Seminar: Technology of Autonomous Vehicles

Topic 2:Camera-Based Vehicle Tracking

Rakesh Reddy Kondeti, Neelam Sattur

Contents

Abstract

This paper addresses salient points of camera-based vehicle tracking by using Histogram of Oriented Gradients (HOG) for feature extraction and linear Support Vector Machine (SVM) for classification. The influence of different colour spaces on prediction accuracy and the overall system performance improvement using the YOLOv3 (You Only Look Once - version 3) network are included. A comparison between the performance of the YOLO network and the linear SVM classifier is shown. An attempt has been made to present ideas to improve the system further is also included in this paper.

1 Introduction

In recent years, much interest has been seen worldwide in Autonomous driving systems. It is expected to be the next big thing in the Automotive industry. One of the challenges in developing intelligent systems to drive autonomous systems is its ability to detect and appropriately react to objects on the road and unpredictable behaviour of other vehicles and pedestrians [1]. Control systems

can analyze data from a variety of sensors, including from a dashboard camera and proximity sensors, to detect objects on the road and select the best navigation trajectory. If necessary, the vehicle should precisely identify the presence of an object and come to stop at a safe distance in order to avoid a collision. While this is a simple and intuitive task for humans, it is a problem to automate detection based on video data obtained from a vehicle's dashboard camera [2].
This study aims to assess the applicability of HOG and SVM for vehicle detection, study their influence on the system's performance, and arrive at optimal values that ensure positive results.
This paper also discusses vehicle tracking using the deep learning approach (YOLO). Finally, the performance of HOG based SVM is compared with the performance of YOLO, and the results are discussed. The future scope illustrates the possibilities of developing the vehicle tracking systems based on the results obtained during the experiments discussed in this paper.

2 Background Knowledge

2.1 Feature Extraction using HOG

Histogram of Oriented Gradients

The feature descriptor extracts useful and relevant information for detecting an object and eliminating unnecessary information such as colour and noise [1]. The core principle behind HOG is that the distribution of local intensity gradients or edge directions can be used to identify the appearance and shape of a local object without relying on all the object details [3]. The histogram of oriented gradients techniques is widely employed in computer vision and image processing for object detection. Using this method, only the count of gradient orientation of pixels in localized portions of the image is utilized to evaluate the image and object recognition. The feature extraction is done in three major steps:

1. A histogram of gradient magnitudes and gradient directions for the pixels in each cell.
2. The gradient magnitude values of these cells are binned according to their gradient orientations and put together into nine bins (0° to 180°) [4].
3. Block normalization is performed on the histograms to improve accuracy.
4. All the local histograms are then concatenated to form the HOG features vector.

Calculation of Gradient magnitudes and gradient directions:

Let $I(x, y)$ be the intensity of an image. Then I_x and I_y are the partial derivatives of the image intensity $I(x, y)$ and are calculated as:

$$I_x(r, c) = I(r, c + 1) - I(r, c - 1) \tag{1}$$

$$I_y(r, c) = I(r - 1, c) - I(r + 1, c), \tag{2}$$

where (r, c) denotes the pixel coordinates.
The magnitude $\mu = \sqrt{I_x^2 + I_y^2}$ and the orientation $\theta = \frac{180}{\pi}(tan_2)^{-1(I_y, I_x) mod \pi}$, where tan_2^{-1} is the four-quadrant inverse tangent, which yields values between $-\pi$ and π, are calculated for each gradient [5].

Cell Orientation Histogram:

The HOG features are extracted as *cells* of the whole frame of an image. Adjacent and non-overlapping regions of the image of size $C \times C$ ($C = 8$ pixels) is called a cell. The local histogram for each cell is computed by binning each gradient orientation in one of the 9 bins. The 9 bins of the histogram represents the angles $0, 20, 40, \ldots 160$, with each bin having a width of 20°. If the calculated

