Machine Learning Engineer Nanodegree
Udacity

Dibakar Sigdel,*Ph.D.*

May 27, 2017

Capstone Project Report

Contents

1 Definition

1.1 Project Overview

Object detection is a rapidly growing fied in computer vision for finding and identifying objects in an image or video sequence. Humans recognize a multitude of objects in images with little effort, despite the fact that the image of the objects may vary somewhat in different view points, in many different sizes and scales or even when they are translated or rotated. Objects can even be recognized when they are partially obstructed from view. This task is still a challenge for computer vision systems. Many approaches to the task have been implemented over multiple decades.

Properly identifying objects through object detection technique is a very important problem in computer vision. For example human detection[2] in surveillance cameras, face and facial expression detection[3] in speech videos etc. Our main goal for this project is identifying cars in a video which is a fundamental step in achieving vehicle automation and ultimately fully driverless cars.

This project is inspired by the Vehicle Detection and Tracking Project that is part of Udacitys Self Driving Car Nanodegree. The link for the dataset is provided by Udacity. This project will build a classifier using two sets of data called vehicle and nonvehicle. Once a model is ready, it will be used to test its performance on test video. Following are the data sources:

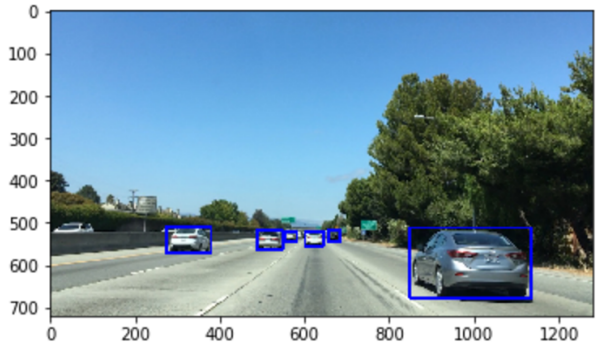
1. Vehicle Images: vehicle-data-link
2. Non-vehicle Images: non-vehicle-data-link
3. Test Video: test-video-link

The image dataset mentioned contains thousands of labeled car and non-car images, which are ideal for use in machine learning, more specifically, supervised learning techniques. A set of features from each images is extracted and then used to train a classifier. In this project we will use a linear support vector machine (SVM). Once trained with the labeled data the SVM will act

as our classifier, predicting if new images contain cars or not.

1.2 Problem Statement

The statement of the problem is **Vehicle detection and tracking in a video using classifier trained over vehicle and non vehicle image data set**. In order to achieve this multiple computer vision and machine learning methods are used in conjunction. The solution to the problem is that, it should be able to select proper pixels in the image frames in a video as vehicle and draw a rectangular boundary.



More technically, the accuracy of the classifier will be calculated with respect to train, test data as well as the performance of classifier on videos. The results thus obtained should be replicable using same data set.

In the implementation of this project labeled images(cars and non-cars) will be processed and represented by some of their feature vectors. The images are separated by the presence of a car in the image or lack of car. By having labeled data we can differentiate the features of an image that has a car and one that does not. These features can then be used to train a classifier such that when given a new image it can predict weather or not there is an car in the given image.

Technically, the solution to this problem will be an algorithm pipeline that will result in a video (which is just a sequence of images) with bounding boxes around the cars that have been identified.

1.3 Metrics

There are two metrics that we will use to quantify different components of this project. First we will measure the accuracy of the classifier used on the training and testing data. Secondly we will measure the performance of our algorithm as a whole by calculating the accuracy of the vehicle detection in the final video result. Since we are using a binary classifier and we want to account for false-positives we will use the following formula to calculate accuracy:

