CS/RBE 549 Computer Vision, Fall 2018
Project Report

Team 13

Member	Signature	Contribution (%)
Ganesh Prasanna Balakrishnan	B Ganesh Prasanna	34
Ajith Kumar Ganesan Govindasamy	Ajith G	33
Abhilasha Rathod	Abhilasha	33

Grading:		
	Approach	**_____/20**
	Justification	**_____/10**
	Analysis	**_____/15**
	Testing & Examples	**_____/15**
	Documentation	**_____/10**
	Difficulty	**_____/10**
	Presentation	**_____/20**
	Total	**_____/100**

ABSTRACT

In this project we implement and experiment with a few Computer Vision (CV) and Deep Learning (DL) algorithms to detect road objects.

1. First, we use classical CV techniques to detect lane lines.
2. Then we try experiment with a few object detection algorithms to detect cars and compare them based on their performance, in terms of accuracy and speed.

To detect lane lines we employ classical Computer Vision algorithms that were taught in class (like calibration, histograms, projective transformation, Hough line detection, binarization, etc.). We then use the pipeline that we built to detect lane lines on a road in our neighbourhood.

Then we move on to detecting cars on the road. We approach this problem in two different ways:

- Using a sliding window approach and classical CV and SVM to detect cars.
- Using a Deep Learning based single shot object detection network (YOLO) for detecting cars.

We compare the two algorithms and talk about the drawbacks of each and briefly discuss the evolution of object detection through the years to what it is today (current state of the art).

Table of Contents

List of Figures

INTRODUCTION:

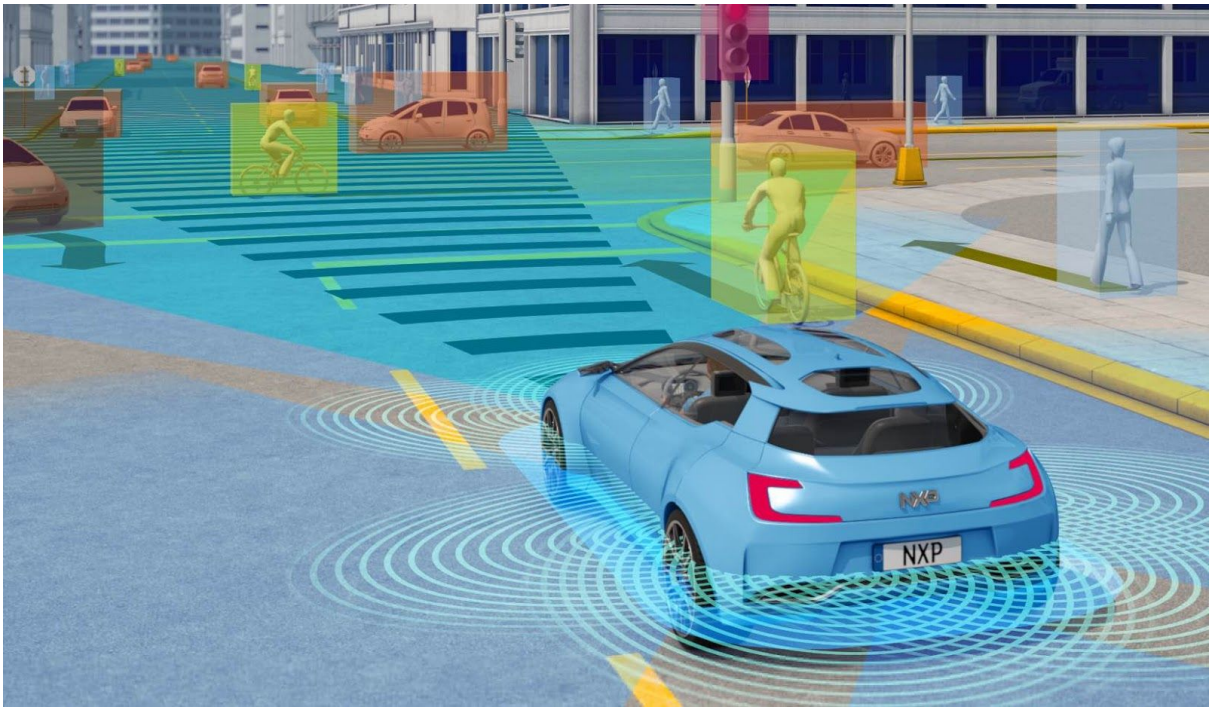

Figure 1: Self-driving car

The Future of transportation is going through a renaissance. Self- driving technology is at its peak and this excitement can be viewed all over the world. The streets of Phoenix are sprawling with self-driving cars testing various traffic scenarios. Well, autonomous vehicles can only get better with time, as these cars are constantly learning new information about roads, other drivers, environments, etc. This information can then be transferred to other fleets, which enhances the learning exponentially. But since the self-driving cars are machines capable of learning, they need to be taught how to perceive their environment.

At a basic level this is an object detection problem. The cars need to know what objects are around them, how are they interacting with the world and how they can track them. This is an essential component of a self driving car, but a special purpose object detection systems need to be fast, accurate, dedicated to classifying a handful but relevant number of objects. These handful of objects would include traffic signals, people, cars, bikes, buses etc. The detection would need to be robust and in real time, as this is crucial for self driving cars to become safer than regular cars, as the goal is to one day replace them.

Our goal on this project is to tackle the detection problem. We detect cars and lane lines on the road using two different approaches, both a Classical Computer Vision approach and a Deep Learning approach.

METHODOLOGY:

Car detection:

1. SVM + HOG Pipeline:

HOG (Histogram of oriented Gradients)

The Histogram of oriented gradients is a popular feature descriptor used in computer vision and image processing applications. A feature descriptor is basically a representation of an image or a part of an image that simplifies the image by extracting useful information.

In the working of the HOG feature descriptor, the input image first is broken into cells of different sizes, then the for each cell the gradients of each pixel are calculated, this could be done by filtering the image with x and y sobel filters as well.

