Project: Vehicle Detection and Tracking

Patrick Cleeve

Objective

The objective of this is to develop a vehicle detection and tracking pipeline for autonomous driving using computer vision techniques, using only camera images. The pipeline draws bounding boxes onto the image taken from a camera mounted on the front of the vehicle.

The project code is available here:
https://github.com/patrickcleeve/CarND-Term1-Vehicle-Detection-P5

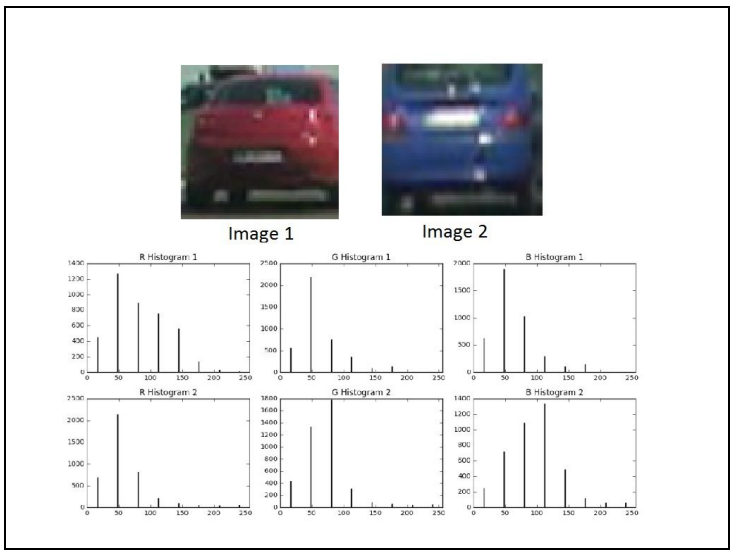
Image Features

We can use a number of different features of an image to detect a vehicles within the image. These features differentiate the vehicles from the rest of the image. We use a combination of features to robustly detect vehicles in the images.

Colour Features

One of the distinct features of vehicles compared to the road and surroundings is colour. We could try to use a correlation or template matching technique to identify vehicles. However, these methods only work well when we can identify close matches, and given the number of different cars (colours), distances (size) and orientations we observe cars on the road, they are not a reliable method.

We therefore need a method that is robust to changes in object appearance. We use the histogram of colour values. We can compare regions of the images with a test image, and locations with a similar colour distribution will show a close match. This removes the dependence on image structure, meaning we can detect vehicles in different orientations, and sizes (by normalising the data). This method relies solely on matching colour values, which may match some unwanted regions.



Colour Histograms (Udacity Lecture)

However, matching individual vehicle colour histogram does not completely solve the problem. We would need to match histograms for each possible car colour on the road (e.g. red, blue, white, etc), which would be a time consuming task, and introduce a significant number of false positives. We can transform the image to a different colour space, in which the colours of vehicles are easily separable from road or environment colours (e.g. HSV, or YCrCB).

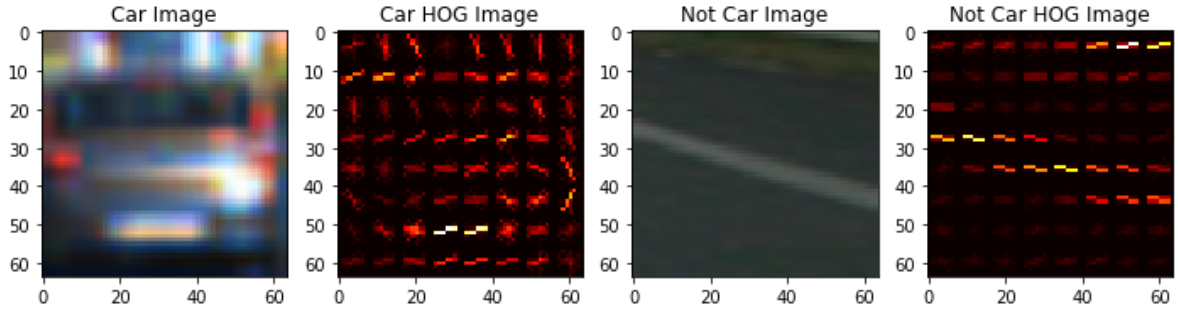
We can retain colour information, but reduce the computational cost by resizing (spatial binning) our images to a lower resolution. As shown below, the car and colour is still easily identifiable at a lower spatial resolution.