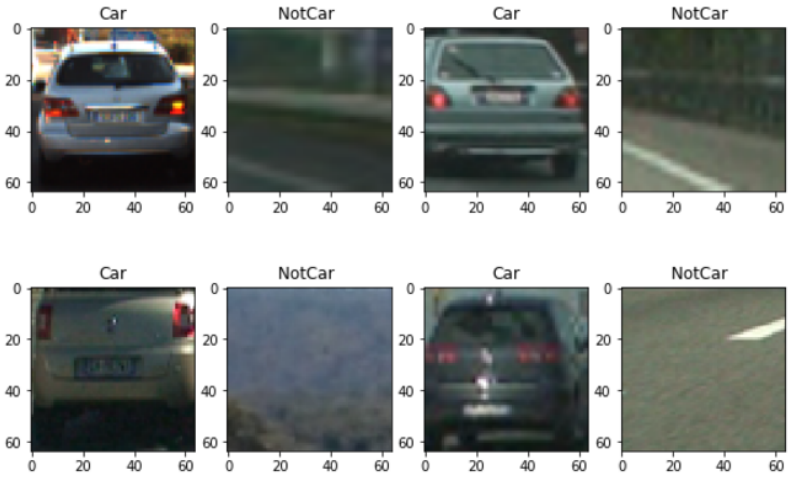
$$\text{Accuracy} = \frac{(\text{true positives} + \text{true negatives})}{\text{size of dataset}} \tag{1}$$

Here are true positives are cars identified correctly. True negatives are image segments without a car identified as such. The size of the dataset is the total number of image segments classified. It has been mentioned that accuracy as a metric is suffered when the dataset is very imbalanced. This is not the case for the dataset in this project. Our dataset is divided into two classes (car and non-car) of equal size. For the implementation of this project we only extract features from exactly 4000 images from each class, making our data completely balanced.

2 Analysis

2.1 Data Exploration

The dataset used in this project is composed of thousands of labeled car and non-car images. All on these images are 64 x 64 pixels which provides a consistent number of features extracted from each image. Here a sample of the images from the dataset.



