The Priority Curve Algorithm For Video Summarization

M. Fayzullin and V.S.Subrahmanian
University of Maryland
{fms,vs}@cs.umd.edu

M. Albanese and A. Picariello
Università di Napoli
{malbanes,picus}@unina.it

ABSTRACT

In this paper, we introduce the concept of a priority curve associated with a video. We then provide an algorithm that can use the priority curve to create a summary (of a desired length) of any video. The summary thus created exhibits nice continuity properties and also avoids repetition. We have implemented the priority curve algorithm (PCA) and compared it with other summarization algorithms in the literature. We show that PCA is faster than existing algorithms and also produces better quality summaries. The quality of summaries was evaluated by a group of 200 students in Naples, Italy, who watched soccer videos. We also briefly describe a soccer video summarization system we have built on using the PCA architecture and various (classical) image processing algorithms.

Categories and Subject Descriptors. Information Systems [Information Storage and Retrieval]: Information Search and Retrieval: Information Filtering
General Terms. Algorithms, Design, Performance
Keywords. Probabilistic, Video, Summarization, System

1. INTRODUCTION

Despite the vast amount of work on video databases, and the existing work on summarizing video [14, 6, 8, 10, 13, 15], there is no commonly accepted solution to the problem of automatically producing video summaries that take content into account and scale to massive data applications. For example, if FIFA (the International Soccer Federation) wanted to sell videos of soccer games, there would be tens of thousands of such videos. Potential customers may wish to watch small clips of the video to decide which videos they wish to buy. Though financial resources may be available to manually summarize each video, the ability to automatically summarize such videos is likely to be attractive.

In past work [7], we proposed a model for video summarization based on three important principles: the summary produced must be *continuous*, must contain high *priority* objects and actions/events occurring in the video, and must avoid *repetition*. This model was called the CPR model (continuity, priority, and non-repetition). We encoded the relative importance of these parameters in terms of an objective function *eval* that evaluated the quality of a summary. The problem then was to find a summary (set of video frames) from the video being summarized such that (i) the cardinality of the set of frames was less than or equal to a desired maximal summary length ℓ and (ii) the summary had the highest possible evaluation. We showed that the problem of finding an optimal summary (w.r.t. the objective function *eval*) was NP-complete - as a consequence, any exact algorithm to find such an optimal summary will be inefficient (even though we gave one) unless $P = NP$. We therefore proposed three alternative heuristic algorithms called CPRdyn, CPRgen, and SEA to find summaries fast. All these methods attempted to use the objective function to find a good summary.

In this paper, we still retain the core idea from [7] that the CPR criteria are important. However, we use a *completely different approach* to finding good summaries fast. Our *priority curve algorithm* (PCA for short) completely eliminates the objective function upon which the previous algorithms were based, but captures the same intuitions in a compelling way. Instead, we leverage the following intuitions.

- **Block creation.** We first split the video into blocks - blocks could either be of equal sizes, or they could be obtained as a result of segmenting the video using any standard video segmentation algorithm [16], [9] [25], [12]. As is common, the segments generated are usually relatively small.
- **Priority assignment.** Each block is then assigned a *priority* based on the objects and events occurring in that block. Yet another alternative would use the audio stream and/or accompanying text associated with the video to identify the priority of the block. The priority assignment can be done automatically using object and event detection algorithms or can be done manually[1].
- **Peak detection.** We then conceptually proceed as follows. Consider a graph whose x axis consists of block numbers and whose y axis describes the priority of the blocks. We identify the blocks associated with the *peaks* in this graph using a *peak identification algorithm.*

[1]Our video summarization application uses both image processing algorithms and some manual annotation

- **Block merging.** Subsequently, we merge multiple adjacent blocks into one. These are cases where the same or similar events are occurring in these blocks even though the blocks were different segments produced by the video segmentation algorithm. Determining when to merge different blocks may be done, for example, by examining standard image differentiation algorithms.
- **Block elimination.** We then have a *block elimination algorithm* which eliminates certain unworthy blocks - these are blocks whose priority is too low for inclusion in the summary. This is done by analyzing the distribution of priorities of blocks, as well as the relative sizes of the blocks involved, rather than by setting an artificial threshold.
- **Block resizing.** Finally, we have a *block resizing algorithm* that shrinks the remaining blocks so that the final summary consists of these resized blocks adjusted to fit the desired total length.

In this paper, we describe the PCA algorithm and provide experimental results using a set of 50 soccer videos to show that the PCA algorithm beats the best known previous algorithm both in terms of computation time (to find a summary) and in terms of the quality of the summary. As most readers are certainly aware, the only really reasonable way to assess the quality of summaries is to have the summaries evaluated by humans as any theoretical model may disagree with the intuitions of humans (which are hard to express and capture computationally). We therefore used a group of 200 students at the University of Naples to test the summaries produced of the above 50 videos not only with PCA , but also with other algorithms developed earlier.

