Real Time Commercial Detection in Videos

Zheyun Feng[*]
Comcast Lab, DC/Michigan State University
fengzheyun@gmail.com

Jan Neumann
Comcast Lab, DC
Jan_Neumann@cable.comcast.com

Abstract

In this report, we describe the project of real time commercial detection in videos. The commercial detection algorithm is based on the combination of visual and audio features. The success of this algorithm mainly relies on shot detection as well as logo and stock ticker detection, and it also involves video decoding, image and short time audio feature extraction, online learning and classification. The novelty of this project is that by using a bottom-to-front scheme, we are able to separate all commercial and non-commercial clips even if there is no distinct separation indicator between two adjacent blocks, and thus improve the detection effectiveness. Our approach enables removal of commercials in sports programs in a real-time way with an average accuracy of 95% *and an average recall of* 95%*. We will present the algorithm, implementation and evaluation in detail in this report.*

1. Introduction

TV programs especially sports programs usually are embedded with large group of commercial blocks. However the contents of TV programs are independent on commercial blocks inserted to them. So commercial blocks have no contribution and even side effects to program processing like analysis, understanding, indexing and retrieval of TV program. Therefore automatic detection and removal of commercial blocks have a great meaning to multimedia broadcasting system.

Although there are a lot of work dealing with commercial detection in videos, almost all of them deals with regular programs like news, series, movies, etc, as well as adopts a top-to-bottom scheme.

In this project, we specifically deal with sports programs. Compared to regular programs, sports are more difficult due to the following aspects. First, there are usually more commercials in sports programs than regular programs. Sometimes, the non-commercial block lasts even a shorter time than commercial blocks. Secondly, regular programs usually has loud speech, and with no or just soft music that lasts only for a short period. In commercials, there are usually speech together with strong music that is loud, with strong rhythm and continues for long times. But in sports programs, the speech and music could be pretty similar as the one in commercial. Third, in sports programs, there are always prediction, playback and summary of the show, which are quite similar as commercials. Forth, in sports programs, there are usually large text regions indicating the sport information like competitor names, clock, score, etc, which usually take place in commercial blocks. All these features increases the difficulty of commercial detection in sports-program videos.

The classification of video blocks are different as traditional classification. There is no training data, nor representative features. Commercials possibly have similar scene as non-commercial programs. Similar videos with similar features could be commercial in one video but non-commercial in another video. More specifically, the type of video block is independent to the video itself including encoding format and topic, block location, and block content, as well as the type of previous and followed blocks. Furthermore, in some videos, there is even no distinct separation between commercial and non-commercial blocks, which makes the commercial detection even hard for human beings.

In our project, to guarantee the commercial blocks and non-commercial blocks can be successfully separated, we adopt a bottom-to-top scheme. We define four layers: frame layer, shot layer, block layer and program layer. In frame layer, we combine the visual feature and audio feature, as well as extract the low-level features such as color information, edge information, logo/stock ticker, text regions. In shot layer, we detect the shot boundaries based on the frame features. In block layer, we detect the block boundaries, i.e., separation between commercial and non-commercial blocks, and then classify the block. In the program layer, we do some further analysis to the program, such as program recognition and indexing. Compared to the traditional

[*]This work is produced during her intership in Comcast Lab, DC, which durates from May to August, 2013. The slides of the project can be found in `http://prezi.com/u_zbybg9rayy/commercial-detection/`

top-to-bottom scheme, which usually searches black frames and frames where the aspect ratio changes to cut the video, our method only merges similar frames, shots and blocks. As long as the similarity between two objects exceeds a certain threshold, we will leave them uncombined and rely on the power of classifier to categorize their types individually. This will greatly increase the recall of commercial detection, especially for videos where commercials are inserted randomly.

2. Frame Layer Feature Extraction

To describe the algorithms we used in this project, we intuitively follow the bottom-to-top scheme, lower levels to high levels.

2.1. Video Decoding

The ffmpeg library[1] is used to decode the video. The video stream may contain image packets, and audio packets, as well as subtitle packets. In this project, we only concern the image and audio packets, and only decode these two kinds of packets. During the decoding phase, ffmpeg will output a sequence of packets, and it will automatically attach the type of packets in class `AVPacket->stream_index`. Hence we can select only the packets we want.

Image Packet Decoding Each image packet contains one and only one image frame. So to decode it, we only select all the video packets and read the image from their associated buffer. Usually within a video, the fps of image stream will not change and the time intervals between two adjacent frames will keep stable. To synchronize with audio, besides the image data we also note the clock and frame type indicator (intra-coded (I), predicted (P) and bi-directional (B)).

Audio Packet Decoding Audio is not synchronized with video, and the duration of each audio packet is also different as the one of an image packet. In order to combine the visual and audio information, we re-organize the audio frame so as to make it last for the same time as an image frame. To synchronize with image, similar as image processing, we note the audio clock and sample frequency besides the samples.

