

CS 6476: Computer Vision, Fall 2019

PS4

Submitted by: Disha Das

GTID: 903542819

Date: 11/07/2019

1 Short answer problems

1. When performing interest point detection with the Laplacian of Gaussian, how would the results differ if we were to (a) take any positions that are local maxima in scale-space, or (b) take any positions whose filter response exceeds a threshold? Specifically, what is the impact on repeatability or distinctiveness of the resulting interest points?

Ans. To perform interest point detection with LoG, the image is convolved with scale-normalized LoG filters. When any position that is local maxima in scale-space is taken, we find interest points that show local-maxima in any one scale only. Thus, repeatability may suffer. An interest point is “repeated”, if the point detected in the first image is also accurately detected in the second one. Since the scale of detected interest point may be different because of different local-maxima in two images, the corresponding points may not be accurately detected.

On the other hand, when positions are taken whose filter response exceeds a threshold, the same interest point may be detected over several scales, increasing repeatability, i.e. increasing the likelihood of the same point being detected in other images.

2. What is an “inlier” when using RANSAC to solve for the epipolar lines for stereo with uncalibrated views, and how do we compute those inliers?

Ans. Using RANSAC, one can find the Fundamental matrix using the following algorithm:

We are given a set of n corresponding points (a_i, b_i) , i in $[1, n]$ where a_i belongs to image a and b_i belongs to image b . Threshold $t > 0$. Number of trials: T .

Iterate T times:

1. Randomly select 8 pairs from the set (a_i, b_i) .
2. Use these pairs to solve for fundamental matrix F .
3. Compute perpendicular distances $d_1(b_m, \text{epipolar line of } a_m)$ and $d_2(a_m, \text{epipolar line of } b_m)$ for each pair.
4. Detect inliers. A pair (a_m, b_m) qualifies to be an inlier if $d_1 > t$ and $d_2 > t$
5. If the number of inliers is the largest so far, store the F matrix and the inlier set.

End loop.

Then, solve for F using all the points in inlier set.

Thus, an ‘inlier’ is a pair of corresponding points in two images such that

1. The perpendicular distance between a_i and epipolar line through b_i lies within a threshold t .
2. The perpendicular distance between b_i and epipolar line through a_i lies within a threshold t .

3. Name and briefly explain two possible failure modes for dense stereo matching, where points are matched using local appearance and correlation search within a window.

Ans. Small window size: Small window methods of local stereo matching can detect disparity in regions that are highly texturized, but produce noisy disparities in regions with little texture. It may amplify depth noise and invalidate the smoothing effect.

Big window size: Local stereo matching using big window sizes can produce smooth disparities in textureless regions, but for highly texturized regions, it is difficult to get accurate disparities. It may result in error matching in some fine structures and discontinuous regions.

4. What exactly does the value recorded in a single dimension of a SIFT keypoint descriptor signify?

Ans. To generate a SIFT descriptor for a particular keypoint, a 16×16 neighborhood is taken around the keypoint. This is divided into 4×4 sub-blocks which are 16 in number. For each of these sub-blocks, 8 bin orientation histograms are created. Hence, for every descriptor, $16 \times 8 = 128$ bin values are created. This is represented as a vector to form a keypoint descriptor. Thus, the value in a single dimension of SIFT vector is a histogram bin value that contains samples from a 4×4 sub-block of the original 16×16 neighborhood.

5. If using SIFT with the Generalized Hough Transform to perform recognition of an object instance, what is the dimensionality of the Hough parameter space? Explain your answer.

Ans. The dimensionality of the Hough parameter space is 4 when using SIFT with GHT to perform object recognition. The dimensions are x-coordinate, y-coordinate, scale and orientation of the detected SIFT feature. Voting is done on such a 4 dimensional Hough space and the values of 4D bins are incremented everytime the above 4 parameters match between a detected SIFT feature and a model feature. The bins with highest votes above a threshold are selected. These bins correspond to detected SIFT features that have the best matches with model features.

