Report Computer Vision Project 3

Single Michael
08-917-445

1 Video search with bags of visual words

For generating the data sets for this project I used two *mp4* movies. The first is a sequence of a *Breaking Bad* episode (called *breakingbad2*) and the second is a demo video found used for various purposes showing some kids playing with their toys (called *test_video*). These video files are stored in the folder *video/*.

I extracted from each movie their frames and stored them as png images in *frames/* using my Matlab function *extractVideoFrames*. For each frame converted to a grayscale image, I computed its corresponding SIFT data using VLFeat's function *vl_sift*. Hence, for each extracted frame I create a *.mat* file that contains

- the SIFT vectors (encoded in the rows)
- the name of the frame.
- the number of detected features
- the orientation of the patches
- the positions of the patch center.
- the scales of the patches.

You can generate your own mat files using my function *computeSiftDataOf*.

Last, I additionally assembled all mat files into one big mat file, which contains all mat files in a sequential order. This is just to simplify the coding that had to be done for the assignment. You can find all those mat files in the folder *data/*.

1.1 Raw Descriptor Matching

Note: run the script *rawDescriptorMatches* in order to reproduce my results.

First we load the precomputed SIFT data of two chosen images (called left and right) from our data sets. Then we let a user select a region (relying on the provided script *selectRegion*) in the left image. Each point within the user-selection is then considered as a feature point in the left image.

Next we try to match these left-image feature points to corresponding, closest feature points in the right image. For the matching process, we calculate the distance (2-norm) between the descriptors of the left features with all possible descriptors from the left image (Remember: those descriptors are stored in the corresponding mat files). In practise we compute the distance between all possible descriptor pair combinations where the first points is from the left descriptor and the second point is from the right descriptor (using the provided *dist2* function) and store those distances in a $M \times N$ matrix D. Note that the m-th row in D corresponds to the m-th element in the left descriptor and the n-th colum in D corresponds to the n-th element in the right descriptor.

For each row, we look for the minimum element, i.e. the minimum distance. This corresponds to finding the column which has the smallest distance value for a fixed row. Since a column index corresponds to the same index in the right descriptor, this retrieved column index corresponds to the best matching feature point index in the right descriptor matching to a given descriptor point in the left image (i.e. for a given row index).

Since we only are interested in good matching feature point pairs, we introduce an additional constraint. Each determined minimum is taken if and only if it is still the minimum distance in its row, when being scaled by a certain factor that depicts a acceptance parameter. This factor has to be bigger than one. This ensures, that we only take strong and reliable matchings. This also implies that

not every descriptor point in the right image has a corresponding point in the left image.

In practise we first look for the minimum distance index in each row. We store these minima and their indices as candidates in an array and then set their values in D equal to infinite. We than again look for the minimum value in this updated D matrix. This will give us the second smallest distance value column-index for each row. We then compare the initial candidate minima scaled by a user specified acceptance value with the second smallest retrieved minima values. If the scaled candidates are still smaller than the second smallest minima (unscaled), then we have found a true minimum otherwise we discard this candidate.

In my code I set this scale equal to two, i.e. if twice the minimum is still smaller than the second smallest row distance, then we have found a true minimum (index), i.e. we have found a matching. So every index pair (row column) of such a true minimum is a matching pair, where the first element corresponds to the left and the second element to the right feature point.

For each feature points we look up its SIFT data, i.e. their position, scale and orientation. We pass those values to the provided function *displaySIFTPatches* and hereby show the corresponding matches in the left and right images (resulting by the given user-selection).

In the following I show my results I got using my implementation. My first example shows the results of the *twoFrameData* data-set that acts as a test case for my implementation. Similarly as shown in the exercise sheet I selected the upper door of the fridge and got similar results shown in FIGURE. Figure 3 shows the feature points in the left image and figure 2 shows the corresponding matching points in the right image.

I applied this feature matching the same way to my own data-sets.

- Breaking Bad data set: White tubus selection shown in figure 3 and the matching shown in figure 4.
- Kids data set: Selection of boy's t-shirt shown in figure 5 and their matchings shown in figure 8.

- Kids data set: Selection of collection of toys shown in figure 7 and the corresponding matchings shown in figure 8.

In summary, the matching produced in my implementation seems to work.

1.2 Visualizing The Vocabulary

Note: run the script *visualizeVocabulary* in order to reproduce my results.

