CS 221: Object Recognition and Tracking

Sandeep Sripada(ssandeep), Venu Gopal Kasturi(venuk) & Gautam Kumar Parai(gkparai)

1 Introduction

In this project, we implemented an object recognition and tracking program that identifies and labels occurrences of 5 objects (mug, stapler, keyboard, clock, and scissors) in a given video clip. The program would be trained on labeled still images and tested on an unseen video clip. We implemented a separate classifier to identify each of the five objects that was trained on the given data. The classifiers that we tried include Logistic Classifier, Boosted Decision Trees and Support Vector Machines. The feature for images were generated based on localized fragments and HOG (Histogram of Oriented Gradients). We also included a motion tracking algorithm based on filtering to help recognize objects using the temporal information.

2 Feature generation

2.1 FAST

We implemented a localized fragment based feature generation using the heuristic described in [Rosten 2007] to obtain interest points and then using regions around them as fragments most representative of an image. We extracted the regions around the interest points to obtain templates along with their positions in the original image which formed the feature dictionary. We trained a classifier using boosted decision trees on a small subset of the training images and obtained the most informative features and discarded the others.

The feature value for any image with respect to a template 'T' is given by the correlation of the image with the template in the neighborhood of the template. To achieve this, we calculate the response image of the input image with respect to T and obtain the feature values by using a technique called Max-Pooling.

The efficiency of this approach lies in choosing the templates for the representative samples. Instead of randomly slicing fragments from a sample, we chose the "interesting points" from the images as templates. The used the idea that corners give us distinctive features of objects and hence applied a corner detection algorithm as in [Rosten 2007].

A corner is defined as a location when there two edges which intersect at a point and the angle between them is crossing a certain threshold value. FAST corner detection algorithm defines the corner at a Point 'P' when the neighboring pixels' intensity is greater than or lesser than a certain threshold t as shown in Figure 1. The pixel values at the neighboring points 1-6, 11-16 are lower than that at point 'p' thereby giving us a corner.

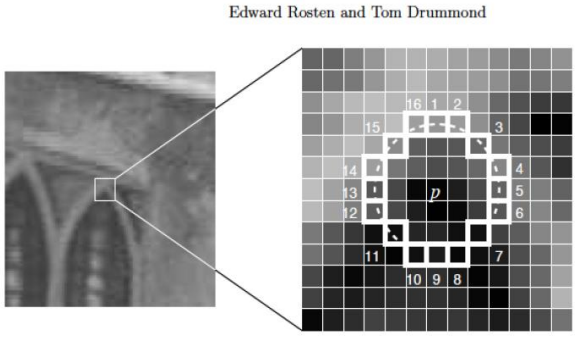

Figure 1: Corner detection using FAST

The flowchart in Figure 2 gives an outline of the method. Some of the steps involved in feature generation are as described below:

Step 1: FAST corner detection algorithm is applied on the input samples for each object. The output is the templates of the samples on the corners/interest points. Some optimization that we used:

- We set the non-maxima suppression parameter to suppress those corners which are non maximas in the neighborhood of the point 'p'.
- As the desired template size was 16x16, we eliminated corners whose templates overlap each other by more than a certain threshold.

The interest points generated for some of the input images is shown in Figure 3. Both the 'red' and 'blue' crosses mark the corners detected with a threshold of 100. But using the overlap optimization we consider only the 'red' crosses as the templates.

The number of templates obtained on the 'mug' data set on various thresholds is as shown in Table 1.

Step 2: The size of the feature dictionary obtained after Step 1 was huge. We took the most informative features by using Boosted Decision trees to learn the training set for the given object and extracted only those templates used in the Boosted trees for splitting.