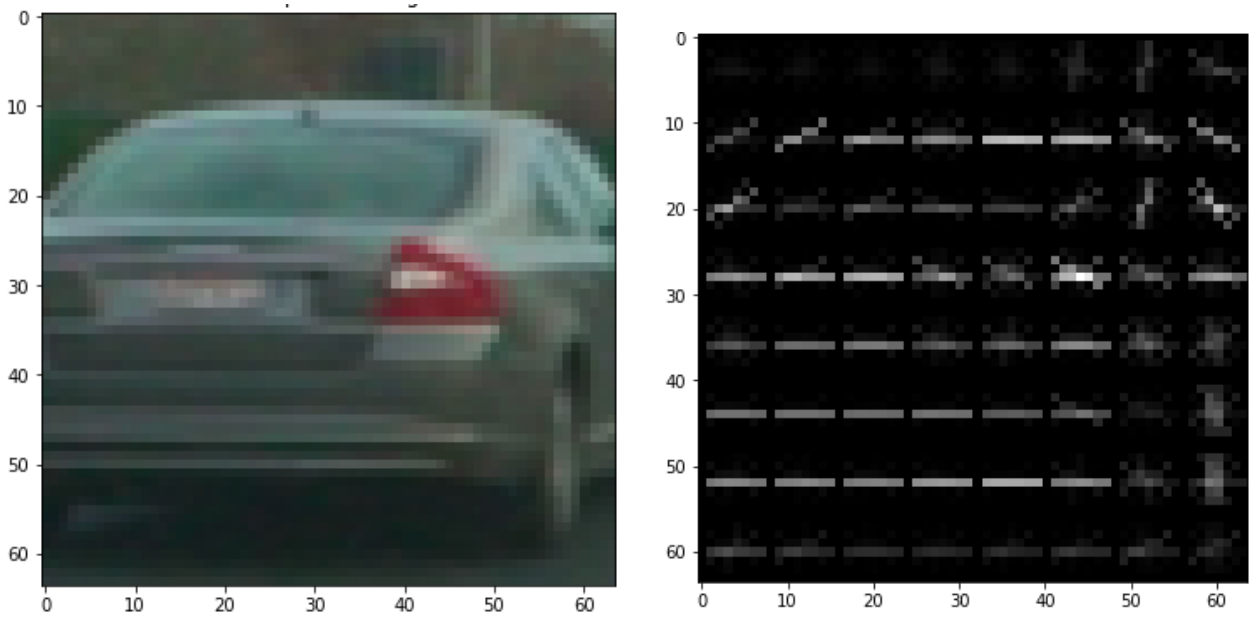


Figure 1: Car and the corresponding HOG features obtained from our results.

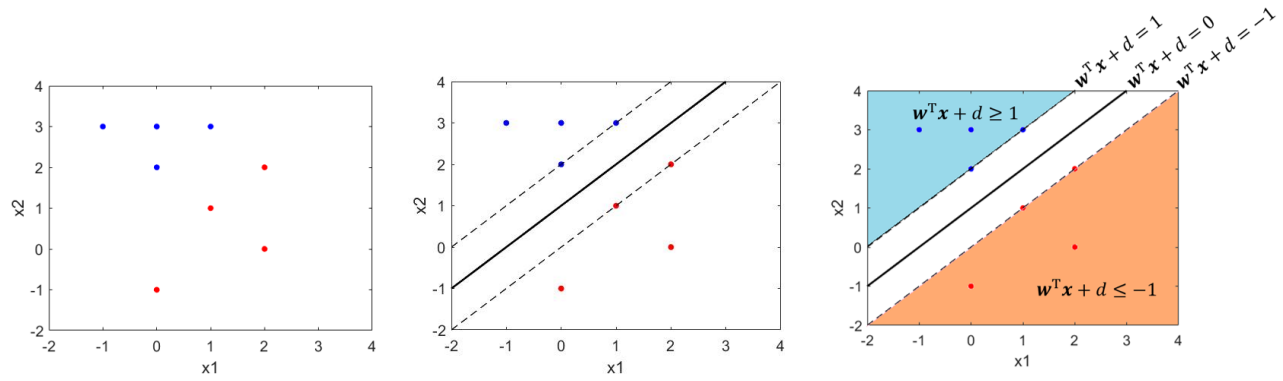
gradient orientation does not correspond to any of the angles in the bins, then the binning process is performed by calculating the *weighted vote* v.
A pixel with gradient magnitude μ and gradient orientation θ contributes a weighted vote $v_j = \mu\frac{c_{j+1}-\theta}{20}$ to bin number $j = \lfloor \frac{\theta}{20} - \frac{1}{2} \rfloor$ [5]. All the pixels in the $C \times C$ cells contributes and are added up to create the 9-bin histogram.

Block Normalization

A histogram in the HOG feature descriptor is a distribution of gradient magnitudes based on their gradient orientations. But for improved accuracy, all the local histograms (histogram within each cell) are block normalized by choosing a larger region of the image (more than one cell), called a *block*. Then, all the cells within the block are then normalized by dividing the gradient magnitude μ for each pixel by overall magnitude within that block.
Block normalization is performed by grouping the cells into overlapping blocks. The general choice of each block is a size of $2C \times 2C$ pixels, and the block stride is C pixels (the block will move one cell at a time horizontally or vertically). In block normalization, each cell is considered four times during the normalization process. If b is the vector representing all the gradient magnitudes in a block, then the block normalization is performed by dividing the vector b by its Euclidean norm: $b \leftarrow \frac{b}{\sqrt{\|b^2\|+\epsilon}}$, where the small positive constant ϵ is used to avoid division by zero in gradient-less blocks [5].
However, block normalization is an optional step in HOG feature descriptor but is recommended to improve the model performance which results in better invariance to changes in illumination [6].
The resultant HOG feature is obtained by concatenating all the block normalized histograms.