2. PRIORITY CURVE ALGORITHM (PCA)

Given an input video v containing $len(v)$ frames, and an integer $0 < k \leq len(v)$, PCA finds a set of exactly k frames from the video. It attempts to ensure that these frames contain high priority objects and events in them (for example, when summarizing a soccer video, it might attempt to find frames containing goals, red cards, and other notable events from the game). It also attempts to ensure that the summary is not full of jitter by picking continuous portions of the video. Last, but not least, it attempts to eliminate repetition.

2.1 Overview of PCA

PCA is a complex algorithm consisting of many parts. Fortunately, many of these parts can be implemented using standard image processing algorithms. Figure 1 shows the key components of PCA . In the rest of this section, we describe the overall functions of the PCA algorithm. In particular, we will show how some of these components can be built directly using standard image processing methods, while others need new contributions. We will use an application we have built for summarizing soccer video to illustrate our techniques. Section 3 describes the details of the new components.

2.1.1 Block creation

The first part of PCA is a *block creation* component that takes a video file as input and automatically splits the video file into blocks. This can be done in one of two ways. In the first way, the person interested in summarizing a collection of videos simply says that each block is a certain number of frames (e.g. he may say that a block is a collection of 1800 frames, representing one minute of the video at a playback rate of 30 frames per second). Alternatively, we may use *any* classical video segmentation algorithm [12], [9] [16], [25] to split the video into a set of blocks. Each block is a segment returned by the segmentation algorithm. The video is thus represented as a sequence of blocks, possibly of varying sizes.

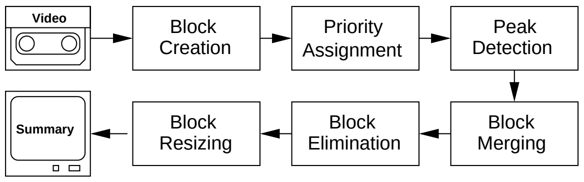

Figure 1: PCA Architecture

2.1.2 Priority assignment

The segmented video is then fed into a *priority assignment* module which examines each block and assigns a priority to it. For example, the person summarizing a soccer video may specify priorities as follows (goal: 10, red card: 7, yellow card: 6, corner kick: 3, fight: 10, and so on). The priority assignment component can also be implemented in many ways. In our soccer video summarization application, for example, the priority assignment could be done by using image processing algorithms for events such as goal shot detection or red card detection. Alternatively. in a military surveillance application that wants to assign high priority gunshot detection in video, an image analysis program that identifies gunshots or explosions may be used to assign high priorities to such events. As a last resource, it can also be annotated by a human.

2.1.3 Peak detection

Once the priorities have been assigned to each block, block IDs (increasing with time), together with their priorities, are shipped to a *peak detection* module. This module creates a graph whose x axis consists of block IDs, and whose y axis shows the priority of each block. The *Peaks()* algorithm we have developed can automatically find *peaks* in this priority curve. Figure 2 shows an example graph and the peaks involved. Intuitively, a peak consists of a sequence of blocks containing high priority events.

2.1.4 Block merging

The set of blocks thus identified in each peak is then shipped to a *block merging* module that examines these blocks and tries to determine if any of them can be merged. For example, it may turn out that there may be three blocks - the first containing the play just before a goal, the second containing the goal itself, while the third shows the post goal celebration. The block merging algorithm uses *rules* to determine conditions under which multiple contiguous blocks can be merged together into a new block (whose priority equals the sum of the priorities of the blocks being merged).

2.1.5 Block elimination

The set of blocks produced after merging is then shipped to a *block elimination* module. This module eliminates blocks whose priority is too low. For example, it may turn out that 10 merged blocks are returned after the block merging algorithm and these 10 blocks have a total of 5000 frames. If we want a summary consisting of just 3600 frames, we may want to re-examine whether a block of relatively low priority should be eliminated. For example, if we compute the average priority of the 10 blocks above to be 25 and the standard deviation to be 3, then we may want to eliminate all blocks with a priority under 16 (this is the classical statistical model which says that for a normal distribution, most objects in the distribution must occur within 3 standard deviations of the mean). Other statistical rules can also be used here.

2.1.6 Block resizing

The next component is the *block resizing component*. For example, even after eliminating "low value" blocks above, it may turn out that we still have 8 blocks containing a total of 4500 frames. This must somehow be reduced to 3600. The block resizing component eliminates frames from the blocks in proportion to the priorities of the blocks involved. For example, let us say that the total priorities of all the 8 blocks is 800, and that a particular block (containing 1500 frames) has priority 200. Thus, this block accounts for a quarter of the entire priority of the 4500 frame sample. As a consequence, the block should be allowed to contribute a quarter of the 3600 frames allowed in the summary, i.e. 900 frames should be chosen from this block for inclusion in the summary. Our block resizing algorithm will show *how* to select the best 900 frames from the 1500 frame block. The block resizing component sequences the resized blocks together to create the final summary.