Remark There could be multiple audio sample format, mainly PCM 16 bit singled, PCM 16 bit singled planar, 32 bit float and 32 bit float planar. In non-planar systems, different channels are interleaved and we thus only need to read the samples in the first channel regardless the number of channels. However, in planar systems, the channels are not interleaved and we need to read the samples from all channels.

[1]http://www.ffmpeg.org/

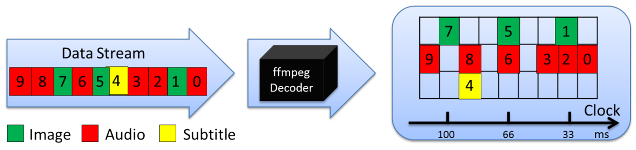


Figure 1: Synchronization of different types of video packets.

Once we have extract both the audio and image packet for a specific interval, we can re-organize the packets and form a so called `Media Data` structure which contains both audio and video packet data associated to the same clock, as shown in Figure 1. Then we can easily extract the combined video feature from this `Media Data` structure.

However, the raw image and audio data take large memory to store. For example, an image with resolution of 1280*780 takes $1280 \times 780 \times 3 = 2995200$ Bytes, and a one-second audio sequence takes $96000 \times 2 = 192000$ Bytes for PCM 16 singled format and 384000 Bytes for PCM 32 float format. So totally a one-second video takes approximately 90 MB memory, assuming the image frame frequency $fps = 30$. And usually the clock gap between an audio packet and an image packet obtained consequently could be more than 1 second. Therefore the strategy that stores the raw data and then do synchronization is too expensive considering the memory.

To address the memory issue, we switch the procedures of feature extraction and synchronization. Once we obtained a whatever packet, we extract and store its feature accordingly, and then do synchronization once possible. Therefore for either image or audio frame, its feature is a vector with dimensions no more than 20. Even take into account the intermediate frames that helps compute certain features like logo, the required storage memory still greatly reduced.

2.2. Image Feature Extraction

Image features can be categorized into four groups: basic information, color information, edge information and object information which includes logo, stock ticker and text. Since the last one is much complex than others, we present it in a single section 2.3.

Basic information The basic features are inherited from the decoded image packet, including frame clock, frame type (i.e., if the current frame is a key frame or not in the video encoding phase) and fps (number of frames per second). These features contribute little to the commercial detection, but they do help synchronize with audio and locate the video sequence.

Color information The color features are extracted based on the statistics of the luminance values of all the pixels in a frame. Multiple features related to the luminance can be obtained.

- Aspect ratio (Letterbox)
  Usually the resolution of a video clip does not change, but for different programs, the aspect ratios are variant. Black pixels are used to make the frame fix a specific resolution. So for a specific frame, we can derive the aspect ratio by removing the black contour and then divide the left frame width by its height.
- Brightness
  The average luminance of all pixels inside the letterbox.
- Number of dark pixels
  Number of pixels whose luminance value is less than 60.
- Uniformity
  The variance of luminance of all pixels inside the letterbox.
- Histogram change between adjacent frames (HC)
  This feature is very helpful in detection of scene change.
- Histogram change between every other frames (HC2)
- Black frame or not
  This feature is derived from all above features.

Edge information The main purpose of extracting the edge information is to detect shot boundaries. Although edge features are effective, the computation of edge is quite expensive, making online commercial detection almost impossible. So to speed up the algorithm, we use color features to find out possible shot boundaries and then use edge feature to verify the decision. Hence, edge feature is extracted for only a small number of frames, which are with a high probability to be shot boundaries.

- Edge change ratio (ECR) [7]
  The edge change ratio is defined as follows. Let σ_n be the number of pixels in frame n, X_n^{in} and X_{n-1}^{out} the number of entering and exiting edge pixels in frames n and $n-1$, respectively. Then

$$ECR_n = \max(\, ECR_{in},\, ECR_{out}\,), \tag{1}$$

  where

$$ECR_{in} = \frac{X_n^{in}}{\sigma_n}, \tag{2}$$
$$ECR_{out} = \frac{X_{n-1}^{out}}{\sigma_{n-1}}.$$