2.2 Support Vector Machine

Support Vector Machine (SVM) is one of the supervised learning methods used for classification and regression. The hyperplane obtained through the SVM method achieves an optimal separation with the most significant distance to the nearest training data point (margin)[2]. In general, if the margin of the classifier is larger, the classifier is less sensitive to noise in the data and hence lower the generalization error of the classifier.



(a) Blue points represents positive samples & red represents negative

(b) Support vectors(on dotted lines) & maximum margin hyperplane (solid lines)

(c) Equations of max.margin hyperplane and inequalities showing either sides of hyperplane

Figure 2: Support Vector Machine

The SVM classifier is often used in computer vision problems and is considered as one of the powerful classifiers. In this project, SVM is used for vehicle detection (cars) after the extraction of HOG features. For this purpose, a training dataset is created consisting of *HOG features of vehicle images* and *HOG features of non-vehicle images*. The SVM classifier is trained on these features to learn the weights.

The general representation of the maximum margin hyperplane is denoted by the equation:

$$H : \mathbf{w^T x} + d = 0, \tag{3}$$

where **w** is the weight vector and d is the bias. The optimal hyperplane is obtained by using scaling **w** vector and d. From the studies, the most efficient way of scaling to obtain optimum hyperplane is to represent in *canonical form*:

$$|\mathbf{w^T x} + d| = 1 \tag{4}$$

When written in canonical representation, the term **x** represents the closest training data to the hyperplane, which are called as *Support Vectors.*

The distance between any training data point **x** and the hyperplane H(**w**,d) is :

$$distance = \frac{|\mathbf{w^T x} + d|}{||w||} \tag{5}$$

considering the hyperplane equation is in canonical representation:

$$distance_{supportVectors} = \frac{|\mathbf{w^T x} + d|}{||w||} = \frac{1}{||w||} \tag{6}$$

From the above the equation, the distance or margin from the support vectors to separating hyperplane is $\frac{1}{||w||}$. So, to give the better maximum separating hyperplane, one has to maximize the margin ($\frac{1}{||w||}$). From quadratic programming, maximizing $\frac{1}{||w||}$ is equivalent to minimizing the function:

$$\text{Minimize } f(\mathbf{w}) = \frac{1}{2}\mathbf{w^T w} = \frac{1}{2}||w||^2 \tag{7}$$

$$\text{subject to } y_i(\mathbf{w}.\mathbf{x_i} + d) \geq 1 \quad \forall i$$

y_i = 1, if x_i belongs to positive class (car images HOG features)

y_i = -1, if x_i belongs to negative class (not car images HOG features)

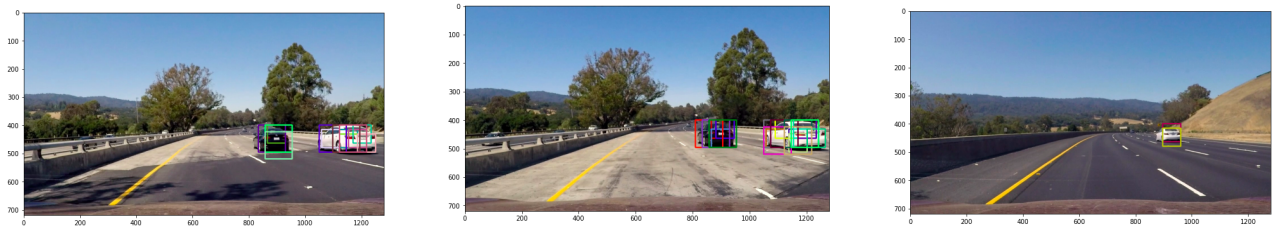


Figure 3: Sliding Windows

Using the in-built functions available in OpenCV like *setSVMDetector*, the SVM classifier is trained on the HOG features to classify the vehicles in the images. During the training phase, the classifier learns the weight vector **w**. The learnt weight vector is then used in the production phase to classify the HOG features accordingly.

2.3 Sliding windows

A sliding window refers to a rectangular region of fixed dimensions that ”slides” across an image in object detection task[3]. A sub-region of the image frame is isolated and then, a classifier is applied to determine if the window has a vehicle or not. Since extracting HOG features is extremely time-consuming and getting the features for each window may overlap. The HOG features are extracted as *windows* of the whole frame at the beginning.

