Project 5: Vehicle Detection

General Project Pipeline¶

1. Feature extraction and experimentation
2. Choose and train a classifier
3. Implement sliding window technique to search for vehicles on each frame
4. Implement pipeline on video and implement tracking

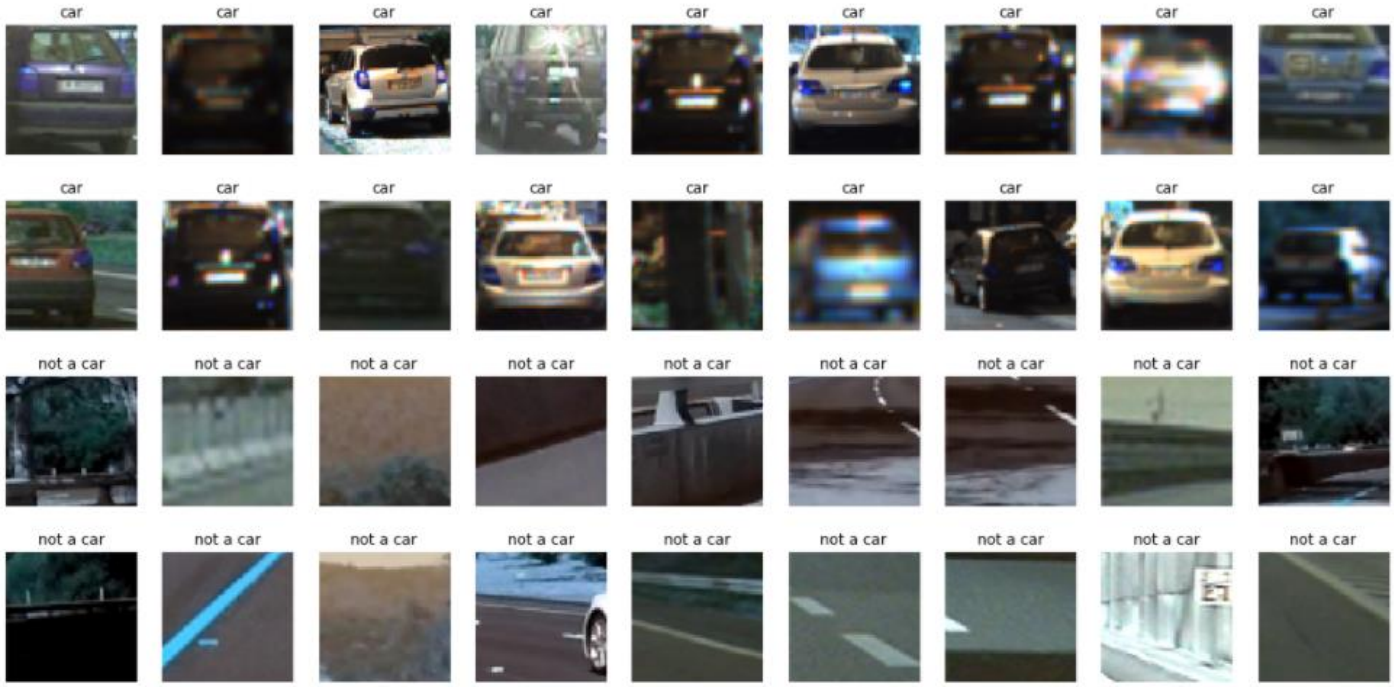
Feature Extraction

Feature Extraction Steps:

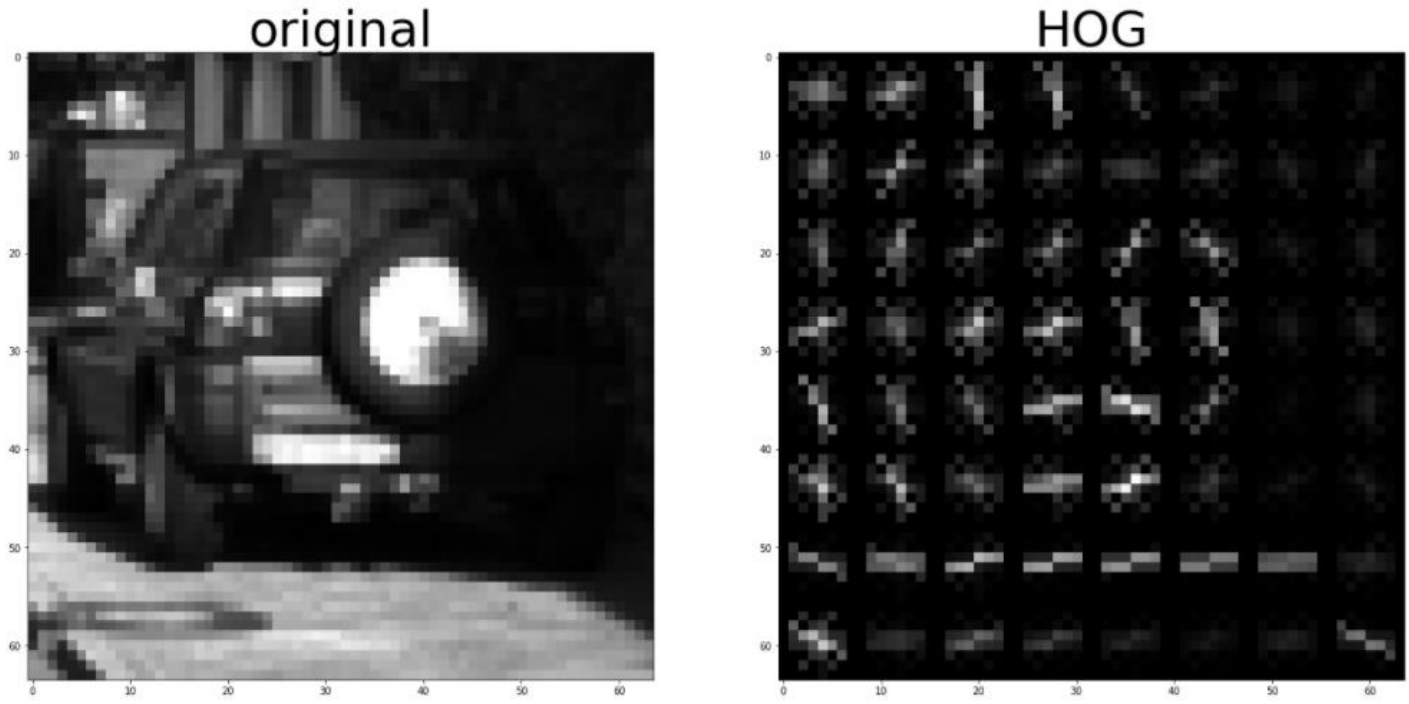
For this project, HOG is used as primary features to be extracted.

1. Read in data
2. Use alogrithm provided in lesson to extract HOG
3. Store data into array

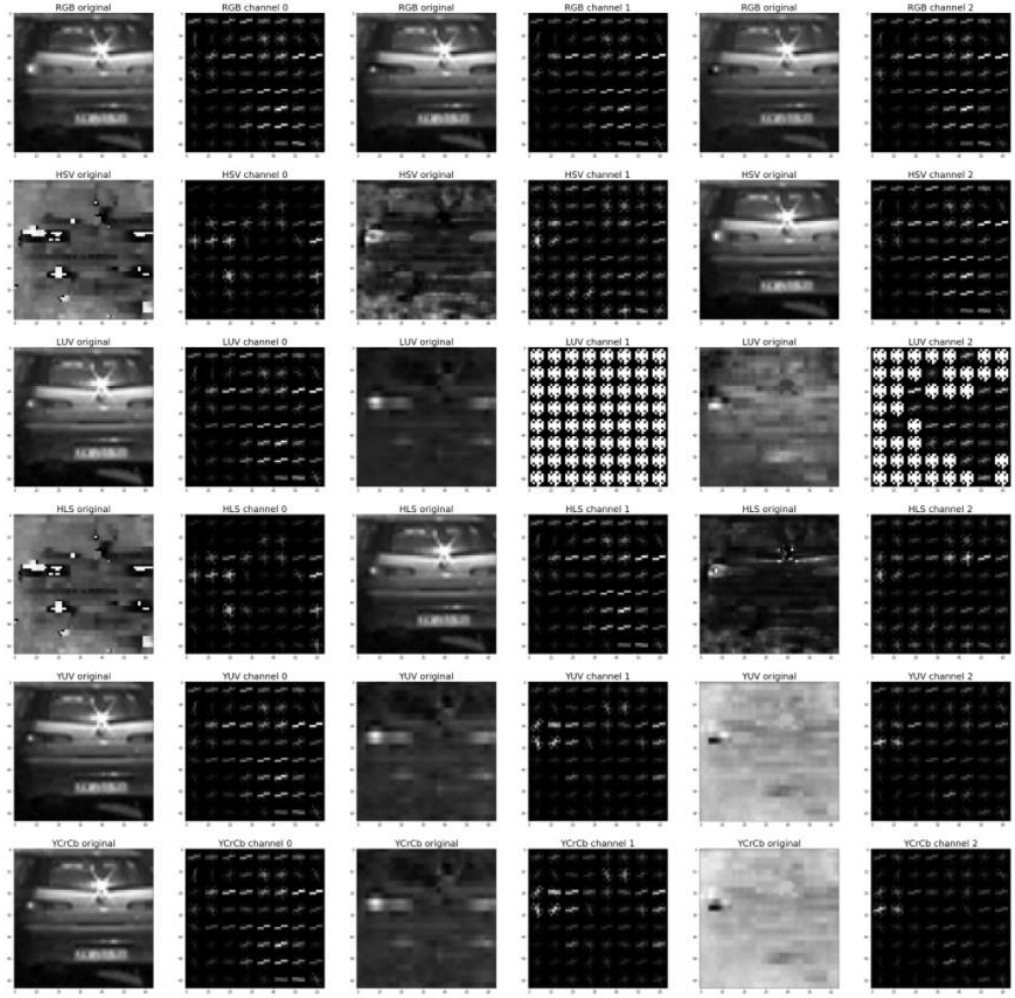
Visualizing data:



Features extracted here are HOG. Here is a visualization of it:



We probably do not need all the channels to make a distinctive set of data. Color spaces and their corresponding HOG was also explored:



It can be seen that there are a couple of options that can be ruled out immediately. These are the options that do not produce anything that resembles what is in the original image. These are:

- LUV channel 1
- LUV channel 2

Next, we eliminate those which do not produce a HOG that captures enough details or those which do not capture the correct direction of gradient that resembles the shape of the car:

- RGB channel 0
- HSV channel 0
- HSV channel 1
- HLS channel 0
- HLS channel 2
- YUV channel 1
- YUV channel 2
- YCrCb channel 1
- YCrCb channel 2

This leaves us with:

- RBG 1 and 2
- HSV 2
- LUV 0
- HLS 1
- YUV 0
- YCrCb 0

Choose and Train a Classifier

Steps to choosing and training a classifier:

1. Create labels
2. Preprocess data
3. Determine a combination of hyperparameters and optimize
4. Train

Parameter Tuning using Scikit-learn Cross Validation. The classifier chosen is SVM. There are three main parameters to tune for SVM:

1. Gamma: defines how far the influence of a single training example reaches (low value - far reaching, high value - close)
2. C: defines the tradeoff between smoothness of the decision boundary and the number of sample
3. Kernel: increase of input space to achieve a more linear separable data

In [161]:
```
parameters = {'kernel':('linear','rbf'), 'C':[1,5,10]}
svr = svm.SVC()
clf = GridSearchCV(svr,parameters)
clf.fit(scaled_X_train, y_train)
```

Out[161]:
```
GridSearchCV(cv=None, error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [1, 5, 10], 'kernel': ('linear', 'rbf')},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
# verifying classifier accuracy
y_pred_test = clf.predict(scaled_X_test)
accuracy = accuracy_score(y_test, y_pred_test)
print('the classifier accuracy is: ', accuracy)
```

```
the classifier accuracy is:  0.9479166666666666
```

95% accuracy may not be enough for this exercise (as learned previously from traffic sign identification project). As an attempt to improve this. The following approach is taken:

- Use a different color space for data prep
- Use all channel to increase data space
- Alter/increase the number of gradient directions in the HOG

```
# testing for accuracy
y_pred_test = clf2.predict(scaled_X_test)
accuracy = accuracy_score(y_test_2, y_pred_test)
print('the classifier accuracy is: ', accuracy)
```

