# VEHICLE DETECTION PROJECT

## TABLE OF CONTENTS

## OBJECTIVE

This project requires the developer to develop a pipeline to identify vehicles from car-mounted camera feeds of highway driving.

## GOALS

Project goals were the following:

1. Extract features from a Histogram of Oriented Gradients (HOG) on a labeled training set of images; train a Linear SVM classifie
2. Optionally, you can also apply a color transform and append binned color features, as well as histograms of color, to your HOG feature vector.
3. Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.
4. Implement a sliding-window technique and use your trained classifier to search for vehicles in images.
5. 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.
6. Estimate a bounding box for vehicles detected.

Note: All of the code developed is in the file p52-5.py, and references from here onward will refer to it thus, [1].

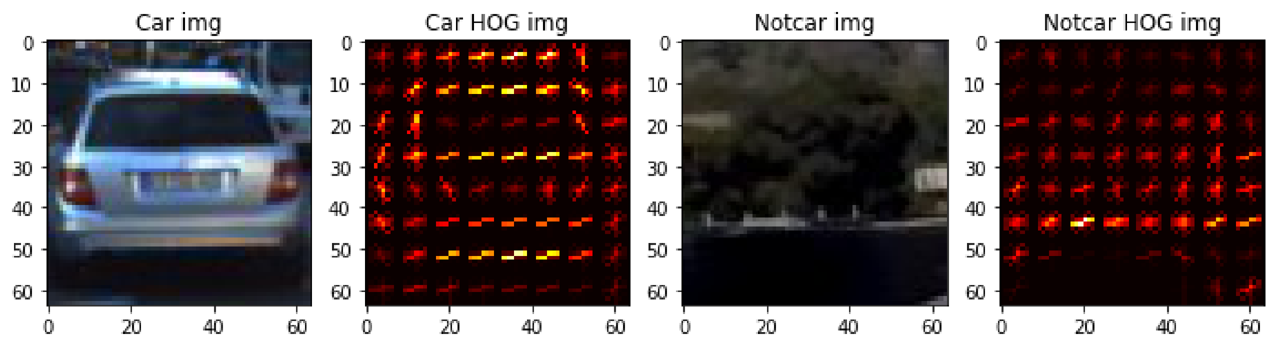
# 1. HISTOGRAM OF ORIENTED GRADIENTS (HOG): FEATURE EXTRACTION

HOG is useful as a feature descriptor that simplifies a digital image, small or large, by extracting useful information while discarding the rest. It is useful as input to classifiers such as a support vector machine (SVM). The process yields a one-dimensional array (called a feature vector) with values calculated thus:

1. Calculate the gradient along the X / Y directions to identify edges while discarding the remaining information. See Open CV's Sobel() function.
2. Calculate the magnitude and direction of the two gradients

```
abs_sobelx=√(sobelx)²
abs_sobely=√(sobely)²
abs_sobelxy=√(sobelx)²+(sobely)² – Magnitude
Angle θ = sobely / sobelx – Direction
```

This yields an image with a magnitude and a direction of that gradient, arranged approximately from 0-180°. Running these operations on the Udacity-provided image datasets (in our case, datasets of images of cars and of "not cars") extract the following information from such images. An example of each is shown next:



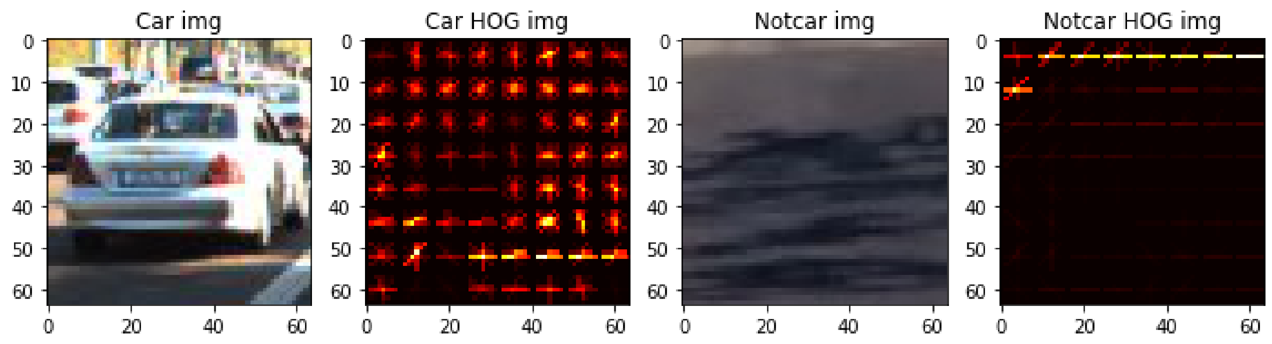
It is worth noting that there is a discernible outline to these features corresponding to the images. These images were extracted with code that has been removed from [1], shown below, however.

```
car_features, car_hog_image = single_img_features(car_image,
      color_space=color_space,spatial_size=spatial_size, hist_bins=hist_bins,
      orient=orient, pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
      hog_channel=hog_channel, spatial_feat=spatial_feat,hist_feat=hist_feat,
      hog_feat=hog_feat, vis=True)

notcar_features, notcar_hog_image = single_img_features(notcar_image,
      color_space=color_space,spatial_size=spatial_size, hist_bins=hist_bins,
      orient=orient, pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
      hog_channel=hog_channel, spatial_feat=spatial_feat,hist_feat=hist_feat,
      hog_feat=hog_feat, vis=True)
```