The goal of this task was to visualize patches associated with two of the visual words, i.e. we are interested in showing the visual vocabulary. This can be achieved by clustering all feature points of all descriptors of a given data-set. For this purpose I use the assembled mat file that has been generated after the generation of the SIFT data of each extracted video frame. This file contains all SIFT data of every frame in a sequential order that belongs to the same data-set.

After loading all the SIFT data of every frame, especially all the descriptors (containing all the feature points), I pass that huge set of descriptors to the provided method *kmeansML*. Basically, this method separates all the descriptors (feature points) into k (this is a user specified constant, for this assignment k is equal to 1500) clusters. Each cluster represents a so called word. In the clustering process, each feature points was classified to a particular word. Thus, when selecting a random word, we can retrieve the corresponding feature points that belong to that cluster. From this feature point set, we select 25 of them. From those feature points, we select their corresponding patches from the image using the provided Matlab function *getPatchFromSIFTParameters* and visualize all these patches.

In the following I show for each of my data-sets two word visualizations:

- Breaking Bad data-set: First visual word example shown in figure 9 and second example shown in figure 10. The first word clusters features that are associated to a patch showing some kind of cabinet with a tubus and the second word aggregates patches that also look very similar (tubus shaped geometry on the right top corner or a bar-like shape on the top right center).

- kids data-set: First visual word example shown in figure 11 and second example shown in figure 12. The first word aggregates patches that belong to features that look like a arm and the is the word that aggregates features that have a patch showing the head of the man playing with the kids (rotated by 180 degree).

We observe that all feature point patches that belong to the same word look similar to each other. The exhibit the same kind of visual appearance in their detail. This holds true for both data sets.

1.3 Full Frame Queries

Note: run the script *fullFrameQueries* in order to reproduce my results.

In order to solve this task we again first compute bag of words by clustering all feature points in the set of all descriptors of a data-set of choice. Similarly, as in the previous task we initially load all assembled SIFT data of the frames belonging to target data-set. We then perform the clustering of the descriptor features as we did before.

Next, we compute a histogram of all frames, indicating how many words belong to a particular cluster. I.e. we count how many features are classified to a certain cluster. For this purpose, we technically rely on the descriptors of frame and apply Matlab's *histc* function.

Next we choose a random frame and compare its histogram to the histograms of all other frames. For comparing the selected histogram by any other histogram (i.e. applying a measure), we compute the normalized scalar product between them. This gives us a number between zero and one. The closer the result, the more identical the two histograms are. This implies that two histograms are the same, if their normalized scalar product is equal to one and they are completely different in case their product is equal zero. We denote their similarity value by the variable *score*.

Last, we want to display the 5 most similar frames. By sorting all the similarity scores in a descending order, we simply have to pick those frames that belong

to the first five scores in the sorted score set.

In my results, the first image is the random selected frame, and all other frames represent depict the frames with the highest scores matching this random frame.

In the following I demonstrate the results when applying the *fullRegionQuery* script to my two data-sets:

- breaking bad data-set: Example one is shown in figure 13 and example two is shown in figure 14.
- kids data-set: Example one is shown in figure 15 and example two is shown in figure 16.

Observation: Frames that are close in time to the selected frame achieve a high score value. In other words, frames that are close in a temporal sequence to the reference frame (i.e. their temporal distance is small) achieve usually a higher score value than frames in later times. This makes sense, since one movie second consists of many frames (approx. 24 frames per second). Within one second, usually, for ordinary sceneries there is not a huge spatial variation between a frame and its successor frame. Therefore, a successor frame is usually very similar to its predecessor frame and thus achieves a large similarity value. However, there are some exceptions, if there suddenly occurs some kind of occlusion for example. then a successor frame will not be close to its parent frame.

1.4 Region Queries

Note: run the script *regionQueries* in order to reproduce my results.

Fig. 1: User selection in left image of provided twoFramesData data set.

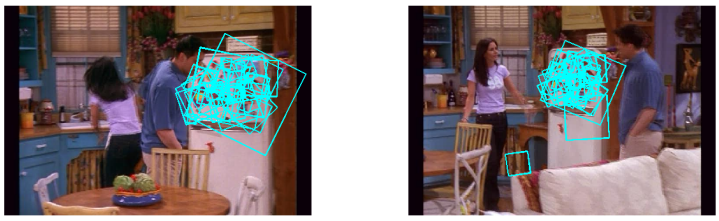

Fig. 2: On the left feature points in the user specified selection region and on the right the their matched feature points.