3. DETAILS OF PCA COMPONENTS

In this section, we describe how to implement the *peak identification*, *block merging*, *block elimination*, and *block resizing* modules. The block creation and priority assignment modules can be readily implemented using classical segmentation algorithms [16], [25], [9], [12] and classical object recognition and/or event recognition algorithms [2], [3], [24] which have been extensively studied in the literature. Hence, we do not describe these components in detail as they are well known.

3.1 Peak Identification Module

Let $b_1, \ldots, b_n$ be the blocks in the video (e.g. after the segmentation process). Let p_i denote the priority of block b_i.

DEFINITION 3.1 ((r,s)-PEAK). *Suppose $r \in (0, n/2]$ is an integer and $s \in [0,1]$ is a real number. Blocks $b_j, \ldots, b_{j+r}$ are said to be an (r,s)-peak iff*

$$\frac{\sum_{j \le i \le j+r} p_i}{\sum_{j-\frac{r}{2} \le i \le j+\frac{3r}{2}} p_i} \ge s$$

Suppose we wish to check if a sequence of r blocks $S_1 = b_j, \ldots, b_{j+r}$, constitutes a peak. The above definition looks at $\frac{r}{2}$ blocks before the sequence as well as $\frac{r}{2}$ blocks after the sequence, i.e. the sequence $S_2 = b_{ji\frac{r}{2}}, \ldots, b_j, \ldots, b_{j+r}, \ldots, b_{j+\frac{3r}{2}}$

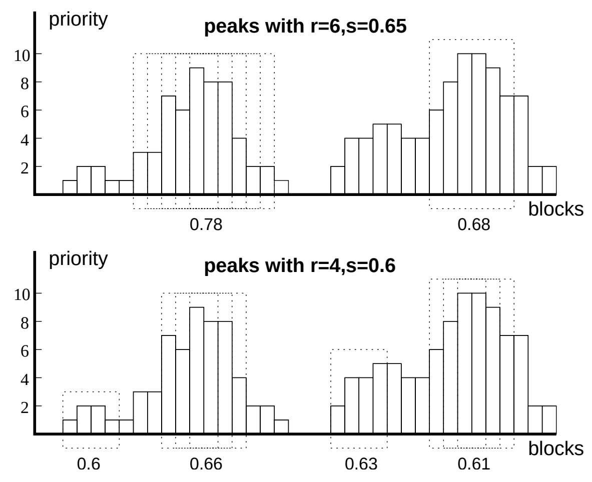

Figure 2: Peaks

is considered. This latter sequence S_2 is of width $2r$. We sum up the priorities of all blocks in S_2 - let us call this sum s_2. Likewise, we sum up the priorities of all blocks in S_1 and call this priority s_1. Clearly, $s_1 \le s_2$. If $\frac{s_1}{s_2}$ exceeds or equals s, then we decide that the contribution of the priorities of the peaks in S_1 is much larger than that in S_2 and so S_1 constitutes a peak. *It is important to note that r and s must be chosen by the application developer.*

Figure 2 shows two examples of peaks corresponding to r, s values of $(6, 0.65)$ and $(4, 0.6)$ respectively. Dotted rectangles signify peaks, with s-values shown for the most significant peaks. As seen from the figure, peaks often occur in clusters. While the upper graph corresponds to wide ($r = 6$) peaks, parameters in the lower graph allow for narrower ($r = 4$) and slightly lower ($s = 0.6$ as opposed to $s = 0.65$) peaks. As result, the lower graph contains more peaks and smaller clusters.

Here is a simple algorithm that, given a sequence of video blocks and r, s values, will find all blocks that belong to (r,s)-peaks:

```
 Algorithm Peaks($v$,$r$,$s$)
      $v$ is a sequence of block-priority pairs
      $r$ is the peak width
      $s$ is the peak height
begin
      $Res := \emptyset$
      for each $j \in [r, card(v) - r]$ do
            $center := 0$
            $total := 0$
            for each $\langle b_i, p_i \rangle \in v$ such that $i \in (j-r, j+r]$ do
                  $total := total + p_i$
            end for
            for each $\langle b_i, p_i \rangle \in v$ such that $i \in (j-\frac{r}{2}, j+\frac{r}{2}]$ do
                  $center := center + p_i$
            end for
            if $\frac{center}{total} \ge s$ then
                  $Res := Res \cup \{\langle b_i, p_i \rangle \in v \mid i \in (j-\frac{r}{2}, j+\frac{r}{2}]\}$
            end if
      end for
      return $Res$
end
```

The $Peaks()$ algorithm slides a $2r$-wide window along a sequence of blocks, computing the total sum of block priorities in that window ($total$). It then computes the sum of block priorities in a narrower r-wide window in the middle of the $2r$-wide window ($center$). When the ratio of these

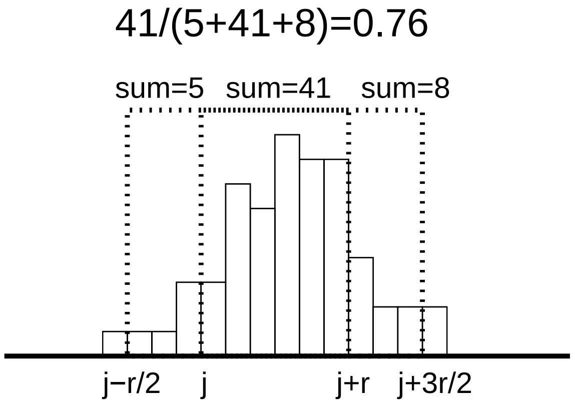

Figure 3: Peaks() Algorithm Analyzing a Peak

two sums $\frac{center}{total}$ exceeds the threshold s, all blocks in the r-wide window are picked as a *peak*.

