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
- I also used a color transform and append binned color features, as well as histograms of color, to my HOG feature vector.
- For those first two steps I normalized my features and randomize a selection for training and testing.
- I implemented a sliding-window technique and used my trained classifier to search for vehicles in images.
- I run my pipeline on a video stream and created a heat map of recurring detections frame by frame to reject outliers and follow detected vehicles.
- I estimated a bounding box for vehicles detected.

My submission consists of the following items:

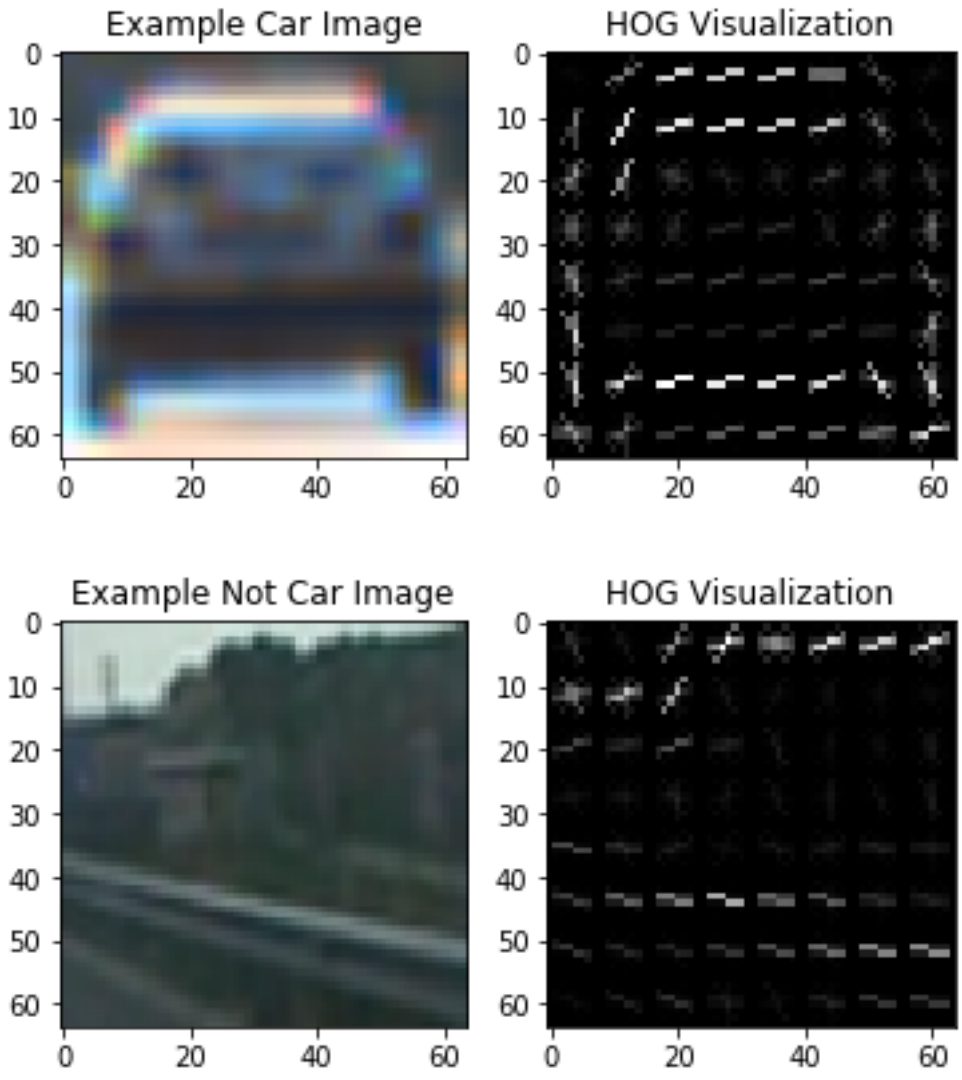
- Write up report – this document
- Python script project5.py where single image processing pipeline is implemented together with output images of each pipeline step
- functions5.py containing some functions needed in project5.py and project5_video.py
- get_hog.py implementing hog features for tests and experiments (the hog features are also used in project5.py and defined in functions5.py)
- Python script project5_video.py where I implemented the pipeline for video processing
- Python script project5_video_dl.py where I implemented the pipeline for video processing using a deep learning classifier that works direct on the images without need for separate feature extraction
- trainClassDL.py where I implemented deep learning classifier
- output_images directory containing the images of the pipeline processing steps from the script project5.py
- project_video_output.mp4 is the output of the video processing pipeline implemented in project5_video.py using HOG and color features and linear SVM classifier
- project_video_output_dl.mp4 is the output of the video processing pipeline using a deep learning pipeline implemented in project5_video_dl.py

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

Histogram of Oriented Gradients (HOG)

The code for this step is contained in the in lines 9 through 19 of the file called get_hog.py (or 13 to 30 in the file functions5.py.

