# Vehicle Detection Project

The goals / steps of this project are the following:

- Perform a Histogram of Oriented Gradients (HOG) feature extraction on a labelled training set of images and train a classifier Linear SVM classifier

- Optionally, you can also apply a colour transform and append binned colour features, as well as histograms of colour, to your HOG feature vector.

- Note: for those first two steps don't forget to normalize your features and randomize a selection for training and testing.

- Implement a sliding-window technique and use your trained classifier to search for vehicles in images.

- 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.

- Estimate a bounding box for vehicles detected.

## Histogram Oriented Gradients (HOG)

I began doing some tests to get a useful color space and finally used YUV.

Here is a table of some of the experiments

| Color Space | Channel | Orientations | Pixels per Cell | Cells Per Block | Accuracy |
|---|---|---|---|---|---|
| RGB | 0 | 9 | 8 | 2 | 0,91 |
| RGB | 1 | 9 | 8 | 2 | 0,94 |
| RGB | 2 | 9 | 8 | 2 | 0,925 |
| RGB | ALL | 9 | 8 | 2 | 0,925 |
| HSV | 0 | 9 | 8 | 2 | 0,955 |
| HSV | 1 | 9 | 8 | 2 | 0,865 |
| HSV | 2 | 9 | 8 | 2 | 0,935 |
| HSV | ALL | 9 | 8 | 2 | 0,99 |
| YUV | 0 | 9 | 8 | 2 | 0,94 |
| YUV | 1 | 9 | 8 | 2 | 0,975 |
| YUV | 2 | 9 | 8 | 2 | 0,96 |
| YUV | ALL | 9 | 8 | 2 | 0,99 |
| LUV | 0 | 9 | 8 | 2 | 0,92 |
| LUV | 1 | 9 | 8 | 2 | 0,95 |
| LUV | 2 | 9 | 8 | 2 | 0,93 |

| | | | | | | |
|---|---|---|---|---|---|---|
| LUV | ALL | | 9 | 8 | 2 | 0,98 |
| HLS | | 0 | 9 | 8 | 2 | 0,95 |
| HLS | | 1 | 9 | 8 | 2 | 0,955 |
| HLS | | 2 | 9 | 8 | 2 | 0,875 |
| HLS | ALL | | 9 | 8 | 2 | 0,975 |
| YCrCb | | 0 | 9 | 8 | 2 | 0,92 |
| YCrCb | | 1 | 9 | 8 | 2 | 0,92 |
| YCrCb | | 2 | 9 | 8 | 2 | 0,94 |
| YCrCb | ALL | | 9 | 8 | 2 | 0,99 |
| | | | | | | |
| HSV | | 0 | 9 | 6 | 1 | 0,965 |
| HSV | | 0 | 9 | 6 | 2 | 0,955 |
| HSV | | 0 | 9 | 6 | 3 | 0,955 |
| HSV | | 0 | 9 | 8 | 1 | 0,975 |
| HSV | | 0 | 9 | 8 | 2 | 0,98 |
| HSV | | 0 | 9 | 8 | 3 | 0,95 |
| HSV | | 0 | 9 | 12 | 1 | 0,955 |
| HSV | | 0 | 9 | 12 | 2 | 0,955 |
| HSV | | 0 | 9 | 12 | 3 | 0,94 |
| HSV | | 1 | 9 | 6 | 1 | 0,86 |
| HSV | | 1 | 9 | 6 | 2 | 0,91 |
| HSV | | 1 | 9 | 6 | 3 | 0,885 |
| HSV | | 1 | 9 | 8 | 1 | 0,86 |
| HSV | | 1 | 9 | 8 | 2 | 0,895 |
| HSV | | 1 | 9 | 8 | 3 | 0,855 |
| HSV | | 1 | 9 | 12 | 1 | 0,87 |
| HSV | | 1 | 9 | 12 | 2 | 0,88 |
| HSV | | 1 | 9 | 12 | 3 | 0,83 |
| HSV | ALL | | 9 | 6 | 1 | 0,985 |
| HSV | ALL | | 9 | 6 | 2 | 0,995 |
| HSV | ALL | | 9 | 6 | 3 | 0,97 |
| HSV | ALL | | 9 | 8 | 1 | 0,99 |
| HSV | ALL | | 9 | 8 | 2 | 0,98 |
| HSV | ALL | | 9 | 8 | 3 | 0,975 |
| HSV | ALL | | 9 | 12 | 1 | 0,97 |
| HSV | ALL | | 9 | 12 | 2 | 0,97 |
| HSV | ALL | | 9 | 12 | 3 | 0,975 |
| YUV | | 0 | 9 | 6 | 1 | 0,975 |
| YUV | | 0 | 9 | 6 | 2 | 0,945 |
| YUV | | 0 | 9 | 6 | 3 | 0,915 |
| YUV | | 0 | 9 | 8 | 1 | 0,94 |
| YUV | | 0 | 9 | 8 | 2 | 0,93 |
| YUV | | 0 | 9 | 8 | 3 | 0,945 |

