Visual Information Analysis for big-data using multi-core technologies.

Nikolaos Mpountouropoulos, Anastasios Tefas, Nikos Nikolaidis and Ioannis Pitas

Artificial Intelligence and Information Analysis Laboratory, Department of Informatics,
54124 Thessaloniki, Greece
{tefas,nikolaid,pitas}@aiia.csd.auth.gr
http://www.aiia.csd.auth.gr/

Abstract. The exponential growth of video data produced by surveillance cameras, cell phones and movie post-production creates the need to process big-data using methods that are able to produce instantaneous result. Video summarization can be accomplished and represented in several manners. The achieved summaries might be a sequence of images or short videos. In our method, an input video is divided into shots. From each shot we calculate key frames using three different key frame definitions, to summarize the video data. The contribution of this paper is to describe how to incorporate techniques that extract *on the fly* results.

Keywords: video summarization, big-data video analysis,mutual information

1 Introduction

Videos are structured according to a descending hierarchy of scenes, shots, frames. Frame is the fundamental unit of a video. A shot is a consecutive sequence of frames captured by a static camera or a moving one. A scene is a sequence of shots. A scene is defined as a collection of one or multiple shots focusing in an object or objects that motivate our interest. Video summarization is the process of detecting the most important and informative frames of a video in order to create a shorter version of it that is still able to convey the original message. The representative frames are called key frames. The importance of video summarization has become really apparent in now days, due to the exponential growth of video production and consumption over the internet and security applications. Only at youtube.com 100 hours of video are uploaded every minute.

Generally shot-cut corresponds to an abrupt frame change. Shot-cuts are classified in two major categories [1]. In the first category belong the cases where transitions between the frames are abrupt and the second one includes the cases of gradual transitions, such as fade in/fade out, dissolve,wipe etc [2]. Shot-cut

detection is easier to detect in an abrupt change rather than in a gradual transition. In the case of dissolve the frames of the shot in a video start to fade out while the next appears and grows clearer as the first one dims. The wipe happens when one shot replaces another in a different spatial regions of the intermediate video frames. The former frames grow using a pattern ,e.g. like a star, of a special shape until it entirely replaces the latter. A fade in/fade out is a gradual disappearance of a frame into black and then the black frame fades into the appearing shot.

Generally shot-cut detection algorithms work by extracting features from frames, then using a similarity measure to detect them. Features used for shot-cut detection include color histogram [5],[6], block color histogram, motions vectors, etc. To measure the similarity between frames using the extracted features is the second step. The similarity metrics can be the Euclidean distance, the chi-squared similarity , mutual information, etc.

We studied many standard techniques for detecting shot-cuts and key-frames. We decided to adopt MI (Mutual Information) for shot-cut detection as mentioned in [3]. Using MI values we determine the shots. Key frames are extracted from these shots using three different key frame definitions. In Section 2 we briefly describe the adopted shot cut detection algorithm. In Section 3, we describe the key frame selection process. Experimental results are provided in Section 4. Conclusions are drawn in Section 5.

2 MI: Mutual Information

2.1 Definitions and Background

In probability theory the mutual information of two discrete random variables is a measure that evaluates the mutual dependence of the two. Let X be a discrete random variable with a set of outcomes $A_X = \{a_1, a_2, \ldots, a_N\}$ with the corresponding probabilities $\{p_1, p_2, \ldots, p_N\}$. $p_x(x = a_i) = p_i, p_i \geq 0$ and $\sum_{x \in A_x} p_X(x) = 1$. Entropy of X measures "unpredictability" and can defined as:

$$H(X) = -\sum_{x \in A_X} p_X(x) \log p_X(x) \tag{1}$$

The *joint entropy* of two discrete random variables X, Y can be obtained by:

$$H(X, Y) = -\sum_{x,y \in A_X, A_Y} p_{XY}(x, y) \log p_{XY}(x, y) \tag{2}$$

where $logp_{XY}(x, y)$ is the joint probability density function for the random variables X, Y. The *conditional entropy* of Y given X (or X given Y accordingly) is expressed as:

$$H(Y|X) = \sum_{x \in A_X} p_X(x) H(Y|X = x) = - \sum_{x,y \in A_X, A_Y} p_{XY}(x,y) \log p_{XY}(x|y) \tag{3}$$

The *mutual information* (MI) of the two variables X and Y is defined by:

$$I(X,Y) = - \sum_{x,y \in A_X, A_Y} p_{XY}(x,y) \log \frac{p_{XY}(x,y)}{p_X(x) p_Y(y)} \tag{4}$$

According to [3] $I(X,Y)$ in (4) is equivalent to

$$I(X,Y) = H(X) - H(X|Y) \tag{5}$$

In our experiments, we have adopted the MI definition in (5).

