

Vehicle Detection Project Report

Jakrin Juangbhanich

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labeled training set of images and train a classifier Linear SVM classifier.
- Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
- Run your pipeline on a video stream (start with the test_video.mp4 and later implement on full project_video.mp4) and create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- Estimate a bounding box for vehicles detected.

Rubric Points

README

Provide a Writeup / README that includes all the rubric points and how you addressed each one. You can submit your writeup as markdown or pdf. Here is a template writeup for this project you can use as a guide and a starting point.

This is my project report for the final project of Term 1: Vehicle detection. I found this project really difficult, but I was able to apply some of the techniques I used and learnt from the previous projects to advance my progress.


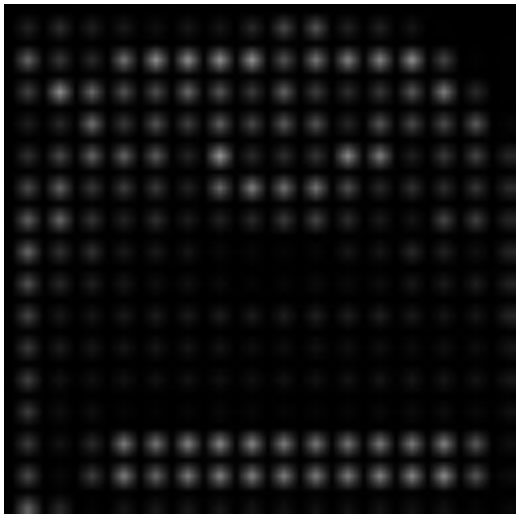

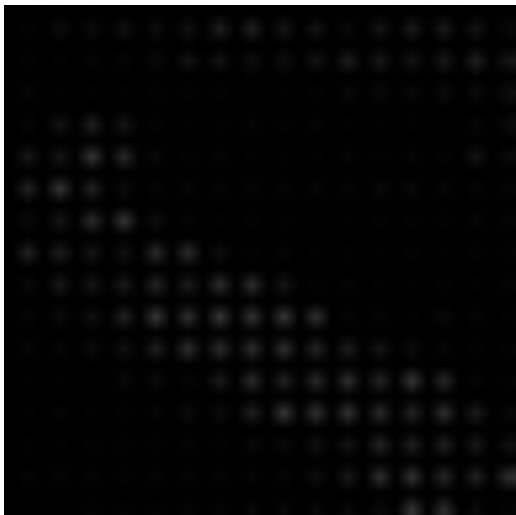
I set up a pipeline for both training and predicting the image data. The pipeline consists of several layers (inspired by how keras worked), where each layer feeds its output into the next, until the end is reached. Each layer also exports a sample image of its work. The layers each contain custom logic to perform on the image input.

Histogram of Oriented Gradients (HOG)

Explain how (and identify where in your code) you extracted HOG features from

the training images. Explain how you settled on your final choice of HOG parameters.

Hog features were extracted in **hog.py**. I turned the input image (scaled down to 32x32) to grayscale, and then used a 6 orientation, 2x2 pixel per cells, 2x2 cells per block as parameters.

Original Image	HOG Features
 Car	 Car HOG
 Non Car	 Non Car HOG

Explain how you settled on your final choice of HOG parameters.

I scaled down to a smaller size first because it improved the speed of the algorithm significantly, and I didn't feel I would lose much accuracy. At first I tried using 4 px per cell, but I also found that using 2 px for cell gave me greater prediction scores. From the output, I was able to see a good distinction between car and non-car features. The cars seemed to have a well defined outline, especially at the top and the bottom. I was hoping that would be a good aspect to train the classifier on.


