Project 5 - Vehicle Detection and Tracking

6th March 2017

Files in the Project

My project includes the following files:

- Vechicle_Tracking.ipynb contains all the code for running the project.
- output_images folder contains the images that are within the writeup report
- project_video_solution.mp4 showing the vehicles being detected and tracked.
- writeup_report.pdf

Getting the Data

We needed to create a pipeline that took in images of cars and non-cars. Then we needed to extract features from the images and feed them into a classifier which we then trained. Then we applied a sliding window technique to scan the images and let our classifier classify the cars in the images. We then filtered out false positives and finally ran the pipeline on the video. An example of a car image and a non car image can be seen in Figure 1. Reading in the images can be seen in *Vehicle_Tracking.ipynb cell 2*.

Extracting Features

Color Space

There are many different types of colorspaces that can be used for this project. During the lesson there were numerous ones explored other then the

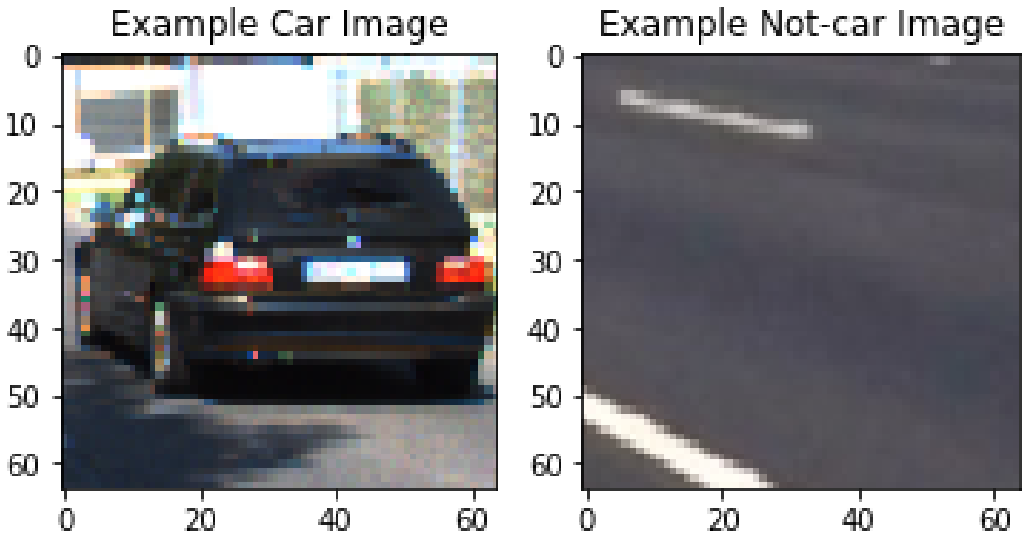

Figure 1: Showing a sample image of a car (left) vs a sample image of no car(on the right)

regular RGB, like, HLS, LUV, YUV, etc.. Depending on what you're looking to accomplish, one colorspace may be more appropriate then another one. I think RGB is not a good choice because, if you are trying to detect vehicles around you, and it becomes very dark, or very bright or rainy, I feel as though the RGB colorspace wouldn't do a good job in allowing our classifier to be able to distinguish the car from the background.

After experimentation, I went with the LUV colorspace, an example of how it looks can be see in Figure 2. This can be seen in *Vehicle_Tracking.ipynb cell 4*.

I found that this gave me the best results. Although with different implementations I would try different colorspaces to make sure I am getting the best results

Histogram of Colors

Now we want to look at the histograms of pixel intensity as features. We end up taking the image and separating each color channel that the image has, L, U and V. I attempted using different combinations of color channels, as well as just using a single color channel, and I found that using all 3

