Automatic video grouping and aggregation

by Anders Spælling (spaelling@gmail.com)
Lauge Starup Jepsen (laugejepsen@gmail.com)

May 10, 2012

Contents

1 Introduction **5**
- 1.1 Overview . . . 6
  - 1.1.1 Video-clip segmentation . . . 6
  - 1.1.2 Grouping and labeling . . . 6
  - 1.1.3 Video summary aggregation . . . 6
- 1.2 Limitations . . . 6
  - 1.2.1 Type of events . . . 6

2 Data set **7**
- 2.1 Building the data set . . . 7
  - 2.1.1 Using videos from Youtube . . . 7
  - 2.1.2 Removal of unwanted material . . . 8
  - 2.1.3 Partly use of some material . . . 8
  - 2.1.4 Pre-processing of the videos . . . 9
  - 2.1.5 Noise in the data set . . . 9
  - 2.1.6 Limitations as a result of our dataset . . . 9

3 Video-clip segmentation **10**
- 3.1 Literature study . . . 10
- 3.2 Method . . . 10
- 3.3 Data set for the video-clip segmentation . . . 13
- 3.4 Test . . . 14
  - 3.4.1 Overall procedure . . . 15
  - 3.4.2 The Algorithms . . . 15
  - 3.4.3 Tweaking . . . 17
  - 3.4.4 Testing . . . 18
  - 3.4.5 Robustness . . . 19
  - 3.4.6 Multi-parameter algorithms . . . 19
  - 3.4.7 Test cases . . . 23
  - 3.4.8 Results . . . 24
  - 3.4.9 Analysis of results . . . 27
  - 3.4.10 Analysis risks . . . 27
- 3.5 Section summary . . . 30

4 Grouping and labeling **30**
- 4.1 Detecting interesting regions in video-clips . . . 30

5 Conclusion **30**

6 Discussion NOT DONE, ONLY NOTES FOR LATER **30**

List of Tables

1 Algorithm configurations 25
2 Algorithm performance 26
3 Sequence data 28

1 Introduction

News is big business, old news is not. We live in a world where we want the news as they happen, which mostly is not feasible. It takes a while for professional news crews to show up at a news-worthy event, but individuals present at the scene can sometimes become a decent substitute. The Internet contains literally millions of video clips from all around the world. This number is increasing exponentially, not just because more people have access to the Internet, but also because more and more people are equipped with video cameras, often in the form of smart-phones. It is therefore no surprise that often, when an event occurs, the first footage available consists of private recordings done by those present at the scene. With the increased access to high-resolution cameras and mobile Internet connections, we not only expect an increase in video uploads, but also in live-video streaming.

Currently, the vast majority of existing videos are published at video-streaming services such as YouTube, which receives almost 8 years of videos footage every day[1]. The current possibilities for mining this enormous dataset are limited. It is possible to search for most videos using tags and keywords, but very little work has been put into grouping videos together around the events they cover. Given the expected increase in video uploads we argue that it will become increasingly important to begin looking at ways to cluster videos together and merge them into aggregated video streams covering entire events. By considering each video a scene located at a point in space and time, we theorize that it should be possible to produce single videos covering specific events, by merging neighboring scenes.

Extensive research already exists in the field of image processing and labeling and the Internet provides access to an abundant source of video footage. However, this footage is commonly of a rather low quality (at least compared to professional recordings). For example it can be too shaky or too dark.

With the processing power of distributed computing and the advancement in the field of image processing and machine learning in mind, we propose a system, which given a collection of video footage recorded at an event is capable of composing a watchable and semantically valid summary of that event shortly after the data becomes available. A system like this could, if it was fully developed, remove some of the human interaction in news curation and, as a result, decrease the delay between when 'the news' happen and

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

when they become available.

1.1 Overview

1.1.1 Video-clip segmentation

Video-clip segmentation refers to the process of identifying usable regions within the raw video footage. Since footage may be very long or of poor quality it is important to be able to divide it into smaller sub-sections from which we can select the very best parts of the original video-clips. Really, video-clip segmentation should also cover identifying the most interesting areas in a video, that is the ones where the actions best describing the event occur. However, determining these areas is non-trivial and requires more complex video image analysis than simple frame quality assessment. Therefore we do this as a part of the grouping and labelling in the subsequent phase. In order to not limit ourselves later on we therefore only perform a 'soft' segmentation initially. While we do identify the usable regions in each video-clip and determine early suggestions for segments, we do not actually cut the video-clips. Rather, we store the results of the quality-based segmentation as 'elastic limits', which can be stretched more or less later on as more information becomes available.

1.1.2 Grouping and labeling

1.1.3 Video summary aggregation

1.2 Limitations

1.2.1 Type of events

The perfect system would be able to handle all kinds of different events. However, in order to limit the scope our work we have restricted ourselves to focusing on one kind of scenario: urban protests and demonstrations. Although this subject still yields diverse footage of everything from public speeches to police arrests it is only natural that some of the results we archive and the conclusions we make will not necessarily generalise to other kinds of scenarios.

2 Data set

Our vision of a full blown system is one that constantly receives raw video footage of an event directly from the mobile devices recording it. This footage would then be processed by the system and used in a near-live coverage of that specific event. Since such a system currently does not exist, our goal is to build a data set which as closely as possible mimics this scenario. We do this by identifying a small collection of recent events, and then collecting usable video-clips of it from Youtube and other publicly available sources.

2.1 Building the data set

Since the vast majority of the videos available online do not suit our needs and since no efficient tool exist for efficiently finding the ones that does, the construction of our data set is a somewhat manual process. Each phase of this project poses specific challenges with regards to the data set needed for testing. Specific choices will be described in detail in their respective sections. However, similar for all, are the problems and decisions detailed below.