Spatial Binning (Udacity Lecture)

Histogram of Oriented Gradients (HOG)

Whilst colour information is useful in identifying vehicles, the number of colour variations can reduce the accuracy of our predictions. We therefore require a more robust, colour invariant method of identifying vehicles in images.

We can use the gradients of the image to identify the ‘shape’ of the car in an image. Instead of using the individual gradients for each pixel, we group sections of the image into cells, and calculate a histogram of gradients (orientations) within each cell. This provides us with a direction and magnitude of the gradient within each cell. We can then use the calculate HOG image as a signature for detecting vehicles in images. We can adjust each of these parameters to improve the reliability of our HOG method in vehicle detection.



Car and Not Car HOG Images

HOG Parameters and Feature Extraction

We are not limited to using only one set of image features to detect vehicles. We can combine multiple sets of features to produce a more robust classification. For this project we evaluate spatial, colour and HOG features. We experimented on the following parameters to determine the optimal combination for vehicle detection.

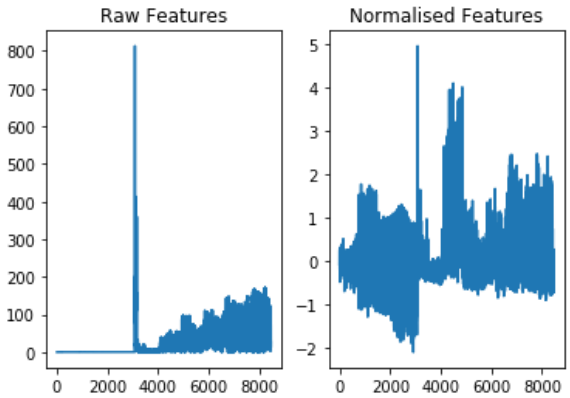
- Colour Space (color_space): Colour space to be used
- Orientations (orient): Number of gradient orientations (HOG)
- Pixels Per Cell (pix_per_cell): Number of pixels per cell (HOG)
- Cells Per Block (cell_per_block): Number of cells per block (HOG)
- HOG Channel (hog_channel): Number of HOG channels (HOG)
- Spatial Size (spatial_size): Image resolution of spatial features (Spatial)
- Histogram Bins (hist_bins): Number of histogram bins
- Spatial Features (spatial_ feat): Spatial features on/off
- Histogram Features (hist_feat): Histogram features on/off
- HOG Features (hog_feat) HOG features on/off

Classifier Training

To evaluate HOG parameters and vehicle prediction accuracy, a set of vehicle and non-vehicles images were loaded. These images, and extracted features were used to train a Linear Support Vector Machine (SVM) to detect vehicles in standard 64x64 images.

Initial experimentation was conducted on a randomised 1000 sample set of images from each category. For each category, features (spatial, colour, and/or HOG) were extracted and concatenated together into a single feature vector. After feature extraction, the data set was split into training and test sets. No validation set was used as the actual model performance is tested on the video pipeline.

The combined feature vector was normalised using the StandardScaler() function, to prevent individual features from dominating. The effect of this normalisation is shown below.

