Goal

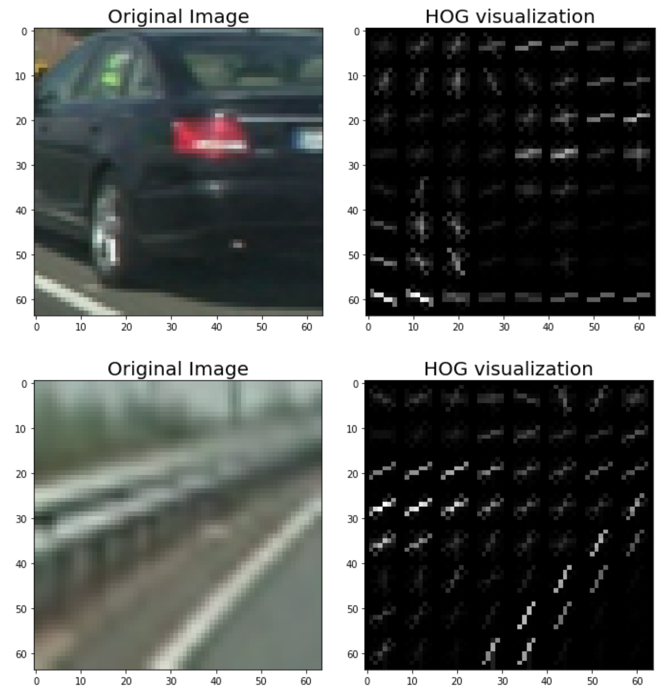
The goal of this project is to identify and draw bounding boxes around vehicles in each frame for a given video stream.

Method

As an overview of the method I used, I first trained an SVM classifier on the provided dataset to classify vehicle images vs. non-vehicle images using HOG and color features. Then I slid search windows of different sizes across the frame to detect vehicles. To enhance the result and speed, I also mixed in a nearest neighbor method to find matching patches to identified vehicles in previous frames. In the following sections, I will go through each step in detail.

HOG feature

I extracted the histogram-of-gradient (HOG) features on the provided data set (8792 vehicle images and 8963 non vehicle road images). This feature computes the orientation of the gradient at all locations and sums up the gradient vectors in each block. I first converted the color image into default gray scale, and used the following parameters: 8 pixels per cell, 2 cells per block and 9 orientations.



I scaled the features using sklearn.preprocessing.StandardScaler().

Color histogram features

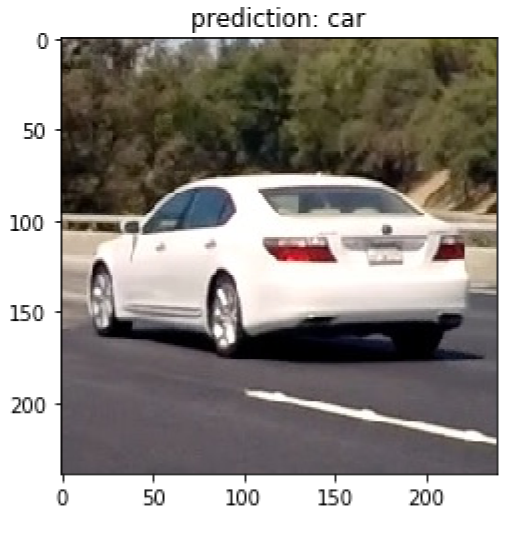
Hog features alone perform very well, but it errors in some edge cases. To correct for this, I added color histogram features. This feature looks at the value of red, green, blue channels respectively, and bins pixel values of each channel into 32 bins. I then concatenated these 3 vectors of length 32 each into a 96-dimensional vector. I applied a separate scaler to scale this feature.