2.1.1 Using videos from Youtube

Our initial idea was to build the data set from Youtube videos recorded at events by selecting them based on location and time. However, it quickly became obvious that this was not feasible. The YouTube data API does not support searches based on specific time spans and even if it had, the timestamps accessible through it are based on the time of upload, not the time of recording. Since only very few videos are uploaded right after they are recorded, we found that the timestamps from videos from a specific event would be spread out over several days, sometimes even weeks. Location based search is however supported, but again we found the information accessible through the API unsatisfactory. Although videos can be tagged with location information, this data does not necessarily correspond to the GPS location of the device that recorded it, but is often generated based on an non specific place name, like a street, city or even country. And again, the information is created at the time of upload not the time of recording and thus based on the user's position at that time.
Given these observation we instead came up with another way to build our data set. When users upload a video to YouTube they can choose to attach a

set of tags/keywords and categories to it. We found that these keywords and categories, when chosen correctly, can be a very efficient way to search for related video footage. For example, events like protests and demonstration can often be found by simply searching for a keyword describing the cause behind the event along with one describing the location (often a city). This way we are able to make a crude connection between the video footage and a specific time and location. Problems arise when the footage from a time and location can not be accurately identified through these attributes. For example, a video promoting a demonstration before it begins and explaining the reasons for it will often be tagged with the same keywords as the footage later recorded at the event. It may be possible to filter out such false-positives by further analysing its meta-data, but there does not seem to be a simple (or general) way to do this.

2.1.2 Removal of unwanted material

Since the purpose of our data set is to mimic raw, unedited footage as much as possible we have to perform a significant amount of manual filtering during our Youtube searches.

Completely unrelated videos
All videos that are obviously unrelated and only a result of ambiguous search queries are removed. Examples of this would be the police chase videos that emerged during our search for protest videos from the COP 15 Climate summit in Copenhagen in 2009.

Related, non-event videos
These videos are related to the event we want, but they are not from the event. Again, this is the result of a ambiguous search query or maybe even an unfit tag being applied to the video. Examples include videos that promote an event or, in the COP 15 example, videos regarding the climate meetings, which had nothing to do with the protests around it.

2.1.3 Partly use of some material

In some cases part of a video-clip may be suitable for a specific test although the clip in its entirety does not live up to our requirements. An example of this would be a video of an event that was obviously produced from several

other video-clips. Although using such a clip as if it was raw footage would severely hurt the credibility of our results, some of its sub-segments (or maybe even the entire video) can sometimes be used in specific tests, for example as a "gold standard" of what good footage or a good video summary should look like.

2.1.4 Pre-processing of the videos

The video-clips available online are recorded on a lot of different devices and stored in many different formats. In order to streamline the later analysis processes all videos are converted to the Apple MPEG-4 video format 'm4v'. The conversion also resizes them to a dimension of 640x360 pixels at 1600 kbit data per second at 24 frames per second. These videos are of a size, which makes them manageable computationally when we later have to analyse them, while still preserving enough information about their content.

2.1.5 Noise in the data set

2.1.6 Limitations as a result of our dataset

A lot of the limitations in our research should be obvious at this point. We chose to only focus on a very small subset of all the possible scenarios a fully automatic system should be able to handle. Furthermore, we screen the data manually in order to make sure it is usable. However, much of the necessity for this is based on the fact that we do not have access to the raw footage as well as all the meta-data originally attached to a video-clip. Our main goal was to create a data set, which mostly contains original raw footage or, in the cases where it makes sense, video-clips, which we can pretend is raw footage. The task of determining whether a video-clip belongs to this category is, in many ways, much easier if you know exactly when and where it was recorded. Furthermore raw footage also often contain information about the source device it was recorded on, which again would tell us that the video data is unaltered. Had we had our own video hosting service we would have access to such information.

3 Video-clip segmentation

3.1 Literature study

We have found several articles describing methods for segmenting and detecting (un)suitable areas in a video clip.
Girgensohn et al. [1] describes a simple method to measure how shaky a picture is at a given point by computing a displacement-vector. Another measure of image quality proposed by Girgensohn et al. is testing if a certain percentage of pixels is above some threshold; an indication of image brightness.
The *spring loading effect*[1] is a way to balance clip lengths. It is a measure of how much a clip can be expanded or contracted as a function of the area under the curve describing the unsuitability of the region the clip would expand into. The purpose of this is to automatically fit segmented clips to match a desired total clip length.
A more comprehensive approach described by ??? [3] is a multi-level classifier that labels each frame as belonging to one of the groups: "blurred", "shaky", "inconsistent" and "stable". Their solution is based on Support Vector Machines (SVMs) and yield satisfactory results.

3.2 Method

Girgensohn et al. [1] describes a method to compute a vector that indicates the displacement in two subsequent frames, where the direction of the vector corresponds to the direction of the displacement, and the magnitude how many pixels was displaced. The advantage of this method is the simplicity in implementation, and low computational costs. The authors suggests a shift of each subsequent frame in four directions of 32 pixels to start with, halving the number of pixels in each step. For each shift the Root-Mean-Square (RMS) of the difference in the two frames is calculated, and the shift with the lowest RMS is the basis for the following shift. Shifting a matrix is a simple matter of cutting away parts of the matrix to align the two, and computing the RMS is archived in a few inexpensive matrix-computations. Since the frame size of a video determines the amount of movement, which can be detected more or fewer shift-iterations may be better. Girgensohn et al. [1] do not mention the frame size of their videos. We simply stick with the same number of iterations as them, which should be more than enough.

