CS771A: Machine Learning: Tools and Techniques

Moving object classification for road surveillance

Group 45

Akash Gupta, Atique Firoz, Parikshit Khanna, Vicki Anand

Mentored By: **Dr. Harish Karnick**

Abstract

Detecting and classifying the moving objects in the video sequence is very central task for automating the video surveillance. This can help in significantly reducing the required manpower to monitor such videos. The very fundamental goal of such a system is to identify the objects moving through the foreground of the video frames and classifying them into different vehicle categories and pedestrian. In this project we aim to train and assess some popular fast classifiers to recognize the category of the moving objects. We have trained and tested our models using the videos gathered from surveillance cameras at the gates of IIT Kanpur. We also implement a very simplistic algorithm for foreground extraction with which we port our classifier for testing on new videos.

Introduction

In this project we target to contribute an important component to the video surveillance system at roads of IIT Kanpur. The complete system for video surveillance can be broken down into different stages. Out of which the first and major two stages would be to recognize the moving objects at the foreground of these videos and to classify them into known classes. Depending on the traffic demography at our IITK gate, we decided to use these six classes for classification - **1)Pedestrian 2)Bicycle 3)Motorcycle 4)Rickshaw 5)Autorickshaw and 6)Car.**

Data Collection and Preprocessing

For experimenting with classifiers and training them, we needed the rectangles around the moving objects of the video and labels for them. For this we used the crowd-sourced data created by approximately 200 students of CS771A course at IIT Kanpur.

Problems with data-set

Apart from humane errors like creating mis-fitting bounding rectangles, there were some other major reasons leading to errors in the data set. Errors were because of people either not following the instructions given for the labeling task or misinterpreting the instructions given. One example of this would be - people were not uniform about whether to take the riders of the two-wheelers into the rectangle of the corresponding or not. Also different people had different interpretation for how occluded the objects should for it to be marked occluded.

Data Filtering

We did a two step data filtering. In first step we used a python script to remove some of the unwanted images like very low resolution images and the repetitive images. First step was done alongside the task of extracting objects from the videos. In second step we did a manual filtering. In manual filtering we were only left with the task of removing some of the occluded and wrongly labeled images. Here we describe two types of filtering done in step one.

Removing the Low resolution images

We observed that a lot of wrong tagging was for the images which were of very low resolution. These low resolution images were of those vehicles/pedestrians which were at the farther end from the camera. For removing these images we take advantage of the fact that each of the objects in the labeled videos were given a unique object Id. So for each of the object Ids we make a list of all the frame numbers in which they were visible (i.e. not occluded and not outside the frame). Then we remove some fraction of frames from the beginning and the end of this list. We keep different values for the beginning and the end fraction depending on from which direction larger traffic is coming from.

Removing repetitive images

Using the crowd sourced labels as it is would lead to a lot of repetition of similar images because for each object we would have images from each and every frame. This would result in huge number of same (/very similar) images. To avoid this we again use the objects Ids and depending on the label of the object we select one out of every ‘n’ consecutive frames. The value of ‘n’ depends on the speed of the object. We have divided into three different speed categories.

Speed Category	Classes	Value of ‘n’
Slow	Pedestrian	15
Medium	Bicycle, Rickshaw	10
Fast	Motorcycle, Autorickshaw, Car	5

Table 1: **Value of ‘n’ depending upon speed of object**

Distribution of data

We had crowd-sourced labels available for following eight videos. Out of which five were used for training and 3 for validation and testing.