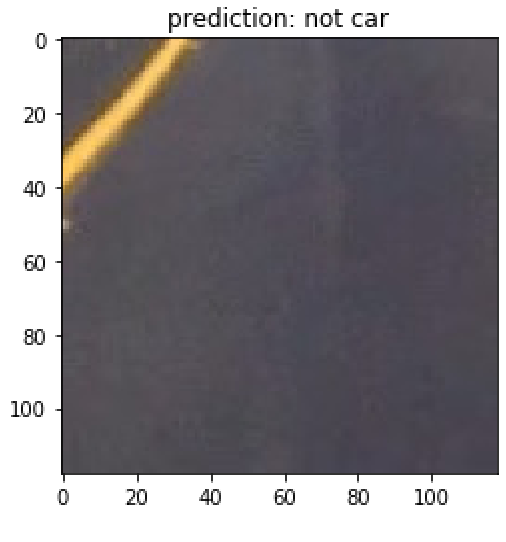
SVM Classifier

I trained an SVM classifier on the scaled features and randomly split 10% of the data as test data. I first just used the HOG feature. There are two options of normalization when generating the HOG feature: L1 norm and L2 norm. When using L1 norm, accuracy on training set is 0.998 and accuracy on test set is 0.987. When using L2 norm, accuracy on training set is 0.924 and accuracy on test set is 0.917. Hence, I chose L1 norm.

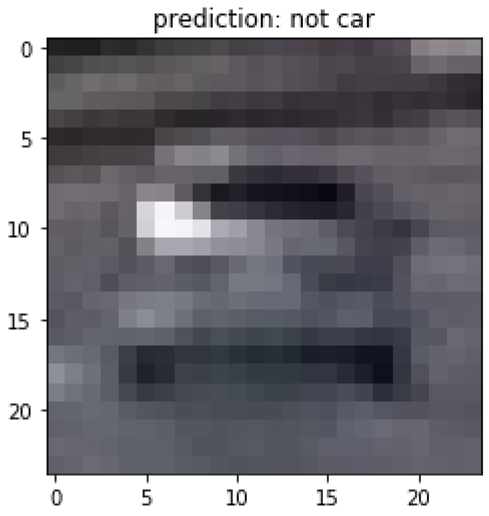
To qualitatively evaluate the SVM classifier on HOG feature, I cropped some square images from the test video and resized them to 64x64 which is the size of the images in the labeled data set. See the four examples below as output of the classifier. The classifier works quite well on obvious cases. It doesn't perform well in two cases: a) when the car is very small (or far in the horizon) and the pixels are blurred, b) side of a car. For the latter, I realized the labelled dataset of vehicles contains almost exclusively images of rears of vehicles, and few, if not none, images of side of the vehicles.

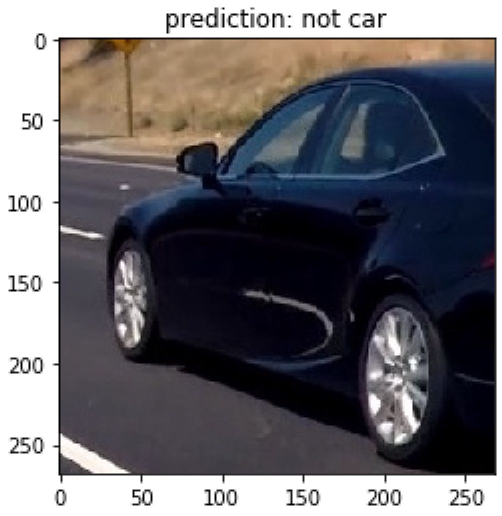
Correct predictions:





Incorrect predictions:





I then added the color histogram feature as well. The SVM classifier trained on the concatenated color and HOG feature shows an accuracy improvement on the test data from 0.9870 to 0.9898. Though the numerical improvement on the labelled data set is small, qualitatively, in this case below, SVM on HOG feature alone falsely classified the highway rail as a car, whereas SVM on HOG feature combined with color histogram features correctly reject the area as a car image.

