Universitat Politècnica de Catalunya
Universitat de Barcelona
Universitat Rovira i Virgili

Master in Artificial Intelligence

Computational vision

Video segmentation

Authors:
Alejandro Suárez Hernández
Johannes Heidecke

January 2017

Contents

List of Figures

1 Background subtraction

We have implemented an algorithm to segment a video into different shots. It bases its functionality on the absolute difference between histograms of frames. We are dealing with color images and we do not want to use the color information to help in the detection of shot transitions. Therefore for each image we concatenate the histograms of the three channels into a single vector. For each image we sum the absolute differences of its histogram vector and the former one. Our algorithm detects a shot transition for each of the peaks in the differences progression. In order to avoid spurs we impose a minimum height for a peak in order to be selected. We have chosen empirically a height of 200000. We also impose a minimum distance of 10 frames between consecutive peaks. Figure 1 shows the evolution of the histogram difference between each frame and the former one. We have also marked the peaks that have been selected by our algorithm as shot transitions.

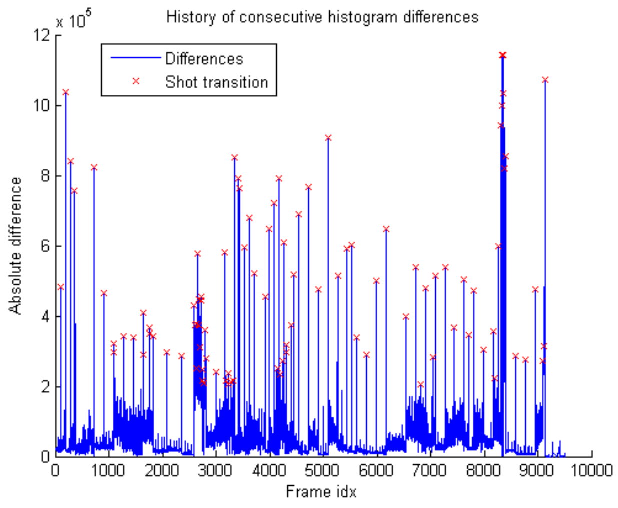


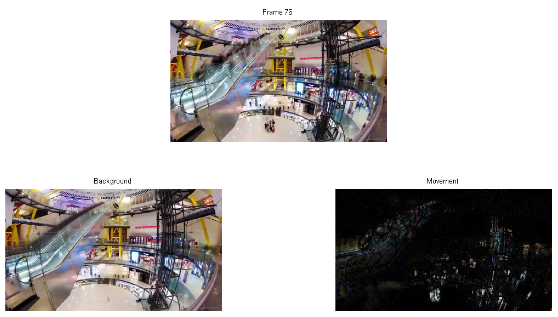
Figure 1: History of absolute differences between consecutive histograms

After the shot transitions have been identified we try to infer the background of those shots in which the camera is static. To do so we calculate the component-wise median of the color for each pixel in the shot. Then this background is used to extract the moving objects of the scene, this is, the foreground. We can see in figure 2 some examples for different shots.

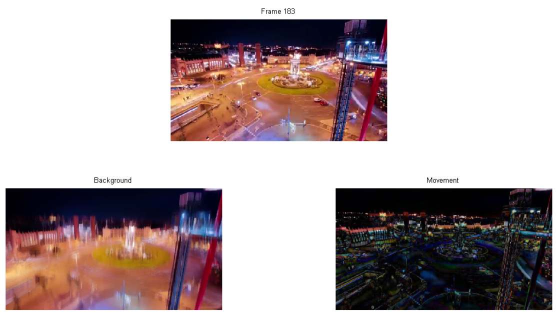
Notice that in the shot 2a the camera is completely static and the background extraction yields pretty good results. However, in the shots 2b and 2d there is some blurring in the extracted background due to the fact that the camera is panning and zooming out, respectively. The case of 2c is special. While the actual shot is longer and there is a slow zoom out there has been a bit of oversegmentation due to the sudden appearance of the bus. In balance the inferred background is quite sharp thanks to the short duration of the extracted shot.

We believe that it is preferable to have some oversegmentation rather than different shots recognized as a single one. Therefore we conclude that the implemented algorithm has good applicability to video segmentation whenever sudden transitions are used. On the other hand the background extraction strategy can have its applications for surveillance or in the film industry (the famous "green" backgrounds).

