Shot Boundary Detection

Seminar
Practical Video Analyses
Summer Semester 2015

Tanja Bergmann, Stefan Bunk, Philipp Otto,
Max Reimann, Felix Wolff

Supervisor:
Dr. Haojin Yang
Prof. Dr. Christoph Meinel

October 21, 2015

Contents

1 Introduction

Videos and movies usually consist of many independent sequences of frames (shots). In the final videos, these shots are then cut one after another, creating the actual movie. The goal of *shot boundary detection* is to detect these cuts, without relying on metadata, solely by looking at the frames of the video. Shot boundary detection is an important preprocessing step for many other video processing tasks, such as video annotation and classification.

An important distinction has to be made between hard cuts and soft cuts. On the one hand, hard cuts are abrupt changes from one scene to another. Only two frames are involved in a hard cut, namely the last frame of the ending scene and the first frame of the new scene. On the other hand, soft cuts are longer transitions from one shot to another, either by mixing the two scenes frame by frame (*dissolve*), or by using other blend effects, such as swipes. Soft cuts can be of arbitrary length and all frames from the last unchanged frame of the old scene to the first unchanged frame of the new scene are considered part of a soft cut. Sometimes, even the rate of change (interpolation between two images, or speed of the swipe effect) can change inside a soft cut.

Shot boundary detection is an easy task for humans. However, computers must rely on changes in color, edges, or other approaches based on the raw pixel values. Especially if the changes are small, e.g., from one dark scene to another, the problem becomes harder for computers. Humans can still detect cuts in these cases, because they can take the context into account.

In this work we approached both hard cut detection and soft cut detection. Our hard cut detection works with traditional approaches from literature, concretely color histograms and machine learning. The soft cut detection uses a new approach: deep artificial neural networks. Deep neural networks have seen a rise in popularity in the computer vision community in the last years. This rise is mostly due to impressive improvements in image and video classification tasks. In this work, we want to automatically learn the characteristics of soft cuts via artificial neural networks. To the best of our knowledge, no one has tried this approach for the soft cut detection problem so far.

This work is organized as follows: Section 2 shows related work for both hard cuts and soft cuts, and gives a short introduction into deep artificial neural networks. In Section 3 we focus on our results for hard cut detection, while Section 4 focuses on soft cut detection. For both we illustrate, how we preprocessed the given data and evaluated our results. Finally, in Section 5 we show, what would be the best steps to improve on our results in future work.

2 Related Work

This section gives an overview about state of the art approaches for cut detection. Afterwards, we give a short introduction to deep neural networks.

2.1 Shot Boundary Detection

Many different techniques have been developed for shot boundary detection. For hard cut detection, a baseline approach compares the change between two frames for each pixel, and builds the sum of these differences. Then a threshold is selected: Every value over the threshold is considered a cut. However, this approach is prone to error, when there are quick camera movements or sudden color changes inside a scene. Therefore, better approaches have been developed, which are summarized in the next paragraphs.

Color Histograms This approach, first used by Zhang et.al. [**?**], sorts the color values into a histogram with a certain bin size, before comparing the histograms. Thus, this approach is less prone to errors than the baseline approach. It can better compensate for minor changes in the frames. However, using histograms also means compressing the data, thereby throwing some information away.

Luminance Values Instead of working on the RGB data, one can also work in other color spaces. Especially luminance values [**?**] have shown good performance, as they emphasize differences in brightness. Thus, they are less susceptible to false positives.

Edge Detection Another approach is to detect the edges in a frame and the subsequent frame. If the edges differ fundamentally, this is a sign for a hard cut. In [**?**], the authors use "edge histograms of Sobel-filtered (vertically and horizontally) DC-frames".

Other Techniques There are many more fine-grained ideas, which can be used to refine the result. One is motion compensation, used by Quśenot et.al. [**?**], to prevent false positives from camera movements. Another one is the idea of adaptive thresholding [**?**], which does not use one fixed threshold but rather adapts the threshold dynamically, based on the current sequence.

Machine Learning For finding exact thresholds or decision boundaries, many approaches employ machine learning, especially by using a support vector machine (SVM). This works by extracting features, such as histogram bins and edges and then pass these to a machine learning algorithm. SVMs are popular, because they are "easy to use and provide a quick classification time after the initial training" [**?**].

It is also possible to combine many of the approaches from above. For finding soft cuts, the state of the art technique is to use a sliding window of frames. The decision

from many frame-to-frame comparisons can then be incorporated into a final soft cut prediction.