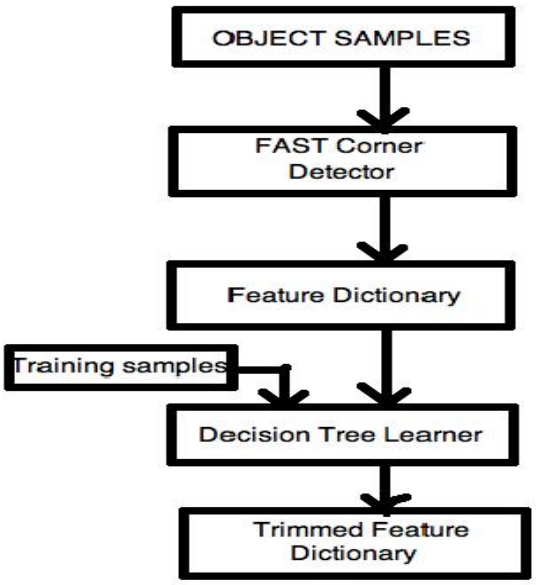

Figure 2: Flow for feature dictionary generation

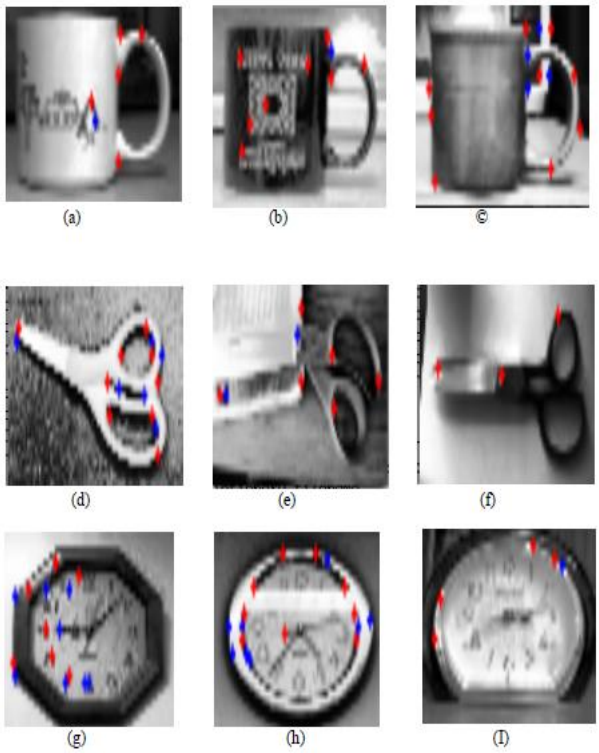

Figure 3: Interest points: (a),(b),(c) represent the corners detected on the samples of 'mug' object, (d),(e),(f) on 'scissors', (g),(h),(i) on 'clock'.

Table 1: Number of templates obtained with varying thresholds on 'mug' dataset using FAST

Threshold	#Templates
80	2594
100	1675
120	950
140	534

Figure 4: Templates generated after Boosting.

Using a Boosted Decision tree learner of depth 3, on the training set for 'mug', we were able to trim down the feature dictionary size to 235. A sample set of the templates generated after employing step 2 are as shown in Figure 4.

We obtained a F1 score of **0.62** using the Template based features on the mug for the "Easy" video (Table 2. But the major drawback of using Template based features was the issue of time and hence we decided to use HOG. The time taken to generate feature vector for the window was high for just a single object and using it for all the images classes was simply infeasible with in the time window allotted.

2.2 Histogram of Oriented Gradients

We used HOG (Histogram of Oriented Gradients) as described in [Dalal 2005] to represent the image rather than the earlier method of max pooling the response image to get the feature values. HOG uses the idea that object appearance and shape can be characterized by the distribution of intensity gradients without knowing the actual gradient positions. HOG is implemented by dividing the image into spatial regions called cells and for each cell a 1-D histogram of gradient orientations over the pixels is calculated and collected to form the image representation. The 1-D histogram of gradient orienta-

Table 2: Comparison of classifiers and features on 'mug' object in "easy" video

Feature	Classifier	F1 score
FAST	Logistic Classifier	0.62
HOG	Logistic Classifier	0.65 (20 bins)
HOG	Boosted Decision Trees	0.845 (20 bins)

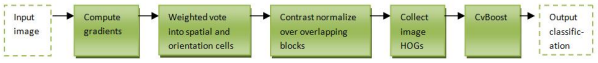

Figure 5: Phases in classification using HOG.

tions is calculated by using a pixel weight assigned based on the orientation of the gradient element centered on it. These histogram entries are then contrast-normalized over larger regions called blocks forming the HOG descriptors. (from [Dalal 2005])

The flow of the algorithm is shown in Figure 5. The steps/phases are described below:

Gradient Computation: We used a gray image and applied the Sobel operator (cvSobel) with an aperture size of 3 as a preprocessing step to normalize the values. Unfortunately due to time constraints, we did not experiment with other masks.