The algorithm below illustrates how the displacement vector for two subsequent frames is computed.

```
f  <- read frame
f' <- read frame
for i in [32, 16, 8, 4, 2, 1]:
     e' <- infinity
     for d in [left, right, up, down]:
          s <- shift f' i pixels in direction d
          e <- compute RMS(f-s)
          if e < e':
               x,y <- i * (d[x],d[y])
               e'   <- e
     x',y' <- x,y
     f' <- shift f' x pixels right
     f' <- shift f' y pixels up
yield x',y'
```

With a frame dimension of 640×480 and a frame rate of 24 frames per second (fps) this method can theoretically trace camera movements of up to $24 \times (32 + 16 + 8 + 4 + 2 + 1) = 1512$ pixels per second, in any direction, ie. even extremely shaky camera motion should be accurately detected (accuracy is then highly dependant on the blurriness of the frames). Thus, we are able to trace camera movement of more than two full frame widths per second in the horizontal direction and more than three frame heights in the vertical direction. Images that are symmetrical around the vertical or horizontal center, ie. two almost identical local minima, can potentially fool the method into the wrong direction due to the greedy nature of the algorithm, but that is a rare statistical anomaly.
Girgensohn et al. [1] also describes a simple method of determining the level of brightness in the image by computing the fraction of pixels above a certain brightness threshold. While this is a sufficient measure for detecting (too) dark images, it does not take ex. extremely bright images into account, or closeups of a textureless surface. To counter this downside we instead computed a measure of contrast in a frame as the standard deviation of the pixel-values in the image-matrix, where a small standard deviation would indicate little contrast/diversity in pixel intensity, and a large standard deviation would indicate a high contrast/much diversity in pixel intensity. Hence, a very dark image or very bright image would have a low contrast as would

images with little or no texture. It will however not detect an image where half of it has a very low contrast, ex. an image of clear sky in the top three-quarters, nor would we necessarily want to discard frames like these.

There are several approaches to computing the suitability for a given frame: A binary approach in which a frame is simply deemed suitable or unsuitable for watching, and an approach where a frame is rated on a scale.
If W_1 is the magnitude of the displacement vector, W_2 is a measure of the image contrast, s is either 1 (suitable) or 0 (unsuitable), and v is a measure of suitability, then a suitability-value could be computed as: $v = \alpha W_1 + \beta W_2$, where α, β are weight-factors, and s could then be determined by $s \equiv v \geq \tau$, where τ is some lower bound. If we wish that $s \equiv W_1 \geq T_1 \vee W_2 \geq T_2$, the values of α and β can be determined as follows:

$$W_1 \geq T_1 \equiv \alpha W_1 + \beta W_2 \geq \tau,$$

the point being that the contribution from W_1 alone can determine the outcome of s, hence we set $W_2 = 0$ and W_1 assumes the lowest possible value, $W_1 = T_1$, and we can determine the value of α as:

$$\alpha T_1 \geq \tau \Leftrightarrow \alpha \geq \frac{\tau}{T_1}$$

The value of β can be determined as $\beta \geq \frac{\tau}{T_2}$.
With this method we ensure that either factor will be able to deem a frame unsuitable, and if both weights are close to their respective treshold the cumulative contribution will still deem the frame unsuitable.
Another approach is $v = (a^x + b^x)^{\frac{1}{x}}$ for a given value of $x > 1$, and then testing v against a given treshold, τ. It is somewhat similar to the previously mentioned approach where we multiply by some factor, and then add the values, the difference being that large values will weigh much more than small ones.
Another way to compute v is $v = \max(\frac{W_1}{T_1}, \frac{W_2}{T_2})$ and $s = v > 1$ where T_1, T_2 are the respective lower bounds.

3.3 Data set for the video-clip segmentation

The main purpose of the video-clip segmentation is to be able to identify regions in the video-clips, that are suitable quality-wise. We do this by looking at the level of contrast in the frames and by estimating the amount of movement and shakiness of the camera. For simplicity we convert all our footage to gray-scale, which brings decreases the complexity and computational costs significantly. Most of the videos used for this data set was found on Youtube and collected as described in section 2 on page 7.
We can roughly divide the videos from YouTube in two: raw and pre-edited video. What we generally want is raw video, but since this is by far the lesser portion of the two we decided to utilise the pre-edited video as a sort of benchmark on the pretense that a large majority of the footage is both of high quality and also contextually interesting.

Raw Youtube videos
The data set contains 25 raw videos. The length of the clips range from 10 seconds to 10 minutes, totaling somewhere around one hour of footage. All videos appear to have been recorded using hand-held devices so the quality of the footage varies a lot. In order to be able to train and test on this part of the data set, we generate a 'gold standard' manually for each video. This is done by watching the video and recording every time the footage switches state from 'bad' to 'good' or vice versa. .

Pre-edited Youtube videos
The data set furthermore contains a fair amount of pre-edited videos. This is because far the most videos available on Youtube have already been edited in some way. Many of these are videos consisting of several different video-clips, which themselves was once raw footage. In order to use these clips we split the edited videos up around their scene boundaries. This is done manually in much the same way as the when generating 'gold standard' for the raw Youtube videos: by simply watching the videos and recording the time of scene boundaries. Since a lot of these edited videos contain segments of unsuitable images, like photos or introduction text, only the parts that actually depict the event is kept. Because the resulting set of video-clips have in fact already been screened and deemed watchable by a human (namely the producer of the edited video), we assume that all of this footage is of a decent quality.

In the edited videos we are actually interested in each individual video-segment. In order to obtain these we manually cut each video into chunks with a simple script that would perform the cut on demand as we watched the clip.
The segmented video-clips were then reviewed, and some discarded based on the following criteria:

- very short segments - usually a 1 second segment would be discarded
- contextually invalid - ex. a montage of still-images or ending credits
- lack of visual quality - in some extreme cases an entire segment can be extremely dark/bright or shaky. This is obviously not acceptable since we assume all these segments to be of a decent quality in our later tests.

Keep in mind that some of these segments are discarded as they are out of context of the entirety of the video-clip, where they would (presumably) make sense to include, but not as standalone segments.
We also accept some "noise" in each end of the segment to complement ex. overly long crossfades. In the end we end up with a large selection of short segments of video-footage that we expect to be of good quality.

Our own recordings
Finally, the data set also contains five videos, which we have recorded ourselves. These videos depict various specific scenarios involving low contrast and darkness, shakiness, fast camera movement and completely still footage. The idea is that these videos serve as more controlled samples for what to tweak and test for.

3.4 Test

The purpose of our tests and analyses are to estimate the overall ability of several different algorithm, to correctly estimate the qualities of the frames in a video and classify them as either good and bad. Further more we want to estimate what parameter(s) yield the best performance for each algorithm.

