# Histogram of Oriented Gradients (HOG)

## Explain how (and identify where in your code) you extracted HOG features from the training images. Explain how you settled on your final choice of HOG parameters.

|  |
| --- |
| Explanation given for methods used to extract HOG features, including which color space was chosen, which HOG parameters (orientations, pixels\_per\_cell, cells\_per\_block), and why. |

|  |
| --- |
| Resubmission 1 |
| In this resubmission I have transferred the following functions to an .py file to have shorter and readable code in jupyter notebook.   * bin\_spatialI * convert\_color * color\_hist * get\_hog\_features   The name of python file is features.py and it is imported in Jupyter notebook.  import features |

**Histogram of Oriented Gradient**

I have a function for extracting the Histogram of Oriented Gradient with following header which has been called under “4. Extracting the Histogram of Oriented Gradient (scikit-image HOG)” cell.

|  |
| --- |
| *def get\_hog\_features(myimg, orient, pix\_per\_cell, cell\_per\_block,vis=False, feature\_vec=True):* |

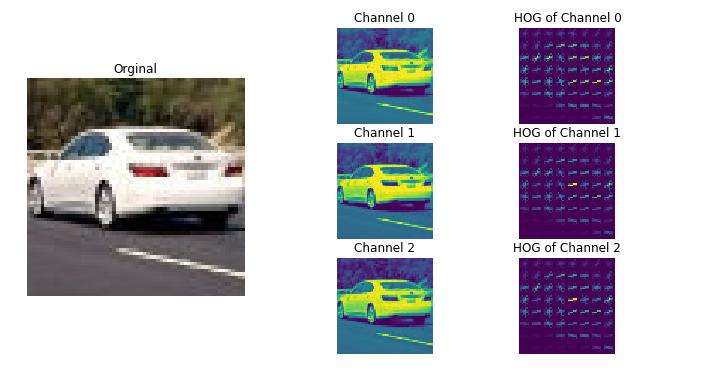
The next figure is the visualization of the output of the *get\_hog\_features* function.

Figure 1: orient=9, pix\_per\_cell=8, cell\_per\_block=2

The main function which is used to extract the image HOG is hog function as in training codes. The get\_hog\_features function has been used in two different part of this project.

* Under “Extract Dataset's features” cell: via calling extract\_features function from training materials.

|  |
| --- |
| car\_features = extract\_features(cars, cspace = color\_space,  spatial\_size = spatial\_size,  hist\_bins = hist\_bins,  hist\_range = hist\_range)  notcar\_features = extract\_features(notcars, cspace = color\_space,  spatial\_size = spatial\_size,  hist\_bins = hist\_bins,  hist\_range = hist\_range) |

* Under “Function for processing each frame” cell: via calling find\_cars function.

|  |
| --- |
| predicated\_windows = find\_cars(image, ystart, ystop, scale, svc, X\_scaler, orient, pix\_per\_cell, cell\_per\_block, spatial\_size, hist\_bins) |

As we can see in the two previous code snippets we have variables which determine which feature must be considered for training and for vehicle finding as well. These variables have been defined under “Global values” cell.

|  |  |  |
| --- | --- | --- |
| Variable name | Value | Description |
| spatial\_feat | True or False | True means spatial feature is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts. |
| hist\_feat | True or False | True means Histogram of Color is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts. |
| hog\_feat | True or False | True means Histogram of Oriented Gradient is considered in feature of image for training and vehicle detection. False means this feature must not be considered in both parts. |

### About orient, pix\_per\_cell, cell\_per\_block variables

I have tested the “Extracting the Histogram of Oriented Gradient (scikit-image HOG)” code snippet not only for 8 pixels per cell but also for 4 and 2 pixels per cell and all of them with 2 cells per block.

|  |  |
| --- | --- |
| Figure 2: orient=9, pix\_per\_cell=2, cell\_per\_block=2 | For example, this figure at the left side is a sample of pixel per cell 2 and cell per block 2.  The pattern of vehicle is clearer than the previous one, but the performance is worse than pix\_per\_cell=8, cell\_per\_block=2  Therefore, I have developed my project with orient = 9, pix\_per\_cell=8 and cell\_per\_block=2 which are defined and assigned under “Global values” section in Jupyter Notebook. |