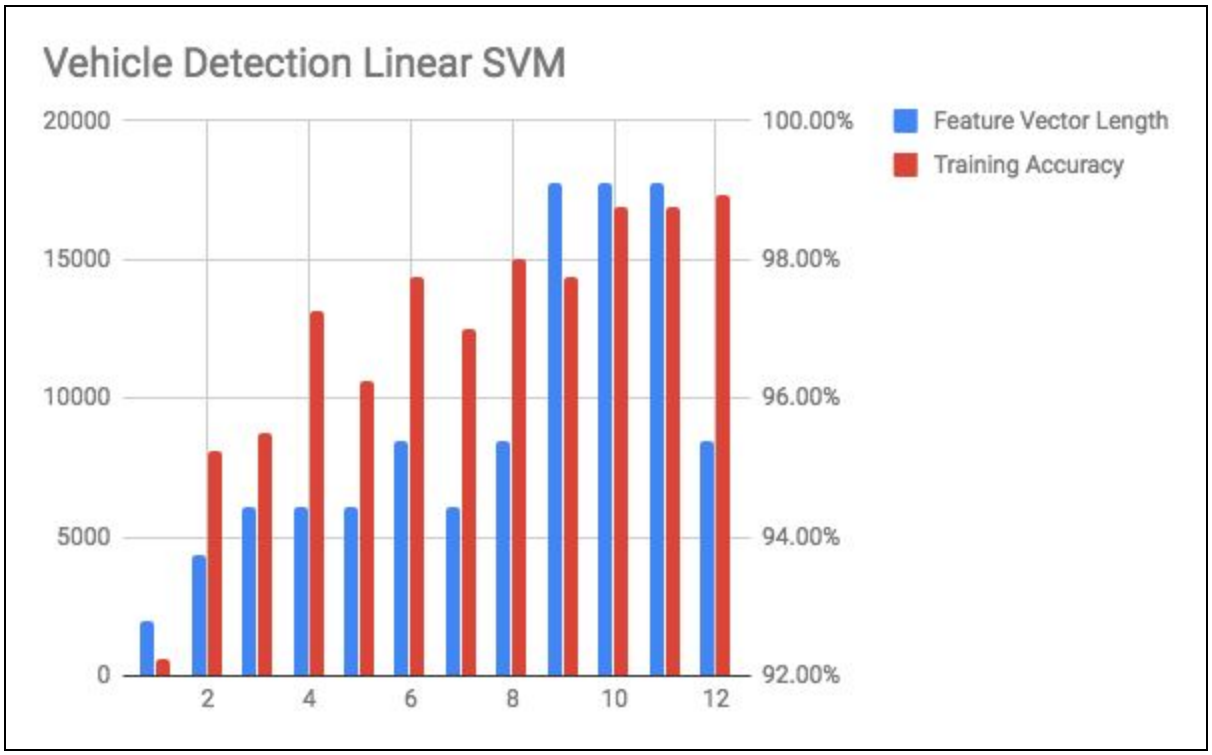


The initial model and feature parameters were as follows:

- Colour Space (color_space): RGB
- Orientations (orient): 6
- Pixels Per Cell (pix_per_cell): 8
- Cells Per Block (cell_per_block): 2
- HOG Channel (hog_channel): 0
- Spatial Size (spatial_size): (16, 16)
- Histogram Bins (hist_bins): 16
- Spatial Features (spatial_ feat): True
- Histogram Features (hist_feat): True
- HOG Features (hog_feat) True
- Linear SVM

The table below shows the experimental process followed:

Iteration	**Change**	**Feature Vector Length**	**Training Accuracy**
1	Initial	1992	92.25%
2	hog_channel = “ALL”	4344	95.25%
3	orient = 9	6108	95.50%
4	color_space=YCrCb	6108	97.25%
5	color_space=HSV	6108	96.25%
6	color_space =YCrCb spatial_size =(32, 32) hist_bins=32	8460	97.75%
7	LinearSVC(C=10) spatial_size=(16,16) hist_bins=16	6108	97.00%
8	spatial_size=(32,32) hist_bins=32	8460	98.00%
9	spatial_size=(64,64) hist_bins=64	17772	97.75%
10	C=100	17772	98.75%
11	C=1000	17772	98.75%
12	Final Model (All Examples) LinearSVM(C=1000) Color_space=YCrCb, orient=9 pix_per_cell=8, cell_per_block=2 spatial_size =(32, 32), hist_bins=32 spatial_feat=True, hist_feat=True, hog_feat=True	8460	98.93%



As shown, increasing the amount and detail of extracted features resulted in small improvements over the base model. Changing the colour space to one where vehicles colour could be easily separated (HSV, or YCrCb) provided the most significant boost in accuracy.

However, as we add extra features and detail, the feature vector increases significantly for relatively little improvement in prediction. Given that the data set is relatively small, meaning the classifier will likely overfit and produce a significant number of false positives, this extra computational cost will have a significant impact on performance with little effective gain. Practically, a more efficient use of time is spent on error detection rather than trying to improve the model prediction (and incurring the increased computational cost).

