Vehicule Detection

Project Writeup

Goals :

The goals / steps of this project is to create an vehicle detection pipeline following the steps bellow:

- 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.

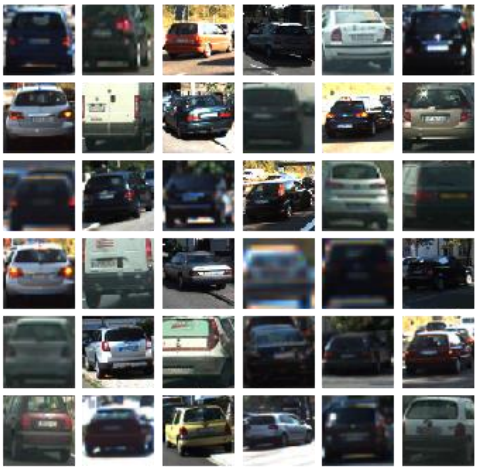
Rubric Points

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

Data Exploration

The DataProvider class offers a set of method to load vehicles images and non-vehicles images from disk to memory and provides them as numpy arrays. To do that, i used the following methods :

1. **deserialize(self,path, fext='.png'):** loads images from disk to memoy and return list of images
   - Path : the path of the vehicules or non-vehicules images
   - Fext : the images type the png is the default format
2. **files(path,fext='.png', show=True):** a static method that abstracts the call to the deserialize() method
3. **gridOfrandomimages(self,fnames, rows=6,cols=6,title=None):** Displays a set of images in a grid. Usely, it is called files method or Features class methods.



Random display of vehicle images

Random display of non-vehicle images

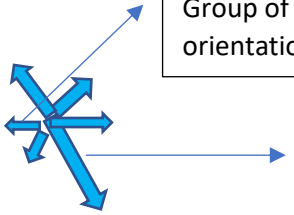
Histogram of Oriented Gradient (HOG)

The starting point for this, is **the Features class** in the **« Features.ipynb** » file. It provides different methods to extract different features :

1. **hogfeatures(self, image, orient, pix_per_cell, cell_per_block, show=False, feature_vec=True)**:

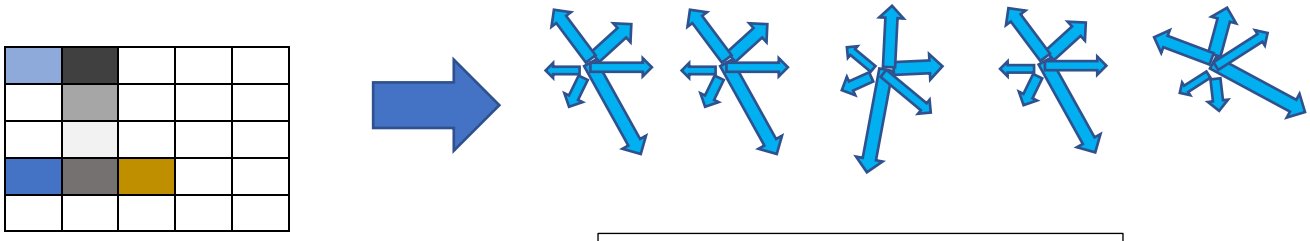
   Overview : the hogfeatures method provides a orientation signature shape of a specific image with the possibilites of tweeking parameters. It computes the gradient magnitude and direction of each pixel, group them in small cells (ex :8*8), compute the histogram of gradient or orientation of each pixel inside the cell (64 pixels), the gradient samples are distribued in n orientation bins and sumed up. The resulting histogram shows the longest bins which indicated the orientation of the pixels. Finally, each pixel in the image gets a vote on which histogram bin it belongs in based on the gradient

direction at that position. The final image resulting from this process is a shape signature representation accepting small variations through some parameters. The figures bellow illustrate the HOG process.



