Capstone Project Report
Machine Learning Engineer Nanodegree

Alberto Rivera
February 7, 2016

Definition

Project Overview

Properly identifying objects is a very important problem in computer vision. Our main interest for this project is identifying cars in a video taken from a “dash cam”. Being able to identify other vehicles 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 Udacity’s Self Driving Car Nanodegree. Most of the image data set used in this project is also used in that project for the purposes of training a machine learning classifier. The dataset was originally provided by the GTI (Grupo de Tratamiento de Imagenes) group at the Universidad Politécnica de Madrid. The original dataset can be found at the following link: https://www.gti.ssr.upm.es/data/Vehicle_database.html

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.

The source video used for the final result of this project is provided as part of the Advance Lane Finding project, which is also a part of Udacity’s Self Driving Car Nanodegree. However for our test we only use the last 201 frames (about 8 seconds). Using the whole video which is about a minute long would mean running the code for over 6 hours, and it is not necessary to quantify the performance of the algorithm implemented in this project. The original uncropped video can be found in the following link:
https://github.com/udacity/CarND-Advanced-Lane-Lines/blob/master/project_video.mp4

Problem Statement

The problem to be solved in this project is accurately identifying cars in a video. In order to achieve this multiple computer vision and machine learning methods are used in conjunction. In the implementation of this project images will be processed and represented by some of their features. In this project we will used a labeled image dataset. The images are separated by the presence of a car in the image or lack thereof. 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.

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.

Metrics

There are two metrics that we will used 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:

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

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.

Analysis

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.

Car Images
Figure 1

Non-car Images
Figure 2

After extracting the features from the images, they will be used to train a classifier. The classifier will then be able to distinguish between features sets that correspond to car images or to non-car images. These images prove extremely valuable because the predictions our algorithm will make are based on the features of the images used to train it.

No abnormalities were found in the dataset. However as mentioned in the previous section we do not use the entire dataset, only the first 4000 images from each class. This is done for a few reasons. Using the entire dataset seems to be overkill for our purposes. Using the whole dataset of around 8000 images for each class might make some browsers crash when using a Jupyter notebook and extracting features. For other approaches like deep learning, it might be more reasonable to use the entire dataset.

The image data set and the input video can be found at the following locations:

Vehicle images
https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/vehicles.zip

Non-vehicle images
https://s3.amazonaws.com/udacity-sdc/Vehicle_Tracking/non-vehicles.zip

Exploratory Visualization

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

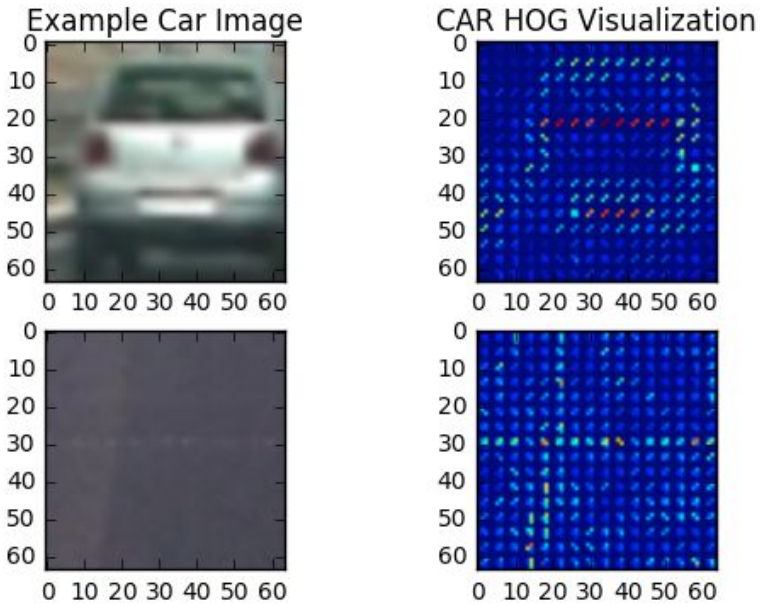


Figure 3

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.

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.

