate, hence dramatically simplifying the task of the mining algorithm. In this case, as the central visual words v_c we pick the two closest regions to the center of the segmented image area. All other words inside the segmented area are included in the transaction (figure 3).

In this section, we propose a simple and very fast motion segmentation algorithm to find such object candidates. The assumption is that interesting objects move independently from each other within a shot. More precisely, we can identify groups of visual words which translate consistently from frame to frame. The grouping method consists of two steps:

Step 1. Matching words. A pair of words from two frames $f(t), f(t+n)$ at times t and $t+n$ is deemed matched if they have the same codebook ids (i.e. they are in the same appearance cluster), and if the translation is below a maximum translation threshold t_{max}. This matching step is extremely fast, since we rely only on cluster id correspondences. In our experiments we typically use $t_{max} = 40$ pixels and $n = 6$ since we operate on four keyframes per second.

Step 2. Translation clustering. At each timestep t, the pairs of regions matched between frames $f(t)$ and $f(t+n)$ are grouped according to their translation using k-means clustering. In order to determine the initial number of motion groups k, k-means is initialized with a leader initialization [12], on the translation between the first two frames. For each remaining timestep, we run k-means three times with different values for k, specifically

$$k(t) \in \{k(t-1)-1, k(t-1), k(t-1)+1\} \tag{2}$$

where $k(t-1)$ is the number of motion groups in the previous timestep. $k(t)$ is constrained to be in [2...6]. This prevents the number of motion groups from changing abruptly from frame to frame. To further improve stability, we run the algorithm twice for each k with different random initializations. From the resulting different clusterings, we keep the one with the best mean silhouette value [13]. We improve the quality of the motion groups with the following filter. For each motion group, we estimate a series of bounding-boxes, containing from 80% progressively up to all regions closest to the spatial median of the group. We retain as bounding-box for the group the one with the maximal density $\frac{number\ of\ regions}{bounding\ box\ area}$. This procedure removes from the motion groups regions located far from most of the others. These are most often mismatches which accidentally translate similar to the group.

The closest two visual words to the bounding box center are now selected as the central visual word v_c for the motion group. Figure 2 shows detected motion groups for a scene of a music videoclip.

[1] Since shot partitioning [6] returns a single keyframe for static shots and several keyframes for moving shots, we can easily detect shots with significant motion.

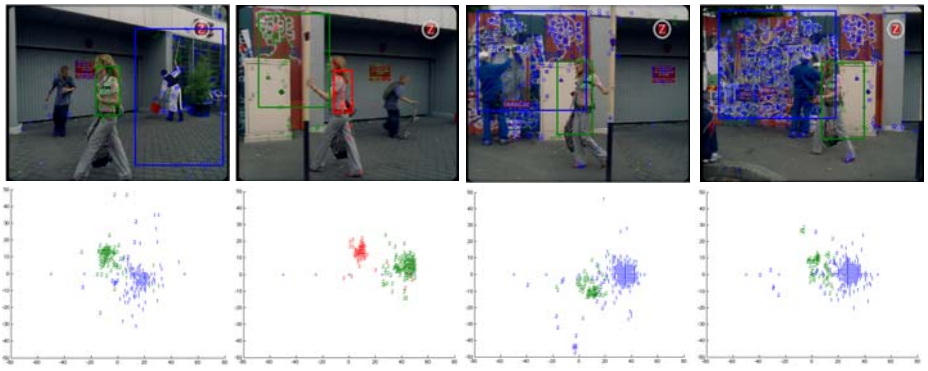


Fig. 2. First row: motion groups (only region centers shown) with bounding boxes. Second row: motion groups in translation space. Note: colors do not necessarily correspond along a row, since groups are not tracked along time.

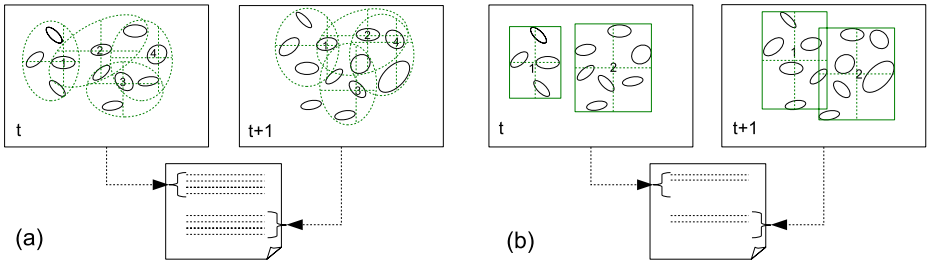


Fig. 3. Creating transactions: (a) static shots: transactions are formed around each v_c from the k-neighborhood. (b) shots with considerable motion: a motion group is the basis for a transaction, thus the number of items in a transaction is not fixed but given by the size of the motion group. With (b) in general fewer transactions are generated.

4 Mining the Entire Video

We quickly summarize the processing stages from the previous sections. A video is first partitioned into shots. For rather static shots we create transactions from a fixed neighborhood around each central word (subsection 3.2). For shots with considerable motion, we use as central words the two words closest to the spatial center of the motion group, and create two transactions covering only visual words within it. For frequent itemset mining itself we use an implementation of APriori from [11]. We mine so called "maximal frequent itemsets". An itemset is called maximal if no superset is frequent. Only sets with four or more items are kept.