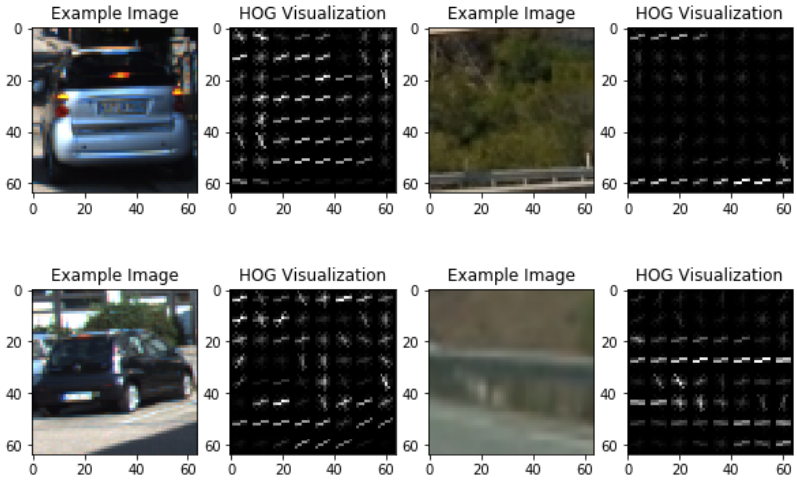
2.2 Exploratory visualization

2.2.1 HOG feature

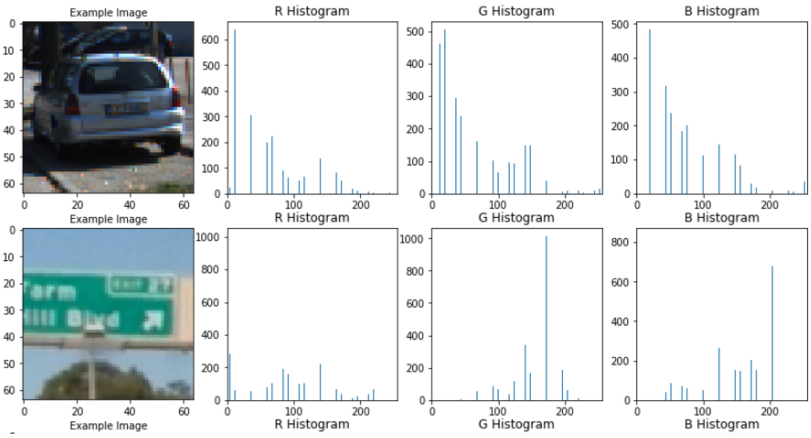
The first set of features extracted from our image dataset is obtained by using a Histogram of Gradients (HOG). For the purposes of this project, the HOG feature extraction implementation provided scikit-image will be used. A HOG feature extraction works by calculating the magnitude and direction of each pixel. These are calculated from the set of pixels surrounding a given pixel. Looking at these values we can calculate a gradient. This gradient is useful getting information about what is represented in an image. The we segment the original image into cell and create a histogram for the gradient values of each pixel. These values are then put into orientation bins on a histogram. The bin containing the largest value is our dominating orientation, but all of the bins are taken into consideration when calculating the orientation of the image segment. The histogram now gives us a sense of what is in the

image cell, and once do the same calculations for each cell of the original image, we get a feature set representing our original image

In the above images we the the input images on the left, and the HOG



visualization on the right. The image on the upper right hand corner has some distinct darker colored lines which indicate a larger gradients. The image on the bottom right lacks any strong gradients. Different objects generate different histograms, and these features can be used to classify them.

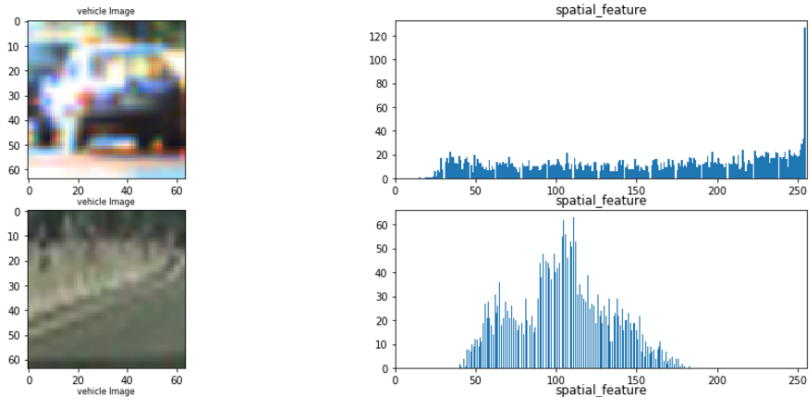


2.2.2 Color and spatial features

The other set of features used to classify our images is a histogram of color. This means calculating the number of pixels of a certain color range. This will give us another histogram which can be used to match know histograms

of certain objects. For example the following images show a car and its corresponding histogram of color.

The resulting histograms can be used to compare new images and check if they match. If they do that could help classify the object in the image.



2.3 Algorithm and techniques

This project uses many techniques from computer vision and machine learning to achieve its goals. Two the methods (HOG feature extraction and Histogram of Color extractions) have been described in the previous section. These two sets of features are combined and used to train a linear support vector machine(SVM) classifier.

2.3.1 SVM classifier - The linear model

A support vector machine plots the feature set in plane or hyperplane, and finds a decision boundary between the labeled features. A hyperplane is when multi dimensional features are represented by just 2 dimensions like a xy plane. More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest

training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

Equation of Hyperplane:The equation of a straight line is $y = mx + c$, where m is the slope and c is the y-intercept of the line. Similarly, the generalized equation of a hyperplane is as follows:

$$w^T x = 0 \tag{2}$$

Here $w = \big(-c, -m, 1\big)^T$ and $x = \big(1, x, y\big)^T$ for 1D and are the vectors and $w^T x$ represents the dot product of the two vectors. The vector w is often called as the weight vector.

In the case of higher dimension, w and data represents the vector which is normal to the hyperplane. This property will be useful once we start computing the distance from a point to the hyperplane.