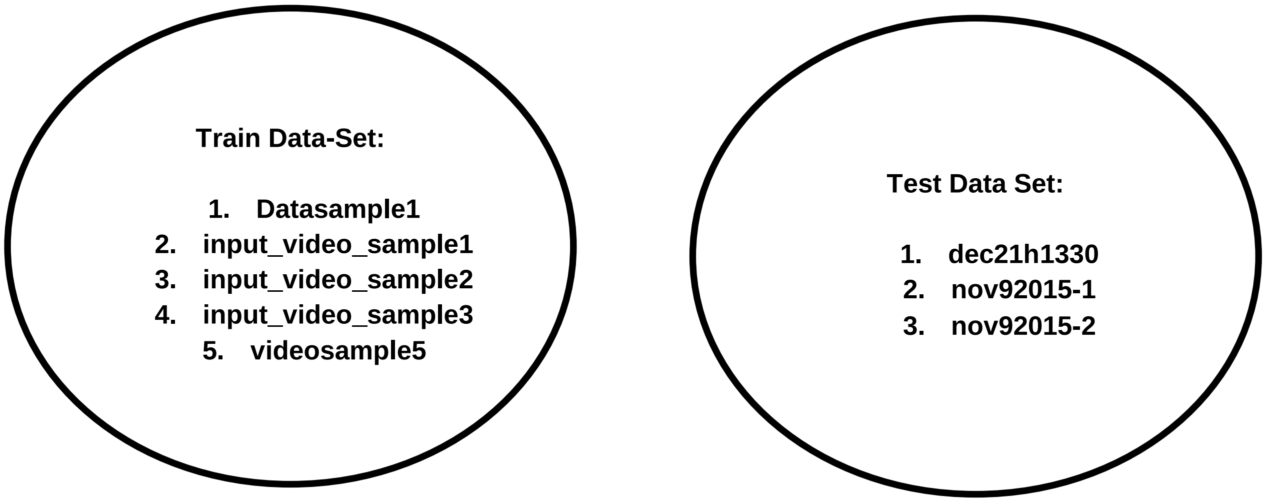


Figure 1: **Distribution Of Data**

Class Label	Training Set	Validation and Test Set
Auto	756	98
Bicycle	798	154
Car	396	92
Motorcycle	812	169
Person	580	116
Rickshaw	564	80
Total	3906	709

Table 2: **Class wise distribution of number of available images after filtering**

Rescaling and Padding

For applying further steps we needed to bring all the foreground object images to a common dimension. We scaled down the images to 30X30 pixels and to 50X50 pixels (grey images). For scaling rectangular images they were padded them with 0 pixel values to make them square. Trying on larger dimension of 50X50 didn't give any improvement in result. Similarly color images of 30X30X3 dimension also didn't show any significant improvement. So all the further stated results are derived using 900 sized vectors to which we reduced all the available images.

Feature Extraction using HOG (Histogram of Oriented Gradients)

HOG (Histogram of Oriented Gradients) are feature detectors used in computer vision for efficient object detection. The technique revolves around considering occurrences of gradient orientation in an image, locally. HoG (Histogram of Oriented Gradients) are feature detectors used in computer vision for efficient object detection. The technique revolves around considering occurrences of gradient orientation in an image, locally. The method is developed by ***Navneed Dalal*** & ***Bill Triggs***. The method relies on intensity gradients and edge directions.