Then, the features are pulled out from the window region as needed. The object detector then slides the window over every image of the pyramid in increments of a few pixels. This is called the stride of the detector. It computes a feature at every position as a vector of numbers that describe the window's contents. These features are then fed to the classifier, which can detect the presence of an object [5]. The window can be of various sizes and aspect ratios. Therefore, it can move in small or longer strides so that the classifier will output higher scores for some classes for some objects. By using this process, the classifier can detect objects of various sizes and locations. However, this technique is not very effective as it is computationally intensive [7].

2.4 Heatmap

A heatmap is used to avoid the overlapping bounding boxes and false positives in the prediction of the SVM classifier[4]. Heatmap is a matrix with values from 0.0 to 1.0. The peaks on the map indicate the presence of an object. Based on the positioning of these peaks, resulting bounding boxes are predicted. To build a heat map, first, start with a blank grid and ”**add heat**” for all pixels within windows where the classifier reports **positive detections**. The ”**hotter**” the parts, the more likely it is a true positive, and we can impose a threshold to reject areas affected by the false positives. Suppose we integrate a heat map over several consecutive frames of the video. In this case, areas with multiple detections get ”**hot**” while transient false positives stay **”cool”**. [8].

2.5 Colour Spaces

Colour spaces are different types of colour modes used for various purposes in image processing. They can be used to estimate the percentage of misclassified samples by using colour-based clustering of the pixels [9]. Colour spaces are also used for shadow suppression based on colour manipulation, which increases the system's performance in object detection, and tracking [10].
The commonly used colour spaces are *RGB, Y'UV, Y'CbCr, HSV, HLV and LUV.*