Understanding the constraints: The training data in our classification problem is of the a pair of x_i, an n-dimensional feature vector and y_i, the label of x_i.

$$\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\} \in \mathbb{R}^n \times -1, 1. \tag{3}$$

Where $y_i = 1$ implies that the sample with the feature vector x_i belongs to class 1 and if $y_i = -1$ implies that the sample belongs to class -1. In a classification problem, we thus try to find out a function, $y = f(x) : \mathbb{R}^n \longrightarrow \{-1, 1\}$. $f(x)$ learns from the training data set and then applies its knowledge to classify the unseen data.

There are an infinite number of functions, $f(x)$ that can exist, so we have to restrict the class of functions that we are dealing with. In the case of SVMs, this class of functions is that of the hyperplane represented as $w^T x = 0$ or $\vec{w}.\vec{x} + b = 0; \vec{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$

This divides the input space into two parts, one containing vectors of class 1 and the other containing vectors of class +1. Specifically, we will consider 2-dimensional vectors. Let $\mathcal{H}_0$ be a hyperplane separating the dataset and

other two hyperplanes $\mathcal{H}_1$ and $\mathcal{H}_2$ such that they also separate the data and have the following equations: $\vec{w}.\vec{x} + b = \delta$ and $\vec{w}.\vec{x} + b =$ -δ.

This makes $\mathcal{H}_o$ equidistant from $\mathcal{H}_1$ as well as $\mathcal{H}_2$. The variable δ is not necessary so we can set $\delta = 1$ to simplify the problem as

$$\begin{aligned} D(x) &= \vec{w}.\vec{x} + b = 1 \\ D(x) &= \vec{w}.\vec{x} + b = \text{-}1 \end{aligned} \tag{4}$$

Next, we want to ensure that there is no point between them. So for this, we will select only those hyperplanes which satisfy the constraints; for every vector x_i either:

- $\vec{w}.\vec{x} + b \leq$ -1 for x_i having the class 1 or
- $\vec{w}.\vec{x} + b \geq 1$ for x_i having the class 1

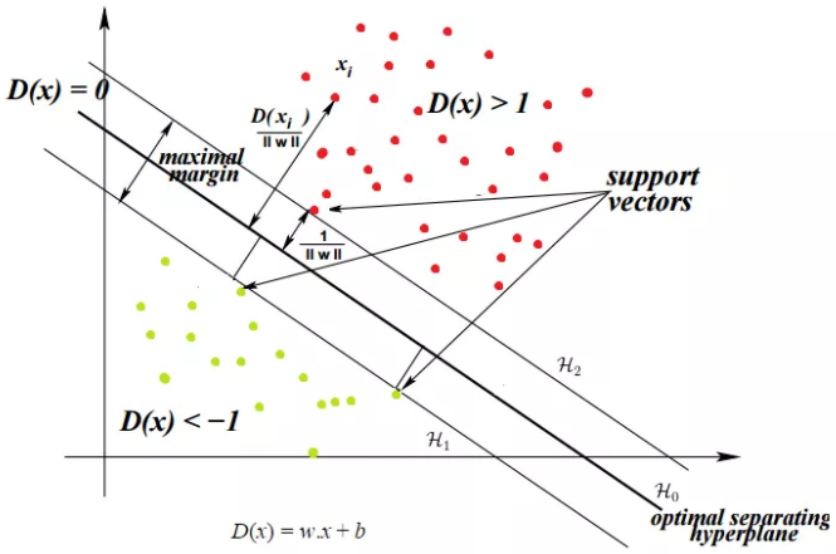


Figure 1: image source: source

Combining the constraints: Both the constraints stated above can be combined into a single constraint.

- For x_i having the class -1, $\vec{w}.\vec{x} + b \leq$ -1. Multiplying both sides by y_i (which is always -1 for this equation) $y_i\,(\vec{w}.\vec{x} + b) \geq y_i(-1)$ which implies $y_i\,(\vec{w}.\vec{x} + b) \geq 1$ for x_i having the class-1.
- For x_i having the class 1, $y_i\,(\vec{w}.\vec{x} + b) \geq 1$ for x_i having the class 1

Combining both the above equations, we get $y_i\,(\vec{w}.\vec{x} + b) \geq 1$ for all $1 \leq i \leq n$. This leads to a unique constraint instead of two which are mathematically equivalent. The combined new constraint also has the same effect, i.e., no points between the two hyperplanes.