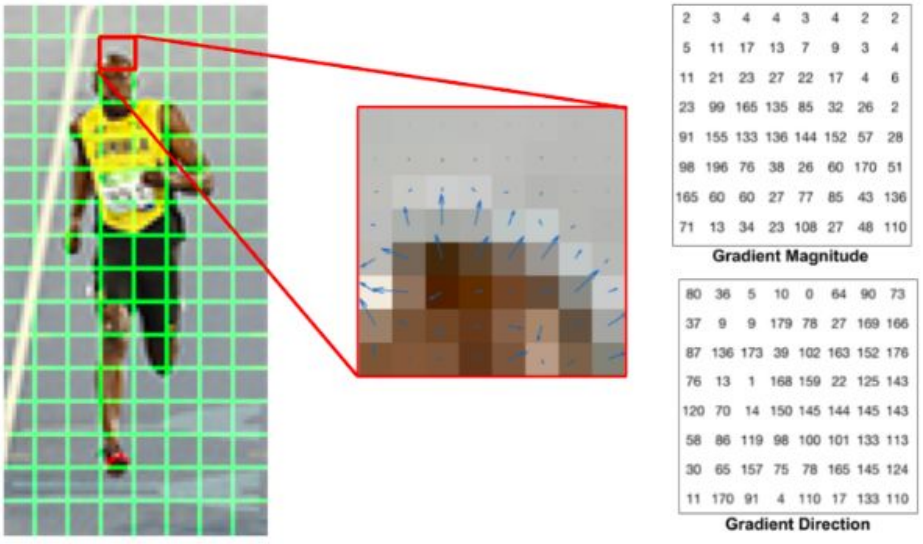

Figure 2: Histogram of oriented gradients

Now the magnitude and orientation of gradient of each pixel is calculated. It could be done for a 8*8 cell, 16*16 or 32*32 cell size. For a cell size of 8*8, it will have 8*8*2 = 128 values for both magnitude and orientation, these values can be placed into 9 bins.This effectively reduces the 64 vector values to 9 values.

The HOG feature descriptor also normalizes the gradients to ensure invariance to illumination changes i.e brightness and contrast. This is represented in the form of a Histogram, which is a graphical representation of the data.

SVM (Support Vector Machine)

A Support Vector Machine (SVM) is a discriminative classifier that finds the optimum hyperplane in an N-dimensional hyperspace that can distinctly classify data points.
The SVM is commonly used for classification and regression task in machine learning.

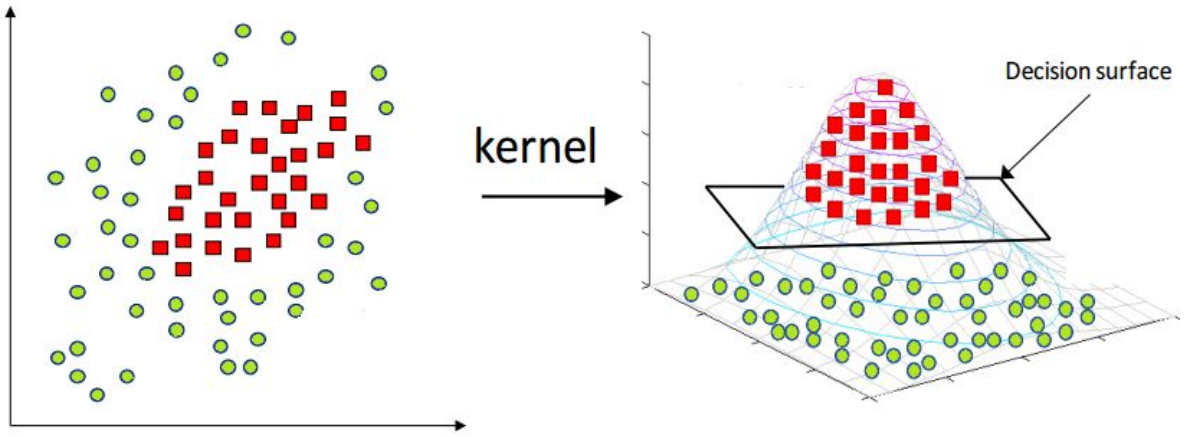

Figure 3: Kernelling of Data points

To separate two classes of data points, there can be many possible hyperplanes that could be chosen. The optimum hyperplane will represent the one with the largest margins i.e the maximum distance between data points of both classes. Maximizing the data points ensures that future data points can be classified with more confidence. For a 2 dimensional feature vector the hyperplane would be a line, since the data is also 2 dimensional.

Support Vectors are data points that are closer to the hyperplane and strongly influence the position and orientation of the hyperplane. Using the support vectors we can optimize or maximize the margins of the classifier.

There are tuning parameters that can help to classify our data well, one such parameters is kernel. Kernels are a great way to transform data that are in multidimensional space into a lower dimensional space. In the case of a 3 dimensional space like in the image shown, we can transform the 2 dimensional points into a 3 dimensional space, separate the data points using a plane between two data classes, and then retransform the 3 dimensional data back to 2 dimensions. In this case the projected plane would be elliptical in shape. This method is called kernelling. The transformations are called kernels.

Figure 4 : Lower Regularization parameter (left) and Higher Regularization parameter (right)

Other tuning parameters like regularization, gamma and margin also play a crucial role for increasing the accuracy of the data points classified. A higher regularization parameter does a good job of getting all the data points correctly classified correctly but has a smaller margin. A smaller regularization parameter has a larger margin but will have a few misclassifications.