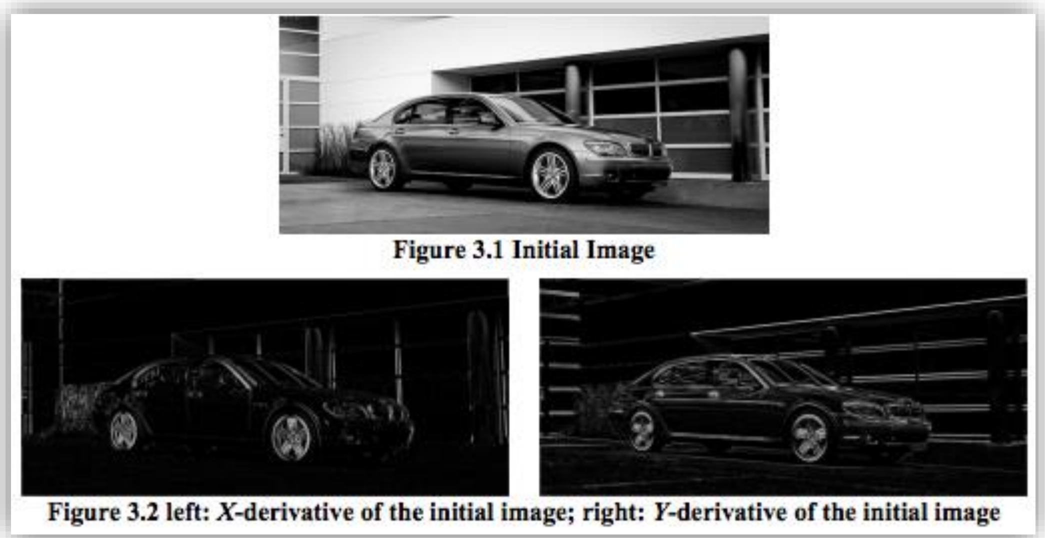
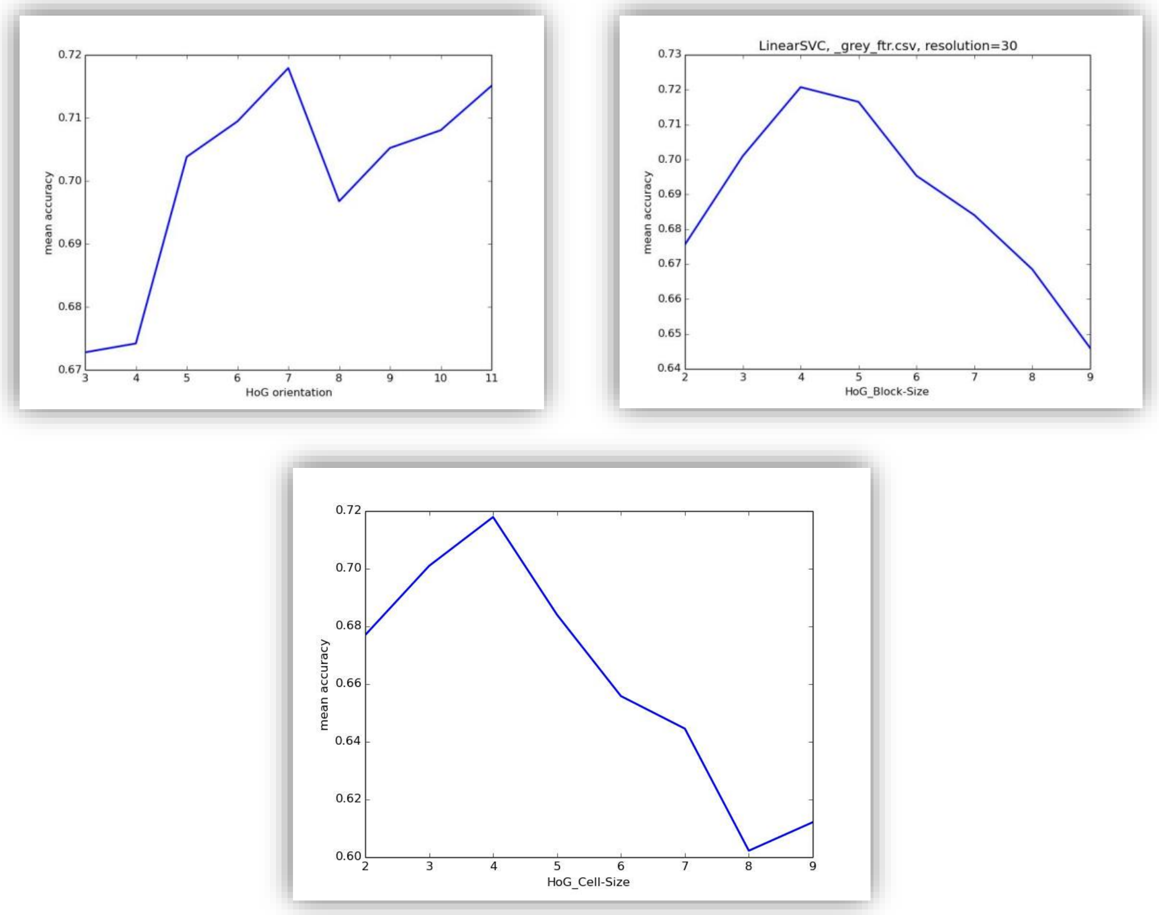


Figure 3.1 Initial Image
Figure 3.2 left: *X*-derivative of the initial image; right: *Y*-derivative of the initial image

Figure 2: **X & Y-derivatives of on object in an image***
*(Retrieved from http://users.utcluj.ro/~raluca/prs/prs_lab_05e.pdf)

Figure 3: **Graphs for HOG Parameter Tuning for Linear SVC**



Best Result came for orientation = 7, Pixels per cell = 4 and Cell per block = 4.

PCA for dimension reduction

Principal Component Analysis (PCA) is a statistical procedure that performs an orthogonal transformation on the data to attain maximum variance for a given number of components.
In our case employing PCA led to significant reduction in accuracy. We believe that the reasons for such reductions are:

1. Proportion of variance explained is low for higher components, i.e variance is uniformly distributed across all the components
2. Due to different dimensions, a significant degree of padding was involved which reduced the rank leading to fewer eigen-values.

Hence we didn’t apply PCA to our vectors post HOG transformation.

Classification

We tried six different classifiers. By varying their parameters we tried to obtain best results for each of them.