Fig. 3: User selection in left image of provided twoFramesData data set.

Fig. 4: On the left feature points in the user specified selection region and on the right the their matched feature points.

Fig. 5: User selection in left image of provided twoFramesData data set.

Fig. 6: On the left feature points in the user specified selection region and on the right the their matched feature points.

Fig. 7: User selection in left image of provided twoFramesData data set.

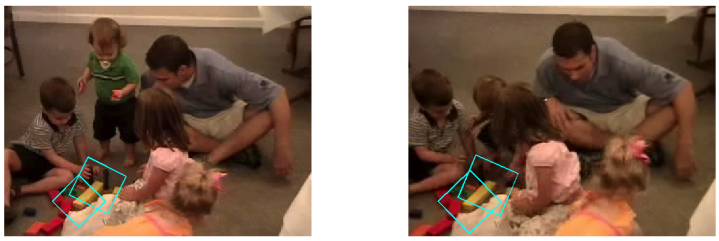

Fig. 8: On the left feature points in the user specified selection region and on the right the their matched feature points.

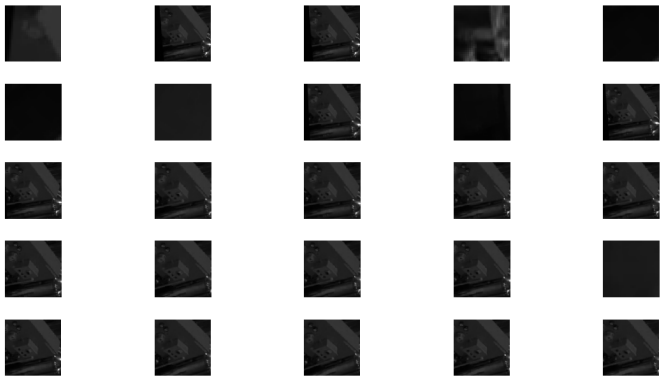

Fig. 9: Breaking Bad data-set: Patches of feature points that belong to the same cluster.

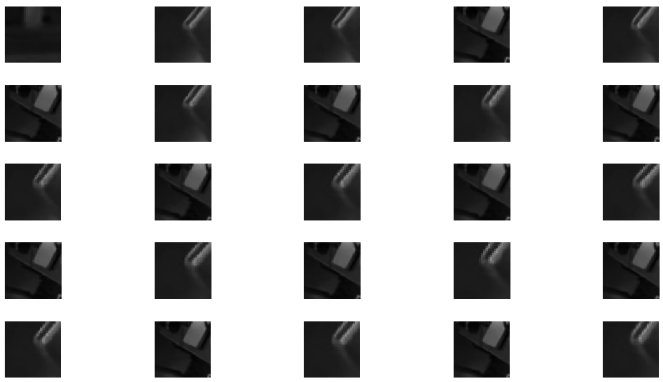

Fig. 10: Breaking Bad data-set: Patches of feature points that belong to the same cluster.

Fig. 11: Kids data-set: Patches of feature points that belong to the same cluster.

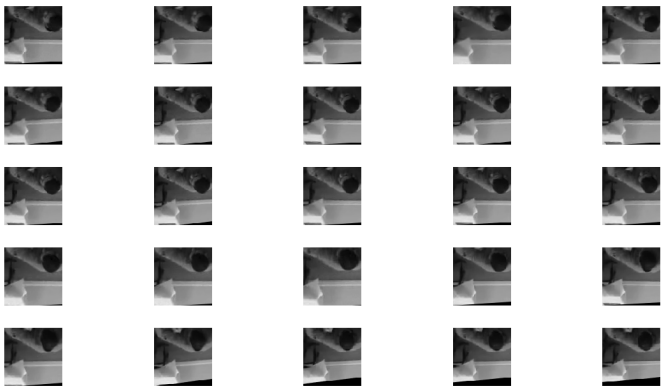

Fig. 12: Kids data-set: Patches of feature points that belong to the same cluster.

Fig. 13: Full Frame Query matchings when selecting frame 8 when using the breaking bad data-set.

Fig. 14: Full Frame Query matchings when selecting frame 132 when using the breaking bad data-set.

Fig. 15: Full Frame Query matchings when selecting frame 157 when using the kids data-set.

Fig. 16: Full Frame Query matchings when selecting frame 206 when using the kids data-set.