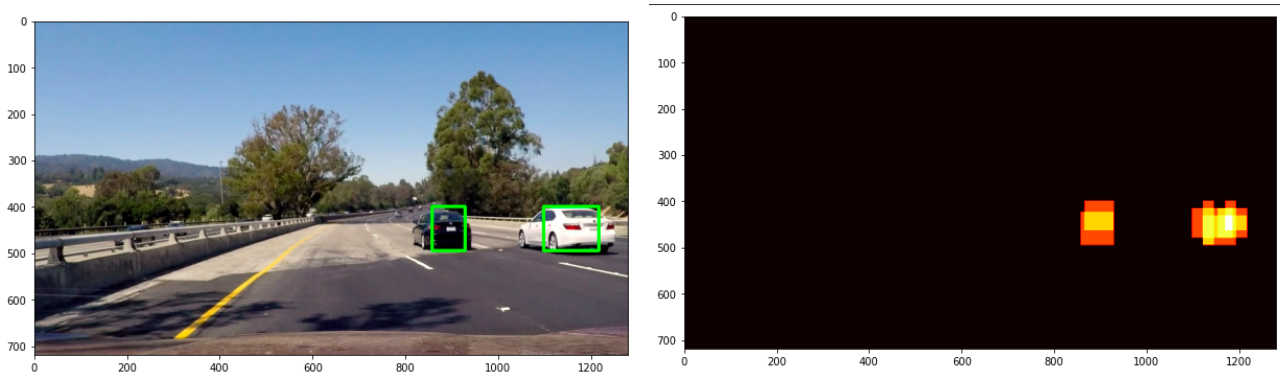


Figure 4: Detected cars and corresponding heatmap

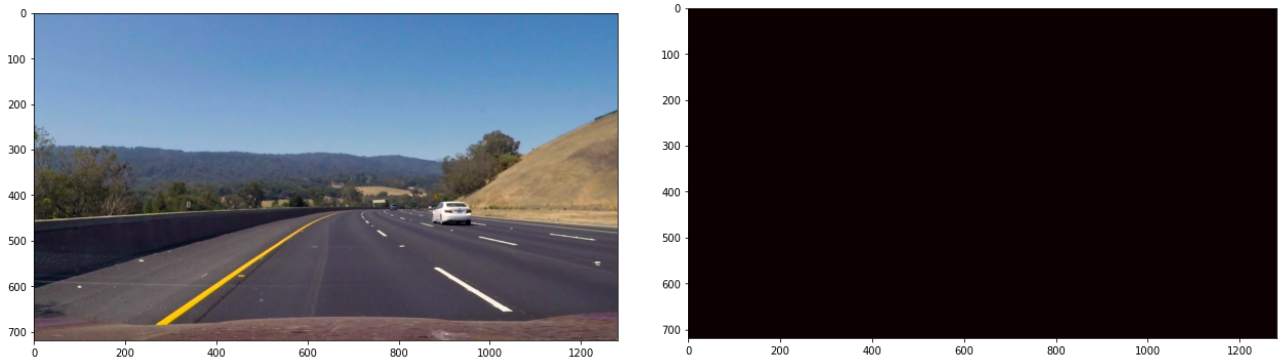


Figure 5: Image with no detection and corresponding heatmap

3 Methodology

3.1 Data Preprocessing

The dataset for this project is taken from the KITTI vision benchmark suite [11]. The data are the colour images with dimensions $64 \times 64 \times 3$, which are used for computing the HOG descriptors of the images. The labelled data contain *8792* vehicle images and *8968* non-vehicle images.
In the prepossessing stage, the primary data augmentation like transformation, translating, rotating, and image scaling is done to increase the number of images, thereby enhancing the prediction's accuracy. All the training images have the exact dimensions. Hence, their HOG feature produces the same vector length, and they are used to train the SVM classifier. Also, the images are converted into various colour spaces like RGB, HLS, YUV, YCrCb and LUV to increase the detector's performance. In this project, YUV colour space and YCrCb colour space produce minor errors compared to all the other colour spaces. So, YUV colour space is considered throughout this project and to perform further steps.

3.2 Feature Extraction

All the pre-processed images are used for feature extraction. They are performed by computing the HOG descriptors of every pre-processed image from the dataset. These descriptors are used to train and test a linear SVM.
For computing the HOG features, we used the direct function *hog()*, imported from the *scikit-image* library, which is a widely used library for image processing in python. The *hog* function takes the *image, orientations, pixels_per_cell, cells_per_block, visualize* and *multichannel* as the input arguments. All the images are passed into *hog()* function one by one with the parameters as *orientataions* = 9

bins (as given in the original research paper [3]), 8×8 is chosen as pixels per cell which determines the size of the cell, 2×2 is chosen as cells per block which is used for block normalization. The parameter *visualize* is a boolean variable which returns the HOG image and it is set to *True*. The extracted HOG features of the vehicle images and not vehicle images are passed to train the linear SVM.

3.3 Training a Linear SVM

The HOG features of both the vehicle and non-vehicle images are used to train the Linear SVM classifier. All the vehicle and non-vehicle features are stored in a list. All the labels are stored in another List, and labels are 1 or 0 according to the vehicle or non-vehicle images. Training the SVM classifier is performed by importing the *LinearSVC* class from the third-party library *sklearn*. The shuffled feature dataset is split into 80% training data and 20% test data using the *train_test_split* function from the *sklearn* library.
Different colour spaces like RGB, YUV, HLS, HSV, LUV and YCrCb are considered for the training of linear SVM. However, YUV and YCrCb colour spaces are observed to produce the lowest error on the test dataset. So, the colour space YUV is used in this project.

3.4 Vehicle Detection

The test video used for the detection purpose is from the KITTI vision suite [11]. This video is the actual dash-cam data. Each frame is extracted from the video, and the sliding window is made to slide on the middle part of the HOG image. Then this part is passed into the SVM classifier, which classifies it as the vehicle or non-vehicle features. The detector produces multiple bounding boxes around the object due to the sliding window approach. Therefore, each object will appear multiple times in the overlapping sliding windows. So, to keep only one bounding box, the heatmap is applied by discarding all the other bounding boxes based upon some threshold.

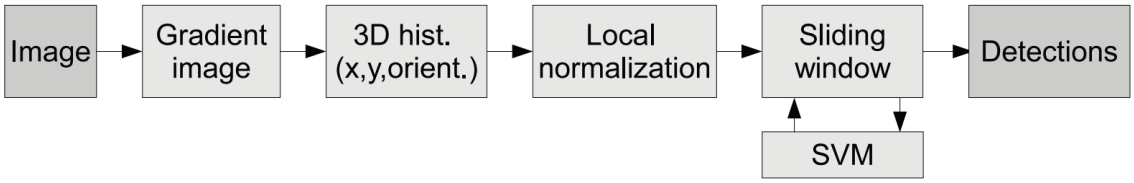


Figure 6: Overview of HOG algorithm [12]

4 Deep Learning-based Vehicle Detection

This section talks about the state-of-the-art technology for object detection, which has always been one of the most interesting problems in the deep learning field. The deep learning method is discussed as an alternative approach to the HOG based SVM in camera-based vehicle tracking. Primarily convolutional neural networks are used for performing object detections.