2 Programming

Note: The scripts assume that the frames and their sift files are located in the following folders:

Parent Folder/PS4Frames/frames/

Parent Folder/PS4SIFT/sift/

1 Raw Descriptor Matches:

This script detects features in the 2nd frame that are most similar to the features in a selected region of the 1st frame. In this script, data of two images is first loaded from 'twoFrameData.mat'. Using the provided function 'selectRegion()', we select a region and extract the row values of the descriptors that lie within the selected region. Using the given function 'dist2()', the Euclidian distances between the descriptors of image 1 lying inside the selected region and all the descriptors of image 2 are computed. This is the 'z' matrix in the script. This matrix is normalized. Since similar descriptors will have a relatively less distance value, we put a threshold of '0.1' and search for descriptor pairs that have values less than this threshold. When such a pair is found, the row value of descriptor in image 2 and their distance value is stored in a matrix 'rows'. After the iteration, we sort 'rows' according to distance value in the ascending order. If required, we could only extract the top n closest values from this matrix. And now, having found the row values of descriptors in image 2 that are the closest in distance from the descriptors in image 1, we feed those row values to given function 'displaySIFTPatches()' to find the features in image 2 that are similar to features in selected region of image 1.

We try to find similar objects or patterns by finding the descriptors that have the least distances between them. In the example given below, a polygon is drawn around the fish magnet on the fridge. This fish magnet has been successfully detected in the second image.

