In this project, goal is to write a software pipeline to detect vehicles in a video. The project comprises of below steps.

- Perform feature extraction using Histogram of Oriented Gradients (HOG), color transforms, binned color features and histograms of color techniques.
- Train a Linear Support Vector Machine (SVM) classifier on the extracted features.
- Implement a sliding-window technique with the trained classifier to detect vehicles in an image.
- Create a heatmap of recurring detections to reduce false positives. Finally Output visual display of bounding boxes around detected vehicles in a video stream.

Code structure & list of files

- **Visualize.ipynb** :- Contains code for visualizing different features, heatmaps, and bounding boxes from test images. It also executes pipeline function on test images.

- **lesson_function.py:-** Contains list of helper functions that were discussed in the course

- **classifier.py :-** Contains code for training a linear SVM classifier on the extracted car and non-car features. It also saves classifier and scaler in a pickle file.

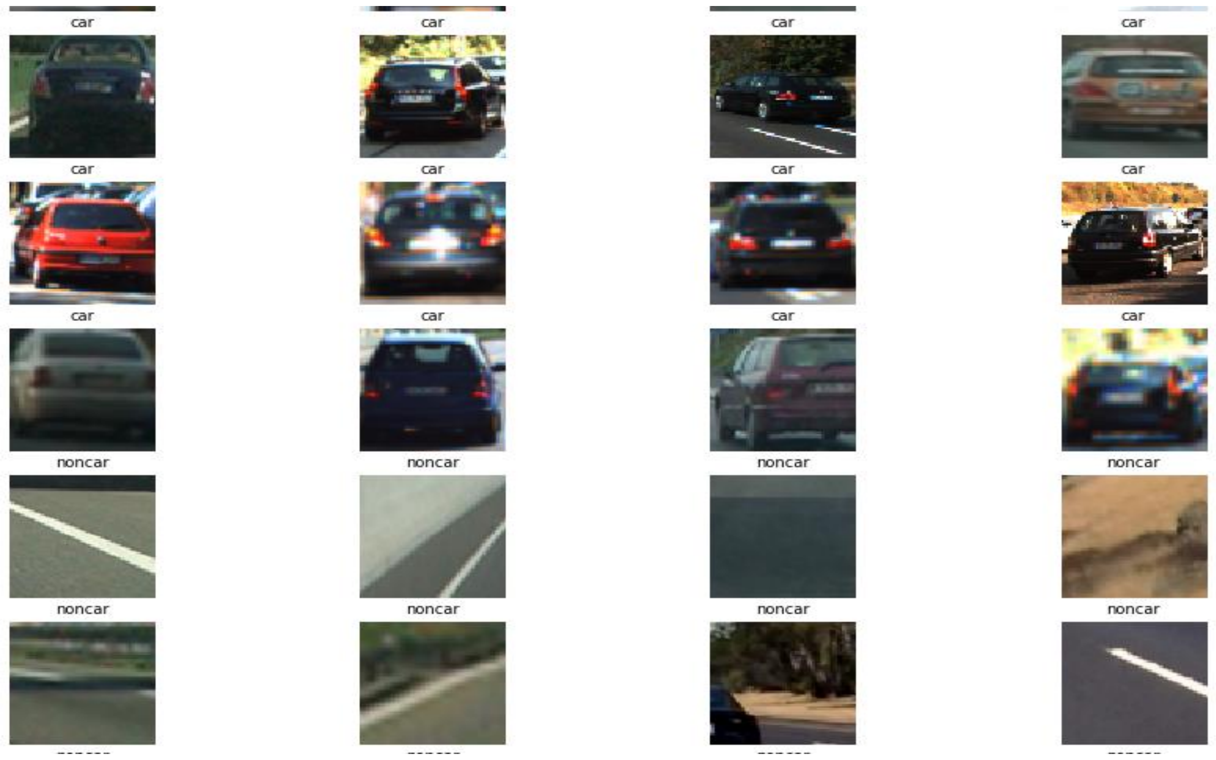
- **extract_data_features.py** :- Contains code for combining all image file names from the different data sources and extracting the features from them. Features are saved in a pickle file.

- **detect_vehicle.py** :- Contains code for detecting vehicles using the trained classifier and drawing bounding boxes around them in a video. It saves the output in **project_video_output.mp4** file

- Vehicle and non-Vehicle images are assumed to be saved in data folder. Due to their huge size, they are not uploaded in Github.

Data Exploration

The first step is to extract features from the dataset. Dataset in this project comprised of labeled images taken from GTI vehicle image database, the KITTI vision benchmark suite. All images are 64x64 pixels. The data is split into vehicles and non-vehicles subsets, and examples of a vehicle image (left) and non-vehicle image (right) can be seen below. Cells 1,2,3 of Visualize.ipynb notebook contain the code for it.

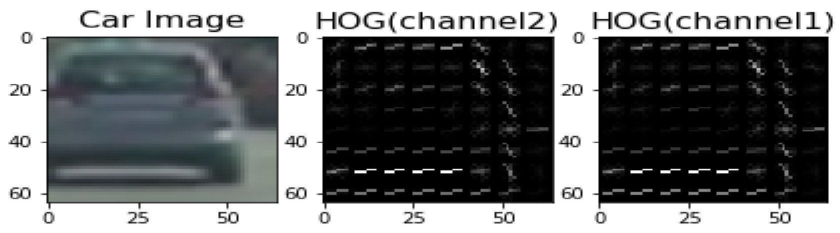


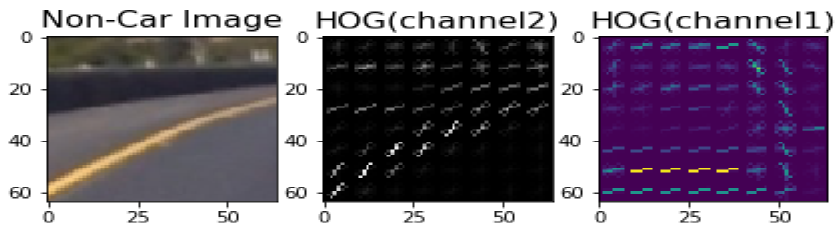
Extraction of HOG, color and spatial features

Next, features are extracted using a combination of Histogram of Gradients (HOG), spatial binning of the color image, and histogram of pixel intensity (color histogram). The HOG features are extracted using the sklearn hog() function defined in lesson_function.py , and the parameters used were found by trial and error. I found below parameters to be most effective. **(orient=9, pixels_per_cell=(8, 8), and cells_per_block=(2, 2)**. The code for extracting the HOG features can be found on lines 144-158 of lesion_function.py. The function bin_spatial() is used to resize the images to 16x16 resolution and transform it into a vector. color_hist() extracts the histogram of the images. I wrapped all of these functions in the extract_features() function, which outputs one feature vector for each image. This code can be found in lines 123-143 of lesson_function.py.

A visualization of HOG features of a vehicle (left) and non-vehicle (right) can be seen below:

```
<matplotlib.text.Text at 0xdb050f0>
```





Choice of parameters and channels for HOG

I experimented with many different combinations of color spaces and HOG parameters. In color space conversion, HLS color space appeared much better compared to RGB. RGB is not a good match under changing light channels. I used a low value of pixels_per_cell=(8,8). Using larger values of than orient=9 did not have a striking effect and only increased the feature vector. Similarly, using values larger than cells_per_block=(2,2) did not improve results, which is why these values were chosen. For HOG features, I used all three HLS channels, because using less than all three channels reduced the accuracy considerably. I didn't use spatial_bin() features in the final algorithm as it didn't contribute much to the accuracy and increased the feature vector size.

```
# Define feature parameters  (final set of parameters used in extract_data_features.py & detect_vehicle.py)
  color_space = 'HLS'
  orient = 9
  pix_per_cell = 8
  cell_per_block = 2
  hog_channel = 'ALL'
  spatial_size = (16, 16)
  hist_bins = 32
  spatial_feat = False
  hist_feat = True
  hog_feat = True
```

Training of linear SVM classifier.

Linear SVM was used as the classifier. It is available in scikit-learn package. 90% of the dataset was assigned to training the classifier and remaining 10% was used as test set to validate the mode. Video sequences have temporal correlation in the image frames. To eliminate it and to avoid overfitting the data, both training and test sets are randomly shuffled. As part of data preprocessing, Images are converted to HLS space. All 3 channels are used for calculation of HOF features. Color histogram with hist_bins=32 are extracted. All these features are concatenated. Next, these features are scaled using standard scaler.

After training the classifier, it resulted in an accuracy of 98% on test set. This code is available in classifier.py file. It took about 300 seconds to train on my i5 windows machine.

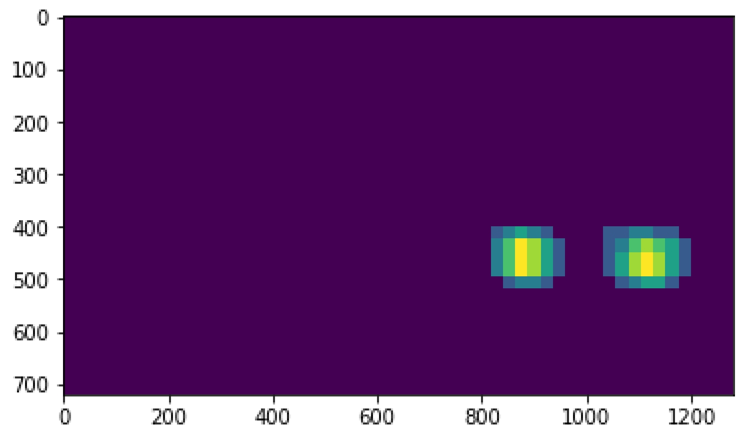
Implementation of sliding window search.

I used hog sub-sampling window search. It extracts hog features once and then it can be sub-sampled to get all of its overlaying windows. Each window is defined by a scaling 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 = 1 would result in a search window overlap of 87.5%. I used a scaling factor of 1.5 for the video pipeline. find_cars() function in detect_vehicle.py extracts features and makes predictions.

Multiple Detections & False Positives

During training, I observed that Window search returns overlapping detections for each car in the image and few frame have false positive detections. (for ex:- guardrail to the left was detected as ca). False positives occur on the side of the road, but also simple lane lines get detected as cars from time to time. Cars driving in the opposite direction also get detected in so far as a significant portion is visible. The false positives were filtered out by using a heatmap approach.

To make a heat-map, we "heat" (+=1) for all pixels within windows where a positive detection is reported by your classifier. Below is the output of heatmap computed on sample output image of prediction algorithm. By choosing appropriate threshold, we can see that below image has 2 cars visible.

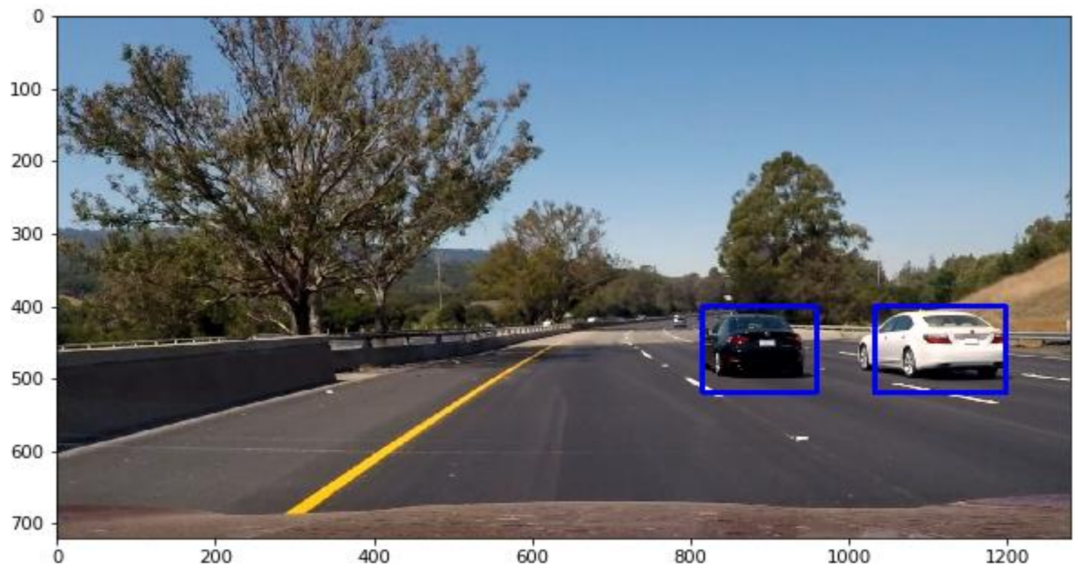


Heatmap generation code is available in lines 69-75 of detect_vehicle.py . "hot" parts of the heatmap corresponds to the location of a car. By imposing a threshold, we can reject areas affected by false positives. This is done in apply_threshold() function in detect_vehicle.py. After few rounds of trails, I choose the value of 2.5 for the threshold.

I used scipy.ndimage.measurements.label() to identify individual blobs in the heatmap. I then assumed each blob corresponded to a vehicle. Finally, I constructed bounding boxes to cover the area of each blob detected. Using the output of scipy.ndimage.measurements.label() on this integrated heatmap results in an image of labels. This is done in draw_labeled_bboxes() function in detect_vehicle.py

Finally, detected boxes are overlayed on input image and this is returned as output of pipeline function.

Sample output of pipeline function



<u>Problems / issues encountered and outlook</u>

1. My pipeline works well on project video. However, it does not process the video in real time. (processing time is greater than video length). I think it can be made real time if evaluation of feature vectors in done in parallel.
2. Some false positives still remain after heatmap filtering. This should be improvable by using more labeled data.
3. My pipeline currently does not detect vehicles which are relatively far away. This can improved by updating y_start and y_stop parameters of sliding window search algorithm. Also, it might be helpful to switch classifier to use non linear kernels such as rbf when larger portion of image is scanned.
4. Another issue my pipeline faces is detecting cars which are occluded by another vehicle. There is a segment of the project video where two cars driving next to each other are detected as one vehicle, producing only one bounding box around the two cars