Maximize the margin: For the sake of simplicity, we will skip the derivation of the formula for calculating the margin, m which is $m = \frac{2}{||\vec{w}||}$ The only variable in this formula is w, which is indirectly proportional to m, hence to maximize the margin we will have to minimize $||\vec{w}||$. This leads to the following optimization problem:

$$\textit{Minimize in } (\vec{w}, b)\{\tfrac{||\vec{w}||^2}{2} \textit{ subject to } y_i\,(\vec{w}.\vec{x} + b) \geq 1 \textit{ for any } i = 1, \ldots, n\}$$

The above is the case when our data is linearly separable. There are many cases where the data can not be perfectly classified through linear separation. In such cases, Support Vector Machine looks for the hyperplane that maximizes the margin and minimizes the misclassifications. For this, we introduce the slack variable,ζ_i which allows some objects to fall off the margin but it penalizes them. In this scenario, the algorithm tries to maintain the slack variable to zero while maximizing the margin. However, it minimizes the sum of distances of the misclassification from the margin hyperplanes and not the number of misclassifications.

Constraints now changes to $y_i(\vec{w}.\vec{x_i} + b) \geq 1 - \zeta_i$ for all $1 \leq i \leq n, \zeta_i \geq 0$ and the optimization problem changes to

$$\textit{Minimize in } (\vec{w}, b)\{\tfrac{||\vec{w}||^2}{2} + C\textstyle\sum_i \zeta_i \textit{ subject to } y_i\,(\vec{w}.\vec{x} + b) \geq 1 - \zeta_i \textit{ for any } i = 1, \ldots, n\}$$

Here, the parameter C is the regularization parameter that controls the trade-off between the slack variable penalty (misclassifications) and width of the

margin. Small C makes the constraints easy to ignore which leads to a large margin. Large C allows the constraints hard to be ignored which leads to a small margin. For $C = \inf$, all the constraints are enforced.

2.3.2 SVM tuning parameters: Kernel, Regularization, Gamma and Margin.

- **Kernel**:

  It often happens that the sets to discriminate are not linearly separable in the feature space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function $k(x, y)$ selected to suit the problem.Example of some kernel functions are:

  1. **Polynomial** : $K(x_i, x) = (x.x)^d$
  2. **Radial Basis Function** : $K(x_i, x) = exp\left(-\lambda||x_i - x||^2\right); \lambda > 0$
  3. **Gaussian Radial Basis Function** : $K(x_i, x) = exp\left(\frac{||x_i - x||^2}{2\sigma^2}\right)$

  The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant. The vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors x_i that occur in the data base. With this choice of a hyperplane, the points x in the feature space that are mapped into the hyperplane are defined by the relation: $\sum_i \alpha_i k(x_i, x) =$ constant. Note that if $k(x, y)$ becomes small as y grows further away from x, each term in the sum measures the degree of closeness of the test point x to the corresponding data base point x_i. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of points x mapped into any hyperplane can be quite convoluted

as a result, allowing much more complex discrimination between sets which are not convex at all in the original space.

- **Regularization**:

  The Regularization parameter (often termed as C parameter in pythons sklearn library) tells the SVM optimization how much you want to avoid misclassifying each training example. For large values of C, the optimization will choose a smaller-margin hyperplane if that hyperplane does a better job of getting all the training points classified correctly. Conversely, a very small value of C will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane misclassifies more points. The images below (same as image 1 and image 2 in section 2) are example of two different regularization parameter. Left one has some misclassification due to lower regularization value. Higher value leads to results like right one.