EXAMPLE 3.1. *Consider the very small fragment shown in Figure 3. At some time, the $Peaks()$ algorithm will focus its window of length $2r$ on the segment from $j - \frac{r}{2}$ to $j + \frac{3r}{2}$ shown in the figure. It will compute the sum of the priorities of the blocks in the entire window of length $2r$ (which is $5 + 41 + 8 = 54$) as well as the sum of the priorities of the window of length r in the center of the window of length $2r$ - the priority there is 41. As a consequence, the ratio of these is $\frac{41}{54} = 0.76$. If 0.76 exceeds the s that the user has picked, then the sequence of blocks from j to $j + r$ is considered a peak.*

EXAMPLE 3.2. *Consider the 35 block sequence shown in Figure 4. We now describe how the $Peaks()$ algorithm finds the peaks in this figure. Suppose $r = 6$ and $s = 0.8$.*

- ***Window from** $1-12$: We initially start by looking at the first 12 blocks. The sum of the priorities of these 12 blocks is 59. If we look at the window of size 6 centered at the middle of the first 12 blocks (these are the blocks 4-9), the sum of the priorities is 36. The ratio, $\frac{36}{59}$ is below $s = 0.8$.*
- ***Window from** $2 - 13$: We now slide the window of length $2r$ one place to the right. At this time, the sum of the 12 block window is 60 and the sum of the 6 center blocks (blocks 5-10) is 44. The ratio is therefore $\frac{44}{60}$ which is below $s = 0.8$.*
- ***Window from** $3 - 14$: We now slide the window of length $2r$ one place to the right. At this time, the sum of the 12 block window is 58 and the sum of the 6 center blocks (blocks 6-11) is 48. The ratio is therefore $\frac{48}{58}$ which is greater than s. Therefore, blocks $6-11$ are returned as a peak.*

The algorithm continues in a similar fashion, finding peaks in blocks $18 - 23$ ($r = \frac{25}{29}$) and $28 - 33$ ($r = \frac{39}{48}$).

Note that the performance of the $Peaks()$ algorithm can be improved by avoiding computation of *center* and *total* iteratively in each iteration of the outer loop. After the first iteration of the outer loop, these values can be updated in

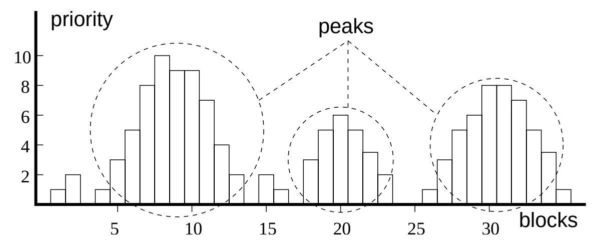

Figure 4: Result of Running Peaks() Algorithm

constant time. We have not included this optimization in the above algorithm as it complicates the simplicity of the algorithm, but it is easy to incorporate.
Complexity of $Peaks()$ algorithm. The $Peaks()$ algorithm has complexity of $O(r \cdot card(v))$ - hence it is linear with respect to the number of input blocks.

3.2 Block Merging Module

The peak identification algorithm eliminates all blocks that are not (r, s)-peaks for the r, s values selected by the application developer. Let $Peaks(v, r, s)$ be the set of all blocks from the original video that contain peaks. Consider the set $\{(b_i, b_{i+1}) \,|\, b_i, b_{i+1} \in Peaks(v)\}$ of all pairs of blocks that are adjacent to each other. In general when adjacent blocks are peaks, they may describe the same event. For example, in our soccer video summarization application, we may have one block (peak block) that describes a goal. The camera may have been switched to another block that also describes the same goal - however, the segmentation algorithm creating the blocks may treat these events as different events when in fact they describe the same event. The main goal of the block merging module is to merge adjacent blocks that may be very similar, so that repeating blocks can be treated as a single block in the later processing steps (such as resizing).

A *block similarity function* is a function *sim* that takes two blocks as input and returns a non negative real number as output. The smaller the number returned, the more similar the blocks are considered to be. There are many ways in which we could implement block similarity functions. Here are a few examples:

1. sim_{diff}: We could use any classical image differencing algorithm $idiff$ [11] to return the similarity between two frames and we could set the similarity between the two blocks to be the similarity between the two most similar frames, drawn from each block.
2. sim_{text}: In the event that the videos in question have an accompanying text transcript, we could identify the text blurb associated with each of the two blocks and set the similarity of the two blocks to be equal to the similarity between the two text transcripts using any classical method to evaluate similarities between text documents.
3. sim_{vec}: As is often common in image processing, we could associate a color and/or texture histogram with each block and return the similarities between the histograms using root mean squared distance or the L_1 metric [23].