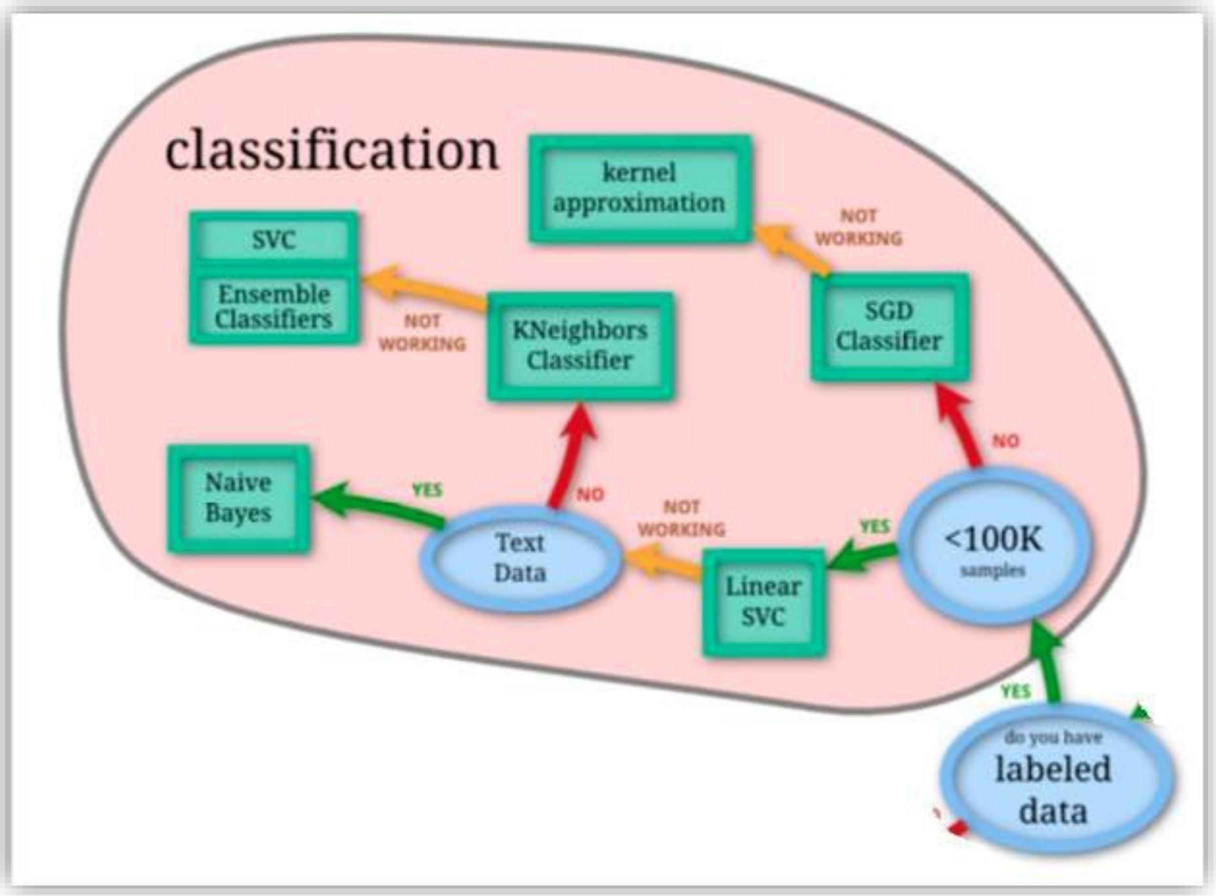


Figure 4: **Schematic Diagram of Classifier Road Map***
*(Image retrieved from http://scikit-learn.org/)

Below we describe how each of them performed:

Linear SVC

The Linear SVC package of the *sklearn* library was used. Linear SVC is a special case of the SVC library which uses the linear instance of SVC (Support Vector Classifier). It is interesting to note that the multiclass classification is handled by Linear SVC using the **One-vs-Rest-Scheme (OVR)**, which we worked upon in the class assignments.

K-Nearest-Neighbours

K-nearest-neighbours (KNN) is a non-parametric model, i.e., not dependent on the probability distribution of the learning set. The assigned label is decided by majority voting amongst the k-neighbours of the point. The *KNeighboursClassifier* Library of the *sklearn* kit was used. Variation of accuracy with K = number of neighbours was observed and plotted.

Decision Tree Classifier

Decision Tree Classifier is a model which predicts the value of the label based on several input parameters.The metric used to choose splitting attribute and value is based on impurity reduction.In the Decision Tree Classifier instant used in our project the criteria is set to ‘Gini’ implying the Gini index is usd for impurity calculation.