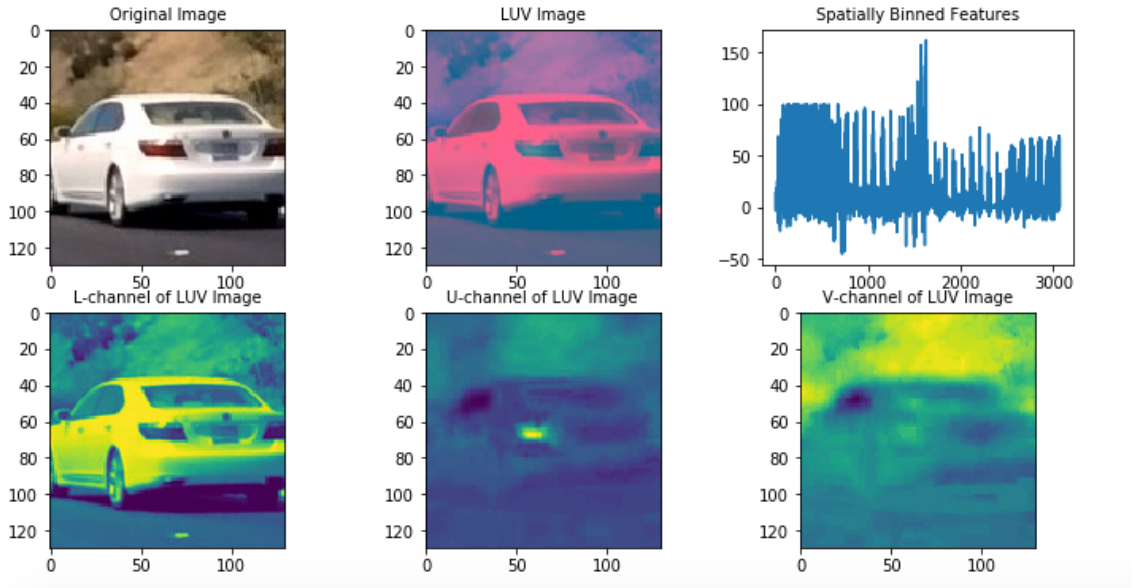

Figure 2: An example of what the LUV colorspace looks like.

color channels for the feature vector, ended up giving optimal results for my case. The color histograms can be seen in Figure 3 separated into each color. We can also perform spatial binning on a full resolution image and still retain enough information to help in finding vehicles. This can be seen in *Vehicle_Tracking.ipynb cell 5.*

Histogram of Oriented Gradients - HOG

HOG is a histogram that is based on the orientations of the gradient at each pixel. The image is divided into a number of cells, which the gradients are calculated and then binned depending on the orientation of the gradient. Essentially the HOG illustrates the dominant gradient orientation of each cell. I set the number of pixels per cell to 8, the number of cells per block to 2 and the number of orientations to 9. All of the hog channels were used in the calculating of HOG. This part of the pipeline involved a lot of trial and error. I tried using just one hog channel at first, but I noticed that it gave poor results compared to using all of the hog channels. I also noticed that increasing the orientations past 9 didn't increase the accuracy of the classifier that much, while also adding to the length of the feature vector (All of the parameters can be found in *Vehicle_Tracking.ipynb cell 8 and the*