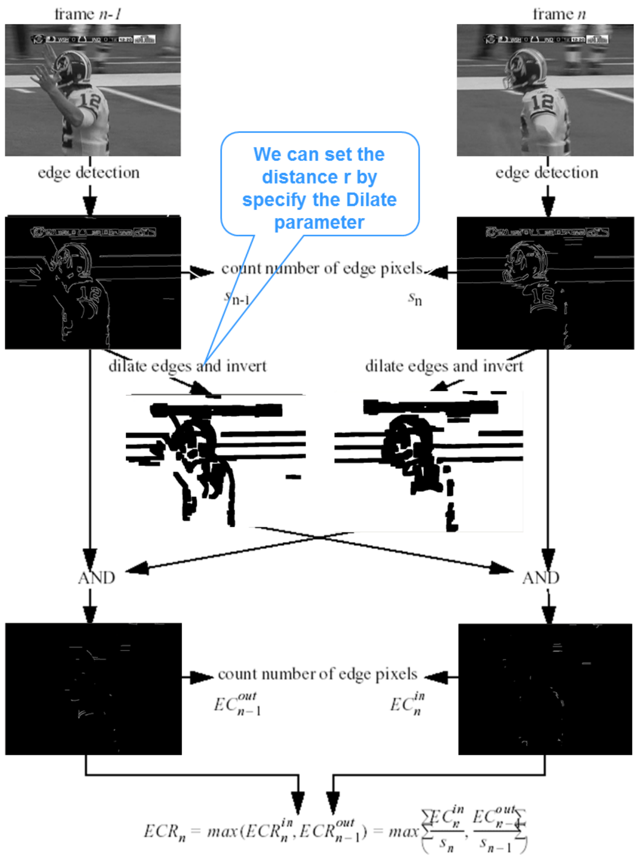

frame n-1
frame n
edge detection
edge detection
We can set the distance r by specify the Dilate parameter
count number of edge pixels
s_{n-1}
s_n
dilate edges and invert
dilate edges and invert
AND
AND
count number of edge pixels
EC_{n-1}^{out}
EC_n^{in}
$ECR_n = max(ECR_n^{in}, ECR_{n-1}^{out}) = max\left(\frac{EC_n^{in}}{s_n}, \frac{EC_{n-1}^{out}}{s_{n-1}}\right)$

Figure 2: Estimation of edge change ratio.

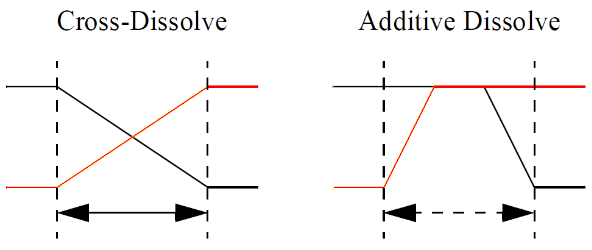
ECR can used to recognize both hard cuts and soft cuts in shot boundary detection. According to [7], hard cuts are recognized as isolated peaks, and fade-ins/fade-outs are recognized when the number of incoming/outgoing edges predominates, while during a dissolve, initially the outgoing edges of the first shot protrude before the incoming edges of the second shot start to dominant the second half of a dissolve. Figure 2 shows the estimation procedures of ECR.

- Edge stable ratio (ESR)
  Edge stable ratio is defined as the ratio between the number of preserved edge pixels and the number of total edge pixels in adjacent frames as follows

$$ESR = \frac{X_{n-1} \cap X_n}{X_{n-1} \cup X_n}, \tag{3}$$

  where X_{n-1} and X_n are the number of edge pixels in frames n and $n-1$, respectively.

  ESR is used to block shot boundary detection, when there is a uniform contour frame in the adjacent images while the image contents change a lot. This kind of frames are usually used to summarize the TV show.



■ intensity scaling function of the outgoing shot

🟥 intensity scaling function of the incoming shot

Figure 3: Typical intensity scaling function applied .

- Edge-based contrast(EBC) [2]
  Edge-based contrast is specifically designed to detect shot boundaries where dissolves happen, of the hardest cut types in shot boundary detection. Typically, there are two types of dissolves: cross-dissolve and additive dissolve, shown in Figure 3.

  The edge-based contract feature captures and amplifies the relation between stronger and weaker edges. Given the edge map $K(x, y, n)$ of frame n, a lower threshold θ_w for weak and a higher threshold θ_s for strong edges. Then the strength of strong and weak edge are defined as

$$w(K) = \sum_{x,y} W_K(x, y), \quad s(K) = \sum_{x,y} S_K(x, y),$$

  where

$$W_K(x, y) = \begin{cases} K(x, y) & if \theta_w \leq K(x, y) < \theta_s \\ 0 & else \end{cases},$$

$$S_K(x, y) = \begin{cases} K(x, y) & if \theta_s \leq K(x, y) \\ 0 & else \end{cases}.$$

  Therefore the EBC is defined as

$$EBC(K) = 1 + \frac{s(K) - w(K) - 1}{s(K) + w(K) + 1}, \quad EBC(K) \in [0, 2]. \tag{4}$$

If an image lacks strong edges, the EBC approximates to 0; while an image contains almost strong edges, the EBC is close to 2.

2.3. Logo and Stock Ticker Detection

Videos in TV program usually contain logos, indicating rating, broadcasting company, name of program, as well as the team logo in sport shows, as shown in Figure 4 and 5.

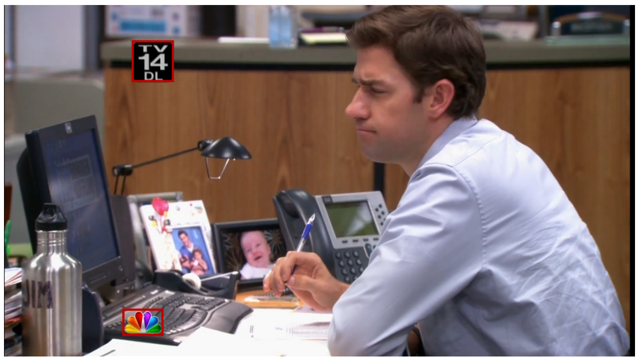


Figure 4: Rating logo (top) and broadcasting logo (bottom) marked with red rectangles.

