A Report on Change Detection in Video Streams Using One Class SVM

Submitted by: Alon Brifman

Directed by: Assaf Glazer and Prof. Michael Lindenbaum

***Abstract**: It is common to model background in order to detect anomalies in a video stream given by a stationary camera. In this report we use a one class SVM in a block-based approach in order to model background. We assume no anomalies during first 100 frames and will use them as a training set. We present several feature extraction methods and several variations for estimating the probability that a change occurred, followed by several post processing methods. We also analyze how the number of blocks into which the frame is divided influences the results. During our work we encountered problems such as different distribution of the background scene during train and test, or variant types of background distributions. We elaborate on how we coped with these problems, and finally added experimental results showing how well our algorithm works on several videos.*

1. ***Goal***: Our objective is to use a one class SVM (OCSVM) in order to model background and thus detect anomalies and changes in a video stream. We restrict ourselves to a framework in which we assume a stationary camera and that the first N (we used N=100) frames in the video are classified as "no change" , this will be our training set, and the rest of the video will be our test set. The datasets that we will work on in this report are taken from http://www.changedetection.net/ , especially those that fit to our framework. For example figure 1.1 presents a frame from the train set in the video "office", another frame from the test set and the expected result.

(a) (b) (c)

(d)

Fig. 1.1

(a) – a frame from the train set
(b) - a frame from the test set
(c) - the expected result for (b)
(d) - our final result

A movie explaining our algorithm outline and showing experimental results can be found at http://www.youtube.com/watch?v=1OBvf8QaMXQ

2. ***Algorithm outline***: The basic outline of the algorithm we will use is the following: Every frame is divided into several blocks (the division is the same for all frames). A feature vector is extracted for each block in each frame. Then, an OCSVM model is learned for each block where the first 100 frames (feature vectors) are used as the training set for each model. We run a cross-validation procedure to fit our model more accurately to the problem. Then, for each frame in the test set, we iterate over all blocks to find anomalies. Blocks that contain anomalies are detected using statistical tests that are based on the output of the OCSVM model corresponding to that block. The output of the OCSVM model is the distance between the tested feature vector and the hyper-plane learned by the OCSVM. Finally, a post-processing procedure is used to refine the detection results. In pseudo-code it would look like this:

Train phase:

1. Divide every frame into small blocks.
2. For every block, train an OCSVM using features from the first 100 frames and run cross validation for each model obtained.

Test phase:

1. For every image:
   1.a For every model:
      1a.1 Calculate the distance of the features from the relevant block in the image
      1.a.2 Use a statistical test to decide whether the current distance comes from the same distribution as in the train.

   1.b Create a binary image corresponding to every model's decision

   1.c Post process the result.

When implementing this basic outline, one needs to decide how many blocks to divide the frame, select a feature extraction method, a statistical test, and a post processing method. During our work we tried different feature extraction methods, various statistical tests, several frame divisions and post processing methods. In this report we will show our results and conclusions.

Key terms: In the following, we are going to use some terms, which their definition is given below:
A model – a block in the frame/image, and its corresponding decision rule obtained by training an OCSVM.
A batch of frames – several sequential frames from the video that are taken usually when trying to estimate the distribution of the data around the middle frame.
p-value – A parameter returned by the statistical test, between 0 and 1 indicating the probability that the tested data comes from the same distribution as the null hypothesis. If this parameter is small we decide that there was a change.
Distance – the distance of a feature vector from a hyper-plane learned by the OCSVM

3. ***Variations during development***: In this section we will elaborate on the changes made till we reached the final form of the algorithm. In the beginning of our work we tried to use the high-dimensional Kolmogorov-Smirnov goodness of fit test. The idea was to check the distribution of the data upon ten OCSVMs and not only one. More specifically, we, again, divided frames into small blocks and built for each a model, yet every model was based upon ten OCSVMs each trained to have a certain percentage of outliers as explained in (1). The ten different subspaces defined by the OCSVMs' decision rules created a hierarchy where each subspace was contained by all previous ones as described in (2). Thus we could build a null hypothesis of the distribution of the data upon those subspaces. That is, we could estimate how many inliers will remain after using the decision rules one after another, according to the hierarchy achieved, thus creating a CDF for the train. Doing the same procedure for a batch of frames in the test set, we will obtain another CDF which will allow us to use the Kolmogorov-Smirnov test to see if the train sample and the test sample come from the same distribution. If the distributions are well fitted, there is probably no change, otherwise a change probably occurred.
   This algorithm was over-sensitive, and although we used cross-validation in order to fix our null hypothesis, and tried several feature extraction methods (some of which will be depicted later), the problem remained. Even after changing the algorithm to a supervised version, test frames that where far away (in time) from the train frames where still full of false detections. Yet, when looking at the results of the OCSVM that was trained with a small percentage of outliers, results were much more accurate when using certain features extraction methods. That led us to the conclusion that training the OCSVM so it would classify the train-set (which contains no changes) to have a certain percentage of outliers makes not much sense. Instead, training a single OCSVM to have a low percentage of outliers and comparing the distribution of the distances of the data-points from the hyper-plane will give us a much more accurate statistical test.

4. ***Features extraction***: In this section we will depict our feature extraction methods. During our work several feature extraction methods were tried:
   features extraction method 1: RGB-values
   features extraction method 2: Normalized image (AVG1)
   features extraction method 3:Normalized models (AVG2)
   features extraction method 4: Light simulation

We remind that we divide our frames into blocks, and our feature extraction method should map such a block to a feature vector.

4.1 Method 1: Features extraction method 1 is the simplest; it is just the RGB values. Each block is just turned to a vector containing all RGB values in it. For example, a block of 25 pixels (5x5) was transformed to a 25X3=75-entry vector. Figure 4.1 shows the RGB values of a certain batch of pixels during a video named "office". The block is marked in red in the left image, and, according to the ground truth, during the video there was no change in it. The graph on the right shows the values in each frame, a different color for each entry in the feature vector (although not all graphs are visible here).

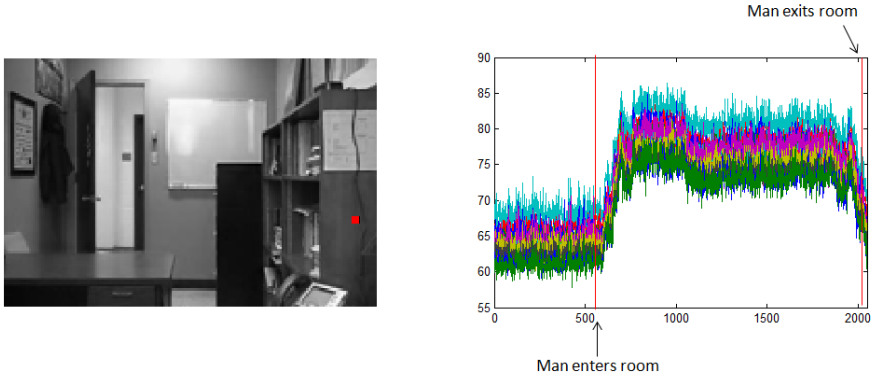


Fig 4.1

The instability of RGB values after training

One can see that although there was no change that we would like to detect in that area, the entrance of a man to the room did have an influence, so values in the test were far from what they had been in training. So here we encountered one of our main problems during this work, change of lighting. When the man enters the lighting is changed, leading to a change in RGB values. Although this is really a change, it is not of the kind that we would like to detect.

4.2 Method 2: In order to make the features more invariant to light changes we tried features extraction method 2, where we first normalize the frame to have a constant average of RGB values. This didn't have much effect.

4.3 Method 3: In features extraction method 3 we localized what we have done in method 2, that is, we normalized each block to have a constant average (a method we call AVG2), the features were much more stable. Figures 4.2 and 4.3 show the feature vectors using AVG2 (method 3) in the same manner that figure 4.1 did.