2.2 Deep Neural Networks

Deep neural networks were very successful in image classification [**?**] and video classification [**?**] tasks in the last years. In the ImageNet Large Scale Visual Recognition Challenge 2014 (ILSVRC2014) [**?**], almost all contestants were using neural networks. This recent rise in popularity in the computer vision community has been fueled by two recent developments. First, the rise in computing power, especially of GPUs, has enabled reasonable training times. Second, with more and more data being collected, more training data is available, which is crucial for the success of these techniques.

Neural networks work by processing the raw input pixel values layer by layer, successively building more abstract features. When building networks with many layers, this idea is called deep learning. Two special network architectures are important for the success of deep learning. *Convolutional networks* [**?**] process the image by looking repeatedly at small patches of the input. This is good at finding locally restricted features, and is especially good for spatial data, such as images or videos. One important property of deep learning (as opposed to shallow learning) is that features are not explicitly fed into the algorithm. Instead, the algorithm learns the features by itself.

Another important technique are *recurrent neural networks*. In contrast to normal neural networks, recurrent networks also have backwards connections in their architecture. Concretely, this means that the activation values from one input image are also feed to the next input image, thereby allowing the network to build up a "short term memory". This technique has proven useful in video classification [**?**]. To know, when a new sequence starts, a recurrent networks needs a binary tagging sequence, which indicates, when a new sequence starts.

While deep learning was popular in the last years, research in shot boundary detection, on the contrary, has ceased. From 2001 until 2007, the TrecVid [**?**] conference series was hosting tracks on shot boundary detection. The TrecVid committee provided a video test collection with manually annotated gold data, which could be used by different research groups to develop and test their algorithms. The tasks range from instance detection (e.g., detecting a person in a video), semantic indexing, event detection, and many more. However, as told before, the last challenge for shot boundary detection was in 2007.

We think this decline in research is largely, because the current approaches are highly developed, and there is not much potential for further improvements. Since the current techniques are all based on shallow learning, we propose to use deep learning approaches for shot boundary detection. The next sections detail our approach.

3 Hard Cut Detection

For our hard cut detection implementation, we chose a standard machine learning approach. We extract features from the frames and use them to train an SVM. The following section explains steps we took for that, presents the problems we encountered, and evaluates our final results.

3.1 Approach

Hard cuts are abrupt transitions from one scene to another. Normally the contents of the two frames involved in such a cut are highly different, see Figure 1, while two consecutive frames in one scene do not differ that much.

Fig. 1: These two consecutive frames form a hard cut.

To detect hard cuts, we therefore want to apply a similarity measure to two subsequent frames. In our case, we represent frames using their color histograms. Each frame has three color channels (red, green, blue) and therefore three histograms. The difference between two frames is then the difference between the histograms. The histogram difference can be represented as a $3*n$-dimensional vector, where n is the number of bins in a color histogram. So, we are presented with a binary classification task (cut or not) in a $3*n$-dimensional space.

In a next step, the dimensionality can be further reduced by simply adding up the vector elements. This step is justifiable, since it does not matter which color changed and how much, but the sum of changes in all color channels does. Our results with the reduced dimensionality were better than those of the high dimensional approach. That's why we use the latter.

To do the actual classification we train an SVM classifier on a labeled training set. In our implementation, we use the SVM provided by the OpenCV library[1]. We transform the input space into a higher dimensional feature space by using a kernelized decision function. The commonly used radial basis function (RBF) kernel is employed:

$$K(x_i, x_j) = exp(-\lambda ||x_i - x_j||^2)$$

[1] `http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`

where λ denotes the width of the kernel and x_i,x_j are vectors from the training set. This is part of the library. The complexity parameter C for the soft-margin SVM, as well as other parameters are optimized automatically by using the *train_auto* method of OpenCVs SVM implementation. It automatically performs a k-fold cross-validation to choose the best parameter values.

3.2 Visualization

After having implemented the ideas presented in 3.1, the results were not breathtaking. Therefore we wanted to receive a feeling for the flaws in our approach by visualizing the features and the decisions that the SVM made. To make the visualization explorable and interactive, we build a web app written in *HTML* and *JavaScript* using the visualization framework *d3*[2]. During hard cut detection, we write a *tsv* file containing all required information, such as the histogram differences, the predicted class, and the actual class for every two subsequent frames. Then a local webserver is started, which has access to the *tsv* file as well as the video frames. The visualization can be explored using a web browser (see Figure 2). When inspecting the visualization of a processed video, we can