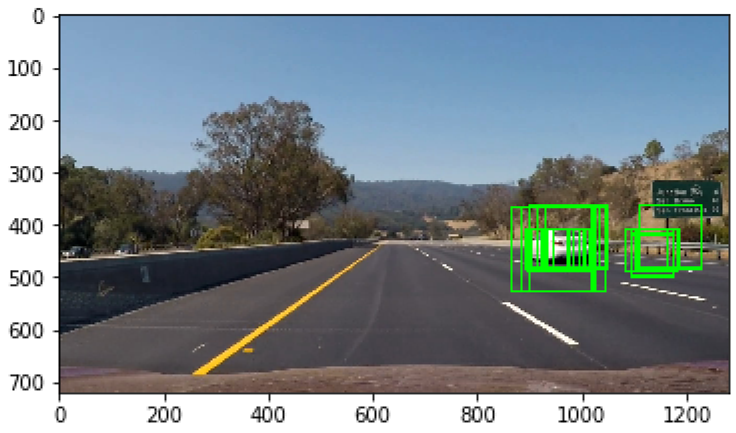


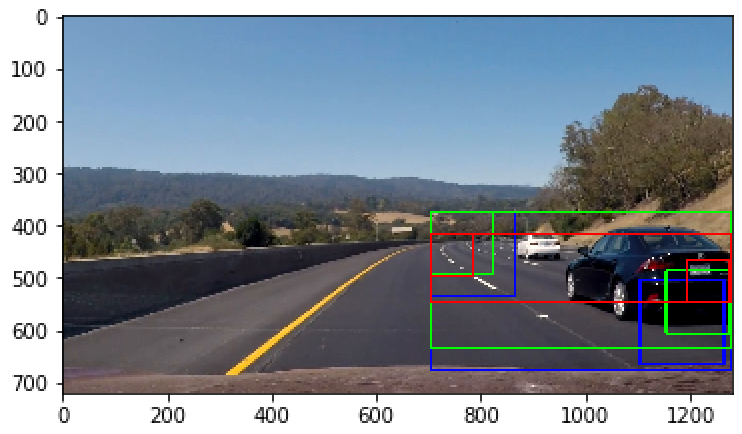
Fig: prediction by SVM with HOG feature alone.

<u>Sliding window search</u>

To identify vehicles in a video frame, first, for speed reason, I cropped the frame to the lower right corner as area of interest. This is the only area where vehicles can possibly appear when driving in the leftmost lane, as it is in the project video. One may argue this is not generalizable;

in practice, one can simply expand the search area to the entire frame (or the lower half of the frame).

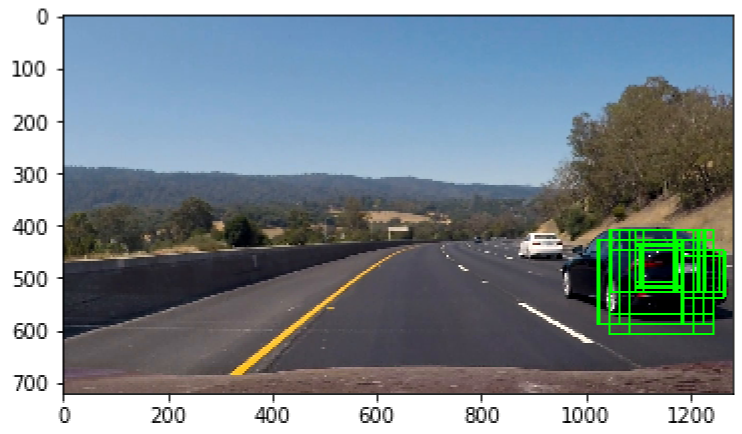
We will slide windows of various sizes over the frame. The choices of the window sizes and stride greatly affect speed and accuracy, and I spent quite some time tuning it.



The image on the left illustrates the sliding windows I chose. I used windows of three different sizes. The largest one, 160 pixels per side, strides over the entire area of interest. Second in size is a window of 120 pixels. Lastly, the smallest window, 60 pixels per side, slides over only the upper middle strip, as it is unlikely that cars of that size will appear lower (closer).

