computer graphics & vision

Inffeldgasse 16/2, A-8010 Graz

www.icg.tu-graz.ac.at

TUG

Content Based Video Retrieval

Christoph Wiesmeier

Inst. for Computer Graphics and Vision
Graz University of Technology, Austria

Technical Report
ICG–TR–
Graz, July 28, 2016

contact: Christoph Wiesmeier c.wiesmeier@student.tugraz.at

Abstract

*As the number of videos available is permanently increasing the management of video collections becomes more time consuming. Video retrieval can be used to minimize the amount of manual work to manage these collections. Automated duplicate detection and assigning external available meta data, like movie names, to local video collections are two examples. Depending on the type of video different types of algorithms are used. In our work we focus on TV-series. We present several algorithms which use time-local descriptors that are computed at cuts. These local descriptors are common visual descriptors like the Gist***?** *descriptor and a time based descriptor. For evaluation we built two small databases containing 100 and 500 full length movies,respectively On this databases we perform both a duplicate search and video retrieval with different types of video manipulations. The evaluation will show show that the times between the cuts are sufficient to match videos. By using visual information it is possible to add more discriminative power which allows matching if that video contains only a few cuts.*

Keywords: *Video Retrieval, Seminar/Projekt*

1 Introduction

Video retrieval is the task of searching a database with a video as query. This can be used to solve different tasks. For example to prevent duplicates in a collection you can look if the video is already in the database before adding it. By using en existing database you can search for the name and other meta data and automatically name and structure videos on your file system. Video platforms like Youtube[1] also use video retrieval to reject upload if there are known license issues.

Comparing videos directly, e.g. , by their pixel values requires a huge amount of computational power. To allow a faster comparison we use descriptors. These descriptors can have different properties. The distinctness describes how good the descriptor is in distinguishing different inputs, Speed(how fast the descriptor can be compared) and size (how many bytes are required to store it) can also be important. Furthermore, we have preimage resistance which tells us if it is possible to find multiple inputs resulting in the same descriptor. Finally, we have robustness, which tells us how much an input can be changed while the descriptors still tell us that the movies are similar. In the context of videos this typically means robust to the following list.

- Image changes
  - Changes in resolution and pixel aspect ratio (anamorphic wide screen)
  - Black bands
  - Changes in intensity and contrast
  - Blurring and distortion
  - TV-Inpaintings (station logo/ advertisements ...)
  - Manipulation to overcome license checking, e.g., horizontal flip
- Temporal changes
  - Changes in frame rate or dynamic frame rate
  - Cropping in time
  - Manipulation to overcome license checking, e.g playing video backwards

[1] http://www.youtube.com/

Fast comparison and small storage can be achieved by cryptographic hash functions like sha1 and md5. They offer pre-image resistance and great distinctness. Cryptographic hash functions are used by many modern applications to sign documents or to identify files and entire file structures in version control software like git[2]. But the pre-image resistance required for signing documents has the disadvantage of absolutely no robustness which makes them unsuitable for video retrieval. Building a robust algorithm which is able to distinguish different videos is our main goal. Further we try to provide a small and fast matchable descriptor. Since we do not provide pre-image resistance it is possible to make huge changes in the input video without changing the descriptor. This should only happen if the video is modified on purpose in a way our algorithm does not detect. There is a similar application known as video hashing**?** which aims on detecting any meaning full change on the input while still preserving robustness.

Our method uses the idea that cuts are salient points in a video. By assigning a local descriptor to each salient point a global description is formed. The size of the global description is proportional to the number of cuts in the video. Various local descriptors like time spans between cuts or visual descriptors can be used. The similarity of the actual videos can now be estimated by the similarity of the global descriptor. An important requirement for this algorithm is that the videos contain enough cuts. This is the case for modern video productions as shown in Section **??**.

2 Related Work

As video retrieval isn't a new field of research there are already a lot of algorithms to use. Not all of them have quite the same goal. Most focus on finding the same video but others like Revaud et al.**?** try to find different videos of the same event like the eruption of a geyser. Furthermore depending on the application different source video material is used. The majority of algorithms uses short clips from websites like Youtube[3] and have the goal to handle databases of millions of movies. In contrast, our algorithm uses TV-movies which offer different properties like hundreds of cuts.

Similar research to video retrieval is done for perceptual audio hashing like Hamza et al.**?**. In this field periodicity based algorithms are common.

One algorithm we would like to mention in more detail is Robust Video Hash Extraction**?** which we have implemented as a reference. They sample the video down to a temporal resolution of 64 frames and a spatial resolution

[2] https://git.wiki.kernel.org

[3] http://www.youtube.com/

of $32 \cdot 32$ pixel. The result is a $32 \cdot 32 \cdot 64$ cuboid of intensity values. This cuboid is then converted to the frequency domain using the discrete cosine transform (DCT). The lowest 4 non zero frequency terms for each direction are used to form a $4 \cdot 4 \cdot 4 = 64$ value vector. This vector is thresholded by its median which results in a 64 bit vector with 32 ones and 32 zeros. The hamming distance of two bit-vectors is used to describe the difference of the two videos.

Since we use the local binary pattern (LBP) descriptor in an algorithm we also mention the work of Lifeng et al.**?** in more detail. They use a uniform sampling to extract frames from the video. A descriptor based on local binary patterns is then extracted from each frame. Those patterns are computed by splitting the image in $3 \cdot 3$ areas, and compare their average intensities. A 8 bit descriptor is computed by those comparisons. Using a w-shingling**?** algorithm, additional temporal information is encoded. Thereby, a 16 bit vector is formed. A histogram of these 16 bit vectors is built to form the final descriptor. They finally present a method for database search with time complexity O(log(n)) which makes their algorithm suitable for web scale video databases.