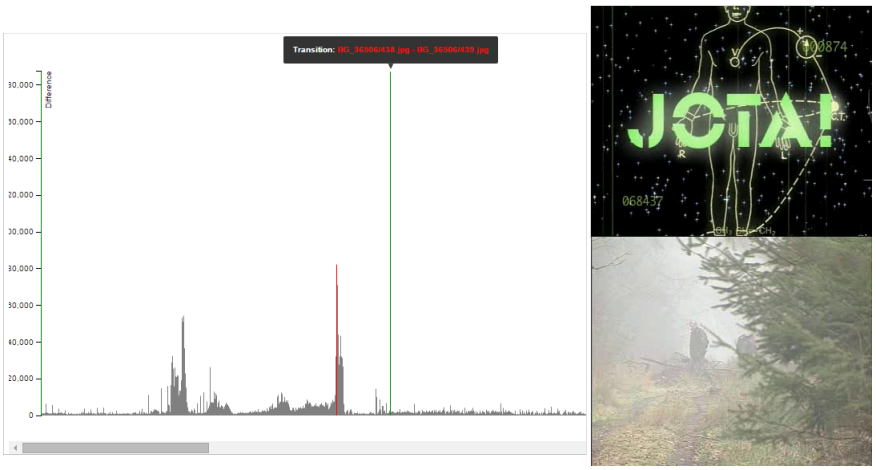

Fig. 2: The height of the bars shows the absolute histogram difference. It is calculated by summing up all elements in the histogram difference vector. The color of the bars indicates the decision: green – true positive, gray – true negative, red – false positive, orange – false negative (not in this picture). When hovering over the bars, the actual frames are shown on the right, together with a tooltip containing the file names.

see that the histogram differences are a useful feature for detecting the hard cuts in a video. In most cases there is one single bar with a high difference surrounded by low differences (see green bar in Figure 2). However, we can also see some other peaks that

[2] `http://d3js.org/`

are not hard cuts and are typically surrounded by noisy clusters (see Figure 3). Those can be soft cut sequences or just rapid changes during one scene. Since the SVM just takes the difference between two frames into account, it does not "see" the surrounding clusters and therefore makes several mistakes. Nevertheless, we can improve the classification performance by showing the SVM more soft cuts and frame sequences, where the colors change dramatically. When learning these negative examples, the decisions made by the SVM are more robust against noisy sequences. We make fewer mistakes on the noise in Figure 2, but high peaks, such as in Figure 3, are still problematic.

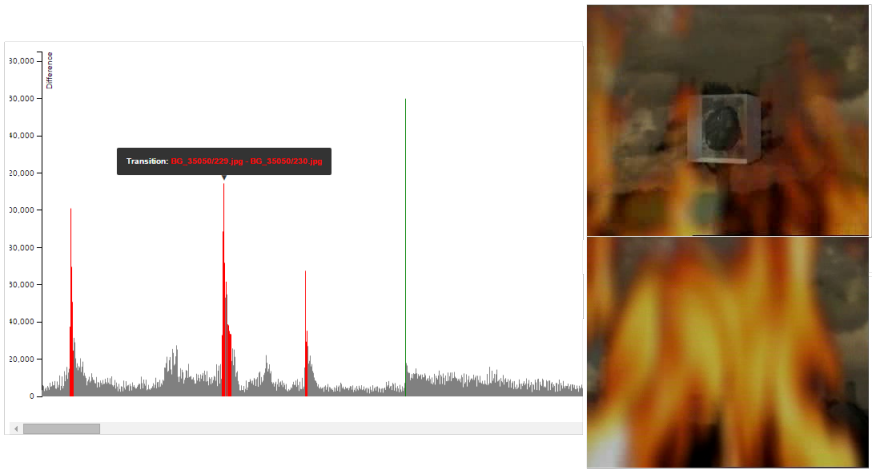

Fig. 3: The histogram differences are problematic when there is lots of noise between two frames. In this case, a burning fire disturbs the classification and introduces lots of false positives.

3.3 Evaluation

The SVM is trained on a mixture of videos from the 2007 TrecVid contest. The videos in their full length do not make a good training set, as the ratio between hard cuts and other frames is about `1:1000`, leading to a failure to predict any hard cuts. A custom training set must be constructed, which contains a sufficient amount of hard cuts. The choice of non hard cuts is also crucial for achieving good classification performance: too homogeneous frames lower the decision boundary, and lead to poor performance on noisy examples. Therefore the negative set should also include soft cuts and particularly noisy frames. The training set we finally build consists of about 250 hard cuts and 2,200 negatives.

We evaluate our approach on complete, held out videos of the 2007 TrecVid contest (with the videos from training data being removed). The results per video are shown in Figure 4, showing that a high recall is consistently achieved, while the precision is dependent on the video. The overall statistic, shown in Table 1, reveals that the objective

of further improvements must be to reduce the number of false positives. In Section 5.2, we present some ideas for future work on this.