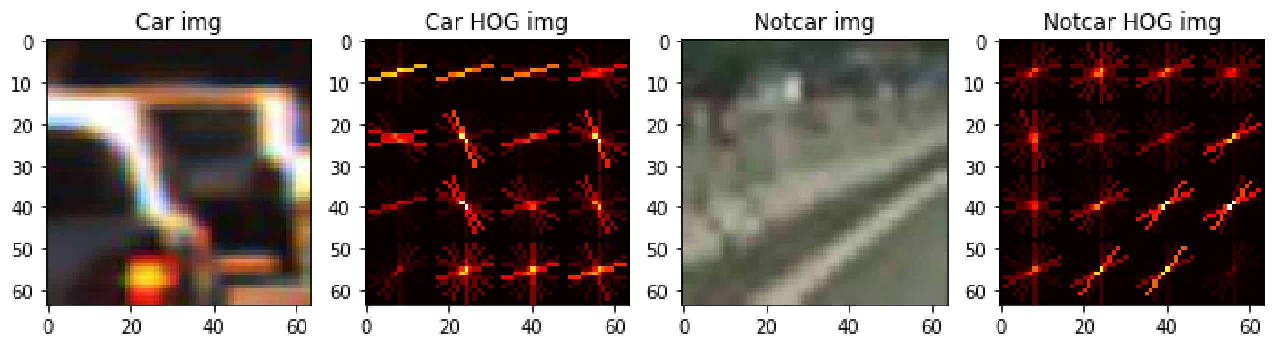
Various parameters (such as the ones shown below) were used in several combinations to get a better idea of the HOG results.

```
Code in file (around line 420 in [1])

#Set up parameters for the functions to be called
color_space = 'RGB' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 9  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = 0 # Can be 0, 1, 2, or "ALL"
spatial_size = (32, 32) # Spatial binning dimensions
hist_bins = 16    # Number of histogram bins
spatial_feat = True # Spatial features on or off
hist_feat = True # Histogram features on or off
hog_feat = True # HOG features on or off
```

For example, HSV color space (channel S, or 1) with 32 orientations produced the below HOG features.



Another example, LUV color space (channel L, or 0) with 16 pixels per cell and 9 orientations produced the below HOG features.



After many manual combinations of the parameters, the final set was fixed as follows: `{YCrCb color space, ALL channels, 9 orientations, 8 pixels per cell, 2 cells per block}`.

## A MORE PROPER APPROACH

A more “scientific” approach was adopted to ferret out the best set of parameters. Time constraints, however, ruled against finishing that line of inquiry.

```
def generate_feature sets():
        #Setup for best parameter set search to feed the SVM training
        color_space_range = ['RGB', 'HSV', 'LUV', 'HLS', 'YUV', 'YCrCb']  # 6 color spaces
        orient_space = list(range(6,10)) # range is 6,7,8,9
        pix_per_cell_range = list(range(6,9)) # range is 6,7,8
        hog_channel_range = ['0', '1', '2', 'ALL']
        hist_bin_range = [16, 32]

        #Establish the total # of combinations to sift through
        iter_count=len(color_space_range)*len(orient_space)*len(pix_per_cell_range)*len(hog_channel_range)*len(his
        t_bin_range)
        featureset_list = [dict() for x in range(iter_count)] #Define a list or an array of dictionaries to store
        the feature set run to gen car_features etc.

        import pickle
        import time

        iter_count = 0 #Reset counter to count again
        for color_space in color_space_range:
            t1 = time.time()
            print ('Generating featureset ', iter_count, 'at:', t1)
            for orient in orient_space:
                for pix_per_cell in pix_per_cell_range:
                    for hog_channel in hog_channel_range:
                        for hist_bins in hist_bin_range:
                            print ('Color_Space:', color_space,' Orientation:', orient,' Pix /cell:',
                                 pix_per_cell,' HOG Chnl #:', hog_channel, ' # of Hist Bins:', hist_bins)
                            featureset_list[iter_count] = [{'Color Space': color_space,'Orientation':
                                 orient,'Pix_per_cell': pix_per_cell,'HOG_Channel #': hog_channel,
                                 'Num_of_Hist_Bins': hist_bins}]
                            car_features = extract_features(cars, color_space=color_space,
                                 spatial_size=spatial_size, hist_bins=hist_bins,
                                 orient=orient, pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                                 hog_channel=hog_channel, spatial_feat=spatial_feat, hist_feat=hist_feat,
                                 hog_feat=hog_feat)

                            notcar_features = extract_features(notcars, color_space=color_space,
                                 spatial_size=spatial_size, hist_bins=hist_bins,
                                 orient=orient, pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                                 hog_channel=hog_channel, spatial_feat=spatial_feat,
                                 hist_feat=hist_feat, hog_feat=hog_feat)
                            fn = 'car_noncar_features'+str(iter_count)+'.p'

                            with open(fn, 'wb') as picklefileh:
                                pickle.dump((featureset_list[iter_count], car_features, notcar_features),
                                 picklefileh, protocol=pickle.HIGHEST_PROTOCOL)
                            print ('Finished in ', time.time()-t1, 'sec.')
                            iter_count+=1

        print (featureset_list[0], '\n',featureset_list[101] )
        print ('Total # of iterations for feature preparation: ', iter_count)
```

The above developed functional code (not part of [1]) generated 576 feature sets (~50 GB), awaiting a plan to continue this investigation in short order! (Datasets can be shared, if needed on Google Drive).

## 2. SUPPORT VECTOR MACHINE (SVM) CLASSIFIER TRAINING

A support vector machine learning model is a versatile one for linear and nonlinear classification of complex but small to medium sized datasets, used here to classify car images from noncar images.

This classifier too comes with its bag of parameters, whose best combination for a given dataset, fortunately, can be found by employing the seemingly magical function GridSearchCV () found in the sklearn library.

The parameters that can be tuned are:
Value of C, gamma and choice of kernel, whether RBF or Linear.

```
'''GRID SEARCH OPTION DONE ONCE  (This code is not part of [1])

parameters = {'kernel':('linear', 'rbf'), 'C':[0.1, 1, 10], 'gamma':[0.01, 0.1, 1]}
svr = svm.SVC()
t=time.time()
grid_search = GridSearchCV(svr, parameters)
t2 = time.time()
print(round(t2-t, 2), 'Seconds to GridSearch SVC...')
grid_search.fit(X_train, y_train)
t3 = time.time()
print(round(t3-t2, 2), 'Seconds to GridSearch SVC...')
print ('\nBEST PARAMS', grid_search.best_params_)
'''

BEST PARAMETERS: {Kernel: Linear, C=0.1, Gamma: 0.01}   (Took 557 secs.)
```

GridSearchCV (), so employed to find the best combination for the image datasets, yielded C=0.1 with a Linear kernel.

The features data was split 80/20% for training and testing and normalized and shuffled.

