

Vehicle detection

The goals / steps of this project are the following:

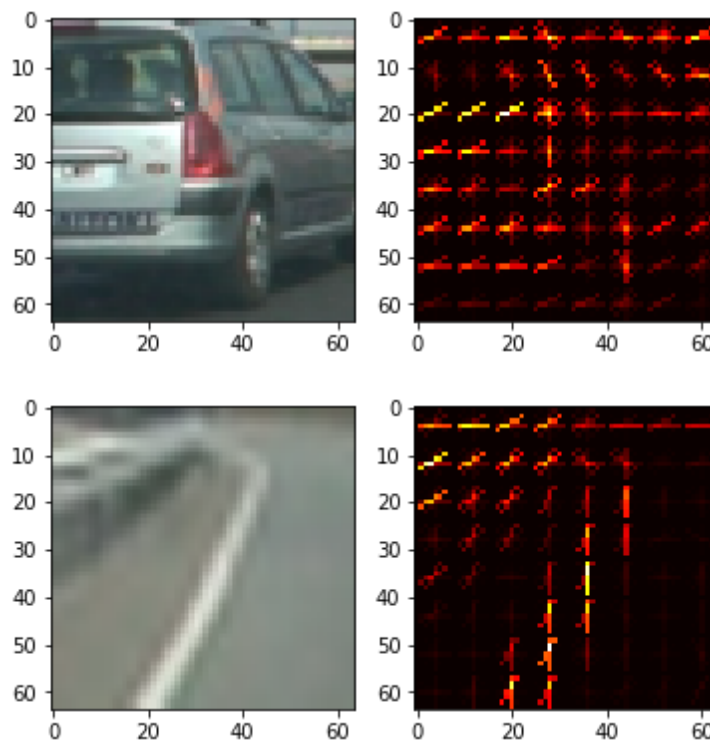
1. Perform feature extraction on a labeled training set of images, this includes HOG (Histogram of Oriented Gradients) feature and color feature
2. Train SVM classifier
3. Implement a sliding window technique and use the trained classifier to search for vehicles in images
4. Run the tracking pipeline on a video stream. Create a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles
5. Estimate a bounding box for vehicles detected

Here's a link to the video that shows the vehicle detection result: <https://youtu.be/piYzwH0Sa4c>

1. Feature Extraction

1.1 HOG feature

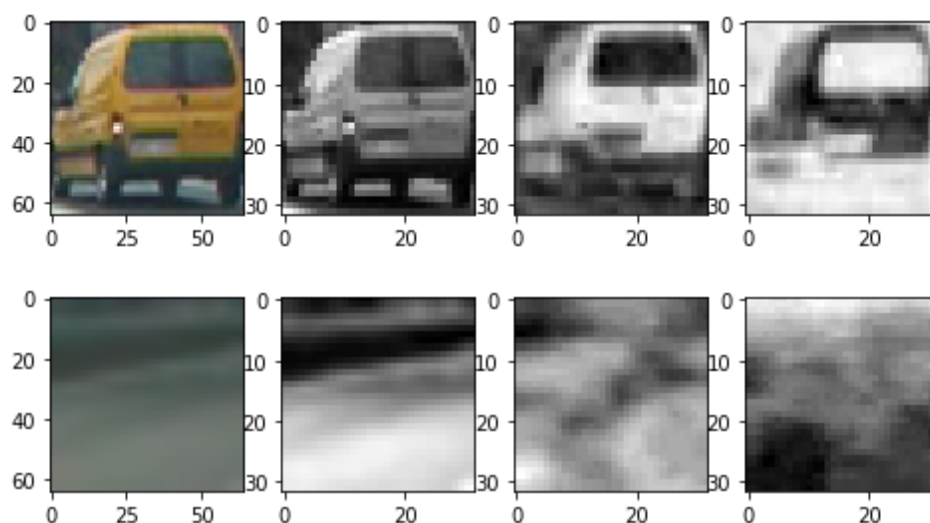
HOG ([Histogram of Oriented Gradient](#)) is widely used in object detection. In this project, I started reading vehicles and non-vehicles images provided by Udacity. Then I use `skimage.feature.hog` to calculate the HOG feature of an image. Here's an example:



1.2 Spatial bin feature

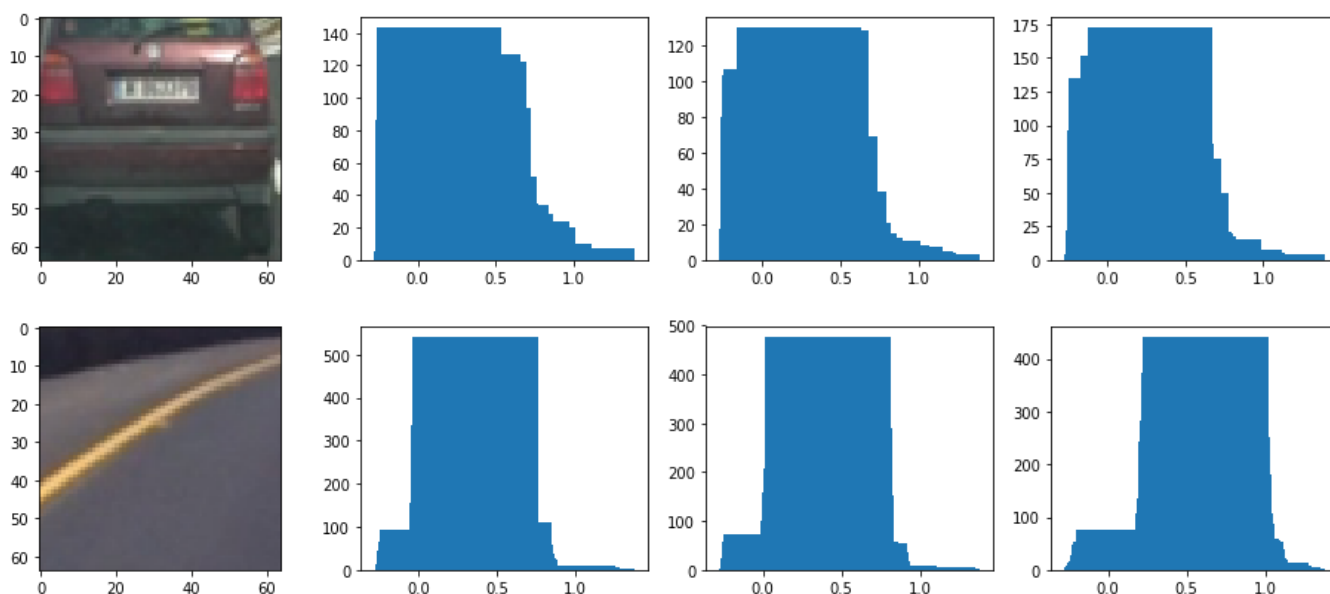
We also want to use image itself as a feature. However, we don't need to keep a high resolution image since a 32x32 image is good enough for us to tell whether it's a car or not. Thus, I use `cv2.resize()` method to reduce the size of an image.

Here are two examples of spatial bin feature of YCrCb color space:



1.3 Color Histogram

We also care about the color histogram of an object. I use `np.histogram()` to calculate the histogram. Here are two examples for vehicle and non-vehicle images:



1.4 Parameters

For HOG feature, I tried many combinations of orientation, pixel per cell and cells per block. I checked the feature image and follow this criterion: feature map should be concise enough and meanwhile keeps enough shape information. For spatial bin and color histogram feature, I just used the parameters mentioned in the lecture. For color space, I use 'YCrCb'. Here's the parameters I use:

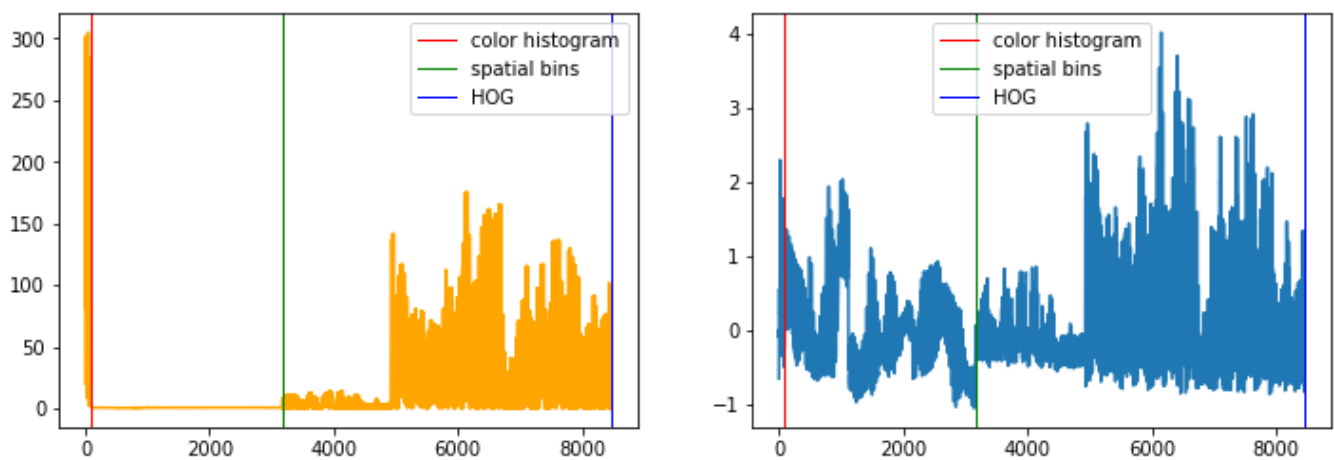
Feature	Parameters	Length
Color space	YCrCb	