| | | | | | |
|---|---|---|---|---|---|
| YUV | 0 | 9 | 12 | 1 | 0,915 |
| YUV | 0 | 9 | 12 | 2 | 0,945 |
| YUV | 0 | 9 | 12 | 3 | 0,94 |
| YUV | 1 | 9 | 6 | 1 | 0,99 |
| YUV | 1 | 9 | 6 | 2 | 0,98 |
| YUV | 1 | 9 | 6 | 3 | 0,995 |
| YUV | 1 | 9 | 8 | 1 | 0,98 |
| YUV | 1 | 9 | 8 | 2 | 0,98 |
| YUV | 1 | 9 | 8 | 3 | 0,965 |
| YUV | 1 | 9 | 12 | 1 | 0,98 |
| YUV | 1 | 9 | 12 | 2 | 0,975 |
| YUV | 1 | 9 | 12 | 3 | 0,985 |
| YUV | ALL | 9 | 6 | 1 | 0,985 |
| YUV | ALL | 9 | 6 | 2 | 0,995 |
| YUV | ALL | 9 | 6 | 3 | 0,99 |
| YUV | ALL | 9 | 8 | 1 | 1 |
| YUV | ALL | 9 | 8 | 2 | 0,99 |
| YUV | ALL | 9 | 8 | 3 | 0,995 |
| YUV | ALL | 9 | 12 | 1 | 0,99 |
| YUV | ALL | 9 | 12 | 2 | 0,99 |
| YUV | ALL | 9 | 12 | 3 | 0,995 |

Working with a small subset of data (the one from the course).

Finally after some tests with the full set I decided to continue with YUV and maintain the 0 x 8 pixels per cell and 2x2 cells per block.

Hog and color features are extracted in procedure extract_img_features in the utilities.py file

```
def extract_img_features(img_file, color_space='YUV',
                         spatial_size=(32, 32),
                         hist_bins=32,
                         orient=9,
                         pix_per_cell=8,
                         cell_per_block=2,
                         spatial_feat=True,
                         hist_feat=True,
                         hog_feat=True,
                         hog_channel='ALL'):

    img_features = []
    img = cv2.imread(img_file)
    img = cv2.resize(img, (64,64))

    feature_image = convert_color(img, color_space).astype(np.float32) /
255

    if spatial_feat == True:
        spatial_features = bin_spatial(feature_image, size=spatial_size)
        # 4) Append features to list
```

```
        img_features.append(spatial_features)
    # 5) Compute histogram features if flag is set
    if hist_feat == True:
        hist_features = color_hist(feature_image, nbins=hist_bins)
        # 6) Append features to list
        img_features.append(hist_features)
    # 7) Compute HOG features if flag is set
    if hog_feat == True:
        if hog_channel == 'ALL':
            hog_features = []
            for channel in range(feature_image.shape[2]):
                hog_features.extend(get_hog_features(feature_image[:, :,
channel],
                                                     orient, pix_per_cell,
cell_per_block,
                                                     vis=False,
feature_vec=True))
        else:
            hog_features = get_hog_features(feature_image[:, :,
hog_channel], orient,
                                            pix_per_cell, cell_per_block,
vis=False, feature_vec=True)
        # 8) Append features to list
        img_features.append(np.array(hog_features, dtype=np.float32))

    # 9) Return concatenated array of features
    return np.concatenate(img_features)
```