2.2 Shot-cut detection using Mutual Information

In our approach the MI is calculated separately for each one of the RGB components. We normalize each RGB pixel luminosity to gray level from $0, \ldots, N-1$. At frame f_t three vectors with dimension N created containing the values on gray level for each pixel. Dividing each element by total number of the pixels of the frame and averaging them gives $p_X(x)$ where $A_X = \{0, 1, \ldots, N-1\}$ contains corresponding probability of each gray-scale pixel. Thus we can calculate the entropy $H(X)$ using (1).

Between two frames f_t, f_{t+1} three $N \times N$ matrices, $C^R_{t,t+1}, C^G_{t,t+1}$ and $C^B_{t,t+1}$ created containing information on the gray-level transitions between these frames. Each one of the components for example $C^B_{t,t+1}(i,j)$ with $0 \leq$i$\leq N-1$ and $0 \leq$j$\leq N-1$ corresponds to the occurrence that a pixel with gray level i in frame f_t has gray level j in the frame f_{t+1}. Dividing by the total number of the pixels of the video frame we find the joint probability in each matrix $C^{JP_R}_{t,t+1}$, $C^{JP_G}_{t,t+1}$, and $C^{JP_B}_{t,t+1}$, that a pixel with gray level i has now gray level j. For two frames f_t, f_{t+1} the matrix of gray scale joint probability is given by:

$$C^{JP}_{t,t+1} = (C^{JP_R}_{t,t+1} + C^{JP_G}_{t,t+1} + C^{JP_B}_{t,t+1})/3 \tag{6}$$

Using the expression in (6) we can calculate the conditional entropy mentioned at (3). Finally we use the expression (5) to calculate the $I(X,Y)$ for the two frames.

In order to detect shot-cuts we define a temporal window W size of N_W. Local MI mean values, and the standard deviation of them excluding the current value $I_{t,t+1}$ at the current window t_c is described at [8]:

$$\overline{I}_{t_c} = \frac{1}{N_W} \sum_{t \in W, t \neq t_c} I_{t,t+1} \tag{7}$$

If the quantity $\overline{I}_{t_c} - I_{t,t+1}$ is greater than the double of the standard deviation of the selected values a shot-cut is detected.

2.3 Fast implementation of the algorithm

In order to reduce the processing time required for this task, we have devised a parallel (multi-threaded) implementation of the original algorithm. We decided to automatically split the video sequences in blocks (non-overlapping subvideos) and process each block independently in different threads. Furthermore, we have designed a parallel video reading/decompresing implementation. The adoption of this operation, further improved the computational speed of the algorithm.

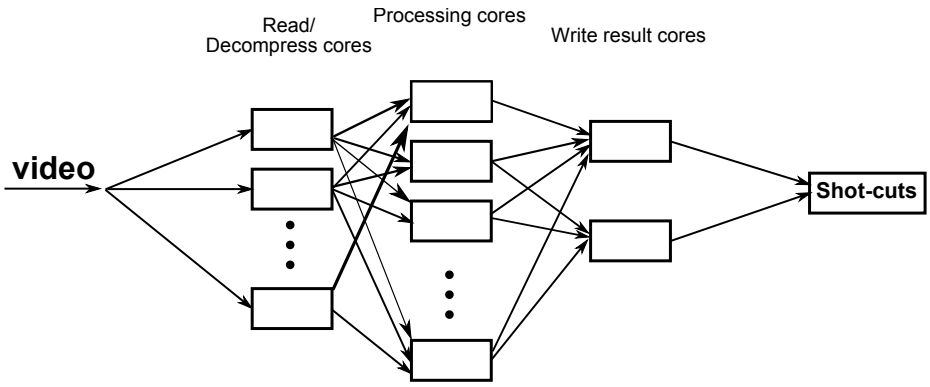

Fig. 1. Processing Cores

The first step for developing this algorithm was to evolve the *OpenCV* library. We created a new multi-threading method which can read and decompress the frames from the compressed video file using all the cores of the current working station. This way we can define the buffer of the processing frames per core. For example a buffer value equal to 60 means that each core will decompress 60 frames and write them to memory. Each one of these video-blocks is assigned to a thread responsible to calculate the MI from one frame to another. We define a matrix with size equal to the length of the movie.When the threads complete the calculations, each thread writes to the corresponding position of the matrix, the results of the corresponding calculations. Combing all these results using an adaptive thresholding approach we can define the shots of the video.