Types of Error

When training and evaluating the model, we encounter two different types of errors:

- False positive: detect vehicle where there is none
- False negative: don't detect vehicle where there is one

The model we have trained is overfitted on the training data, resulting in a large number of false positives. In order to properly evaluate the model we must assess whether it is appropriate for the model to be biased in such as way, and what are the consequences for doing so. Ideally, we

would like to reduce both types of error, but this can be a significantly challenging and time consuming task.

For each type of error we can evaluate the consequences and assess the severity, in order to determine which we should try to avoid. We can also evaluate methods to minimise or resolve the effect of these errors on our pipeline.

- False Positive: When a vehicle is detected, but does not exist, the car will have to brake excessively or potentially emergency braking at ‘phantom’ vehicles, potentially causing rear end accidents for vehicles behind. We can reduce these types of error by thresholding predictions over multiple image frames in a video stream to ensure that random noise and singular predictions are filtered out.
- False Negative: When no vehicle is detected, but does exist, the car will likely be unable to avoid other vehicles on the road, which can result in dangerous collisions (especially at high speed). We don't’ have an effective method of improving on false negatives, as we cannot ‘predict’ were these errors occur as robustly as false positives.

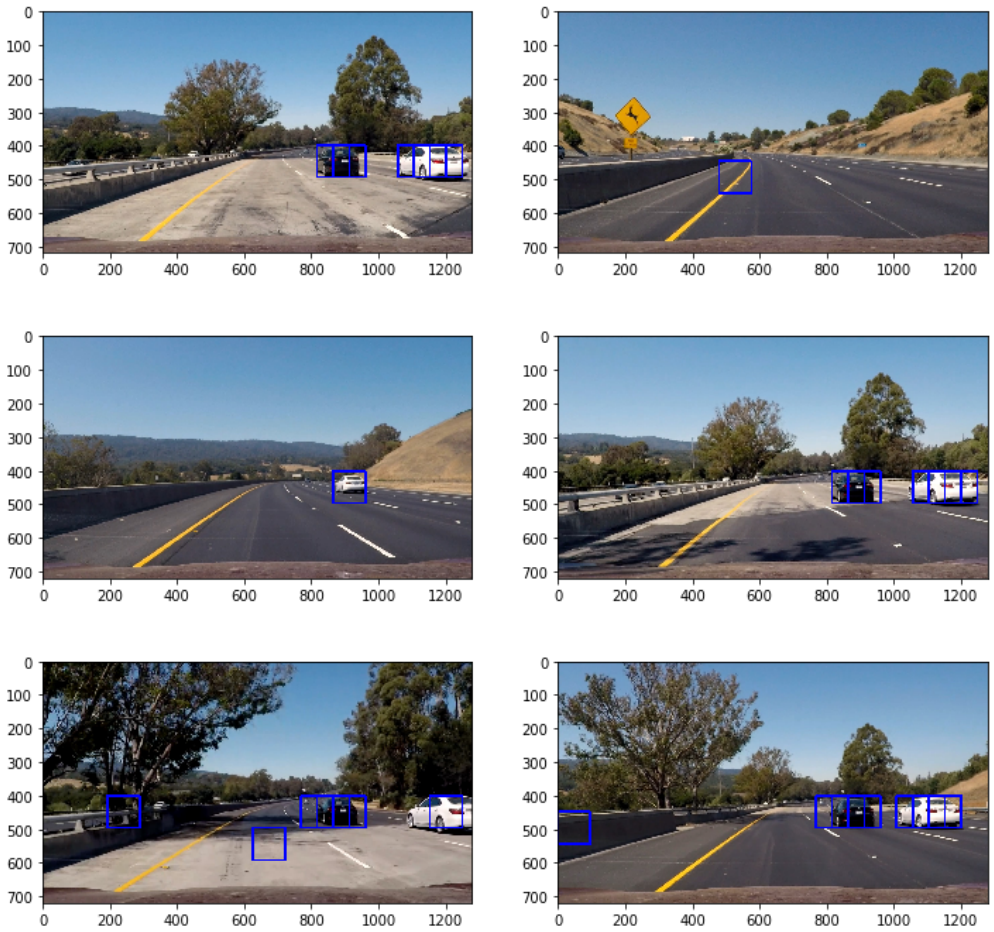
The consequences for false negatives (high speed collision) are more severe than false positives (rear end collision). In addition, we have more effective methods of reducing false positive types of errors, than false negatives. It is therefore more appropriate that we bias our predictions towards false positives.

