Practical Advertisement Detection in Video

Du,Jiaen (taokayan13@gmail.com)

1. Background

Usually there are many advertisements in television. This is not efficient for video recording as the advertisement part takes a lot of extra space for storage. It is very good if all the advertisements can be removed in televisions video recording and storage automatically. Therefore, it is good for us to design a method to identify the advertisement parts in TV streams.

2. Types of video clip

We classify video segments into the following three types.

Movies: These usually last for approximately 15 minutes each time, 4 times per hour. We should minimize the number of news/movies that wrongly detected as advertisements.

Connections: There are typically several seconds of video connecting the advertisement part and news/movies part. This part is very useful and will be expanded further in the following sections.

Advertisements: These usually last for 3-4 minutes each time, 4 times per hour. We should maximize the number of extracted advertisements.

3. Repetition of advertisement

Here we use movie data from observations of "Hong Kong free-to-air TV". It is not rare to see an advertisement appearing more than one time during the course of a movie, but there are few movie segments that appear exactly the same. Hence we can detect advertisements that have repetition throughout hours of videos. Here we divide advertisement into two subtypes: advertisements that have repetition in a video, naming **repetitive advertisements**, and advertisements that have no repetition in a video, naming **non-repetitive advertisements**. In this project, we devised our algorithm for fast frame search and detection of advertisements and connections.

4. Practical advertisement detection

4.1. Detection of repetitive advertisements and connections

For a video, usually there are about 25 frames per second. Although movies with different fps exist, it would not affect the basic idea of our algorithm. For 3 hours of video, the number of frames rapidly become 25 x 60 x 60 x 3 = 270,000. Searching two similar frames by brute-force would cost 73,000,000,000 frames comparisons which is totally unacceptable. To eliminate the number of computation, we hash every frame into the hash table.

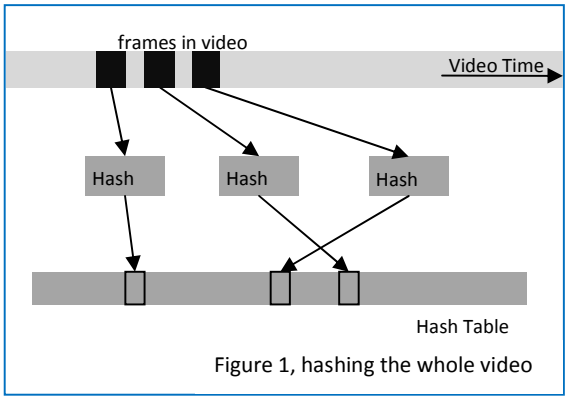


Figure 1, hashing the whole video

The way we devise the hash value of one frame is: we select four different regions with fixed positions and sizes, computing the gray level of each region. Then we normalize each gray level into $[0, levelmax - 1]$. Here we choose *levelmax* = $32 = 2^5$, using 5 bits of space to store one

gray level. Combining 4 gray levels we finally get a hash value.

$$Hashvalue = (graylevel(0) \ll 15) \parallel (graylevel(1) \ll 10) \parallel (graylevel(2) \ll 5) \parallel graylevel(3)$$

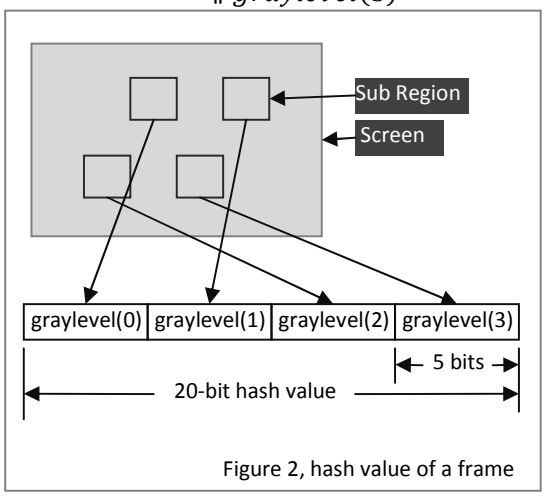


Figure 2, hash value of a frame

After doing hashing, similar frames appear in the same hash values.