3 Framework

The goal of our framework is to allow simple testing of our algorithms at a reasonable computational cost. The framework can be split into input alteration, descriptor extraction, and database.

The input alteration can modify videos before the are processed by the descriptor extractor. This process is used during search operations to simulate different modifications which are applied to a video. To alter the videos we use a chain of filters. The filters are applied in realtime on each frame.This on the fly alteration saves us from previously creating and storing the 1920 videos we currently process during testing.

The descriptor extraction makes use of the same filter chain. It uses special filters which add annotations to each frame. Currently we use a difference calculator and a cut detector before the actual data extraction. This design allows simple component replacement, e.g., using pixel or histogram based differences for cut detection. The actual data extraction is implemented as two filters. One for our cut based algorithms and another for the reference algorithm. A typical filter chain can be seen in Figure 1.

The results of the descriptor extraction are now stored in a relational database. Using a relational database allows us storing additional data, like images of each cut, without loading them during search. Furthermore it

provides locking which is required during concurrent execution.

Building a database in our framework means disabling input alteration and storing all the descriptor in a database file. During search we use different types of input alteration, extract the descriptors and compare the results against our database.

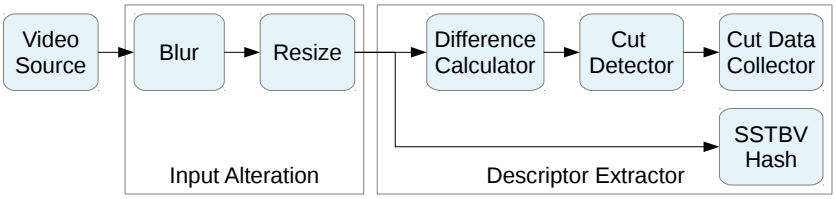


Figure 1: Sample filter chain as used for video search. The filters (Blur/Resize) are used to simulate a modified source video and are part of the test system. The difference calculator and the Cut detector adds additional information. Finally the Cutdata-Collector and SSTBV-hash produce the video descriptors.

For performance reasons we implemented most of our framework in C++ using OpenCV**?** and the Qt framework[4]. The test system is a collection of python scripts which are manly used to analyze the results. As database engine we use SQLite**?**.

4 Shot boundary detection

Shot boundary detection in any situation is a quite complicated task. This is a result of the large number of different types of boundaries. Common are hard cuts, fades, and dissolves. Hard cuts have no transition between to shots. Faded means the video becomes gradually darker until it is black, or staring from a black image until the full illumination is reached. Dissolve means that one shot is gradually blend into the other. More details and an algorithm to detect all these shot boundaries can be found in Shot Boundary Detection at TRECVID 2007**?**.

4.1 Algorithm

Fortunately, hard cuts are the most common of all these shot boundaries and the fades and dissolves are mostly used to separate different scenes. This allows us to only use hard cuts for our algorithms, particularly because

[4] http://www.qt-project.org/ (2014)

we only need to find the same shot boundaries, in all versions of the movie but not all of them. This allows us to use significantly simpler and faster algorithms.

Our algorithm consists of two parts. First, calculating the distance of two frames, and second, detecting the actual cut. To calculate the distance we simply use the mean intensity difference on a pixel basis. For this purpose the image is converted to gray scale and the average of the absolute difference over all pixels is calculated. One example of such a signal can by seen in Figure 2. Based on these differences we now have to decide where a cut happened. Our assumption is that if the inter-frame distance is higher than a threshold, a cut happened. It was not possible to define a global threshold which is higher than the difference caused by viewpoint changes but lower than the difference caused by all cuts. We thus did use a relative threshold based on the differences before and after. This detector works in most situations but leads to false positives on still images, if the compression occasionally adds some noise. To overcome these situations we added an absolute threshold which is larger than this noise level. The result is the detector in Equation (1).

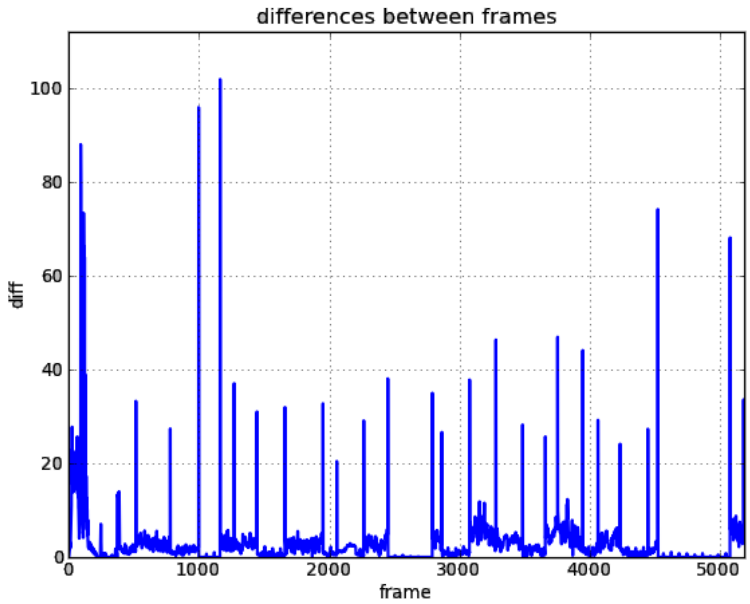


Figure 2: Average per pixel difference of a TED-Talk (first 5000 frames). The spikes correspond to cuts.

$$
\begin{aligned}
cut[i] = & \text{diffs}[i] > \text{minRelDiff} \cdot \text{diffs}[i-1] \quad \& \\
& \text{diffs[i]} > \text{minRelDiff} \cdot \text{diffs}[i+1] \quad \& \\
& \text{diffs[i]} > \text{minAbsdiff}
\end{aligned} \tag{1}
$$

We also tried a histogram based approach for cut detection, but there was no significant improvement in quality to justify the increase in computing time. The evaluation of our cut detector will be sown in Section **??**.