HOG	orient = 9 pix_per_cell = 8 cell_per_block = 2	5292
Color histogram	nbins = 32	96
Spatial bin	Size = (32, 32)	3072

2. Train the model

2.1 Normalization

To train the model, I firstly extract spatial bin, color histogram and HOG features from the training data, concatenate these features and normalize it:



2.2 SVM classifier

In this project, I use SVM to classify an image. I also use `sklearn.model_selection.GridSearchCV` to find the best parameters for the SVM classifier. I tried a combination of kernel = {'linear', 'rbf'} and C = {1, 10} and it comes out that with 'rbf' kernel and C=10, the classifier outputs the best result.

However, it takes only 29 second to train a linear SVM classifier with accuracy of 98.8% while it requires 239 second to train a rbf SVM classifier with accuracy of 99.4%. It takes 10x longer to gain an extra 0.6% accuracy!

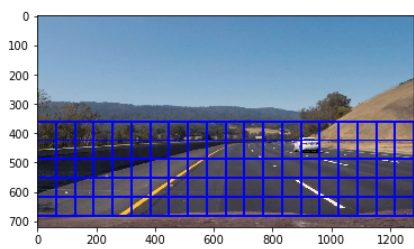
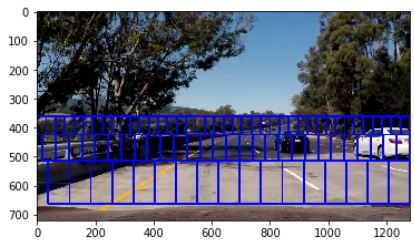
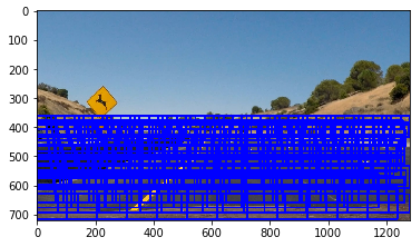
Btw, at first, I assumed that color feature is redundant since shape information itself is good enough. However, I found that I was totally wrong. Here's a table of SVM + feature combination:

Model	Feature	Accuracy
Linear SVM	HOG	92.59%
Linear SVM	HOG + spatial bin + color histogram	98.8 %
SVM: kernel = rbf, C = 10	HOG	96.00%

SVM: kernel = rbf, C = 10	HOG + spatial bin + color histogram	99.38%
---------------------------	-------------------------------------	--------