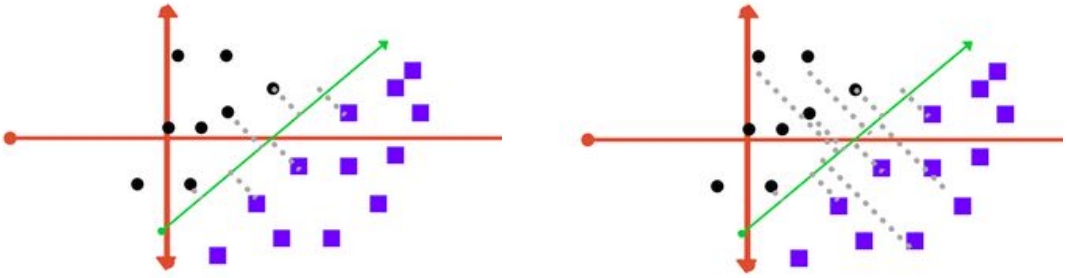

Figure 5 : High Gamma parameter (left) and lower Gamma parameter (right)

The Gamma parameter defines the influence of individual points on the classification. For a lower gamma parameter the points which a further away from the separation line have a higher influence, whereas for a higher gamma parameter the points closer to the separation line have a higher influence on the model. The Third important parameter is the Margin. A Margin is a separation of the plane to the closest point. A good margin is one where the separation is large for both the classes. If the margin is closer to one of the classes then it is not considered as a good margin.

Training:

The Dataset used for extracting feature vectors is the GTI Vehicle image dataset. This Dataset consists of equal amounts of car and non-car images. Since the dataset is quite small, we can use data augmentation techniques to increase the number of images in our dataset. Since the machine learning algorithm considers images as different which are only slightly different. There are several ways to augment data, cropping ,flipping, rotation, scaling, translation, gaussian noise and even we can use advanced augmentation techniques like conditional GANs (Generative Adversarial Networks). This can increase the performance of the model. Training is done by extracting the HOG features from the GTI Vehicle image dataset.

Visualization of the Dataset:

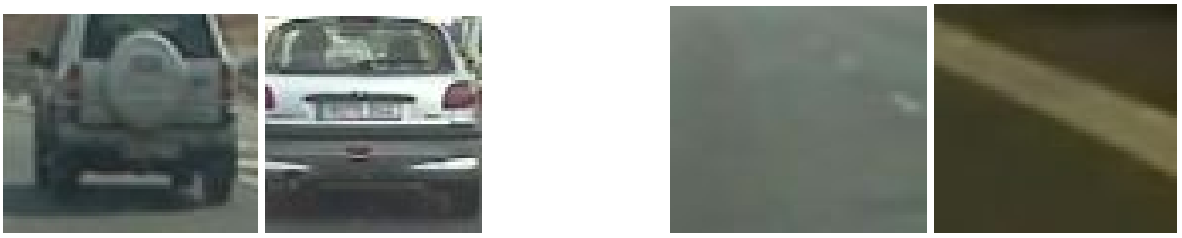

Figure 6: Car and Non-Car Images (GTI Dataset)

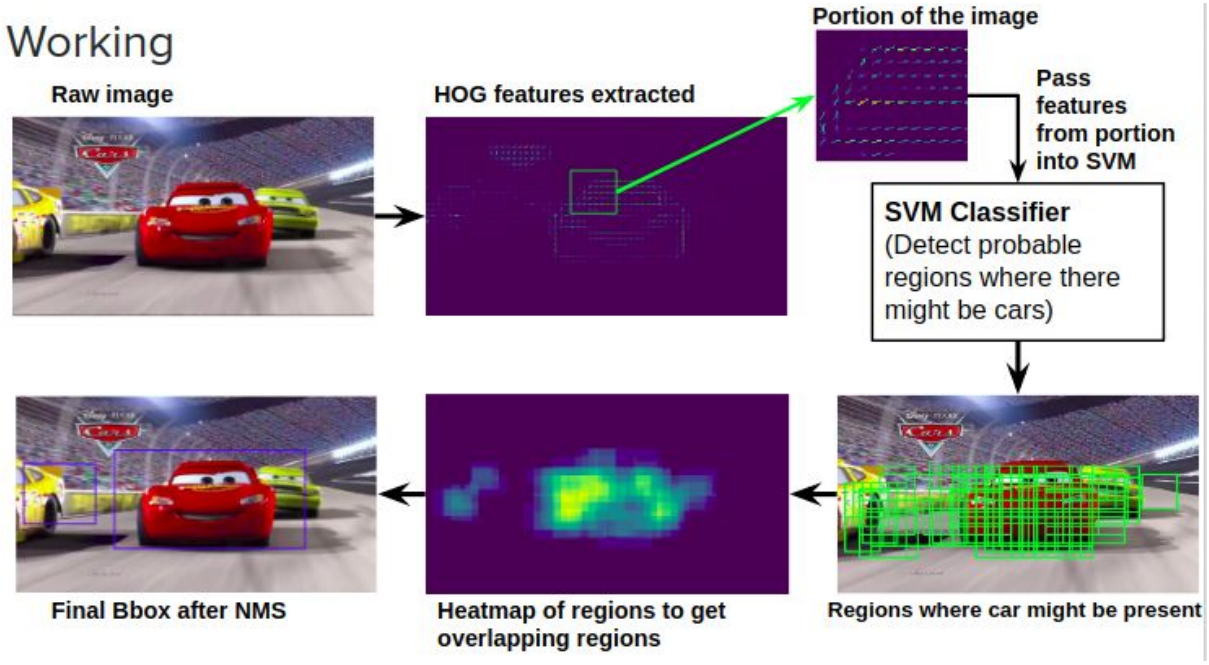

Figure 7: SVM + HOG pipeline

Working :

After training the GTI Vehicle Image Dataset on the Support Vector Machine (SVM) classifier. Testing can be performed on the model using a random sample image from the internet. First the raw image is passed in through, using a sliding window approach we segment the image into multiple windows of varying sizes. For each of these windows, the HOG features are extracted using the HOG feature descriptor. The feature vector obtained is then passed through the trained SVM model.

If the prediction of the SVM comes out as positive or that it is a car, those windows are displayed. Since there can be many possible windows of many sizes that can contain the car. All the possible windows are displayed. Most of the windows get overlapped, this can be better represented using the concept of heatmaps. The heat maps are just representations of maximum likelihood that an object exists in the represented region. Everytime there is an overlapping window in the image, the image intensity is added as represented as a heatmap.