- **Gamma**: The gamma parameter defines how far the influence of a single training example reaches, with low values meaning far and high values meaning close. In other words, with low gamma, points far away from plausible seperation line are considered in calculation for the seperation line. Where as high gamma means the points close to plausible line are considered in calculation.

- **Margin**

  And finally last but very importrant characteristic of SVM classifier. SVM to core tries to achieve a good margin. A margin is a separation of line to the closest class points. A good margin is one where this separation is larger for both the classes. Images below gives to visual example of good and bad margin. A good margin allows the points to be in their respective classes without crossing to other class.

In our cases we have two labels for our features, car and non-car. These features are then plotted and a decision boundary is drawn where they are best separated. The decision boundary is found by calculating the largest margin between the classes. The margin is calculated by drawing lines as close as possible to each class. Then the decision boundary is placed in the middle of the margin, equidistant from the margin lines drawn before. The

SVM can now be used to classify new sets of features by their location once they are arranged in the same plot.

Figure 6 illustrates how a support vector machine works. The circles and xs represent two distinct classes. The dotted lines represent the margin lines, and the diagonal solid line between the margin lines represents the decision boundary. New points on the plot can be classified by where they are located in respect to the decision boundary.

Sci-kit learn provides us with various SVM implementations. For our project we will use Linear Support Vector Classification (LinearSVC). We chose LinearSVC over other implementations because it is a simpler model that scales better when using large datasets.

2.3.3 Sliding Windows

Another technique used in this project is called sliding window search. This is useful in retrieving image segments from larger images. In our case we use this technique to identify cars in a dash-cam image. For example, we designate a window size and move the window throughout the image. This helps us retrieve the image encapsulated by the window which we can then feed to our classifier. The windows can have different sizes and overlap by a predetermined amount.

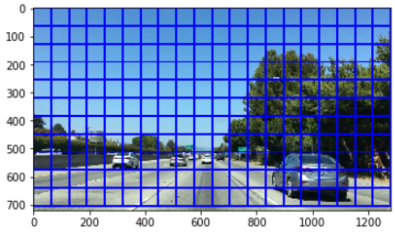
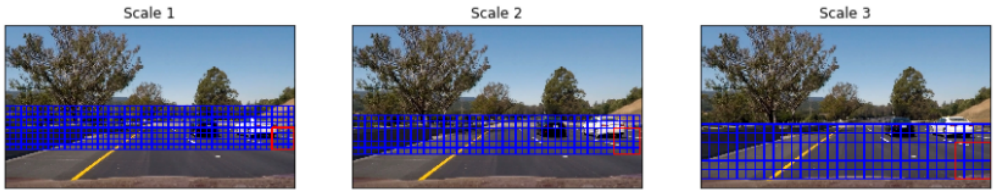


Figure illustrates a the result of plotting the sliding window onto an image. The window size is actually four times larger than the visible boxes because of the overlap between windows. Some overlap is useful to better identify and locate an object in an image. In

Figure 7 and in our implementation we use a 50% overlap in both the x and y directions. This means that the following windows will be placed half-way on the previous window. For example if our window size is 64 x 64 pixels the upper left corner of a window is placed in point (10, 300) the next upper left corner of the next window will be placed in point (42, 300) and so on for the following windows. The next row of windows will be placed similarly but half-way down the previous row, for example point (10, 332) for the upper left corner of the first window on the second row.



2.4 Benchmark

As a benchmark model we will use a report by Old Dominion University titled Exploring Image-Based Classification to Detect Vehicle Make and Model This report uses a algorithm pipeline very similar to the one proposed for this project. Hover the object of this report is slightly different. This report intends to identify different types of vehicles (cars, motorcycles, trucks, etc.) on a source video which is captured from a fixed location. In our case we are only identifying cars and our source video was captured from a moving vehicle itself. While the report used as a benchmark model is different to the project described in this proposal, the methodology is similar and therefore the results for the performance of the algorithms are comparable. The benchmark report can be found here: Link

3 Methodology

3.1 Data Processing

There is some pre processing needed for our algorithm to work properly, specially regarding the images. Before the images retrieved from the windows can be classified we must first resize them to be the same size as the images used to train the classifier (in this case 64 x 64 pixels). We must also ensure that the images only have 3 channels and if the have a fourth or alpha channel it must be removed before feeding the image to the feature extraction function.