5 Matching algorithms

We describe a video as a sequence of cuts. The detection of these cuts has been explained in Section 4. We calculate a descriptor for each of these cuts and store the sequence of descriptors. The descriptors we use will be discussed in Sections 5.1 to 5.4.

To match the descriptor sequences we use either the longest common sub-sequence algorithm**?** or a slightly modified version we call minimum distance sub-sequence. The longest common subsequence algorithm tries find the longest sequence of matching descriptors which is common to both descriptor sequences. When using the minimum distance sub-sequence we define a distance function between two cut descriptors and a fixed distance if a descriptor has to be inserted or removed from the sequence. The lowest possible distance is our video distance.

When using the longest common sub-sequence we are only interested in the length of this sequence. By dividing the length of the longest common sequence by the length of the longer input sequence we get a score from zero to one. During search we now compare the query video to every video stored in the database. The video of the database with the highest score is reported as the result. In our experimental evaluation, this design is used by the Time_Longest_Sequence and BP_Longest_Sequence algorithm. The GIST_minDist_Sequence, BP_minDist_Sequence and the DCT3D_minDist_Sequence algorithms use the minimum distance sub-sequence algorithm.

Important to note is that these algorithms calculate optimal solutions for the given problems and are time consuming. The subsequence algorithms have quadratic runtime based on l, which is the number of cuts of a movie. Further we compare a query video against all n movies of our database. The result is a run time of $O(l^2 \cdot n)$. For larger databases we suggest using heuristic algorithms which may not guarantee the optimal solution but are a lot faster.

5.1 Binary Patterns

For this algorithm we use two single byte Binary Pattern (BP) descriptors, one before and one after the cut. This descriptor is calculated by converting the image to gray scale, splitting it in 3 times 3 rectangles, and comparing their average intensity. This progress in illustrated in Figure 3. The idea of this descriptor is taken from Lifeng et al.**?**. The descriptor uses two different comparison groups. First there are 4 comparison with the center. Second there are 4 comparison of the corner regions. The idea is that the center regions have a lower chance to be altered by TV-inpaintings and therefor could be weighted different. We currently not utilize the different qualities of the bits.

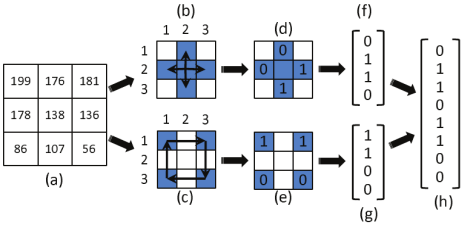


Figure 3: The one byte BP descriptor as described in Lifeng et al.**?**

This BP descriptor is used with two different matching algorithms, BP_Longest_Sequence, where only exact matches are accepted and BP_minDist_sequence, where the previous discussed Minimum Distance Subsequence algorithm is used. In this case the distance is defined as the number of different bits.

5.2 DCT2D

This feature is inspired by our reference algorithm**?**. The descriptor uses the lowest frequency terms of the 2 dimensional Discrete Cosine Transform (DCT) except the terms which contain a zero frequency in either direction. The values of the DCT are thresholded by their median. This results in a descriptor which contains an equal number of ones and zeros. We use the lowest 8 frequencies in both directions, which leads to a 64Bit descriptor before and after each cut.

5.3 GIST

This method is based on the Gist descriptor from Olivia and Torralba**?**. It is selected because it is one of very few descriptors actually designed to describe a hole image. As parameters we use one scale with 4 blocks and 8 orientations per block. Furthermore the images are resized to a fixed aspect ratio as suggested in the implementations documentation. In our case this is done by resizing to $320 \cdot 180$ pixels without preserving aspect ratio. The descriptor is used in its gray-scale version. For integration in our framework we use a Gist implementation in C from LEAR[5].

Since the descriptor uses a high dimensional vector of float numbers we use the sum of squared differences (SSD) to calculate their distance.

5.4 Time

This is the simplest algorithm. It uses time information as a descriptor for cuts. The basic idea is using the time between two cuts as a descriptor. But since some OpenCV versions do not provide correct timestamps for the frames we decided to use frame numbers. To overcome the problem of different frame rates we normalize these frame counts. This leads to the descriptor

$$d[n] = \frac{\text{frame}[n] - \text{frame}[n-1]}{\text{frame}[n+1] - \text{frame}[n]} \tag{2}$$

where $d[n]$ is the descriptor for the n'th cut and frame$[n]$ is the frame number of the n'th cut. If two descriptors differ less than 10% we treat them as equal during the longest common subsequence search. This condition is formalized in

$$(d_a \cdot 1.1 > d_b) \& (d_b \cdot 1.1 > d_a) \Rightarrow d_a == d_b \tag{3}$$

where d_a and d_b are two cut descriptors calculated according to Equation (2).

To estimate the discriminative power of this descriptor we calculate the average entropy of a cut descriptor $d[n]$. Equation (4) calculates the entropy H of one cut descriptor based on it's probability to occur. This is in our case the number of descriptors in the equality range Equation (3) divided by the number of cuts in the database. The result is about $4.3bit$ per cut and is slightly dependent on the data set. An evaluation of the entropy per movie can be seen in description of the dataset Section **??** Figure **??**.

$$H = -\sum^{cuts} p \cdot log_2 p = \sum^{cuts} p \cdot log_2 \frac{1}{p} = \frac{1}{NrCuts} \cdot \sum^{cuts} log_2 \frac{NrCuts}{MatchingCuts} \tag{4}$$

[5] http://lear.inrialpes.fr/software

6 Reference Algorithm

As a reference we use an algorithm proposed in robust video hash extraction **?**. In contrast to our algorithm it is not cut based. It has been chosen because it uses a straight forward approach and was implementable with reasonable effort.