Polygon to select region in 1st image

Draw a polygon around region of interest



Region detected in 2nd image

Matched features in Image 2



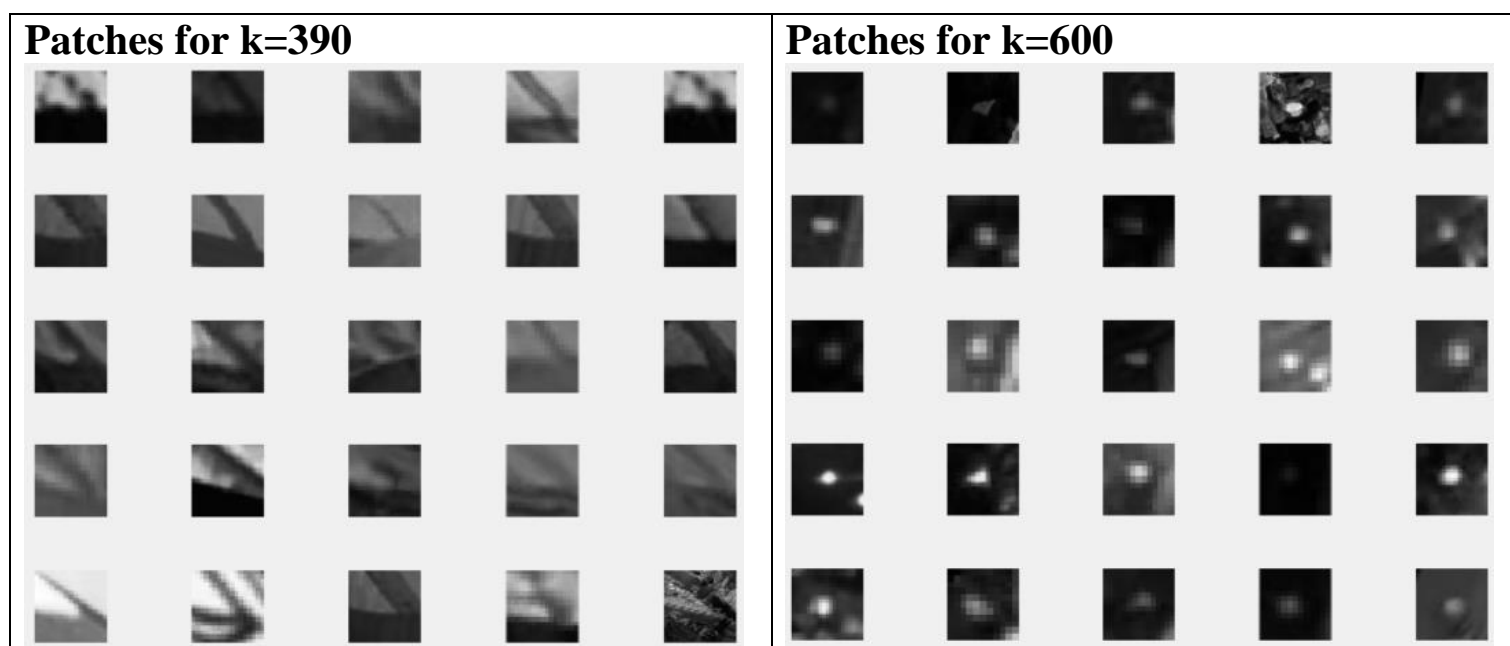
2. VisualVocabulary

This script finds patches that have been labelled with a particular visual word. Before the creation of this script, 500 frames were selected randomly and their descriptors, positions, scales etc were stored in a matrix 'representative1.mat'. Then, 50% of descriptors were selected randomly from this list of descriptors and were used to compute k-means clustering with k value 1500. The cluster centre matrix (size 1500x128) and the labels of selected descriptors were stored in 'visualVocabulary.mat' file. These two files are loaded in the script.

First, two visual words are selected, $m=390$ and $n=600$, out of the 1500 visual words. For each word, distance matrix is calculated between the cluster centres of word m (size 1×128) and all the descriptors of the 500 frames that we had selected earlier. This matrix, z_m , is normalized. This matrix contains the distances between every descriptor in those 500 images and the centre of cluster m . We sort the values in z_m in ascending order and extract the top 25 descriptors that are closest to the cluster centre of word m . The same is repeated for the cluster centre n .

To display these descriptors, we need to find the images they belonged to. 'featList' is a 500×2 matrix that contains the starting descriptor and ending descriptor values of each image. Using this, we find the image that the descriptors in 'rowsm' belong to and this image is used to extract patch using 'getPatchFromSIFTParameters()'. 25 such patches are extracted for each word and displayed.

In the example below, word m gives patches that have a similar pattern in them. Word n seems to describe patches which have a lighted centre in the middle.









3. fullFrameQueries

This script finds frames that are most similar to a given frame based on the similarity scores of their respective bag-of-words histograms. In order to implement the above script, two helper functions were created- 'bagOfWords.m' and 'similarityScore.m'. 'bagOfWords()' takes the name of an image as input and returns the bag-of-words histogram for that particular image. To save computation time, the histograms of all the frames were computed and stored in the file 'histograms.mat'. 'similarityScore()' takes two histograms as input and returns their similarity value.

3 frames have been selected. Their histograms are computed. Rows1, rows2 and rows3 store the similarity scores between selected frames and all the other frames. These matrices are sorted in descending order according to their similarity scores and their top 5 values are extracted. These top 5 values correspond to the top 5 frames that are the most similar to given frame. This is repeated for each of the 3 selected frames.

In the images below, the top left image is the originally selected frame with a similarity score of 1. The others that follow are displayed in rank order of their similarity scores.

<p>friends_000003402.jpeg Similarity: 1.000000e+00</p> 	<p>friends_000003401.jpeg Similarity: 7.652455e-01</p> 
<p>friends_000003400.jpeg Similarity: 7.360265e-01</p> 	<p>friends_000003390.jpeg Similarity: 7.283540e-01</p> 
<p>friends_000003387.jpeg Similarity: 7.264651e-01</p> 	<p>friends_000003388.jpeg Similarity: 7.234788e-01</p> 