Spatial and Orientation cells: The weight of each pixel was the orientation of the gradient element centered on it. We used a linear gradient voting into 30 orientation bins between 0 to 180 degrees. The other parameters are as follows: Window sizes - 64x64, 32x32, Cell sizes - 8x8, Block sizes - 2x2 (of cells).

After normalizing these histograms over overlapping blocks[1], the obtained values were used as features to represent an image. These were used to train a logistic classifier, boosted decision trees and SVM. The idea of using an SVM was dropped after initial tests because of the difficulty in obtain the confidence value using the openCV implementation of SVM (or libSVM [Chih09]).

Using the logistic classifier, the F1 score obtained was **0.65** which was better than the score obtained using the template based features. The F1 score using boosted decision trees was **0.845** using the same number of orientation bins (20), which was a significant improvement over the earlier versions. Also, the running time was far less using HOG which prompted us to use this as the final feature representation.

The effect of the number of bins was also studied and experimented. As the number of orientation bins increase, more information is captured from a single cell. After trying with 20 and 30 orientation bins, we chose 30 because of its performance on images where the background noise is more.

Optimizations: We used a different window size for elongated objects so that the shrinking effect has a possible match. We moved to this window from the normal 64x64 to maintain the aspect ratio of the elongated objects in the images (3:1 for samples in 'keyboard', 'stapler' and 'scissors' classes). This window was also used in the sliding window algorithm while testing. This provided far more accurate results when compared to using the old 64x64 window for elongated objects. (Note: the window size was 64x64 for the remaining objects - 'mug' and 'clock')

[1]Referred to: http://smsoftdev-solutions.blogspot.com/2009/08/integral-histogram-for-fast-calculation.html

Table 3: Effect of #bins on 'mug' and 'stapler' classes (MD: Motion Detection)

Class	#Bins	#HOG vals	F1	F1(with MD)
Mug	20	3920	0.845	0.877
Mug	30	5880	0.828	0.877
Stapler	20	4480	0.546	0.522
Stapler	30	6720	0.6	0.516

Results showing the effect of bin sizes on the image class 'mug' and 'stapler' are shown in Table 3. (Data: "Easy" video, Classifier: Boosted Decision Trees)

3 Classifiers

3.1 Logistic Classifier

We implemented a logistic classifier using batch gradient update with termination condition of a maximum of 5000 iterations or theta difference of 0.00001. The effect of various features was checked as shown in Table 2.

The major drawback with logistic classifier was the issue of time as the number of iteration taken to obtain effective parameter values was high. Also, the F1 score obtained using logistic classifier was lower when compared to Boosted Decision trees.

3.2 Boosted Decision Trees

A decision tree is a classifier whose internal nodes are tests and whose leaf nodes are categories. A decision tree assigns a class to the input based on the filtering steps at the internal nodes of the tree. Boosting is a class of ensemble learning techniques, in which many "weak" classifiers are combined to produce a more powerful classifier. Decision stumps are often found to be sufficient as weak classifiers.

We have used decision trees with boosting for classification in our project. We have used the cvBoost class in the OpenCV library for generating boosted decision trees[2]. We experimented with depths 1(decision stumps), 3, 5 and above. In general, we got the best results with depths 1, 3 and the accuracy decreased substantially with trees of depths 5. As we increase the depth of the decision trees, overfitting occurs and hence, the higher testset errors. We used both the GENTLE & REAL boosting

[2]www.stanford.edu/class/cs221/handouts/CS221-OpenCVTutorial.ppt

Table 4: Variation of F score with boosting parameters:'mug'

Method	Tree depth	Video	F1 score
Gentle	1	easy	0.83
Gentle	1	vid3	0.061
Gentle	1	training001.xml	0.455
Real	1	easy	0.788
Real	1	vid3	0.031
Real	1	training001	0.381
Gentle	3	easy	0.823
Gentle	3	vid3	0.289
Gentle	3	training001	0.5
Real	3	easy	0.853
Real	3	vid3	0.145
Real	3	training001	0.4
Real	5	easy	0.0485
Real	5	vid3	0.0567
Real	5	training001	0.0571

techniques. In almost all cases we obtained better performance using GENTLE boost. The split criteria used was CV::DEFAULT for GENTLEBOOST and CV::GINI for REAL. The number of weak classifiers used was 100. We experimented with different number of weak classifiers 1, 10, 50, 100,150. We observed that the accuracy increases with the number of weak classifiers but reduces after some optimal number is reached. Moreover, computation time also increases with the increase in the number of weak classifiers. We found the optimum results when the number of weak classifiers was 100.