The algorithm represents the video in form of the low frequency terms of a three dimensional DCT. Therefore the video is filtered and sub-sampled in time to 64 frames, then filtered and sub-sampled in space to $32 \cdot 32$ pixel. The resulting video is then transformed by a DCT and the 4 frequency terms in each direction, which represent the lowest non zero frequencies, are used to form a 64 dimensional vector. This vector is thresholded by its median which results in an equal number of ones an zeros. The result is a 64Bit descriptor for the movie where the number of different bits represents the distance between two movies.

Our implementation differs slightly from the original algorithm. The changes are made to allow a reasonable integration into our test system. The original algorithm uses first a temporal sub-sampling and afterwards the spatial sub-sampling. We changed this order to reduce the memory consumption by only storing the low resolution images. Second the temporal sub-sampling is done by a successive reduction of the frame buffer every time the buffer reaches 1024 frames. This is a result of not knowing the number of frames at the beginning and the goal to reduce the memory requirement on long videos. Both changes should have no significant impact on the retrieval performance.

7 Experiments

Now we will discuss the experimental evaluation of our algorithms. First, we will introduce the datasets we built for evaluation and discuss some of there properties we noticed. Afterwards, we will evaluate the cut detector. Finally we will test the results in database search and finally in duplicate search.

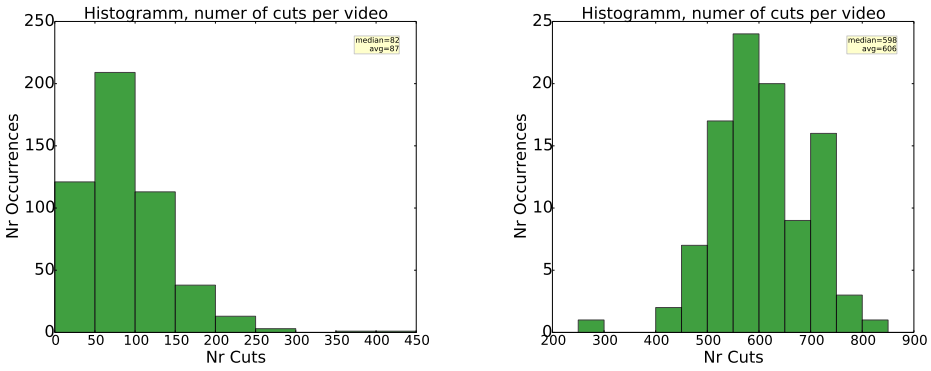
7.1 Dataset

For the following tests we use two data sets. The data set TED(500) contains a collection of TED-talks[6] which are presentation recordings. We use a random selection of 500 videos of the 1303 talks available on Nov/21/2012. The videos have a typical length of 5 to 20 minutes and are available in 3 different

[6] http://www.ted.com/

qualities: 480p, standard and light (The light version uses a reduced frame rate). If not otherwise mentioned we used the light version. Since some of these talks contain very few and in some cases no cuts at all, we generated a second data set Stargate(100) from 100 randomly selected Stargate SG1 Episodes. The use of two datasets allows us to analyze the quality on different types of video material. From each data set we built a database containing all descriptors of all videos. A comparison of the number of cuts and the time between cuts can be seen in Figure **??** and Figure **??**, respectively. Finally, we calculated the estimated entropy per movie which could be used by the Time_Longest_Sequence algorithm. This is a result of Equation (**??**) which calculates the entropy H based on the portability p that the time difference of to frames fits in the 10% margin. The result can be seen in Figure **??**.

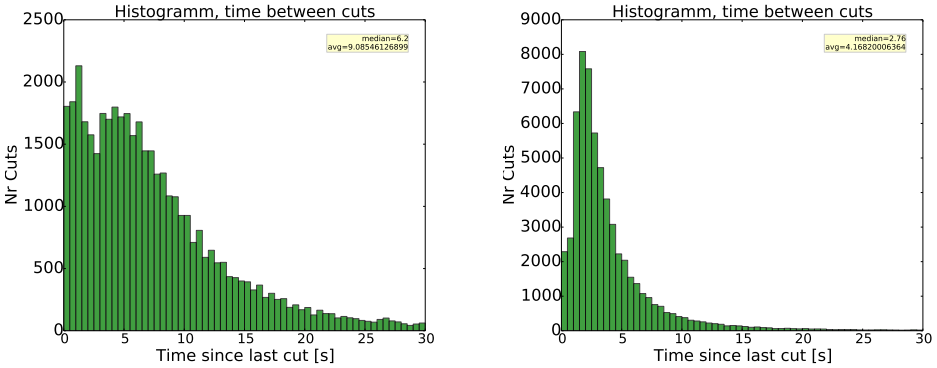
$$H = \sum^{cuts} p \cdot log_2 \frac{1}{p} \tag{5}$$



(a) TED(500)