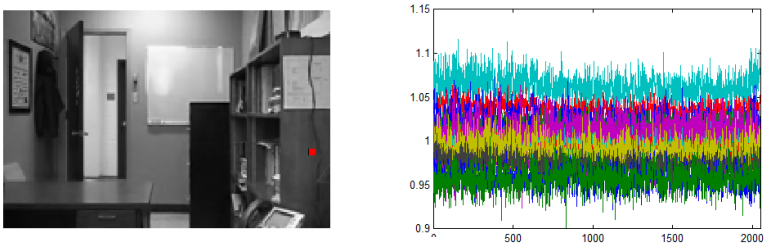


Fig 4.2
The stability of AVG2

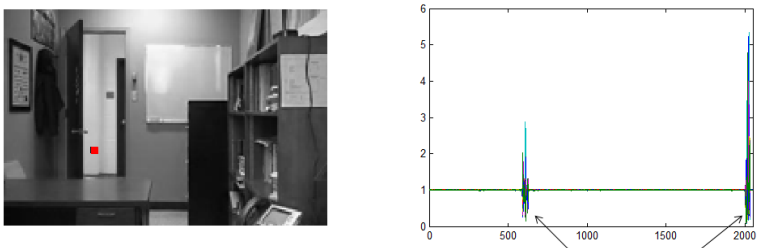


Fig 4.3
The stability of AVG2

One can see that the entrance of the man has only a small effect where there are only light changes as in figure 4.2, and figure 4.3 shows that these features are still sensitive to changes we would like to detect.

Yet now we faced another problem, this time using the statistical test. The features were so stable during training so the OCSVM returned a very strict classification rule, and as a result every small change made a very big difference in the distribution of the data. This made it very hard numerically to determine an appropriate threshold from which p-value to classify a model as changed, and thresholds varied from different sets. For example, figure 4.4 shows how small changes in the feature vectors led to great changes in the distribution.

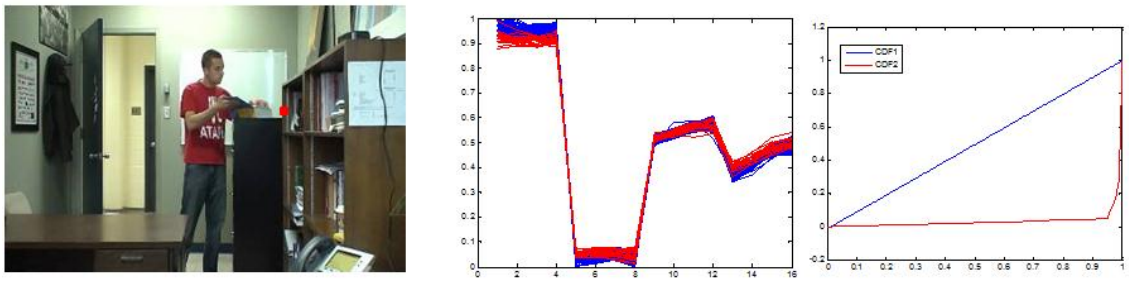


Fig 4.4
AVG2 causes problems in the statistical model phase

The red square in the left image shows the model tested. Clearly there is no change there that we would like to detect, but the man's close presence makes a little difference in the features vector. The middle graph presents the feature vectors. Each feature vector here has 16 entries, and is presented as a line

from 1 to 16 which goes through the corresponding values to each entry. Feature vectors from the training set are colored in blue and from the test set are colored in red. One can see that there is only a small change from test to train. The rightmost graph shows the distribution of the feature vectors in a CDF. The blue line corresponds to the train and the red one to the test. It shows that the two come from a different distribution, although actual changes are very small. Choosing a very low threshold gave very good precision but very low recall. Increasing the threshold damaged the precision pretty fast because it was harder to differ between changes in the distribution due to small noises as in Fig 4.4 and real changes since both changed the distribution significantly. Fig 4.5 is a precision-recall graph for method 3 on the "office" video, showing the damage to the precision.

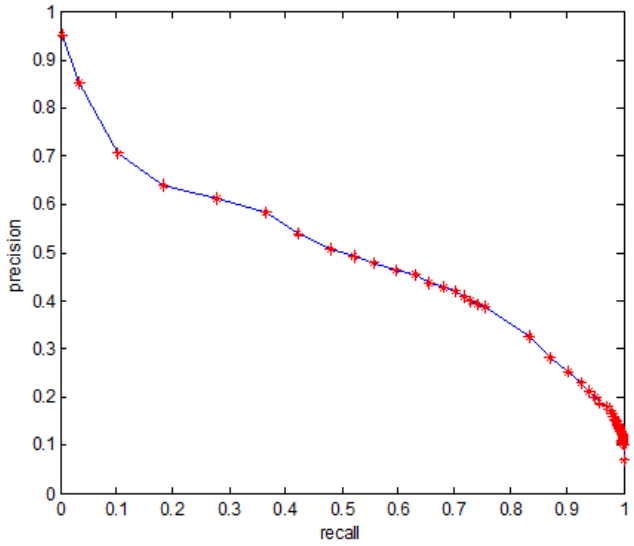


Fig 4.5
AVG2: precision deteriorating fast

4.4 Method 4: Features extraction method 4 is light simulation. It is the same as RGB-values (method 1), but for the training phase. In the training phase, before extracting the features we manipulate the whole image, simulating change of light. We will explain how we do it and simultaneously show a mini-example. Our example will focus only on what happens in one color channel, but the same occurs in all color channels simultaneously. Say our full image is the following 5x5 matrix:

12	1	5	3	7
15	4	8	20	13
6	21	14	9	25
5	28	18	16	10
2	38	22	19	19

We create a Gaussian the size of the image, with a random center. For example:

0.535261	0.778801	0.882497	0.778801	0.535261
0.606531	0.882497	1	0.882497	0.606531
0.535261	0.778801	0.882497	0.778801	0.535261
0.367879	0.535261	0.606531	0.535261	0.367879
0.196912	0.286505	0.324652	0.286505	0.196912

Is a the Gaussian formed with a center at (2,3) (in yellow), STD of 2. Each cell can be calculated using the formula $e^{-\frac{(row-2)^2+(col-3)^2}{2\sigma^2}}$ where row is the row of the cell and col is the column, σ in this case is 2. Then we randomly and uniformly pick a number, r, between 0.5 and 2 and transform the Gaussian linearly to values in $\text{int}(1,r)$ (that is, the interval between 1 and r).

For example if r=2, the transformation will result in:

1.421311	1.724564	1.853686	1.724564	1.421311
1.510055	1.853686	2	1.853686	1.510055
1.421311	1.724564	1.853686	1.724564	1.421311
1.212888	1.421311	1.510055	1.421311	1.212888
1	1.111561	1.159062	1.111561	1

Now we multiply this result with our image, element by element. In our case we achieve:

17.05573	1.724564	9.26843	5.173693	9.949175
22.65082	7.414744	16	37.07372	19.63071
8.527865	36.21585	25.9516	15.52108	35.53277
6.064439	39.7967	27.18098	22.74097	12.12888
2	42.23931	25.49936	21.11965	19

Cell (2,3), for example was obtained by $8 \cdot 2$, and cell (3,1) by $6 \cdot 1.421311$.
The result of this simulation is:
If r is greater than 1, the chosen center is multiplied by r (brightened), and the most distant pixels from the center are unchanged. If r is smaller than 1 the most distant pixels from the center are darkened and the center is unchanged. Figures 4.6 and 4.7 show the results of such a procedure on a frame from the video “sofa”.