In commercial detection tracks, there are usually two types of logos: known and unknown.

Known logos could be inputed by customers or extracted from the training/supervised videos. When known logos exist, the main purposes are usually to detect, track and recognize their scale/rotation variant counterparts. SIFT [3] and SURF [1] features are believed to achieve these tasks excellently [6].

However in this project, we do not have any supervised information, thus our task is to detect, track the unknown logos, and then to recognize them to help classify the videos. Although most shows contain logos while most commercials do not contain logos, there are still many exceptions, for example, Papa John's Pizza [2] contains its own logo, and many sports program in United States have logos from time to time but not continuously. Thus logo itself is hard to tell a video is commercial or not for sure, but usually commercial and non-commercial videos contain different logos. So we need to not only find out if an image contains any logo or not, but also identify different logos.

Logo detection takes place on key frames. First, we continuously compare the adjacent key frames and find out the long-lasting stable regions, and then select the regular and dense regions as potential logos. Secondly, for each potential logo, we need to validate it from four aspects:

- Size and aspect ratio
  Logo cannot be too large nor too small. And the width of a logo could be much larger than its height, but usually the height cannot be much larger than the width.
- Density and stability
  Logo should be very stable in a certain time. That means the difference of two adjacent key frames inside the logo area should be relatively small, while the difference outside the logo area should be relatively large. However, for some transparent logos, this dif-

[2]http://www.youtube.com/watch?v=Q1n3XitgsH0

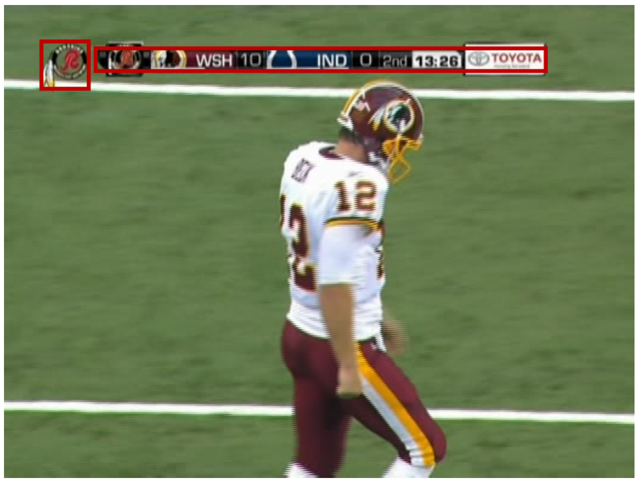


Figure 5: Team logo (left) and time/score ticker (right) marked with red rectangles.

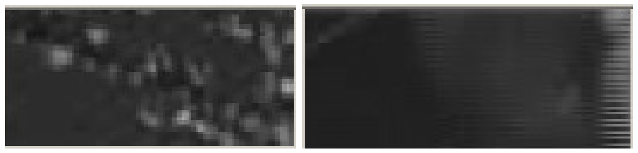


Figure 6: Invalid logos whose uniformity is too small.

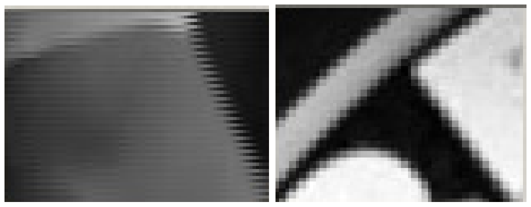


Figure 7: Invalid logos whose uniformity is too small.

ference could be still large. So in that cases, the edge information is necessary. We are going to compare the edges of adjacent key frames instead of color.

- Uniformity
  A logo usually contains abundant colors to be attractive. So an object with small uniformity usually is not a real logo, as shown in Figure 6.
- Edge
  A logo should contain sufficient edges, as well as a closed exterior contour. The lack of both of them leads to invalid logos, as shown in Figure 7.

Once a logo is validated, we can assign it an unique id and take advantage of it to classify the video blocks.

For a stable logo, its detection takes averagely eight key frames. Thus the necessary time depends on the encode format. For a .mpg format, it takes 5 to 8 seconds for commercial and around 20 seconds for TV show; for a .mp4 format, the commercial logo detection takes similar time but the TV logo takes around 2 minutes.

To make the logo detection more efficient and effective, we don't use detection techniques all the time. Once a logo is validated, we store it in our logo libraries and then try to track each logo at each frame or key frames or in a specific time interval, or whatever.