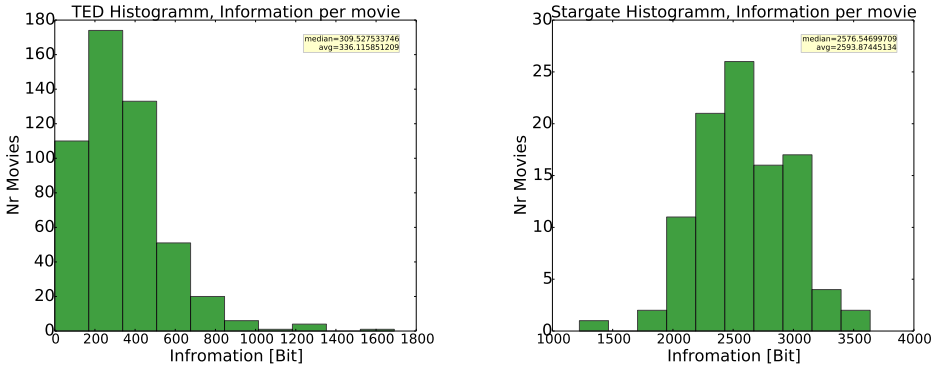
(b) Stargate(100)

Figure 4: Histogram which shows the number of cuts per video. Data extracted using our basic cut detector.



(a) TED(500) (b) Stargate(100)

Figure 5: Histogram which shows the distances between cuts over all videos of the data set. Data extracted using our basic cut detector.



(a) TED(500) (b) Stargate(100)

Figure 6: Histogram which shows the entropy per movie in bits.

7.2 Cut detector evaluation

To create test data for verification of the cut detection algorithm we implemented a tool for marking these frames in the movie. The tool opens the directory of images and allows seeking forward and backwards in the video and has some features for jumping between the marked cut frames. It displays two consecutive frames. If there is a cut between the left and the right image, this can be stored by a simple key press. Afterwards this information can be exported. A screenshot of the tool can be seen in Figure **??**.

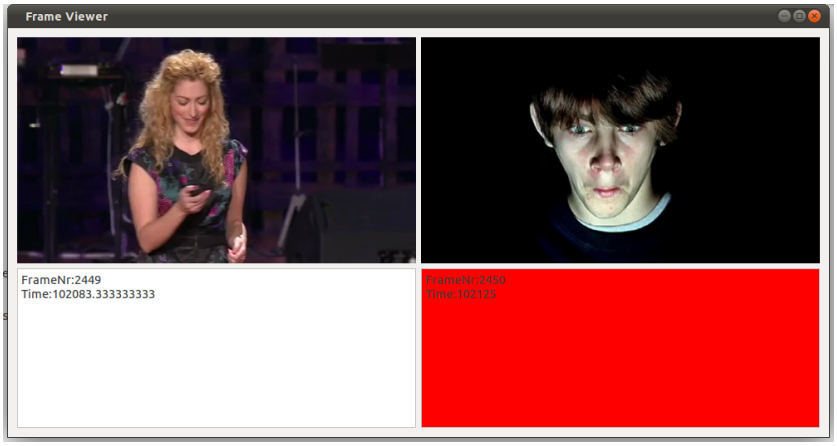


Figure 7: Screenshot of the tool which was built to annotate the cuts in a movie.

For evaluation, a selection of 10 TED-Talk's[7] was manually annotated. The thresholds minRelDiff and minAbsdiff from Equation (1) are optimized using previously annotated videos. During this optimization we use the score function

$$(truepositives - falsepositives)/nrAnotatedCuts. \tag{6}$$

This function assumes that a false positive detection is as bad as a missed cut. The result is a score where 1 is the perfect result. The lower limit depends on the average number of frames between cuts in the dataset, but basically every negative sore represents a quite bad result. The results of changing the thresholds can be seen in Figure **??**. The threshold minAbsdiff = 8.10 and minRelDiff = 1.81 delivered the best results. Since the too low thresholds have greater impact on the quality than the too high ones we decided to use minAbsdiff = 10 and minRelDiff = 2. Using this thresholds we detect 660

[7] http://www.ted.com/ (2012)

(97.2%) of 679 annotated cuts and 64 false positives of remaining 2147667 frames which are theoretical cut positions.

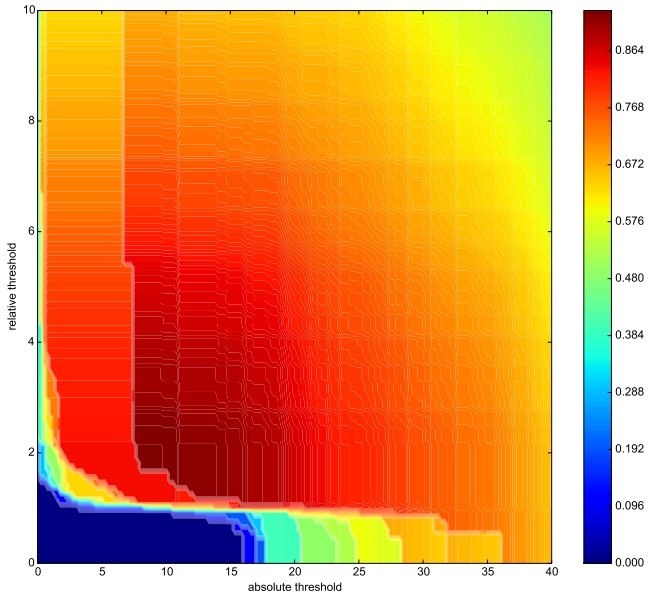


Figure 8: Variation of the parameters of the cut detector absolute-/relative-threshold on the horizontal/vertical axis. The color encodes the average of Equation (**??**) over 10 annotated videos. The maximum is $minAbsdiff = 8.10$ and $minRelDiff = 1.81$ which results in score of 9.29.

The described algorithm works good for the original videos. But the assumption that a cut can only have one frame leads to problems when using a movie which is captured from a screen. The problem is the missing synchronization between the monitor and the shutter of the camera. One exemplary result is illustrated in Figure **??**. This is basically a very short dissolve which is not covered by our algorithm.