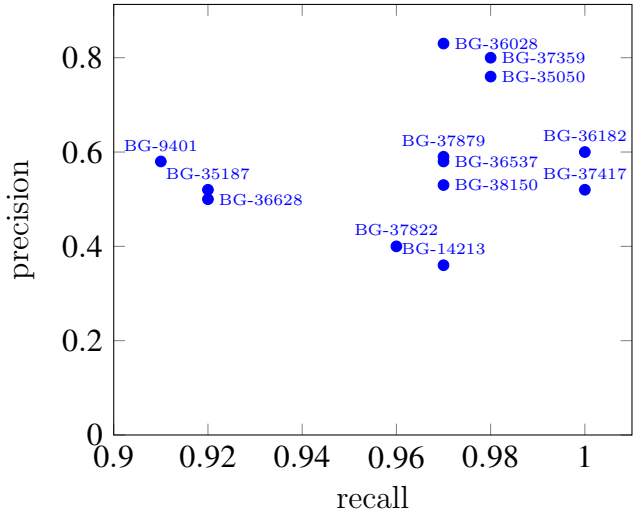

Fig. 4: Results on TrecVid 2007 Videos

Precision	Recall	F1	TP	FP	TN	FN
0.55%	0.96%	0.7%	1661	1358	532759	69

Table 1: Results on complete 2007 TrecVid dataset

4 Soft Cut Detection

In contrast to the traditional approaches presented in Section 2, we chose a deep learning approach to detect soft cuts in a video. In the following subsections, this approach is presented in more detail.

4.1 Data Generation

The raw data set from the 2007 TrecVid contest contains 1,592 soft cut frames and 637,755 non soft cut frames. This is a highly imbalanced ratio, which impedes the training. Also, the number of soft cuts in the data set is too small for training a deep neural net. As stated by Greg Corrado, senior research scientist at Google: "Training deep learning models really does take a lot of data" [**?**]. Therefore, we generated more soft cuts by blending random sequences into each other.

The procedure for generating a new random sequence based on the data set provided by the 2007 TrecVid contest works as follows: First, we randomly pick two subsequent

cuts from the gold standard provided by the TrecVid contest, each with a start and end frame respectively. Between the end frame of the first cut and the start frame of the next cut, we randomly select a subsequence with the desired transition length. Because there is no hard or soft cut between two subsequent cuts in the gold standard, it is guaranteed that there is also no hard or soft cut in the random sequence we picked. Afterwards, we repeat this process with two other random cuts. As the result, we now have two randomly selected sequences of the same length, which we now blend into each other. To blend two random sequences, we have multiple options for tweening behaviour. The standard tweening function is linear, but we also used ease-in, ease-out, and others. The tweening function is randomly selected, as well. As the transition type we use a classic *dissolve*. To achieve further variance in the generated data, we flip the two sequences randomly at the x- or y-axis or at both axes. With this approach we generated about 450,000 sequences of length 11 and length 21 each. Length 11 was chosen as a first approach, because it is slightly larger than the minimum size of soft cuts. The intuition was, that also the smallest soft cuts can be detected with this approach. Length 21 was chosen, because it is the average soft cut size in the original data. In each case 50% of the the sequences are non cuts and 50% are soft cuts. This makes a total of around 7,200,000 soft cut frames and 7,200,000 non cut frames.

4.2 Approach

In the following sections we first present different merging strategies to get a single per-frame prediction. Then, we present a technique for increasing the precision or recall of the merging results. Finally, we present different network architectures we evaluated.

4.2.1 Merging Strategies

For the soft cut detection we decided to use a deep learning approach. More concretely, we used the RNN/LSTM implementation by Jeff Donahue[3] for the Caffe[4] framework. This LSTM implementation takes two different inputs: On the one hand the raw pixel values and on the other hand a tagging sequence. The tagging sequence tells the LSTM, where a new training example starts, as we process more than one training sequence per batch. The LSTM implementation allows us to use a short term memory in the neural network. The network remembers information from the beginning of a frame sequence until the last frame of that sequence. So the net remembers previous decisions through a sequence of frames.

But using this architecture has one problem, as stated by Jeff Donahue: "Backpropagation [through the LSTM] is truncated along the batch boundaries" [**?**]. So one or more frame sequences have to fit exactly into the batch size used by the LSTM. This is hard to achieve, if we want to classify variable-length frame sequences. Therefore we

[3] `https://github.com/BVLC/caffe/pull/2033`
[4] `http://caffe.berkeleyvision.org/`

decided to use a fixed size for the sequences of frames in our training data, i.e., we would generate only transitions of length 10 in our training data.