friends_000003321.jpeg

Similarity: 1.000000e+00



friends_000003318.jpeg

Similarity: 7.805276e-01



friends_000003322.jpeg

Similarity: 7.791922e-01



friends_000003320.jpeg

Similarity: 7.545768e-01



friends_000003323.jpeg

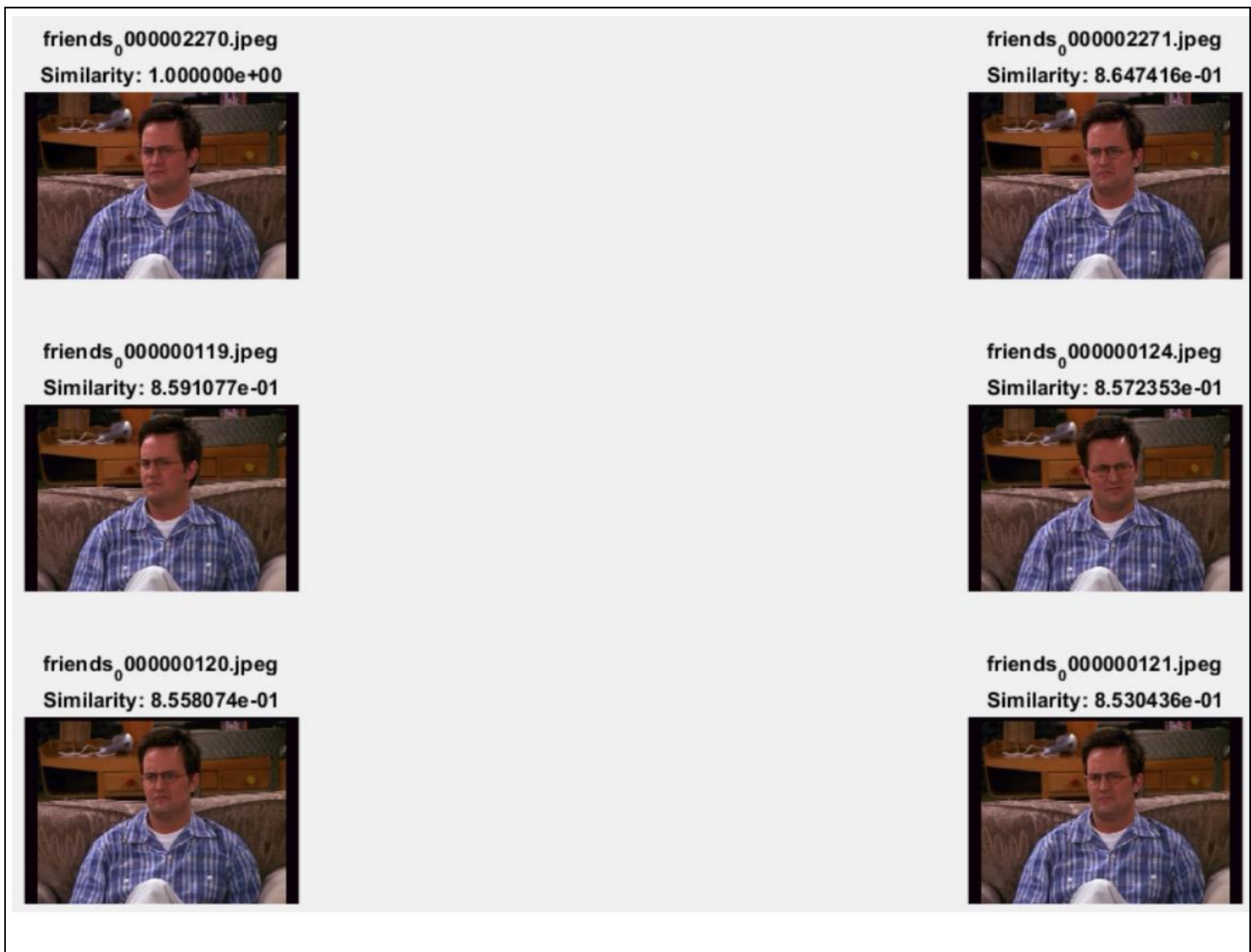
Similarity: 7.441589e-01



friends_000003319.jpeg

Similarity: 7.296498e-01





4. regionQueries

This script detects features in other frames that are most similar to the features in a selected region of a given frame using visual vocabulary. For this script, the x and y coordinates of the polygons of selected regions were first stored in 'oninds.mat' file and loaded into regionQueries.m file. For each given frame, the features belonging to their selected regions are extracted and their histograms are created, i.e, a single histogram for each selected regions- 4 in number.

Distance between the centre values of k words and the descriptors in the selected regions is computed and stored in 'z'. 'z' is normalized. Using min(z) gives the minimum values along each column of z and their indices (stored in 'class'). If a descriptor has the least distance from the centre of a word m, then it is labelled with word m. Thus, the matrix 'class' contains the class labels of each of the features in the selected region. Using this 'class', the bag-of-words histogram for features in selected region is computed and stored in 'h1'.

