# Histogram of Oriented Gradients (HOG)

## 1. Explain how (and identify where in your code) you extracted HOG features from the training images.

Code reference: `classifier.py`

I use `skimage.feature.hog` to extract hog features from input images, in a very standard way as we see in the lecture videos.

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

Code reference: `classifier.py`

I extract HOG features on all YUV channels. I use YUV because I find it has better prediction accuracy than RGB model (98% vs 96%). I did have the concern that using all 3 channels will slow the training and prediction process, but it turns out that the performance isn't that bad. In the beginning I tried use GRAY channel only, but the accuracy is only about 95% so I didn't use it.

For other parameters like `orientation`, `pixel_per_cell`, `cell_per_block`, I use convention value and I don't feel much of the need to tune it as it performs already very well.

Here's the final parameters for HOG:

```
pixel_per_cell = 8
cell_per_block = 2
orientation = 9
```

```
cspace = 'YUV'
hog_channel = 'ALL'
```

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

Code reference: `classifier.py`

I use `sklearn.svm.LinearSVC` as my implmenetation of SVM classifier. I concatenate all hog features extracted from 3 YUV channels, and flatten them ( `ravel` ) into a 1-dimensional vector. In the end, every image has a training data that is a vector of length 7x7x2x2x9x3 = 5292, and the label for every image is just 0 or 1 indicating it's car or non-car.

I use `sklearn.model_selection.train_test_split` to split 20% of my training data into test data. I also did random shuffling to both my training and test data set. Then I fit the data in the trainer and I get about 98.5% in terms of testing accuracy.

## Sliding Window Search

### 1. 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?

Code reference: `detector.py`

Here's how I scan the image:

- I use different scales of window size: 64px(scale 1.0), 80px(scale 1.25), 96px(scale 1.5), 128px(scale 2.0), 192px(scale 3.0). Using different scale can generate both bigger or smaller bounding boxes.
- I move the sliding window so that it covers 75% of the previous window on the axis it moves. Having overlap will increase the offset precision of the bounding boxes.
- I only calculate the HOG features once for each scale. I followed the instructions in the lecture to first calculate the HOG feature multi-dimensional matrix for each scale, then I can just pickup the sub-matrix that corresponds to the current window position. Besides, I also implemented a brute-force (slower) version so I can have a reference of how my faster algorithm works. They generate the same bounding box but the fast version (only calculate HOG features once) is about 3~5 times faster.
- For different scale (window size), I scan different areas. For smaller window, I scan location on further away road. For bigger window, I scan bigger area.

Scale 1.0, window size 64:

Scale 1.25, window size 80:

Scale 1.5, window size 96:

Scale 2.0, window size 128:

Scale 3.0, window size 192:

## 2. Show some examples of test images to demonstrate how

## your pipeline is working. What did you do to optimize the performance of your classifier?

Here's some test images:


NEXT
15 MILES

Like I mentioned earlier, I used GRAY channel at the first time, but it too many false positives. After switching to YUV all channels it performs much better.

# Video Implementation

## 1. 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.)

Please see `output_project_video.mp4` in my submission.