7.3 Video Manipulations

To test the descriptors we have to artificially manipulate the videos to get query videos. These manipulations are described below.

Image Scaling: in this case we change the resolution of the video to the given resolution in pixel. If only one value is given this means we resize to a quadratic image without preserving aspect ratio.

(a) First frame (b) Cut frame (c) Next Frame

Figure 9: These three frames illustrates the result when capturing the video from the screen and synchronization mismatches. (Ted talk Jane McGonigal)

Random frame drop: In this scenario we use a random number generator to decide for each frame whether to drop it or not. A drop probability of 0.1 means that 10% of the frames are removed on average. This simulates the impact of temporal re-sampling if the frame rate changes.

Flip: Flip mirrors the image along the x-,y-, or both-Axes. This is sometimes used to overcome license checking.

Black water mark: Here a black rectangle in the upper left corner is added. The parameter in this case is the relative side length of the rectangle compared to the image with/height. This operation is used to simulate TV-in-paintings as well as black stripes which are used to change the aspect ratio.

Crop Front: cuts away the first part of the movie. A value of 0.1 means the first 10% of the movie are cutted away. This stands for every manipulation where trailers, scenes or the production information (end of the movie) are cut away.

Gaussian filtering: Blurs the movie with the given standard deviation in pixel. Simulates all sorts of operation which reduce image quality, like compression artifacts of blur when capturing a movie with a camcorder.

7.4 Retrieval Results

In this test we calculate the number of correct retrievals for different modifications of the video. For this test 20 Videos are selected from each data set, TED(500) and Stargate(100), and the robustness is analyzed.

First, we test the impact of a change in resolution as can be seen in Figure **??**. It can be seen that for reasonable resolutions larger than $50 \cdot 50px$ there is no significant impact on the results.

Next, we test the results when dropping frames from the movie. The results in Figure **??** show a large impact on the Time_Longest_Sequence algo-

rithm. This is not a real surprise since removing frames changes the temporal information, which is the foundation of the algorithm. More interesting is the reduced retrieval rate of the other algorithms. Because algorithms using min distance matching are loosing quality we expected more impact on cut detection than on visual features. Finally it should be considered that the huge frame drops of up to 90% are not common. Common values are more likely in the range of 4% e.g. as a result of converting 25 to 24 fps.

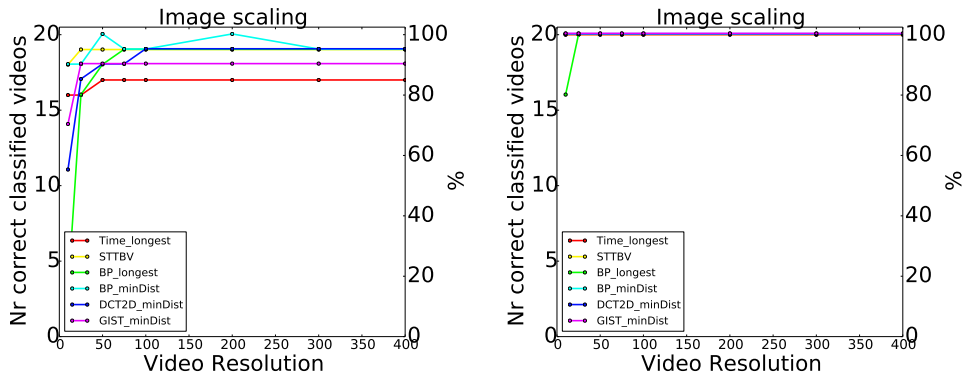
Now we take a look at what happens if we flip the image, see Figure **??**. Obviously this has no impact on the Time_Longest_Sequence algorithm which is invariant to this type of transformation. The really interesting algorithm is GIST_minDist_Sequence which looses quality as can be seen on the TED data set but still works good enough to receive perfect score on the Stargate data set. This means that the left and the right half of the image are usually visually similar for the gist descriptor. This does not mean similar pixel values but maybe similar texture. For instance, if a scene is taken in the forest, the background is quite similar on both sides.

In Figure **??**, we see the results of our black watermark test. It's manipulation has a large impact on the BP_longest_Sequence algorithms. A watermark size of $1/3 = 0.\dot{3}$ already forces two bits to fixed values. If we assume a equal distribution of all descriptor values this means 75% of the descriptors are changed. The BP_longest_Sequence matching allows only exact matches and therefore only 25% of the cuts remain for comparison. In this case the BP_minDist_Sequence has a big advantage. Since it can also match similar descriptors it is less sensitive to this sort of manipulation. The Gist_minDist_Sequence algorithm is also sensitive to this type of modification. In this case we expect large changes on only a part of the descriptor values. The comparison using sum of squared differences can cause a high difference as a result of a few large changes. In this particular case a matching using the sum of absolute differences would be preferable.

One of the most important tests is the truncation of the beginning of the video. It is very common to remove trailers or credits from a movie. In this case the reference algorithm STTBV has the biggest trouble, as can be seen in Figure **??**. This lies in the nature of the algorithm which can not match parts of a movie. The other algorithms are basically influenced by the reduced number of cuts which can be used for matching. Nevertheless, every cut that is lost by the truncation is treated as a mismatching descriptor. The negative impact of this effect could be decreased by reducing the mismatch penalty on some of the algorithms.