3. Sliding Window Search

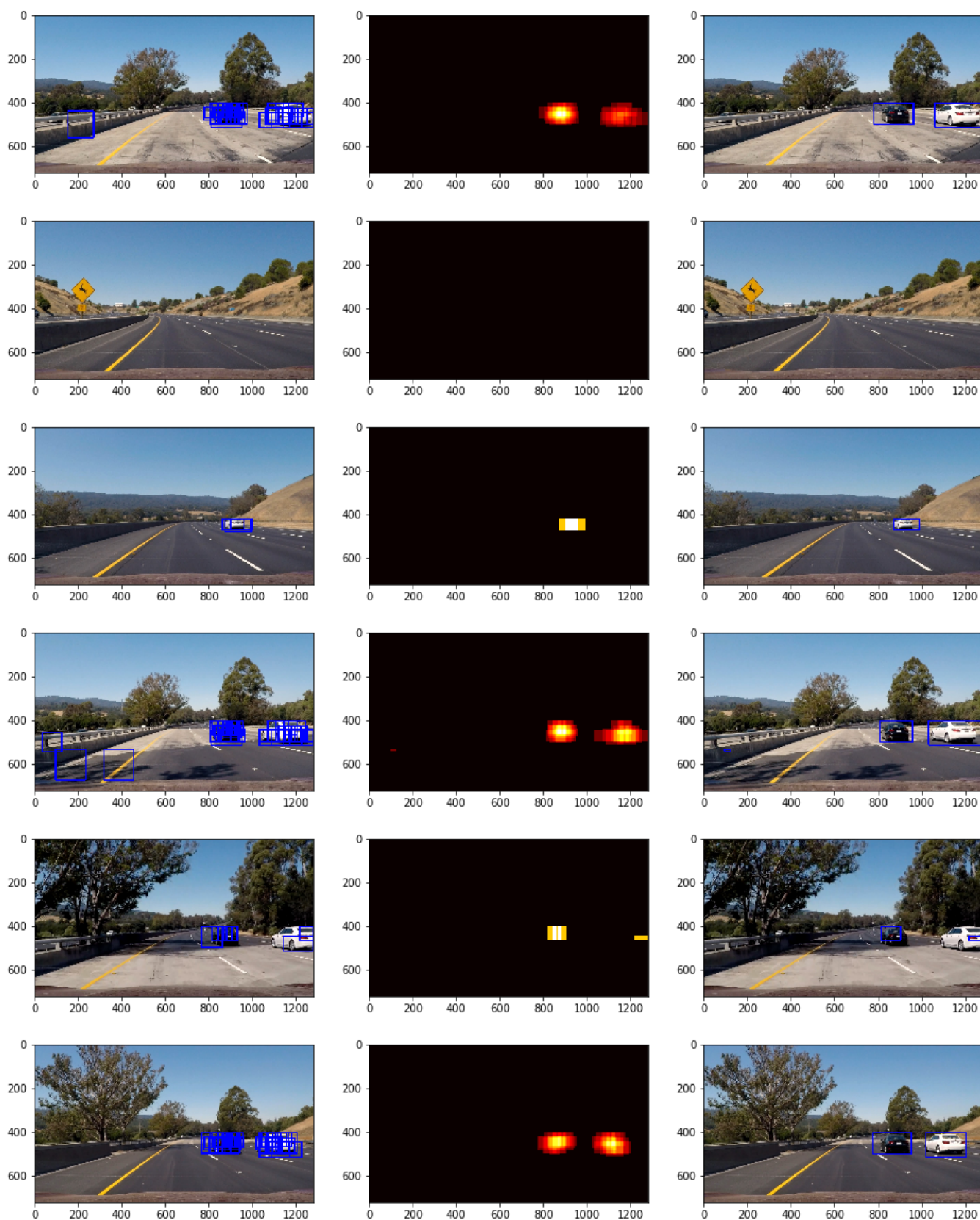
In order to detect objects within an image, a sliding window method is required. I implemented 3 methods to generate sliding windows.

method	description	example
<code>slid_win()</code>	It generates sliding window with same size across the screen.	
<code>perspective_slid_win()</code>	It generates sliding window with perspective view, which means that the closer the window is, the bigger the window appears to be.	
<code>multi_slid_win()</code>	It generates perspective sliding window with multiple scales. It's basically a wrapper around <code>perspective_slid_win()</code> , and it takes couple extra parameters to control the lower and upper window size and the increment of a window size	

But in the later stage of the project, I found out that the `perspective_slid_win()` and `multi_slid_win()` wasn't helpful. The reason is that the window sizes it generates are different and we can not use the wonderful idea of "extract HOG feature once and use it for all windows". What a pity!

4. Object Detection

In this step, I basically generate a list of sliding windows, then apply a SVM classifier to the resized window. If the prediction is positive, then add one to a heatmap. To remove false positive, I apply a threshold to the heatmap. Then I use `scipy.ndimage.measurements.label` to generate a bounding box around the car. Here are some examples:



5. Improvement

I applied the above processing method to a video stream but the result is not good as I expected. Even with such a high accuracy, there are still too many false positives since the number of sliding windows is

huge. If I use rbf SVM classifier, I can get a better output, but it takes 3.5 hours to process the whole video! What a pain!

So I decided to make two improvements on the processing method:

1. Extract HOG feature once
2. Average over n frames

Let's talk about them separately.

5.1 Extract HOG feature once

If we set `feature_vector` parameter to false when extracting [HOG](#) features, it returns a hog feature matrix. We can then use index to access the hog feature for a specific window. This can save couple seconds on an image.

I don't have much to talk about this since I mainly transcript the code from Udacity project Q&A section. I spent some time on understanding the logic though.

5.2 Average over N frames

I made a great effort in making this part work and this is original. Let's walk through the algorithm.