To simplify the block merging process, let us assume that $Peaks(v, r, s)$ returns a set of block-priority pairs of the form $\langle b_i, p_i \rangle$, as opposed to a set of blocks, and adjacent blocks can be concatenated with the $\oplus$ operator. The block merging algorithm then takes as input, *any* block similarity function between blocks, together with a set of block-priority pairs, and returns a new set of merged blocks-priority pairs, as follows.

```
Algorithm Merge($v$,$sim()$,$d$)
    $v$ is a sequence of block-priority pairs
    $sim()$ is a similarity function on blocks
    $d$ is the merging threshold
begin
    $Res := \emptyset$
    $B :=$ first block-priority pair $\langle b_1, p_1 \rangle \in v$
    for each $\langle b_j, p_j \rangle, \langle b_{j+1}, p_{j+1} \rangle \in v$ do
        if $sim(b_j, b_{j+1}) \geq d$ then
            $B := \langle B.b \oplus b_{j+1}, B.p + p_{j+1} \rangle$
        else
            add $B$ to the tail of $Res$
            $B := \langle b_{j+1}, p_{j+1} \rangle$
        end
    end for
    add $B$ to the tail of $Res$
    return $Res$
end
```

The $Merge()$ algorithm considers all pairs of blocks b_j, b_{j+1}, concatenating them together into a bigger block $B.b$, as long as $sim(b_j, b_{j+1})$ value stays above the threshold d. The priority $B.p$ of the newly merged block is computed as the sum of individual priorities of its parts.

EXAMPLE 3.3. *Let us continue with Example 3.2. The peaks identified are* $6-11$*,* $18-23$*, and* $28-33$*. The* $Merge()$ *algorithm merges these blocks as follows. For the sake of this example, let us define* $sim(b_1, b_2) = 1 - \frac{|p_1 - p_2|}{p_1 + p_2}$*, where* p_1 *and* p_2 *are priorities of blocks* b_1 *and* b_2 *respectively, and set the threshold* $d = 0.9$*. Given these parameters, blocks* $8-10$ *will be merged into a single new block with* $p = 28$ *and so will blocks* $19-21$ *(*$p = 16$*),* $28-29$ *(*$p = 11$*), and* $30-32$ *(*$p = 33$*). Thus, the total number of blocks decreases from* 18 *to* 11 *after merging.*

Complexity of $Merge()$ algorithm. The $Merge()$ algorithm has linear complexity with respect to the number of blocks in its input.

3.3 Block Elimination Module

Suppose S is the set of blocks from the original video after the block merging step has been applied to the set of blocks in $Peaks(v, r, s)$. In the block elimination module, we would like to remove from this set all blocks whose priorities are less than a certain threshold. In addition, we would like to consider eliminating blocks that are repetitive. For example, in our soccer application, we may have replays of a goal long after the goal was scored. Both the original goal and the later replay may have high priorities, but our summary should probably not include both of them.

The block elimination module may use a similarity function similar to those used in the block merging module to first identify similar blocks. Any similarity function designated by the application developer may be used here. Blocks are grouped into equivalence classes w.r.t. similarity and for each equivalence class, one member is retained.

After removing repeatitions, our block elimination module computes the mean μ and standard deviation σ for the priorities of blocks in S. Given a real number $m \geq 0$, let us define a function $Drop(S, m)$ that drops from S all blocks whose priorities are less than $\mu - m\sigma$. $Drop()$ can be easily implemented by iterating over all blocks returned by the $Merge()$ algorithm. Thus, the result of

$$Drop(Merge(Peaks(v, r, s), d), m)$$

will be a set of all non-repeating high-priority merged peaks taken from v, with respect to the r, s, d, m parameters.

EXAMPLE 3.4. *Let us continue with Example 3.2 and assume that there are no repetitive blocks. The average priority of the peaks is* $\mu = \frac{122}{11} = 11$ *and the standard deviation is* $\sigma = \sqrt{\frac{1089}{11}} = 10.4$*. If we choose* $m = 0.25$ *and thus delete all blocks whose priorities are less than* 8.4*, the remaining blocks will be* $8-10$*,* $19-21$*,* $28-29$*, and* $30-32$*. Notice that these are merged blocks whose priorities have been bumped up during the merging process.*