Our pipeline could give multiple detections for a single car object in the image. Since we need to draw the best possible bounding box around the object. This can be solved using the Non-max Suppression algorithm. If there are more than one detections, then the detection with the maximum probability is then taken as the final detection and the ones which have lesser probability are suppressed, hence the term Non-max suppression.

2. DEEP LEARNING:

There is a plethora of options for object detection using Deep Learning. We read through a lot of papers and realized that the best one to go on with would be the YOLO.

Why not any among the RCNN series?

The first RCNN paper [4] uses a selective search algorithm and uses three separate networks - a CNN for feature extraction, an SVM for classification and a model for the bounding box regression. The selective search method gives around 1000-2000 bounding boxes for the image proposing them as probable regions where the objects might exist and each of them are passed into the CNN individually, resulting in a very slow process. The three networks have to be trained individually and it is very hard to make the networks to work in tandem, complimenting each other when they are trained individually.

The second paper - Fast RCNN [3] avoids the three networks by using a softmax layer and also regresses from the feature vector obtained from the CNN. The paper also discusses about passing in the entire image as a whole and using a Region of Interest Pooling mechanism. The way it works is by calculating which region of a feature map would correspond to the region proposal obtained from selective search and pooling out those regions to generate a feature vector. Shared computations make the network run much faster but still the bottleneck remains selective search making the overall run time very long.

The third paper - Faster RCNN [5] uses a network to obtain region proposals (bounding boxes) and uses these proposals to perform ROI pooling. This significantly reduced the number of proposals and also the run time of the algorithm. However training the network is very hard as you would manually have to train the two networks four times. It is described as a four step training process in the paper and it is very hard to train given the time constraints.

YOLO

Based on the reasons mentioned above we decided to go ahead with YOLO [2] which is currently the state of the art in object detection. YOLO is a single shot detection algorithm where the network given an image, simultaneously predicts the bounding boxes containing objects and the probability of the object belonging to the classes.

So, YOLO basically divides the image into sxs grid cells and for each of those cells it detects k bounding boxes. For each bounding box the network predicts 5 values:

- Width of the bounding box
- Height of the bounding box
- X coordinate of the bounding box
- Y coordinate of the bounding box
- Probability of an object being present

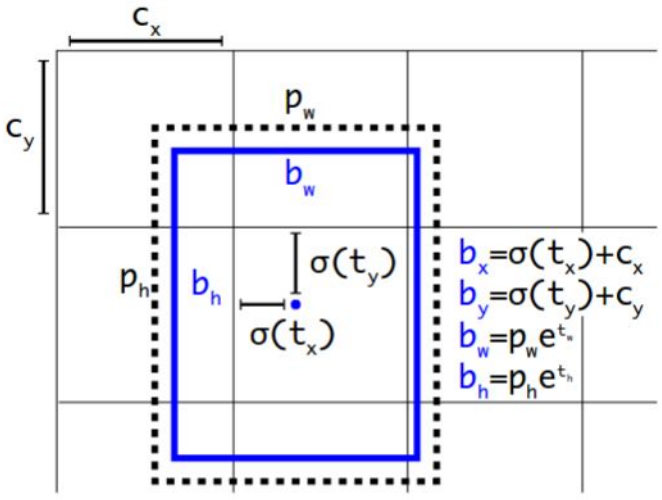

Figure 8: Bounding boxes with dimension priors and location prediction

The network only predicts the c probabilities for a bounding box only if the box has an object within it. For each of those bounding boxes it determines the probability with which the object can belong to any of the c classes that it was trained to detect. The output of YOLO for an image is an (s x s) x (5k + c) tensor

Anchor boxes

Anchor boxes are a core concept of YOLO. They act as priors to the bounding box regressor part of the algorithm. These give the regressor a head start. To find the ideal number of anchor boxes(k) to use, we plot the width and height of the bounding boxes in the dataset in a 2d plane and cluster the data into clusters. We use Expectation Maximization [7] to do a soft clustering and use the Bayesian Information Criterion [8] to determine the optimum number of clusters.

Figure 9: Clusters of bounding boxes in data (left); Bounding boxes with

dimensions corresponding to cluster centers (right)

We find that the number of anchor boxes for our dataset is two (the two shown above). We tried to train the network on the publicly available LISA dataset and the network did not converge. So we went ahead with pre-trained weights.