The results quoted here were run using the HOG features on the dataset. Using the HOG features in the boosted decision trees we tweaked our threshold parameters for each classifier (mug/stapler/keyboard/clock/scissors) so as to obtain good trainset accuracy and then ran the entire system for multi-class classification.

The threshold values for each class is given below:

The experiments conducted on BDTs are as follows: Table 4 for mugs gives the scores as parameters are varied, Table 5 for staplers gives the scores as parameters are varied, Table 6 gives scores for other objects based on optimal parameters, Table 7 shows the effect on F1 by changing the number of weak classifiers.

4 Additional Features

4.1 Different window sizes

We used different window sizes while using the sliding window algorithm for different classes. Some of the

Table 5: Variation of F score with boosting parameters:'stapler'

Method	Tree depth	Video	F1 score
Gentle	1	easy	0.527
Gentle	1	vid3	0.346
Gentle	1	vid0	0.194
Real	1	easy	0.5
Real	1	vid3	0.339
Real	1	vid0	0.286
Gentle	3	easy	0.545
Gentle	3	vid3	0.333
Gentle	3	vid0	0.466
Real	3	easy	0.476
Real	3	vid3	0.279
Real	3	vid0	0.362
Gentle	5	easy	0.545
Gentle	5	vid3	0.4
Gentle	5	vid0	0.459

Table 6: F score for other objects

Class	Method	Tree depth	Video	F1
KeyBoard	Gentle	1	training001	0.041
Scissors	Gentle	1	vid0	0.0769
Scissors	Gentle	1	easy	0.235
Clock	Gentle	1	easy	0.611
Clock	Gentle	1	vid1	0.943

Table 7: Variation of F1 with #weak classifiers (WC)

Method	Tree depth	#WC	F1
Gentle	1	1	0.102
Gentle	1	10	0.386
Gentle	1	50	0.812
Gentle	1	100	0.829
Gentle	1	150	0.776

objects to be recognized were elongated and using the square windows did not prove effective in such cases. Hence we used a separate window size maintaining the resolution ratio of the input samples (i.e. 3:1) in 'keyboard', 'scissors', and 'stapler' classes. The window size considered was 120x40 for the elongated objects.

4.2 Non-maxima suppression

4.2.1 Using overlap and confidence

Due to multiple responses near from object detection, we had a large number of false positives. The technique of non-maxima suppression in our project, detects highly overlapping responses and eliminates those which have low confidence values, thus decreasing the number of false positives. It is a highly effective technique; we observed that for the template based detection our F1 score improved by 0.15 (over milestone submission) by applying non-maxima suppression. During multi-class classification the algorithm proceeds as mentioned above except that we only make comparisons between the same objects.

4.2.2 Using distribution of confidence

Using sliding window technique, we have obtained multiple windows of varying confidences of the object in a neighborhood. Our aim is to return a representative window for all of them. For doing this non-maxima suppression, one technique we have used is to calculate a rectangle which encapsulates those pixels whose distribution is greater than $(mean factor * deviation)$. The 'mean' is the mean of the confidences of all the pixels and 'deviation' is the standard deviation.

The algorithm is as follows:

```
Create an empty confidence_array [WIDTH][HEIGHT]
for each rect in DetectedRectangles
 for x = 0 to rect.width
  for y = 0 to rect.height
   confidence_array[x][y] += rect.confidence

m = mean(confidence_array)
dev = deviation(confidence_array)

lower_threshold = m  factor * dev
upper_threshold = m + factor*dev

min_x = min x such that
confidence_array[x][y] > lower_threshold
and confidence_array[x][y] < upper_threshold, all y
min_y = y such that
confidence_array[x][y] > lower_threshold
and confidence_array[x][y]< upper_threshold, all x

max_x = x such that
confidence_array[x][y] > lower_threshold
and confidence_array[x][y] < upper_threshold, all y
max_y = x such that
confidence_array[x][y] > lower_threshold
and confidence_array[x][y] < upper_threshold, all x

return rectangle bounded by (min_x, min_y)
and (max_x, max_y); confidence is 'm'
```

4.3 Motion detection using filtering

While using the HOG features and Decision trees for detecting 'mug', we ran across the problem of false negative output when the mug's handle just went off the frame. To circumvent the problem, we re-trained the classifier while including the left side parts of the mug to the +ve class for 'mug' objects. But we didn't obtained much gain from it. Hence we wanted to incorporate Motion detection feature.