3.4 Block Resizing Module

Even after eliminating some low-priority blocks in the previous step, the total frame count of the remaining blocks may still exceed the limit k imposed in the beginning of this paper. In such a case, we have to truncate some blocks to fit the limit. Clearly, blocks with higher priorities must have more prominence in the summary and thus occupy a larger percentage of frames. We then devise an algorithm that allocates to each block a number of frames proportional to its priority and truncates blocks to fit the limit of k frames.

```
Algorithm Resize($v$,$k$)
    $v$ is a sequence of block-priority pairs
    $k$ is the desired summary length
begin
    $Res := \emptyset$
    $p_{total} := \sum_{\langle b,p \rangle \in v} p$
    $p' := 0$
    $k' := 0$
    for each $\langle b, p \rangle \in v$ do
        if $len(b) \leq \frac{p \cdot k}{p_{total}}$ then
            $Res := Res \cup \{\langle b, p \rangle\}$
            $v := v \setminus \langle b, p \rangle$
            $p' := p' + p$
            $k' := k' + len(b)$
        end if
    end for
    $p_{total} := p_{total} - p'$
    $k := k - k'$
    for each $\langle b, p \rangle \in v$ do
        $alloc := round(\frac{p \cdot k}{p_{total}})$
        $b' := b$ truncated to $alloc$ frames
        $Res := Res \cup \{\langle b', p \rangle\}$
    end for
    return $Res$
end
```

EXAMPLE 3.5. *Let us continue with Example 3.2. There are four blocks that have survived merging and elimination:* $8-10$ *(*$p = 28$*),* $19-21$ *(*$p = 16$*),* $28-29$ *(*$p = 11$*), and* $30-32$ *(*$p = 33$*). Notice that all four are merged blocks, hence there are ranges instead of single numbers. Assuming that each "original" block corresponds to a single frame and the user requested a summary of* 5 *frames, let us see what the* $Resize()$ *algorithm does to our summary. First of all, given the total priority* $p_{total} = 88$*, all blocks will have to be resized. Block* $8-10$ *has an allocation of* $\frac{5 \cdot 28}{88} = 1.59$ *frames. As we cannot split frames, this block has to be truncated to two frames* $8-9$*. Block* $19-21$ *has an allocation of* $\frac{3 \cdot 16}{60} = 0.8$ *and therefore gets truncated to a single frame* 20*.*

By repeating this process, we also obtain frames 28 *and* 31. *Thus, the final summary is made of frames* 8, 9, 20, 28, 31.

4. IMPLEMENTATION

Our prototype system, implemented to evaluate the efficiency and effectiveness of PCA , consists of a database storing an annotated collection of soccer match videos, an implementation for the PCA summarization algorithm, as well as CPRgen, CPRdyn, and SEA algorithms [7], and a user interface for specifying desired summary content. The system is capable of automatically segmenting video into shots and detecting events, for annotation purposes.

Shot boundary detection is the first step in video processing. In spite of the long research history, it has not been completely solved yet. Sports video is arguably one of the most challenging domains for robust shot boundary detection due to i) strong color correlation between shots, due to a single dominant background color (soccer field, etc.), ii) large camera and object motions, and iii) cuts and gradual transitions (such as fades and dissolves) often present in sports video clips.

To detect shots in soccer videos, we have adapted an algorithm from [4], based on the observation that frames belonging to the same shot are more similar than frames from different shots. The similarity is computed with respect to color, texture, and shape features, measured at the *focus of attention* (FOA) spots of the frames.

Automatic event detection in soccer videos is an open problem actively addressed by several sports institutions. In this work, we are interested in a simple detection of such events as "goal", "celebration", "yellow card", and "red card". To detect these events, we have adopted a feed-forward neural network with a back-propagation algorithm [5] that is fed information about focus of attention, color histogram, texture, shape, and sound volume changes.

Note that in order to summarize a video, we can use data structures such as those in AVIS [1] to determine what activities and objects occur in frames. This will clearly speed up the algorithms within the PCA model.

5. EXPERIMENTS

In this section, we describe a set of experiments conducted to evaluate the performance of the PCA system. We compare the PCA algorithm to the CPRgen, CPRdyn, and SEA algorithms proposed in [7], both in terms of the time spent to compute a summary and the quality resulting summaries. As the only way of evaluating quality of summaries is via human subjects, we used a group of 200 students from the University of Naples.

Our data set consists of about 50 soccer videos, totaling about 80 hours. The videos were segmented into blocks and annotated, as described in section 4. The resulting blocks have an average length of about 10 seconds, with a relatively low variance.

Performance: To assess performance, we fixed the desired length of the summary to 60 seconds. We then varied the number of candidate blocks in the 4-75 range, by choosing an increasing number of events and subjects of interest.

