Writeup

Vehicle Detection Project

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 (https://review.udacity.com/#!/rubrics/513/view) Points

Here I will consider the rubric points individually and describe how I addressed each point in my implementation.

Writeup / README

1. 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 (https://github.com/udacity/CarND-Vehicle-Detection/blob/master/writeup_template.md) is a template writeup for this project you can use as a guide and a starting point.

You're reading it!

Histogram of Oriented Gradients (HOG)

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

The code for this step is contained in the 6th code cell of the IPython notebook.

I started by reading in all the `vehicle` and `non-vehicle` images.

```
Vehicles Images Count : 8792
Non-Vehicles Images Count : 8968
```

I used the function in the lecture `get_hog_features` to get HOG features from an image. Here are some random examples of the output of this function:

I then used the function in the lecture `bin_spatial` and `color_hist` to help extract color features.

The function `extract_features` extract features from a list of images. I used it to extract vehicles features and non_vehicles features, then stored them in `car_features` and `notcar_features`. The parameters I used are:

```
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 8  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 'ALL' # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions|
hist_bins = 32
```

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

I started by using the original parameters from lectures. The result was OK. Then I only tried changing two parameters, `hog_channel = 0` to `ALL`, and `spatial_size = 32` to `16`. This obtained better test accuracy and used fewer time.

```
13.37 Seconds to train SVC...
Test Accuracy of SVC =  0.9927
```

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).

After I extracted features from images to `car_features` and `notcar_features`, I first concatenate these features and generated corresponding label vector `y`. Then I used StandardScaler to fit and scaled the features.

```
# Create an array stack of feature vectors
X = np.vstack((car_features, notcar_features)).astype(np.float64)

# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)

# Define the labels vector
y = np.hstack((np.ones(len(car_features)), np.zeros(len(notcar_features))))
```

Then I splited up data into randomized training and test sets using `train_test_split function`:

```
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(scaled_X, y, test_size=
0.2, random_state=rand_state)
```

I first tried to use the `GridSearchCV` function:

```
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
svr = SVC()
svc = GridSearchCV(svr, parameters)
svc.fit(X_train, y_train)
```

But after waiting for over 10 minutes this didn't return any results.