However, we still want to find soft cuts of arbitrary length in a video. To achieve this, we repeatedly test fixed-size frame sequences. In Figure 5, we show an example. We

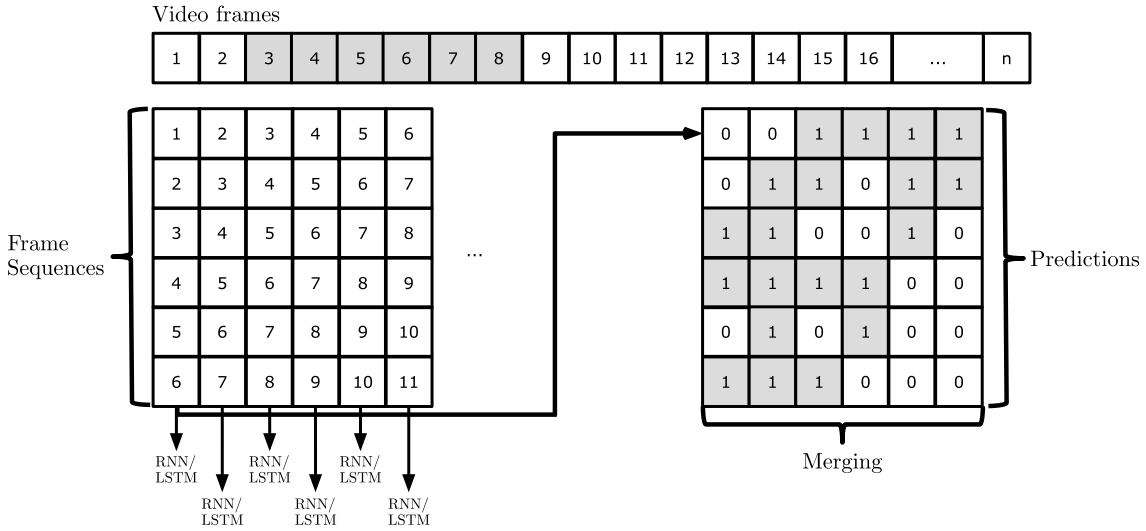

Fig. 5: To classify soft cuts of arbitrary length, we repeatedly test fixed-size frame sequences. In this example we test sequences of size 6. Afterwards, the predictions given by the LSTM are merged, so that we have one prediction per frame.

have a video with n frames. The frames from three to eight represent a soft cut. Now, for each frame, we generate a frame sequence of size six starting from that frame. This is equivalent to moving a sliding window over the frames. Those sequences are then classified by the LSTM. The output of the LSTM is zero, if a frame does not belong to a soft cut, and one, otherwise. In the end, we have up to six predictions per frame, which have to be merged, so that we have only one prediction per frame. After merging, consecutive frames with a predicted value of 1 represent one soft cut.

In the following, several strategies for combining multiple frame predictions into one prediction are presented. An overview over all strategies can be found in Figure 6.

Take First A first simple strategy is to take the first prediction of each sequence as the prediction of that frame. This is equivalent to having no prior knowledge about the frame, as the LSTM has not seen any previous frames and therefore could not remember anything.

Take Last 'Sequence' A second simple strategy is to take the last prediction of each sequence as the prediction for the frame that started the sequence. Concretely, the last prediction is not a prediction for the frame under consideration, but the intuition behind