3.2 Implementation

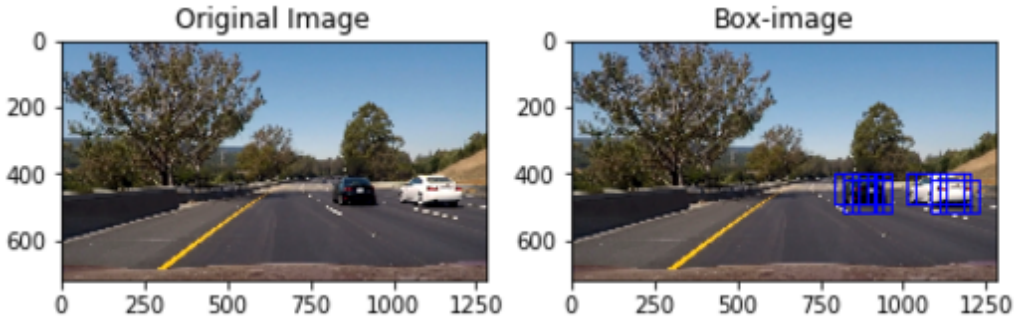
The whole project will be divided into following steps:

1. **Feature Extraction** : This step will provide detail of feature extraction. Proper image size will be selected. Various color chalels like: 'BGR','HSV','LUV','HLS','YUV','YCrCb' will be studied for better feature. HOG features and color features are the main part of the features.

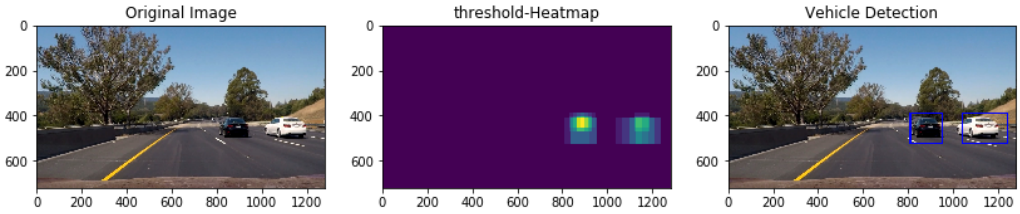
2. **Train a linear SVM**: This step will provide detail of training process, selection of parameters and theory of the model. Feature extracted from step 2 will be normalized and splited into train test part and will be fed in to the training process.

3. **Technique Slidding Window Search** : This step will discuss about the utilities used for search of vehicle in the different parts of an images. An image is segemented in to a smaller parts and some parts are given priority for searching vehicle. This results in a set of windows from which suspected part of the image is extracted and tested for presence of vehicle.

4. **Search and Classifiy an image** : This step uses all steps mentioned above, pretrained classifier model uses tiny section of the large image one-by-one through sliding window technique. If the classifier predict that tiny section as a car, a rectangle will be drawn around that section.



5. **Search and Classifiy a Video**: This is the final part of the project which prepares the video along with vehicle inside a rectangular box. Since video is a time series of many images, we use the step 5 in all images in video and reprocess them to form final output video. To be more precise in the result, we will use heatmap technique in vehicle detection to catch false positive cases as well.