Fig 4.6
Simulating change of brightness, r=1.2
The cross is at the center of the Gaussian

Fig 4.7
Simulating change of brightness, r= 0.7
The cross is at the center of the Gaussian

What led us to this feature extraction method is that even when trying other methods, such as: gradient maps (both in polar and Cartesian coordinates), or a directional histogram, we always ended up with one of the problems described above: the features change from train to test although there was nothing we would like to detect as changed (as in RGB features), or were too stable in the train set and thus causing a problem in the statistical test (as in AVG2).

This led us to search for features with variance, so that the statistical test will work, but also have the same variance in the train, so that training will work. Since our main problem was change of light we focused on that issue. Choosing features that are invariant to lighting won't have much variance, yet choosing features that do vary with different lightings will always have a different distribution on train and test no matter if there was a change or not. Our solution was to use RGB features but to "add illumination noise" to the training set by simulating change of light. If we succeed in simulating the change of lighting our features will vary, but the train will vary as well. And indeed, this method gave us the best results.

Indeed, this method has the advantage of features with variance, when variance is even greater during train phase than in the test phase. This makes the algorithm less sensitive, and allows the OCSVM to use a lot more of the information given in the image. For example, using AVG2 maybe makes the features invariant to changes in lighting but also looses the information of the intensity in each color channel making it even impossible to differ between a white patch and a black patch. However, method 4 has no problem since the RGB values in a white patch differ from those in a black patch, even after our light change simulation.

A remark on light simulation: Additive models were tried as well but worked poorly. The reason for that may be that the amount of light returned from a surface depends on the direction of the light and the surface's albedo, both which we don't know. For example, for a Lambertian surface we know that $radiance = \frac{\rho}{\pi} irradiance$ where ρ is the albedo. Simulating changes in irradiance, just by adding more radiation will ignore the albedo and the light direction. Yet, because radiance is linear in the irradiance, multiplying the irradiance hitting the surface is the same as multiplying the radiance. So, multiplying the radiance simulates increasing the illumination by a certain factor and yet doesn't ignore the albedo.

5. ***P-value estimation:*** We remind that in the basic algorithm (framed in section 2) one has to decide how to estimate a p-value according to which we decide whether a change occurred or not. In this section we will present several ways (statistical tests) to do that. We tried the following variations:
Variation 1: Kolmogorov-Smirnov test
Variation 2: Kolmogorov-Smirnov test combined with Chebyshev inequality
Variation 3: Chebyshev inequality
Instead of trying every variation here with every feature extraction method in section 4, we tested these variations with the best method we got in 4 (method 4). Yet, results for other feature extraction methods can be found in section 5.4

5.1 Variation 1: Variation 1 was to use the Kolmogorov-Smirnov (KS) goodness of fit test. We took a batch of frames of length BATCH_LENGTH and for every model calculated the distances using the trained OCSVM for each frame. Than we calculated how well the distribution of these distances does fit to the distribution of the distances in the training phase using the KS-test. If they would fit poorly, the block of the middle frame would be classified as changed. We used an asymmetric version of the KS-test (we used kstest2 in Matlab with 1 for 'type'), because if distances are more negative than expected, this probably means that the features are more likely to signal no-change although the distribution is different. Figure 5.1 shows precision recall graphs for several videos using this statistical test. One can see that in several videos precision remains high longer than what we've seen in figure 4.5, yet still in most of the videos' precision starts decreasing fast before we achieve satisfying recall. So this variation is probably too sensitive.

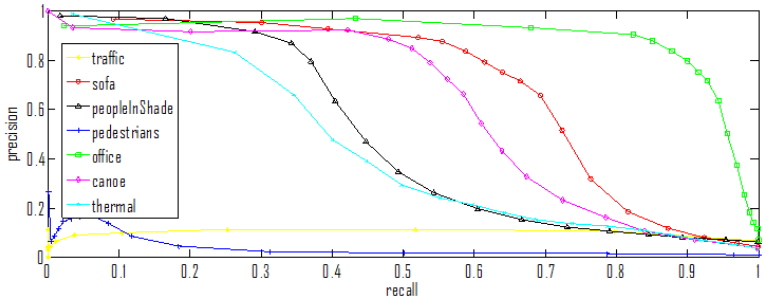


Fig 5.1
Precision-Recall of KS-test

5.2 Variation 2: In variation 2 we tried to get a lower bound on the CDF Using Chebyshev's inequality rule. The inequality is the following: $\Pr(|X - EX| \geq k\sigma) \leq \frac{1}{k^2}$ where sigma is the STD. We build a CDF such that for each distance in the train set the value would be $1 - \frac{\sigma^2}{(X-\mu)^2}$ (where μ is the mean of the distances).
Figure 5.2 shows the results for this method.

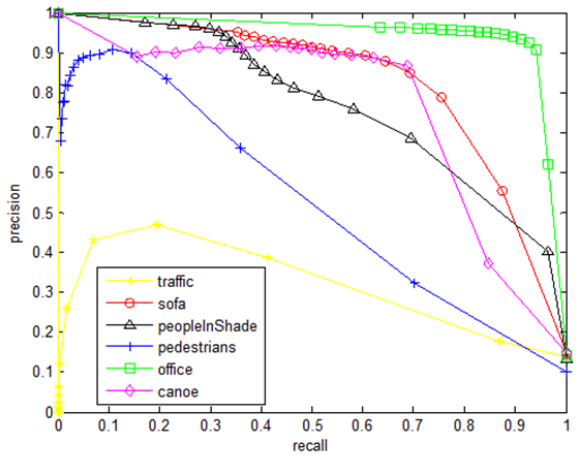


Fig 5.2
Precision-Recall graphs for variation 2
using feature extraction method 4

One can see that there is an improvement compared to variation 1,and indeed we used a lower bound on the CDF in order to make the test less sensitive. Yet improvement differs between datasets, and some are still far from satisfactory. A great disadvantage of this test is that it is based on a distribution in a batch. Our batches are pretty small, and estimating a distribution based on them is quite hard. Besides that, the fact that we manipulate the training set and only simulate changes, also probably has its effect.

5.3 Variation 3: In variation 3 we used only the Chebyshev inequality. Again, for each model we've calculated the distance using the OCSVM, but now using the mean and STD of the distances in the train set we got from the Chebyshev inequality an upper bound on the probability that such a distance may appear. If that upper bound was small this is probably because of a change in that model. This variation gave us the best results, but actually here we used the one sided Chebyshev inequality. That is, the code looked like:

1. $dist \leftarrow current\ distance\ \ from\ the\ hyper - plane$
2. $k \leftarrow (E(distances\ in\ train) - dist) / STD(distances\ in\ train)$
3. $pvalue \leftarrow \frac{1}{1+k^2}$
4. $if\ \ pvalue < thresh\ mark\ \ model\ \ as\ changed$

Figure 5.3 shows a Precision-Recall graph for this variation (using feature extraction method 4).

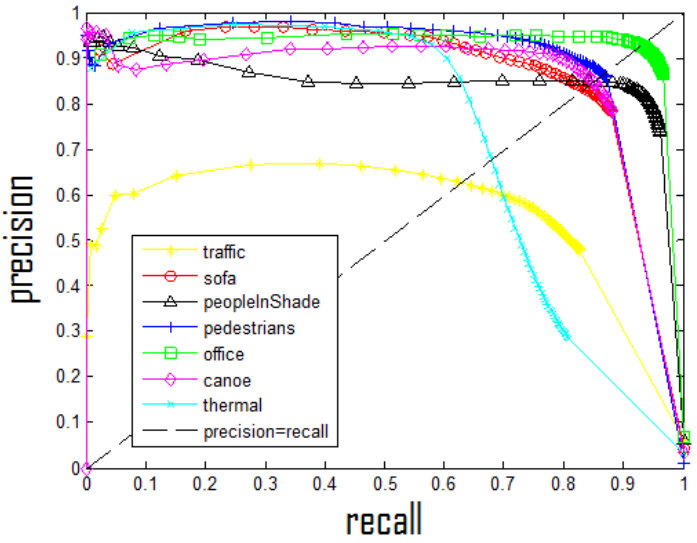


Fig 5.3
Precision-Recall for Chebyshev inequality

One can see that this variation, combined with these features gives very stable results. In most cases precision barely decreases till high recall.