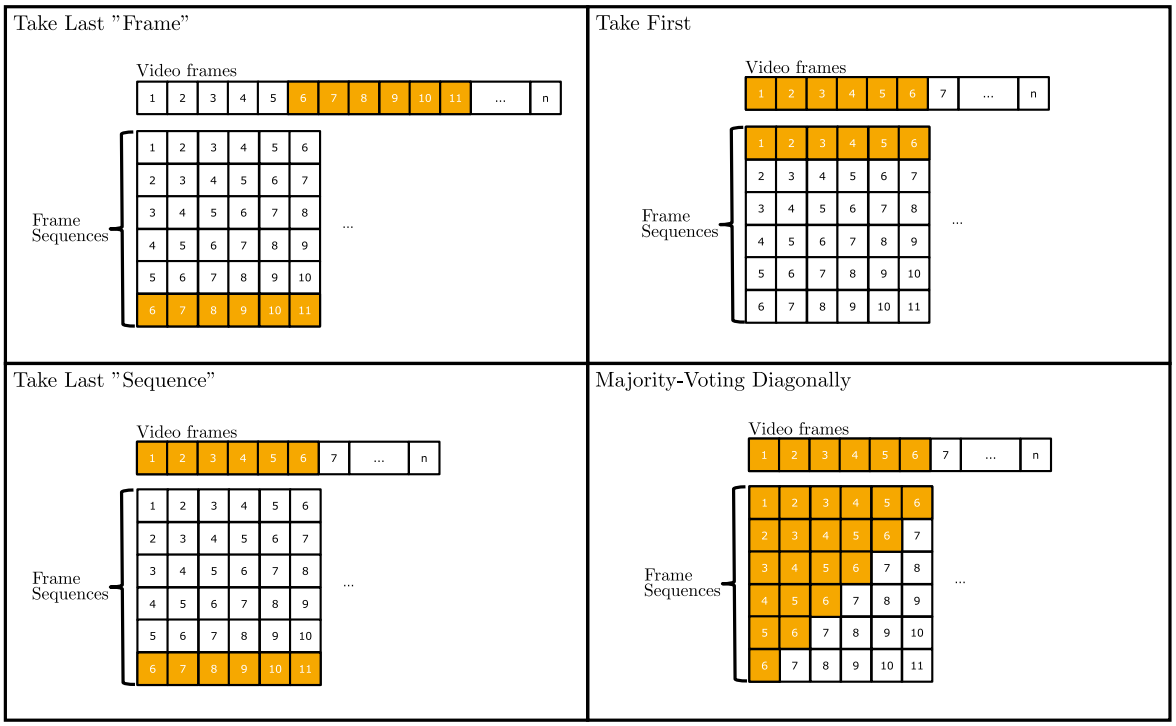

Fig. 6: To merge the multiple predictions per frame into one, we implemented several strategies: *Take Last 'Frame'* (top left): The prediction of the frame, where the frame is the last of a frame sequence, is taken. *Take Last 'Sequence'* (bottom left): The last prediction of the frame sequence belonging to a frame is taken. *Take First* (top right): The first prediction of the frame sequence is taken. *Majority-Voting Diagonally* (bottom right): The majority voting among all predictions of the frame is taken.

this is that an LSTM becomes more and more certain after having seen multiple frames of the sequence.

Take Last 'Frame' In the *Take Last 'Frame'* strategy, for every frame we take the frame sequence, in which that frame is the last one. The intuition is similar to the previous case, but now we also use the prediction for the actual frame.

Majority-Voting Diagonally Each frame is predicted up to n times. In our example it is up to six times. In the *Majority-Voting Diagonally* all of those predictions are taken and the majority voting among those is taken as prediction for the frame. Ties are resolved by placing more weight on the half of the predictions, where the frame appeared later in the sequence.

4.2.2 Gap Filler

After merging, we have one prediction per frame indicating whether the frame is part of a soft cut or not. Then, a sequence of frames that are part of a soft cut represents a soft cut. However, there could be misclassified frame predictions. We implemented a *Gap Filler* to find and correct some of those misclassified frame predictions. The *Gap Filler* is looking for sequences of frames that belong to a soft cut and are interrupted by some non cut frames. If the number of interrupting frames is not too large, they are also classified as soft cut frames. The idea behind the *Gap Filler* is that two soft cuts do not occur close to each other. So, if two soft cuts are only a few frames apart, they probably belong to the same soft cut. In Figure 7 an example of the *Gap Filler* is shown. There, a sequence of soft cut frames is interrupted by two non soft cut frames. Those non soft cut frames are detected by the *Gap Filler* and classified as soft cut frames.

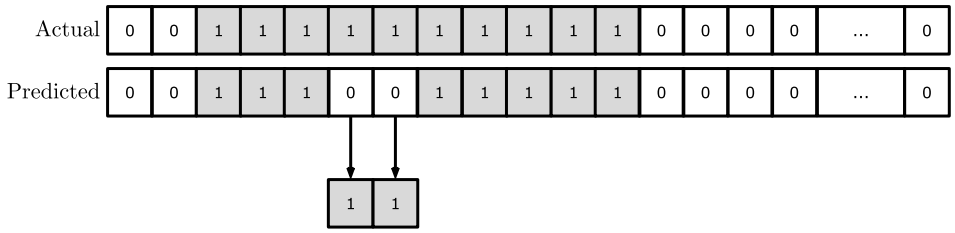

Fig. 7: Two soft cuts do not occur close to each other. Therefore the *Gap Filler* finds misclassified frame predictions after the merging step and corrects non soft cut frame predictions, if they are interrupting a soft cut.

The gap filler presented so far is recall-oriented: It increases the recall of soft cuts, while trading precision at the same time. By switching the roles of 0s and 1s, we can make the *Gap Filler* also precision-oriented: In this case, sequences of 1s, which are not long enough, are filled with 0s.

4.2.3 Network Architectures

To classify the frames as soft cut or non soft cut frames, we used an LSTM. We tested many different architectures and parameter settings. In the following we present a selection of those with significant differences:

CNN + one LSTM For the CNN we used the architecture of the *Caffenet*[5]. We also used the pre-trained weights of this net, so that we do not need to train our model from scratch. We only fine-tuned the *fc6* layer of the *Caffenet* with a learning rate of 0.1. The CNN is followed by one LSTM, whose weights were initialized with the *Xavier* method, because this method performs better than other as mentioned in [**?**]. Besides, the general learning rate was set to 0.01 and gradient clipping was used.