3.4.1 Overall procedure

Since our dataset is limited in size we perform 5-fold cross validation on it in order to increase the amount of videos in the training set. The dataset is split into five buckets by sorting the video-clips by length and inserting them, one at a time, in a circular manner. The result is five sub-sets each containing more or less the same amount of frames between all the videos in each set. We then tweak each algorithm's parameter(s) on each of the training sets created by combining four out of the five sub-set and subsequently test the configuration on the remaining test set (the remaining fifth sub-set) and average the results into final scores. We also attempt to estimate how *robust* each algorithm is against changes to its parameter(s). That is, we want to quantify how good the algorithm is overall. This analysis is based on the entire dataset and happens after the 5-fold cross validation.

3.4.2 The Algorithms

All our algorithms work solely with the two variables *shift vector magnitude* and *contrast*. The values of these variables are quantified as how good (or bad) they are compared to how bad they can be.

The magnitude of a shift is limited to 63 pixels, which would happen if the image is shifted in the same direction in each shift-iteration. The *shift vector magnitude* ratio, *S(s)* is therefore simply defined as the *shift vector magnitude* normalised into the [0,1] range:

$$S(s) = s/63$$

, where s is the *shift vector magnitude*.

Since we work with 8 bit gray-scale images we know that all pixels in all our frames are in the color range [0,255]. If the *contrast*, c, is defined as the standard deviation between the lightest and the darkest pixel in the frame, we know that the maximum value of c is 127.5 (if these two pixels have the values 0 and 255, respectively). The *contrast* ratio, *C(c)* is therefore defined as:

$$C(c) = 1 - (c/127.5)$$

The reason we subtract the amount of contrast from one is because we want the value to be comparable to that of the *shift vector magnitude*, where

smaller values are preferred over bigger ones

We found it advantageous to square the values of these variables in order to increase the algorithm's sensitivity toward those in the bad end of the spectrum. Obviously, the squared values are still normalised into the [0,1] range. Secondly we deploy *triangular smoothing* on the values using a smoothing degree of 12 which roughly translates to averaging over a number of frames equal to one second, in order to remove some of the worst noise in the data.

Contrast only (CO)
This algorithm only takes the level of *contrast* in the frames into consideration. If the *contrast* is above the limit defined in the configuration of the algorithm, the frame is marked as bad.

Shift magnitude only (SMO)
This algorithm only considers how much a frame was shifted in order to attempt to match it to the frame before it. That is, it only looks at how much the images are moving or shaking. Just like with the CO algorithm we simply measure if the *shift vector magnitude* is beyond the configured limit, and mark the frame good or bad accordingly.

Independent contrast and shift magnitude (ICSM)
This algorithm takes both parameters into consideration and looks at if any of the two exceeds a given threshold. This mean that we ignore the cumulative effect of the two as a determinant for whether the video segment is unsuitable. The frame state classifier, S(c,s), is defined as:

$$S(c, s) = MAX(c \cdot (1 - c_{lim}), s \cdot (1 - s_{lim}))$$

, where c and s are the *contrast*- and *shift vector magnitude*-ratios respectively and c_{lim} and s_{lim} are the limits defined by our configuration. The function effectively normalises the effect of the limits so that the two values can be compared by the MAX statement. If $S(c, s) > 1$, one of the two parameters is beyond its limit and the frame is marked as bad.

Cummulative contrast and shift magnitude (CCSM)
Simplified contrast and shift magnitude (CCSM^3)) The last algorithm is a very simplified version of *CCSM*. It attempts to classify the frames by

simply taking the mean of *contrast* and *shift vector magnitude* and checking to see if it above a given threshold. The frame state classifier, S(c,s), is defined as:

$$S(c,s) = \sqrt[3]{\frac{c^3 + s^3}{2}}$$

The parameter values are cubed (and the result normalised to the [0,1] range) in order further increase their sensitivity toward bad values. Informal experimentation shows that the power of sensitivity does not effect the outcome result in any significant way, so it has been left out as a parameter.

3.4.3 Tweaking

In turns out that defining what final score to tweak for is not as simple as it sounds. Ordinary approaches usually tend to focus on the maximisation of positive precision and recall. A very effective method for determining the overall performance of an algorithm, with this as an priority, is to use ROC diagrams, as described by [REFERENCE]. However, due the disproportion between the number of good and bad frames in our dataset this approach is not very suitable for us. Instead we decided that our priority would be to maximise the negative recall as well as the positive precision, since this would mean that a good score signified an algorithm/parameter combination that was not only good at avoiding unsuitable frames, but also could be relied on for delivering mostly true-positive good frames. Of course we don't want to throw away too many good frames due to a low positive recall and a low negative precision, so any negative performance of these two factors should be considered as a cost. What we have come up with is a homemade measuring tool inspired by ROC diagrams. The cost, C, and benefit, B, are defined as:

$$C = 2 \cdot \frac{positive_{recall} \cdot negative_{precision}}{positive_{recall} + negative_{precision}}$$

and

$$B = 2 \cdot \frac{negative_{recall} \cdot positive_{precision}}{negative_{recall} + positive_{precision}}$$

We create a cost/benefit diagram, by defining the benefit as the harmonic mean of the positive precision and the negative recall and the cost as one minus the harmonic mean of the positive recall and negative precision. These values can be used to produce a diagram, which in many ways resembles a classic ROC diagram (see fig. [REFERENCE]). The resulting cost and