### About color space variable

I have also tested all the color spaces which have been defined in project as shown in following figure.

|  |  |
| --- | --- |
| Figure 3: RGB color space | Figure 4: HSV |
| Figure 5: LUV | Figure 6: HLS |

|  |  |
| --- | --- |
| Figure 7: YUV | Figure 8: YCrCb |
|  |  |

But I have decided to use YCrCb color space.

## Describe how (and identify where in your code) you trained a classifier using your selected HOG features (and color features if you used them).

|  |
| --- |
| The HOG features extracted from the training data have been used to train a classifier, could be SVM, Decision Tree or other. Features should be scaled to zero mean and unit variance before training the classifier. |

In the following figure we have the flowchart of my training and test Linear SVC Classifier mechanism.



This part has been developed under “Load Training/Test Dataset” cell in Jupyter Notebook.

All car and notcar pictures are loaded but with an If statement they are differentiated.

The training code is started from cell “Global values”, where I have defined the variables and values for training the model as in following.

|  |
| --- |
| color\_space = 'YCrCb'  orient = 9  pix\_per\_cell = 8  cell\_per\_block = 2  hog\_channel = "ALL"  spatial\_size = (16, 16)  hist\_bins = 128  spatial\_feat = True # Spatial features on or off  hist\_feat = True # Histogram features on or off  hog\_feat = True # HOG features on or off  hist\_range = (0, 256) |

In “Load Dataset” cell I have defined two loops to make two list of vehicle and non-vehicle image names for training and testing the linear SVC model.

|  |
| --- |
| images = glob.iglob('./all/vehicles/\*\*/\*.png',recursive=True)  cars = []  notcars = []  for image in images:  cars.append(image)  images = glob.glob('./all/non-vehicles/\*\*/\*.png',recursive=True)  for image in images:  notcars.append(image) |

After defining variables and preparing the vehicle and non-vehicle list, I extract the feature/s of the car and non-car images with *extract\_features* function and create an array stack of the extracted features to pass to *StandardScaler().fit()* function.

|  |
| --- |
| # Scale the feature vectors  X = np.vstack((car\_features, notcar\_features)).astype(np.float64) |

*StandardScaler().fit()* fits the scale per-column and I apply the calculated scaler to the created array stack as shown below.

|  |
| --- |
| # Scale the feature vectors  X = np.vstack((car\_features, notcar\_features)).astype(np.float64)    # Define the labels vector  y = np.hstack((np.ones(len(car\_features)), np.zeros(len(notcar\_features))))    # Fit a per-column scaler  X\_scaler = StandardScaler().fit(X)    # Apply the scaler to X  scaled\_X = X\_scaler.transform(X) |

After getting the scaled data, they must be split into two Train and Test dataset.

|  |
| --- |
| rand\_state = np.random.randint(0, 100)  X\_train, X\_test, y\_train, y\_test = train\_test\_split( scaled\_X, y, test\_size=0.2, random\_state=rand\_state) |

And then the model is trained.

|  |
| --- |
| Resubmission 1 |
| Another point that I have considered in this resubmission is the fine tuning of the C parameter of the LinearSVC classifiert as following:  svc = LinearSVC(C=0.0005) |

|  |
| --- |
| svc = LinearSVC(C=0.0005)  svc.fit(X\_train, y\_train) |

|  |
| --- |
| Resubmission 1 |
| Another new logic that I have developed in code is saving the variables and model because it was not time efficient for me to run the whole code each time that I wanted to develop the step after training the mode. This logic is developed in “Save model” cell.   |  | | --- | | obj = {  "svc": svc,  "scaler": X\_scaler,  "orient" : orient,  "pix\_per\_cell": pix\_per\_cell,  "cell\_per\_block": cell\_per\_block,  "spatial\_size": spatial\_size,  "hist\_bins": hist\_bins,  "color\_space": color\_space,  "hog\_channel": hog\_channel,  "spatial\_feat": spatial\_feat,  "hist\_feat": hist\_feat,  "hog\_feat": hog\_feat,  "hist\_range":hist\_range  }  pickle.dump(obj, open('svc\_pickle.p', 'wb')) |   And in “Read model” cell there is a logic for reading/reloading the saved variables and model. |