5.4 Other experiments with statistical tests: We also tried to use the Hoeffding inequality in the same manner we used the Chebyshev inequality (variation 3). But this inequality needs upper and lower bounds on the values of the distances. The bounds that we used were too weak and didn't enable us to distinguish between change and no-change.

For comparison figures 5.4 to 5.6 show other features extraction methods using Chebyshev inequality (variation 3). One can conclude from these graphs that a good statistical test is not enough; it is the combination of the statistical test and the features extraction method that leads to proper results.

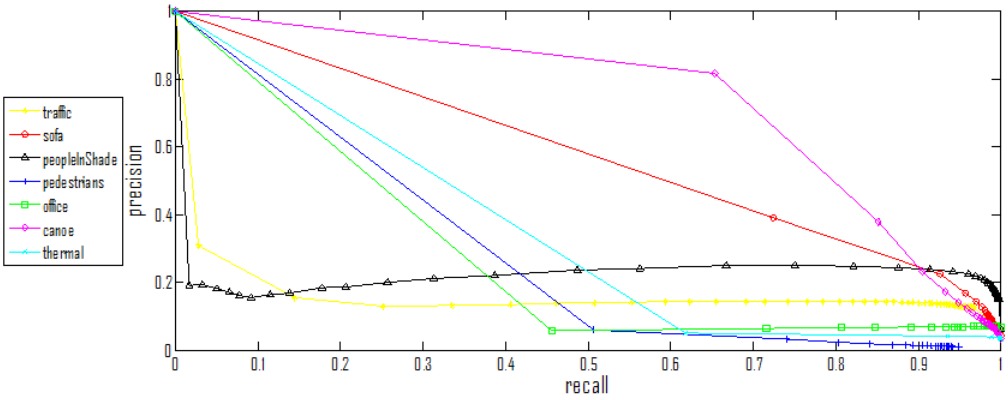


Fig 5.4
Precision-Recall for Chebyshev inequality
Using RGB as features

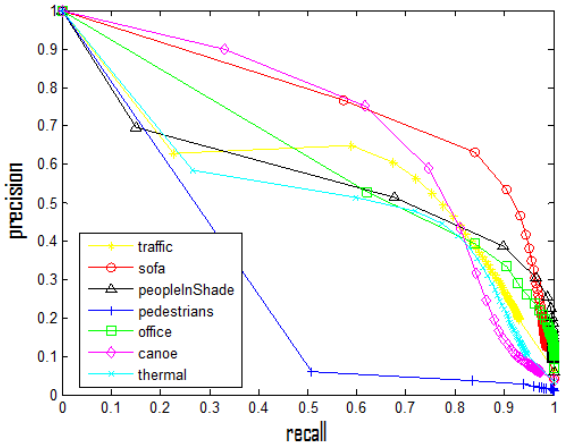


Fig 5.5
Precision-Recall for Chebyshev inequality
using AVG2 features with light simulation in train

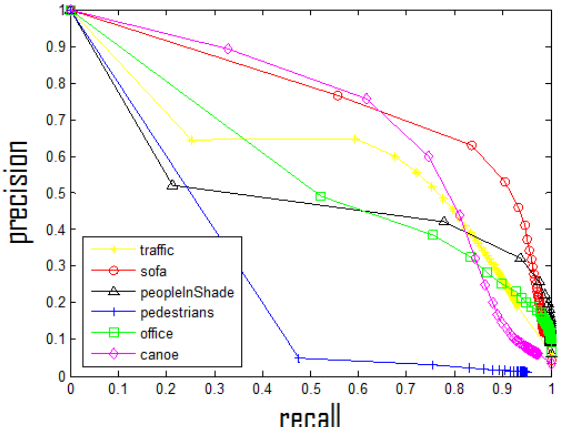


Fig 5.6
Precision-Recall graph Chebyshev inequality using AVG2
features only

(One can also see that the precision recall graphs of AVG2 alone compared to the one with light simulation are very similar. This implies that this features extraction method really is invariant to small light changes.)

A remark on the results: There are two videos that still give poor results, “traffic” and “thermal”. The first suffers from camera jitter, yet jitter in the train is much smaller than in the test. This problem needs to be dealt by different means than which we’re discussing now, so we will leave it for now. The problem in the “thermal” video, however, is that in some point the scene is much brighter than in the test. Our light simulation should have solved this as it did in the other videos, but this time the difference is bigger than usual.

6. ***Number of models***: In this section we will show how the number of models influences the results. There is a tradeoff between the number of models in which one parts the image to and the resolution of the detection, the algorithm’s time consumption and the quality of the learning phase. In our case, we decided that every model will cover 25 pixels (a 5x5 square) and we will resize the image so it will contain the right amount of pixels in order to have the wanted number of models. So instead of asking how many models we need, we asked what the appropriate resizing factor is. Figure 6.1 shows the f-measure of several videos with a certain threshold (the final one we used,0.26) at different resizing factors.

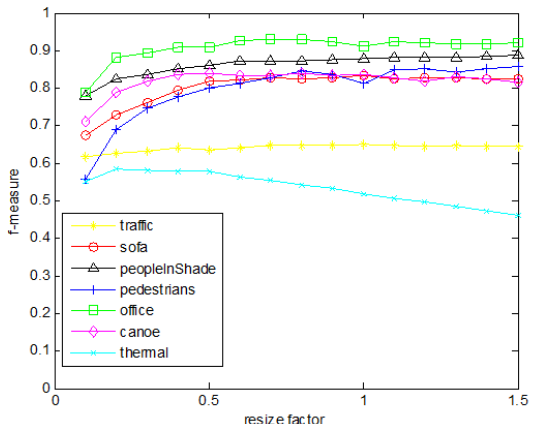


Fig 6.1
F-measure as a function of the resize factor

One can see that in this case, in most videos the f-measure gets into a saturation point at about 0.5-0.8. Yet, an interesting result is that although one might think that a bigger resizing factor (which leads to smaller models in the original image) will give better precision(since the resolution of the detection is finer) in practice the opposite happens. Increasing the resizing factor decreases precision and increases recall. A possible explanation is that resizing the image will bring up the need to interpolate data, which is only a speculation and adds noise. This makes the model more sensitive because a small change in the

data will bring to a big change in the model, and so recall will grow and precision will decrease. Another possible explanation is that when the resize factor is big every model is associated only with a small block in the image, and so it will be sensitive to local changes, whereas with a smaller factor the model will be associated with a larger neighborhood making it more invariant to small local changes which we probably don't want to detect.

We chose to use a factor of 0.8. We preferred this factor over smaller ones since it gave a bit more recall, though decreasing precision. Since precision may be a bit misleading some times, we preferred a bit more recall over precision; this gave visually the best results. On the other hand, we didn't take a bigger factor, although from figure 6.1 this seems the right thing to do. The reason for that is that bigger factors are much more sensitive to changes in the threshold, and decrease their f-measure with a slightly different threshold. Figure 6.2 shows this phenomenon in the videos "office","sofa" and "canoe".

