Vehicle Detection and Tracking

Introduction

Object detection and its localization is a very important task in self driving car, through which car is able to detect and localize objects of interest. The goal of the project is to detect cars/vehicle in the image/video frame and localize them by bounding with rectangular boxes. Below are the summaries of the task performed to identify/detect cars/vehicle and track them in successive frames of the video:

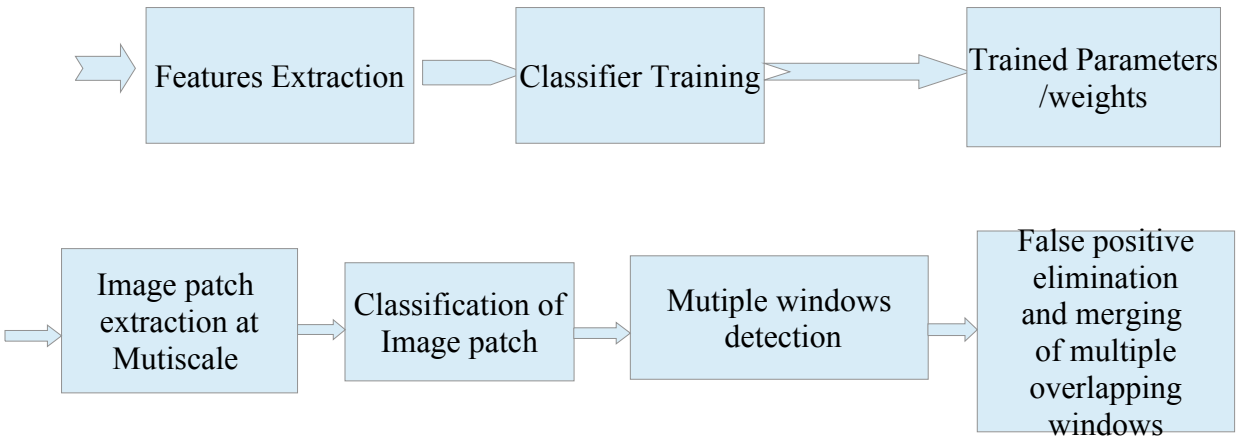
- Extraction of image feature vectors like HoG features, spatial raw pixels, color channels' histogram
- Train a classifier like linear SVM using the extracted feature vectors
- Identify object of interest/car through multi-scale sliding windows approach
- Detection and elimination of false positive and merging multiple overlapping detected windows
- Localizing and drawing bounding boxes around the detected instances of objects/cars

Below sections describe the above mentioned task in details.

Pipeline

The conventional approach of vehicle/object detection pipeline involves features extraction from the image, training a classifier with labeled training data sets using extracted features' vector, performing inference using trained classifier parameters/weights on different part of the image through sliding windows using different scales. The whole task is divided as:

1. Feature extraction and training a classifier
2. Classifying/Predicting image part/patches using trained classifier



The feature extraction can be performed using convolutional neural network (CNN) or using conventional method where HoG transform, color histogram, spatial raw pixels are used to create

features vector from the image. In this project, conventional method like HoG, color channel histogram and spatially binned raw pixels are used to create features' vectors for the image. Linear SVM classifier is used for the classification task in the project, though other type of classifier like Non-linear/ker nelized SVM, deep neural networks etc can be used. The sliding windows at different scales are used to extract image patches from the image frame. These windowed image patches are fed to object detection pipeline to classify them as vehicle and non vehicle image regions. Thus, they form multi-scale windowed regions that are classified as vehicle object instances. Since there are multiple overlapping windows which would be detected as positive, so those overlapping regions are merged using heat map method as described later and one final window is created using those overlapping areas.

"object_detect_pipeline.py" file has the object detect pipeline implementation as :

- def vehicle_detect_pipeline(..): It implements vehicle detect pipeline using APIs provided by other modules like "color_feature", "object_detect"
- def object_detect(..): This is the main API which performs vehicle detection on image frame or video frames.