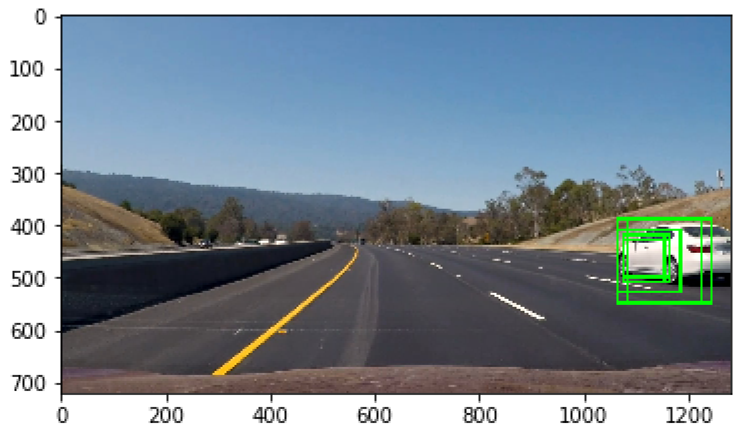
Computing hog features is slow, and much of it is redundant work, as a sliding window overlaps from a bit with its previous window (say, a few pixels to the left). So another speed optimization I did was to compute the hog features of the entire frame only once for a window size, and subsample hog features for each window from overall hog feature vectors.

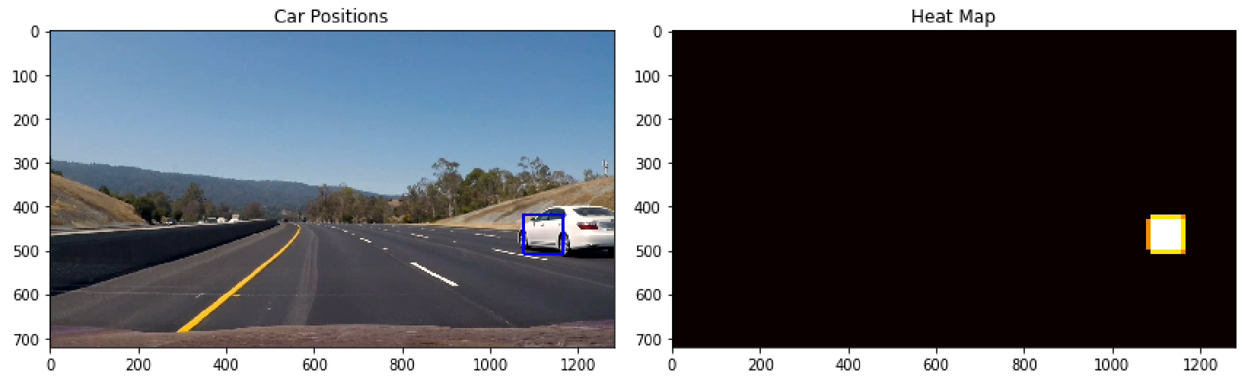
Below is the output after the sliding window search. It takes about ~3-5 seconds per frame to search.



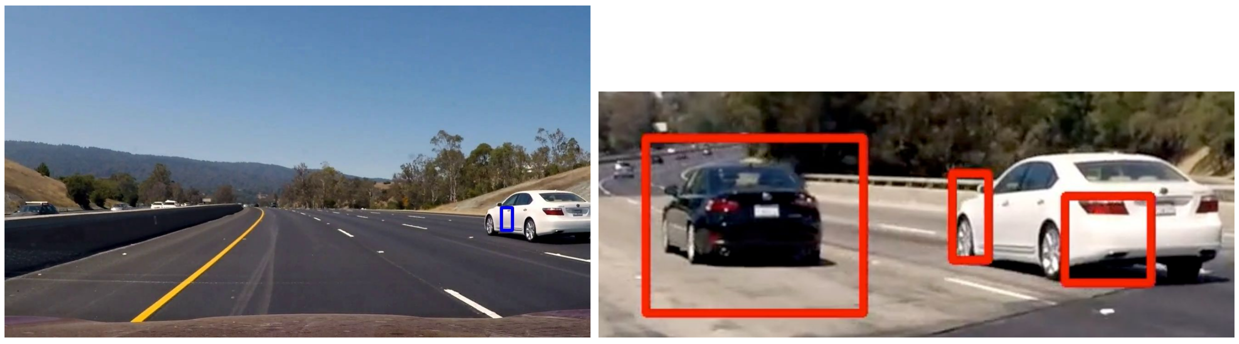
<u>Heatmap</u>

To integrate the output of the sliding window search into one bounding box per car, I used heat map. For each box drawn over a pixel, I added a heat of one to that pixel. When the heat is above certain threshold, then that area would be considered as pixels of a vehicle image. With the settings of my search windows, I tuned the threshold to be 3. I then found the upperleft-most and lowerright-most corners of an isolated patch of the qualified area of the heatmap, and drew a bounding box around it as the final output.