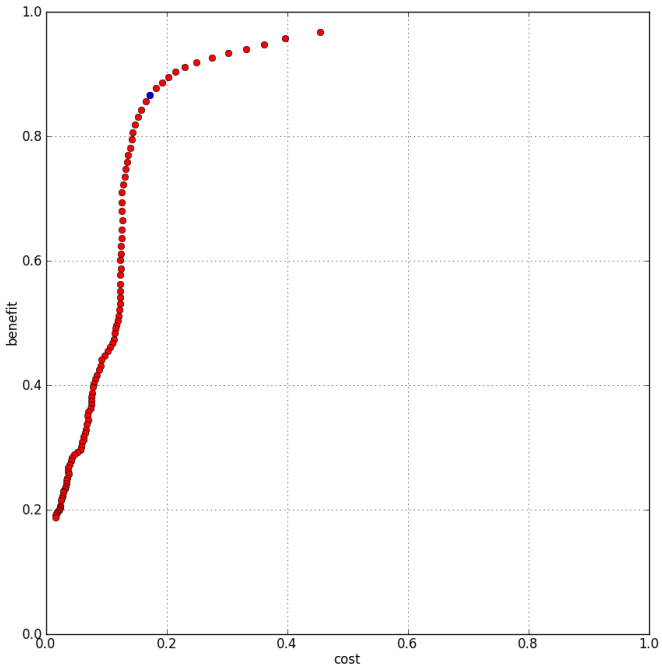

Figure 1: An example of a 2D cost/benefit diagram, the blue configuration is the one with the lowest distance score

benefits based on different parameter choices for an algorithm are plotted along the two axis and as with a normal ROC diagram we want the points to lie as close to the top left corner as possible, as this signifies a large benefit with a small cost. We calculate the distance from each point to this corner and use it as a final score, that can be used to measure how well that specific configuration performs.

3.4.4 Testing

For each algorithm, in each iteration of the 5-fold cross validation, we determine the optimal choice of parameter(s) and then use this configuration to classify the test-set. Obviously, we want the resulting distance-score to be as small as possible as this signifies a good performance, but we also try to capture some of the side-effects of the specific configuration. One of the

things we are interrested in knowing is how many frames long the average good video-clip is. This in itself does not tell us much, since even a very good classifier could end up with very short video-clips if the footage it was classifying is very bad. However, if several algorithms perform equally well this could work as a tie-breaker, as we in general prefer long good-quality clips since they give us more room for manoeuvre later on when we have to actually segment the footage. The variance of this length along with the median lenght is also calculated.

3.4.5 Robustness

On top of the ordinary tweaking and testing we also attempt to measure the overall robustness of each of the algorithms with respect to the change of parameter. This is done by calculating the area under the curve (AUC) of the cost/benefit diagrams generated by each algorithm. For this we use the entire dataset, which is to say that it happens after the 5-fold cross validation. Any algorithm, whos benefit-value is generally larger than its corresponding cost-value, across all choices of parameter(s) will have an AUC larger than 0.5. The larger the AUC is the more robust the algorithm is toward changes to its parameter(s). Since the points produced by most of the algorithms are not evenly spread over the entire diagram, we insert points at the (0,0) and (1,1) coordinate in order to cover the entire span of the diagram along the x-axis. It can be argued whether this is the most correct way to handle this, but it does produce comparable AUCs across all algorithms. Unlike the tweaking and testing described above this does not tell us anything useful about the best parameter to choose since we have no way of knowing how this would generalise. But it does tell us something about the nature of the algorithm itself. In general we would prefer our algorithm of choice to be as robust as possible as it signifies a sensible selection of parameters.

3.4.6 Multi-parameter algorithms

All our algorithms, except one, only have one parameter. This makes them easy to tweak and test and also very straightforward to generate cost/benefit diagrams for. The *ICSM* algorithm, however, has two parameters and this causes some problems. First and foremost the additional parameter makes tweaking much more computational expensive. With the other algorithms we tweak the one parameter by one percentage point in each iteration, which

limits the number of configuration to a hundred. To do so with the *ICSM* algorithm would require 10.000 tweak iterations, which is much too expensive even when the results of intermediate calculations are cached on disk. In order to bring this number down we start we start out by attempting to learn something about the base characteristics of the algorithm by probing it using some uniformly selected configurations. In the case of the *ICSM* algorithm we see that the *contrast* parameter has a huge influence on how well the algorithm is defined in the cost/benefit spectrum. For very low values of *contrast* almost all results are undefined because all frames are marked as good, which makes it impossible to calculate precision and recall correctly. On the other end of the spectrum, when *contrast* is large, the effect of changing it is very small, signifying that it converges toward some optimal value. Based on this information we choose to decrease the precision with which we tweak *contrast* to ten percentage points per iteration. This brings the total number of tweak iterations down to 1000. This still takes a long time, but it is manageable.

The second problem is encountered when generating the cost/benefit diagrams and calculating the AUC. The effect of changing more than one variable is not illustrated very well in a two-dimensional cost/benefit diagram. The points, which really belong in a higher dimensional space, are flattened into the plane resulting in several curves without any clear connection to each other. This also causes problems when calculating the AUC for the algorithm, since curves generated by bad performing configurations can be hidden below curves generated by good performing ones, which in the end results in a over-optimistic robustness score. In order to overcome this problem we simulate having only one parameter by locking the second one (contrast) to a specific value and then performing our analysis as usual. We repeat this over and over again, locking the contrast parameter to a new value in each iteration until we have covered all configuration combinations. For the robustness analysis we then add an aditional axis, representing the locked parameter, to our cost/benefit diagram and draw our points in three dimensions. The plane created illustrates the robustness of the algorithm much better. Since the choice of contrast values across all iterations are uniformly distributed, we can easily determine the AUC for the algorithm by simply calculating the individual AUCs for each selection of contrast value and compute an average.

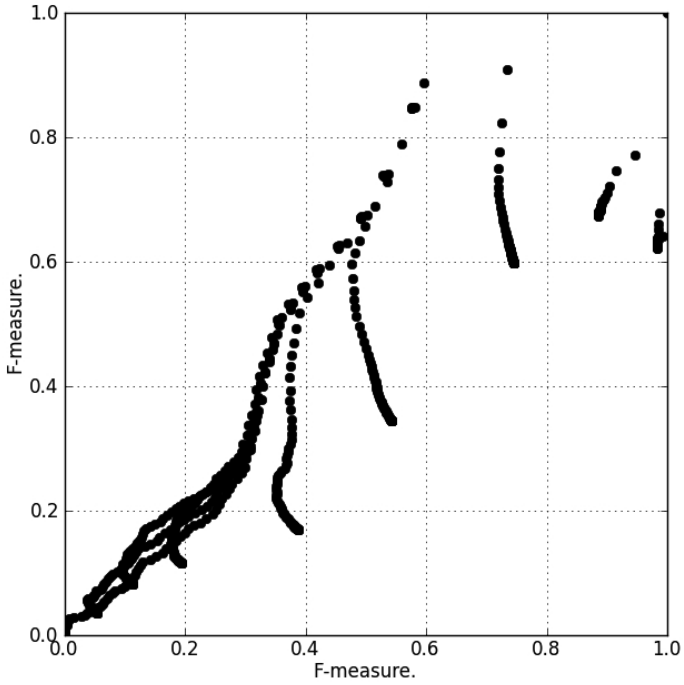

Figure 2: An example of a 2D cost/benefit diagram for an algorithm with two variables

