Aim:

To detect and track vehicles using svm classifier

Images:

output_images/

Output videos:

project_video.mp4

Feature Extraction:

Feature extraction is done in train.py and following features/params were used finally:

```
params = {
    'colorspace': 'YCrCb', # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
    'spatial_size': (32, 32),
    'hist_bins': 32,
    'orient': 18,
    'pixels_per_cell': 8,
    'cells_per_block': 2,
    'hog_channel': 'ALL', # Can be 0, 1, 2, or "ALL"
    'spatial_feat': True,
    'hist_feat': True,
    'hog_feat': True
    }
```

Feature extraction code is in feature_extraction.py. HOG features with ALL channels, color features in YCrCb space and spatial features were used as they together gave best results.

Training:

Linear SVC is used after scaling features to zero mean and unit variance (train.py). Classifier is defined in classifier.py

```
features = feature_extraction.extract_features_from_images(cars +
noncars,
        params['colorspace'], params['spatial_size'], params['hist_bins'],
        params['orient'], params['pixels_per_cell'], params['cells_per_block'],
        params['hog_channel'], params['spatial_feat'],
        params['hist_feat'], params['hog_feat'])

X = np.vstack(features).astype(np.float64)
X_scaler = StandardScaler().fit(X)
scaled_X = X_scaler.transform(X)

# Define the labels vector
y = np.hstack((np.ones(len(cars)), np.zeros(len(noncars))))

# Split up data into randomized training and test sets
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)

clf = classifier.get_linear_svc(X_train, y_train)
```

Sliding window search:

Implemented in search.py
find_cars() searches for a car in a given region, scale. search() implements logic for which scale, overlap to use. Following code shows important search params used:

```
y_start_stops = [(400, 425), (400, 700), (400, 700)]
x_start_stops = [(500, 900), (200, 1260), (200, 1260)]
yscales = [0.25, 1.5, 1.5]
xscales = [0.25, 1.5, 2]
cells_per_step = [4, 2, 1]
```

```
bboxes = []
allbboxes = []
prob = []
#for i in range(len(scales)):
for i in range(len(yscales)):
    ystart, ystop = y_start_stops[i][0], y_start_stops[i][1]
    xstart, xstop = x_start_stops[i][0], x_start_stops[i][1]
    b, p, a = find_cars(img_vec, feature_params, window_size,
                ystart, ystop, xstart, xstop, yscales[i], xscales[i],
                cells_per_step[i], clf, scaler)
```

Overall idea is to search at different scales at different regions of the image and also use different overlap according to the scale. Y scale and X scales have also been kept different as cars are usually not square in outer regions of the image. Lower scale is used for far away cars, but the region is restricted. Larger scale is used for nearer cars.

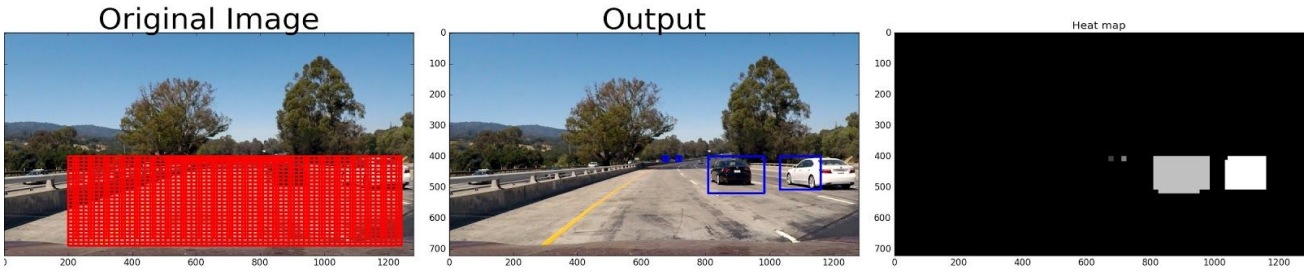
With larger scale more overlap is used (lower cells_per_step) as cars would be more compact at this scale. Conversely, less overlap(higher cells_per_step) is used at lower scale.

Optimizing classification performance:

Performance was optimized by experimenting with features params, and search params (scale, region of search). Some images showing detections at various experimentation stages are shown below. Images showing region of search, detected regions and heat maps are also shown.


Original Image
Output


Original Image
Output
Heat map



Filtering False Positives:

False positives are filtered by combining last few heatmaps and thresholding. (track_and_filter.py)

```
def track(self, img, hmap):
        self.hmaps.append(hmap)
        integrated_hmap =
sum(self.hmaps[-1*Track.NUM_FRAMES_TO_TRACK:])
        new_hmap = heat.apply_threshold(integrated_hmap, 2)
        new_labels = heat.get_labels(new_hmap)
        return new_labels
```

Discussion:

One of the major challenge in this project was that the algorithm takes quite some time to run, making incremental changes harder. Another challenging aspect was getting the right params for search window in order to not make it quite slow and also be reasonably useful.

The pipeline might fail if the cars are too close for some reason. Classification could fail if there are cars at angles not seen previously, for example somebody coming from right or left. Lighting might also have an affect, cars at night might not be detectable by same classifier.

To make the pipeline more robust, classification could be improved by collecting and augmenting data. Apart from that, we could also detect smaller features like tyres, mirrors, head/brake lamps to gain confidence.