# Sliding Window Search

## Describe how (and identify where in your code) you implemented a sliding window search. How did you decide what scales to search and how much to overlap windows?

|  |
| --- |
| A sliding window approach has been implemented, where overlapping tiles in each test image are classified as vehicle or non-vehicle. Some justification has been given for the particular implementation chosen. |

**sliding**

The sliding window mechanism, which I have used is from training materials but with some changes.

**Scale**

In “Read model” cell I have considered 7 different scales. Because vehicles in different distances have different scales.

**Overlapping**

Instead of overlapping the following code snippet has been used in “Using LinearSVC Classifier to detect vehicles” cell in find\_cars function.

|  |
| --- |
| cells\_per\_step = 2 # Instead of overlap, define how many cells to step |

All the sliding window, overlapping and scale has been considered in find\_cars function.

## Show some examples of test images to demonstrate how your pipeline is working. How did you optimize the performance of your classifier?

|  |
| --- |
| Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.) |

# Video Implementation

## Provide a link to your final video output. Your pipeline should perform reasonably well on the entire project video (somewhat wobbly or unstable bounding boxes are ok as long as you are identifying the vehicles most of the time with minimal false positives.)

|  |
| --- |
| The sliding-window search plus classifier has been used to search for and identify vehicles in the videos provided. Video output has been generated with detected vehicle positions drawn (bounding boxes, circles, cubes, etc.) on each frame of video. |

The output video is available.

## Describe how (and identify where in your code) you implemented some kind of filter for false positives and some method for combining overlapping bounding boxes.

|  |
| --- |
| A method, such as requiring that a detection be found at or near the same position in several subsequent frames, (could be a heat map showing the location of repeat detections) is implemented as a means of rejecting false positives, and this demonstrably reduces the number of false positives. Same or similar method used to draw bounding boxes (or circles, cubes, etc.) around high-confidence detections where multiple overlapping detections occur. |

|  |
| --- |
| Resubmission 1 |
| Another logic that I have changed is in this part. The reviewer suggested to use deque object.  from collections import deque  history = deque(maxlen = 8)  ...  history.append(current\_heat\_map)  ...  The length of deque is 10 in my code to collect the predicted and actual boxes from each frame. |

To have a reliable result I have changes the heat-map technique a little via collecting all founded boxes form each scale iteration to filter the false positives.

I have considered two extra features in my development for

* Heat
* Threshold (was explained before under “Some discussion is given around how you improved the reliability of the classifier i.e., fewer false positives and more reliable car detections (this could be things like choice of feature vector, thresholding the decision function, hard negative mining etc.)”)

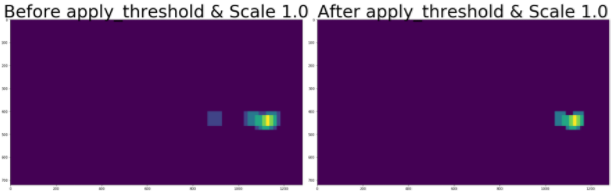
In sliding step each box, which was predicted as a car object, was added or appended to the box list and then we added heat to each box of list. Through the implementation I have noticed if I use threshold=1 or threshold<1 remain some flecks in heat after applying threshold which are considered as a box when I called label function.

Therefore, I decided to make the boxes which are mostly predicted as car object more heater. This code snippet has been developed under “Using LinearSVC Classifier to detect vehicles in an Image via defined function” cell in “process\_each\_image” function.