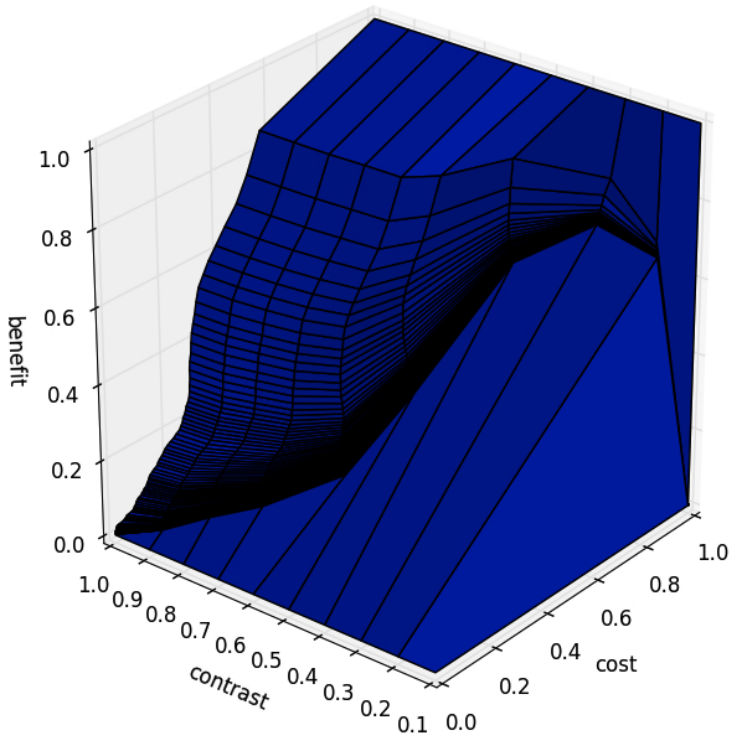

Figure 3: An example of a 3D cost/benefit diagram

3.4.7 Test cases

Frame by frame classification (FFC)
The first test case only looks at each frame independently. For each frame we determine the level of contrast and the shift vector magnitude and use this information solely to perform a binary classification of the frame as good or bad using the algorithms described above. The results are compared with our gold standard for the videos and the amount of true and false positives and negatives are used to calculate precision and recall for both classes.

Temporal classification (TC)
In the second test case we extend the classification to take the surrounding frames into account as well. Since the human eye does not perceive a movie as a sequence of frames but rather as a moving picture, we expect good and bad frames to show up in chunks. That is, we expect single bad frames to be ignored by a human watcher, if all their surrounding frames are generally good. We assume the same for single good frames surrounded by a lot of bad frames. In this test case the goal is to see how well the algorithms are able to switch into the correct state and stay there for its entirety. In order take this into account we have come up with the following method of measurement:

We start by identifying all the positions in the sequence of frames where the state changes from good to bad or vice versa. These are the areas where we are tolerant of mistakes by the classifier, that is, we will not punish it very hard for misclassifications around these positions. The areas between the the pointers correspond to the sub-sequence of the video where we stay in a state. Misclassification in these areas are not tolerated and should be punished harder. They way we accomplish this is by generating a *penalty function* defined as:

$$P(x) = 1 - MAX_{b \in B}(e^{-\frac{(x-b)^2}{2c^2}})$$

The content of the MAX clause is just a Gaussian function, x is the position at which we want to know how much we should punish mistakes, B is a set of all positions where a change of state happens and c is the level of tolerance we have for mistakes in general. The further away from a change of state a misclassification is the more it is punished. See [fig reference]. What the function does is that i places a reverse Gaussian bell curve at each position of change of state. *P(x)* always returns a value between 0 and 1, that tells

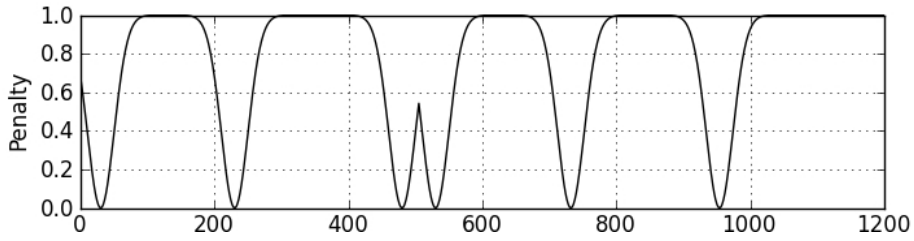

Figure 4: An illustration of a curve generated by the penalty function

us how severe a misclassification in this area is. All correct classifications are treated just like in the FFC test case. However, if a misclassification occurs, we look at the *penalty function* to see how the classification should be scored. However, this makes maintaining accurate true and false negative and positive values slightly less trivial. If a misclassification occurs at frame x and *P(x)* is very close to 0, how true (or false) is the classification)? The way we handle this is simply by adding *P(x)* to the *false* ratio for this type (positive or negative) and *1 - P(X)* to the opposite type. That is, we quantify the classification as being both true and false to a varying degree determined by the *penalty function*.

Smoothing of the algorithm suggestions
Finally, for both algorithms, we also investigate the effect of smoothing the classification results. This is done to remove outliers in the final results and should not be confused with the smoothing of the *contrast* and *shift vector magnitudes* in the algorithms themselves.