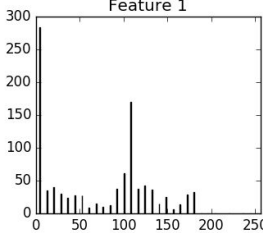
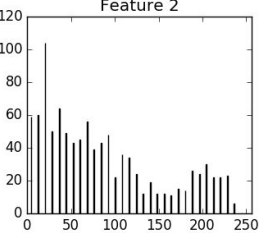
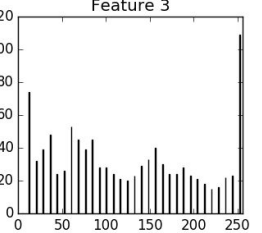
Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

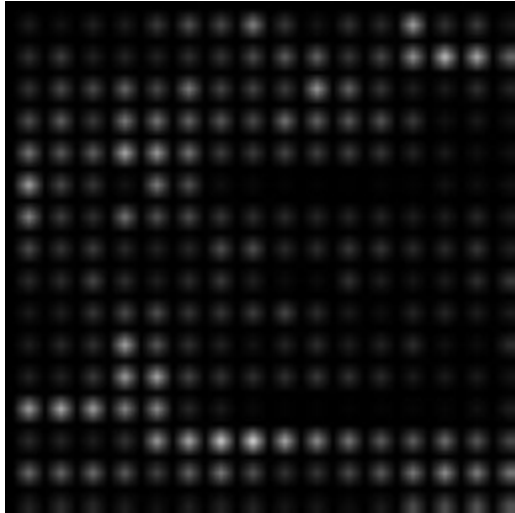

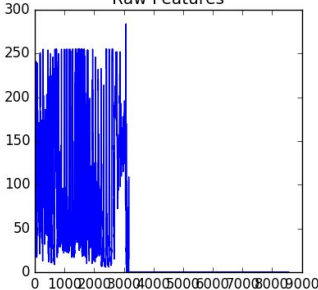
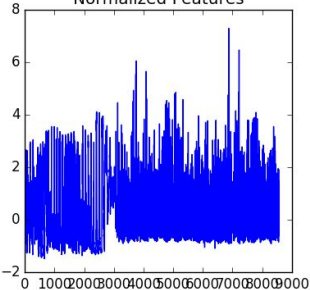
I trained a linear SVC (sklearn's LinearSVC) in classify.py, using max iterations of 1000 and C value of 2.0. The default C value of 1.0 was underfitting, causing a lot of false positives when I was testing it. I tried C values of 1.5 and 2.5 as well, it seemed that 2.0 gave me the best results.

For the feature vector, I combined the HOG features (hog.py) with color histograms (colorhistogram.py) (HSV) and spatial binning features (spatialbin.py). The features are joined and normalized in featuremodel.py and normalize.py respectively, and then fed into the classify.py (which is a wrapper I created for a classifier in case I wanted to change it to a different one later).

The classifier gave a test accuracy of around 97%, which looked good enough for me to move onto the next step. I definitely know it can be improved though because other students have reported scores of 99%.

Here are some visualizations of the training pipeline:

Spatial Bin	
Color Histogram with HSV	<div><div>Feature 1</div><div>Feature 2</div><div>Feature 3</div></div>

HOG Features	
Normalized Features	<div>Original Image</div>  <div>Raw Features</div>  <div>Normalized Features</div> 

Sliding Window Search

Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

I discovered, while running this on test images and videos, that a good sliding window implementation is absolutely key for the algorithm to work. The windows must be at the right size and the right step to get good input for the classifier.

My sliding window is implemented in windows.py. I started with the implementation offered by the lectures, but I wanted more flexibility. A fixed window size didn't allow me to track vehicles at different distances from the camera.

I scraped that implementation and created a new one where the windows will scale, from a small starting size, all the way up to a factor of 2.5x the original size as it gets near to the bottom of the image.