Feature Extraction

In order to train and subsequently predict object classes, image features vector needs to be obtained. There are multiple image features that can be considered to be suitable candidates for feature vectors such as sub-sampled raw image pixels in different color space, color channels' histogram for different color space like HLS,HSV, RGB etc, histogram of gradient orientations (HoG) transform of the color channels. The idea of defining suitable features should address below valid concern:

- The features should be distinct for vehicle and non vehicle classes
- Those features should perform well in different lighting, environmental condition

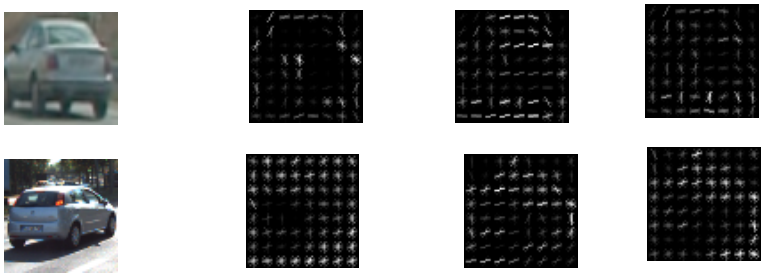
In the project, combination of multiple features are used to form image features vector as below:

- HoG features obtained using all the channels of HLS color space. 2X2 cells per block, with 8x8 pixels per cell and 9 orientations are used for HoG transform
- Histogram of all the color channels in HLS color space using 32 bins
- Spatially binned as 32x32 image pixels in HLS color space.

Above feature configs are selected based on the experimentation which resulted in better accuracy for object detection

"color_feature.py" file has the implementations of different functions that extract image features vector as needed for the classification task. Below example images show HoG feature and color histogram for all the channels of HLS color space of car/vehicle and non car images

Image class H channel L-Channel S-Channel



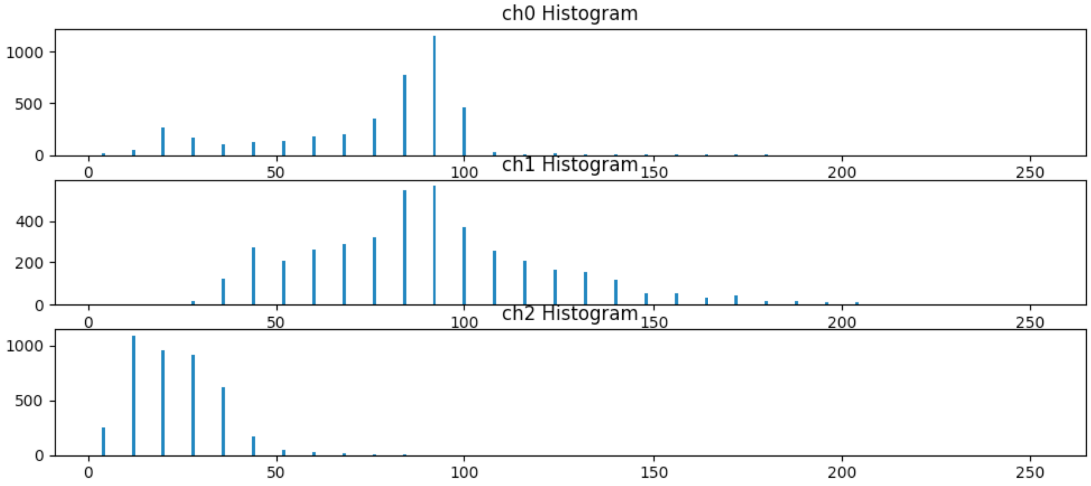


Illustration 1: Example of HLS Color channels histogram for a car image

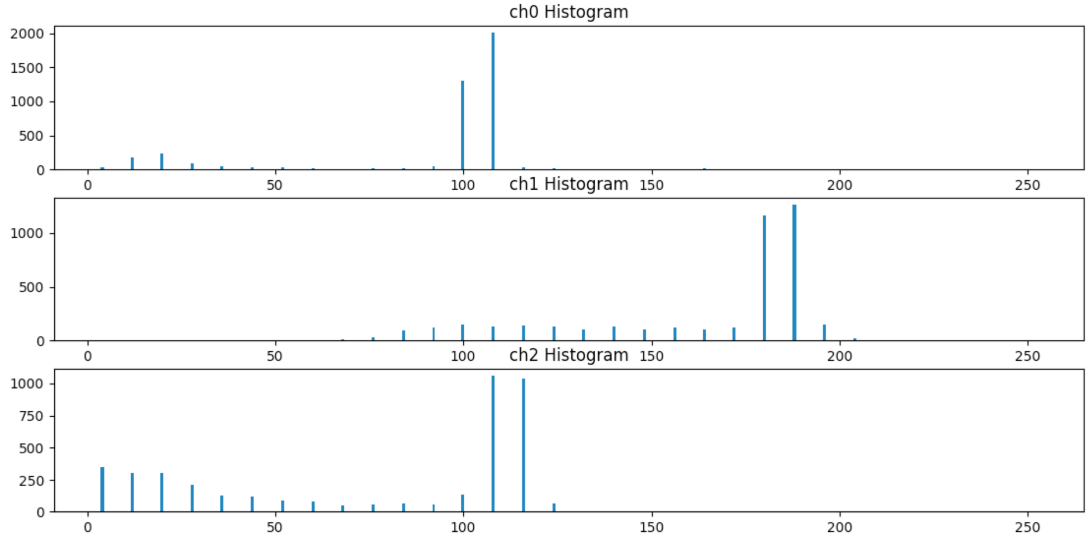


Illustration 2: Example of HLS Color channels histogram for non car object

Classification

Linear SVM classifier is used for the classification of vehicle and non vehicle classes. Using training