3.4.8 Results

In the table below we have a number of different permutations of algorithms. The parameter is the average over the 5-fold cross validation. Smoothness degree describes how much the result is smoothed using triangular smoothing. The comparison method is either frame-by-frame (FFC) or temporal (TC), see section 3.4.7. The parameters relative standard deviation is a measure of how much the parameter generalize (a low standard deviation means high level of generalization). The distance score is explained in section 3.4.3, and AUC in section 3.4.5. A good sequence length (GSL) is the number of consecutive good frames, and a bad sequence length (BSL) is the number

Label	Algorithm	Smoothness degree	Comparison method	Tolerance	Parameter(s)
ICSM1	ICSM	0	FFC	N/A	(0.052, 0.720)
ICSM2	ICSM	12	FFC	N/A	(0.054, 0.720)
ICSM3	ICSM	0	TC	24	(0.060,0.720)
ICSM4	ICSM	0	TC	48	(0.064,0.720)
ICSM5	ICSM	12	TC	24	(0.058,0.720)
ICSM6	ICSM	12	TC	48	(0.064, 0.720)
ICSM7	ICSM	0	TC	12	(0.056, 0.720)
CCSM1	CCSM	0	FFC	N/A	0.092
CCSM2	CCSM	12	FFC	N/A	0.088
CCSM3	CCSM	0	TC	24	0.100
CCSM4	CCSM	0	TC	48	0.108
CCSM5	CCSM	12	TC	24	0.094
CCSM6	CCSM	12	TC	48	0.110
CCSM7	CCSM	0	TC	12	0.098
CO1	CO	0	FFC	N/A	0.462
CO2	CO	12	FFC	N/A	0.460
CO3	CO	0	TC	24	0.466
CO4	CO	0	TC	48	0.474
CO5	CO	12	TC	24	0.468
CO6	CO	12	TC	48	0.474
CO7	CO	0	TC	12	0.466
SO1	SO	0	FFC	N/A	0.048
SO2	SO	12	FFC	N/A	0.048
SO3	SO	0	TC	24	0.056
SO4	SO	0	TC	48	0.060
SO5	SO	12	TC	24	0.056
SO6	SO	12	TC	48	0.060
SO7	SO	0	TC	12	0.054
$CSM^3$1	CSM^3	0	FFC	N/A	0.366
$CSM^3$2	CSM^3	12	FFC	N/A	0.366
$CSM^3$3	CSM^3	0	TC	24	0.370
$CSM^3$4	CSM^3	0	TC	48	0.388
$CSM^3$5	CSM^3	12	TC	24	0.370
$CSM^3$6	CSM^3	12	TC	48	0.382
$CSM^3$7	CSM^3	0	TC	12	0.370

Table 1: Algorithm configurations

Label	Parameter(s) std. dev. %	Distance score (DS)	DS std. dev. %	AUC
ICSM1	(7.7%, 5.6%)	0.54	27.17%	0.51 (39.44%)
CCSM1	4.3%	0.54	27.90%	0.66
CO1	2.5%	0.81	28.10%	0.40
SO1	15.6%	0.55	27.08%	0.65
$CSM^3 1$	2.2%	0.76	30.27%	0.44
ICSM2	(14.8%, 5.6%)	0.54	27.52%	0.51 (39.69%)
CCSM2	19.6%	0.54	27.85%	0.67
CO2	2.4%	0.81	28.21%	0.40
SO2	15.6%	0.54	27.54%	0.65
$CSM^3 2$	2.2%	0.76	30.87%	0.44
ICSM3	(18.3%, 5.6%)	0.29	31.10%	0.66 (41.71%)
CCSM3	15.5%	0.30	34.30%	0.87
CO3	1.1%	0.50	28.73%	0.66
SO3	14.3%	0.30	31.28%	0.86
$CSM^3 3$	0.0%	0.45	31.90%	0.70
ICSM4	(15.9%, 5.6%)	0.21	34.29%	0.70 (42.57%)
CCSM4	17.0%	0.22	34.24%	0.92
CO4	2.9%	0.39	18.54%	0.79
SO4	18.3%	0.22	35.54%	0.91
$CSM^3 4$	3.8%	0.36	20.37%	0.81
ICSM5	(20.1%, 5.6%)	0.29	32.02%	0.66 (41.78%)
CCSM5	19.7%	0.30	33.81%	0.87
CO5	0.9%	0.49	28.31%	0.66
SO5	14.3%	0.29	32.22%	0.86
$CSM^3 5$	0.0%	0.45	32.62%	0.70
ICSM6	(15.9%, 5.6%)	0.21	35.18%	0.70 (42.62%)
CCSM6	19.1%	0.22	35.00%	0.92
CO6	2.9%	0.40	18.65%	0.79
SO6	18.3%	0.22	36.61%	0.91
$CSM^3 6$	18.3%	0.22	36.61%	0.91
ICSM7	(14.3%, 5.6%)	0.36	29.34%	0.62 (41.44%)
CCSM7	11.9%	0.36	31.71%	0.82
CO7	1.1%	0.60	26.18%	0.56
SO7	14.8%	0.36	30.14%	0.81
$CSM^3 7$	0.0%	0.55	28.93%	0.60

Table 2: Algorithm performance

of consecutive bad frames. Both measures is an average (over each test-set) and the relative standard deviation along with the longest GSL and BSL is also supplied in Table **??**.

3.4.9 Analysis of results

3.4.10 Analysis risks

In order to limit the scope of the testing and analysis phase we make some decisions which should be noted as a lot of areas related to determining the best algorithm and the optimal algorithm-parameter combination are left to be explored:

All our algorithms square the *contrast* and *shift vector magnitude* values they come up with for each frame. The reason for this is that we want to increase their sensitivity toward bad values as compared to good ones. However, we have not done any real experimentation in this context and it is possible that a higher or lower power would yield better results. Furthermore we smooth these values across neighbouring frames in order to eliminate the worst outliers. It does not seem to have any significant effect on the final correctness of the algorithms, but this is neither an area in which we have done any real experimentation. Another important thing to remember is that we actually compute *contrast* differently than Girgensohn et al. [1]. Our method should be more robust toward very uniformly colored frames, but their original method could be more effective in general.