In tracking, we compare the logos with respect to color and edge. Solid logos can be easily tracked by either color or edge while transparent logo can only be tracked by edge. Since some logos may have holes in the middle, see the one in the left rectangle in Figure 5, in order to exclude the holes, we adopt a mask which is exactly the stable regions generated by comparing the adjacent key frames. When during the tracking stage, we just compare the color and edge insider the mask and estimate the similarity of the logo and tracking area, which is defined as follows.

$$Sim(l_0, l_i) = \frac{n(mask(l_0) \cap diff(l_0, l_i))}{n(mask(l_0))},$$

where l_0 is a validated logo, l_i is the sub-image in the tracking area (the area inside the red rectangle in Figure 5). l_0 and l_i can simultaneously be either color images are edge maps. $mask(l)$ represents the mask, i.e., stable regions, of l. It can also be color image or edge map, depending on the type of l_0 and l_i. $n(l)$ defines the number of non-zero pixels in image l. $diff(l_1, l_2)$ is a binary image with the same size as both l_1 and l_2. When l_1 and l_2 are color image, let the pixels be 1 where the difference of l_1 and l_2 are less than 10, and 0 otherwise. When l_1 and l_2 are edge map, let the pixels be 1 where they belongs to the edge map for both l_1 and l_2, and 0 otherwise.

In the tracking stage, when $Sim(l_0, l_i) > 0.6$ we consider logo l_0 appears, when $Sim(l_0, l_i) < 0.3$ we consider logo l_0 disappears, and when $0.3 \leq Sim(l_0, l_i) \leq 0.6$ we consider logo l_0 still exists if it also appears in the previous frame, and consider it not exists if it does not exist in the previous frame. Compared with the logo detection, logo tracking involves less computation and be much more efficient and immediate.

We only track the logo with the same location and scale. If two logos are pretty the same but with different scales or locations, we still consider them as two and assign them with two different ids.

Ticker detection is pretty similar as logo detection. The only difference is that in the validation procedure, the validation conditions are a bit different. For tickers, both the size, aspect ratio are pretty much larger than the ones of logo, and it also need to contain sufficient uniformity(not so uniform) and edges.

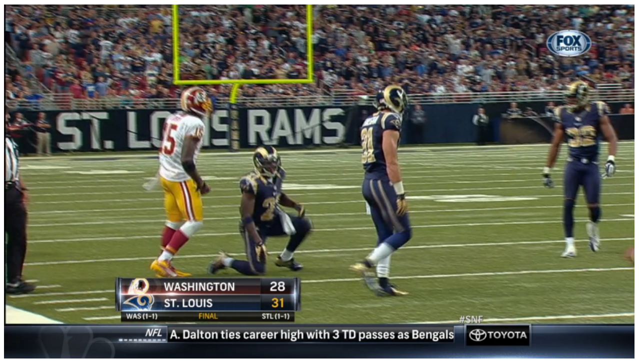


Figure 8: Example of extremely stable bar (at the bottom), which prevents ticker and logo detection.

Remark When logos and tickers appear simultaneously in the same frame and they are very close to each other as shown in Figure 5, it may be hard to distinguish both of them. Then in this situation, we can assume the two are not separated and consider is as a ticker. Since logo is usually more stable and lasts longer than ticker, we are able to identify the logo as soon as the ticker disappears or changes. Once the logo is detected, we can easily track it immediately when it appears even if it simultaneously occurs with a ticker.

Besides, in some videos, there are some frames surrounding the image which are extremely stable and will be miss-considered as a logo or time ticker, as shown in Figure 8. So we need to exclude these kind of bar regions before logo and ticker detection. More details about excluding this bar can be referred to `FrameInterface::check_if_StableContour()`.

2.4. Text Region Detection

The text region detection is modified based on `ShotBoundaryTextRegions.cpp`, and the detection stages are more or less the similar, including procedures as follows.

- Compute brightness image
  This image is a one-channel image obtained from a color image, but when color image is not available, a grayscle one also works. For a color image, each pixel of the brightness image is assigned with the maximum one among the three channels of its color counterpart.

- Compute horizontal contrast image
  The horizontal contrast image is computed based on the brightness image. For each pixel, we compare its value to the ones of its two left neighbors and two right neighbors. The pixel is assigned to 1 if the difference with any of its above neighbors is less than 64, and 0, otherwise.

- Fill horizontal holes
  Once the horizontal contrast image is obtained, the strokes of texts are visible, so we need to fill the strokes by filling the horizontal holes between two pixels in a horizontal lines if their distance is less than a predefined threshold. Usually this threshold can be set to 16 by default. However, if the texts are large in a video or image, we can set this threshold larger.

- Detect text regions
  First we need to adopt closing operators, i.e., erosion followed by dilation, to make the filled strokes more compact. Then the contour of possible regions areas are detected. We can obtain the text regions by checking its size, aspect ratio, and the difference between the contour area and the bounding rectangle.

- Select text regions
  In commercial detection, actually we do not detect the texts nor recognize the texts, since both commercial and non-commercials as well as program parade may contain texts. What we try to find out is the long sentence regions, which is usually only contained in commercial videos, to describe the products or give out the contact information, i.e., telephone number, web link, address, etc. So we filter and just preserve the text regions whose aspect ratio is larger than a threshold and estimates its duration time. This threshold can be predefined as 10 empirically.