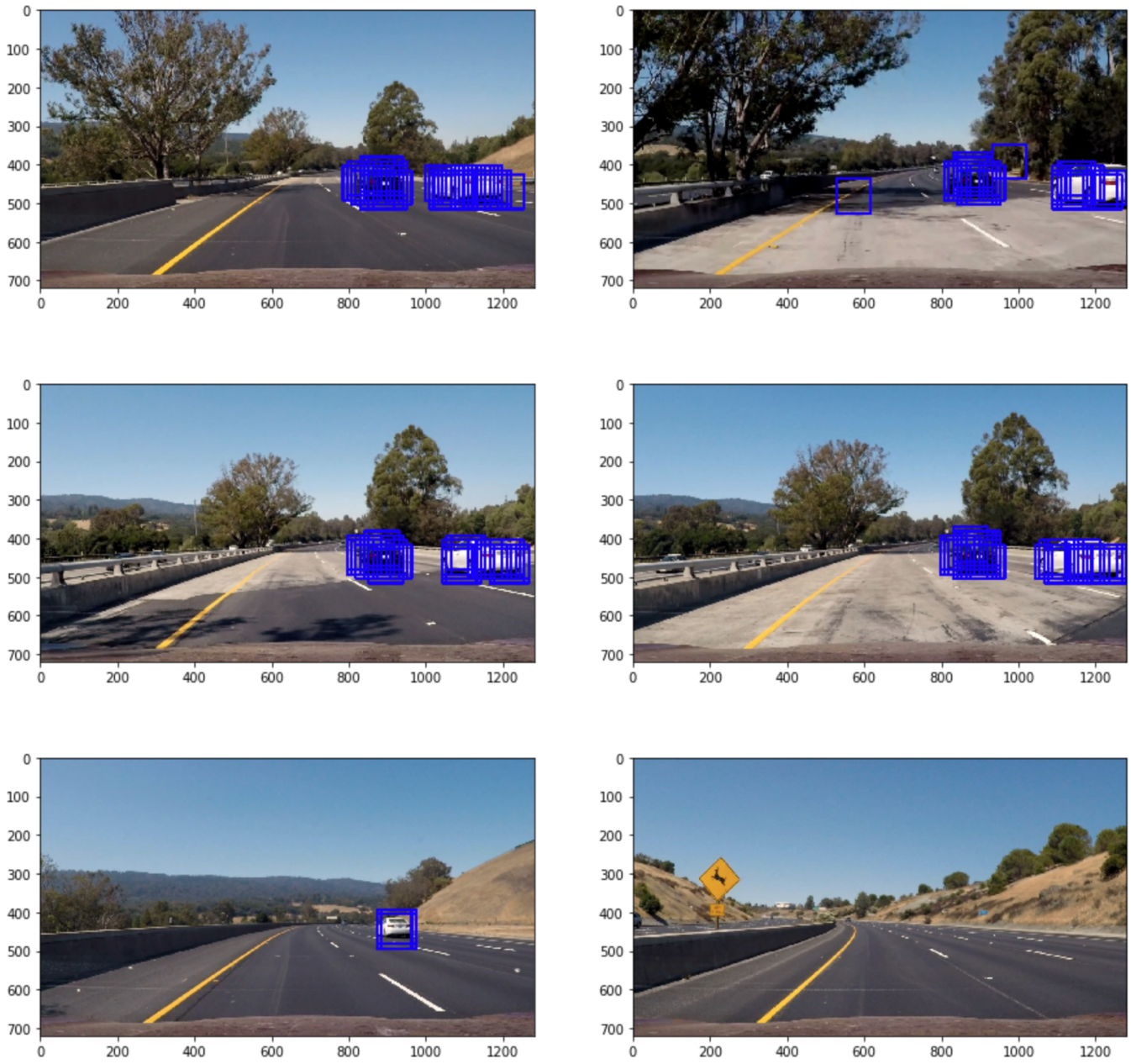
So I decided to use `LinearSVC()` first, and it obtained good enough results. So I didn't go back to tune `parameters`.

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?

I first tried to implement sliding windows and then apply feature extraction to each window and use the trained classifer to detect cars. But this is too slow even test on test images. So I decided to use the idea from lectures, where we computed HOG for the whole image, then extract corresponding features for each window. This was in the 19th cell in the IPython notebook.

This elementation let me experiment parameters on test images. I had to set `ystart = 350`, `ystop = 656`, `scale = 1.4`, `cells_per_step = 1` in order to detect all cars in test images while not getting too many false positives and keeping the searching area as small as possible (especially for saving time when processing videos). Here is the result on test images:

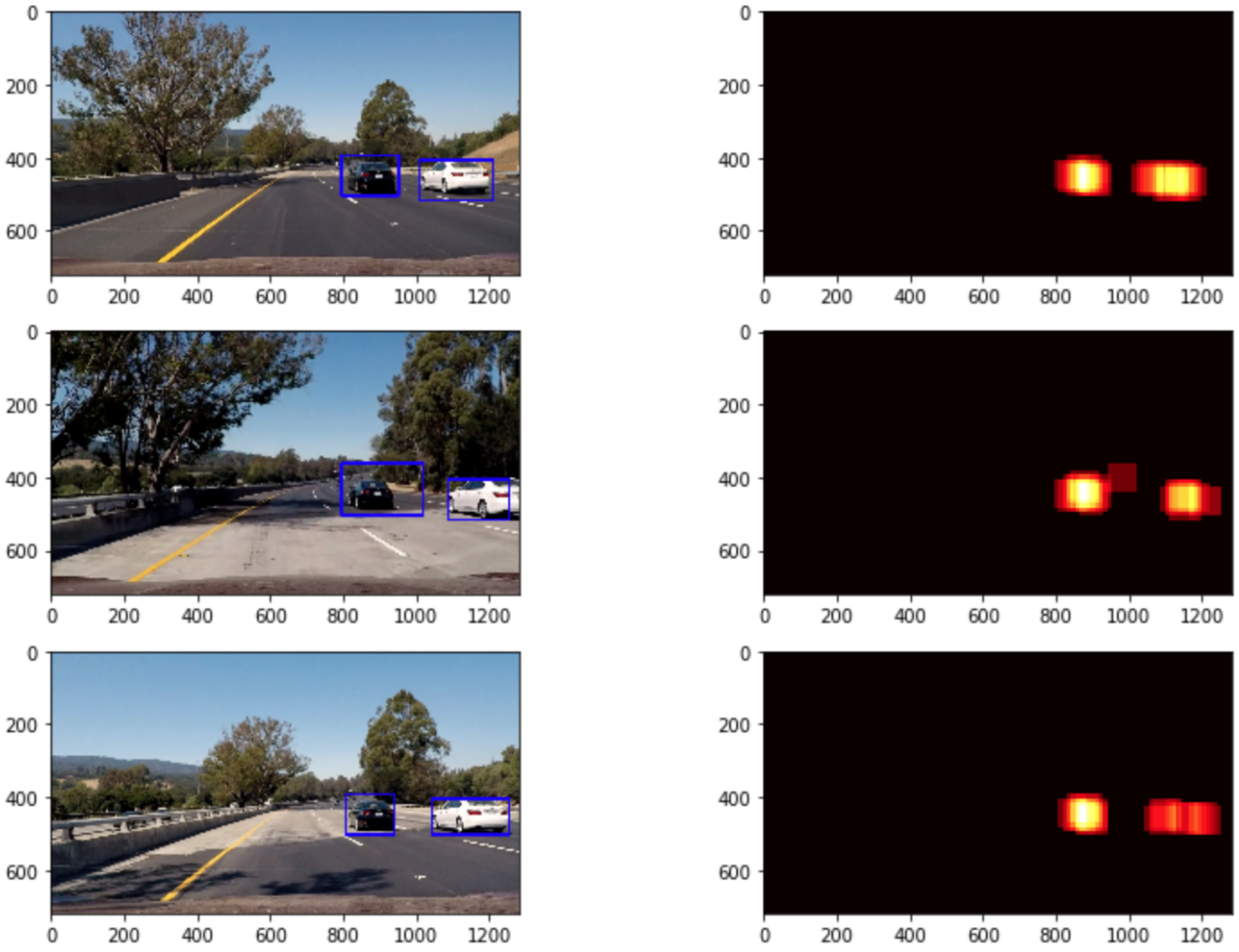


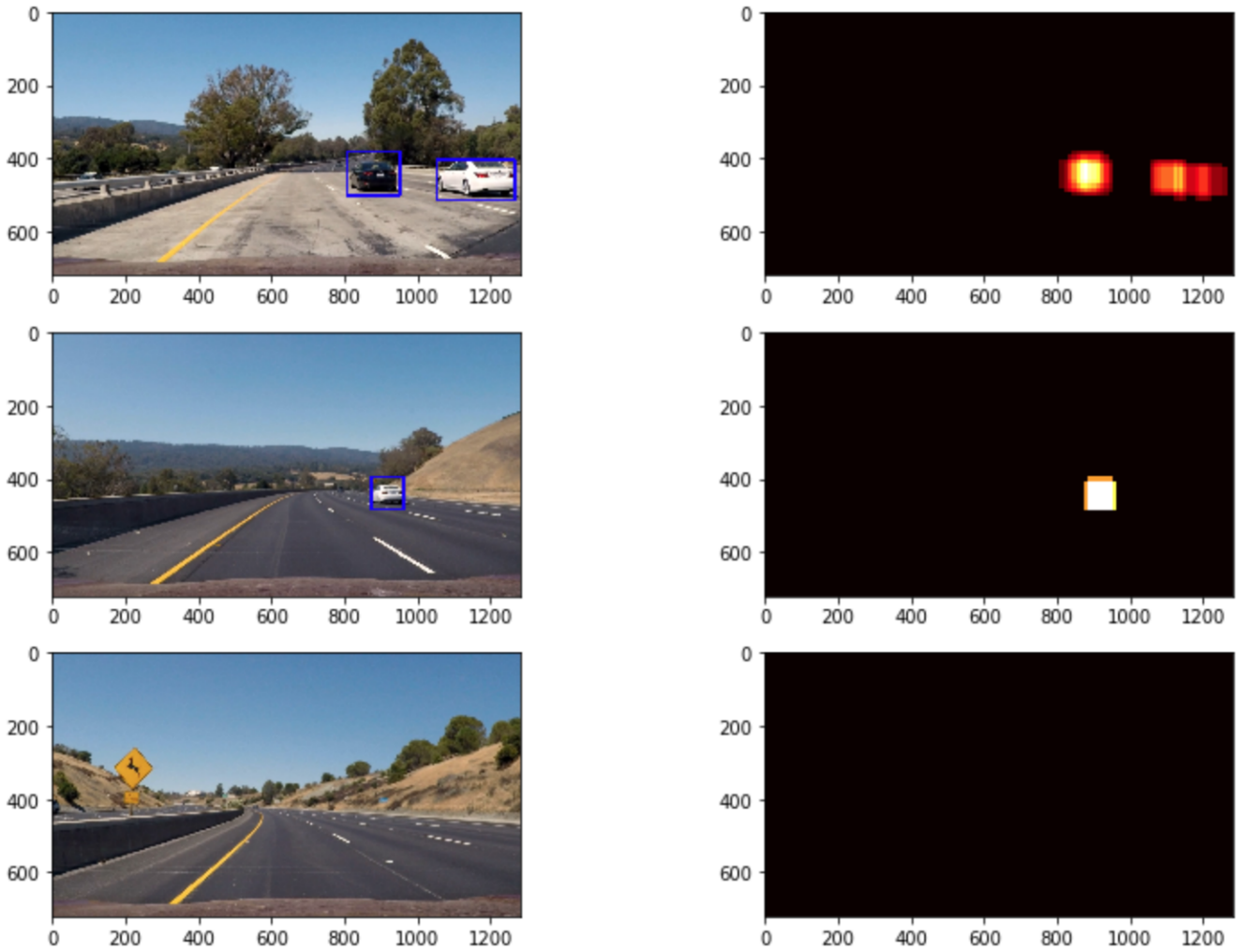
2. Show some examples of test images to demonstrate how your pipeline is working. What did you do to optimize the performance of your classifier?

I then implemented the threshold method and `label` function taught in the lecture to combine multiple detections and eliminate false positives. The codes are in the 21st and 22nd cells in the notebook. I set the `threshold = 2` to eliminate all false positives in test images while keeping all the cars.

Ultimately I searched on two scales using YCrCb 3-channel HOG features plus spatially binned color and histograms of color in the feature vector, which provided a nice result.

Here is the result on test images:





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.)

My video file `video_output_project.mp4` is in the zip file.

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.

I recorded the heatmap in each frame of the video, but only kept the current heatmap and previous two heatmaps to compute the average heatmap. I then thresholded that map to identify vehicle positions (`threshold=2`). I then used `scipy.ndimage.measurements.label()` to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. I constructed bounding boxes to cover the area of each blob detected.

I first test on `test_video.mp4`. The result looked good. Then I processed `project_video.mp4` using the same pipeline and saved the output file.

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?

One issue is about parameters tuning. For some parameters I get too many false positives, and for others I sometimes can't detect that white car if it went too far (because it becomes smaller and there won't be enough heat scores.)

I think this pipeline may fail given other test_videos. Because the test_images I used are all from the same test video. The parameters I ended up choosing are all suitable but only for this test video. I really worry that if given other test videos, there may be too many false positives or sometimes car may not been detected.