data sets as provided for the project, linear SVM classifier is trained and its trained weights/parameters are saved in a file which is used for the classification task later in object detection pipeline. Before training process, features' vector extracted from all the training image data are “mean” and “scale” normalized.

“classify.py” file implements all the necessary functions needed for training the linear SVM classifier, the details of important APIs are as below:

- def get_image_features(img_database, features_cfg): It implements image features extraction and their normalization for all the training images
- def train_svc_classifier(X, y): It implements training of the linear svm classifier

Once the classifier is trained, its weights vector and normalization statistics like mean and std deviation for each feature vector are saved in a pickle file(example file: svc_classifier_new.pkl)

Object Detection and Localization

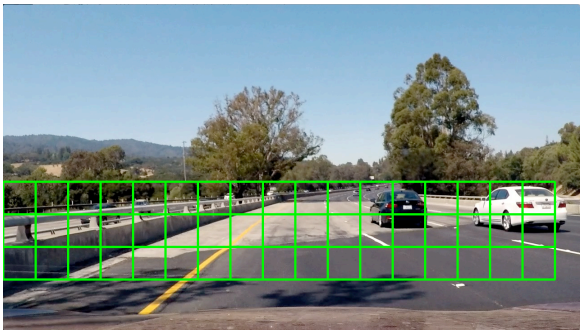
Sliding Window

In order to detect regions in an image frame that contain vehicle class object instances, the image RoI is searched using sliding windows at different scales. In order to optimize and reduce the number of search windows, top half of the image is left out since that image area generally contains trees and sky. Also, different scale windows are used for different area of the image, for example bigger scale window is used near to bottom region and small scale window is used for the area that is far away from the bottom of the image. Below example images show the different scale window used for search operation. The example image shows window drawn with 0.5 x and y overlapping which is just an example, in the actual search, the overlap of 7/8 and 6/8 are used for different scale images.

Illustration 3: Example of sliding window of size 64x64 with (0.5, 0.5) overlap and regions where they used

Illustration 4: Example of sliding window with size 96x96 with (0.5, 0.5) overlapping and regions where they used

Illustration 5: Example of sliding window of size 144x144 with overlap of (0.5, 0.5)



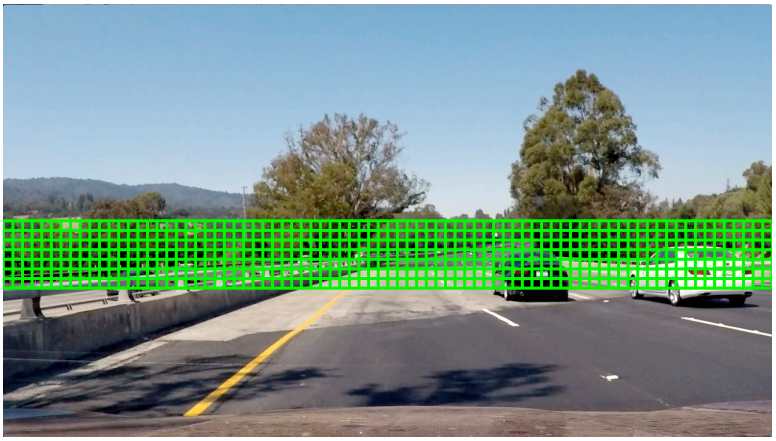


Illustration 3: Example of actual overlap used, (0.75, 0.75) overlap for 64x64 window size and its region of search