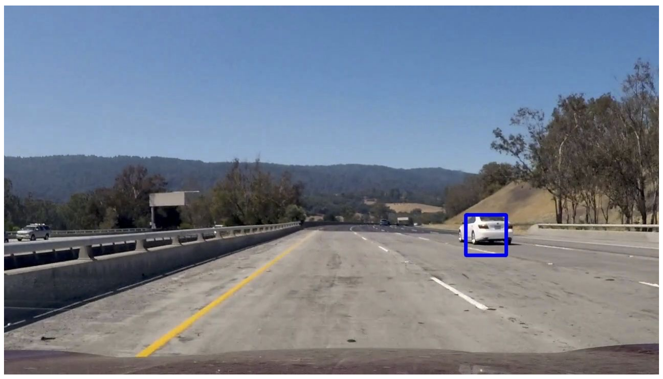
In practice, I also set a width : height threshold of 1:2 for a qualifying bounding box. Boxes more elongated than that are unlikely shapes of vehicles. This prevented false positive cases like below.



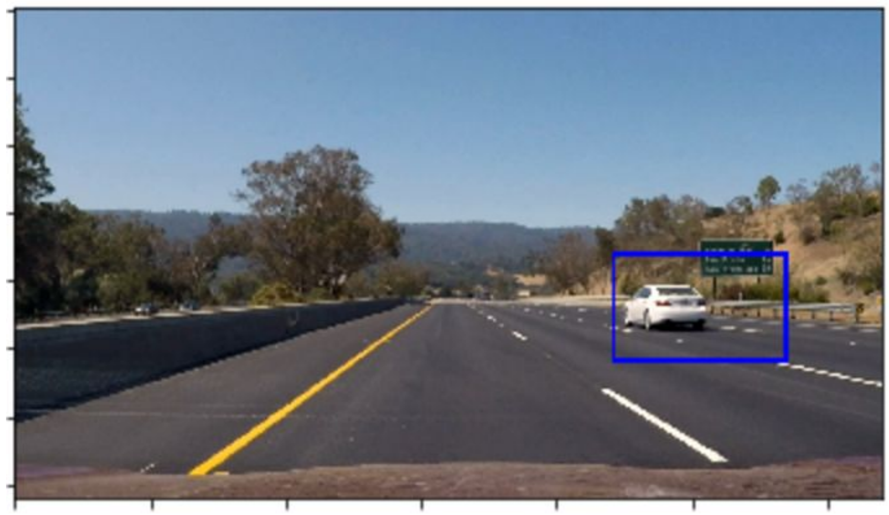
Nearest Neighbor Matching on Saturated Area

The method described so far works fine on individual frames. However, when we process frames from videos, this method has three issues: a) It can correctly classify a car in a previous frame but falsely misses out the same car in a later similar looking frame. b) Bounding boxes from frame to frame, even if all contain the vehicle, vary in shapes and sizes and hence are not stable, c) It is slow to compute each frame.