Note how frequent itemset mining returns sparse but discriminative descriptions of neighborhoods. As opposed to the dot-product of binary indicator vectors used in [5], the frequent itemsets show *which* visual words cooccur in the mined transactions. Such a sparse description might also be helpful for efficiently indexing mined objects.

4.1 Choosing a Support Threshold

The choice of a good minimal support threshold s in frequent itemset mining is not easy, especially in our untraditional setting where items and itemsets are constructed without supervision. If the threshold is too high, no frequent itemsets are mined. If it is too low, too many (possibly millions) are mined. Thus, rather than defining a fixed threshold, we run the algorithm with several thresholds, until the number of frequent itemsets falls within a reasonable range. We achieve this with a binary split search strategy. Two extremal support thresholds are defined, s_{low} and s_{high}. The number of itemsets is desired to be between n_{min} and n_{max}. Let n be the number of itemsets mined in the current step of the search, and s be the corresponding support threshold. If the number of itemsets is not in the desired range, we update s by the following rule and rerun the miner:

$$s^{(t+1)} = \begin{cases} s^{(t)} + \frac{(s_{high} - s^{(t)})}{2}\,,\ s_{low} = s^{(t)} \text{ if } n > n_{max} \\ s^{(t)} - \frac{(s^{(t)} - s_{low})}{2}\,,\ s_{high} = s^{(t)} \text{ if } n < n_{min} \end{cases}$$

Since the mining algorithm is very fast, we can afford to run it several times (runtimes reported in the result section).

4.2 Finding Interesting Itemsets

The output of the APriori algorithm is usually a rather large set of frequent itemsets, depending on the minimal support threshold. Finding *interesting* item sets (and association rules) is a much discussed topic in the data mining literature [16]. There are several approaches which define interestingness with purely statistical measures. For instance, itemsets whose items appear statistically dependent are interesting. A measure for independence can be defined as follows. Assuming independence, the expected value for the support of an itemset is computed from the product of the supports of the individual items. The ratio of actual and expected support of an itemset is computed and its difference to 1 serves as an interestingness measure (i.e. difference to perfect independence). Only itemsets for which this difference is above a given threshold are retained as interesting. This was suggested in [11] and had in general a positive effect on the quality of our mining results.

Another strategy is to rely on domain-specific knowledge. In our domain, itemsets which describe a spatial configuration stretching across multiple sections tl, tr, bl, br are interesting. These itemsets are less likely to appear by coincidence and also make the most of our spatial encoding scheme, in that these configurations respect stronger spatial constraints. The number of sections that an itemset has to cover in order to be selected depends on a threshold $n_{sec} \in \{1 \dots 4\}$. Selecting interesting itemsets with this criteria is easily implemented and reduces the number of itemsets drastically (typically by a factor 10 to 100).

4.3 Itemset Clustering

Since the frequent itemset mining typically returns spatially and temporally overlapping itemsets, we merge them with a final clustering stage. Pairs of itemsets which jointly appear in more than F frames and share more than R regions

are merged. Merging starts from the pair with the highest sum $R + F$. If any of the two itemsets in a pair is already part of a cluster, the other itemset is also added to that cluster. Otherwise, a new cluster is created.

5 Results and Conclusion

We present results on two music videos from Kylie Minogue [17,18]. In particular the clip [17] makes an interesting test case for mining, because the singer passes by the same locations three times, and it even appears replicated several times in some frames. (Figure 4, third row). Hence, we can test whether the miner picks up the reappearing objects. Furthermore, the scene gets more and more crowded with time, hence allowing to test the system's robustness to clutter.

A few of the objects mined from the 1500 keyframes long clip [17] are shown in figure 4. The full miner was used, including motion grouping and itemset filtering with $n_{sec} = 2$. The building in the first row is mined in spite of viewpoint changes, thereby showing this ability of our approach. The street scene in the second row is mined in spite of partial occlusion. Finally, in the third row the singer is mined, based on her shirt. The second video frame of this row is particularly interesting, since the singer appears in three copies and all of them were mined.

Figure 5 shows typical results for mining with a fixed 40-neighborhood, i.e. without motion segmentation, akin to what proposed by [5]. As can be seen in subfigures 5a and 5b, only smaller parts of the large objects from figure 4 are mined. More examples of objects mined at the 40-neighborhood scale are shown in the other subfigures. Comparing these results to those in figure 4 highlights the benefits of defining the neighborhood for mining based on motion segmentation. Thanks to it, objects can be mined at their actual size (number of regions), which can vary widely from object to object, instead than being confined to a fixed, predefined size. Additionally, the singer was not mined when motion segmentation was turned off. The last row of figure 4 shows example objects mined from the clip [18] with a 40-neighbourhood. Our algorithm is naturally challenged by sparsely textured, non-rigid objects. As an example one could mention the legs of the main character. There are few features to begin with and the walking motion strongly changes the configuration of those, thus not the whole body is detected as object.