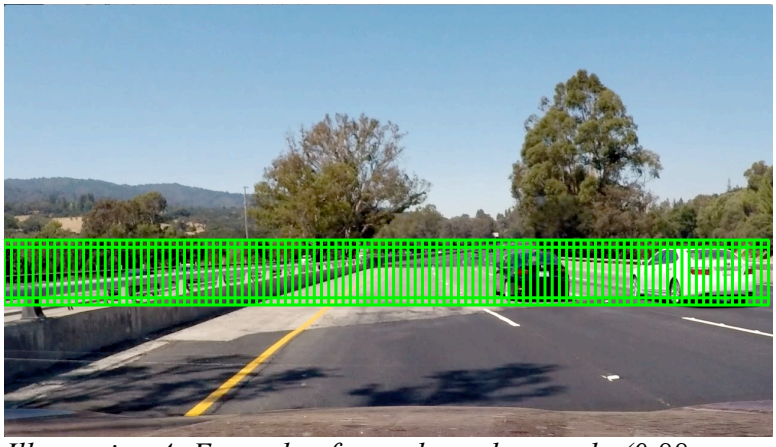


Illustration 4: Example of actual overlap used , (0.88, 0.88) overlap for 96x96 window size and its region of search

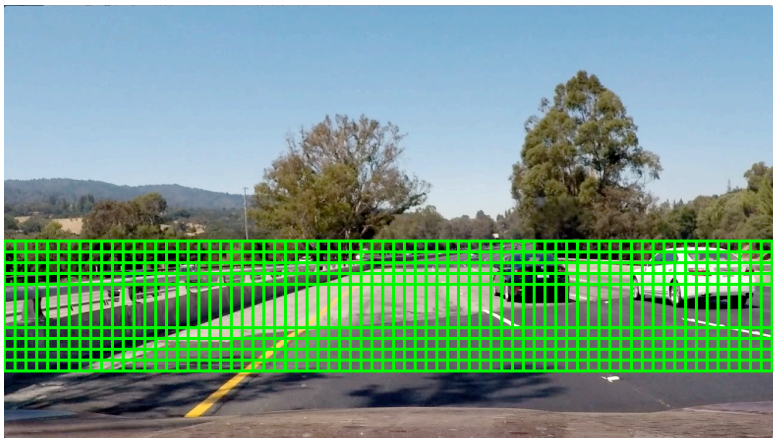


Illustration 5: Example of actual overlap used (0.88, 0.88)overlap for 144x144 window size and its region of search

"object_detect.py" file contains function that implements sliding window search on image region.

def detect_object(..) and def detect_object_multiscale(..) are the main functions

Object detection at multiple scale

At each scale of windows, a particular region of the image, as shown in above example images, is searched for the vehicle class object instances. Once the image patch extracted from the regions are detected as positive through svm classifier, the windows' coordinates are added in detected window list that are used to draw boxes around the detected instances of the vehicle. Below example images show detected car windows using multi-scale window search. In the project, 64x64, 96x96, 144x144 sliding windows are used to search vehicle and non vehicle regions.

Illustration 6: Car detection windows using sliding window of size 96x96

Illustration 8: car detection windows, using sliding window of size 144x144

Illustration 9: Example image where all multi scale detected windows are drawn

Elimination of False positive, Merging multiple overlapping windows

In order to eliminate false positive detection and merge all the overlapping detected windows, a heat map image is created by adding "1/one" to zero initialized pixel intensity/value for each detected window, resulting in hotter window area based on the confidence of detection(as happens when multiple detected windows show same area). The heat map image is further thresholded that helps in eliminating any false positive detection. Below examples show generated heat map image and its thresholding that eliminates false detected window on the left side of the image where there is only one window detection(less confidence). "object_detect.py" file contains function that generates heat map and false positive detection through thresholding the heat map image

- def generate_heat_map(..), def apply_threshold(..)

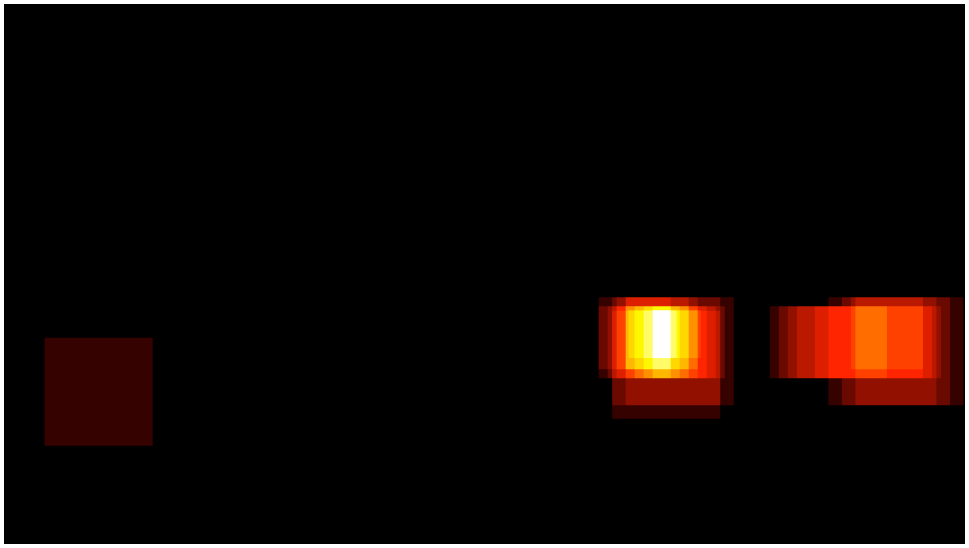


Illustration 10: Example heat map image generated by adding "1" to pixel value for a detected window region