[5] `https://github.com/BVLC/caffe/tree/master/models/bvlc_reference_caffenet`

CNN + two LSTMs The architecture of this net is basically the same as the previous. There is only one difference: Instead of using just one LSTM, two LSTMs were used. Also the general learning rate was set to 0.001 and no gradient clipping was used. The architecture of the net can be found in Figure 8.

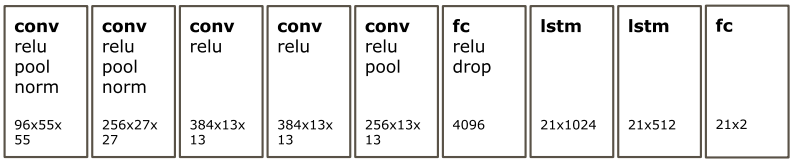

Fig. 8: Architecture of the LSTM consisting of a CNN and two LSTMs.

One convolutional layer + two LSTMs The last architecture that was tested uses only one convolutional layer before the two LSTMs. The intuition behind it is as follows: for the CNN, it is hard to detect a soft cut frame, because it works only on one frame. By looking at only one frame, it would be difficult to predict a soft cut, even for a human. However, by looking at a sequence of frames, this becomes easier. This is exactly, what the LSTM does, which is why we put more emphasis on this part and less emphasis on the CNN part. The net was trained from scratch as no pre-trained net existed. The weights of the LSTMs were again initialized using the *Xavier* method. The learning rate was 0.001 and gradient clipping was used during training.

4.3 Evaluation

First, we took a closer look at the performance of our three different neural network architectures, we just presented. To train the neural networks we used our generated data, which is explained in Section 4.1. We also used generated data for testing, to get a first impression while training the model in *Caffe*. We had about 225,000 sequences in train and test set each. The amount of soft cuts and non soft cuts in the training and test data was the same. The accuracy reported by *Caffe* during training represents the accuracy on per-frame predictions. This accuracy is shown in Table 2. The *CNN + two LSTMs* has by far the highest accuracy. Therefore, we decided to take this neural network architecture as our basis for the following evaluations.

CNN + one LSTM	69,80%
CNN + two LSTMs	80,42%
one convolutional layer + two LSTMs	70,07%

Table 2: Accuracy of the different neural network architectures given by *Caffe*.

Next we evaluated the different merging strategies. This evaluation was done on the actual video data provided by TrecVid. We used the frame sequence predictions from the

CNN + two LSTMs and merged those predictions in the four different ways, presented in the last section (see Figure 6). Afterwards, we compared the resulting frame predictions with the actual values (belongs to a soft cut or not) and calculated accuracy, precision, and recall for both classes. The results can be found in Figure 9. As the figure shows the