The processing times were computed for each algorithm by averaging the results of 10 executions for each video. Figure 5 shows times taken by different algorithms. From this figure, we can conclude that the PCA algorithm outperforms

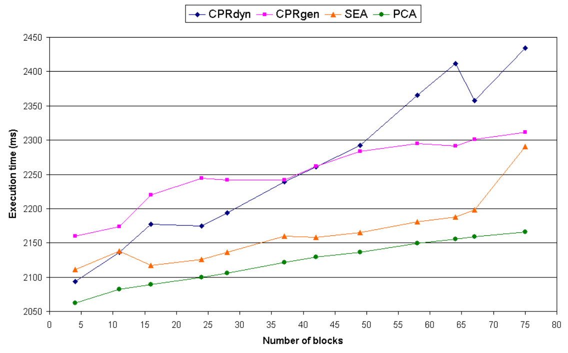

Figure 5: Summary Creation Times

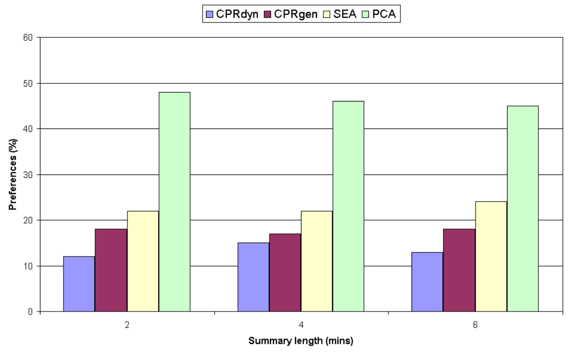

Figure 6: Summary Quality Ratings

the other three algorithms. This is true even without using the optimization for $Peaks()$ mentioned earlier.

Quality: To assess the quality of results produced by the four algorithms being compared, we asked a group of approximately 200 students at the University of Naples to rate the resulting summaries on a 1 to 5 scale. The experiment was repeated three times, with desired summary lengths of 2, 4, and 6 minutes, for all videos. The results, shown in Figure 6, indicate that summaries produced by the PCA algorithm have been rated best in 48%, 46%, and 45% of all cases respectively. These percentages are significantly better than those for the other three algorithms.

6. RELATED WORK

Summarizing video content is important for several applications including archiving and providing access to video teleconferences, video mail, news broadcasts, security videos, etc. The approaches to video summarization fall in two broad categories:

- *Reasoning Based Summarization:* Reasoning based approaches use logic or neural algorithms to detect certain combinations of events based on the information from different sources (audio, video, natural language). Examples of such approaches are video skims from the Informedia Project by Smith and Kanade [15] and movie trailers from the MoCA project by Lienhart et al [14]. Sometimes multiple characteristics of a video stream are employed simultaneously: the video analysis is combined with the audio analysis (speech, music, noise, etc.) and even with the textual information contained in closed captions. The heuristics, used to identify video segments of interest with respect to all these characteristics, are encoded with logical rules or neural networks.

- *Measure Based Summarization:* Measure based approaches use various importance and similarity measures within the video to compute the relevance value of video segments or frames. Possible criteria include duration of segments, inter-segment similarities, and combination of temporal and positional measures. These approaches can be exemplified by the use of SVD (Singular Value Decomposition) by Gong and Liu [10], or the shot-importance measure by Uchihashi and Foote [8].

It is worth noting that most systems summarize video by key-frame extraction. For example, the *Video Skimming System* [15] finds key frames in documentaries and news-bulletins by detecting important words in the accompanying audio. Systems like *MoCA* [14] compose film previews by picking special events, such as zooming of actors, explosions, shots, etc. Finally, Yahiaoui, Merialdo et al [20] propose an automatic video summarization method in which they define and identify what is the most important content in a video by means of similarities and differences among videos. They also suggest a new criterion to evaluate the quality of the summaries that have been created, through the maximization of an objective function. In contrast to the previous work discussed above, our paper introduces a more general framework which takes into account user content preferences (via block priorities) and produces continuous summaries while avoiding repetition. In addition, our method is not limited to a certain type of videos, but general enough to address many different classes of videos.

7. CONCLUSIONS

There is growing interest in summarizing video. Commercial enterprises with large video banks such as the US NBA or NCAA sports organizations, as well as military organizations deploying Predator and other video, have huge amounts of interest in identifying and summarizing video.

In this paper, we have proposed a new architecture for creating video summaries. We have introduced the novel concept of a block priority curve and shown how the peaks in this curve can be used to create video summaries. Unlike prior work in video summarization that we are aware of, our approach is not limited to selecting key frames, but attempts to maximize priority, continuity, and non-repetition of the video summary. We have conducted detailed experiments which clearly show that the proposed PCA algorithm is faster and produces much better ummaries than our previous algorithms described in [7].

Acknowledgements. This work was supported in part by the Army Research Lab under contracts DAAL0197K0135 and DAAD190320026, the CTA on Advanced Decision Architectures, by ARO contracts DAAD190010484 and DAAD190310202, by DARPA/RL contract number F306029910552, and by NSF grants 0205489 and IIS0329851.

8. REFERENCES

[1] S. Adali, K.S. Candan, S.-S. Chen, K. Erol, and V.S.Subrahmanian. *Advanced Video Information Systems.* ACM Multimedia Systems Journal, Vol. 4, 1996, pp. 172-186.

[2] N. Ancona, G. Cicirelli, A. Branca, and A.Distante. *Goal Detection in Football by Using Support Vector Machines for Classification.* Proc. Int. Joint Conference on Neural Networks, Vol. 1, 2001, pp. 611-616.