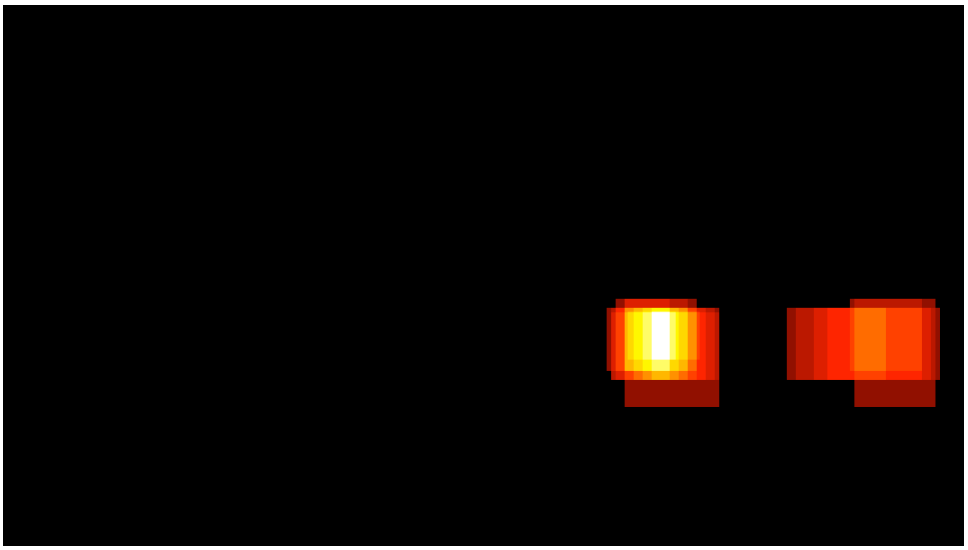


Illustration 11: Thresholded heat map image that eliminates false positive window on left side of the image

Window and Label creation using thresholded heat map image

Using thresholded heat map image, labels for object instances are obtained using "scipy.ndimage.measurements" library package which provides number of labeled objects and a labeled gray scaled image with different pixels values for each label. This helps in extracting bounding boxes for each object label. Once the bounding boxes are obtained, they are drawn on the image frame as shown in below example image. In video frames, the heat map for the current frame is obtained by accumulating previous frames' heat map images with current frame heat map. It helps in eliminating false positive window detection and also helps in smooth object tracking in the video

Illustration 12: Example of vehicle localization with final bounded boxes

Code organization and package details

Below are the code files that implements different function used in the projects:

- color_feature.py: It implements the feature vector extraction from the image
- classify.py: It implements linear SVM classifier, feature vector normalization and training of the classifier
- object_detect.py: It implements multi scale sliding window search for object detection and its localization, false positive elimination through heat map image generation
- object_detect_pipeline.py: It implements main function for object detection pipeline for image and video file
- *svc_classifier_new.pkl: classifier saved file that is used to load the classifier parameters in object detection pipeline API.*

Each module has its own test application which tests module specific functionality.

Command line to be used to run the object detect pipeline on test images and video:

python object_detect_pipeline.py <svm classifier file> <video(1) or image file(0) flag> <input test-image or video file name > <output file save path for image or video file name to be saved>

Example:

Image File:

python src/object_detect_pipeline.py ./svc_classifier_new.pkl 0 CarND-Vehicle-Detection/test_images/test6.jpg ./

Video file:

```
python src/object_detect_pipeline.py ./svc_classifier_new.pkl 1 project_video.mp4
project_video_output.mp4
```

Conclusion

In the project, image features such as HoG transform, color channel histogram and spatially binned image pixels are successfully used along with linear SVM classifier for vehicle detection and its localization in the image. The method works well on test images and video. But it is also observed that the method used doesn't work well on the images or videos captured during night time due to poor lighting condition. Further, it fails to draw two separate bounding boxes, when two cars are occluded or they are close/near by as can be seen in the output video due to limitation in the heat map generation method. As a next step, newer methods of object detection and its localization should be explored such as based on CNN.