Vehicle Detection

Sliding Windows Search

We now have the ability to classify whether images contain vehicles. However, we cannot simply run the classifier on an entire road image, as we are interested in where in the image the vehicle occurs. The classifier can only predict whether an image contains a vehicle.

Therefore, we can break the whole image into smaller windows, and run the classifier on each individual window. We can then return the window locations (bounding boxes) that return vehicle classifications. This is the basis of the sliding window search, we split the image into windows and search each individually. We can adjust the size of the window, and the amount of overlap with the next window to search. Higher levels of overlap will produce more robust search, but can have a significant computational cost, whilst different window sizes will be useful for classifier cars at different distances (sizes).



Sliding Window Search (96x96 window, 0.5 overlap)

In addition, we do not need to search the entire image, as vehicles will only be location on the road (until we invent flying cars). Therefore, we can restrict the search space for the algorithm to improve performance.

HOG Subsampling

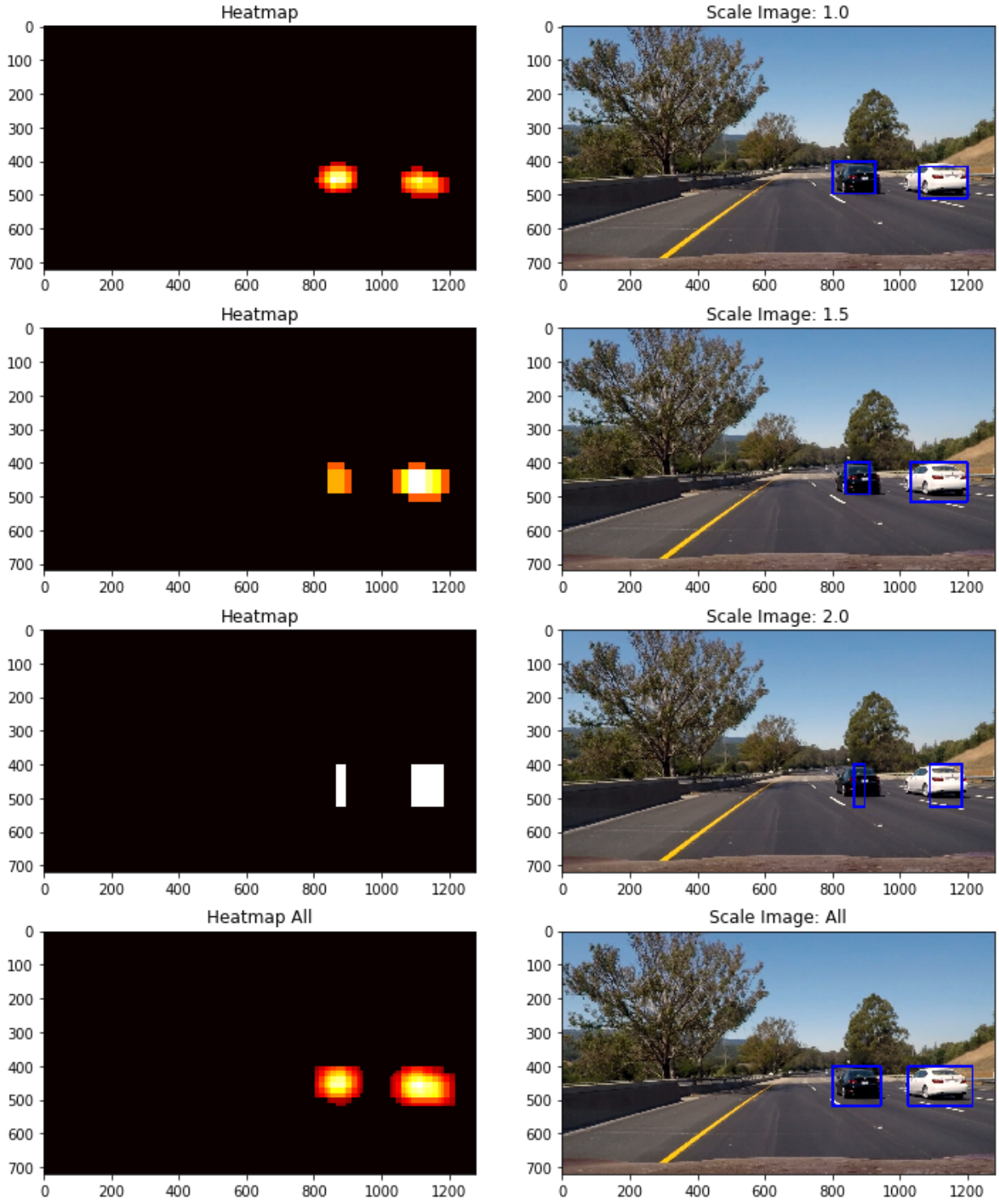
The sliding window search extracts HOG features for each window, instead, we can extract them for the entire image, then subsample the HOG feature array for each window. This produces a slightly different HOG feature vector, due to not having edges of each window delineated, but is significantly faster as feature extraction is one of the more computationally expensive steps in pipeline.