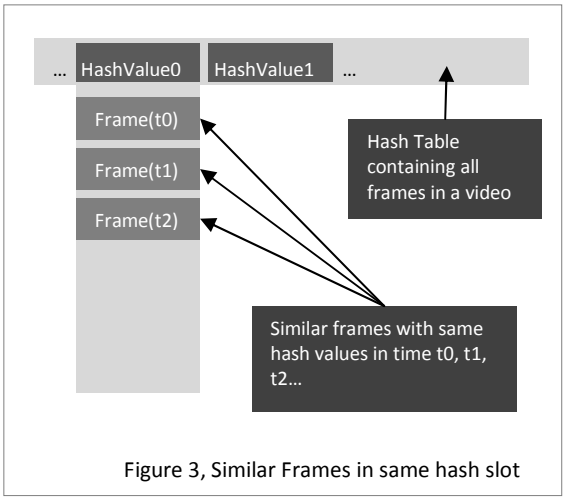


Figure 3, Similar Frames in same hash slot

Although similar frames will have the same hash value, it is not guarantee that any two frames in the same hash slot will be similar. Hash value can only reduce the computational complexity but how far we can improve our performance by using frame hashing totally depends on how good we devise our hash function. The best hash function should uniformly distribute all frames into different hash values.

For every two frames with the same hash value at time t0, t1 respectively, naming frame(t0), frame(t1), we aim to find the two similar video intervals that contain t0 and t1 respectively. To detect whether two frames are similar or not, we have devised the following frame similarity calculation method:

1). Divide the bitmap of the frame at time t into sub blocks. The more sub blocks we divide into, the more accuracy of image comparison we get, but also the more computational time and disk storage are required. To compromise time complexity are accuracy, we use 12 x 9 sub blocks to represent a frame, as most advertisements are at ratio of 4:3. Then we calculate average red, green, blue color of each block respectively; Denote them as *red(t,i)*, *green(t,i)*, *blue(t,i)* where t stands for the time of the frame and i is for block index.

2).For two frames at time t1 and t2, similarity is defined as follow:

$$Similarity(t1,t2) = 1 - (\alpha \frac{\sum_i \left(red(t1,i) - red(t2,i)\right)^2}{255^2 \times 12 \times 9} + \beta \frac{\sum_i \left(green(t1,i) - green(t2,i)\right)^2}{255^2 \times 12 \times 9} + \gamma \frac{\sum_i \left(blue(t1,i) - blue(t2,i)\right)^2}{255^2 \times 12 \times 9})$$

α, β, γ are weights of red, green, blue respectively, which satisfy $\alpha + \beta + \gamma = 1$; values of red, green, blue range from 0 to 255.

Finding two similar video intervals via similarity calculation:

We set a threshold of determining whether two frames are similar:

- If *Similarity*(t1,t2) >= *similarity_threshold*, frame(t1) and frame(t2) are similar.

- If *Similarity*(t1,t2) < *similarity_threshold*, frame(t1) and frame(t2) are not similar

In practice, we set *similarity_threshold = 0.999*

We start comparing frames *t0* and *t1*, then *t0+1* and *t1+1* , then *t0+2* and *t1+2*,…, until two frames are no longer similar. Then we get a pair of similar segments $[t0\ , t0 + \Delta t_{after}], [t1\ , t1 + \Delta t_{after}]$.

Again, we compare frames *t0-1* and *t1-1*, then *t0-2* and *t1-2* until two frames are no longer similar. $[t0 - \Delta t_{before}, t0], [t1 - \Delta t_{before}\ , t1]$.

Combining the former results we have a pair of similar segments $[t0 - \Delta t_{before}\ , t0 + \Delta t_{after}], [t1 - \Delta t_{before}\ , t1 + \Delta t_{after}]$.

The following graph illustrates our calculation.

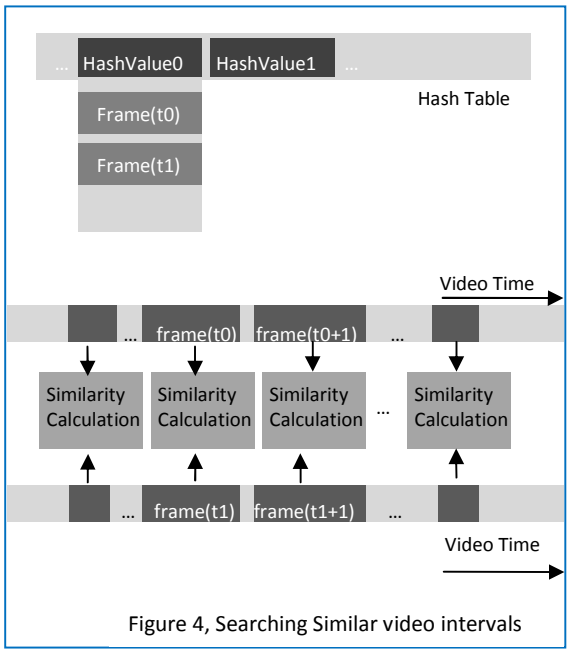


Figure 4, Searching Similar video intervals

Once we have the two similar video segments $[t0 - \Delta t_{before}\ , t0 + \Delta t_{after}], [t1 - \Delta t_{before}\ , t1 + \Delta t_{after}]$, we have find one pair of video repetition of length $\Delta t_{before} + \Delta t_{after}$ starting at time $t0 - \Delta t_{before}$, $t1 - \Delta t_{before}$ respectively. To further deduce what type these two video segments belong to, we use the following criteria:

- If $\Delta t_{before} + \Delta t_{after} < Minimum\_time\_of\_connection$, they belong to nothing, ignore them
- If $\Delta t_{before} + \Delta t_{after} > Minimum\_time\_of\_connection$, they belong to advertisements or connections

In this project, we set the value of $Minimum\_time\_of\_connection$ to be 6 sec.

4.2. Detection of non-repetitive advertisements

Advertisements that have no repetition will be more difficult to detect. Only ignore those advertisements will let the result accuracy drop dramatically. To detect this type of advertisement, we use the everyday knowledge that, each advertisement section in a video will last for 3-4 minutes inside which are all advertisements. Further more, if two advertisements have been detected using previous repetition searching methods, and if those two advertisements happen within a 3-4 minute time slot, we can conclude that all video frames between those two advertisements should also belong to advertisement part. (Illustrated in the following graph)

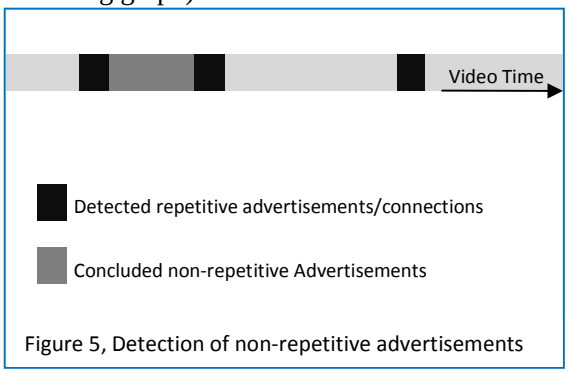


Figure 5, Detection of non-repetitive advertisements

We set a threshold max_adv_time, representing the maximum time interval of consecutive advertisements. We conclude a time interval to be a non-repetitive advertisement time interval if:

- Length of interval < max_adv_time
- There is a repetitive advertisement/connection right before the interval
- There is a repetitive advertisement/connection right after the interval.

In practice, we set $max_adv_time$to be 200 seconds.

5. Tests and Results

Movie A
Source: Pearl Channel, TVB, HK
Length: 3 hours 45 minutes

Test Setup:

Test 1	First 45 minutes of movie A
Test 2	First 1 hour of movie A
Test 3	First 1 hour 45 minutes of movie A
Test 4	First 2 hours 35 minutes of movie A
Test 5	The whole movie A

System Configuration:
Windows XP SP2;
AMD Athlon64x2 TK-57 at 2300MHz
Integrated Geforce 7150 / nForce 630M
1G RAM
120G Hard disk.

Test Results

	Test 1	Test 2	Test 3	Test 4	Test 5
Length of detected ads.	0	4 mins 45 secs	17 mins 10 secs	29 mins 44 secs	46 mins 13 secs
Length of actual ads.	10 mins 32 secs	14 mins 2 secs	26 mins 37 secs	37 mins 57 secs	54 mins 24 secs
Length of false detection	0	0	0	0	0
Precision	##	100%	100%	100%	100%
Recall	0%	34%	64%	78%	85%

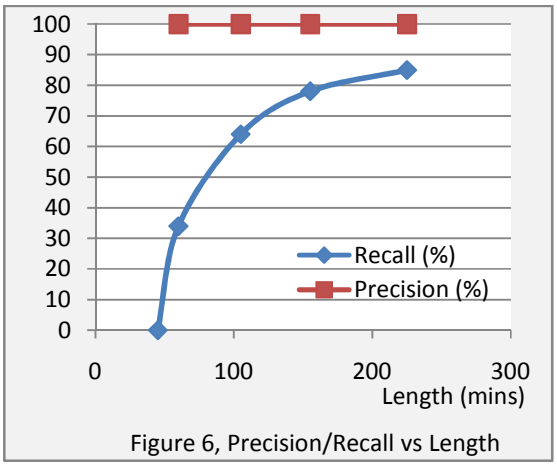


Figure 6, Precision/Recall vs Length

6. Conclusions

This method works in the situation that advertisement repetition exists. Non-repetitive advertisements can be estimated in the knowledge of detected repetitive advertisements. If more videos are analyzed, the number of repetitive advertisement will increase, while the number of non-repetitive advertisements will reduce, resulting in a higher success rate we can archive.