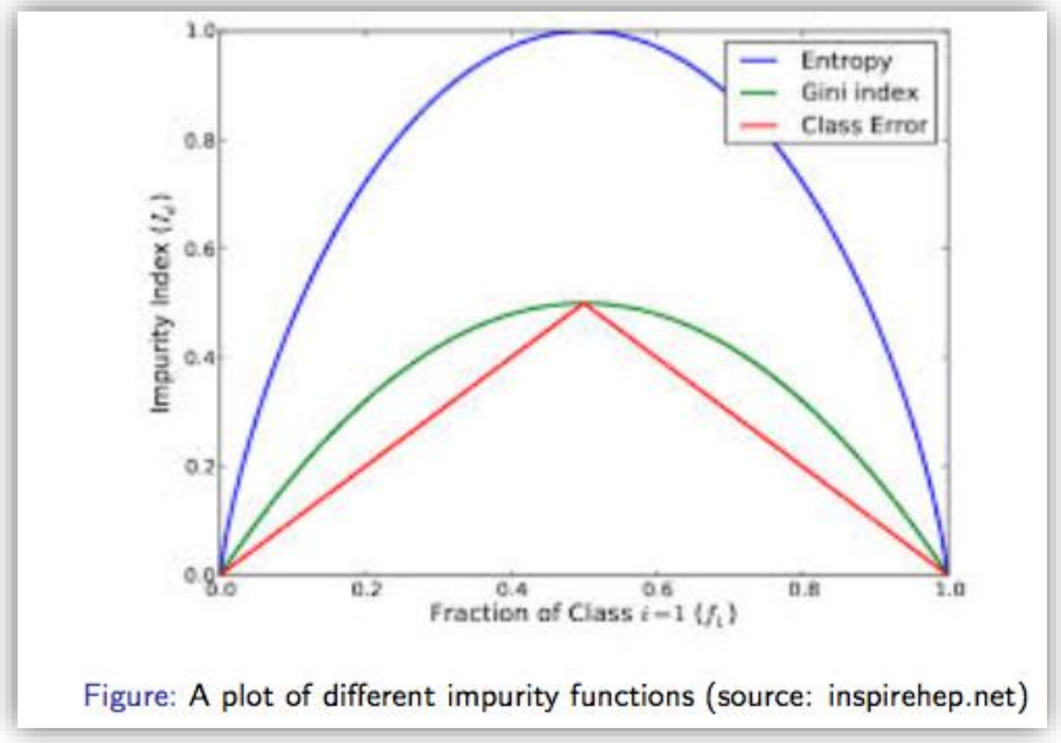


Figure 5: **A plot of different impurity functions** (source: inspirehep.net)

We compared the accuracy of the Decision Tree with the max depth parameter where we modeled unrestrained depth by mapping Max_depth = None to max depth = 1000

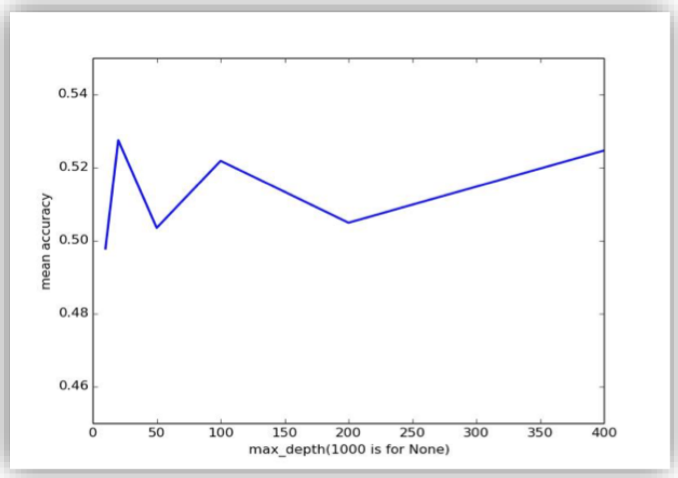


Figure 6: **Mean Accuracy vs Max_Depth**

And observe that the accuracy increases with deeper trees (as expected) .

Random Forest

Random Forest is basically an ensemble method built using weak tree classifiers (stumps).We used the *RandomForestClassifier* of the *Sklearn*.*Ensemble* library. In this case we compared the accuracy variation with respect to number of trees used in the forest and witnessed a largely asymptotically increasing accuracy with respect to the number of tree.