Now, for each frame in the database, similarity scores are computed between 'h1' and the histograms of all other frames. These are stored in 'rows1' and sorted in descending order according to their similarity scores. Now, for the top 5 frames, the Euclidian distances between features in selected region and features in the top 5 frames is computed. Only the top 25 features are displayed for each of the 5 matching frames. This process is repeated for 3 other frames with distinct selected regions.

Selected region of image 1



In the above example, Chandler's shirt has been detected over two non-consecutive frames. This shows that the above algorithm is great at detecting patterns. The above example is an example of a success case.

Selected region of image 2



In the above example, a lamp is selected as region of interest. However, the detected frames show features with a bright circular centre (lighted windows in buildings, shirt buttons etc), which albeit look similar to our lamp, but are in no way what we intended to detect. The above is an example of a failure case as the object meant to be detected is not being detected because it is similar in appearance to a variety of different patches.

Selected region of image 3



In the above frame, the region of interest is the blinder on the window. We know that this algorithm is great at detecting patterns. The same blinder has been detected in multiple non-consecutive frames as seen above. Not only that, a similar window with blinder has been detected because the blinders share the same patterns. This is an example of a success case as the same object and some similar objects have been detected. This is because the region of interest consisted wholly of a pattern. Hence, patches with similar patterns were detected.

Selected region of image 4



In the above image, Rachel's shirt is being detected over consecutive frames because there are more than 5 number of frames that have high similarity scores to the pattern in selected region of the above image. This is an example of a success case.

3 Optional

3.1 Implementation of tf-idf weighting to histograms and using stop list.

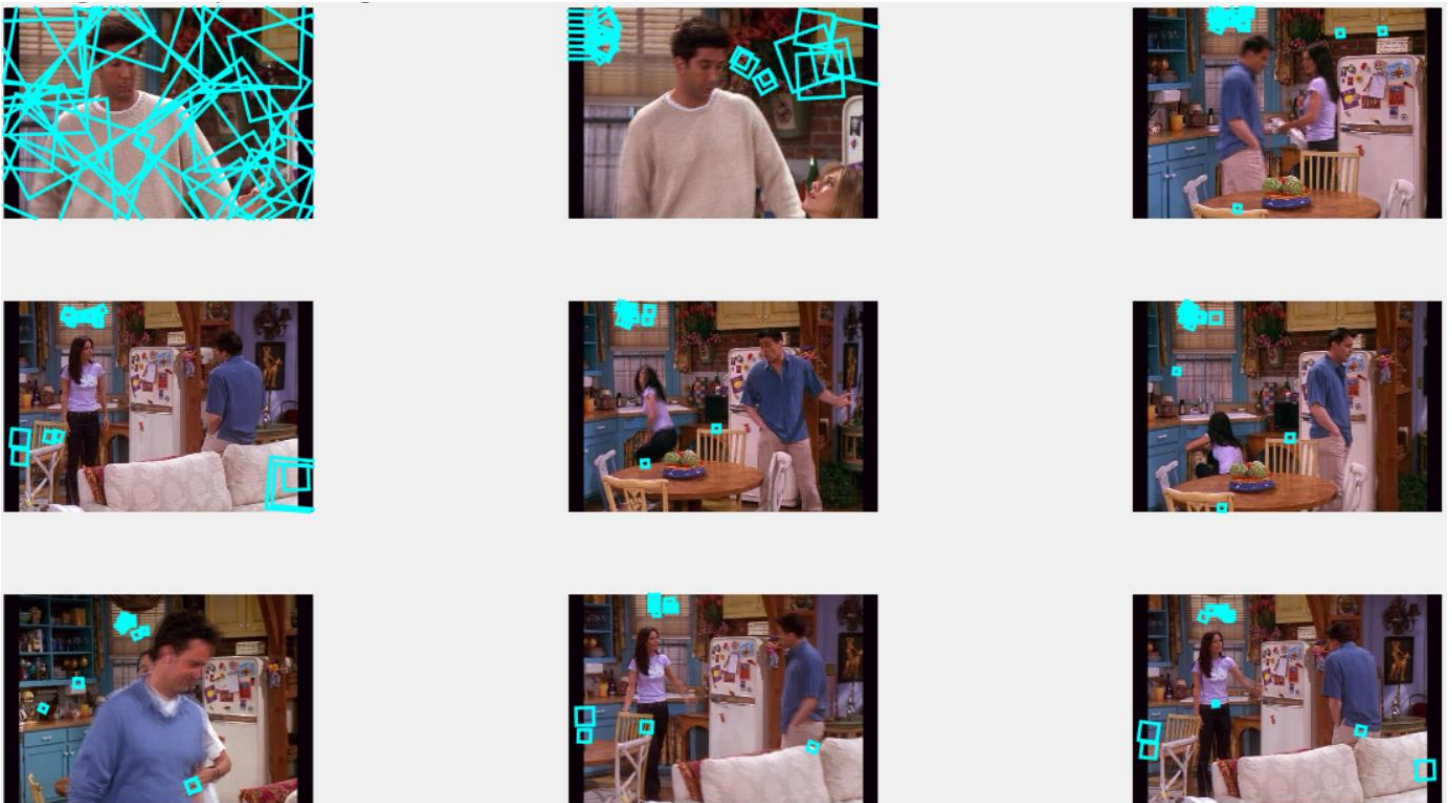
Using the script 'tfidf.m', we created a tf-idf weighted histogram list of all 6671 images and stored them in 'tfidfHistograms.mat' file. In the script 'stopWords.m', the top 200 words with least tf-idf weights were selected and their word index numbers were stored in 'stopList.mat' file. To test the difference between outputs of matching algorithm using tf-idf weighted histogram and normal histogram, we use the script 'TFIDFImplementation.m'.