When generating the cost and benefit scores we calculate the harmonic mean between the positive precision and the negative recall, as well as the positive recall and the negative precision, respectively. In our experimentation we have simply weighted the two values in each pair evenly, but it is possible that a skewed weighting in one or both of means would perform differently. It is even possible that this pair combination is not the right one for the task, although we argue that it intuitively makes sense.

In our experiment with smoothing the classification suggestions generated by the algorithm, in order to remove outliers, we restrict ourselves to only testing two different degrees of smoothing. More could be learned by experimenting with this value as an additional parameter of the algorithms.

Label	GSL	GSL std. dev. %	longest GSL	BSL	BSL std. dev. %	longest BSL
ICSM1	205	217%	5105	40	155%	536
CCSM1	214	214%	5121	40	160%	607
CO1	290	28%	11079	73	196%	1690
SO1	198	221%	5105	40	157%	743
$CSM^3 1$	264	326%	11078	69	204%	1691
ICSM2	248	199%	5115	47	152%	608
CCSM2	245	200%	5118	47	164%	739
CO2	328	317%	11079	84	185%	1718
SO2	231	206%	5115	47	170%	1087
$CSM^3 2$	309	301%	11078	80	190%	1692
ICSM3	221	210%	5105	39	156%	509
CCSM3	222	210%	5121	39	162%	607
CO3	340	308%	11079	69	215%	1690
SO3	209	214%	5105	38	147%	491
$CSM^3 3$	319	298%	11078	66	223%	1691
ICSM4	229	209%	5122	39	149%	509
CCSM4	225	213%	5123	38	154%	537
CO4	378	316%	12080	71	217%	1690
SO4	216	209%	5122	37	147%	482
$CSM^3 4$	388	280%	12082	59	225%	1687
ICSM5	257	186%	5115	47	149%	608
CCSM5	248	198%	5118	46	156%	739
CO5	411	290%	11079	80	202%	1690
SO5	246	200%	5115	45	143%	608
$CSM^3 5$	364	280%	11078	75	211%	1692
ICSM6	266	198%	5122	45	144%	534
CCSM6	260	200%	5129	45	142%	537
CO6	431	298%	12080	82	201%	1690
SO6	259	195%	5122	45	139%	482
$CSM^3 6$	397	284%	12082	74	215%	1687
ICSM7	212	213%	5105	39	153%	510
CCSM7	220	212%	5121	39	162%	607
CO7	340	308%	11079	69	215%	1690
SO7	208	214%	5105	39	149%	491
$CSM^3 7$	319	298%	11078	66	223%	1691

Table 3: Sequence data

The same can be said about the tolerance-value in the Temporal Classification test case (page 23). Exactly what degree of tolerance is the most reasonable is up for debate and any changes could, and probably would, change the outcome of our results.

Also mentioned ealier, when calculating the AUC's for each algorithm we insert the points (0,0) (1,1) along with the points generated by the cost/benefit calculations in order to make sure each algorithm's curve cover the entire span of the x-axis. This has the effect of normalising the AUC around 0.5, which is useful for AUCs based on a curve that is only defined partly along the x-axis. It is however possible, that the very fact that a curve is not defined all along the x-axis is a sign of low robustness especially if the curve only is present near x = 1, that is, if the cost value is generally very high. If this is the case we risk ending up with invalid assumptions of robustness.

In the *ICSM* algorithm we chose to simplify our tweaking process by decreasing the precision on the second parameter, *contrast*. The rationale for this is described in section 3.4.6 on page 19. This means that our understanding of the influence of contrast in this algorithm is less detailed.

Finally, it should be mentioned that all our tests are performed by treating the quality assessment problem as a task of binary classification. As described in section 1.1.1 on page 6, we are actually not interested in such a hard classification in the final system. Rather we want the quality of a region of video-footage to be expressed as a soft (un)suitability curve, from which we can make some general assumptions. However, the nature of our tests, especially the way in which we generate our gold standard (section 3.3 on page 13), requires us to simplify the problem into this binary form. Some of the aspects of the problem, which are lost by doing this, are to some extent accounted for, for example in the Temporal Classification test case (page 23), which allows for misclassifications to be treated in a more soft manner, but others are undeniably lost. This should also be taken into account, when considering the generalisability of our results.

3.5 Section summary

4 Grouping and labeling

4.1 Detecting interesting regions in video-clips

Detecting suitable regions in the video-clips is not enough. The content in the regions also has to be interesting. Hanjalic, A. [2] describes a way to identify such regions by measuring the level of excitement in a sports video-clip based on a manually selected set of features. These features can be both visuals (like the movement in an image or the change of camera positions) or audial (namely the energy contained in the audio track).

5 Conclusion

6 Discussion NOT DONE, ONLY NOTES FOR LATER

Had we had our own video hosting service we would have access to such information.Had we further had our own client application in which all video was recorded we would even be able to acquire additional information about the circumstances in which the footage was recorded. Examples of this could be information about the movement of the camera (from accelerometers) and proximity to other devices recording the same scene (through Bluetooth). The purpose of our method for building our data set has been to simulate that we do in fact have access to some of this information, as well as making it as easy as possible to estimate the rest.

References

[1] Andreas Girgensohn, John Boreczky, Patrick Chiu, John Doherty, Jonathan Foote, Gene Golovchinsky, Shingo Uchihashi, and Lynn Wilcox. A semi-automatic approach to home video editing. In *Proceedings of the 13th annual ACM symposium on User interface software and technology*, UIST '00, pages 81–89, New York, NY, USA, 2000. ACM.

[2] A. Hanjalic. Adaptive Extraction of Highlights From a Sport Video Based on Excitement Modeling. *Multimedia, IEEE Transactions on*, 7(6):1114–1122, 2005.

[3] Si Wu, Yu-Fei Ma, and Hong-Jiang Zhang. Video quality classification based home video segmentation. *Multimedia and Expo, IEEE International Conference on*, 0:4 pp., 2005.