So, at this point, the next step became training the SVM classifier using the above parameters. The code is as shown below.

```
# Using a linear SVC  (Code starts at line 463 in [1])
svc = LinearSVC(C=0.1)

# Check the training time for the SVC
t=time.time()
svc.fit(X_train, y_train)
t2 = time.time()

print(round(t2-t, 2), 'Seconds to train SVC...')

# Check the score of the SVC
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))
```

Now, with the previously noted feature parameter combination (YcrCb, ALL channels etc.), the classifier reports a test accuracy of 99.77%. See screenshot below.

```
Number of car images:      5966
Number of non-car images:  8968
Using: 9 orientations 8 pixels per cell and 2 cells per
block
Feature vector length: 8412
74.0 Seconds to train SVC...
Test Accuracy of SVC =  0.9977
```

Due to some reported (unconfirmed by the student and beyond the scope of this project) aberration in "intercept_scaling" in LinearSVC(), the standard SVC() function was used to train the model (the training time, 74 sec, is a lot more than SVC()'s training time, shown below).

```
Number of car images:      5966
Number of non-car images:  8968
Using: 9 orientations 8 pixels per cell and 2 cells per
block
Feature vector length: 8412
22.31 Seconds to train SVC...
Test Accuracy of SVC =  0.9977
```

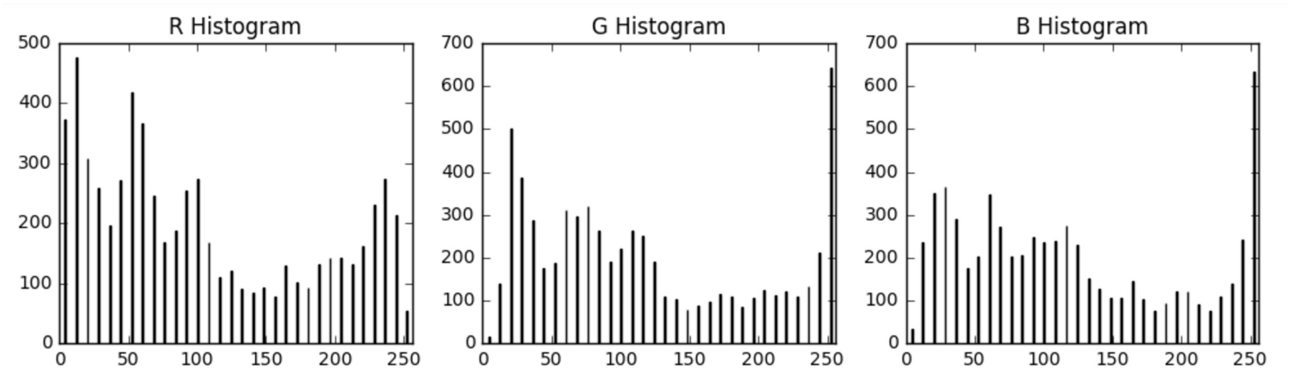
Due to nary a difference between the two in classifier accuracy, the latter one was chosen for this project due to its faster classification time.

With some other changes as noted below ( using SVC() ):

```
Number of car images:      5966
Number of non-car images:  8968
Using: 6 orientations 8 pixels per cell and 2 cells per
block
Feature vector length: 6696
11.03 Seconds to train SVC...
Test Accuracy of SVC =  0.9987
```

## 3. COLOR HISTOGRAMS AND SPATIAL BINNING AS FEATURE DESCRIPTORS

Histograms of colors (the constituent channels, for e.g., Red, Green and Blue in RGB) can also act as features. As they affect performance of the classifier, they have been included as part of the pipeline.



So too is the spatial binning feature, which is simply upsizing or downsizing an image from the image in the dataset, each of which is 64 x 64 pixels (RGB).  So, the images are downsized by a factor of ½ to 32 x 32 pixels.

So, both of these are included as features in this pipeline. Example images above are shown from the Udacity course material.

# 4. DETECTION (SLIDING WINDOW TECHNIQUE) & PREDICTION BY SVM CLASSIFIER

The technique is to divide the "to-be-searched" image into as finely or coarsely as the detection shows. The bigger image below is divided into say (64x64) chunks starting from a given point, which has been chosen to be (0,400) in pixels, about the vertical midpoint so that the extraneous information (skyline, trees in the foreground etc.) can be filtered out.

The above process yields a specific number of windows. Now, the task is to

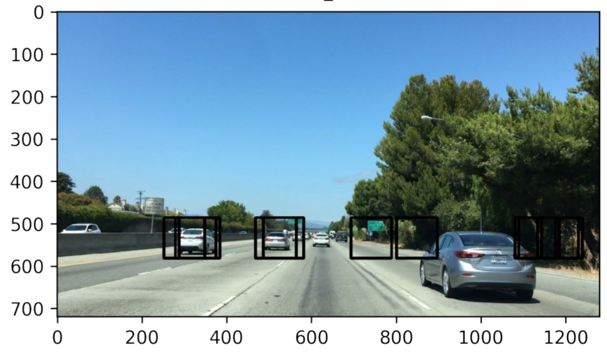
1. Scale this window down to the original dataset image size (in this case, 64x64 pix).
2. Extract the feature vector for this smaller (or sub)image (HOG, color histogram and spatial as selected), and have the trained SVM classifier prove its mettle on it (ie, predict).
3. If a car is detected, then note this window (add to a list of "hot' windows). The windows so detected are now considered to have car objects in them (or not, ergo, called a false positive).

This algorithm too brings its bag of parameters. So, after much tweaking, two parameter sets were chosen (based on the test images). Figures are shown later in this section.