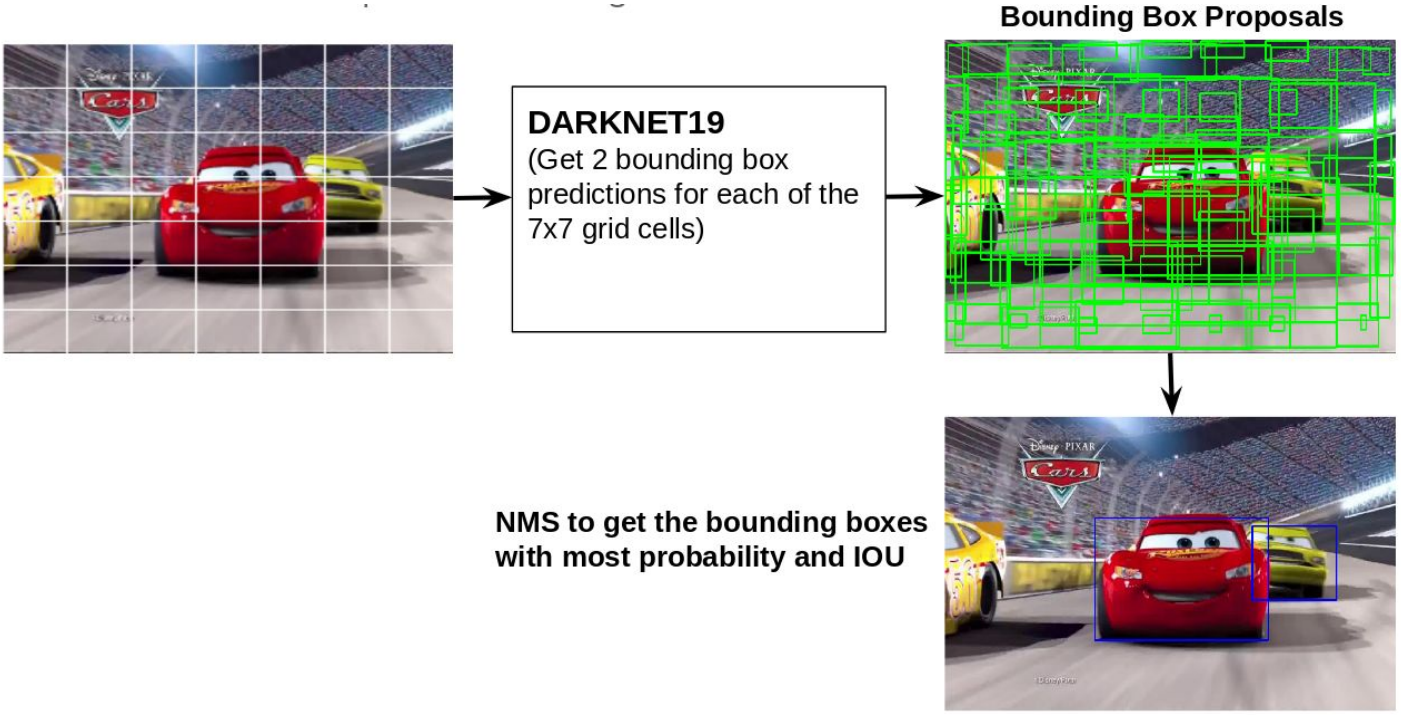

Figure 10: YOLO framework

So YOLO basically predicts the width and height of the box as offsets from cluster centroids and predicts the center coordinates of the box relative to the location of filter application (cell) using a sigmoid function. More details regarding the architecture and the loss function that helps the network to regress good bounding box values and classification can be found in [2].

Lane line detection:

For our model to work we have a set of assumptions that we need to make sure that we follow:

- The car (and thereby the camera) is in the center of the lane.
- The lane lines are angled at -30 to 30 degrees wrt the vertical line.
- Lane lines are white or yellow.
- The lane lines at the bottom are almost as wide as the image.

Our approach consists of the following :

- Camera Calibration
  We need to calibrate the camera to correct for any distortions that the camera might introduce in the image. The most common kinds of distortion are:
  - Radial Distortion

Real cameras use curved lenses to form an image, and light rays often bend a little too much or too little at the edges of these lenses. This creates an effect that distorts the edges of images, so that lines or objects appear more or less curved than they actually are. This is called radial distortion.

- Tangential distortion
  Less common and occurs when a camera's lens is not aligned perfectly parallel to the imaging plane, where the camera film or sensor is. This makes an image look tilted so that some objects appear farther away or closer than they actually are.

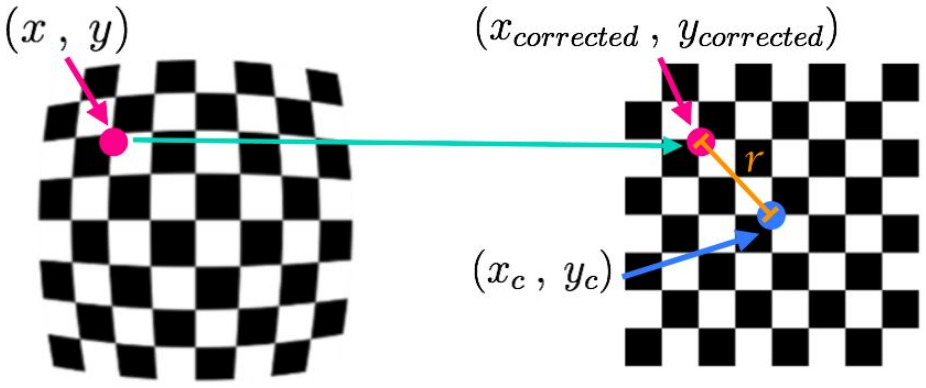

Figure 11: Distorted and undistorted image of chessboard

$$x_{corrected} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \qquad x_{corrected} = x + [2p_1xy + p_2(r^2 + 2x^2)]$$
$$y_{corrected} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) \qquad y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2xy]$$

K1, k2, k3, p1 and p2 are the distortion parameters that we need to correct the image.

- Binarization
  After correcting to image to rectify effects due to distortion we binarize the image to isolate pixels belonging to the lanes. There are two stages of binarizing that we do:
  - Gradient based:
    We know that lane lines are slanted about 30 degrees clockwise and 30 degrees anticlockwise. We find the gradients in the x and y direction and find the direction of each gradient and threshold out pixels whose direction of gradients do not lie within the range mentioned above.

  - Value based:
    This still leaves us with lines that are within that range of directions but do not belong to lanes, like lines corresponding to the pavements. We need to remove these lines. To remove them we use a thresholding to extract pixels from the image with values corresponding to either white or yellow pixels. Then we take the intersection of gradient based and value based threshold and get the pixels that belong to lane lines.

Figure 12: Binarization

- Projective Transformation
  Once we binarize the image we perform a perspective transformation to convert the image into a birds-eye view. This is done to eliminate the vanishing point in the image shown above.The projective transformation is also called a homographic transformation. It is represented by: $x' \propto Hx$

$$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

  H has 8 DOF and thus we need 4 2D points to determine the H matrix. So we choose a trapezoid around the lane lines in the image and map it into a rectangle. This would convert the seemingly convergent lines into parallel ones.