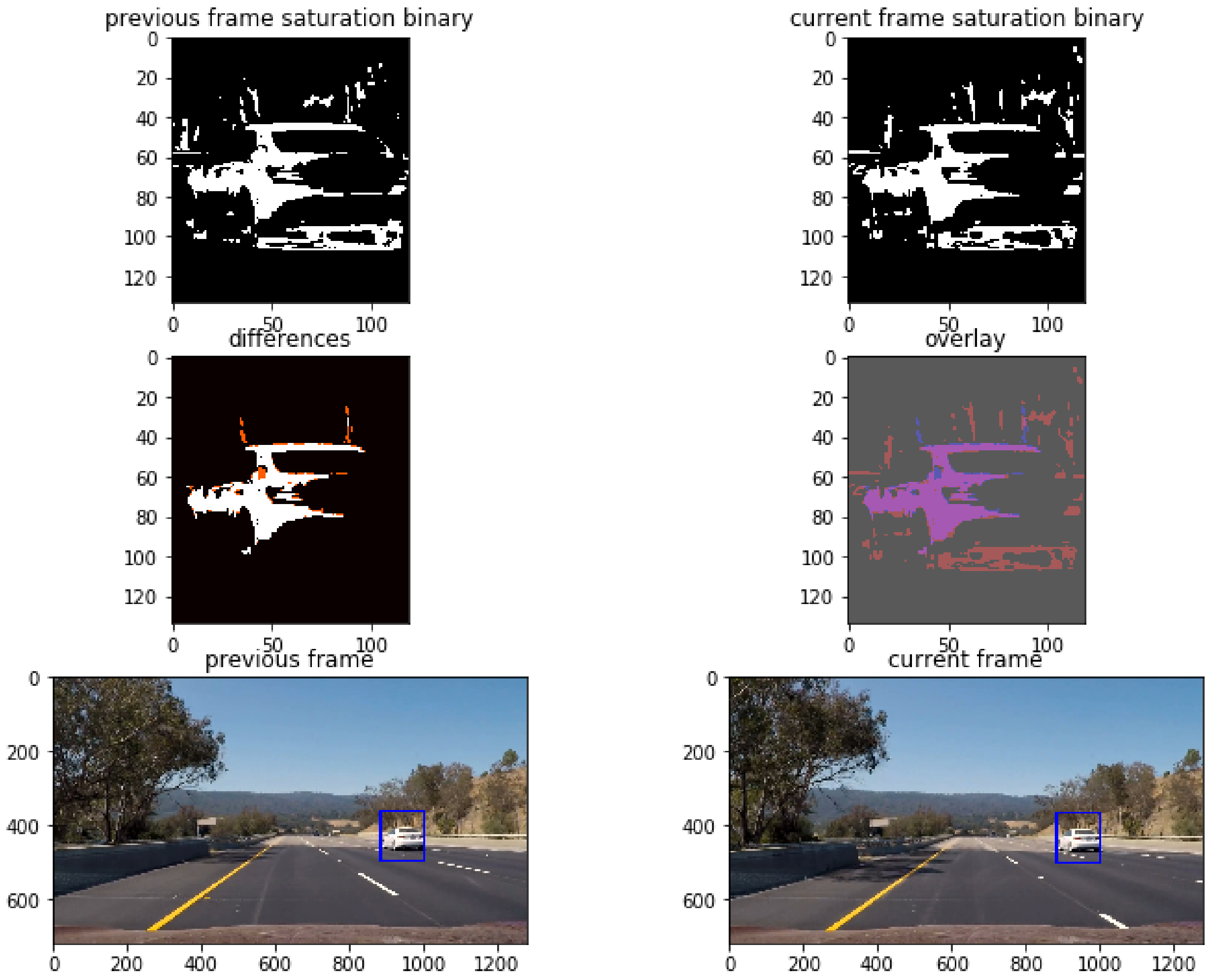
For a video stream, we should take advantage of the fact that adjacent frames look similar. So instead of classifying from scratch, I used a nearest neighbor method. For example, in this frame the SVM classifier found a car, so we cropped out that area. In subsequent frames, we can look for area that look most similar to this patch.



There are different ways to define similarity. The most naive way is to compute the squared sum differences of each pixel. I found this method not robust. If the bounding box contains a large area of road, like the example below, then its squared sum difference from any area that contains large area of road will be small, as road pixels look similar to each other.



Instead, we care whether the *car* in the two patches look similar. Applying learning from the lane detection project, road pixels are not saturated, whereas cars are usually of saturated colors. So I converted the patch to hue-saturation-value (HSV) space, and took the saturation channel. I converted the saturation channel to a binary image thresholding at saturation value of 100. Then I found the largest consecutive saturated area and assumed that to be car. This empirically works quite well. To conclude two patches are of the same car, they need to have significant (>50%) overlap in the saturated car pixels. We do this search in the adjacent area of where we found the car in the previous frame, and the window with the largest overlap wins.