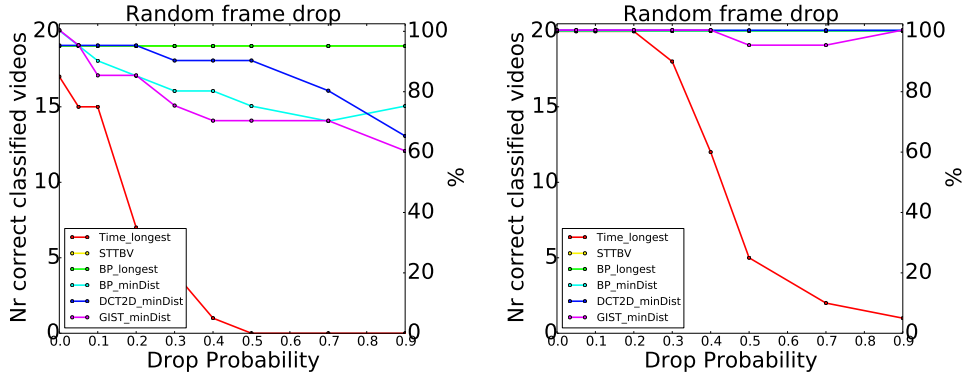
Finally, in Figure **??** we test the impact of Gaussian filtering. Here we basically find two effects. First, the filtering has an impact on the cut detector, which leads to the quality loss of the Time_longest_Sequence, BP_longest_-

Sequence and BP_minDist_Sequence algorithms on the TED dataset. The second effect is filtering away higher frequencies used by the GIST, and DCT2D descriptors.



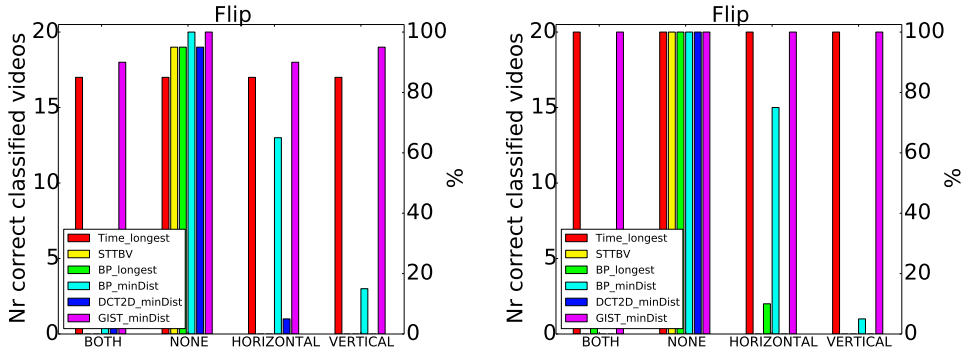
(a) TED(500) (b) Stargate(100)

Figure 10: Number of correct retrievals over the resolution of the video.



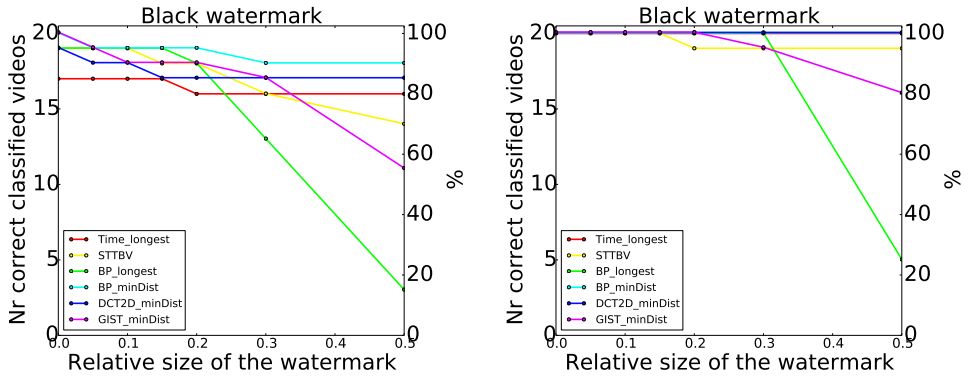
(a) TED(500) (b) Stargate(100)

Figure 11: Number of correct retrievals over the drop probability.



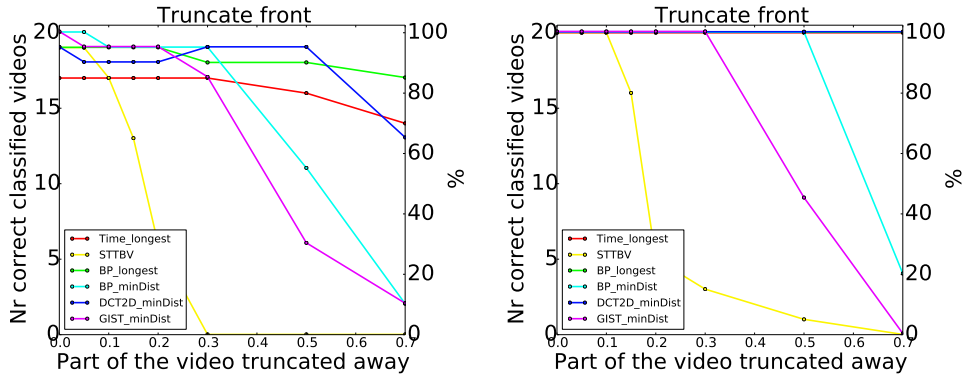
(a) TED(500) (b) Stargate(100)

Figure 12: Number of correct retrievals when the video is flipped horizontal,vertical, or both