3 frames are taken. For each of these, tf-idf weighted histogram and normal histogram is computed. Stop list is used on the first histogram only. Then taking each histogram, frames with largest similarity scores are computed in a similar manner to that implemented in 'fullFrameQueries.m' file.

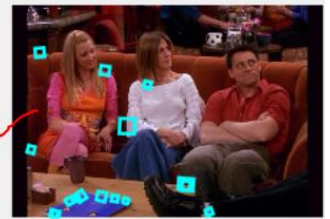
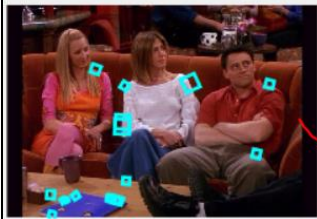
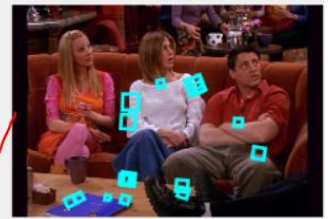
In the frame below, the first set of subplots are found using tf-idf weighted histograms. The top left subplot is the original frame. After ignoring common words, it seems like the key words in the frame are related to the window blinder patterns. These patterns are detected in all the other frames that are detected. In the subplot set that doesn't use tf-idf weighting, we find some detected frames that don't have the same blinder in them, but a similar pattern. Such frames have a red tick mark against them.

This clearly shows that when we use tf-idf weighting, only the most important features in an image are searched and detected in other frames, which is not the case with the other method of frame search.

Tf-IDF with stop words (Frame 1)