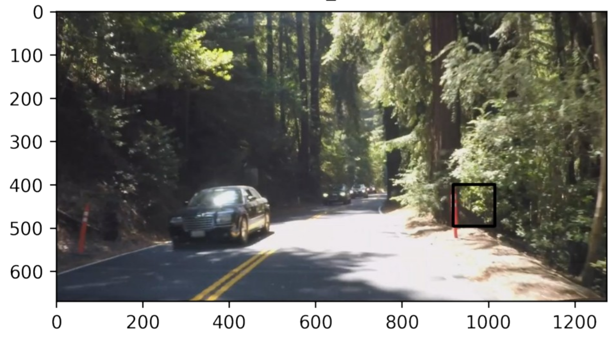
```
To find this code: please search for
"#SINGLE IMAGE WINDOW SEARCH AND CLASSIFIER" label in [1].
```

```
{ Subwindow size: 96 x 96 pix, (X/Y overlap) = (0.9/0.1) } setting yielded this:
34 hot windows
```

1

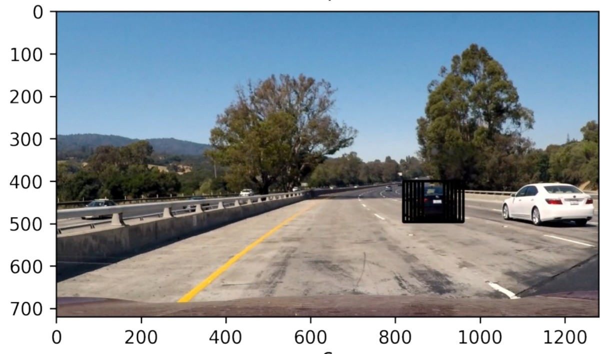


2

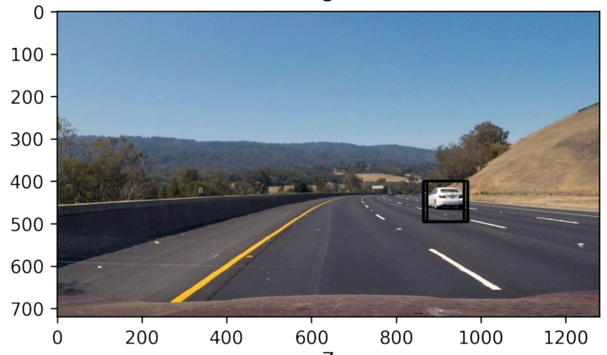


3

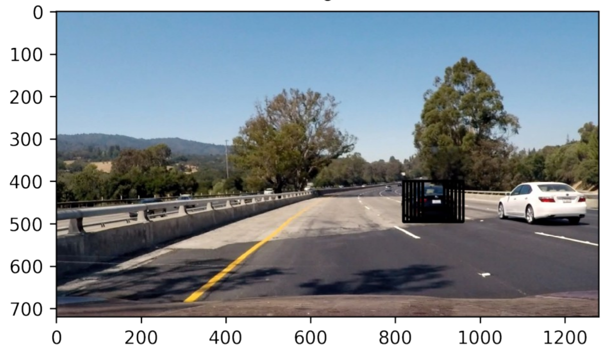
4



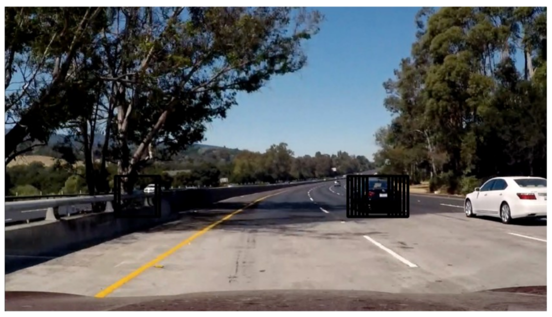
5



6

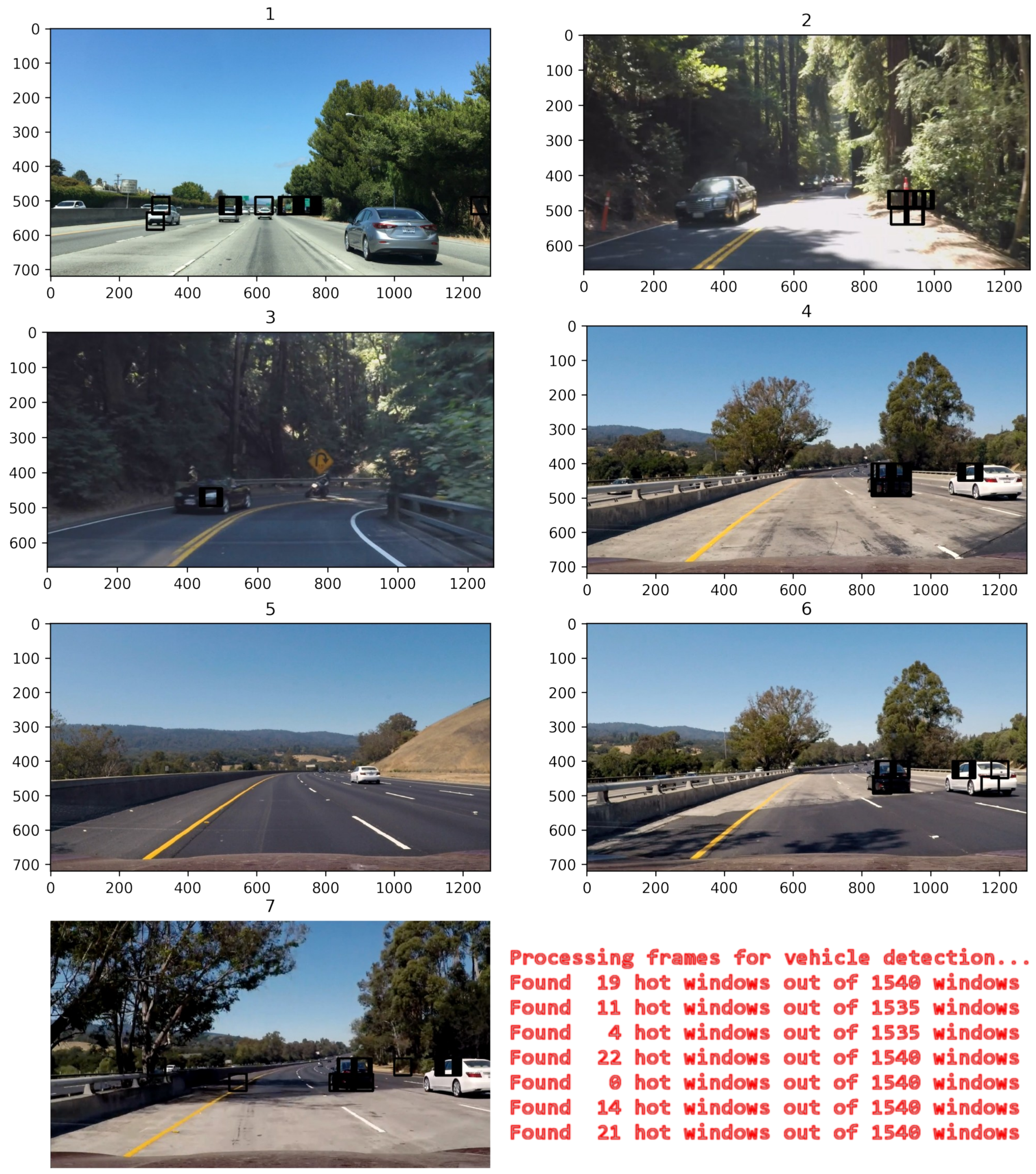


7



```
Processing frames for vehicle detection...
Found  11 hot windows out of 264 windows
Found  1 hot windows out of 264 windows
Found  0 hot windows out of 264 windows
Found  6 hot windows out of 264 windows
Found  2 hot windows out of 264 windows
Found  6 hot windows out of 264 windows
Found  8 hot windows out of 264 windows