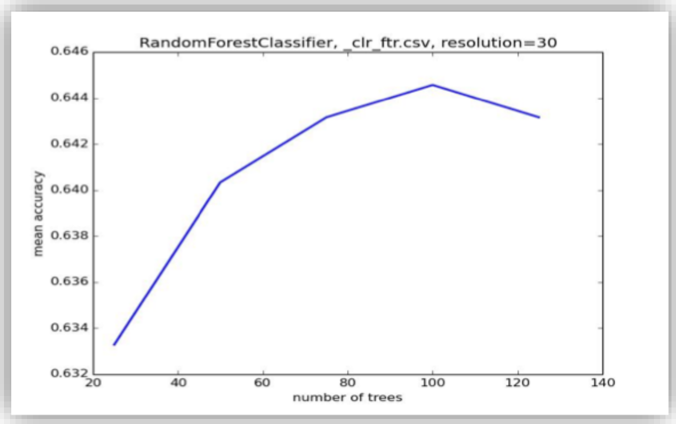


Figure 7: Mean Accuracy vs Number Of Trees

Adaboost (Adaptive Boosting)

The AdaBoost algorithm is a variation of the ensemble method with the added capability of tweaking subsequent models in favour of previously misclassified points .We observed a significant drop in the accuracy with AdaBoost as compared to Random Forests. We believe the reason for this was the high sensitivity of the method to noise. Since the data was manually sorted (to a very high extent) *a suitable degree of randomness was not injected in the learning set which rendered the model weak.*

Results

As it was expected we got the best result for SVC and Random Forest (Ensemble) classifiers. For both of these we got approximately 72% accuracy.

Table 3: **Accuracies of various models**

Model	HoG	Without HoG
K-NN	61.9%	62.5%
SVM	71.79%	61.8%
Decision Tree	52.60%	52.75%
Random Forest	71.6%	64.5%
AdaBoost	50.02%	44.87%

Table 4: The confusion matrix obtained with Linear SVC :

	Person	**Bicycle**	**Motorcycle**	**Rickshaw**	**Auto**	**Car**
Person	**79**	32	4	1	0	0
Bicycle	2	**84**	40	17	6	5
Motorcycle	0	15	**106**	19	22	7
Rickshaw	0	4	6	**65**	1	4
Auto	0	2	13	4	**74**	5
Car	0	1	5	0	0	**86**

Confusion Matrix Discussion:
We observed the individual entity accuracy i.e.

Entity	**True Identification**
True Person	68.1 %
True Bicycle	54.4 %
True Motorcycle	62.7 %
True Rickshaw	81.2%
True Auto	75.5%
True Car	93.47 %

We observe that car was identified with the highest accuracy. A probable rationale behind this was that it didn’t share a category with any other identity (unlike 2-tyre vehicle or 3-tyre vehicles). Bicyle was observed to have been identified with the maximum error and most often it was misclassified as Motorcycle (57 % error attributed to the latter), as expected. However the fact that the same phenomena was not observed with the Motorcycle was a bit surprising to us. We hence note that despite bare minimum preprocessing and a not so rigorous crowd labelling, the results are quite satisfactory, malfunctioning only on similar objects, indicating a huge scope for improvement with a more refined labelling.

Foreground Identification & Classification

Now our goal was to test our learnt classifier to classify all the objects in a random given video stream into the above mentioned six classes. For that we used foreground separation method to detect the objects and then classified the detected object to show the result. The steps included are mentioned below:

Frame Differencing:

The underlying assumption for the working of foreground separation is that, our video stream is largely static and unchanging over consecutive frames in the video. We tried to exploit this fact for modeling our separation method. While modeling the foreground, we monitored each frames for substantial change with respect to the local frames. If there was substantial change detected in a region or multiple regions, it was detected. This change basically correspond to the motion in our video stream.

We used the last local frame as reference for the detection of the foreground, and applied the frame differencing method first between the current frame and last stored frame to get a delta (difference) map for the foreground objects. In this we experimented with completely assigning our first frame as the reference frame as video recorder is static in nature. We also tried with setting the reference frame to the 2nd last, 3rd last, etc. Among which the last frame gave the best results.

Motion Detection:

To test the classifier we have learnt on the visual data of the security footage and to classify them in our six classes, first we needed to identify all the moving objects from the video in each frame to apply on our learnt classifiers. Various steps included are enlisted below:

Gaussian Blur

We used gaussian blur method to counter for the various factors affecting the video recording. Due to the open environment, even background (non-moving part) of consecutive frames of the video were affected by camera sensors, lighting conditions and many more other factors.

To account for this and reduce the effect of it by applying Gaussian blurring to average the pixel intensities across the frame as shown in the following figure