7. References

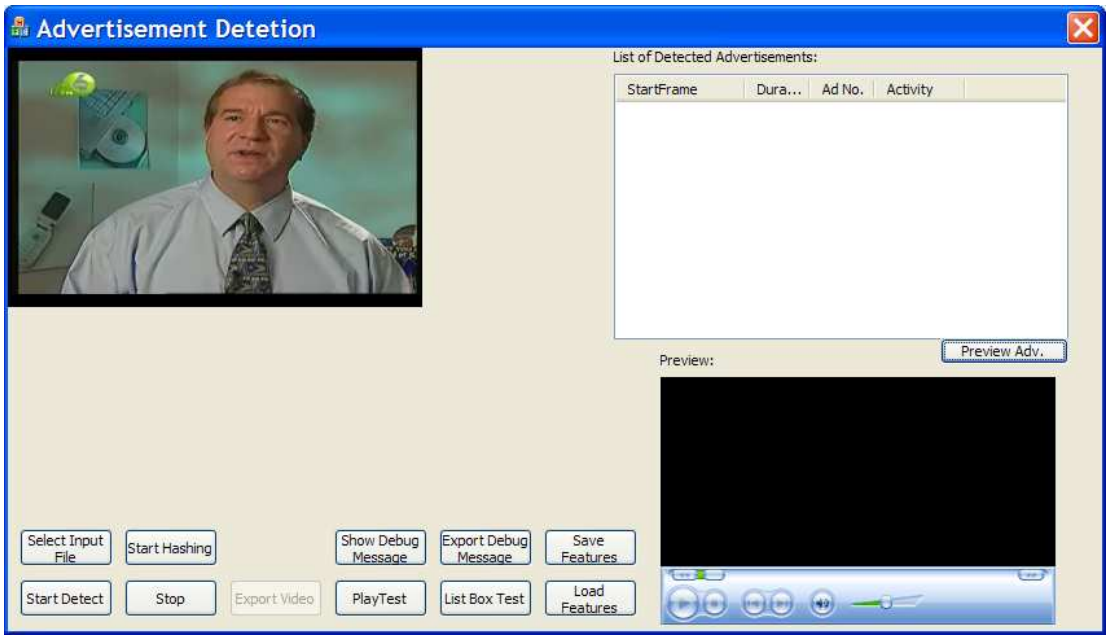
[1]. Advertisement Detection and Replacement using Acoustic and Visual repetition, Michele Covell, Shumeet Baluja, Michael Fink, IEEE Workshop on Multimedia Signal Processing, Victoria BC, October 2006.
[2]. Identification of New Commercials using Repeated Video Sequence Detection, John M. Gauch, Abhishek Shivadas, Image Processing, ICIP 2005. IEEE International Conference, 2005
[3]. Image Similarity, COMP343 Multimedia Computing, David Rossiter, HKUST

[4]. A comparison of measures for visualizing image similarity, Kerry Rodden, Wojciech Basalaj, David Sinclair, Kenneth Wood, Challenge of Image Retrieval, Brighton, 2000
[5]. A flexible search-by-similarity algorithm for content-based image retrieval, J.Fournier, M.Cord, International Conference on Computer Vision, Pattern Recognition and Image Processing (CVPRIP 02) - March 2002
[6]. http://msdn.microsoft.com/en-us/library/ms783323.aspx, DirectShow, Microsoft Development Network, MSDN, 2009 Microsoft Corporation
[7].http://www.codeproject.com/KB/audio-video/framegrabber.aspx, Extracting bitmaps from movies using DirectShow, CodeProject, Sep 2001
[8].http://msdn.microsoft.com/en-us/library/ms783752(VS.85).aspx, Grabbing a Poster Frame, MSDN, 2009 Microsoft Corporation

8. Appendix

Our software for advertisement detection is implemented in Visual C++ 2008 with Microsoft Foundation Classes (MFC) and DirectShow API.

Screenshot of software



System requirement (recommend)
CPU: Intel Core 2 Series or AMD Athlon64 Series or above;
Windows XP SP2 or above, with 512MB memory;

Supported Input format
AVI, WMV

Using the software
Step 1, click "*Select Input File*" and select a video file;
Step 2, click "*Start Hashing*". It will take about several minutes hashing all the frames, depending on the length of the corresponding video. Click "Stop" for terminating the hashing process.

Step 3, click “*Start Detect*”. Detection will soon start and be finished in about several seconds to several minutes.

Resulting detected advertisements (including repetitive and non-repetitive) will appear in the upper right list once detection has finished, containing start time, duration and other attributes of each detected advertisement.

By double clicking one advertisement in the list box, the corresponding advertisement will be played in the preview windows.