2.5. Audio Feature Extraction

Four types of audio features are used in commercial detection: the volume, the high zero-crossing rate ratio, the low short-time energy ratio, and the spectrum flux. The first is calculated within a frame, while all the other three are estimated in a short time window, usually a 1 second hamming/hanning window.

- Volume
  The volume of a frame is defined as

$$Volume = \frac{1}{M}\sum_{m=1}^{N} |x(m)|,$$

  where M is the number of samples in the current frame, and $x(m)$ is the value of the m-th sample in that frame.

  Volume is useful to detect the block boundary, since when the program changes, there usually be a silence interval, although it may very shot.

- High zero-crossing rate ratio (HZCRR) [4]

This parameter estimated based on the zero-crossing rate (ZCR) [5], which is very useful to identify speech and music. Its definition can be written as

$$HZCRR = \frac{1}{2N}\sum_{n=1}^{N}[sgn(ZCR(n) - 1.5avZCR) + 1],$$

where n is the frame index, N is the total number of frames in the short time window, $sgn[]$ is a sign function and $avZCR$ is the average zero-crossing rate of frames in that window, and it is defined as

$$avZCR = \frac{1}{N}\sum_{n=1}^{N} ZCR(n),$$

and

$$ZCR(n) = \frac{1}{2(N-1)}\sum_{m=1}^{M-1} |sgn[x(m+1)] - sgn[x(m)]|,$$

where M is the number of samples in the n-th frame, and $x(m)$ is the value of the m-th sample in that frame.

Empirically, speech signal has a significantly higher HZCRR than music ones.

- Low short-time energy ratio (LSTER) [4]
  Low short-time energy ratio can be considered as a variation of short-time energy (STE) [5], which is also used to discriminate speech from music.

  LSTER is defined as the ratio of the number of frames whose STE are lass than 0.5 times of the short-time energy in a short time window as the following,

$$LSTER = \frac{1}{2N}\sum_{n=1}^{N}[sgn(0.5avSTE - STE(n)) + 1],$$

where n is the frame index, N is the total number of frames in the short time window and $avSTE$ is the average short-time energy of frames in that window, and it is defined is defined as

$$avSTE = \frac{1}{N}\sum_{n=1}^{N} STE(n),$$

and

$$STE(n) = \log(\int_0^{w_0} |F(w)|^2 dw),$$

where $F(w)$ denotes the Fast Fourier Transform (FFT) coefficients, $|F(w)|^2$ is the power at frequency w, and w_0 is the half audio sampling frequency. The FFT is performed to signal sequences in the n-th frame.

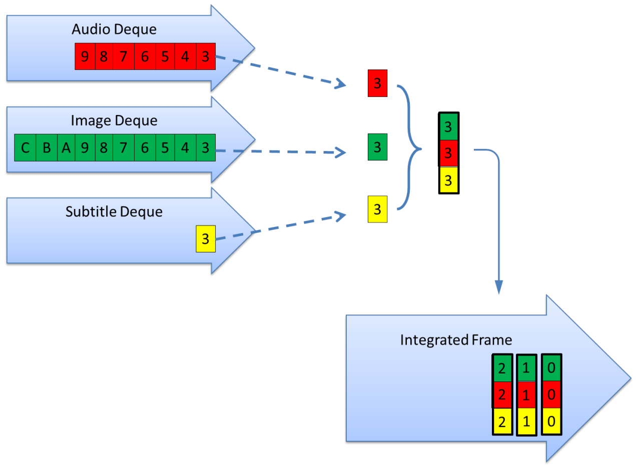


Figure 9: Synchronization and integration of different types of frame features.

- Spectrum flux (SF) [4]
  Spectrum flus os defined as the average variation value of spectrum between the adjacent two frames in a short time window.

$$SF = \frac{1}{(N-1)(K-1)}\sum_{n=1}^{N-1}\sum_{k=1}^{K-1}[\log(A(n,k)+\delta) - \log(A(n-1,k)+\delta)]^2,$$

where $A(n,k)$ is the Discrete Fourier Transform (DFT) of the n-th frame of input signal:

$$A(n,k) = \left|\sum_{m=-\infty}^{\infty} x(m)w(nL-m)e^{j\frac{2\pi}{L}km}\right|,$$

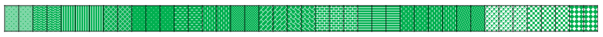
and $x(m)$ is the original audio data, $w(m)$ the window function, L is the window length, K is the order of DFT, N is the total number of frames and δ a very small value to avoid calculation overflow.

2.6. Frame Layer Feature Synchronization and Integration

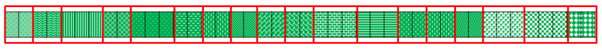
Once we obtain the image and audio features, sometimes maybe even subtitle features if available, we can integrate them and form a combined frame feature. The integration is purely based on the clock of each separate features. Only frames corresponding to the same time interval will be merged together. A simple synchronization and integration scheme is shown in Figure 9.