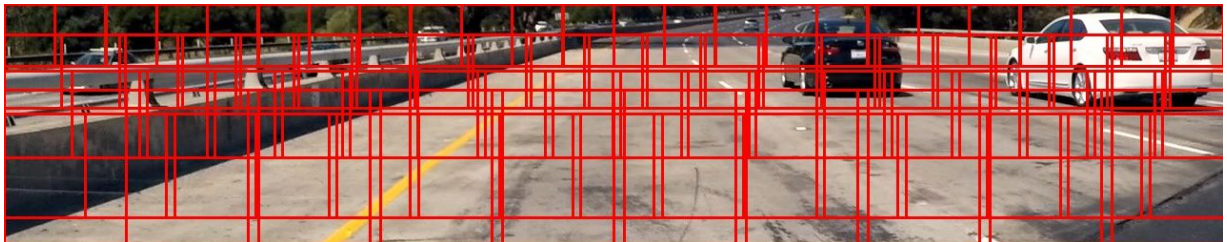
I also changed the windows to be wider, because I thought it made sense that cars usually occupy a rectangular space, not a square.

Here are my parameters for the sliding window:

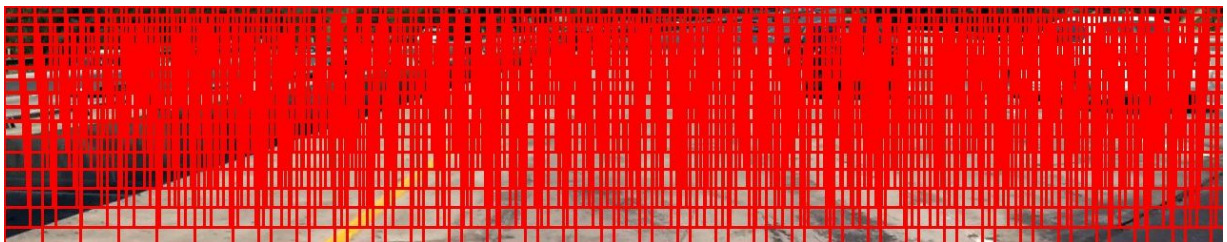
Window Starting Height	65 px
Window X Overlap	85%
Window Y Overlap	85%
Window Max Scale Factor	250%
Window Aspect Ratio	1.65

Here are two pictures to demonstrate the scaling factor of the windows (using a 0.5 overlap) and the actual windows used with a higher overlap.

Scaling Demonstration



Actual Windows Used



A high overlap is helpful because it gives me a lot of positive identification for vehicles at all positions in the frame. So I can much more easily reject the false positives.

Show some examples of test images to demonstrate how your pipeline is working

My search pipeline can be seen in main.py from line 75. The layers consisted of:

Crop: Cropping the image to a smaller region of interest.

Windows: Create the sliding windows.

Identify: Run the classifier on each window.

Heatmap: Positive windows are added to the heat map. Hot areas with 3 points or less, or are too small, are discarded.

Detect: Positive heatmap labels are then added to the 'search model,' which tried to work out whether it is part of a previously identified vehicle, or if it is a new one. Each positive ID will add a confidence score to that tracker. Trackers with less than a certain score will be ignored. The bounding box is estimated based on a width around the heat map detection.

Here is a visualization from each step of the pipeline:

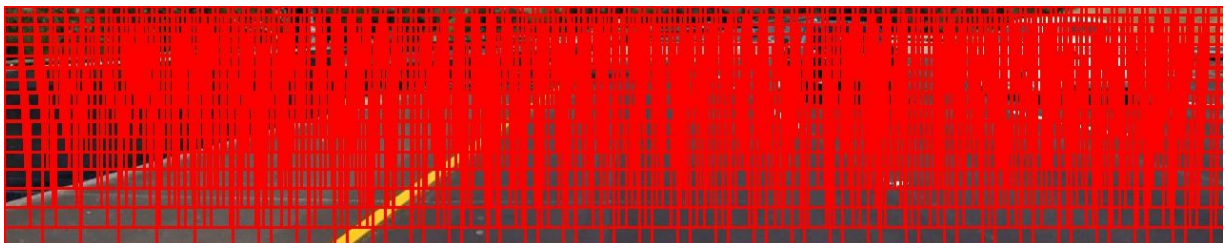
Original Image



Cropping Layer



Windows Layer



Identify Layer



Heat Map Layer

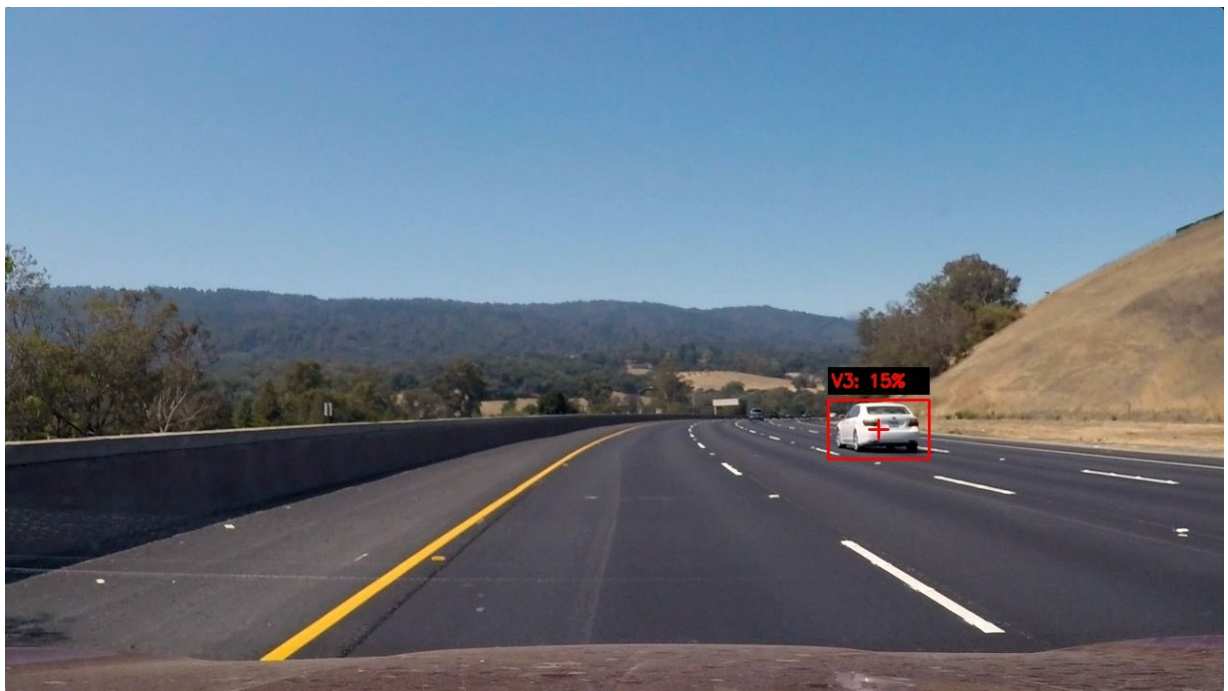


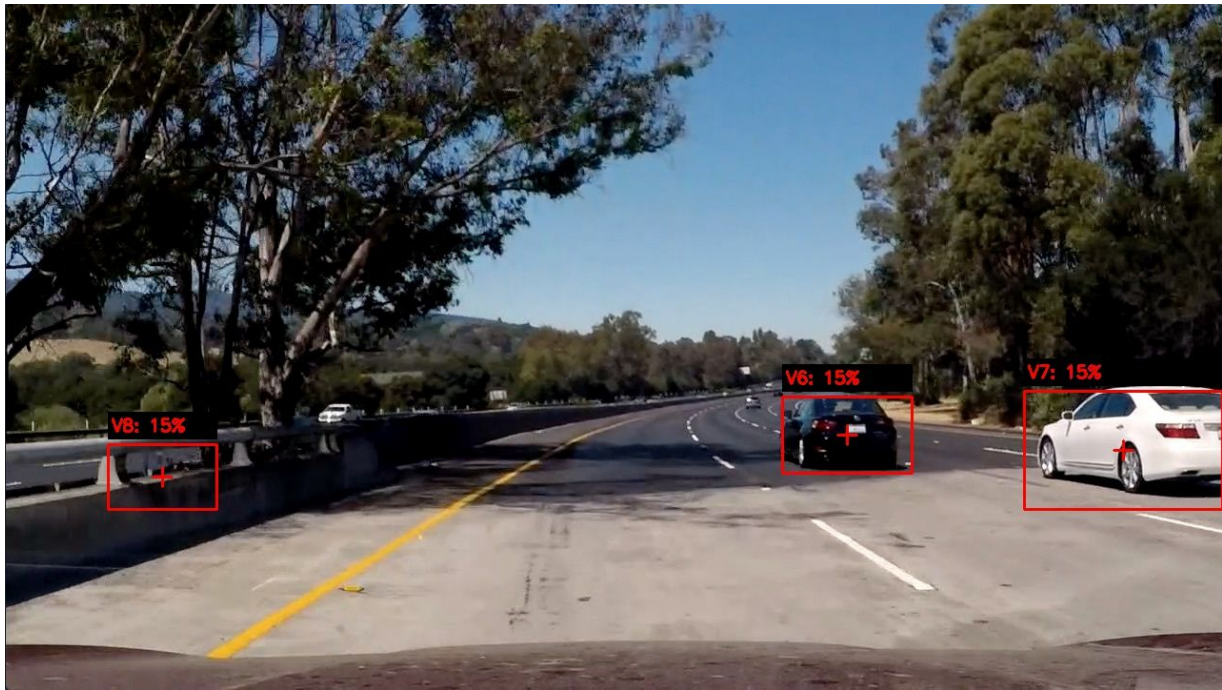
Detection Layer



Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

My classifier used a spatial binning, HSV channel histograms, and a grayscale HOG as its features. In combination with the sliding windows and detection tracking, I felt it was performing quite strongly. Here are some examples:





There are some false positives, but that is mostly filtered out in the detection and tracking step of my pipeline.

Video Pipeline

Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

Here is a link to my video result:

Project Video Output

Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

The detection layer can be seen in detect.py. It first takes the heatmap from the previous layer as input, and then uses scipy label to find the 'islands' of positive IDs. Then I found the center point of these labels, and used that to track the vehicle. I drew a rectangular bounding box on that area, based on the width.

The logic for tracking and smoothing vehicle detections can be seen in statemodel.py.

Each detection creates or contributes to a 'Car Point' which does the following things:

- Adds confidences to a detection within a certain radius from frame to frame.
- Loses confidence every frame without a detection.
- Below 35% confidence, will not show up in the video.
- Below 50% confidence, will show up as red.
- Below 85% confidence will show up as yellow.
- Over 85% will show up as green. We can be quite certain a vehicle is being tracked here.
- Estimates the bounding box size based on the label's width.
- Smoothly interpolates the position and size between frames.

Discussion

Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

Even though I feel like I've come far, and the pipeline seems to work reasonably well, I know that virtually every aspect of the pipeline has much room for improvement. Here are some problems and what I think about them:

- The pipeline is too slow. It took me 30 minutes to process a 1 minute video. I read up about other detection techniques (such as CNN and YOLO) which can produce faster results. Also, I was not able to subsample the HOG with my approach, and I had a LOT of sliding windows, which I'm sure slowed me down incredibly.
- The classifier struggled with the dark car when it was big in the frame. I'm not sure whether this was a sliding window problem or a classifier problem (it seems to detect the white car fine when it was close, but also detects the black car fine when it was further away).
- It struggles to handle the case when the cars were very close to each other. I thought about a way to fix this, by splitting my heatmap labels into two different regions if I detect that the width to height ratio was unreasonable, I can maybe assume that it is two vehicles being detected.

On a positive note, I felt that the false positive rejection, the bounding box estimation, and the smoothing of the tracking worked well.