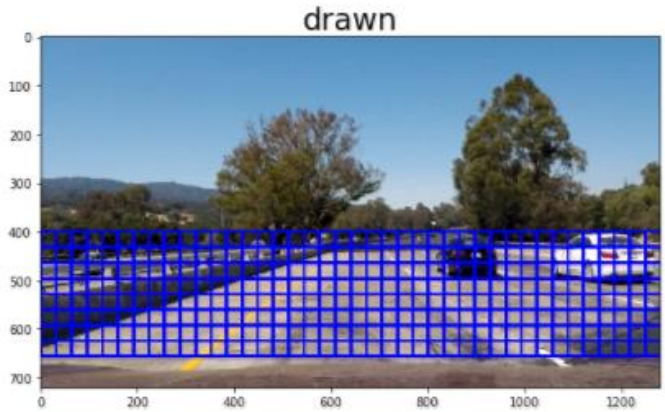
```
the classifier accuracy is:  0.982545045045045
```

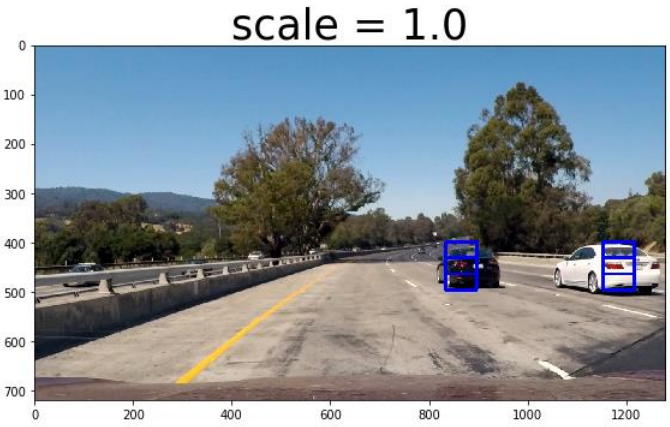
As shown above, the secondary classifier has a much higher accuracy. Therefore, the secondary classifier should be used instead.

Implement a Sliding Window Technique

Steps to implementing the sliding window technique

1. Define start stop position for x and y for area of search
2. Define the sizes with which the windows will be created
3. Define overlap percentage for sliding window
4. Combine the detection with sliding window
5. Heat map to filter out false positive

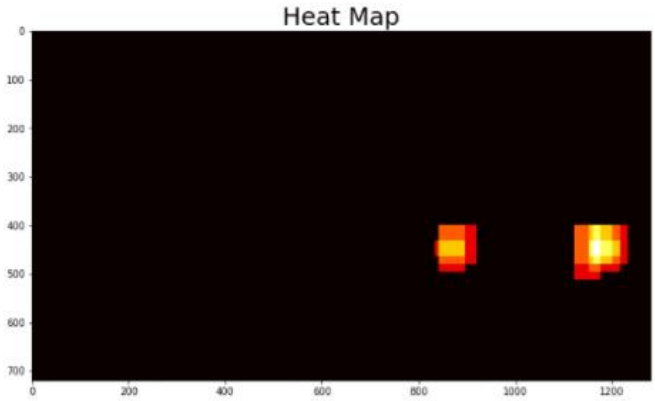




Add in a heat map to filter out false positives

steps to adding heat map filter:

1. Take a zero data image of identical size to the video frame, add 1 for each time the region is confirmed positive
2. Apply a threshold to filter everything that is below the threshold
3. Use label function to determine the number of cars found
4. Redraw boxes on region that are above the threshold



Discussion

Because feature vectors are entirely made of one color space, the pipeline would likely fail under lighting conditions when such color space in no longer capable of capturing distinctive features.

At a glance, the vehicles training samples are all the same angles. Though practically sufficient, should a vehicle appear at a different angle (head on, for example), the pipeline may not be able to identify it since the classifier has not been exposed to this kind of training sample.

Another problem is the amount of false positives. Although the secondary classifier achieved a test accuracy of 98 percent, it still regularly detects car on the left yellow line.

To address the first problem, redundancy can be added into the feature vectors by hstacking it. The second problem can be resolved via introducing a more diverse sample variety.

Finally, the third problem can be improved by either adding more defining features into the feature vector, or having a more robust heat map filter algorithm (one that takes into account the neighbouring heat intensity).

Video links:

- First video: This was done with the first classifier (with a test accuracy of 94%)
- Second video: This was done with the secondary classifier (with a test accuracy of 98%)
- Third video: This was also done with the secondary classifier, however, the heat map filter was set to 2 (instead of 1)