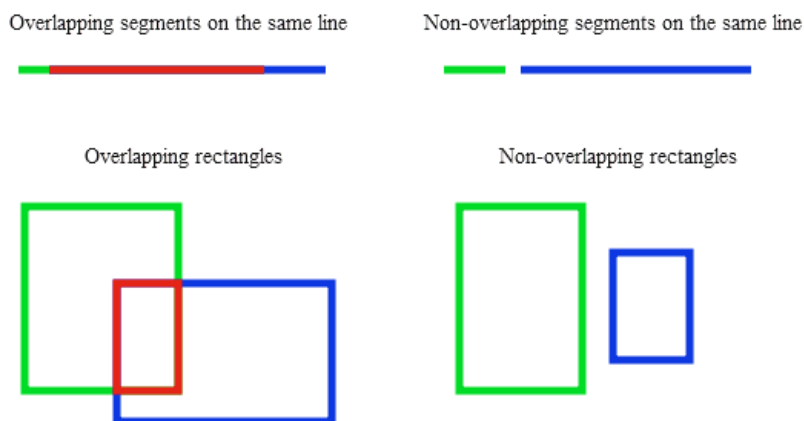
We've use heat map threshold to remove false positive within an image. Now we want to remove false positive across N frames. In order to do this, I keep track of a list of cars for previous N-1 frames. Now, we are moving to the N-th frame and we found a list of bounding boxes from it:

carlist: [c1, c2, c3, c4]

bboxes: [b1, b2, b3, b4, b5]

Now, we would like to know which car does this bounding box belongs to, or maybe it's just a false positive. I then calculate the overlap ratio for two boxes:

$$\text{ratio} = \text{overlap} / (\text{rect1} + \text{rect2} - \text{overlap})$$



[Overlap rectangles example](#)

If overlap ratio is less than 0.5, I would say this bounding box doesn't belong to any of previous cars. I use a 'idxes' array to store the calculation result. For instance, the following array means b1 belongs to c3, etc:

idxes: [3, 1, -1, 2, -1]

Finally, I update carlist according to the idxes array (if index ≥ 0 , we find a car in the carlist, we add the new bounding box to the car; if index == -1, we add a new car to the carlist) and calculate an average bounding box for the carlist. The algorithm works pretty well and here's an example:

1. Right top image: sliding windows to detect cars
2. Right middle image: threshold over heatmap to remove false positive
3. Right bottom image: threshold over frames
4. Left image: final output



5.3 Parameters

I tried all combinations of the following parameters:

Parameter	Value	Description
Window size	64, 96, both	Sliding window size
Heatmap threshold	3, 5, 0.1x, 0.5x	N = 3 or 5 N = 0.1 x len(windows) N = 0.2 x len(windows) Any position that contains more than N pixels is considered as valid car position
N, thresh	(15, 10), (18, 12), (20, 14)	N - average over N frame Threshold - if an object appears thresh times within N frame, it's not a vehicle; Otherwise, it is.
Overlap ratio	0.4, 0.5, 0.6	If the overlap ratio of two rectangles is less than R, this means they are not similar enough

If there are too many false positive within each frame, I will increase heatmap threshold.

If there are too many false positive across each frame, I will increase the 'thresh'; or increase 'overlap ratio'.

I finally choose this combination of parameters:

Window size = both, heatmap threshold = 5, N, thresh = (15, 10), overlap ratio = 0.4

6. Discussion

Here are some experience I learnt when I was working on this project:

1. openCV imread() method always returns image range from 0 to 255, while matplotlib imread() returns (0, 255) for jpg and (0, 1) for png format. This could make a huge difference on your classification result.
2. SVM with non-linear kernel yields a higher accuracy result. However, the prediction itself is still not satisfying enough and it requires 3.5 hours to process the whole video. I let my computer ran for two rounds and I finally gave up on using non-linear kernel.
3. SVM with linear kernel reduces processing time to 30 min which is manageable. I can tune the parameters more frequently and it generates a better output than non-linear kernel.
4. Extract HOG feature once and use it for all windows definitely a good idea. Try it.
5. Keep track of most recent N results and average over them. This could help to reduce False Positive across frames and the window looks more smooth!
6. Build the pipeline first. Don't waste time on inventing new algorithms if your pipeline is not done because you could spend your time in a part that doesn't need to be improved at all. In my case, I waste couple hours on different sliding window methods while it turns out they don't fit into the "HOG feature once" idea.
7. Spare some time on watching Stanford cs231n on [localization and detection](#). It totally worths the time and you will find the beautiful idea of R-CNN and fast R-CNN. I don't have time to implement them right now, but I will try that in the future.