We have implemented the YOLO network (version 3) for vehicle tracking. YOLO architecture [7] is based on CNN using anchor boxes. Currently, YOLO is one of the state-of-the-art object detection techniques already employed in a wide range of real-life problems, such as vehicle detection and pedestrian detection on the roads. Similar to other deep learning architectures, YOLO uses a convolutional neural network to detect objects. It treats the object detection problem as a simple regression problem to find bounding boxes. Each of these bounding boxes is weighted by the class probabilities

of the objects detected [13].
The YOLO network takes an image and a label as inputs. In this study, a car/vehicle is considered as the only single class to be detected. After passing the inputs to the YOLO network, the image is divided into $S \times S$ grid. The grid for each object is assigned according to the centre of the object. During the training phase, the network learns the weights and utilizes these weights to detect the bounding boxes of the detected objects.
Since YOLO uses only convolutional layers to detect objects, it is a fully convolutional network (FCN). The input image ($n \times n$) is passed only once through the network, which results in an output prediction image of $m \times m$.

4.1 Architecture

One of the most popular versions of the YOLO approach is the third version, called YOLOv3. This version uses multi-scale prediction and bounding box prediction by considering it as a regression problem.

YOLOv3 uses the darknet-53 framework. This framework consists of 53 convolutional layers and hence the name *Darknet-53*. Darknet-53 is just an improvement of the previous Darknet-19 framework, which was used in YOLOv2 network, by adding skip connections and residual blocks. One of the main advantages of YOLOv3 is its ability to predict small objects, which was missing in the previous YOLO versions. The residual and skip connections enhanced the network ability to capture smaller features.

However for the detection tasks, another 53 convolutional layers are added to the darknet-53 framework, by making it as the 106-layered Fully Convolutional Neural Network. YOLOv3 predicts the objects at the three different layers: at 82 layer, 94 layer and 106 layer. At each of these layers, the original image is downscaled by a stride of 32, 16 and 8 respectively. The 82 layer is used for predicting the big objects, 94 layer is used for medium-size objects and 106 layer is responsible for detecting the smaller objects. Unlike the previous YOLO versions, 1×1 convolutional filter is applied on the feature maps to predict the class probabilities instead of using softmax operations [14].

The other important point of YOLOv3 is it does not use pooling operation to reduce the spatial resolution. It repeatedly performs kernel operation with the filter size of 3×3 and 1×1 and of stride 2. Pooling often looses information when used to decrease the spatial resolution. But convolution makes sure that the information is not lost during decreasing the spatial resolution. This enhances the YOLOv3 network to capture all the fine features and thus able to make better predictions.

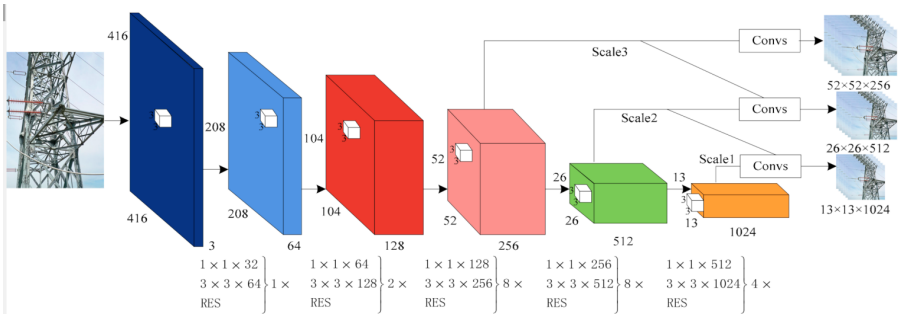


Figure 7: YOLOv3 architecture.

The architecture splits the input image in an $s \times s$ grid. For each grid, the network generates three bounding boxes and class probabilities. Bounding boxes contain potential objects, and the class probabilities correspond to the object's class [15].

4.2 YOLO Loss

YOLO loss is calculated in the training phase, which lets us know the learning rate of the network. Also one of the common problems of object detection networks is predicting multiple bounding boxes over the same object with different class probabilities. So, this section briefly introduces the concepts of Intersection over Union (IoU) and Non-max Suppression (NMS) which helps in discarding multiple boxes over the single object, thereby increasing the prediction accuracy.
Intersection over Union (IoU) is calculated by considering the both the predicted and ground truth bounding boxes. The area of intersection is divided by the area of union of both the bounding boxes. If this calculated value is greater than the threshold value (generally 0.5), then the prediction is good enough. Intuitively, the more the value of threshold, the more is the prediction of the model.

Non-max Suppression (NMS) is used to discard the multiple boxes around the object. It first takes the bounding box with the highest class probability and then calculates the IoU value for all the other boxes. Then the boxes with the highest IoU values are discarded or suppressed as they are similar to the bounding box with highest class probability. Next, it considers the bounding box with the next highest class probability and calculate IoU values to discard the similar boxes. This process is repeated till the one bounding box with the highest class probability is found. This bounding box is saved for the corresponding object and is shown as prediction result.