3. Commercial Detection Framework

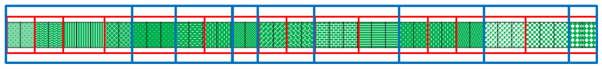
We use a bottom-to-top scheme to detect the commercial. Once the integrated frame feature is obtained, we



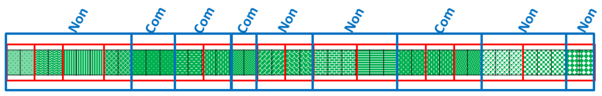
(a) A set of frame sequence



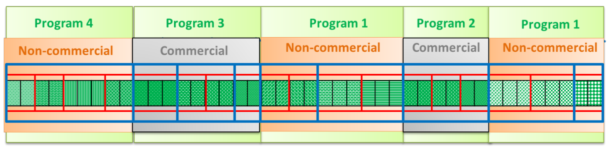
(b) Merge similar frames into a shot



(c) Merge similar shots into a block



(d) Compute block features and categorize the block into different types. ”Com” denotes commercial blocks and ”Non” denotes non-commercial blocks.



(e) Merge blocks with same decision types into a program

Figure 10: Panorama of commercial detection procedures.

detect the shots and come up to the shot feature. Then based on the shot feature, we detect the block boundaries and get the block feature, based on which we can categorize the block into commercial, non-commercial or program parade, or even specific test video frames without content. Then continuous commercial block sequence forms the commercial programs, the continuous TV show block sequence forms a TV program segment, and other types of block sequence form the correspondent program. The commercial detection framework is shown in Figure 10, which gives a panorama of the detecting stages.

3.1. Shot Boundary Detection

Shot detection is very important in commercial detection, because beside the block duration, shot cut rate is the most significant and stable factor to do the categorization. Cut rate denotes the number of shots per second, and shot means a sequence of frames with pretty the same scene and music. Although some other factors like key frame distance, average volume, music/speech, text regions, and black frames helps commercial detection, but they are not absolute that in different videos the values and thresholds for these parameters could variate too significantly to help make meaningful decision. For example, in .mpg format videos with fps=30, the average key frame distance for commercial is 0.8 key frames per second, and 0.65 key frames per second for non-commercial; and in .mp4 format video with the same fps, the average key frame distance for commercial is 0.4 key frames per second, and 0.2 key frames per second for non-commercial. When the fps and other parameters change, the key frame distance also changes greatly.

However, cut rate is quite stable for different videos with different encoding format. For instance, in both .mpg and .mp4 format videos even with different fps, the average cut rate for commercial is 0.18 shots per second, 0.4 shots per second for non-commercial, 0.8 shots per second for program parades.

Using the previous estimated frame features, mainly HC, HC2, ECR and EBC. Since the computation of edge map is expensive, we compute the HC and HC2 for every frame. The estimation of softness for a transit can be referred to `ShotInterface::ComputeShotTransitFeature.cpp` in the project. Only when HC is smaller than a threshold, we consider there exist a potential shot boundary, also referred as transit. This threshold is empirically set to 0.96. Then for the frames whose HC is less than 0.96, we compute their edge map and then estimate ECR and EBC, in order to confirm if there is a true shot transit or not. We use *softness* to denote the probability of existing a shot boundary, which is estimated based on the HC, HC2, ECR, EBC parameters as well as others like the variance of brightness and uniformity.

Then we set a predefined threshold, and consider there is a shot cut if the softness is larger than that threshold, and there is no shot cut if the softness is no larger than that threshold. There is a trade-off in the setting of this threshold. The higher the threshold is, the higher the shot boundary detection precision and the lower the recall. We can set the threshold according to different requirements and videos, or using cross-validation to find an optimal one. In this project, this threshold is set to 0.2 empirically.

3.2. Block Boundary Detection

The commercial detection relies much on the shot cut rate. However, the one in a single shot is not reliable because in non-commercial video blocks we can have short shots and in commercial video blocks we can also have long shots. So we merge similar shots or shots tightly concatenated together into blocks, and then estimate the average cut rate for that block.

In order to detect the block boundary, we define a parameter so called *hardness* to describe the probability that there exists a block boundary. Block boundary means the previous and followed video sequences are very different and they are clearly separated.

Some parameters affects the hardness. Black frames is a solid factor that leads to the hardness equal to 1.0, and aspect ratio change also leads to a high hardness. Some other factors are listed as: logo disappearing, different logos, sudden image content change including sudden change in histogram, simultaneous changes of ECR-in and ECR-out. However, not all shot transit satisfied above conditions are hard transit. Only the ones with a silence (volume less than 10), and a drop in ESR make hard transit. The hardness of a transit can be estimated from the parameters mentioned above and some other auxiliary ones like brightness change, uniform change, and number of dark pixels change. The estimation of hardness for a transit can also be referred to `ShotInterface::ComputeShotTransitFeature.cpp` in the project.

Similar as shot boundary detection, we then set a predefined threshold, and consider there is a block cut if the hardness is larger than or equal to that threshold, and there is no block cut if the hardness is smaller than that threshold.