3.3 Refinements

The most explicit refinement in this project is the addition of the Color features to the feature set, and the retraining of the classifier. The accuracy of the classifier on the training and testing data are the following: Train Accuracy of SVC = 1.0 Test Accuracy of SVC = 0.915625 When using both Color and HOG features the accuracy of the classifier increases: Train Accuracy of

SVC = 1.0 Test Accuracy of SVC = 0.9625 Using the extra set of features improves the testing accuracy by almost 5

4 Results

4.1 Model Evaluation and Validation

The final evaluation or the algorithm pipeline implemented in this project is done by review the results of the yielded by running a 8 second long dash-cam video through the algorithm. The resulting video is a copy of the original video but with bounding boxes drawn over the locations where cars have been identified by the classifier. In order to quantify the result we will use the following definition for accuracy.

$$\text{Accuracy} = \frac{(\text{true positives} + \text{true negatives})}{\text{size of dataset}} \tag{5}$$

In the above definition we will use the total number of window images to be classified to be the total population. True Positives are the window images classified as having cars (identified by a green bounding box) that actually have cars. True Negatives are the window images identified as not having cars. We calculate the true negatives by taking the total number of windows minus the number of true positives and false positives. This process had to be done by hand, reviewing each frame of the video. A spreadsheet containing the calculations has been included as part of the project submission. The total accuracy of the classifier on the source video is 96.973

4.2 Justification

Using our definition of accuracy is a good metric for our classifier since it accounts for for false positives which seem to be the biggest problem in our implementation. However the accuracy seems to be quite high because the

classifier does a decent job of identifying non-car images as such, which is overwhelmingly the case for our source video.

Our algorithm actually performs better than the benchmark model when identifying cars. The benchmark model had a 87

5 Conclusion

5.1 Free-form Visualization

The result of running the source video through the pipeline can be found here: Link Here is a single frame example:

In this single frame we can see that the algorithm correctly identifies the two cars on the right lanes and even two of the cars in the oncoming direction on the other side of the barrier. However there is quite a number of false positives around the barrier on the left side of the image.

5.2 Reflection

Overall the project is relatively successful. An accuracy of 97 unsafe. This project does illustrate how machine learning techniques using proper data are able to identify objects with reasonable accuracy. While most of the techniques used in this project are straightforward and have much documentation online, using them together was an interesting challenge. One problem I ran into early on was that I was getting extra extra features when performing feature extraction on new images. After attempting to debug the code I found that the code was not the problem but rather the image itself. The images used to train the classifier had 3 channels (Red, Green and Blue) but the new images had an extra alpha channel for transparency. The extra channel ment that the image yielded extra information when extracting features. This was fixed easily by only feeding the first 3 channels to the feature extraction function. Once the feature extraction was fixed, training the classifier went smoothly. After training the classifier, the next step was to extract image segments from a larger image, which is done with the sliding window technique. Identifying the proper window location was a bit of a hassle since the zero on the y-axis is visually on the upper left corner of the image which is normal in the computer vision field but somewhat unusual in math. After I wrapped my head around it and figured out the proper coordinated the next challenge was coming up the appropriate window sizes for our image frame, so that they would all fit properly while still covering most of the image. The other main difficulty when implementing the project solution was the time sink that is extracting features from thousands of images. Perhaps testing the solution on a server or a computer with more graphic processing capabilities would have made the testing and improvement of the project more timely.

5.3 Improvements

While the results are reasonably successful, there is much room for improvement. While the dataset used to train the classifier is reasonably large, an even larger one could yield some improvement in accuracy. Perhaps some images specific to lane barriers which is where our algorithm seems to have

the most trouble. Using more windows of different sizes could also be useful for identifying vehicles at the edge of the image. Other than the suggestions made above and perhaps some further parameter tuning, Considerable improvements could be done by using other techniques in conjunction to the ones showcased in this project. One such technique could be found in deep learning. Of Course if this were to be applied to a real life, safety-critical scenario, any detection rate lower than 100detection and accuracy would be to use a sensor suite as well as computer vision and perform some sensor fusion to better identify objects.

References

1. Object Detection Wikipedia
2. Human detection from images and videos: A survey
3. Machine Learinig Technique for Face Analaysis
4. Vehicle Detection in an Image
5. A Two-Stage Approach to People and VehicleDetection With HOG-Based SVM
6. Histograms of Oriented Gradients for Human Detection
7. character-recognization link
8. SVM scikit learn-link
9. SVM-review

10. Parameter tuning