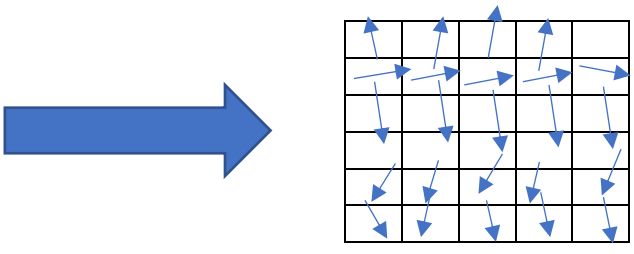
Group of pixels with low gradient orientation in the cell

Group of pixels with high gradient orientation in the cell



Pixelized image with each colored cell corresponding to group of pixels

Each star represents the pixels cell (left) histogram of gradient orientation



The image of the final step of the process is a shape signature representation

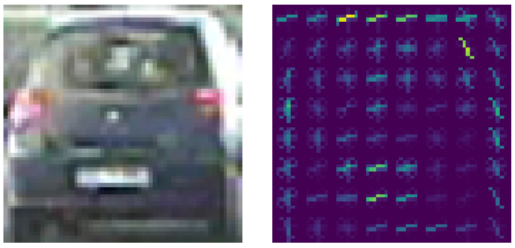
<u>Hogfeatures' Description:</u> the **hogfeatures** encapsulates the scikit-image builtin function to extract HOG features called Hog(). The **hogfeatures** takes parameters :

- orient : represents the number of orientation bin of the histogram (remember the sum of the orientation gradient of pixel per cells per groups) ➔ 9
- pix_per_cell : parameter specifies the cell size over which each gradient histogram is computed. This paramater is passed as a 2-tuple so you could have different cell sizes in x and y, but cells are commonly chosen to be square ➔ 8

- cell_per_block : parameter is also passed as a 2-tuple, and specifies the local area over which the histogram counts in a given cell will be normalized. Block normalization is not necessarily required, but generally leads to a more robust feature set. ➔ 2
- show=False :
- feature_vec=True

Features Extraction Workflow : The gethogfeatures(show=False) static method execute the following flow :

- Calls the DataProvider files static method to load the vehicles and non-vehicles images from disk to memory lists
- Calls the Features instance method hogfeatures with parameters listed above to get the a vector of hog features
- If show = True, it visualize the hog image
- Returns back the hog features



The show_featuresOfrndimage()'s static method executes twice the single_img_features(), before importing images from disk to memory and display randomly one vehicule and one non-vehicule images after :

- YCrCb conversion
- Hog_features extration

Choice of HOG parameters

After several tries, i retain the following values for the parameters :
orient = 9
pix_per_cell = 8
cell_per_block = 2
#bins = 32
Hog_channel = ALL

Classifier Training

The run() method of the Vehicle class executes the following flow control :

- Loads vehicules and non-vehicules images from disk to memory
- Extracts features (YCrCb, HOG) from images
- Builds X_features dataset using the vehicle and non-vehicule features
- Builds Y_features dataset using the ones array for vehicules and zeros for non-vehicules
- Splits Dataset into train dataset and test dataset
- Normalizes the train and test datasets using StandardScaler
- Makes different tries of Linear SVC training to choose the best C, penalty, loss Hyperparameters reagrding the accuracy for each combination of those parameters
    - 1. Penalty Value C: 0.08
    - 2. Penalty: l2
    - 3. Loss: hinge

2. **Windows Sliding**

The Windows class offers an instance method :

```
slide_window(self, img, x_start_stop=[None, None], y_start_stop=[None, None],\
        xy_window=(64, 64), xy_overlap=(0.5, 0.5))
```

I tried several window sizes, overlaps, and search areas. But, finally, i choose the following window size, overlap, and search areas which perform better on images and videos :

- xy_window = (96, 96)
- xy_overlap = (0.75, 0.75)
- y_start_stop = [400, 600]
- x_start_stop = [None, None]