This subsampling is demonstrated inside the find_cars function. After subsampling, each feature vector (spatial, colour, HOG) is concatenated as previously.

Scale

Instead of changing the window size, we can resize the image to detect vehicles of different sizes (distances). Whilst having a similar effect as changing window size, we can combine multiple scale detections into a single image to be able to predict vehicles of different sizes within one image. The image below shows the detection of the classifier running at three different scales (1.0, 1.5, 2.0), and using these combined predictors. As shown, the combined prediction is much more robust than any individual predictor.

Increasing the number of scales used can provide a more stable prediction at different distances. However, this can have a significant computational cost, as for each different scale the algorithm must be re-run. We could improve performance by restricting the search window for different scales. For example, smaller vehicles (further away) only appear closer towards the horizon, (the centreline of the image), compared to larger vehicles (closer). These multi-sized windows for different scales are not used for this project.



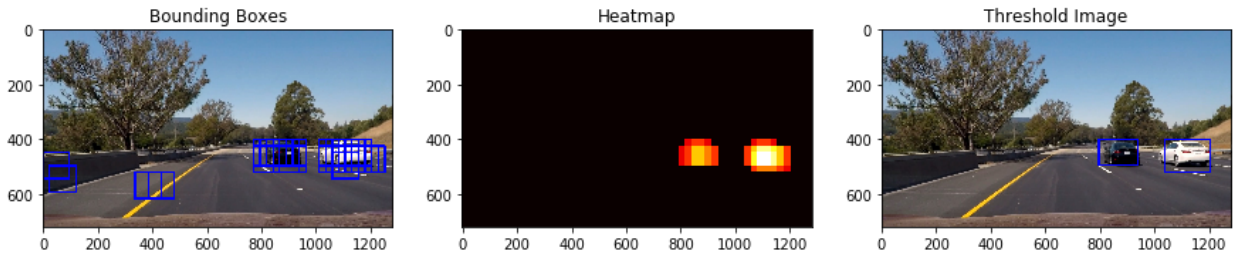
Multiple Scale Prediction - Find Cars Algorithm

Video Pipeline

A link to the final video output is included in the github repository linked at the start of the report.

Heatmaps

As we are using a sliding window search with overlapping windows and different scale predictions, the predicted bounding boxes often overlap with others. We consolidate these bounding box predictions using a heatmap. Each box contributes 1 heat to the area of the image it covers, therefore areas with many overlapping boxes have high heat values. These high heat values are likely to be strong predictions. We can then threshold these heatmap values to only return high heat values, helping to reduce false positives and reducing the impact of random noisy predictions. Using the SciPy function Label we can consolidate individual heated regions into a single bounding box representing a vehicle detection. This process is illustrated in the images below.