[3] S. Ayub and P. Bonissone. *Goal Recognition in Complex Domains.* IEEE Int, Conf. on Systems, Man, and Cybernetics, Vol. 2, 1994, pp. 1409-1414.

[4] G. Boccignone, A. Chianese, V. Moscato, and A. Picariello. *Foveated Shot Detection for Video Segmentation.* to be published in IEEE Trans. on Circuits and Systems for Video Technology, 2004.

[5] A. Chianese, R. Miscioscia, V. Moscato, S. Parlato, and A. Picariello. *A Fuzzy Approach to Video Scene Detection and Its Application For Soccer Matches.* to be published in IEEE Intelligent Systems Design and Applications, Budapest, August 2004.

[6] D. DeMenthon, D.S. Doermann, and V. Kobla. *Video Summarization by Curve Simplification.* Proc. ACM - Multimedia, Bristol, England, 1998, pp. 211-218.

[7] M. Fayzullin, A. Picariello, M.L. Sapino, and V.S. Subrahmanian. *The CPR Model for Summarizing Video.* ACM Workshop on Multimedia Databases, New Orleans, 2003.

[8] J. Foote and S. Uchihashi. *Summarizing Video Using a Shot Importance Measure and a Frame-Packing Algorithm.* Proc. of the Int. Conf. on Acoustics, Speech, and Signal Processing, Phoenix, 1999, Vol. 6, pp. 3041-3044.

[9] U.Gargi, R. Kasturi, and S.H. Strayer. *Performance Characterization of Video-Shot Change Detection Methods.* IEEE Trans. on Circuits Systems Video Technology, Vol. 10(1), 2000, pp. 1–13.

[10] Y. Gong and X. Liu. *Video Summarization Using Singular Value Decomposition.* Proc. of Computer Vision and Pattern Recognition, 2000, pp. 174-180.

[11] R. C. Gonzales and P. Winz. *"Digital Image Processing"* Addison-Wesley Publishing Company, Knoxville, Tennesee, 1987.

[12] A. Hanjalic. *Shot-Boundary Detection: Unraveled and Resolved?* IEEE Trans. Circuits Systems Video Technology, Vol. 12, 2002) pp. 90-105.

[13] L. He, E. Sanocki, A. Gupta, and J. Grudin. *Auto-Summarization of Audio-Video Presentations.* ACM Proc. on Multimedia, 1999, pp. 489-498.

[14] R. Lienhart, S. Pfeiffer, and W. Effelsberg. *The MoCA Workbench: Support for Creativity in Movie Content Analysis.* Proc. IEEE Conf. on Multimedia Computing and Systems, Hiroshima, Japan, 1995, pp. 314-321.

[15] T. Kanade, M. Smith, S. Stevens, and H. Wactlar. *Intelligent Access to Digital Video: The Informedia Project.* IEEE Computer, Vol. 29(5), 1996, pp. 46-52.

[16] D. Li and H. Lu. *Model Based Video Segmentation.* IEEE Trans. Circuits Systems Video Technology, Vol. 5, 1995, pp. 533-544.

[17] H.Martin and R.Lozano. *Dynamic Generation of*

Video Abstracts Using an Object Oriented Video DBMS. Networking and Information Systems Journal, Vol. 3(1), 2000, pp. 53-75.

[18] H.R. Naphide and T.S. Huang. *A Probabilistic Framework for Semantic Video Indexing, Filtering, and Retrieval.* IEEE Transactions on Multimedia, Vol. 3(1), 2001, pp. 141-151.

[19] E. Oomoto and K. Tanaka. *OVID: Design and Implementation of a Video-Object Database System.* IEEE TKDE (Multimedia Information Systems), Vol. 5(4), 1993, pp. 629-643.

[20] I. Yahiaoui, B. Merialdo, and B. Huet. *Generating Summaries of Multi-Episode Video.* IEEE Int. Conf. on Multimedia and Expo, 2001, pp. 22-25.

[21] C. Stauffer and E. Frimson. *Learning Patterns of Activity Using Real-Time Tracking.* IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 22(8), 2000, pp. 747-757.

[22] V.S. Subrahmanian. *"Principles of Multimedia Database Systems"* Morgan Kaufmann, 1998.

[23] M.J. Swain and D.H. Ballard. *Color Indexing.* Int. Journal of Computer Vision, Vol. 7(1), 1991, pp. 11-32.

[24] V. Tovinkere and R. J. Qian. *Detecting Semantic Events in Soccer Games: Towards a Complete Solution.* IEEE Int. Conf. on Multimedia and Expo, 2001, pp. 833-836.

[25] B.T. Truong, C. Dorai, and S. Venkatesk. *New Enhancements to Cut, Fade, and Dissolve Detection Processing Video Segmentation.* ACM Multimedia, 2000, pp. 219-227.

[26] D. Zhong and S.-F. Chang. *Video Object Model and Segmentation for Content-Based Video Indexing.* IEEE Int. Conf. on Circuits and Systems, Hong Kong, 1997.