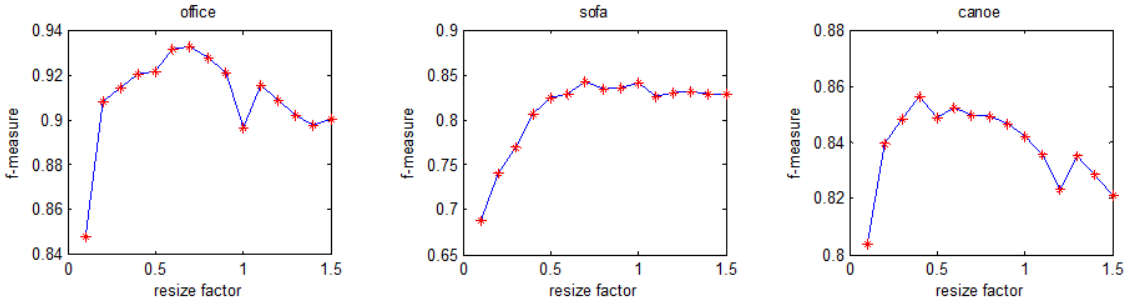


Fig 6.2
Change of threshold decreases the f-measure in high factors

In these graphs a slightly different threshold (0.35) was used. We can see that the f-measure in 0.8 was almost unchanged, whereas the f-measure in factors greater than one decreased.

Besides that, choosing 0.8 instead of a bigger factor improves time consumption (less models to learn) without damaging the results.

7. ***Post processing***: In this section we will offer several ways to post process a binary image in order to classify changes more accurately. We will elaborate on three methods:

   Method 1: Adding edges and filling holes

   Method 2: Connecting close components using convex hull

   Method 3: Adding edges and then expanding and shrinking the detections.

7.1 Method 1: The first method is an edge driven method; Using the input frame we extracted the edges using the Canny method (using default parameters in Matlab). We added edges which are only 5 pixels away from a detection to the binary image. (step 1)

Then we filled all background areas that are not reachable from the border of the image (holes)(using the imfill function in Matlab), yet in order to fill areas in the border of the image we add another border around the image before filling holes.(steps 2-3)

We then used morphological open with structural elements of a vertical line and a horizontal line in order to delete redundant edges. (step 4)

Each area filled which is greater than a certain threshold is divided into several smaller parts. The division is from top down, left to right to small chunks. Each chunk that its RGB mean (a 3-dimentional vector) is far from the mean surrounding the area is deleted. (step 5)
In the end very small detections are deleted.(step 6)
The pseudo code would be the following:

1. Find edges using canny method and add edges that are at most EDGE_THRESH1 from a detection to the binary image.
2. Add a border to the image in order to fill areas in the original border of the image.
3. Fill holes
4. Perform morphological open with a horizontal and a vertical line of length LINE_LENGTH1.
5. For each connected component that was filled:
   5.a If the size of the component is larger than TH_SZ1 do:
      5.a.1 Compute the mean (in every channel) of pixels that are at most TH_SURROUND1 from the area.
      5.a.2 number the pixels of the component top-down, left to right and for every chunk of SMALLER_AREA1 pixels do:
         5.a.2.a If the norm of the difference between the mean of the chunk and the surrounding is greater than TH_MEAN1 delete that chunk

6. Delete every component that is still smaller than TH_SMALL1

Figure 7.1 shows the procedure on a frame from “peopleInShade”.

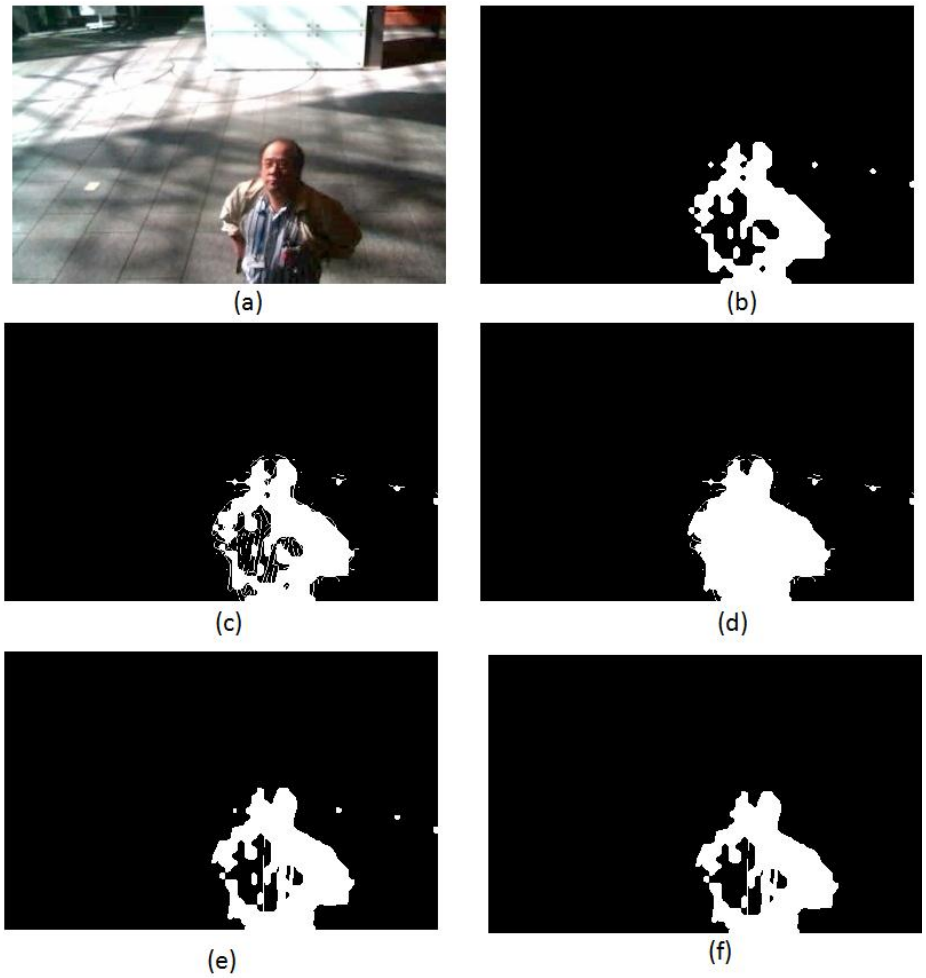


Fig 7.1

The first method on “peopleInShade”

a- The original image
b- The binary image
c- After adding close edges
d- After filling
e- After deleting big areas with a different mean than their surrounding
f- Final result, after deleting small components

This is a very safe method that usually increases both precision and recall, but changes are minor, and mostly aren’t visible.

7.2 Method 2: The second method is a convex hull approach. Connected components that are close enough are considered as one connected component (steps 1-2), which is found by the convex hull of the two (step 3).

To avoid redundant fillings due to a concave outline of the object, only pixels close to the original detection in the convex hull will be added (step 4).

Finally very small components are deleted. (step 5)

In pseudo-code it will look like this:

1. Add to the binary image any pixel that is at most TH_DIST2 from a detection.
2. Find connected components and mask with the original binary image. (now we have components that are at most 2* TH_DIST2 away from one another labeled as one component. That is, close components are labeled the same)
3. Find the convex hull of every component (that is, for every label).
4. Delete any new detection that is more than TH_DIST2 pixels away from an original detection.
5. Delete every component that is still smaller than TH_SMALL2

Figure 7.2 shows the process on a frame from “sofa”

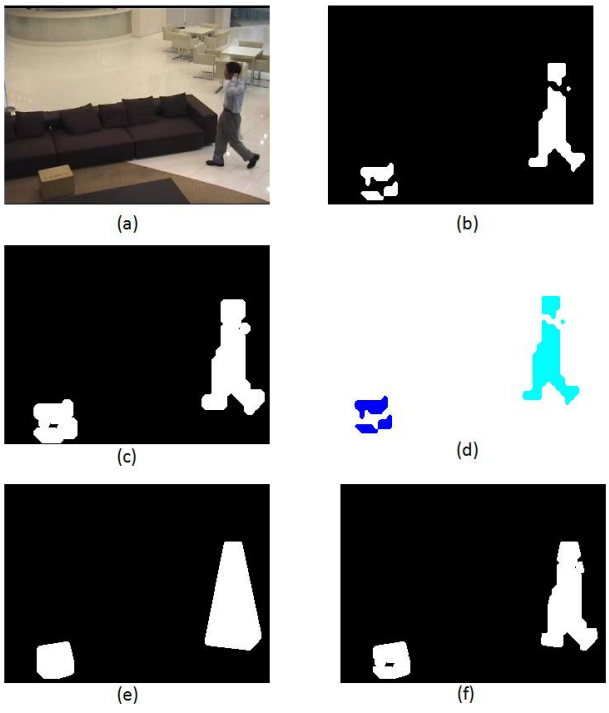


Fig 7.2

The second method on “sofa”

a- The original image
b- The binary image
c- After step 1
d- After step 2. Every component is marked in one color
e- After step 3
f- After step 4. Step 5 has no effect since there are no small components

This method increased recall yet damaged precision because of concave outlines of objects that added a lot of false detection around them.