Video Implementation

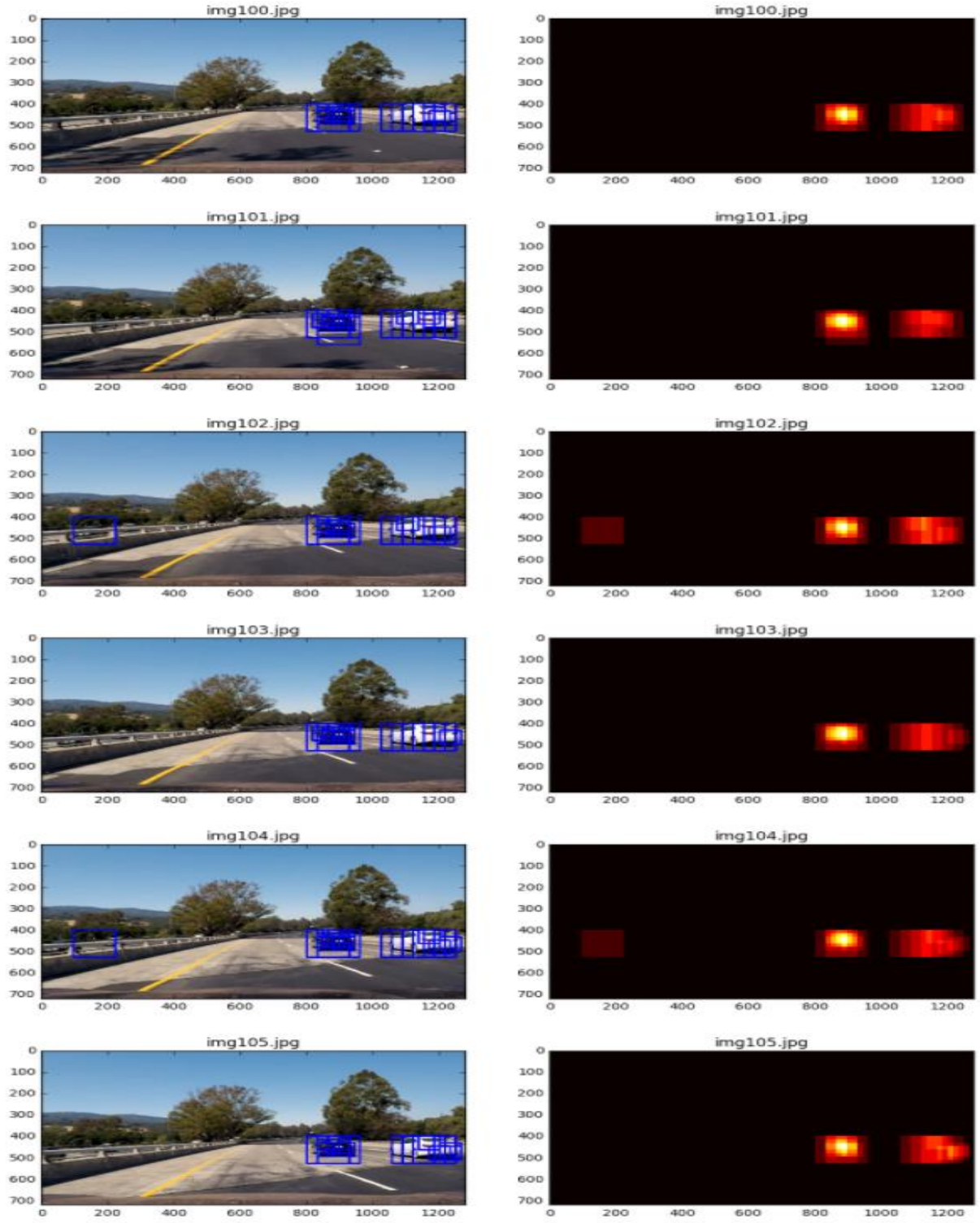
---

1. **Video File :**

The result of the pipeline is video file named : processed_project_video.mp4

2. **Heat Maps and False Positive**

When i applyed the pipeline so far, i had a lot of duplicate and false positive detections. I used heat-maps to remove both false positives and multiple detectors. Heat-map work as follows. 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.



Discussion

Drawbacks

1. The sliding windows algorithm is very complicated and need to be refined
2. The detection on the video frames seems to be slower even the processing is made on a video. The bboxes take sometimes to be resized to the detected vehicle. So therefore we deduce that this implementation could not fit a real time video capture

Future Works

- Find the root cause of the detection latency and implement a better solution
- Implement the solution with real time video capture.