3 Key frame Selection

The data entry for this algorithm can be composed of the shots defined from the previous method. For each shot a key frame will be selected. Key frames are intended to be informative regarding the corresponding shot. According to this, a key frame is defined as the one that provides the smallest distance from the remaining frames in the shot. Three algorithm implementations are available. These are: *distance between frames*, *distance from the average frame* and distance between *frame histograms*.

All three of them need to compute distances between frames. For the two first algorithms, the distance between two frames is the sum of all their corresponding (having same coordinates) pixel distances. Pixel distances can be computed by two methods using the distance of the *average* or the *euclidean* (9) pixel distance. The *average* distance can be defined, where $p_{R_t}, p_{G_t}, p_{B_t}$ is the pixel color RGB values of the current frame at position (x, y) and $p_{R_{t+1}}, p_{G_{t+1}}, p_{B_{t+1}}$ the next frame f_{t+1}, at the same position as:

$$d_{avg} = (|p_{R_t} - p_{R_{t+1}}| + |p_{G_t} - p_{G_{t+1}}| + |p_{B_t} - p_{B_{t+1}}|)/3 \tag{8}$$

The *average* pixel distance is the distance of their average values based on the RGB values of the pixel. The *euclidean* distance of two pixels is given by (9). The third algorithm calculates the distance between the histograms using correlation, chi-square, intersection and Bhattacharyya distance. Details on these metrics can be found in [4].

$$d_{euc} = \sqrt{(p_{R_t} - p_{R_{t+1}})^2 + (p_{G_t} - p_{G_{t+1}})^2 + (p_{B_t} - p_{B_{t+1}})^2} \tag{9}$$

3.1 Simple Distance of frames

The former algorithm initially computes the distance for each shot frame pair (that is for frame pairs $1-2, 1-3, \ldots, 2-1, 2-3, \ldots$) where the distance between two frames is defined as the sum of their corresponding (having same coordinates) pixel distances, as mentioned above. Let $x_i \in \mathbb{R}^N, i = 1, \ldots, M$ be the video frames in vectorized form. Let s_j be the key frame for the j-th shot.According to this we have:

$$s_k = \arg\min_i \sum_j \|x_i - x_j\|^2 \tag{10}$$

After all distances among shot frames are computed, the key frame can be derived as the one that has the smallest sum of frame distances, meaning that is the one closest to most other shot frames. The initial implementation of the algorithm was reading-decompressing two frames from the shot calculates their distance and stores it appropriately. Thus the complexity of reading-decompressing and comparison was $O(n^2)$. We make two significant improvements. We create a multi-thread version of the algorithm where each shot is being process by one core. Since the shots are unequal, when a thread finishes a new shot is being

reassign to it. The reading-decompressing of each shot is done exactly only one time assuming that the working station has enough memory to fit in the shots being processed.

3.2 Average Frame

The second algorithm is computationally faster ($O(n)$ instead of $O(n^2)$ of the first method) but less accurate. We can average the frames of the shot to calculate the average frame of the shot by computing the average value of all corresponding frame pixel luminosities. The distance from the average frame is calculated by the Euclidean distance or the average distance as seen:

$$s_k = \arg\min_i \|x_i - x_{avg}\|^2 \tag{11}$$

where x_{avg} is the "avegare" frame.

After that, each frame is compared with the average frame of the shot, by using any type of distance mentioned before. In this case, key frames could be the ones that are the most similar (least distance) to the average ones. This is by far the fastest algorithm of the two but slightly less accurate. The complexity is $O(n)$.

3.3 Frame Histogram Distance

This algorithm follows a similar process to first to produce its key frames, with the only difference being that the distance between two frames in this algorithm is not the sum of their corresponding pixel distances, but the distance of their histograms in RGB color space. This algorithm is the most context sensitive of the three, and can yield drastically different results from the first two. The third algorithm we use is distance computation in a histogram level rather than pixel level, which is more robust to noise and camera movement. The distance function we use is correlation, chi-square, intersection and Bhattacharyya.

4 Experimental results

In this section we describe experiments conducted in order to illustrate the decrease of execution time observed by our parallel implementation. We set the buffer values to 20, 40, 60 frames and change the number of threads for the MI algorithm as seen at Tables 2,3,4. The experiments were conducted on an Intel(R) Core(TM) i7-4770 CPU 3.40GHz PC which has 4 physical cores and can manipulate 8 logical threads. Each core processes a block of frames and writes the result to the corresponding position of a matrix. When the threads finish, the matrix contains the MI values for each pair of frames. We used Hollywood movie *Movie1* for the experiments of MI as seen at Table 1. Using four physical cores the ideal reduction of time will be 75%. According to [7] using the Hyper-Threading Technology we gain a little more time archiving at 79,72% setting the

buffer to 60. The rest videos used as input were other Hollywood movies as seen at Table 1. The metrics for the distance algorithms *simple frame distance,average frame* was the average distance of the pixels, while in *histogram distance frame* was correlation.