```

```
{ Subwindow size: 50 x 50 pix, (X/Y overlap) = (0.9/0.1) } setting yielded this:
91 hot windows
```



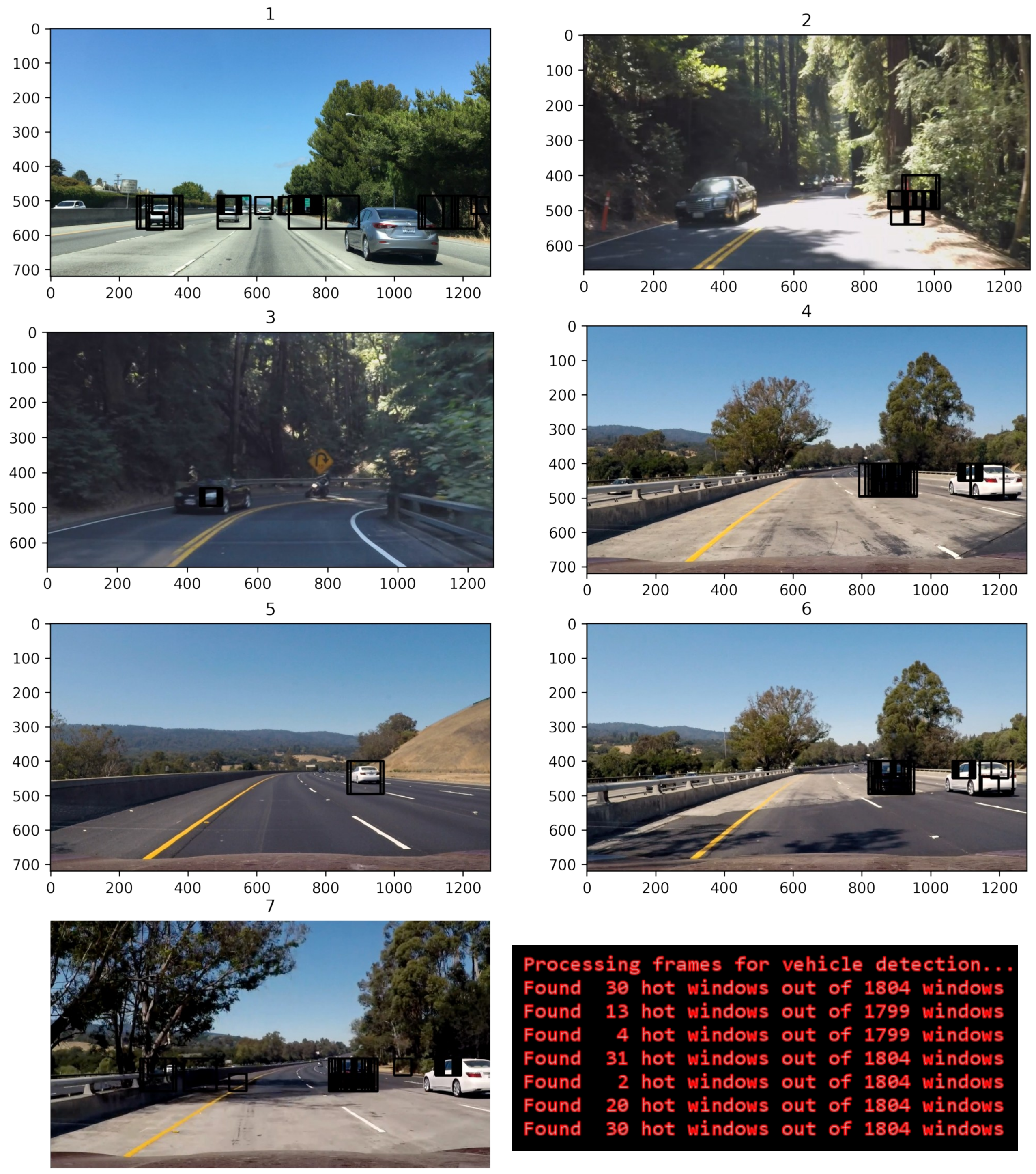
```
Processing frames for vehicle detection...
Found  19 hot windows out of 1540 windows
Found  11 hot windows out of 1535 windows
Found   4 hot windows out of 1535 windows
Found  22 hot windows out of 1540 windows
Found   0 hot windows out of 1540 windows
Found  14 hot windows out of 1540 windows
Found  21 hot windows out of 1540 windows
```

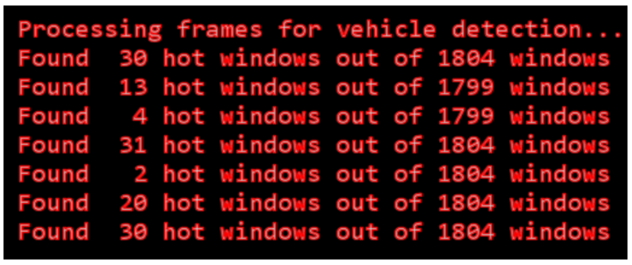
Note that this subimage size (50x50 pix) runs counterintuitive to the recommended integer multiple of the original (trained) image size (64 x 64), and produced more windows than the earlier setting including detecting a car in the image from P4's challenge video (see image 3).