In table 1 we compare quantitatively mining with motion segmentation, and with a fixed 40-neighborhood for the clip [17]. Note that there are only 8056 transactions when using motion segmentation, compared to more than half a million when using a fixed 40-neighborhood. While the runtime is very short for both cases, the method is faster for the 40-neighborhood case, because transactions are shorter and only shorter itemsets were frequent. Additionally, in the 40-NN case, the support threshold to mine even a small set of only 285 frequent itemsets has to be set more than a factor 10 lower. The mean time for performing motion segmentation matching + k-means clustering) was typically about $0.4s$ per frame, but obviously depends on the number of features detected per frame.

In conclusion, we showed that our mining approach based on frequent itemsets is a suitable and efficient tool for video mining. Restricting the neighborhood by

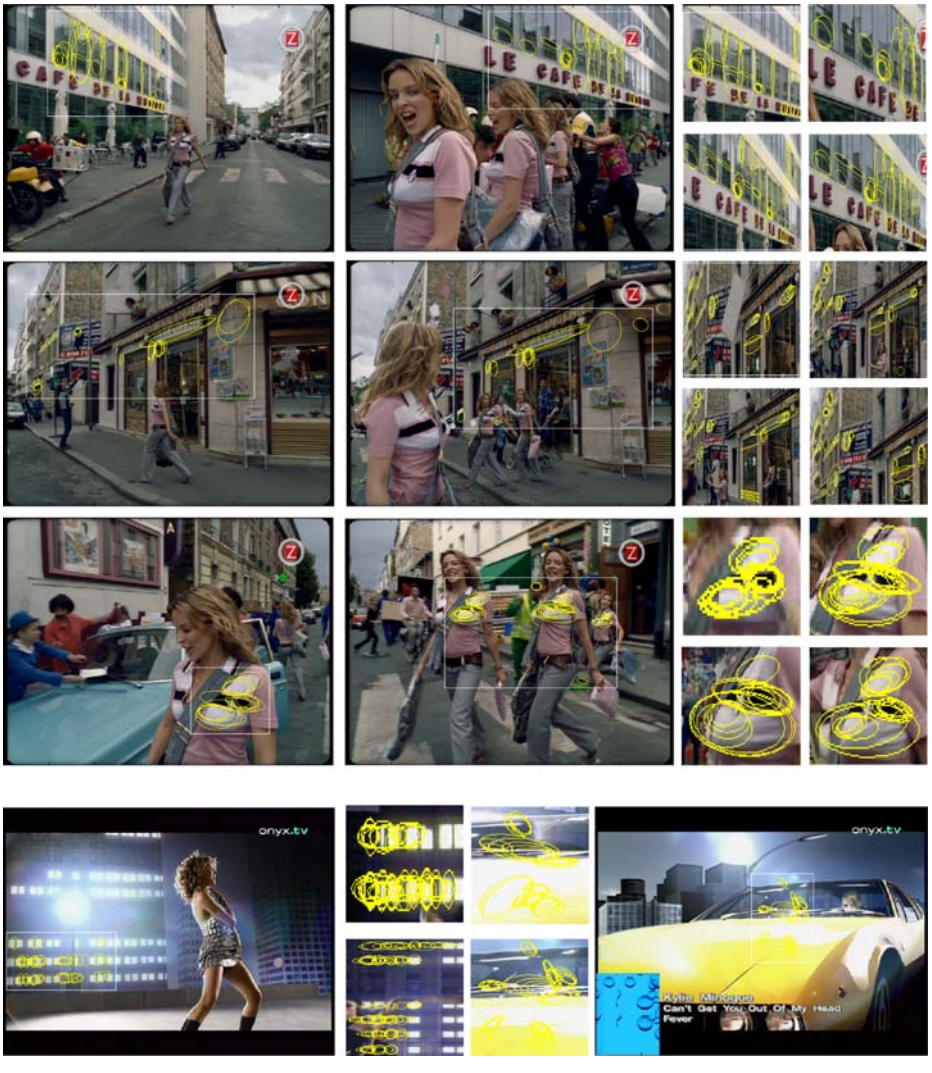
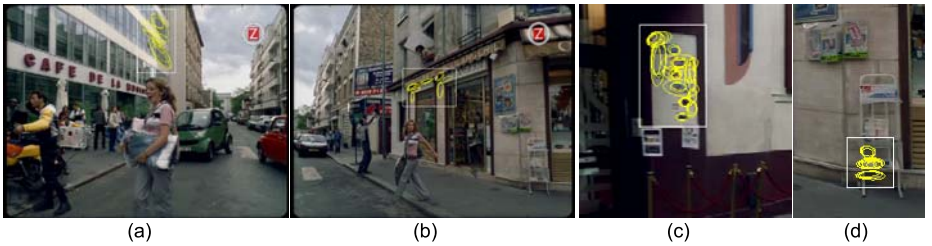


Fig. 4. Top three rows: results for clip [17]. Each row shows instances from one itemset cluster. Bottom row: results for clip [18].



(a) (b) (c) (d)

Fig. 5. Examples for clip [17] mined at a fixed 40 neighborhood