I started by reading in all the vehicle and non-vehicle images. I then explored different color spaces and different skimage.hog() parameters (orientations, pixels_per_cell, and cells_per_block). I grabbed random images from each of the two classes and displayed them to get a feel for what the skimage.hog() output looks like. Here is an example using the RGB color space and HOG parameters of orientations=9, pixels_per_cell=(8, 8) and cells_per_block=(2, 2):



Final choice of Features and Parameters

I tried various combinations of HOG parameters like channels, orientations, pixels_per_cell, and cells_per_block and selected ones with the best classifier performance.

Also I tried different features like HOG features, color histogram and bin spatial features as well as their combinations. Finally I used all three classes of features: HOG, histogram and spatial features since that was combination that provided the best classification performance on the test set (about 98% correct classifications).

Classifier Selection

I trained a linear SVM using provided car and not car images in the lines 166 to 229 of the project5.py file (or in the same vain in lines 255 through 321 in project5_video.py file).

The images are at first normalized using "StandardScaler". Than the provided images are split into train set (80%) and test set (20%). After the classifier training the performance is estimated using test set. The performance on the test is used as criteria which features and which parameters to include for image classification.

I also tried non-linear SVM classifier using polynomial of grade 3. Although this classifier provided better performance on the test set (over 99% correct classifications) in comparison to the linear classifier (about 98% correct classifications) no noticeable performance gain was observed in the video. Furthermore non-linear classifier needed much more time for training and execution than the linear one.

Finally, I also tried a deep learning classifier (similar to the classifier I used for the project Behavioral cloning). The deep learning classifier is implemented in trainClassDL.py and the video processing using the classifier is implemented in project5_video_dl.py.

Image Processing Pipeline

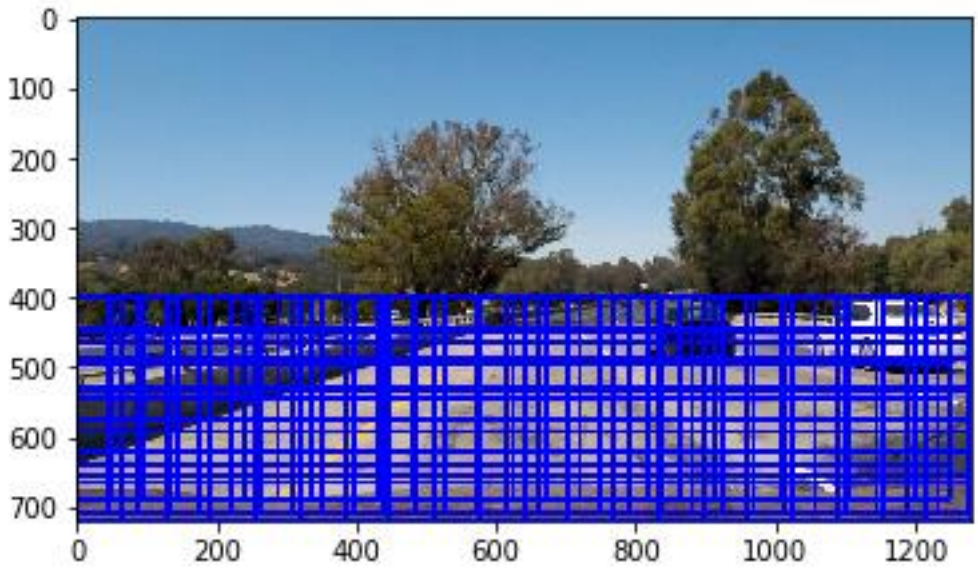
My image processing pipeline consists of the following steps:

- Sliding window search
- Using classify for each window to find positive detections in each window
- From the positive detections I created a heatmap and then thresholded that map to identify possible vehicles
- 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 is the example of the pipeline for test image 1 (other images are provided in the directory output_images:

Sliding window search:

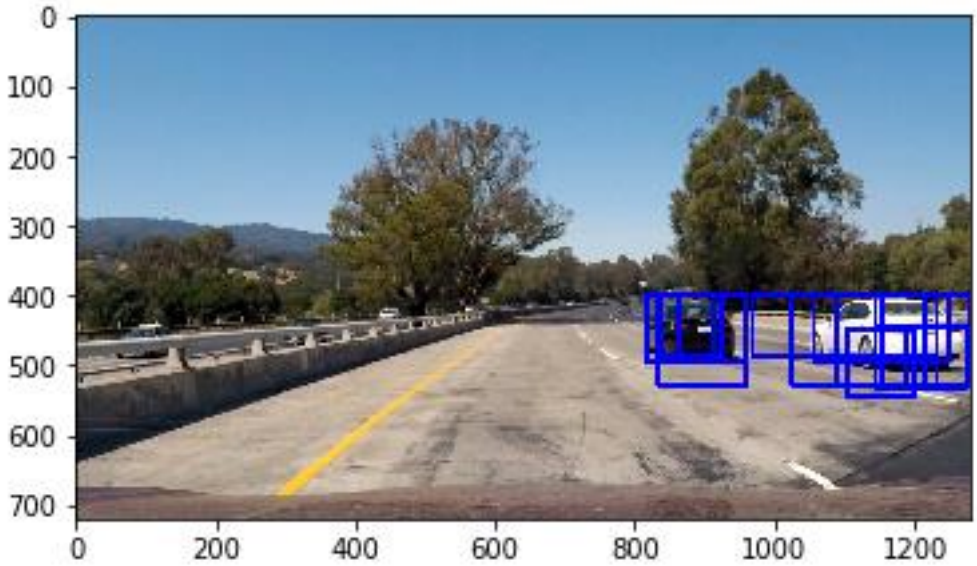
Sliding window search is performed on three scales using boxes of sizes 88x88, 96x96 and 128x128 pixels with 50% overlapping. The search is done only in the lower half of the image, where vehicles are supposed to be. Here is the example of the search windows mask:



Feature extraction and classification

I used HOG features plus spatially binned color and histograms of color. I put the features in the feature vector, and used SVM classifier to select possible windows with cars (see the function search_windows in project5.py, lines 79 through 107).

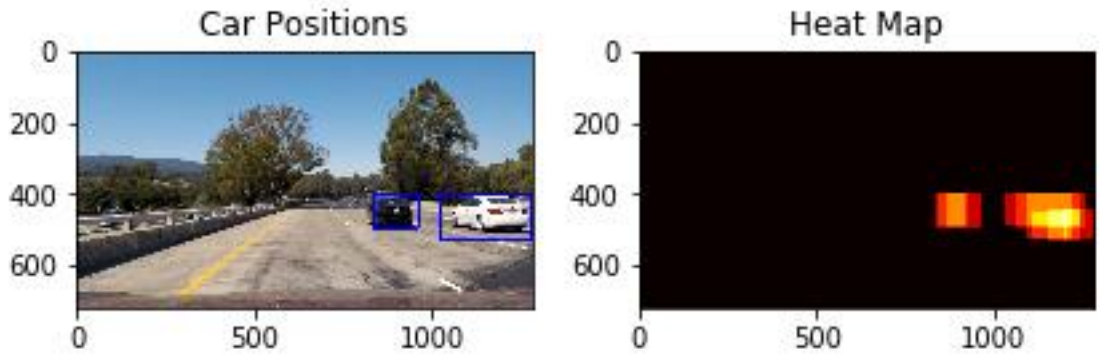
Here is the image with “positive” windows:



Heatmap

From the positive detections I created a heatmap and then thresholded that map (with threshold = 1) to identify possible vehicles (see functions add_heat and apply_threshold, lines 108 through 122 in project5.py).