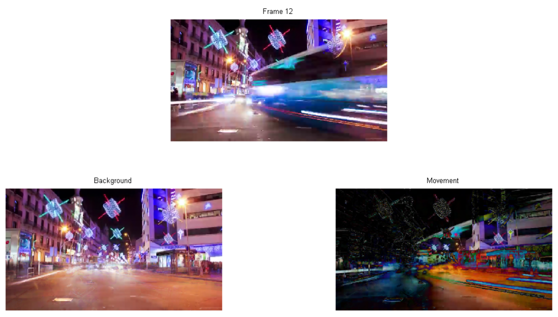
The code used for this section is available in the `video_segmentation.m` and the `play_shot.m` functions. In order to learn more, we refer the reader to the annex and to the source itself.



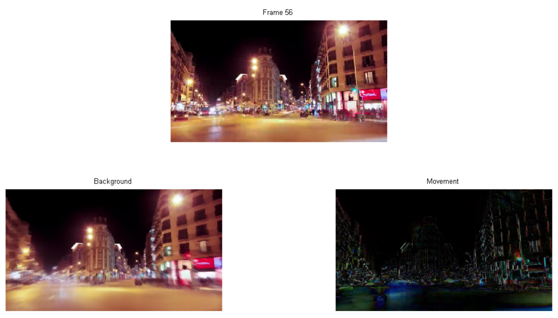
(a) Shot from 0:29 to 0:35



(b) Shot from 0:35 to 0:39



(c) Shot from 0:39 to 0:43



(d) Shot from 0:50 to 0:58

Figure 2: Examples of background extraction for different shots

2 Video segmentation for event extraction

2.1 SVM based approach

Our first approach is training a SVM linear classifier. We believe that the distributions of colors in the image provides a great deal of information about whether the image corresponds to an indoor or outdoor scene, so we have use a sampled down version of the histogram as the SVM's input. More specifically we have obtained for each image all the channels' histograms with only 5 bins. This means that for each image we have a total of 15 features. Also, in order to train the SVM we had to tag the images manually (see annex for more details) [1].

We have taken a random sample of 100 images for training and the 117 remaining ones were left for testing. After training our classifier and testing it against the test images we have obtained an accuracy of about an 83.76% which is actually a quite good result. For completeness, we show the confusion matrix in table 1. As we can see, the problem is not unbalanced and the classifier is not particularly biased towards predicting *Indoor* or *Outdoor*.

Figure 3 shows some examples of correct and bad guesses. We believe that what causes the classifier to miss in figure 3c is the intense glare at the top left corner, while the reason behind the misclassification in 3d is that the sky is obstructed by a tree which is much darker.

The relevant code for this strategy is located at `tag_images.m`, `svm_train_and_test.m` and `demo_svm.m`. The files `GroundTruth.mat`, `Features-s42.mat` and `SVMModel.mat` contain precached variables. See the annex for more details.

Guess\Truth	**Indoor**	**Outdoor**
Indoor	52	13
Outdoor	6	46

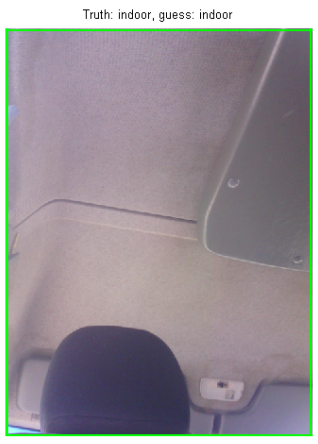
Table 1: Confusion matrix of the SVM classifier

2.2 Texture Based Approach

To compare with the results, we also implemented a classification approach based on texture features from the LM filter bank as in lab 9. For all training images, we use standard deviation to aggregate the filter responses of all images and combine them with mean values of the color channels. To classify a new image, we construct its feature vector and pass it to a knn-search algorithm that returns the k nearest neighbors of the image in the feature space. The algorith then looks at the labels of these neighbors and decides based on majority, if the image should be classified as indoor or outdoor. Besides choosing the right features to represent the textures (we opted for taking the best results from lab 9), the parameter k has to be tuned. Our experiment shows in figure 4 that a relatively high k of 81 yields the best accuracy of $59,83\%$. The corresponding confusion matrix is depicted in 2.

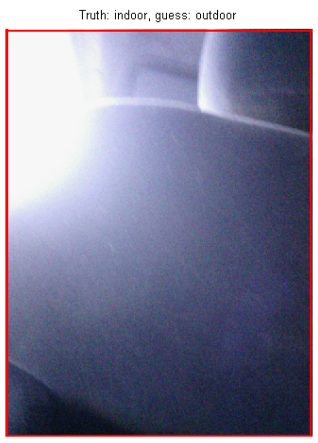
The obtained best accuracy is below the one using the SVM.