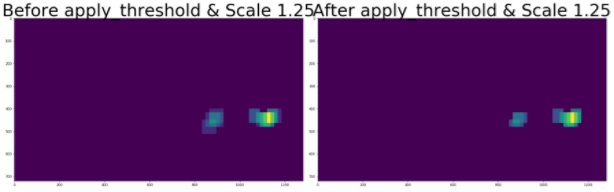
In brief and short description of the pipeline. At first, we don’t know the scale with which we should process the image, but we can guess min scale can be 1.0 and max scale can be 2.0.



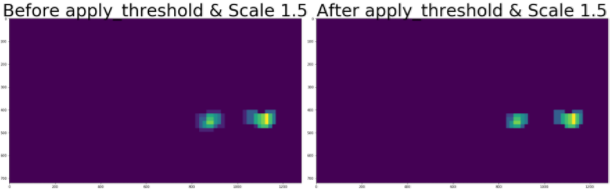
The next figure left is heat variable before applying threshold and left if after applying threshold. We a small fleck is removed.



We see the removed fleck is appeared again because it’s predicted again as car and it’s not removed from heat after applying threshold.



As we see the boxes which are really positive are detected again and the flecks are getting bigger it means the boxes which will be drawn around the objects will surround the whole object.



Furthermore, I sum the heat of current loop to the previous heat therefore each fleck which is a correct prediction get heater and bigger.

*#------------------------------------------------------*

*# This line is for Multiple Detections & False Positives*

*#------------------------------------------------------*

heat = np.zeros\_like(image[:,:,0]).astype(np.float)

scale = 1.0

heat\_threshold = 0

percentage\_fraction = 5

**while** (scale <= 2):

*# Find boxes which are predicted as car object via find\_carII*

*# function and save in box\_list*

*# Add heat to each box in box list*

heat = add\_heat(heat,box\_list)

*# Apply threshold to help remove false positives*

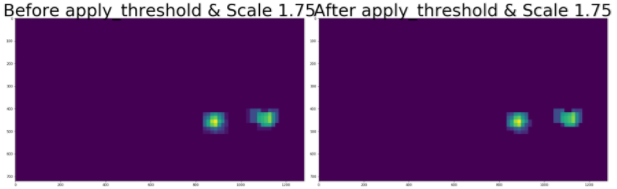
heat\_threshold = (max( heat.max(axis=0)) \* percentage\_fraction)/100

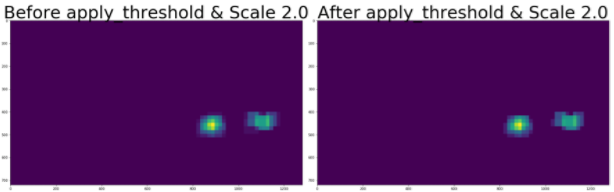
**if** heat\_threshold < 1 : heat\_threshold = 1

heat = apply\_threshold(heat,heat\_threshold)

scale = scale + 0.25

box\_list.clear()





And finally, we can draw the boxes which are extracted via label function.

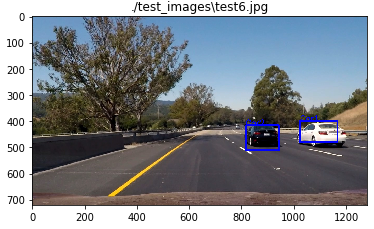
*# Visualize the heatmap when displaying*

heatmap = np.clip(heat, 0, 255)

*# Find final boxes from heatmap using label function*

labels = label(heatmap)

draw\_img = draw\_labeled\_bboxes(np.copy(image), labels)



# Discussion

## Briefly discuss any problems / issues you faced in your implementation of this project. Where will your pipeline likely fail? What could you do to make it more robust?

* I had problem with detecting the notcar objects. Maybe it’s not a good idea to use only this method to distinguish car and notcar object from each other. Because for detecting all the notcar objects in the environment a collection of e.g. 1000 image is not enough and for car object as well. I think this method can be one of the mechanism with them we can classify car and noncar.
* I process a whole image for all the scales that I have but I should segment the whole image and assign a scale to each segment and then process the frame of image which I consider in sliding. It helps me to have better performance.