Bounding Boxes and Heatmap Prediction

Error Detection

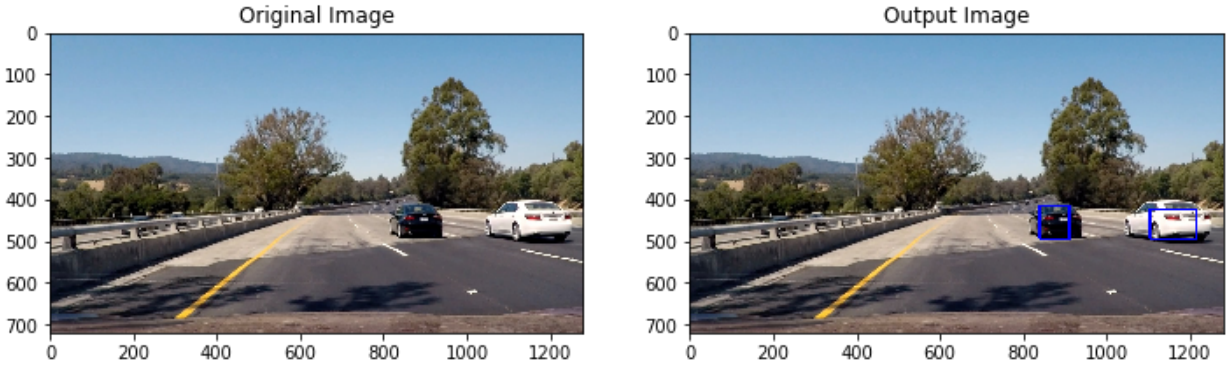
Given we are predicting vehicles in a video stream, we can average the predictions over a number of previous frames to produce a more stable and robust prediction. As a video is simply a series of individual images, we can run our classification on each of the image frames and store each frames heatmap. Once we have accumulated enough frames (10 frames), we begin to threshold on an average of the last five (5) frame's heatmaps, and predict bounding boxes based on this average.

The heatmap averaging reducing the image of noise from single frames, and helps produce more stable bounding boxes for vehicle position. We are able to average the heatmap, because the position of the vehicle is based on the position in the previous frame and we don't have to be concerned with the vehicle rapidly changing position from frame to frame. Once we have predicted the position of a vehicle, its next positions will be consistently around the initial position. We could improve the method by weighting more recent predictions more heavily to prevent the bounding boxes from lagging behind the vehicle at high speed.

Final Model and Pipeline Parameters

The following table shows the parameters for the entire project and pipeline.

Type	**Properties**
Feature Extraction	Color_space=YCrCb, orient=9 pix_per_cell=8, cell_per_block=2 spatial_size =(32, 32), hist_bins=32 spatial_feat=True, hist_feat=True, hog_feat=True
Prediction Model	Linear SVM (C=1000)
Sliding Window Search	HOG Subsampling window=64 cells_per_step=2
Error Detection	Scales=[1.0, 1.5] Heatmap Averaging: last five frames Heatmap Threshold: 8



Final Vehicle Detection Pipeline Output

Discussion

The pipeline uses computer vision techniques including spatial, colour and gradient (HOG) to extract features from images. These extracted features are used to train a classifier (Linear SVM) to detect vehicles. The pipeline can then uses this classifier to detect the location of vehicles in images using a sliding window method, allowing vehicles to be detected in a video stream.

The current feature extraction process is very manual and time intensive to set up. The feature extract step is quite computationally intensive, and therefore the HOG subsampling was an enormous performance improvement. The computer vision method feels very fragile, and even after spending a significant amount of time experimenting with combinations, the model produced a large amount of false positives. This is likely due to the size of the training set, but did not feel scalable or generalisable. The next step would be to compare the computer vision approach to a deep learning method.

During experimentation, the model was accidently trained on unnormalised data and therefore was producing a huge amount of false positives in testing. On reflection it was clear that a single feature was dominating all the others as discussed previously in the report. Once this error was corrected, the false positives dropped dramatically.

Combining different images scales provided the largest improvement on vehicle detection, whilst heatmap averaging was the biggest reduction in false positives. The current pipeline still produces some false positives, seemingly at random points throughout the video. In addition, the bounding boxes don't always completely bound the vehicle.