[1]Suggestion: for next years provide the students with the correct tags for each image, so their approaches (even if they are not based on a Machine Learning technique) can be tested against a decent number of images and not just a few ones. We offer our tags if they are of any value (they are stored in the `GroundTruth.mat` file).

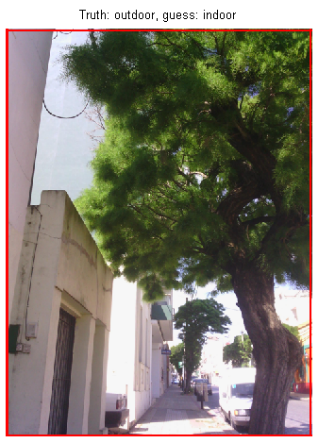


(a) Indoor scene correctly recognized

(b) Outdoor scene correctly recognized



(c) Failure recognizing indoor scene



(d) Failure recognizing outdoor scene

Figure 3: Some examples of correct and incorrect guesses with our SVM classifier

Guess\Truth	Indoor	Outdoor
Indoor	32	21
Outdoor	26	38

Table 2: Confusion matrix of the texture classifier

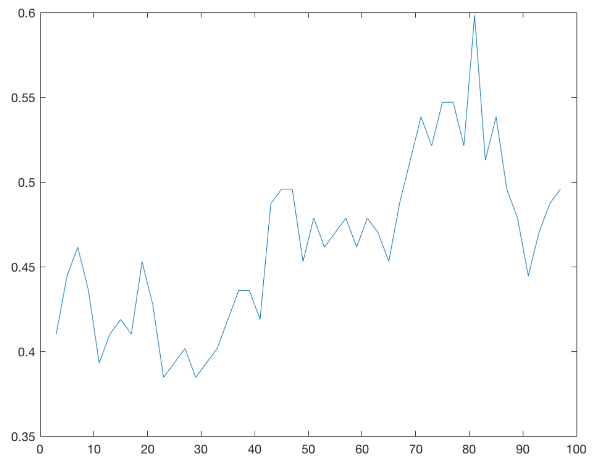


Figure 4: Accuracy values for different choices of k

Appendices

A Source code

Along with this report we provide the following source code:

- `play_shot.m`: a function that, given a 4D matrix of frames (height, width, channels and time), a background and a frame rate, plays the video fraction showing at the same time the background and the foreground (absolute difference between the current frame and the background).

- `video_segmentation.m`: performs all the steps of the first exercise. This is, it computes the differences of histograms between consecutive frame to infer at which points of the video there is a shot transitions; and then it extracts the background and foreground from each shot. It makes use of the `play_shot.m` method described above to play the video and give graphical feedback. NOTE: the computation of the shot transitions can take a while if the video has to be read from beginning to end to compute the histogram differences. For this reason we provide the already cached history of differences at `history_diff.mat`. Of course, the recomputation of the history of differences can be forced (check the comments in the code).

- `tag_images.m`: an utility function that iterates over all the images inside `materialLab10`, shows them to the user and ask them by means of a dialog window whether it is an indoor scene (1) or not (0). Afterward the tags are stored in a file called `GroundTruth.m`

- `svm_train_and_test.m`: loads the name and the tags from all the images from `GroundTruth.m`. Then it calculates the features of each image and stores them in a matrix `X`. Since this step can take a while, the matrix of features is stored in `Features-s42.m`. After this it splits the data into a training set and a test set. It uses the training set to build a linear SVM classifier. The classifier is used upon the test data and the accuracy and confusion matrix of the classifier are inferred from the results. The SVM model is stored in `SVMModel.m` so it can be later used in `demo_svm.m`.

- `texture_train_and_test.m`: loads the name and the tags from all the images from `GroundTruth.m`. Then it calculates the texture features of each image and stores them in a matrix `Rs`. Since this step can take a while, the matrix of features is stored in `Textures-s42.mat`. After this it splits the data into a training set and a test set. The classification is done based on a KNN approach: obtaining the class of the k nearest neighbors and then voting which class has a majority. The code automatically calculates the validation set accuracy for $k \in \{x | x = 2k + 1, k = \{1, 2, \dots, 48\}\}$.

- `demo_svm.m`: it uses the trained SVM model to show and classify all the images from the `materialLab10` folder, giving visual feedback to the user as a colored frame around the image (green if the guess is correct, red if it is not).

As always, we suggest the reader to read our code if they want be aware of all the details.