5.3 Method 3: The third method we tried is some sort of a hybrid between the first two. It is, again, edge aided. First we delete small components that are far away from any other component (step 1). Then edges in the input that are close to detections are added to the binary (step 2).
In order to connect close components we expand the objects in the binary and then shrink them back (steps 3-7). As a result holes are filled, yet the outline of the object is almost unchanged.
In the end, we again use morphological opening with lines to delete remainders of edges (step 8).
Since these operations could have deleted some of the original detection we add the original binary image (step 9).
Finally we delete very small components. (step 10)
In pseudo-code it will look like this:

1. For each component that is smaller than TH_SZ3:

   1.a calculate the minimal distance in pixels to another component. If it is greater than TH_DIST3 delete that component.

2. Find the edges in the original image using the canny method (using default parameters in Matlab), add edges that are at most TH_EDGE3 pixels away from a detection to the binary image.
3. Add a border of width EXPAND3 to the binary image.
4. Add pixels that are at most EXPAND3 pixels from a detection in the resulting binary (expanding the objects)
5. Find the borders of the expanded objects (we've used Prewit edge detector on the new binary image)
6. Leave only detections that are at least EXPAND3 pixels away from the borders found in step 5. (shrinking the objects back)
7. Delete the border added in 3
8. Use morphological open with a horizontal and a vertical line of length LINE_LENGTH3. (two separate open operations)
9. Add the original binary to the result (maybe things were deleted)
10. Delete every component that is still smaller than TH_SMALL3

This method usually increased recall without damaging precision severely. Figure 7.3 shows the process on a frame from "thermal"

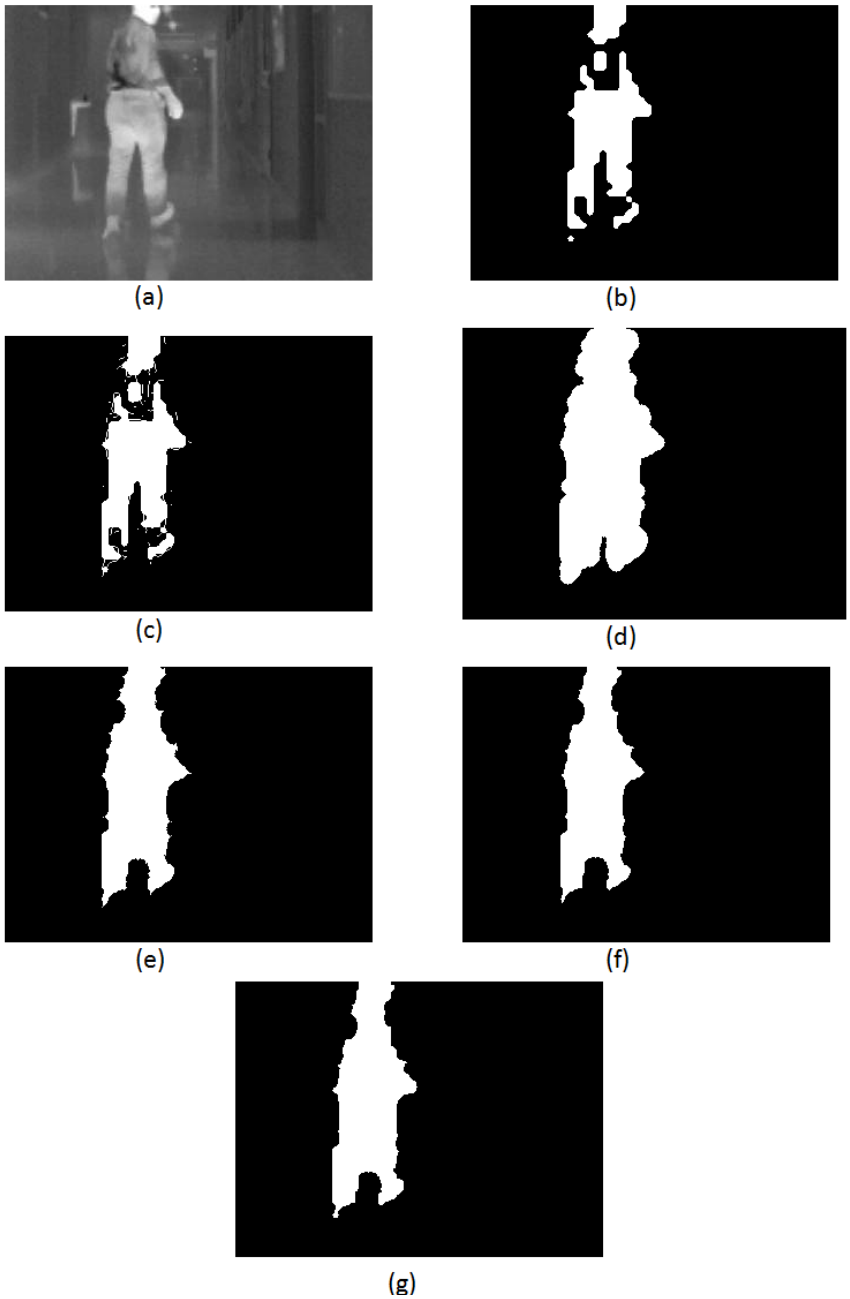


Fig 7.3

The third method on “thermal”

a- The original image
b- The original binary image
c- After step 2
d- After step 4
e- After step 6
f- After step 8
g- After step 9, this is also the final image

7.4 Summery of Experimental results

The following table concludes the precision recall results for each method (using feature extraction method 4 and p-value estimation by variation 3)

Video name	No post processing		First method		Second method		Third method	
	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall
traffic	0.562	0.7602	0.5981	0.7755	0.492	0.8678	0.4894	0.8553
thermal	0.4232	0.7495	0.4256	0.7635	0.3916	0.8599	0.4255	0.8656
sofa	0.8605	0.7934	0.8661	0.82	0.7678	0.9404	0.8282	0.9286
peopleInShade	0.8338	0.9178	0.8459	0.9427	0.7772	0.9808	0.8217	0.9832
pedestrians	0.9254	0.7760	0.937	0.7756	0.7513	0.8188	0.8408	0.8547
office	0.9295	0.9265	0.9416	0.9383	0.8289	0.9822	0.9013	0.9932
canoe	0.9073	0.7832	0.9537	0.8168	0.8402	0.9343	0.8657	0.9425

Since false detections close to true detections are less problematic, we conducted the following experiment: we re-calculated precision for the second method ignoring false detections 5 pixels away from the ground truth. As expected the precision jumps:

Video name	Precision after post processing.
traffic	0.5256
thermal	0.4584
sofa	0.8672
peopleInShade	0.8170
pedestrians	0.8926
office	0.9025
canoe	0.9069

This indicates that in most videos, for the second method, false detections were caused due to concave outlines that were bounded by a convex shape.