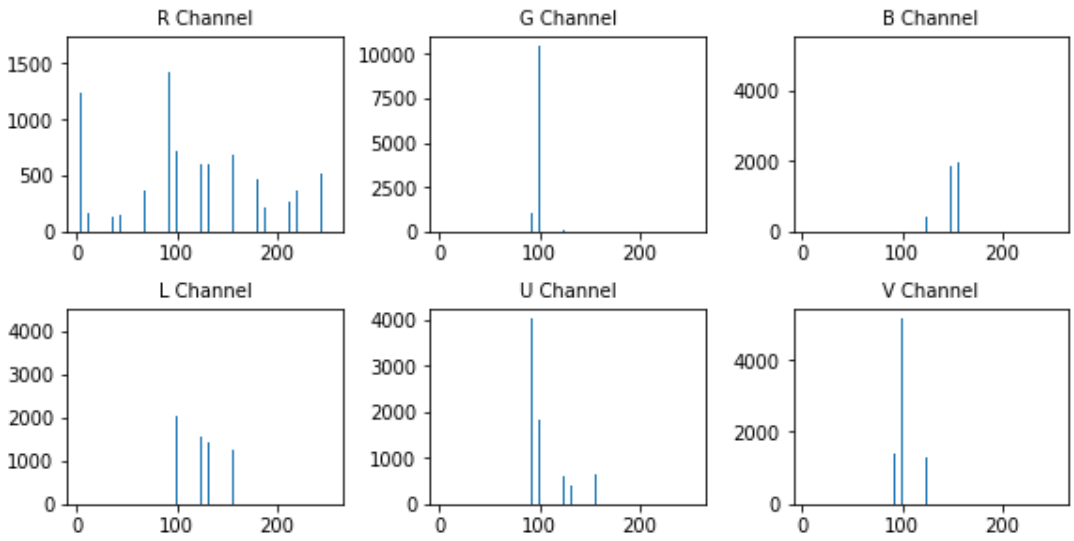

Figure 3: The pixel intensities as a histogram for each color channel of a sample image in RGB (top row) and LUV (bottom row)

HOG function can be found in cell 6). The HOG visualization can be seen in Figure 4

The Classifier

A linear support vector classifier was used (can be found in *Vehicle_Tracking.ipynb cell 8*). I first extracted all of the features from the car and non-car images. Then I stacked the features from both sets together. I then normalized my data and then split it up into training and testing data. I set the testing size of the data to 10% of all of the data. I then fit the classifier to the data, which achieved an accuracy of 99.38%.

Identifying cars on the images and videos

Sliding Window Technique

Once our classifier is trained we want to start identifying images. We implement a technique called Sliding Window (*Vehicle_Tracking.ipynb cell 9*). This technique will create a little window that slides across the image,

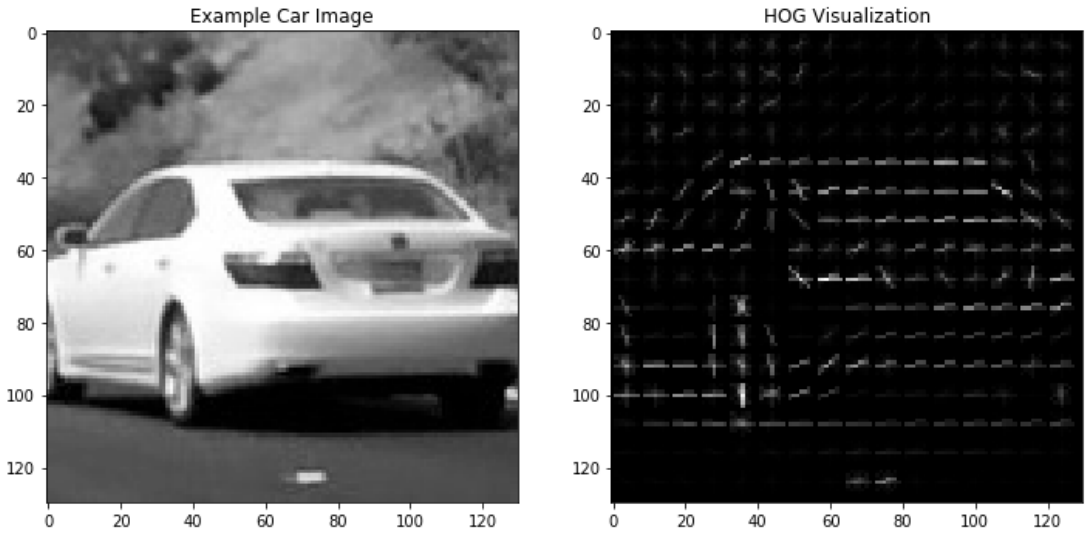

Figure 4: Greyscale Image to the left vs HOG visualization to the right

in set intervals. While the windows are sliding across the image, for each individual window, the classifier will run (using the search_windows function in *Vehicle_Tracking.ipynb cell 10*), and attempt to classify whether or not there is a car in that particular window.

As can be seen in Figure 5, we see that the classifier does a good job identifying the cars when they're in the image and also not identifying any cars when they're not in the image (top right).

At first I set the sliding window function to search the whole image, but not only did that add in compute time, but that also added in the misclassifications. So to combat this, I only allowed the window to slide between the values of 400px and 656px in the y-direction, because then this limited the amount of space the function had to search, but also increase the ability of the classifier to predict the correct outcomes.