There is also a trade-off in the setting of this threshold. Similar as the softness threshold, the higher the threshold is, the higher the block boundary detection precision and the lower the recall. And additionally, high hardness threshold leads to long blocks which is easier to classify, but may not separate all commercial blocks from other types of videos, while low hardness threshold leads to shorter and more block segments, which are less statistically stable and thus hard to classify, but different types of video blocks will be surely well separated. We can set the threshold according to different requirements and videos, or using cross-validation to find an optimal one. In this project, this threshold is set to 0.4 empirically, but for many videos which only insert commercials after black frames, we can set this threshold to be 1.

3.3. Block Type Classification

Since we do not have any training data, and the features for different videos might extremely variate, we only use some intuitive and basic parameters like block duration, cut rate, text regions, volume and music/speech to do the classification. Because the detection is real-time, to improve the classification performance, we learn the videos as we classify them, and then use the learned information to help classify the later video blocks.

The block type decision is made on four levels in descending order of confidence as follows. In the project, we categorization results is represented using a enumeration type `CommercialDetectionType`. The last bit of this value denotes the classification result and the other bits represent the confidence. If the last bit equals to -1, it means the algorithm have not made a decision yet; 0 indicates testing frames without content; 1 represents non-commercial sport shows; 2 leads to commercials and 3 results in program parade (forecasting). Then you can ignore the last bit and only concern the other bits to figure out the classification confidence to the very block. The lower the value is, the more confident the decision is.

Extremely level (void) In this level we only classify the videos with high confidence. Video blocks whose duration exceeds the predefined maximum single commercial size will naturally be categorized as non-commercial. And long duration shot with its image never change and without any audio will be categorized as testing frames without content. Decision made in this level will have only one bit, which indicates the block type. The confidence value is 0 (highest confidence). More details about the decision rules can be referred to function `BlockInterface::isNonCommercial_Absolute()`.

Independent level (_L) In this level, the decision is made out of the video block itself and no need to information from other video blocks. The decision is mainly made based on cut rate, logo, ticker, text region, duration, volume, SF, HZCRR, LSTER, and black frames duration. More details about the decision rules can be referred to function `BlockInterface::isCommercial_DecisionTree()`.

Decision made in this level contains not only the decision type, but also the confidence value, which is in the range of $[10, 99]$, and all the type contains "_L".

Dependent level (_M) In this level, no decision can be made based on the features of the video block itself, so we need to use the information learned when classify the other blocks. The decision is made based on the decisions of its neighborhood blocks, together with its own features like aspect ratio, cut rate, duration, and volume. More details about the decision rules can be referred to function `BlockInterface::isCommercial_BlockSegments()`.

Decision made in this level contains not only the decision type, but also the confidence value, which is in the range of $[100, 199]$, and all the type contains "_M".

Correcting level (_H) In this level, decisions have been made for all video blocks, however they may not be correct or reasonable for some blocks, for example, a commercial block lasting for 15 seconds between two non-commercial blocks. Then this block is more likely to be a non-commercial one as its neighbors. More details about the decision making procedure can be referred to function `ShotBoundaryCommercialDetector::CombineSameProgram()`.

Decision made in this level contains not only the decision type, but also the confidence value, which is in the range of $[200, 299]$, and all the type contains "_H".

4. Evaluation

the evaluation results can be referred to the attached documents.

5. Future Direction

1. Add Closed Caption.

2. Make full use of audio information. Currently, audio is just simply used. We can use audio and image to detect block boundaries separately, and then take their intersection as real block boundary. We can estimate the difference between the real audio and the predicted one (residual). If the residual is large, then there should be a block boundary. Use spectral characteristics to check if music or speech exists or not would also be helpful.

3. Also the music signature can help not only detect commercials but also recognize commercials.

4. Now I only use text region detection to help detect commercials. Actually, the text recognition results could greatly improve the commercial detection results. If text recognition is not performed, the filtering of non-text regions can also help a lot.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[2] R. W. Lienhart. Comparison of automatic shot boundary detection algorithms. In *Electronic Imaging'99*, pages 290–301. International Society for Optics and Photonics, 1998.

[3] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[4] L. Lu, H. Jiang, and H. Zhang. A robust audio classification and segmentation method. In *ACM Multimedia*, pages 203–211, 2001.

[5] L. Lu, S. Z. Li, and H.-J. Zhang. Content-based audio segmentation using support vector machines, 2001.

[6] K. Schoeffmann, M. Lux, and L. Bszrmenyi. A novel approach for fast and accurate commercial detection in h.264/avc bit streams based on logo identification. In *Advances in Multimedia Modeling*, Lecture Notes in Computer Sciences, pages 119–127, Berlin, Heidelberg, New York, Jan. 2009. Springer.

[7] R. Zabih, J. Miller, and K. Mai. A feature-based algorithm for detecting and classifying scene breaks. In *Proc. ACM Multimedia 95*, pages 189–200, 1995.