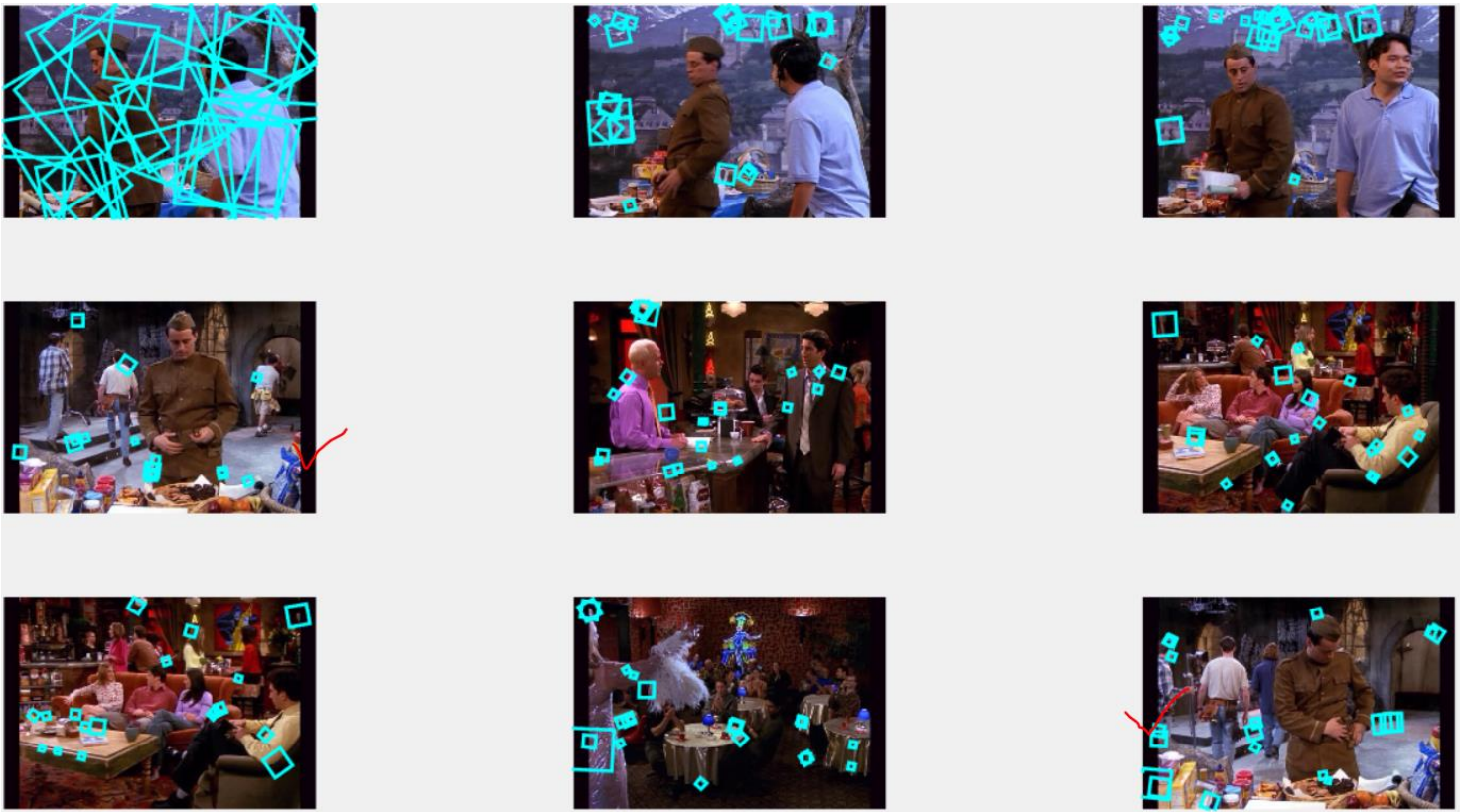
Regular (Frame 1)



Tf-IDF with stop words (Frame 2)