Automated parameter optimization is recommended here as the next investigation step.

As there was no clear outcome, both settings were run independently and the window lists concatenated thus —

```
windows  = windows1 + windows 2; which produced the following:
```

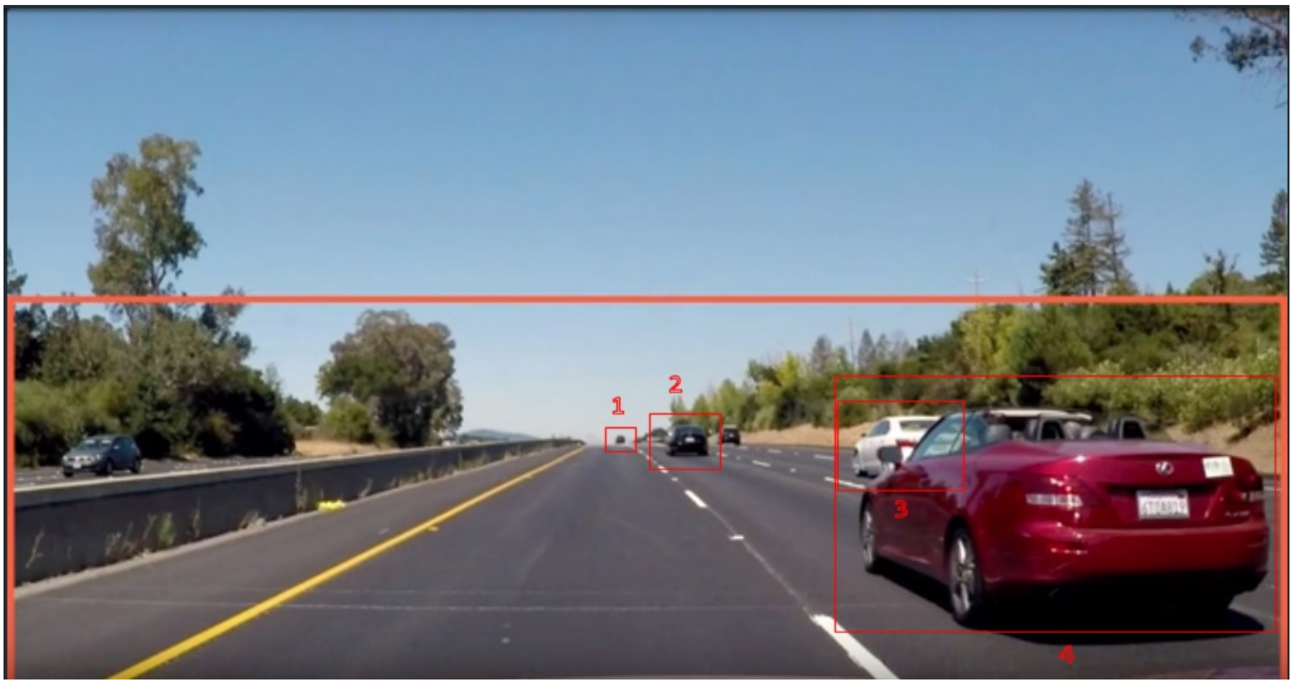




```
Processing frames for vehicle detection...
Found  30 hot windows out of 1804 windows
Found  13 hot windows out of 1799 windows
Found   4 hot windows out of 1799 windows
Found  31 hot windows out of 1804 windows
Found   2 hot windows out of 1804 windows
Found  20 hot windows out of 1804 windows
Found  30 hot windows out of 1804 windows
```

This produced 130 hot windows in total including several false positives; but the outcome is acceptable from looking at the quality of detections (and the low number of FPs).

Using a scale to size the search windows up and down (required) to be able to detect cars far away and the ones close to the camera can be a useful task.