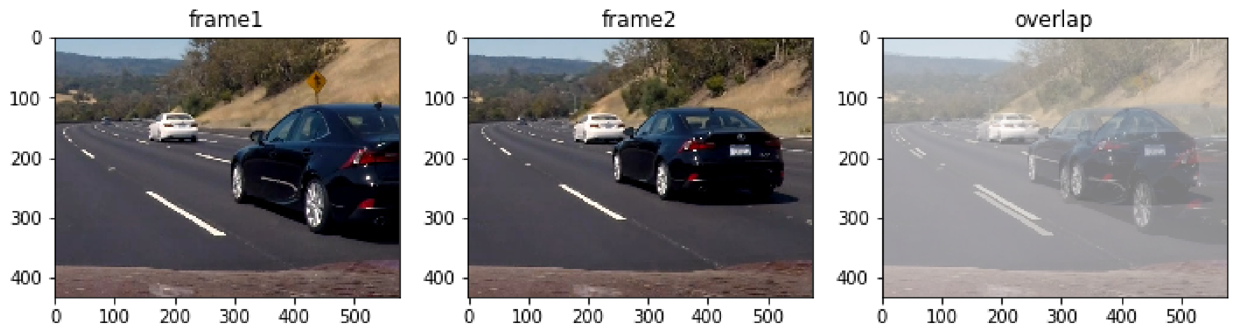
As a caveat, it is important to always compare against the original patch identified from the SVM classifier many frames ago, instead of the nearest neighbor output patch from the most recent frame. Otherwise, it is easy to propagate errors.

Detect substantially different frames

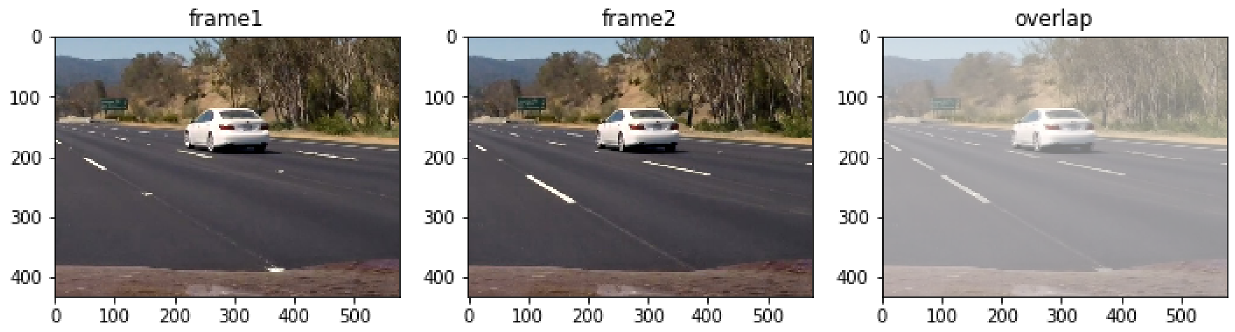
One danger of keeping using the nearest neighbor tracking only is that when a new vehicle enters the view, while nearest neighbor tracks down the existing car, the SVM classifier is not activated to detect the new car.

When should we use the SVM classifier again? To answer this, we do a pixel to pixel comparison between the current frame and the last frame we ran the classifier on using squared sum differences of the pixel and set a threshold.

For example, the two frames below (overlaid on each other) are substantially different (squared sum of difference is above the threshold) such that it warrants re-activation of the SVM classifier:



In contrast, these two frames look similar and do not need to re-run the classifier.



Skipping frames

As a last speed optimization trick, we only analyze every 5 frames. For the 4 out of the 5 frames, we simply blindly copy the location of the boxes from the first frame over. There are 60 frames per second, and frame to frame variability is very little. This optimization works very well and does not compromise any accuracy. It speeds up the pipeline by 5x.

Video pipeline summary

To summarize, in the video pipeline, we follow this procedure:

- Classify only every other 5 frames.
    The subsequent 4 frames directly copy the result from the last classified frame.