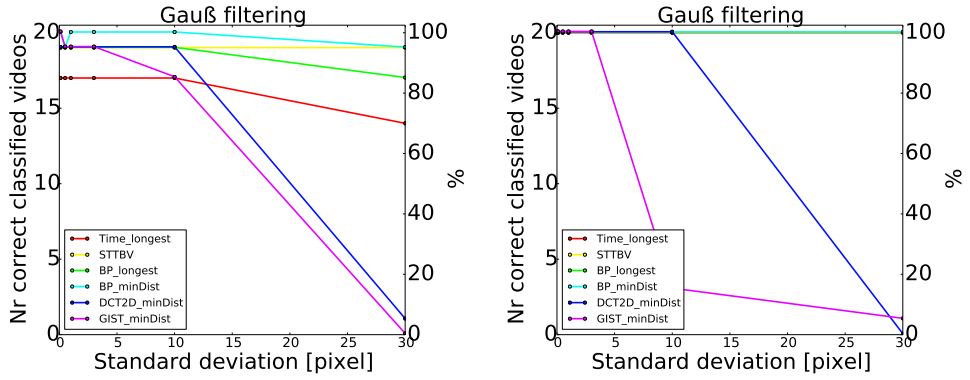
(a) TED(500) (b) Stargate(100)

Figure 13: Number of correct retrievals over the size of the watermark where 0.5 means a side length of watermark is 0.5 times the image width/height



(a) TED(500) (b) Stargate(100)

Figure 14: Number of correct retrievals over the length of the video cropped away at the front $(0 - 70\%)$.

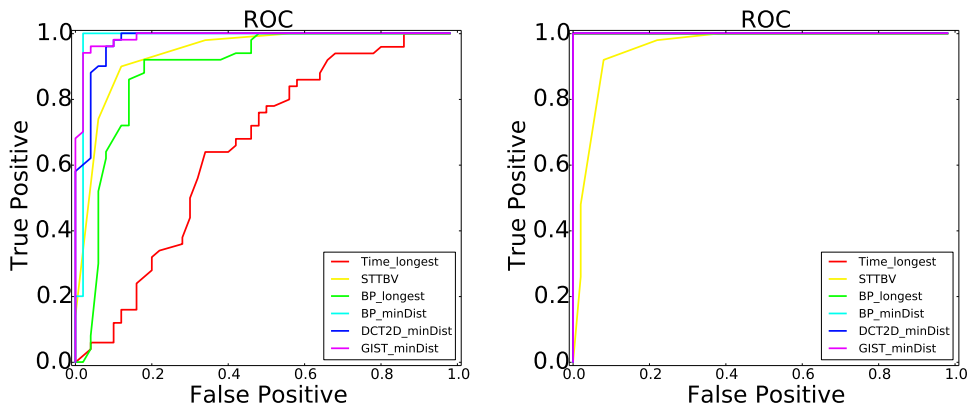


(a) TED(500) (b) Stargate(100)

Figure 15: Number of correct retrievals over standard deviation of the Gauss Kernel.

7.5 Duplicate Search Results

In this experiment we simulate the task of finding duplicate in a video database. For both our TED(500) and Stargate(100) dataset, we form a query set based on 50 videos which are included in our dataset, and 50 which are not. The videos which are not in the dataset are taken from the corresponding sources (TED, Stargate episodes). These 100 query videos are now searched in the same databases as in the previous task. Because we also would like to find slightly modified duplicates we again filter the query videos. We use 10% crop front, resize to $200 \cdot 150px$ and a black watermark of 0.1 side length. We now use the quality level returned by the search tool to decide whether the movie is in the database or not. By changing the threshold of the classification we get the Receiver Operating Characteristic (ROC). Our results can be seen in Figure **??**. The true positives represent the movies which are correctly classified to be in the database. The false positives are movies which are classified to be in the database but actually are not. Therefore, if a curve reaches the upper left corner, we have perfect results. The worst result is the diagonal from lower left to the upper right corner which could be reached by simple guessing. Now the curves show that the Time_Longest_Sequence algorithm performs poorly on the TED(500) data set and is only a little bit better than guessing. Whereas all cut based algorithms were able to perfectly classify the Stargate(100) data set. Our reference algorithm (STTBV) has similar results on both data sets because it is not affected by the different number of cuts.



(a) TED(500) (b) Stargate(100)

Figure 16: Receiver operating characteristic for both data sets

8 Conclusion

Our algorithms require videos with enough cuts. They do not perform very well on the TED-Data set. But on modern TV-productions, for which they are designed, they work fine. Furthermore our algorithms are very robust against cropping in time because of the longest common sub-sequence matching. In some situation the robustness has to be limited because detecting the half of a movie as a duplicate is not always desired. Matching using the GIST descriptor is a quite interesting option if the detection of flipped videos is desired. In our opinion the best version is LBP_longest_Sequence, since it produces good results, while having the smallest (16Bit) descriptor. Time_Longest_Sequence is also interesting because it is very robust to image manipulations. But the low information of a about 4.3 bit per cut results in the requirement of many cuts and seams to be not ideal for faster matching algorithms which are required for larger databases.

An important thing to note is that for all tested methods it is easy to create a movie which results in the same hash. Especially, the Time_Longest_-Sequence algorithm can be fooled by replacing the whole video content only preserving the cut's. To summarize, cut based video retrieval works good but not on any video material.

9 Future work

To use the algorithms in real world applications the matching speed is the most important point of improvement. Therefore, our simple brute-force matcher has to be replaced. We suggest matching the cut descriptors directly against a fast data structure and only test the actual sequence on the best candidates.

In our algorithms we use cuts as salient points in movies. Using other prominent points like regions with minimal motion could lead to similar results but without requiring the video to contain cuts. This could allow a larger range of applications.

Finally it would be interesting if a cut detector could work only on metadata of compressed video files. This metadata could be the number of bytes the video codec requires to represent a frame. If this is possible the Time_longest_Sequence algorithm could be used without decoding the video. This would result in a huge speedup of the descriptor extraction.