The following table shows the f-measure for each video and method, calculated from the first table:

Video name	No post processing	First method	Second method	Third method
traffic	0.6462	0.6753	0.6279	0.6226
thermal	0.541	0.5465	0.5381	0.571
sofa	0.8256	0.8424	0.8454	0.8755
peopleInShade	0.8738	0.8917	0.8672	0.8952
pedestrians	0.8441	0.8487	0.7836	0.8477
office	0.928	0.94	0.8991	0.945
canoe	0.8407	0.88	0.8848	0.9025

Looking at the tables above we've decided to use the third method.

8. ***Conclusions and possible future work***: In this work we've gone over several feature extraction methods, acknowledging that features during train should vary in the same way they vary in the test. Invariant features will lead to an "over fitted" model which will be very sensitive. Yet features that vary differently in train and test will also be too sensitive. In order to solve that we offered to simulate changes during training phase, and introduced a way to simulate light changes. We've seen that in light simulation it would be better to multiply the noise instead of adding it. Future work can be done in order to simulate camera jitter as in the "traffic" video or find other ways to overcome this problem like baseline-alignment. Also one might try to find a way to determine automatically to what extent should we simulate light changes, as we've seen, the change needed in most of the videos was different from the change needed in the "thermal" video.

   We've tried several statistical tests, and concluded that it is better to estimate the probability that a certain sample is taken from some distribution as using chebyshev inequality, than to do so for a set of samples like in the KS test. We've investigated how the number of models influences precision and recall and decided on a resize factor of 0.8. Three methods for post processing were introduced. Future work can focus on shade subtraction which afterwards will ease the post processing part and probably will make it more accurate.

References

1. B.Scholkopf,J. C.Platt,J. Shawe-Taylor, A. J. Smola,R. C. Willison. *Estimating the Support of a High-Dimensional Distribution*
2. A. Glazer, M. Lindenbaum, S. Markovitch. *Learning High-Density Regions for a Generalized Kolmogorov-Smirnov Test in High-Dimensional Data. NIPS, 2012*

Technical Appendix

The parameters we use during this work:

N=100 – the number of frames in the training set. (see section 1)

The sigma of the Gaussian in light simulation - 120 (probably, this will need to be changed when working with images with a significantly different size than what we have worked with). (see section 4.4)

RANGE=[0.5 2] - the range from which a number, r, is picked during light simulation in order to transform the Gaussian to the interval between 1 and r. (see section 4.4)

Resize factor: 0.8 – A factor for resizing the image, thus determining the number of blocks in an image (see section 6)

Block length: 5 - Each frame is divided into blocks of size (Block length) x (Block length) , see section 2

Threshold for p-value : 0.26 – A threshold for determining according to a p-value whether a change occurred or not. A p-value smaller than the threshold indicates change. This is the threshold for variation 3 in section 5 (the final one we used)

BATCH_LENGTH=21 – The number of frames used when using variations 1 and 2 in section 5.

EDGE_THRESH1 = 5 – The maximal distance from a detection for an edge to be added to the binary image in step 1 in the first post processing method (section 7.1)

LINE_LENGTH1 = 3 – The length of the structural elements in step 4 in the first post processing method (section 7.1)

TH_SZ1 = 200 – The minimal number of pixels in a component in order for it to be considered as too big in step 5.a in the first post processing method (section 7.1)

TH_SURROUND1=10 – The distance from a component, defining its surrounding in step 5.a.1 in the first post processing method (section 7.1)

SMALLER_AREA1 = 50 – The chunks' size into which big components are divided to in step 5.a.2 in the first post processing method (section 7.1)

TH_MEAN1 = 20 – The maximal difference between a chunk's mean and the surrounding mean in order not to delete that chunk in step 5.a.2.a in the first post processing method (section 7.1)

TH_SMALL1 = 200 – The maximal number of pixels in a component in order to consider it as a small component in step 6 in the first post processing method (section 7.1)

TH_DIST2 = 5 – If two components have a path between them, so every pixel in that path is at most TH_DIST2 pixels away from a detection, these components will be considered as one in the second post processing method (section 7.2 see steps 1-2). That is, this parameter specifies how close components should be in order to be united. This parameter also restricts the convex hull of the merged components, see step 4.

TH_SMALL2 = 200 - The maximal number of pixels in a component in order to consider it as a small component in step 5 in the second post processing method (section 7.2)

TH_SZ3 = 150 – The maximal number of pixels in a component in order to consider it small in step 1 in the third post processing method (section 7.3)

TH_DIST3 = 15 – The minimal distance from a component to another component in order for it to be considered as a distant component in step 1.a in the third post processing method (section 7.3)

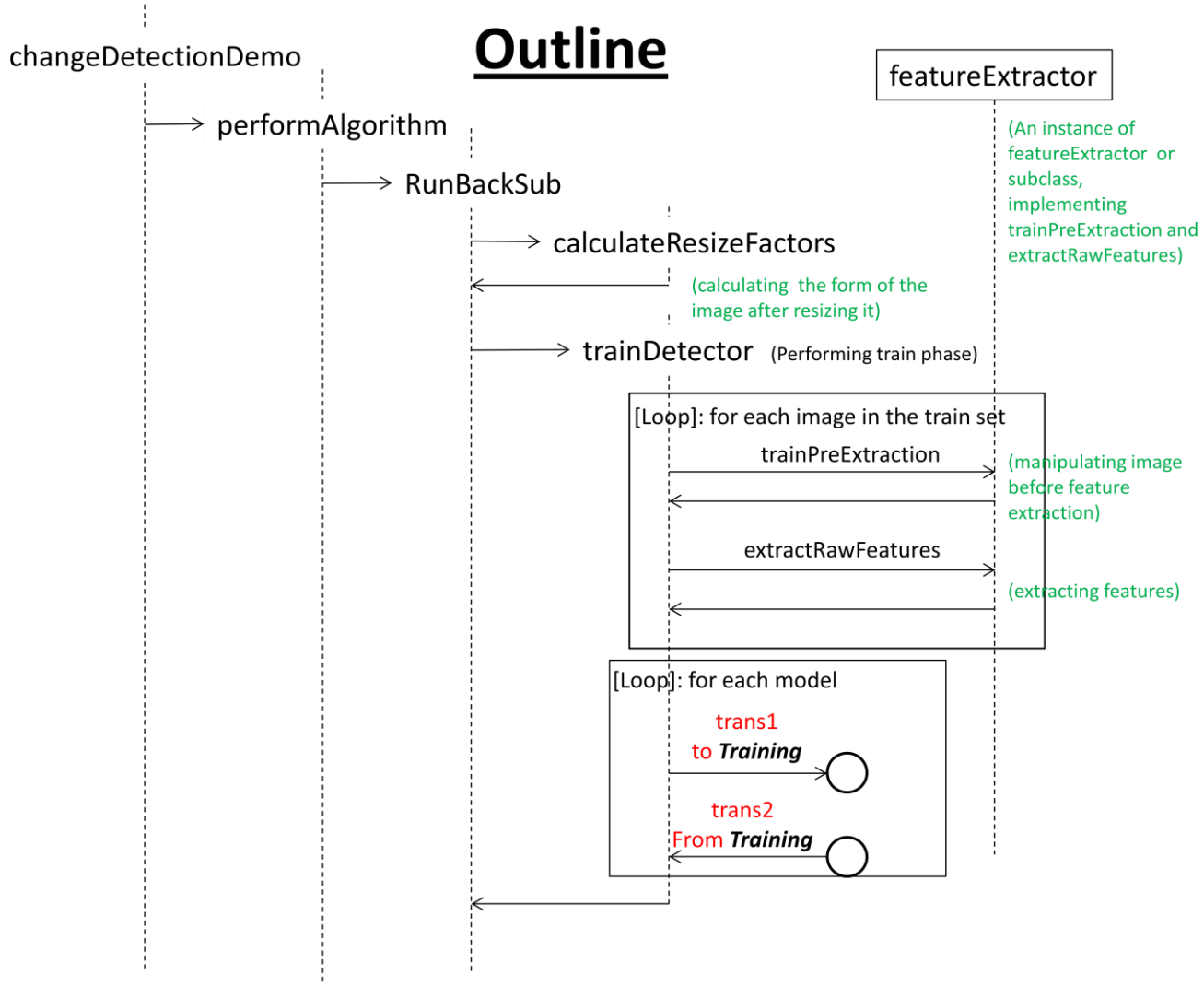
TH_EDGE3 = 5 - The maximal distance from a detection for an edge to be added to the binary image in step 2 in the third post processing method (section 7.3)