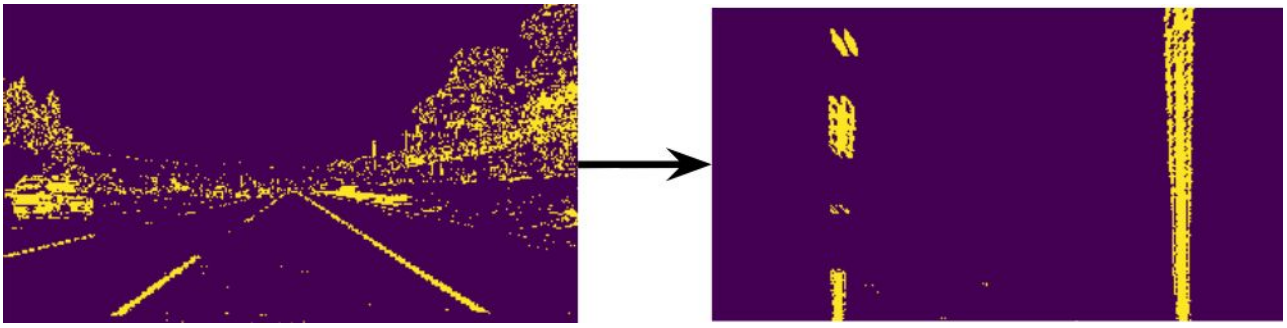

Figure 13 : Projective transformation

- Histogram and sliding window
  After applying the projective transformation we can see that our image still has some noise and we need to find out which pixels belong to the lanes. We generate a histogram for all the pixels located at the bottom half of the image. The peaks of the

histogram represent the approximate location of the vertical lines that is the lane lines.

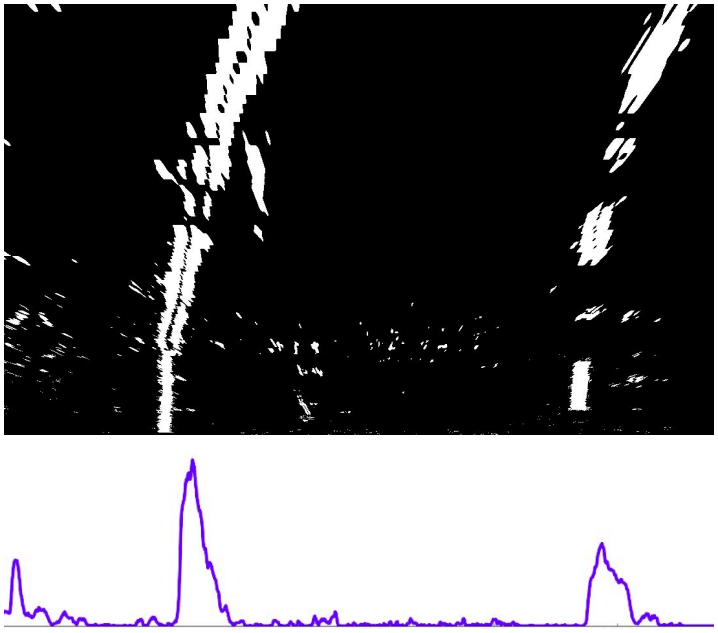

Figure 14: Histogram of lane lines

We split the histogram into two halves (left and right) corresponding to the left and right lanes. We can use the two highest peaks from our left and right histograms as starting points for determining where the lane lines are, and then use sliding windows moving upward in the image (further along the road) to determine where the lane lines go. We then center the sliding window at the peak of the histogram and find all the binarized 1 pixels in the window and mark them as left and right lane lines. We then recalculate the mean of left and right lane pixel positions and re-center the bounding box and slide it over the image as shown below. This process is repeated until we have detected all the pixels belonging to the lane lines. As it can be seen from the figure, the window adjusts its position as the lane curves across the image.

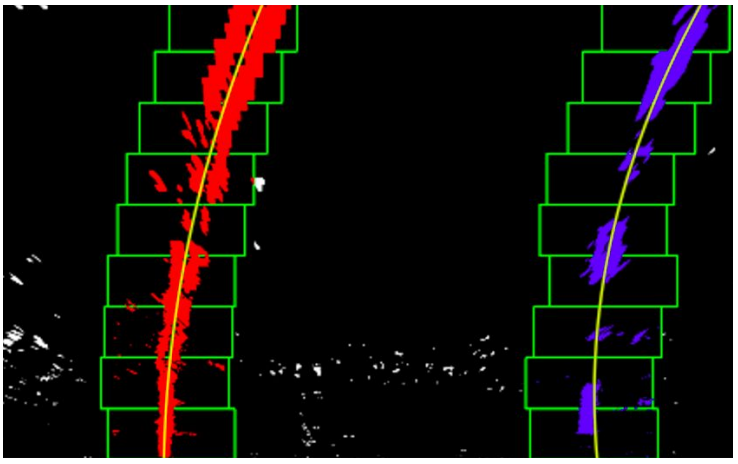

Figure 15: Sliding window technique

- Curve Fitting
  We use a least squares fitting technique to fit a quadratic polynomial in the best way possible for the lane pixels that were detected.

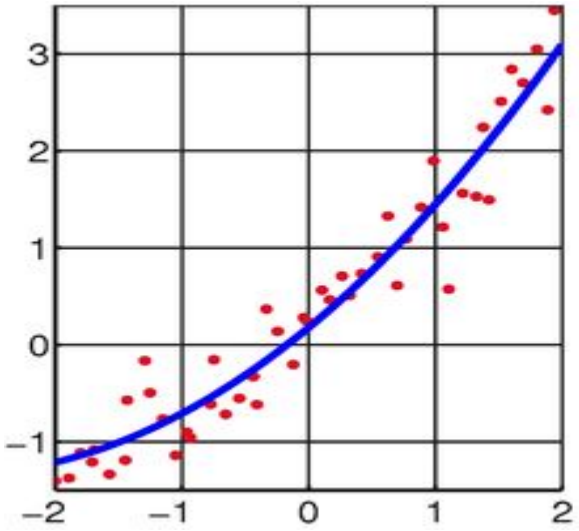

Figure 16: Curve fitting

We have to make sure that we do not detect any noise as a lane pixel as the least square fitting technique is very sensitive to outliers. Thus we need to make sure that the width of our window is sufficiently small. Once we fit our curve we convert our image back into the perspective view using the inverse of the H matrix obtained above.