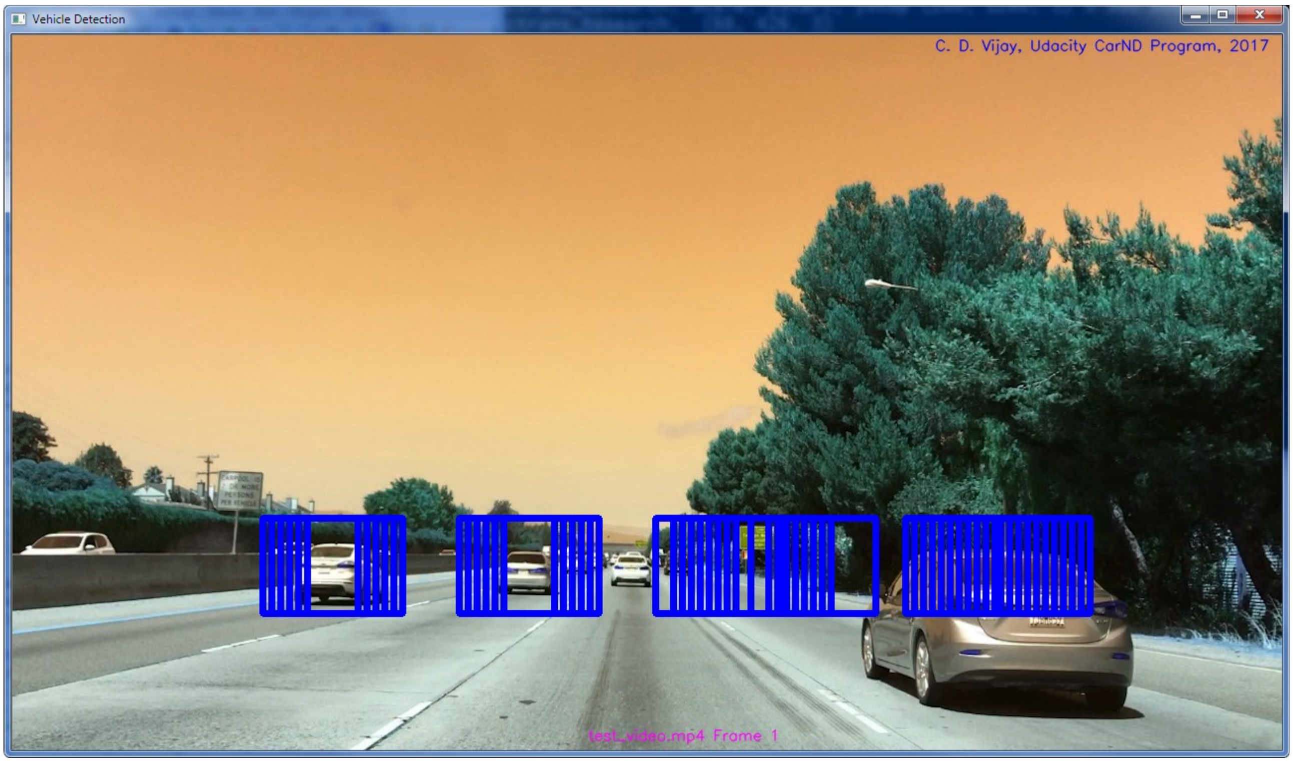


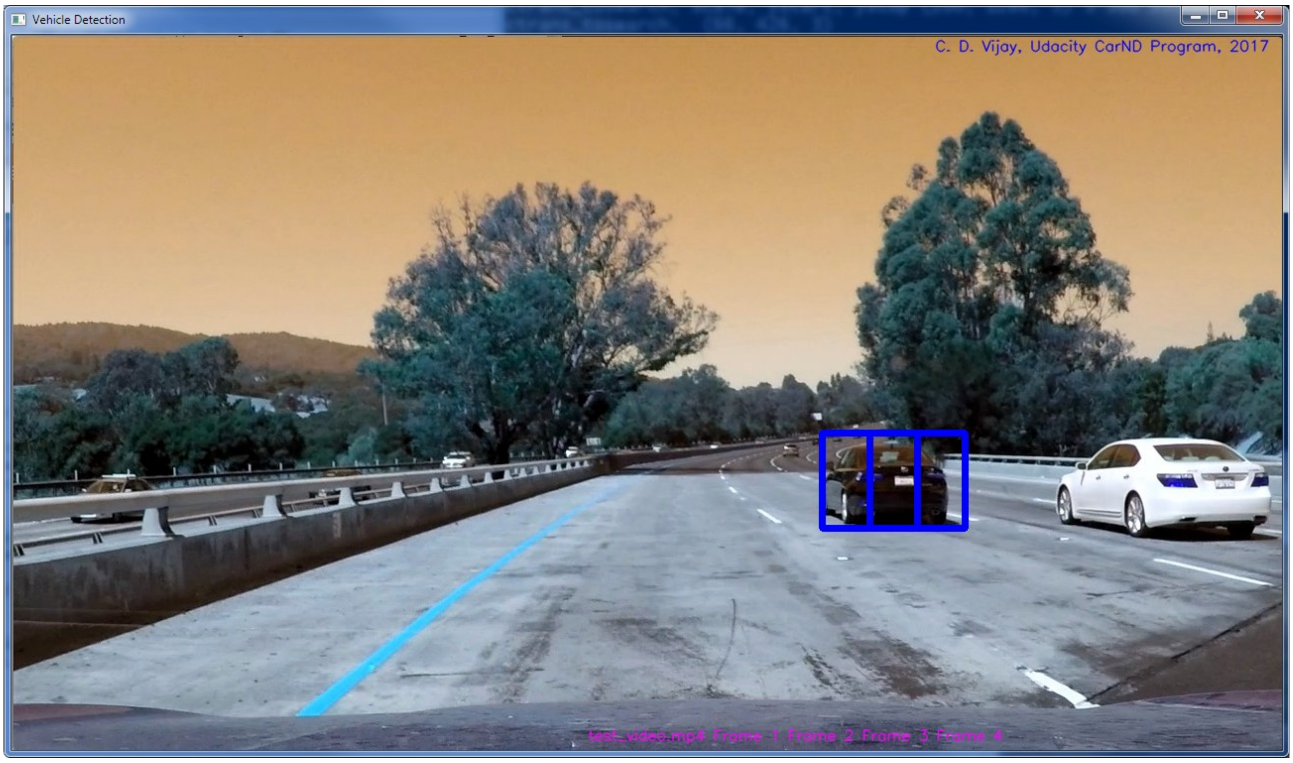
The plan was to search using four scales, shown as numbered windows above. Each window, as can be seen, fits within a horizontal band, which can be programmatically included in the code (time constraints did not allow that to happen).

But, an appoximate algorithm for doing so would be:

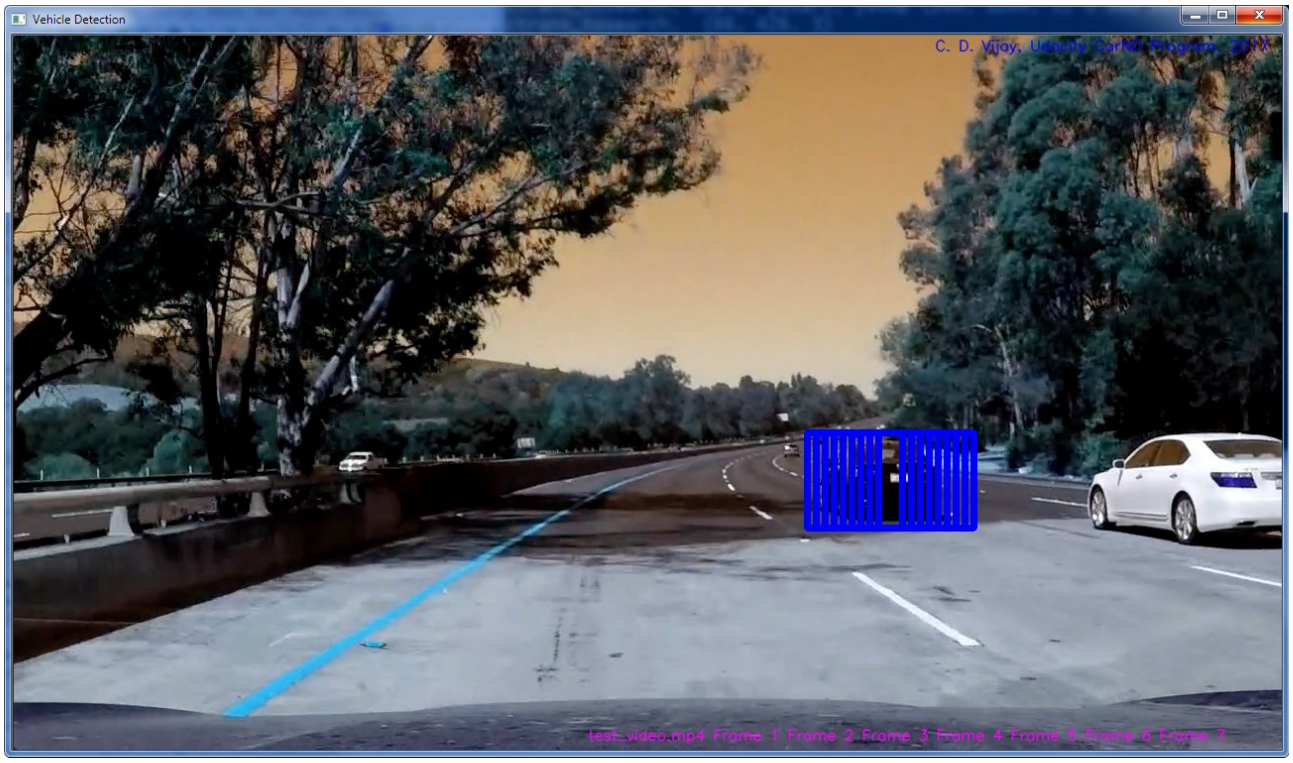
```
# Horiz bands for the 4 windows by picking the start and stop points in the y-
axis

ystart_stop = ([[y1, y2], [y3, y4], [y5, y6], [y7, y8])

for ystart, ystop in ystart_stop:
      for scale in linspace (1, scale_max, 0.25):
            find cars (img, scale)
            #Detect false positives
            #Draw video / display image
```

Unfortunately, some of my (hasty) code did not work as expected.  However, a few of the frames had detections.



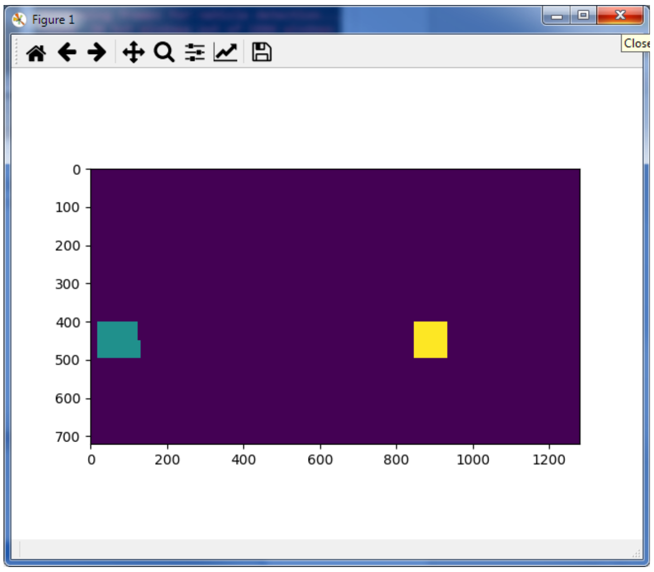

Vehicle Detection
C. D. Vijay, Udacity CarND Program, 2017
test_video.mp4 Frame 1 Frame 2 Frame 3 Frame 4

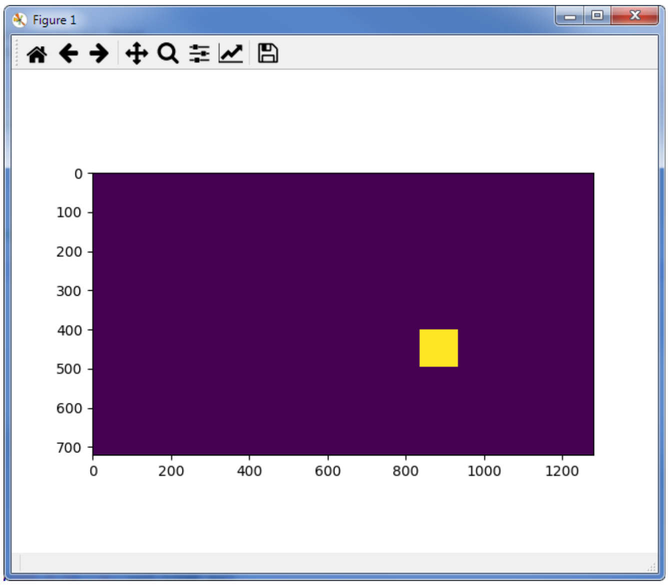
So, a similar scaling must be applied with more tweaking to detect cars in frames/video.


Vehicle Detection
C. D. Vijay, Udacity CarND Program, 2017
test_video.mp4 Frame 1 Frame 2 Frame 3 Frame 4 Frame 5 Frame 6 Frame 7

# 6. ELIMINATING FALSE POSITIVES

One technique to eliminate false positives is by making a heatmap – a method in which each "hot" window is marked (given added heat) in every frame. Over the course of several windows, the cars if truly present in a frame will get multiple detections thus making that zone "hot".





Code is around line 579 of [1]. After this, using the very useful label() function from scipy.ndimage.measurements, multiple contours are labeled, while weak ones are removed.

And then, a composite with the original image is made by collecting all of the redrawn frames. Code is at line 579. Link to [Youtube (project_video.mp4)](#)

## 8. DISCUSSION

Several things are noteworthy here.

1. This algorithm is more proof-of-concept than an industry-grade one.  Of course, any code can be optimized but the concept does not lend itself to real-time operation, unless more powerful hardware becomes affordable.

2. With a lot more data (maybe SVM will not be the classifier of choice then), detections will become easier.

3. Due to the programmatic nature of the code, again as in P4, I cannot help but be intrigued with the possibility of using deep learning for this task as several latest firms are demonstrating.

## REFERENCES

[1]. Source code file, p52-5.py, submitted with this project

[2]. Udacity CarND Term1, Course material