HOG Sub-sampling Window Search

To decrease in the amount of run time for the algorithm, we only want to extract the HOG features once (found in *Vehicle_Tracking.ipynb cell 14*). The find_cars only has to extract hog features once and then can be sub-sampled to get all of its overlaying windows. Each window is defined by a scaling

Figure 5: Example of the classifier identifying if there is a car in the sliding window going across the image.

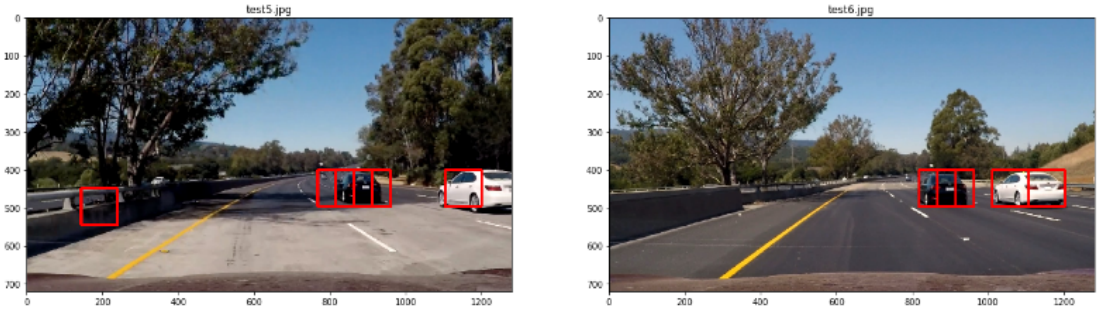

Figure 6: Example of the classifier misclassifying a car (left) vs correctly classifying a car (right).

factor where a scale of 1 would result in a window that's 8 x 8 cells then the overlap of each window is in terms of the cell distance. This means that a cells_per_step of 2 would result in a search window overlap of 75%. Its possible to run this same function multiple times for different scale values to generate multiple-scaled search windows.

Heat Map

Can be found in *Vehicle_Tracking.ipynb cell 14*

I recorded the positions of positive detections in each frame of the video. From the positive detections I created a heatmap and then thresholded that map to identify vehicle positions. 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.

Here's an example result showing the heatmap from a series of frames of video, the result of scipy.ndimage.measurements.label() and the bounding boxes then overlaid on the last frame of video:

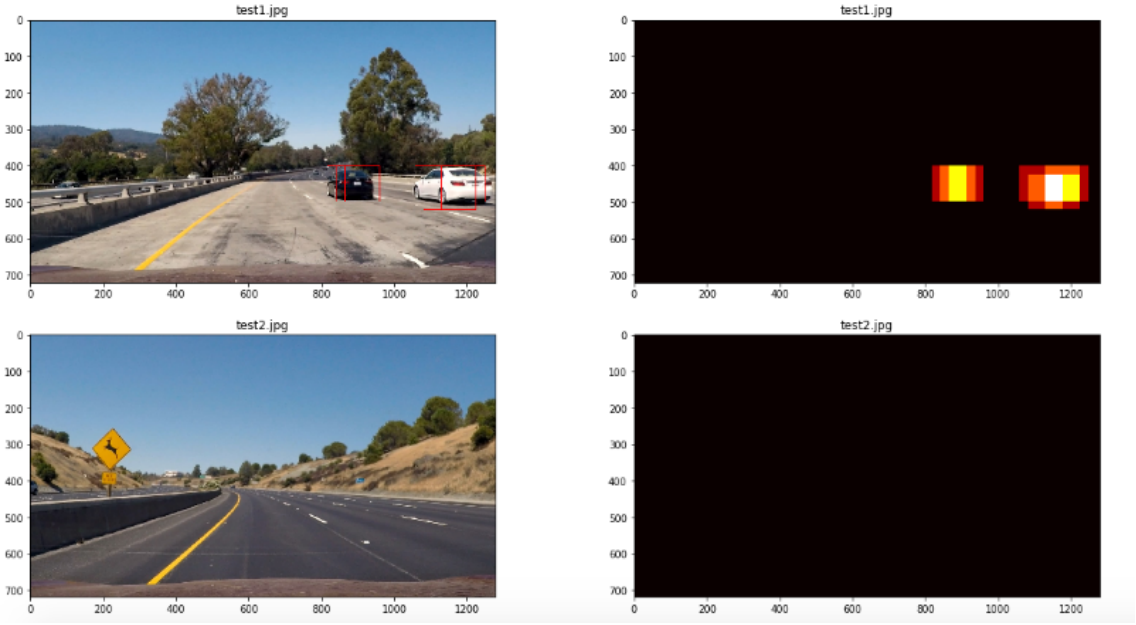

Figure 7: Example of images with the corresponding heatmap

Figure 8: Example of images with the corresponding heatmap.

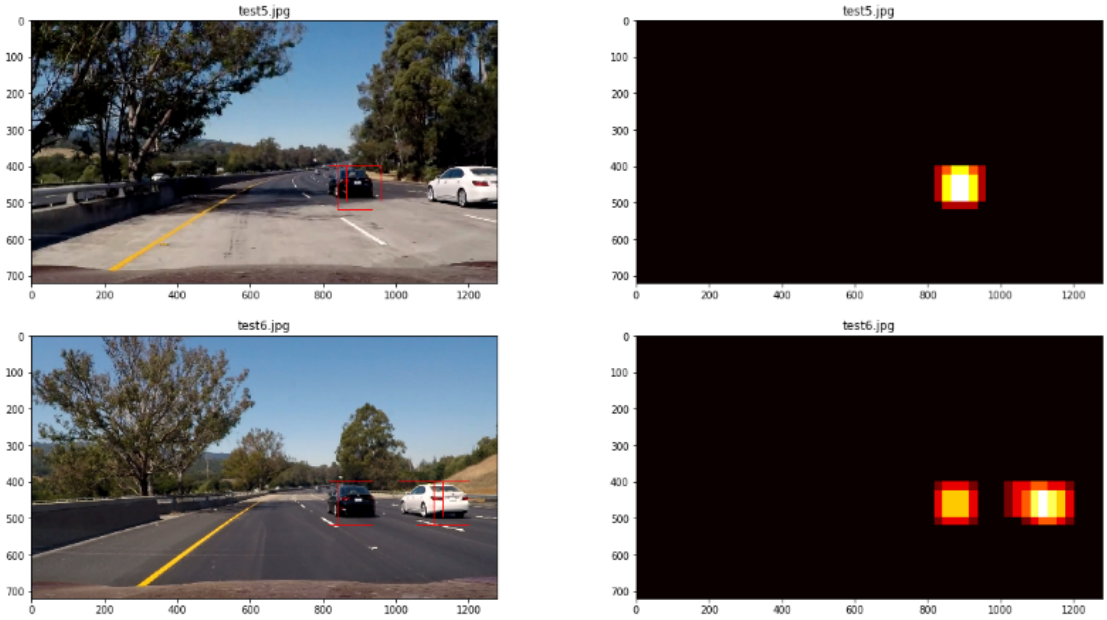

Figure 9: Example of images with the corresponding heatmap.

Discussion

I think the pipeline works fairly well. There are obviously some improvements to make. During the video I noticed that over the white pavement the algorithm didn't track the car, which might be able to be fixed with the use of a different colorspace?

Also I found it interesting when both cars were near each other in the video, it was classified as one vehicle. Im not sure how this could be fixed, but maybe more precise scanning of the image along with more training data could fix that.

Another aspect that would have to change from my pipeline is the false positives or the lack of detecting a vehicle. If you had false positives in a real life scenario, the car may be on the highway and then slam on the breaks because it thinks a car is right next to it coming head on. Also in my pipeline where the car wasn't being tacked for a couple of seconds is also bad, because if that car were to change into my lane, and slow down, then my car would hit it, which also would not be good.

Also I would think there is some way to optimize the code so that maybe it could run in real time.

With more time, I would add the lane detection project to this video so that the pipeline would be tracking both the lane and the car. I would also like to try and take a deep learning approach to this problem instead of somewhat hardcoding the parameters.

Overall I did enjoy this project and found it challenging yet rewarding.