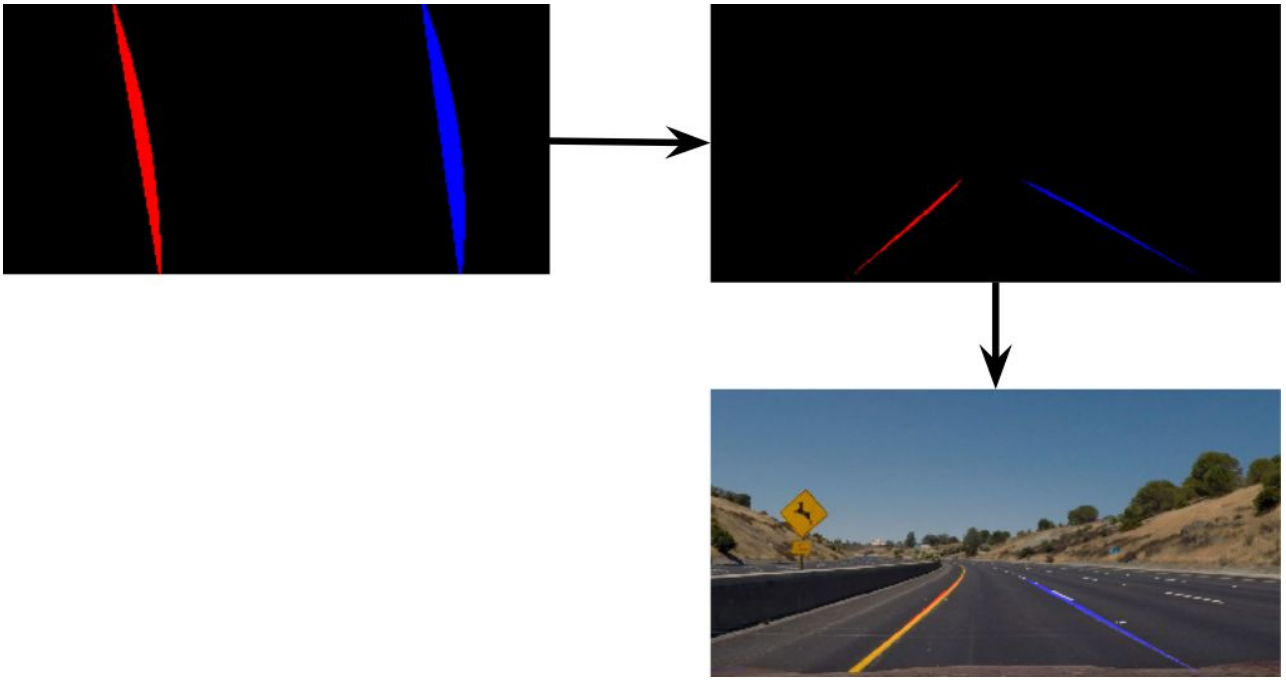

Figure 17: Curve fitting and perspective view of the lane lines

- Video Feed Processing
  Now we have a system that works for a single image. We can use the same algorithm to process the video frame by frame but it would be very computationally expensive to keep running the sliding window based search on every frame in the video. We make a slight modification to the algorithm to use the information that we already know. In a video the $(t+1)^{th}$ frame and the t^{th} frame have very similar lane lines. So we use the curve that we fit in the previous frame and search around the curve from the previous frame to look for lane pixels in the current frame. Then we again fit a new polynomial for the current frame and we keep doing this. This significantly reduces the time taken by our algorithm from 4 fps (only sliding windows) to 21 fps (search around curves from previous frames).

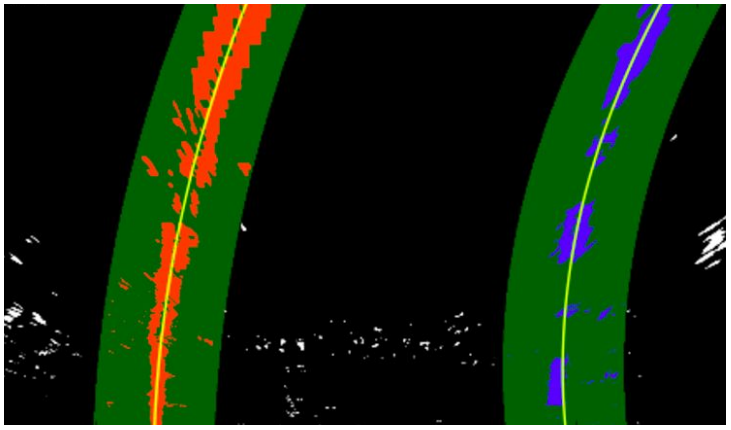

Figure 18: Pixels around the curve

RESULTS:

Car Detection:

For car detection we trained and tested our models - HOG+SVM and YOLO on various videos. For videos with cars that are further away from the camera both the models perform in a very similar fashion. HOG+SVM sometimes detects a few false positives but it detects all the true positives that YOLO detects. This is because the SVM was trained on images of cars that are randomly cropped as shown above.

SVM + HOG: **YOLO:**

Figure 19: Comparison of SVM+HOG(left) to YOLO(right)

However for cars that are very close the camera, the pixels within a window are mostly not representative of features corresponding to a car (as shown below). Thus the true car is never detected and a lot of false positives are detected. YOLO does a really good job in this aspect as we consider this aspect ratio in the anchor boxes that we use for YOLO.

Figure 20: SVM False Positive Reasoning

As shown above most windows that are extracted from the car constitute pixels that are not similar to the training data. Thus the bounding boxes are detected as shown below. On the contrary a window of pixels obtained from the buildings above the car would seem like a few car images present in the dataset due to the alternating occurence of glass and a solid opaque lines similar to that of the backside of a car. This results in a lot of false positives.