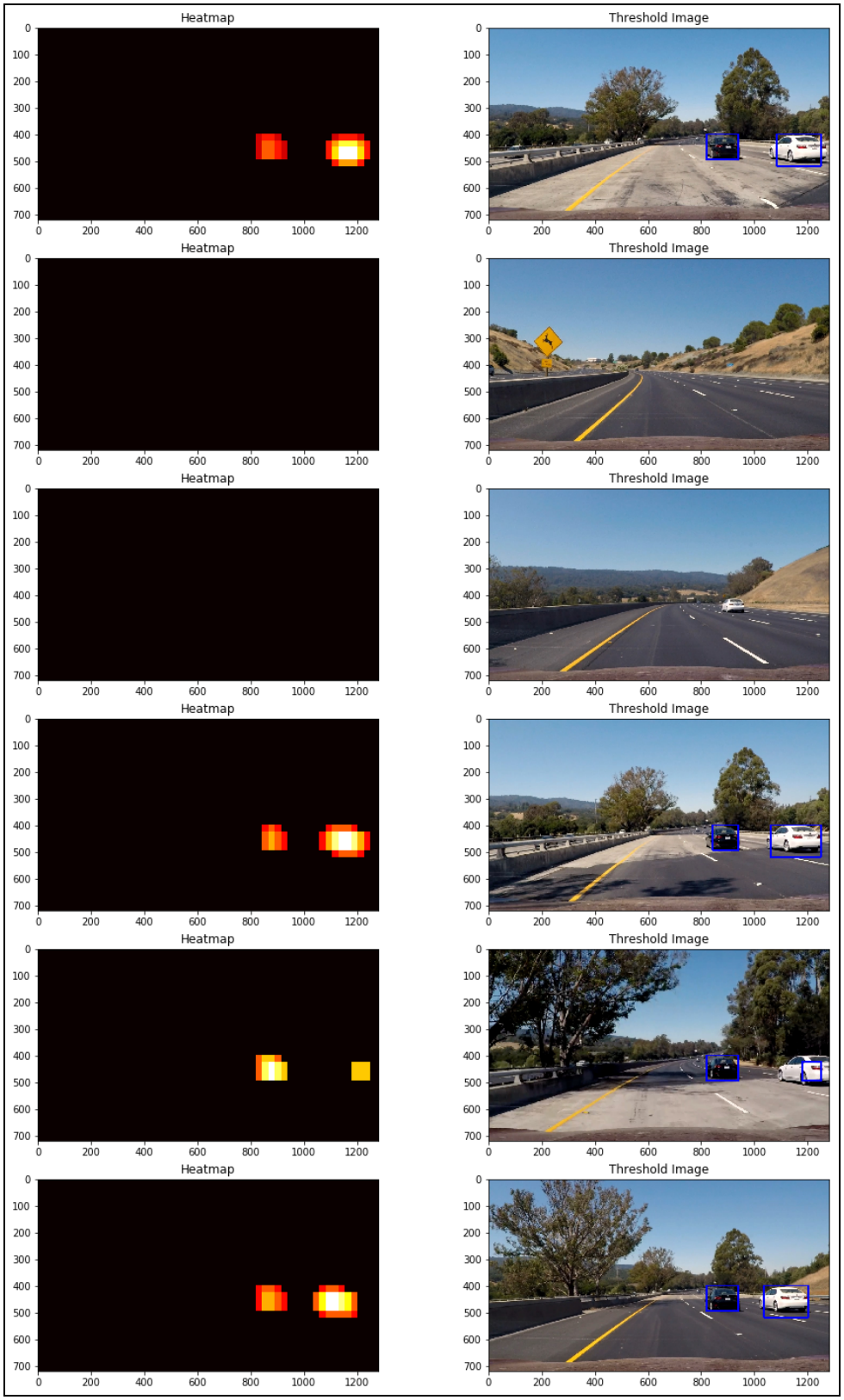
Pipeline performance is currently unacceptable for real-time processing. The 50 second video takes approximately 50 minutes to process on a MacBook Air. The performance must improve by orders of magnitude before being able to run on an actual autonomous vehicle.

Reflection

The project demonstrated the power, but complexity and time consuming nature of a computer vision and simple machine learning approach to the vehicle detection problem. It was a very manual process to set up, with feature extraction and false positive detection causing the pipeline to feel fragile.

To improve the pipeline:

- Implement a deep learning approach to the same problem, as suggested by other students
- Improve error detection by improving the way heatmaps are averaged, including adding increased weights to more recent predictions to improve bounding boxes.
- Improve the performance of the pipeline (50 min processing for 50 sec video)
- Test and try other videos.



Testing Road Images

Reference

Udacity, Self-Driving Car Project Q&A | Q&A
https://www.youtube.com/watch?v=P2zwrTM8ueA&feature=youtu.be