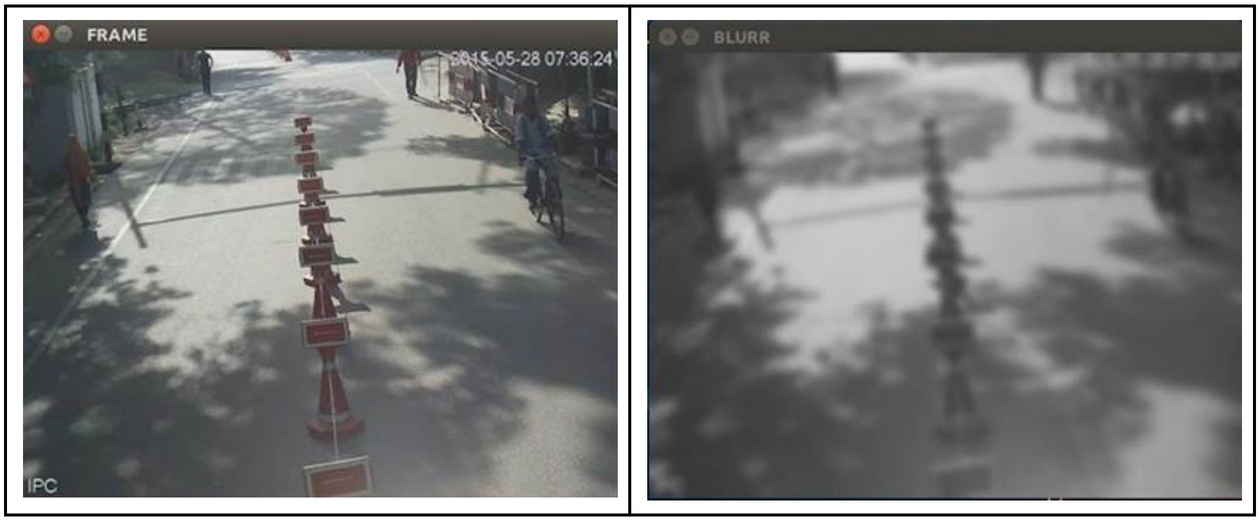


Figure 8: **Video Frame vs Gaussian Blur**

Binary Conversion & Contour Detection

Using the frame differencing on the gaussian blurred current and last frame we got a difference matrix of the frame with difference at locations corresponding to the motion location in the frame. We used thresholding and dilation method to completely convert the frame into binary image (white and black regions). With white corresponding to the object in the frame and black to the background respectively as shown in the following figure:

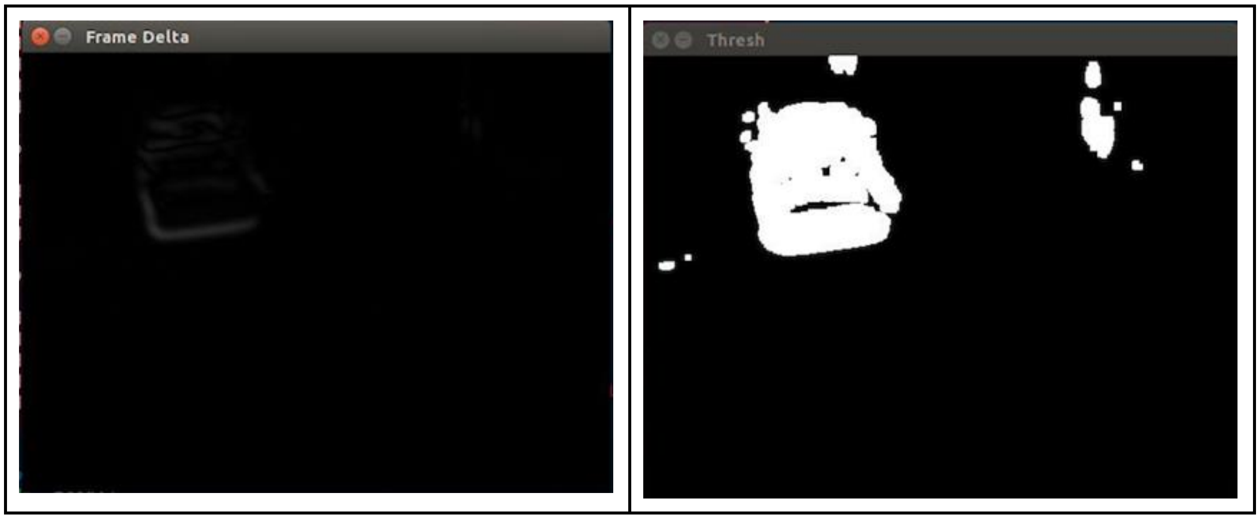
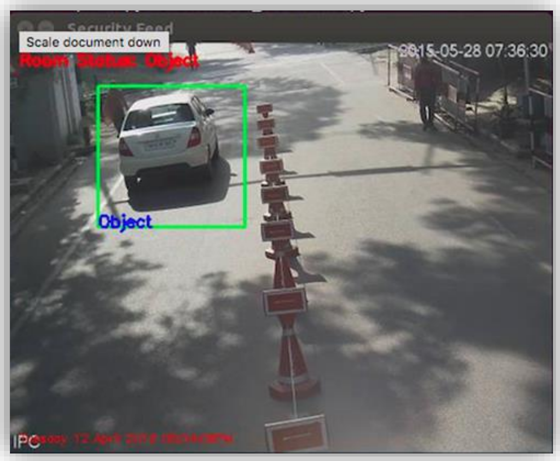


Figure 9: **Frame differencing and Binary image**

Then we applied contour detection method to get the contour around all the white regions in the frame. We applied thresholding on the min area required, to be a valid object and to handle smaller contours. As a result we got the x,y,w,h (x,y,width,height) locations of the valid objects as shown in the following figure:



.