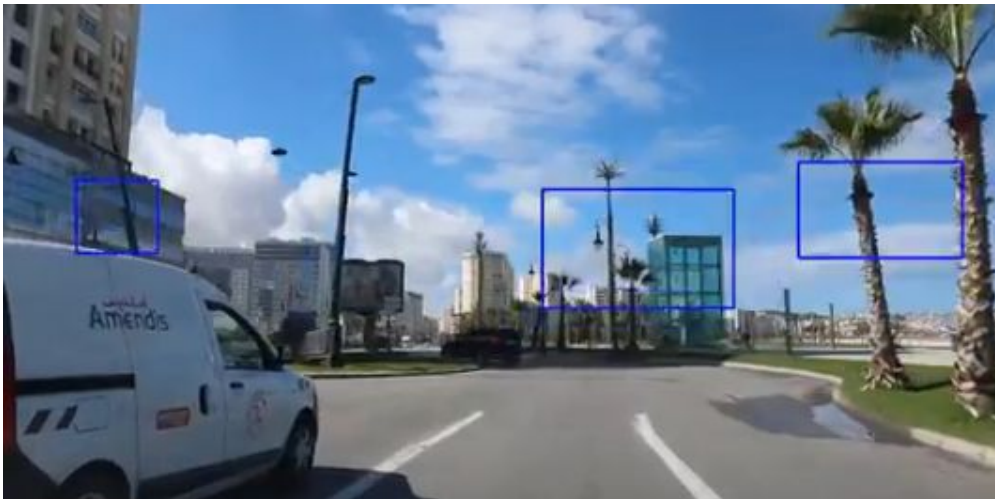

Figure 21:SVM + HOG

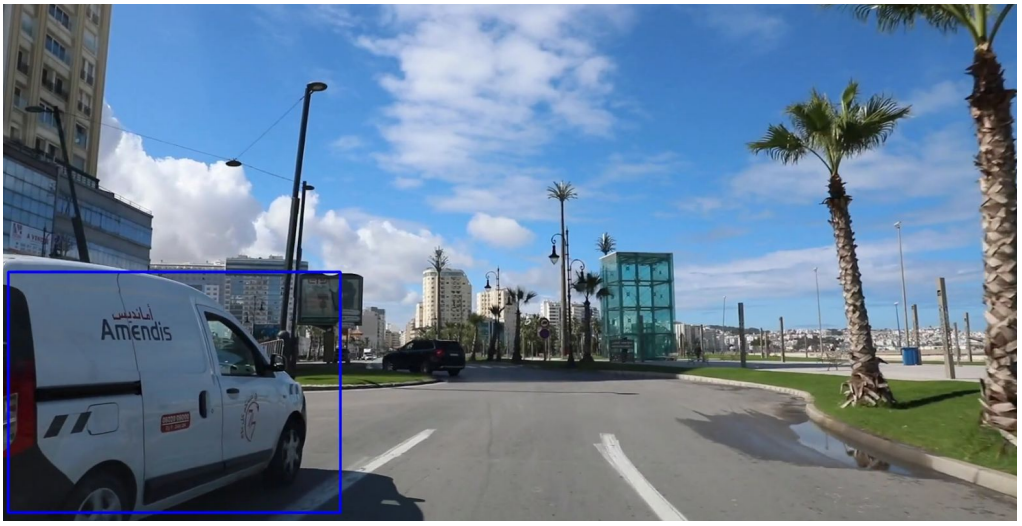

Figure 22: YOLO

We can see that YOLO performs so much better than the HOG + SVM system that we have developed.

Lane Line Detection:

The lane line detection algorithm works when our assumption holds true. Two of the four assumptions we have made aren't necessarily always true. The assumption that the car is in between the lanes is not true when the car is switching lanes and also when an image is taken not by a camera on the car but rather a pedestrian or a cyclist. This will require the user to manually tune the trapezoid around the road in such a way that the road is within it.

Figure 23: Lane Line Detection

Camera not on car **Camera on car when switching lanes**

Figure 24: Cases where no lane is detected

FUTURE WORK:

Car Detection :

- **SVM+HOG :**
  - Train on a larger and a more diverse range of croppings
  - Use a better feature extraction techniques than HOG
- **Deep Learning :**
  - Use a larger dataset to train
  - To detect cars that are close to each other increase the number of grid cells
  - Use more anchor boxes

Lane Line Detection :

- Implement LaneNet [10] for instance segmentation.
- Devise an adaptive method for choosing region to be projected into birds-eye view rather than hard-coded trapezoid.

CONCLUSIONS:

Based on trying out various algorithms for car detection and lane detection we realized that classical computer vision algorithms require a lot of parameter tuning which in turn require an in depth mathematical understanding of Computer Vision. Classical CV sometimes also requires hard-coded variables and thus might not be able to generalize well. Deep Learning techniques are much more robust to new samples but however require a large amount of data for them to work and generalize well. Without the knowledge of Classical Computer Vision it would be hard to figure out if our Deep Learning models are able to extract any meaningful features. In our work we also need a few preprocessing steps before feeding in our image into the

network which requires basic understanding of CV techniques. Therefore in the presence of a large amount of data, Deep Learning can be seen as a much better tool for extracting image features that are robust when compared to classical computer vision algorithm. But, when there is not much data then classical CV triumphs.

REFERENCES:

1. Redmon, J., Divvala, S.K., Girshick, R.B., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779-788.
2. Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, Faster, Stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6517-6525.
3. Girshick, R.B. (2015). Fast R-CNN. *2015 IEEE International Conference on Computer Vision (ICCV)*, 1440-1448.
4. Girshick, R.B., Donahue, J., Darrell, T., & Malik, J. (2014). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580-587.
5. Ren, Shaoqing et al. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks.” *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39 (2015): 1137-1149.
6. Udacity Self Driving Car Nanodegree
7. Elkan, Charles. “Expectation Maximization Algorithm.” *Encyclopedia of Machine Learning* (2010).
8. Ujjwal Das Gupta,Vinay Menon and Uday Babbar, “Parameter Selection for EM Clustering Using Information Criterion and PDDP”
9. https://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html?highlight=hog
10. Neven, D., Brabandere, B.D., Georgoulis, S., Proesmans, M., & Gool, L.V. (2018). Towards End-to-End Lane Detection: an Instance Segmentation Approach. 2018 IEEE Intelligent Vehicles Symposium (IV), 286-291.
11. Learnopencv.com
12. Pixels.com
13. Medium.com