## 2. 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.

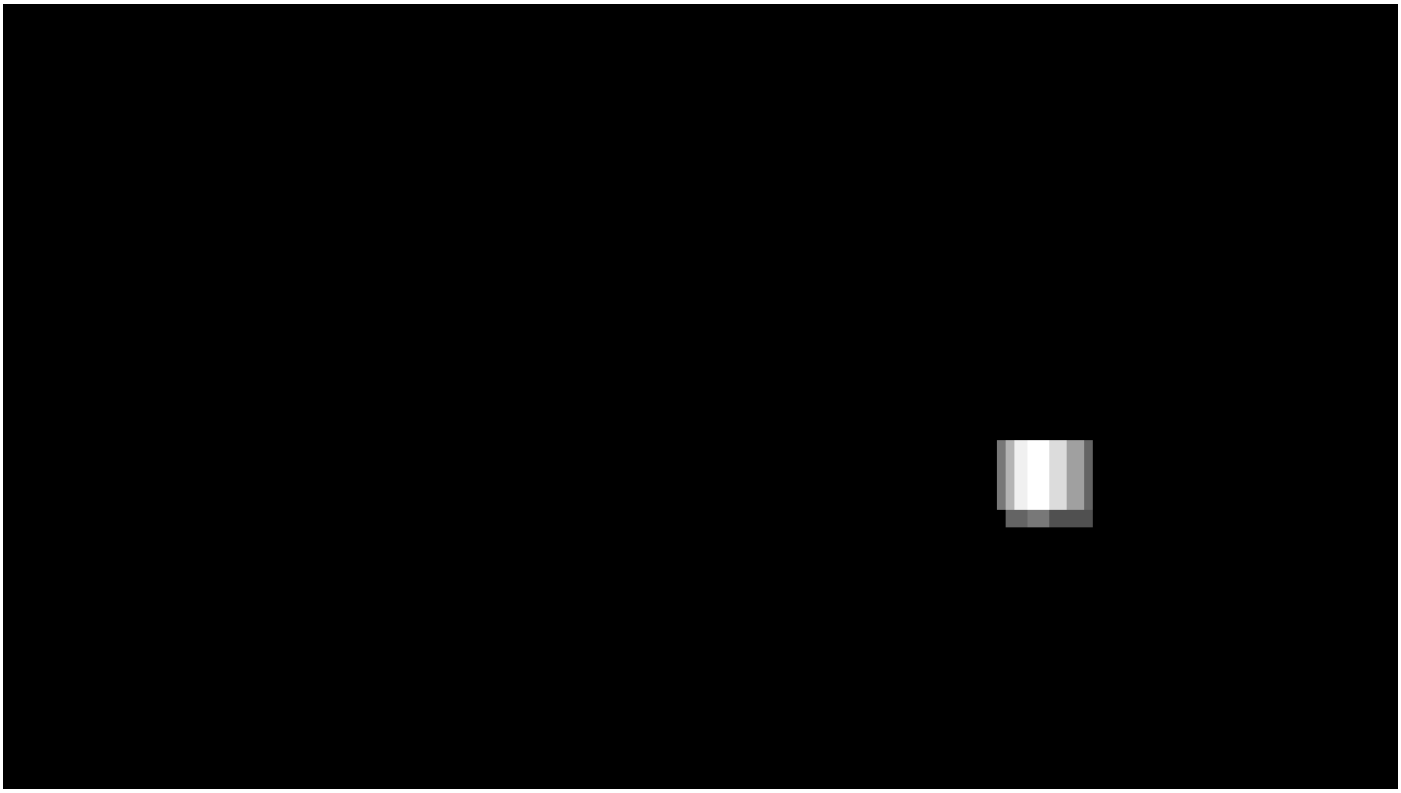
Code reference: `detector.py`

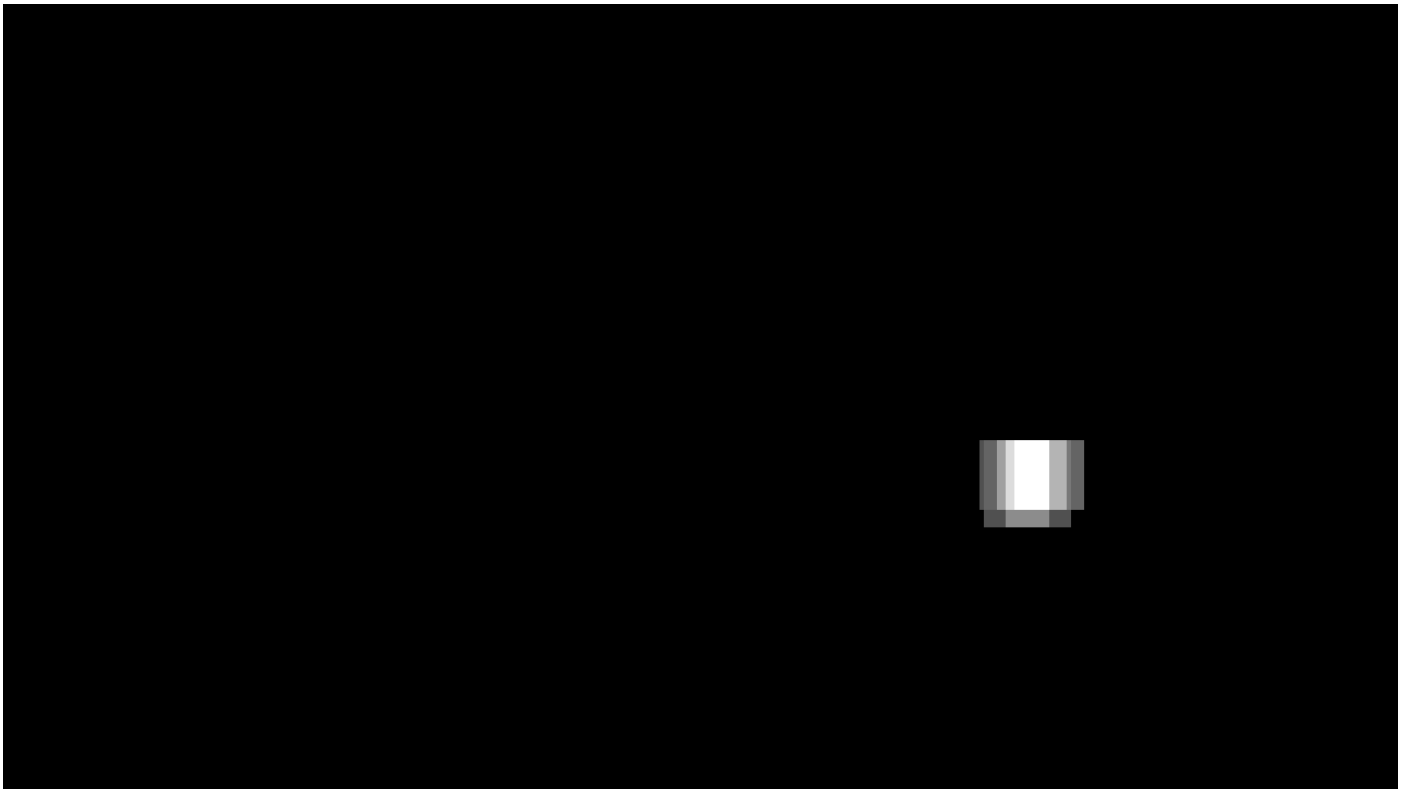
To fight with false positives, here's the things I did:

- Use heat map plus a sliding-window buffer to smooth detections. Some false positive bounding box only apprear in one frame then disappear, so I use a buffer to store recent 15 bounding boxes, and for all pixels inside a bounding box, I will add 1 to its heat map value. And then I set a threshold that only activates the pixel that has value bigger than 3. For transient bounding box, it won't continously contribute enough to the heatmap to pass the threshold thus it will be ignored. For stable bounding box, the heat map value will accumulate and eventually it will activate.
- Limit the search area. We can ignore the area that's on the left most and right most side. We can also ignore the area right in front of our car.

After we have our heatmap, I use `scipy.ndimage.measurements.label()` to turn activated heat map pixels into rectangles, and then draw in on the original image.

## Here are several frames and their corresponding heatmaps:





# Discussion

## 1. 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?

Here's the issue I experienced:

- More false positives than expected. My classifier has over 98% accuracy, but there's still many false positives. I had to use heatmap and some smoothing techniques to filter the transient bounding box. But the result is still not perfect.
- Detection in video is different than detecion in an image. In video, we have more context (prior frames) that can help us make reasonable detections. The sliding window buffer for bounding boxes wouldn't work for single still image.
- Region of interest (or region of uninterest) is very important. Not to mention that limiting the area for search can boost performance, it also help to avoid false

positives. For some reason, the highway's edge is sometimes classified into cars, and I have to limit the seach area to avoid that.

- One more thing that I can do is to smooth the box center, height and width. It doesn't make sense for it to change suddenly between frames and wobble, so maybe I can smooth it using prior frame's result so it won't wobble. I didn't have the time to implement it though.