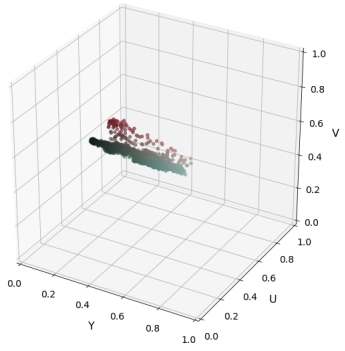
YOLO loss is calculated in three different forms and they are *classification*, *localization* and *confidence* loss. The sum of these three individual losses is computed to produce YOLO loss. For computing the YOLO loss, the bounding box with the highest class probability (highest IoU values with the ground truth) is considered.

To calculate loss, the YOLO network uses the sum-squared error between the predictions and the ground truth. The loss function composes of:

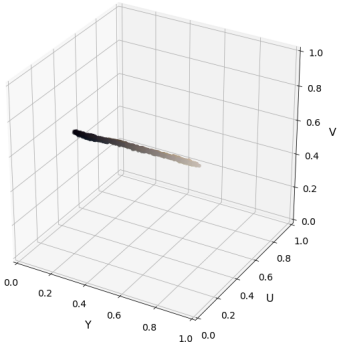
1. Classification loss,
$$\sum_{i=0}^{s^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \bar{p}_i(c))^2 \tag{8}$$
where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0.
$\bar{p}_i(c)$ denotes the conditional class probability for class **c** in **i**.

2. Localization loss(errors between the predicted boundary box and the ground truth),
$$\lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} [(x_i - \bar{x}_i)^2 + (y_i - \bar{y}_i)^2] + \lambda_{coord} \sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} [(\sqrt{w_i} - \sqrt{\bar{w}_i})^2 + (\sqrt{h_i} - \sqrt{\bar{h}_i})^2] \tag{9}$$
where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0 λ_{coord} increases the weight for the loss in the boundary box coordinates.

3. Confidence loss(the objectness of the box),
$$\sum_{i=0}^{s^2} \sum_{j=0}^{B} 1_i^{obj} (C_i - \bar{C}_i)^2 \tag{10}$$
where, $1_i^{obj} = 1$ if an object in cell **i**, otherwise 0.
$\bar{C}_i$ is the box confidence score of box **j** in cell **i**



(a) YUV colour space for 1st vehicle image



(b) YUV colour space for 1st non-vehicle image

Figure 9: YUV colour space of 1st vehicle and 1st not-vehicle image

By adding up classification loss, localization loss and confidence loss, the total loss can be calculated.

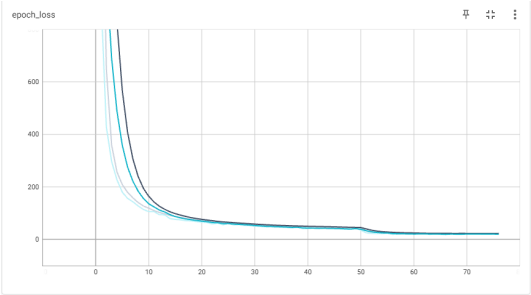


Figure 8: Epoch Loss for 500 images (black line: training loss, blue line: validation loss)

5 Results

In order to determine the best performance provided by a colour space, several experiments using various colour spaces were performed, keeping the same values for the HOG and SVM parameters. We executed the code with different colour spaces, namely HLS, RGB, YUV, HSV. Table 2 shows the performance in terms of the error percentage of various colour spaces. According to the results obtained, YUV and YCrCb have the least misclassified samples or the best performance. In this paper, we used YUV colour space for predictions.
A test video collected from the KITTI dataset is used. The vehicle tracking is performed using the already trained SVM classifier. The SVM classifier predicted well on the cars which are bigger and nearer to the dashboard camera. However, the SVM classifier is not able to make any predictions on the cars which are small and far away from the dashboard camera. One of the major drawbacks of the HOG based SVM classifier is the computational costs. The test video used is approximately 50 seconds in length. The SVM classifier took almost 15 minutes to process and track the vehicles. This is due to the sliding window approach. The HOG features are calculated for each window and then sent to the SVM classifier. The SVM classifier then predicts the window that has the car's HOG features and thus detects the cars. If the number of sliding windows is less, then maybe the SVM classifier makes faster predictions, but the accuracy drastically drops. Hence considering the computational costs, the HOG based SVM classifier is a poor choice for real-time predictions.