As we investigated various ways we found a simple filtering algorithm for motion detection at[3].

Citing the algorithm used:

- Smoothen the current and previous frames to reduce noise
- Compare the previous frame to the current frame and obtain the difference of the two images
- Convert the difference image to gray scale
- Obtain the 'difference' image and pass it through a binary filter to get a binary image where the 'white' pixels denote that motion has taken place at that pixel and 'black' means there is no motion detected
- To calculate if motion has taken in an area, then count the no. of white pixels in the area and see if the sum is greater than a threshold

We have used motion detection algorithm in the reverse way, i.e. our heuristic is that if at a given rectangle on the previous frame has not changed much to the current frame, then we are assuming that the object detected in the previous frame is present in the current frame too.

The algorithm we have used is:

- Get the difference image for the two frame
- For each object detected in previous frame
  - a. Calculated the amount of motion taken place in a small rectangle at the object's center between the two frames
  - b. If motion $<$ threshold, then add this object to Queue

[3] http://dev.saulius.me/2009/05/10/movement-detection-using-opencv/

Table 8: Final F1 score on sample videos (MD: Motion Detection)

Video	Mug	Clock	Keyboard	Stapler	Scissors	F1	Mode
Vid0	-	-	-	0.389	0.146	0.283	no MD
Vid0	-	-	-	0.448	0.135	0.279	MD
Vid3	0.03	-	-	0.412	-	0.289	no MD
Vid3	0.06	-	-	0.487	-	0.301	MD
Easy	0.829	0.372	-	0.6	0.25	0.498	no MD
Easy	0.861	0.4	-	0.545	0.21	0.473	MD

- Return Queue

The problem with this approach is that the detected objects linger in the succeeding frames, even though the actual objects has passed away from the frame. To circumvent this problem, we are calculating the confidence of the new object detected from motion detection feature as: $confidence_{new} = confidence_{old} * (1 - motion/threshold_{motion})$

If $confidence_{new} > threshold_{object}$, then add the new object to the Queue else discard the object. We have observed that motion detection does help in detecting objects in some instances, but the amount the gain obtained did not increase much of the high amount of false positives during the detection. (Table 2)

4.4 Combination of classifiers

We take 5 boosted decision tree classifiers one for each class. Given a test frame, we send it to all the classifiers. The prediction of each classifier is thresholded and if it exceeds, the object is added to a common pool of objects with its class label and confidence. After this step, the common pool of objects is subjected to non-maxima suppression, which yields the final set of classified objects.

5 Issues and observations

- Equalize histogram: The cvEqualizeHist method used to normalize and enhance contrast provided bad results.
- Calculating thresholds: Calculating the confidence thresholds for each classifier was sample dependent and was difficult to tune it optimally.
- SVMs: Obtaining confidence measure to eliminate overlapping detections was not possible.

6 Conclusion

The final F1 score obtained on the videos used in last years competition (i.e. vid0 (firsthalf), vid3 and easy) is as shown in Table 8.

References

[Rosten 2007] Faster and better: A machine learning approach to corner detection.
Edward Rosten, Reid Porter and Tom Drummond.
IEEE Trans. Pattern Analysis and Machine Intelligence, 2009.

[Dalal 2005] Histograms of Oriented Gradients for Human Detection.
Navneet Dalal, Bill Triggs.
International Conference on Computer Vision & Pattern Recognition, June 2005.

[Jazayeri] Interest operators in close-range object reconstruction.
I. Jazayeri, C.S. Fraser.
Department of Geomatics, The University of Melbourne, Melbourne, VIC 3010 Australia.

[Gould] STAIR Vision Presentation.
Stephen Gould.
STAIR Lab, Stanford University.

[Chih09] A Practical Guide to Support Vector Classification.
Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin.