Vehicle Detection & Tracking

using Hog, Color Histogram, & Spatial features

ZEESHAN ANJUM

2017

# Objective:

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.
* 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.

# Dataset:

I am using the Udacity’s vehicle and non-vehicle dataset with 64x64 size image.

Some sample from the dataset



Non-Vehicle but possible images on road



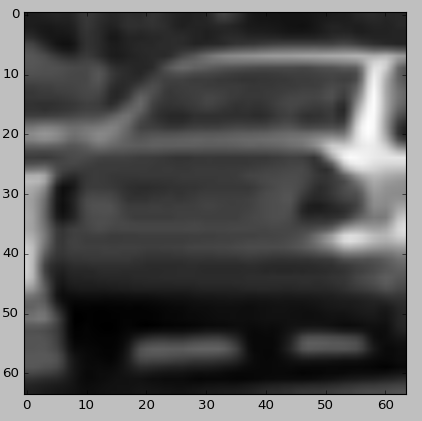
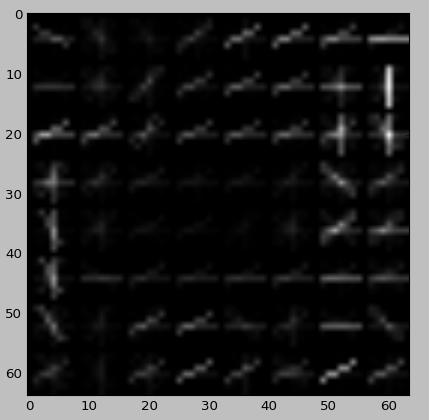
# Feature Extraction:

I used HoG, spatially binned color and color histograms.

## Histogram of Oriented Gradients:

The histogram of oriented gradients (HOG) is a feature descriptor used in computer vision and image processing for object detection. The technique counts occurrences of gradient orientation in localized portions of an image. I am using HOG feature extraction defined in FeatureExtractor.py file, on line no 11 in function \_get\_hog\_features. I used starting values for hog parameters default from Udacity’s class and tweak some of these to get the desired result. I tried RGB, LUV and YCrCb color channel and choose YCrCb because it gives better results than other.

Here is an example using the **YCrCb** color space and HOG parameters of orientations=8, pixels per cell=(8, 8) and cells per block=(2, 2):

I tried other color-spaces as well like HLS and RGB but other color-spaces are affected by illumination variance, RGB results are nice but not good as YCrCb.

I started with 32 histogram color bins and 32 spatial size bins. And reduced it to half to increase performance, I tried to lower the pixel per cell value but I hit memory threshold and lower values also took much time to compute, increasing value gives lots of false-positive result, for me 16 is good size for both feature bins.

I reduced detection timing by half by replacing SVM with MLP Classifier, and reduce the bins count to 16 for Histogram of color bin and spatial size.

# Classification

I created a class for Vehicle Classifier in VehicleClassifier.py file that loads images, extract features defined in FeatureExtractor.py file, train dataset, save and load model, predict, and return score for certain features coming from the feature extractor.

In FeatureExtractor class I defined a function ‘extract\_features’ at line no 83 in file FeatureExtractor.py which reads files and extract spatial, color histogram, and hog features and concatenate as a single feature for a single sample.

I tried SVM (Support Vector Machine) Linear Classifier and MLP (Multi-layer perceptron) classifier, and choose MLP for this project as this has very good accuracy and performance over SVM for this type of case.

For training, I normalized the feature vector with zero mean and unit variance. In the VehicleClassifier.py file, line no 93-95 I normalized the features before stacking into 1D feature vector. I used MLP with default parameters as it provides me desired result.

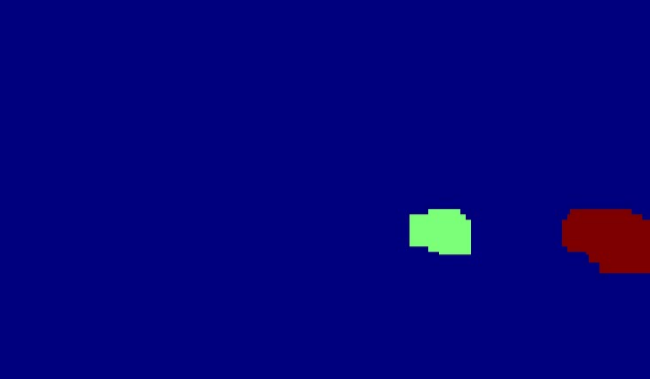
## Sliding Window Search

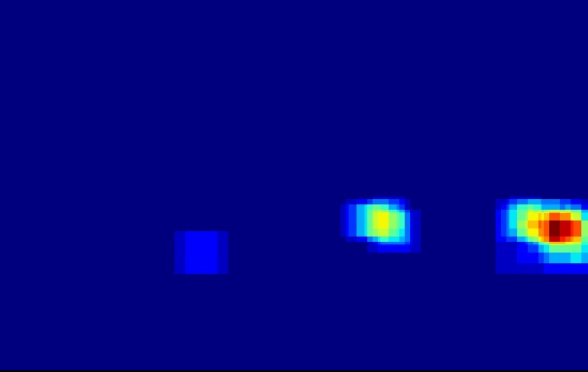
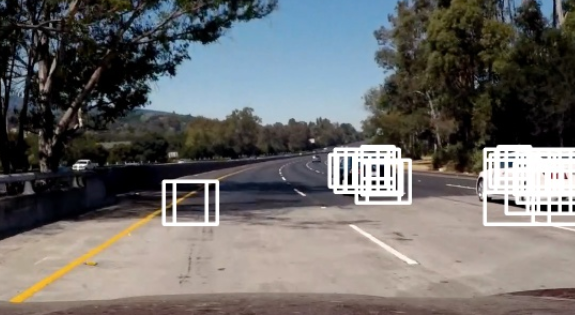
As we are detecting vehicles on road only, which almost always appears on the less than half height of the image as it is placed at the center front of the camera. So, I split the area by 50% of the height.

As we can see object seems small near horizon and larger near to the camera, So I use small searching windows only near horizon, medium between horizon and bottom of the image. And large near the bottom of the camera.

For final result, I used HOG features, histogram of color channels, and spatially binned color and combined these into a single feature vector. I preserve some frames to approximate and smooth the movement of rectangle marker on detection.

After passing an image through classifier it predicts vehicles multiple times and some non-vehicle areas as well, as classifier might match with some of the vehicle feature in the image. Then I generate the heatmap for all the detected areas and apply threshold of 2 overlapping rectangles.



As we can see on left, multiple frames are detected as vehicle by the classifier, and some of the results are false positive. So, I used heatmapping technique in VehicleDetector.py file line no: 80 which takes all rectangle and assign pixels to the detected area in image, high intensity pixels considered as detected vehicle, then I applied thresholding to eliminate low heatmap (on line: 89 in VehicleDetector.py file)

## Video Processing:

Detecting vehicle on each frame make wiggly window movement on vehicles. So I used average sum of 2 heatmap and then apply threshold to make the detected window movement smooth.

# Result

Here is link of video output:

<https://youtu.be/SJmEnVunI0s>

# Discussion

I think resolution of image should be reduced at least by half of the actual resolution, this algorithm would fail in different lighting condition, and different orientation of vehicles. To resolve this problem, we need to augment the dataset and use more data with different orientations. It also has problem of merging heatmaps of two vehicles together and output it as single vehicle, to solve this problem we can predict car position using some predictive algorithm. This pipeline is not for the real-time use. But good to start with. It can be optimized by reducing the input image size and the number of searching windows, feature size and tracking the detected vehicle and classify some area of the sequence per frame, for example: 20% bottom left in first frame, 21% to 40% in next frame, after 100% combine their heat-map to produce the final heat-map.