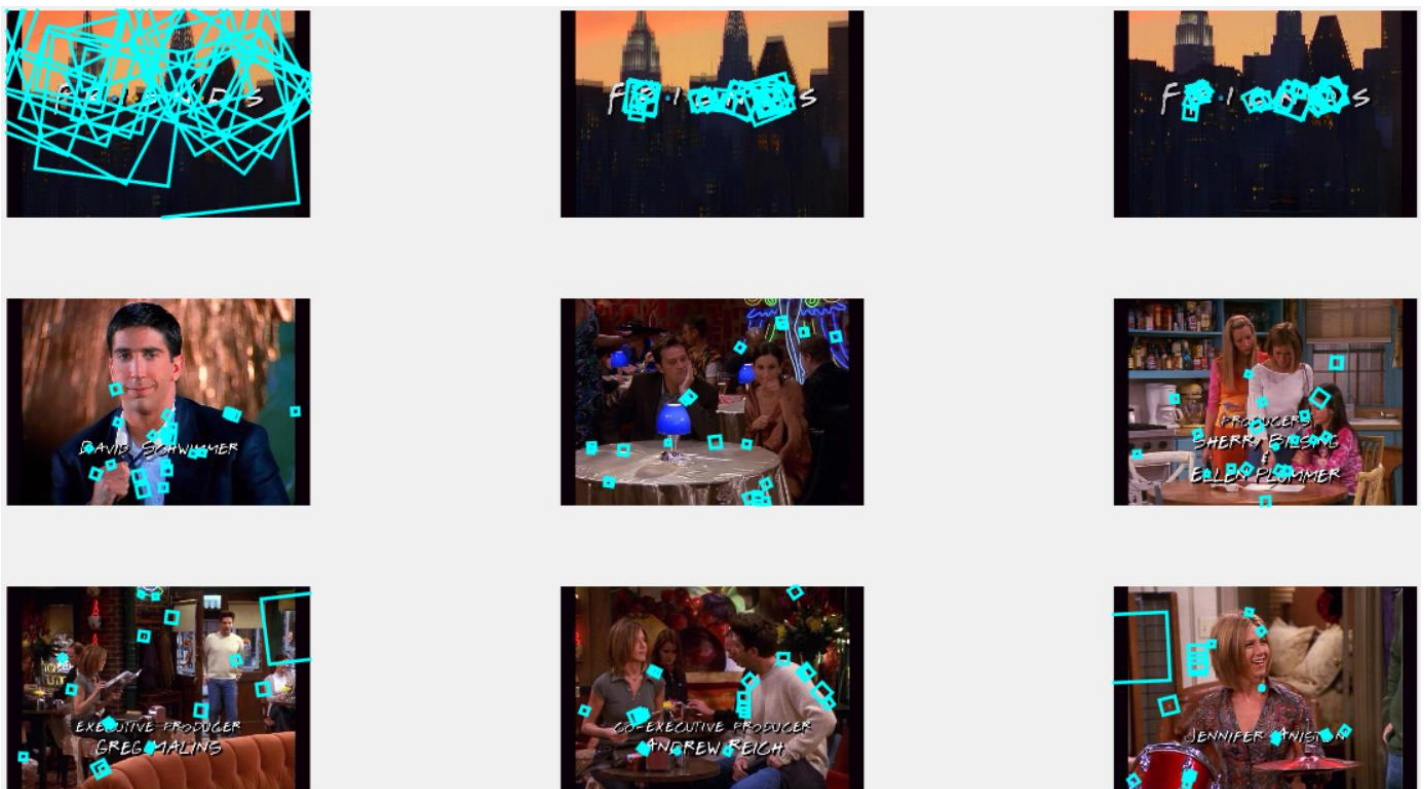
Regular (Frame 2)



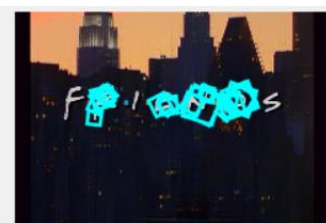
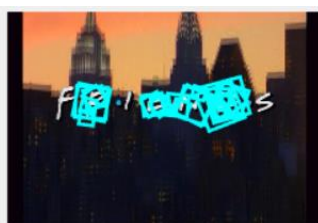
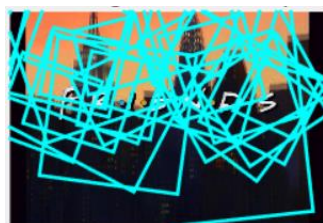
In the 2nd image above, we want to detect frames similar to the frame in the top-left. The tf-idf method chooses the blue patches as the keywords of the frame and thus, these are detected in the following 8 frames. However, without using the tf-idf method, we can see that the frames with red tick against them have Joey in the uniform.

An explanation for this is that while eliminating common words, some important features may get eliminated and we end up with keywords that may not be appropriate to the image. In the below example, Joey in uniform could be a key-feature which got eliminated while implementing our stop-list. Hence using tf-idf method may not be advantageous in all cases, taking cues from the example above.

Tf-IDF with stop words (Frame 3)



Regular (Frame 3)



In the 3rd example, we see that the keywords for frame in the top-left seem to be the features around the title 'F.R.I.E.N.D.S'. Using tf-idf method, the frames that have been detected contain subtitles in the same font as the title. This shows that tf-idf method has been successful in finding only frames that have similar keywords. On the other hand, the regular method doesn't zero-in on the keywords and does a general search, ending up with frames that contain random features. Only the frame with a red tick mark against it contains subtitles. This case shows the advantages of using tf-idf weighted histograms with stop words as opposed to regular bag-of-words histograms.