The other approach discussed in this paper is the YOLO network approach. Training of the YOLO

	Successful detections	Failed detections
YOLO	40	9
SVM	3	45

Table 1: Comparison of SVM classifier and YOLO network

(a) Cars far away are not detected using HOG+SVM

(b) Cars far away are also being detected by YOLO

Figure 10: Comparison of SVM and YOLO)

network takes much longer than training SVM classifier using HOG features. However, after the training phase, YOLO performs very well in the production phase. Unlike the SVM classifier, which took around 15 minutes to process and detect vehicles in the video, the YOLO network took only 98 seconds. This prediction time can be reduced further when applied different techniques (like bringing down the image resolutions). YOLO, like the SVM classifier, fails to detect cars far away from the dashboard camera. Unlike the SVM classifier, YOLO can predict the smaller cars.
The YOLO network can predict 40 cars and failed to predict nine cars. In contrast, the SVM classifier could only predict three cars and failed to detect 45 cars. Considering the accuracy and significantly less computational costs in the production phase, YOLO is a good choice for real-time vehicle tracking.

Colour Space	Error Percentage
HLS	3.35
RGB	3.19
HSV	1.76
LUV	1.61
YCrCb	1.43
YUV	1.43

Table 2: Colour Space Performance

6 Conclusions

We have successfully implemented a pipeline for detecting vehicles using HOG and linear SVM. We have studied the influence of various colour spaces on performance and concluded that YUV and YCrCB colour spaces have the best performance parameters. However, it should be noted that the performance of these parameters also depends on the input image and video. The goal was to find optimal values for which the parameters have the best performance possible. We have shown that using the HOG feature and linear SVM is an acceptable approach for vehicle detection in images and videos. Using the YOLOv3 network, we have found that the vehicle detection performance has improved since the number of false positives is reduced significantly compared to the SVM classifier.

However, due to GPU disruptions, we could not train the YOLO network on the complete dataset. The training was done on 500 and 1000 images.

6.1 Future scope

There are also numerous HOG extensions and various filters which can improve the overall performance. The system can be made more accurate by using non-linear kernels, such as, Radial Basis Function (RBF) kernel but there is a trade off in the computational time. Another possibility is to use random forests model. However, these changes are not recommended because the of the computation costs and hence HOG features cannot be used in real time detectiona and tracking.

YOLOv3 is state-of-the-art object detection network as of now. Kalman filter can be used to improve the vehicle tracking. Kalman filter is expected to increase the confidence the bounding boxes and also helps in tracking the vehicles during occlusions.

References

[1] N. Tomikj and A. Kulakov, "Vehicle detection with hog and linear svm: Vehicle detection with hog and linear svm," *Journal of Emerging Computer Technologies (JECT)*, vol. 1, no. 1, pp. 6–9, 2021.

[2] A. Iqbal, "Obstacle detection and track detection in autonomous cars," in *Obstacle Detection and Track Detection in Autonomous Cars, Autonomous Vehicle and Smart Traffic, Sezgin Ersoy and Tayyab Waqar, IntechOpen*, 2020. [Online]. Available: https://www.intechopen.com/books/autonomous-vehicle-and-smart-traffic/obstacle-detection-and-track-detection-in-autonomous-cars

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[4] L. Weng, "Object detection for dummies part 1: Gradient vector, hog, and ss," *lilianweng.github.io/lil-log*, 2017. [Online]. Available: http://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html

[5] C. Tomasi, "Histograms of oriented gradients," *Duke University*, 2015.

[6] Wikipedia, "Histogram of oriented gradients," available at https://en.wikipedia.org/wiki/Histogram$_o f_o riented_g radients$, 2021.

[7] A. Rosebrock, "Sliding windows for object detection with python and opencv," 2015. [Online]. Available: https://www.pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv/

[8] Mithi, "Vehicle detection with hog and linear svm." [Online]. Available: https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a

[9] A. Rasouli and J. K. Tsotsos, "The effect of color space selection on detectability and discriminability of colored objects," 2017.

[10] P. Caleiro, A. Neves, and A. Pinho, "Color-spaces and color segmentation for real-time object recognition in robotic applications," *Revista Do DETUA*, vol. 4, 01 2007.

[11] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[12] I. M. Creusen, R. G. Wijnhoven, E. Herbschleb, and P. de With, “Color exploitation in hog-based traffic sign detection,” in *2010 IEEE International Conference on Image Processing*. IEEE, 2010, pp. 2669–2672.

[13] Ö. Kaplan and E. Saykol, “Comparison of support vector machines and deep learning for vehicle detection.” in *RTA-CSIT*, 2018, pp. 64–69.

[14] A. Kathuria, “What’s new in yolo v3?” 2018. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[15] C. Liu, Y. Wu, J. Liu, and Z. Sun, “Improved yolov3 network for insulator detection in aerial images with diverse background interference,” *Electronics*, vol. 10, no. 7, p. 771, 2021.