Table 1. Characteristic of movies

	Total no.Frames	Resolution	Duration
Movie1	181764	960x540	2h 6m 21s
Movie2	196224	960x540	2h 16m 24s
Movie3	150361	960x540	1h 44m 31s

Table 2. Experimental results(buf=20) for MI.

Number of Threads	1	2	4	6	8
Elapsed Time	34m 55s	20m 09s	12m 33s	10m 43s	9m 48sec
Decrease in time (%)		42,3%	64,1%	69,4%	72,1%

Table 3. Experimental results(buf=40) for MI.

Number of Threads	1	2	4	6	8
Elapsed Time	33m 09s	15m 36s	9m 36s	8m 27s	7m 28sec
Decrease in time (%)		53%	71,1%	74,6%	77,5%

The algorithms of key frame selection especially the *simple distance frame* and *Histogram Distance Frame* are slower to our standards and takes more than the real time of the movie to complete. As seen at Table 5 we managed to achieve an enormous reduction. Among the three of them the best time is the *Average Frame*. Although is not as robust as the other two the execution time is magnitude times faster than the other two as seen at Table(5). With *MI* and *Average Frame* can produce results on the fly.

We try to combine techniques for processing big-data video to accomplish instantaneously execution time. The fastest method for the selection of key frames, *average frame* might not be as robust as the others but completes calculation magnitude times faster. The next step is to examine algorithms to use those key frames to create a video summarization. The ideal is to adopt the proper algorithms for the creation of summary so that the total time of processing doesn't overrun the length of the movie.

Table 4. Experimental results(buf=60) for MI.

Number of Threads	1	2	4	6	8
Elapsed Time	33m 41s	13m 35s	8m 29s	7m 38s	6m 50sec
Decrease in time (%)		59,68%	74,82%	77,74%	79,72%

Table 5. Time needed for key-frame selection in Movie1.

	Simple Distance Frames	Average frame	Histogram Distance Frame
Single-core	1d 2h 52min 32sec	1h 3m 8sec	2d 1h 49min 33s
Multi-core	4h 59m 25s	32m 36s	9h 7m 11s

Acknowledgment

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant agreement number 316564 (IMPART). This publication reflects only the authors views. The European Union is not liable for any use that may be made of the information contained therein.

References

1. Weiming Hu, Nianhua Xie, A survey on visual content based video indexing and retrieval, IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, vol. 41, no. 6, pp. 797819, November 2011.
2. C. Cotsaces, N. Nikolaidis, and I. Pitas: Video shot boundary detection and condensed representation: A review, IEEE Signal Processing Magazine, vol. 23, no. 2, pp. 2837, March (2006).
3. Z. Cernekova I.Pitas C.Nikou: Information theory-based shot cut/fade detection and video summarization, IEEE Transactions on Circuits and Systems for Video Technology, vol.16, January, (2006).
4. Opencv metrics for histograms, `http://docs.opencv.org/modules/imgproc/doc/histograms.html?highlight=comparehist#comparehist`
5. S. W. S. HongJiang Zhang, Atreyi Kankanhalli: Automatic partitioning of full-motion video, ACM Multimedia Syst.,vol. 1, no. 1, pp. 1028, April (1993).
6. Z.Cernekova, C. Kotropoulos, I. Pitas: Video shot segmentation using singular value decomposition, SPIE Journal of Electronic Imaging, vol. 16, no. 4, December (2007).

Table 6. Shot-cuts and fastest method of key-frames

	Shot-cut using (MI)	Key frame selection using Average frame	Total Time
Movie1	6m 50s	32m 36s	39m 26s
Movie2	9m 33s	35m 13s	44m 46s
Movie3	6m 7s	27m 7s	33m 14s

7. Chen, Y. K., Holliman, M., Debes, E., Zheltov, S., Knyazev, A., Bratanov, S., ... & Santos, I.: Media Applications on Hyper-Threading Technology. Intel Technology, Journal 6, Issue(1) (2002).
8. I. Pitas and A. Venetsanopoulos, Nonlinear Digital Filters: Principles and Applications. Kluwer Academic, (1990).