Figure 10 : Contour detection in frame

Porting With Classifier

Once the moving objects in the foreground have been recognized and bound by the rectangles, the pixel values from that rectangle is treated as a test image and passed to the classifier. For porting the classifiers we save the learnt models as pickle files. Saving them to pickle files help us save the time required for training the model, which otherwise we had to do again and again each time program was run on a new video.

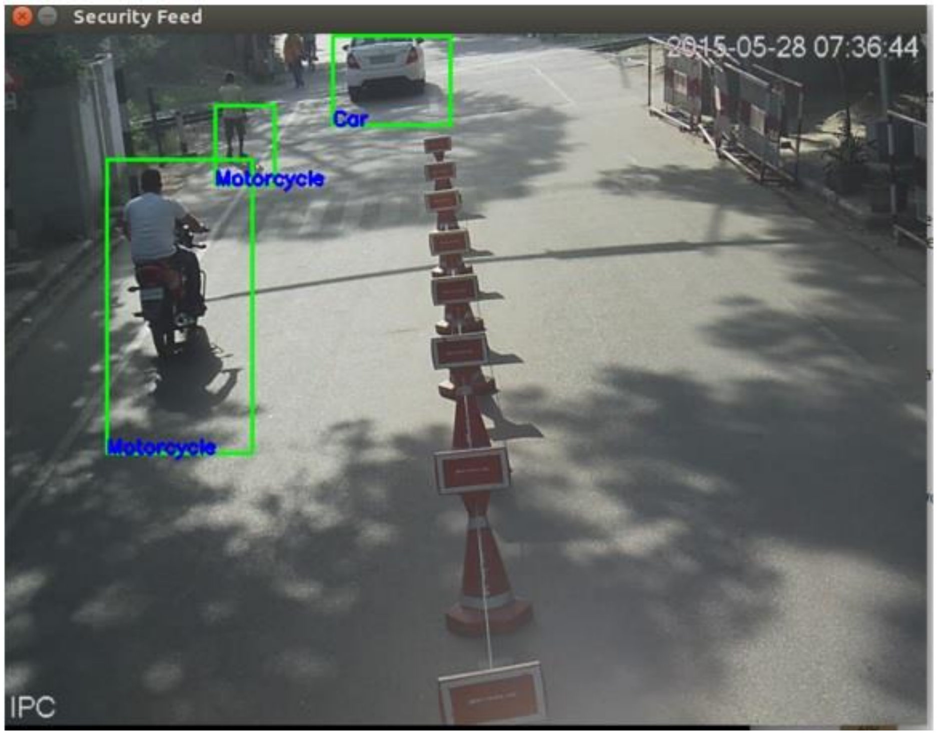


Figure 11: Final Classification

Further improvements

The results obtained with the classifiers can be further improved by creating a better and larger image dataset for training the models. There are issues with the crowd-sourced data that we used for our project. For instance there is no consistency regarding selecting/discarding the riders on two-wheelers.
Also with the improvements in the algorithm used by us for foreground detection, the results obtained with the final ported program would be much better. Currently as we can see in the images above, even the shadow of the moving objects are detected inside in the bounding rectangles. However in our learning images shadows were not a part, so fixing this would improve the accuracy.

References:

1. **Duda.R, Hart.P, Stork.D** (*2001): Pattern Classification, Wiley, U.S.A.*

2. **”Scikit-Learn”,** retrieved from: http://scikit-learn.org/stable/.

3. **N.Dalal, B.Triggs**, IEEE *2005*: *”Histogram of Oriented gradients for human detection”*, retrieved from: http://users.utcluj.ro/~raluca/prs/prs_lab_05e.pdf.

4. *”Basic motion detection and tracking with Python and OpenCV”* retrieved from: www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv.

5. **Qi Zang and Reinhard Klette**, *The University of Auckland*: *Object Classification and Tracking in Video Surveillance*, retrieved from: http://citr.auckland.ac.nz/techreports/2003/CITR-TR-128.pdf.