Here is the result of heatmap generation:

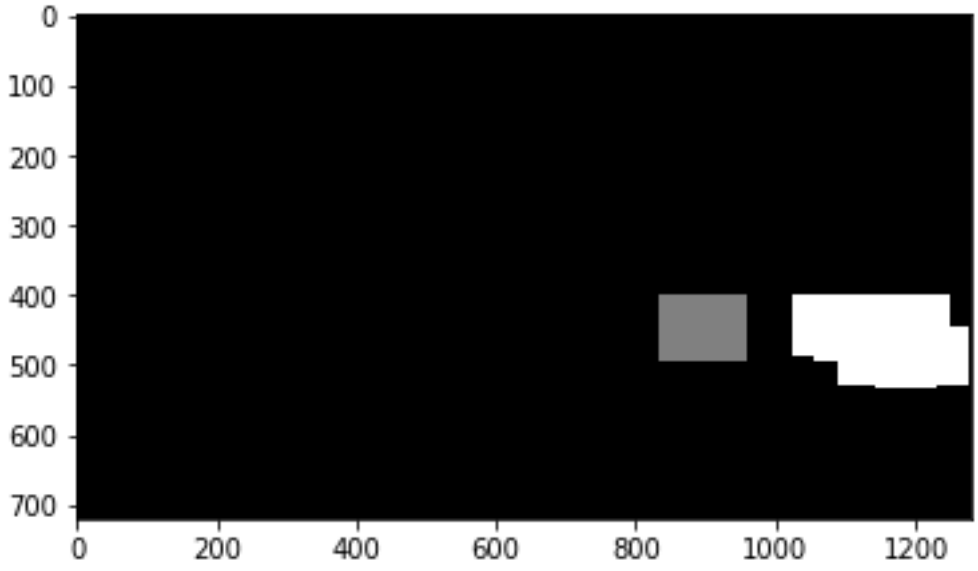


Labels

To find out how many cars I have in each frame and which pixels belong to which cars, I used the label() function from scipy.ndimage.measurements.

Here is the result of the function call for the heat map above:

2 cars found



Detected Cars

Finally, I draw the image with detected cars in the example below:

Video Implementation

The video: project_video_output.mp4 provides results using SVM classifier and HOG and color features described above.

project_video_output_dl.mp4 provides the result with the deep learning classifier using only "raw" images as features.

Filter for false positives

As described above 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.

Furthermore, I implemented the class Vehicles() in the project5_video (lines 209 through 213) with attributes: probs (probability of the vehicle), boxes (averaged boxes for the detected cars) and ndet (number of previous detection in the close position). This class tracks vehicles in the video, saves their average positions, probabilities and number of previous detections in order to reduce number of false detections. In the function draw_labeled_bboxes (lines 152 to 207 in project5_video.py) only boxes with the probability over certain threshold and boxes already detected in previous frames are drawn. To that purpose, I used the function check_box (lines 124 through 150), where for each possible box is checked how far it is for previously detected boxes. Only boxes close to previous ones (I used 15 pixel threshold) are drawn, provided their probability is also over the threshold. The probability is estimated (using moving average) and checked in draw_labeled_bboxes function. Also the check for previous detections is also performed in the function. On this way the number of false positives was reduced.

Deep Learning Classifier

Finally, I also tried a deep learning classifier (similar to the classifier I used for the project Behavioral cloning). The deep learning classifier is implemented in trainClassDL.py and the video processing using the classifier is implemented in project5_video_dl.py.

The performance of two classifiers SVM and deep learning are similar, as can be seen by comparison of the videos: project_video_output.mp4 (with SVM classifier) and project_video_output_dl.mp4 (with the deep learning classifier). But the deep learning classifier has big advantage that it works on the "raw" image and no extensive feature extraction and selection should be done as in the case of "classical" machine learning classifiers like SVM.

Discussion

Main difficulty was to reduce false positives. That's why I introduced class Vehicles() and estimated probabilities of the vehicles at given positions. It helped, but there is inherent trade-off between reducing the number of false positives and the number of true positives. Furthermore, better tracking algorithms would help better estimation of vehicle positions and thus reduce the number of false positives.

I also used a deep learning classifier as described above. The advantage of this classifier is that we don't need to engineer the features. It could be also more accurate than SVM classifier and needs less time since we don't need to extract the features. Further improvements in the classifier architecture are possible by adding or omitting some layers. Both classifiers could profit from increasing the number of training examples.

Also detections of faraway vehicles, especially by bad weather, lightening and video quality, remain a challenge.