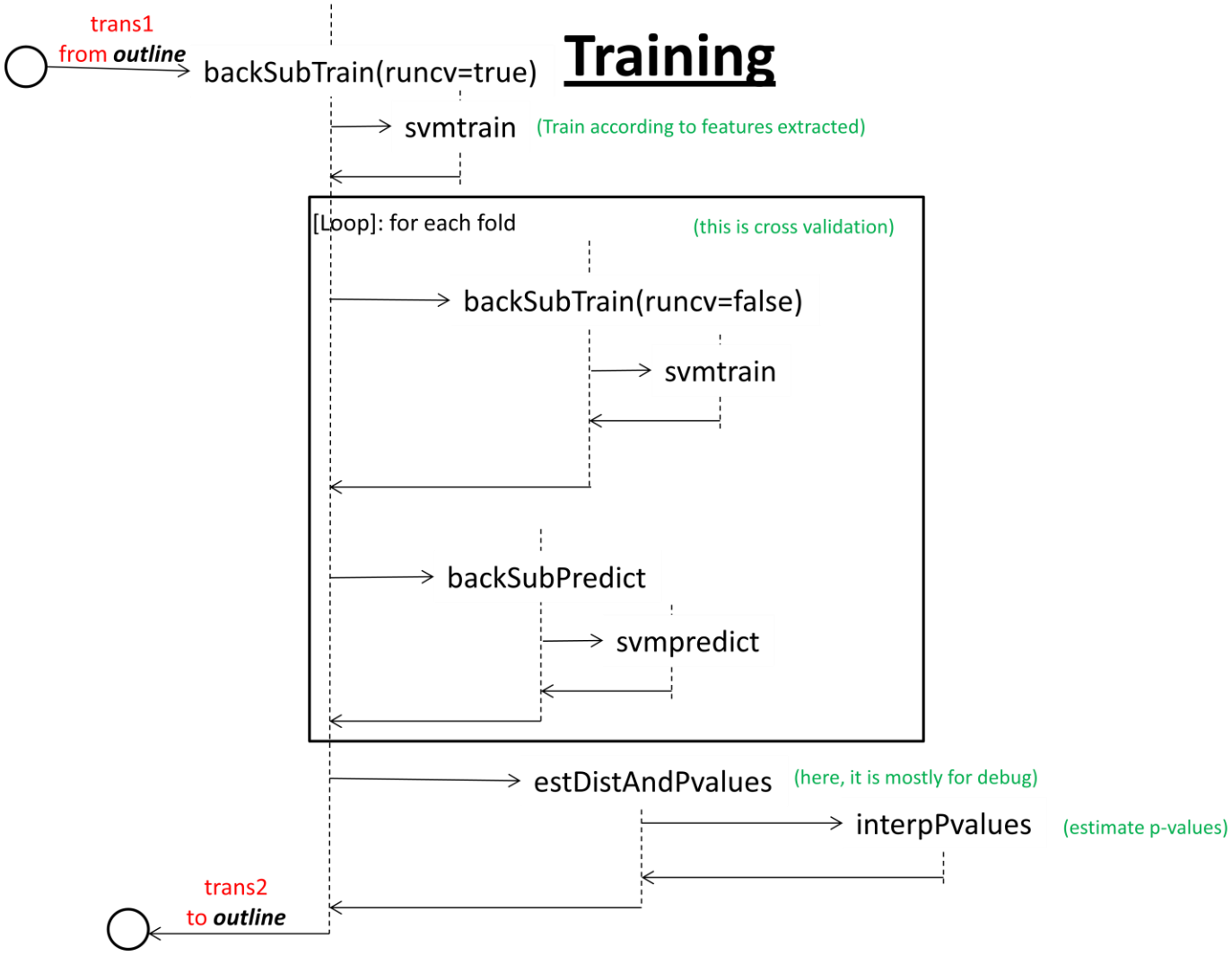
EXPAND3 = 7 – The number of pixels in which to expand and shrink the detections in steps 3-6 in the third post processing method (section 7.3)

LINE_LENGTH3 = 3 – The length of the structural elements in step 8 in the third post processing method (section 7.3)

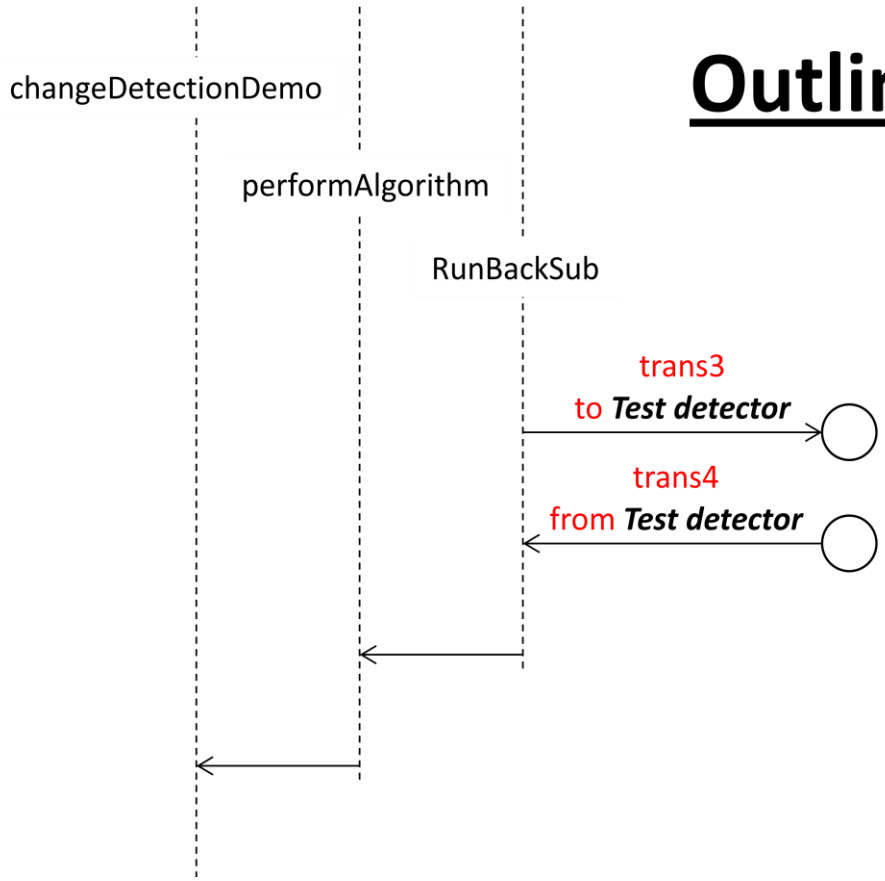
TH_SMALL3 = 200 - The maximal number of pixels in a component in order to consider it as a small component in step 10 in the third post processing method (section 7.3)

Flow Chart of Supplied Algorithm




trans1
from outline
backSubTrain(runcv=true)
Training
svmtrain
(Train according to features extracted)
[Loop]: for each fold
(this is cross validation)
backSubTrain(runcv=false)
svmtrain
backSubPredict
svmpredict
estDistAndPvalues
(here, it is mostly for debug)
interpPvalues
(estimate p-values)
trans2
to outline

Outline Cont.



Test detector