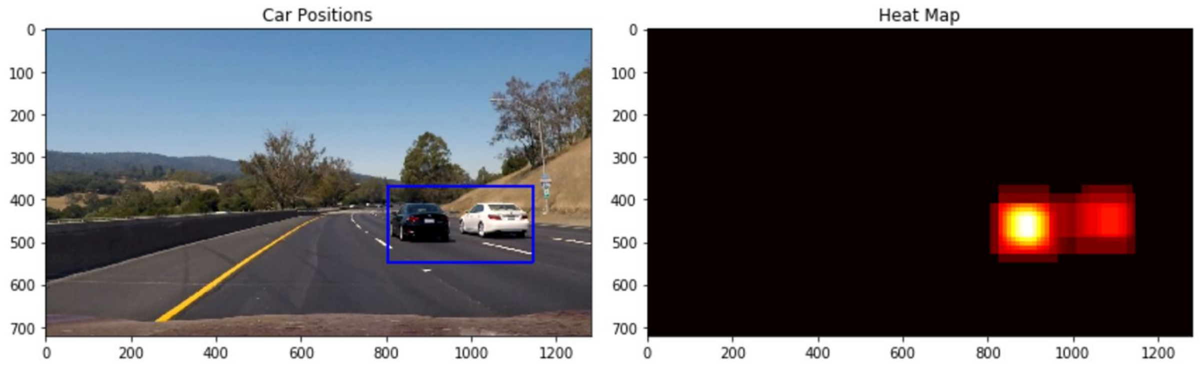
- Go through the SVM classifier if satisfying any of the condition:
    a) first frame or previous frame does not have identified car yet
    b) OR current frame is substantially different from the last classified frame
    c) OR nearest neighbour method returns fewer identified vehicles then the previous frame

    * in condition b) and c), we will do a limited search,
      avoiding the vehicles already identified from nearest neighbour
    ** if new vehicles are identified in the limited search, then re-search the omitted area for
      for possible merges (a newly classified area could just be another part of the same car
from the
      nearest neighbor search)

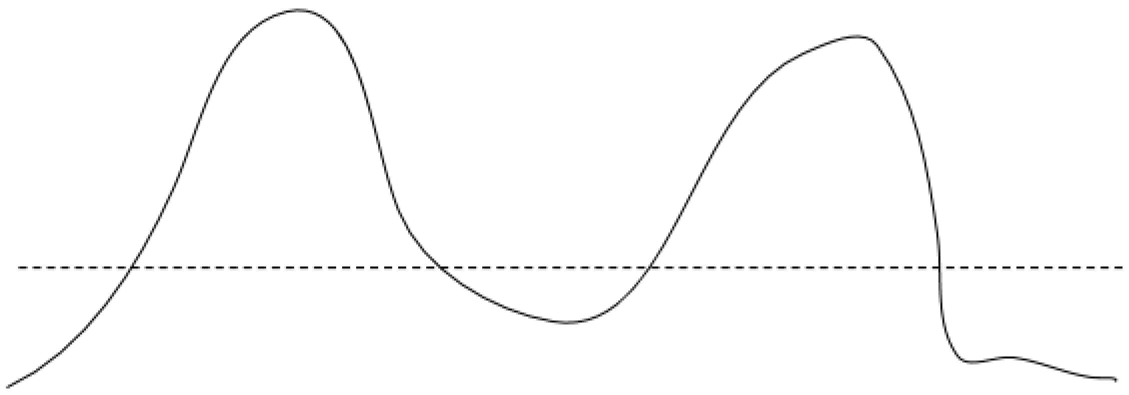
- Otherwise, use nearest neighbor search to save classification time

Discussion

Using a combination of a variety techniques, the output video shows desired results. One area of improvement is when there are two vehicles close to each other in the frame, the algorithm doesn't effectively separate them into two boxes.



To human eyes, the heatmap clearly has a valley in between. I attempted to address this issue in the following way: for each identified box, we increase the heat threshold and see if they separate into two areas. An analogy for this method is to imagine two neighboring mountains. If we increase the sea level and if there is a valley in between, then at some point the mountains will be two separate islands.



Unfortunately, this method addresses the problem in the frame above but causes false separation in other case. Hence I abandoned the approach in the final output.