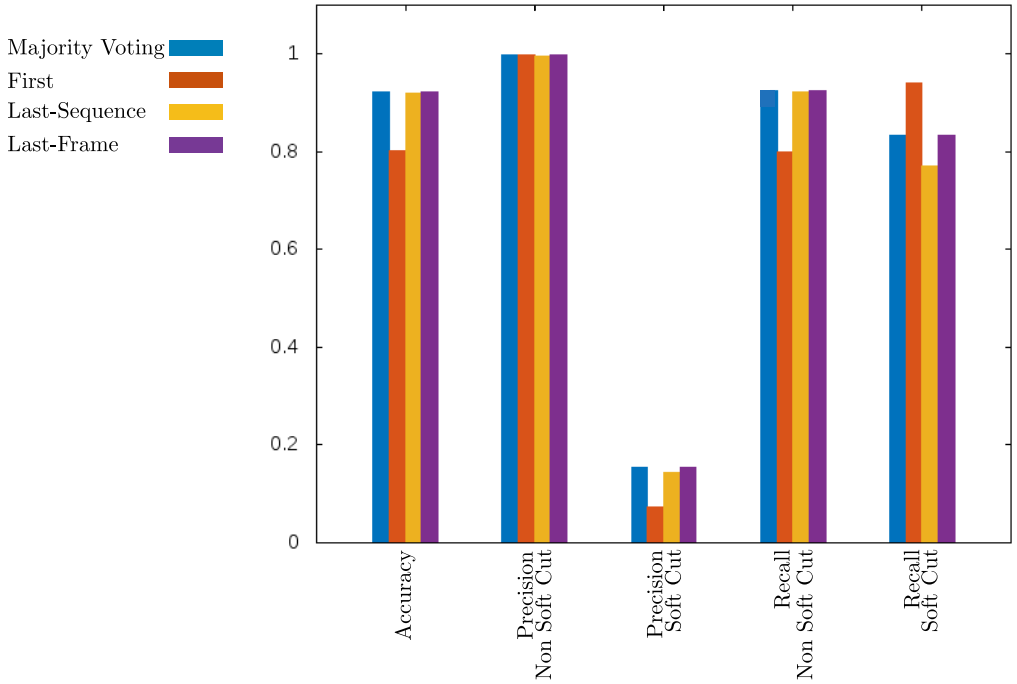

Fig. 9: Results of the different merging strategies: *Majority-Voting Diagonally* (Majority Voting), *Take First* (First), *Take Last 'Sequence'* (Last-Sequence), *Take Last 'Frame'* (Last-Frame). The *Majority-Voting Diagonally* strategy performs best.

Take First strategy is by far the worst. The other three strategies perform equally well, but the *Majority-Voting diagonally* is slightly better with an accuracy of 92.29%. The recall for both classes (soft cut and non soft cut) is also pretty high: 92.43% for non soft cuts and 83.46% for soft cuts. However, the precision of non soft cuts is almost 100% for all four strategies, whereas the precision for soft cuts is only around 15%. This means, that our deep neural network did not learn to recognize soft cuts well. Only 15% of the predicted frames were soft cuts. Basically, the deep neural network recognizes almost every frame as non soft cut.

Because we have such a low precision for soft cuts, we also tried to increase the performance by trying out the *Gap Filler*, concretely the precision-oriented version. Unfortunately, this had no impact at all. The precision of soft cuts did not increase. Thus, it seems that the deep neural network does not predict long-enough sequences of soft cuts for the *Gap Filler* to help.

We first tested with a sequence length of 11: Using a sliding window approach to

detect soft cuts, we repeatedly tested soft cuts of length 11. Having such a low value, allows us to detect parts of a soft cuts, which can afterwards be merged. Still, we also tested with sequences of length 21, as this is the average length of the soft cuts in the original data. Unfortunately, the results were even worse than before.

5 Conclusion

In the end, we summarize our findings concerning both hard cut and soft cut detection.

5.1 Hard Cut Detection

As we see in Table 1, the low precision of our approach is a problem. We have discussed the main reason for that in Section 3.2. Our approach just takes two frames into account and therefore cannot notice, whether the transition under consideration is part of a noisy cluster or a single peak (see Figure 3). One idea to overcome this "blindness" is to use some surrounding histogram differences as additional features. The classifier is trained with a sliding window of histogram differences, resulting in specific patterns. The classification task would then be a pattern matching task, where hard cuts are characterized by a single peak, soft cuts by a gradual curve, and noise by random peaks. Future work could also try to employ a neural net for this kind of pattern matching. A different approach is to do post-processing, where certain highly unlikely events are filtered out. Looking at the false positives in Figure 3 (red bars), one can see, that many of them occur directly after each other. Those cut patterns are not realistic. They might either be noise or a soft cut.

Furthermore, our approach is not applicable for low quality black and white videos (such as the *BG_11362* video of the TrecVid 2007 data set). The problem is, that there are strong brightness changes from one frame to another within one scene. Therefore, we can hardly distinguish between cuts and non cuts. An idea to solve this problem is to calculate the brightness difference between the raw pixel values. If the resulting difference image has homogeneous color, the brightness difference between the two frames is the result of poor image quality.

5.2 Soft Cut Detection

Unlike traditional approaches, we tried to detect soft cuts in a video using a deep neural network. As shown in Section 4.3, we could only achieve a precision of around 15% for soft cuts. We think that there are various reasons for that: First of all it could still be a lack of insufficient data. Also the fixed frame sequence length could be a problem. Training a neural network only with soft cuts of a specific length means that it expects these lengths also on the test set. However, in our sliding window approach only a part of a long soft cut is processed at a time. Maybe the network will not recognize

such parts, because the changes are much less and only one of the frame sequences is in the foreground. The higher accuracy when evaluating on generated test data seems to support this hypothesis. This could be evaluated by training multiple models each with a different fixed sequence length. For predicting a soft cut of a specific length the corresponding model could be used.

But we think that the main problem is that the convolutional network cannot really learn good features, because there is nothing characteristic about a single soft cut frame. In the case of image classification, the CNN can learn concrete features, e.g., to detect a cat. This may include specific colors or edges, and then more abstract features like feet and ears in later layers. For soft cut detection, no such concrete features can be learned. Rather, the network would need to recognize two images, which are blended into each other, which would probably also be hard for a human by looking only at one frame.

In general, we still believe that the soft cut detection could be solved with deep learning, because these techniques have proven their performance in the areas of image and video processing. However, the basic approach with using a recurrent neural network and merging strategies does not work. More work and innovation is required in this area.

References