Figure 4

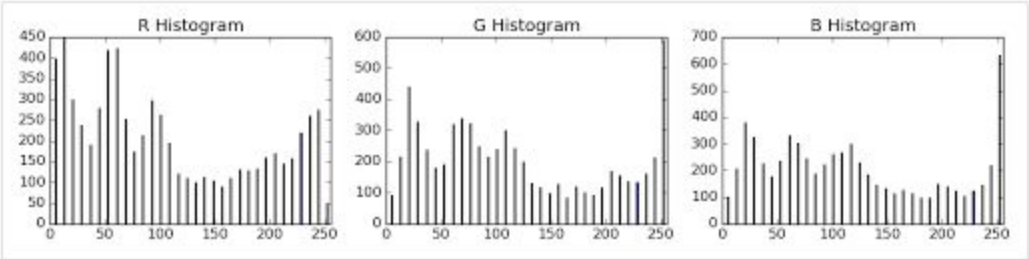


Figure 5

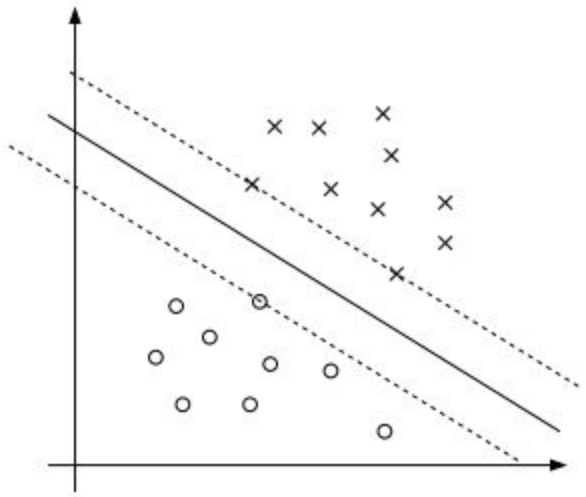
The resulting histograms can be used to compare new images and check if they match. If they do that could help classify the object in the image.

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 classifier.

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. 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.



http://cs229.stanford.edu/notes/cs229-notes3.pdf

Figure 6

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.

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.

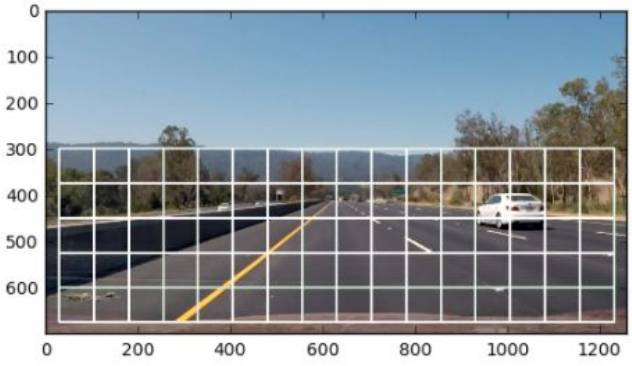


Figure 7

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.

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:
http://www.uidaho.edu/engr/research/niatt/tranlive/database/klk900-001-image

Methodology

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.

Implementation

The design of this project followed an incremental approach divided into seven parts, each building on the previous parts.

Part 1: HOG Feature Extraction
The first part showcases the implementation of a Histogram of Gradients (HOG) Feature extraction. Here we use a test image as input in order to demonstrate how it can be converted into a set of features and show a visual representation of the

features. Then the same feature extraction will be done for all the images in the data set.

Part 2: Train a Linear SVM classifier
Using the features extracted from the car and non-car datasets in the previous part we train a Support Vector Machine (SVM) classifier. Then the accuracy of the classifier

Part 3: Add Color Features
In this part further features will be extracted. In the features come from the color of the image rather than the gradient. Features from all of the images in the dataset will be extracted and combined with the HOG features.

Part 4: Re-Train the Linear SVM classifier
The classifier from Part 2 is now retrained with the combined HOG and Color features.

Part 5: Sliding Window Search
This part focuses on segmenting a large image into smaller more practical images in order to search for vehicles. This will result in a set of windows from which images will be extracted to then be classified.

Part 6: Search and Classify an Image
Here all the previous parts come together to achieve vehicle detection in a single image. This image will be a single frame of the “dash cam” source video. A sliding window search will be used to break down the image into more manageable images than will then be resized in order to be classified properly. If the classifier predicts that there is in fact a car in the image a bounding box will be drawn around the car.

Part 7: Search and Classify a Video
This last part is very similar to Part 6, with the exception that the the search and classification will be applied to each frame of the source video and the result will be a copy of the source video but with the vehicles identified by bounding boxes.

Refinement

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%. This increase is quite considerable and greatly improves the overall performance of our algorithm.

Results

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.

$$\text{Accuracy} = (\text{True Positives} + \text{True Negatives}) / \text{Total Population}$$

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%. The accuracy is slightly higher than that of the classifier on the training data, which is quite remarkable. This might be because the video only features a single car for most of its duration, and it is easily identifiable by our classifier. The video result has been included with the submission of this project.

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% accuracy when using a similar feature set, and a 92% accuracy when not using HOG features. Our model outperforms all the results presented in the benchmark models. However we must take into consideration that the source videos are not the same and that the benchmark models attempts to further classify the vehicles into sub-categories rather than just “car”.

Conclusion

Freeform Visualization

The result of running the source video through the pipeline can be found here: https://youtu.be/4I0X4tK3uIw

Here is a single frame example:

Figure 8

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.

Reflection

Overall the project is relatively successful. An accuracy of 97 % is quite high for our pipeline. However for practical applications 97% accuracy is not acceptable. Imagine a self driving car that could only see 97% of the vehicles around it. It would be quite

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.

Improvement

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 100% would be unacceptable. The way to achieve near perfect

detection and accuracy would be to use a sensor suite as well as computer vision and perform some sensor fusion to better identify objects.