## 3.- Describe how you trained the classifier with the selected HOG and colour features

I wrote the code for 2 classifiers, SVM and a CNN derived from the one for the traffic signs to be able to compare the two approaches.

Fast_training.py is the code for training the svm. To made it easier to add more images it just looks at two subdirectories, vehicles and non-vehicles and processes them recursively.

The procedure is:

```
X = np.concatenate((car_features, notcar_features, false_features,
positives_features), axis=0).astype(np.float64)
X_scaler = StandardScaler().fit(X)
scaled_X = X_scaler.transform(X)

with open(scaler_filename, "wb") as f:
    pickle.dump(X_scaler, f)


# Labels Vector
y = np.hstack((np.ones(len(car_features)),
               np.zeros(len(notcar_features)),
               np.zeros(len(false_features)),
               np.ones(len(positives_features))))

# slit and shuffle
rand_state = np.random.randint(0, 100)
X_train, X_test, y_train, y_test = train_test_split(
    scaled_X, y, test_size=test_percent, random_state=rand_state)
```

```
#svc = LinearSVC()

svc = SVC(kernel=kernel, C=5, gamma=0.00005, verbose=True)
# Check the training time for the SVC
print("Training...")
t = time.time()
svc.fit(X_train, y_train)
t2 = time.time()
print("Checking...")
accuracy = svc.score(X_test, y_test)
print("Time to train", t2-t, "Accuracy ", round(accuracy, 4))
print(svc.get_params())
with open(model_filename, "wb") as f:
    pickle.dump(svc, f)
```

The filelist is obtained by recursively roaming a directory. That helps adding and subtracting data.

I have done some varying C and gamma and changing the data by adding false positives.

Finally I have used  C = 5 and Gamma = 5E-5 with an rbf kernel which gave me a 0.9917 accuracy on the validation set.

For the CNN I used the learn_cars.py program over the Udacity data but first had tu use the extract.py program to generate 64x64 images of cars and non cars from the data.

As the volume of data was quite big I used a generator for reading the images into a Keras model.

I just converted images to YUV (probably not necessary) and scaled them to be between -0.5 and +0.5.

Once trained over 15 epochs on Udacity set data I did 15 more epochs over the project + false positives data.

## Sliding Window Search

The Sliding window approach is used either in the SVM models and in the CNN models. It is done in the find_car_boxes_svm function for the svm models and the find_car_boxes_cnn for the cnn, both in the utilities.py file.

```
def find_cars_boxes_svm(img, svc, X_scaler, ystart=None, ystop=None,
scale=1,
                orient=9,
                pix_per_cell=8,
                cell_per_block=2,
                spatial_size=(32,32),
                hist_bins=32,
                color_space='YUV', filter=False):


    img = img.astype(np.float32) / 255

    if ystart is None:
        ystart = int(img.shape[0]/2)

    if ystop is None:
```

```
        ystop = int(img.shape[0])

    img_tosearch = img[ystart:ystop, :, :]
    ctrans_tosearch = convert_color(img_tosearch, conv=color_space)

    # ctrans_tosearch = convert_color(img_tosearch, conv='RGB2YUV')
    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1] /
scale), np.int(imshape[0] / scale)))

    ch1 = ctrans_tosearch[:, :, 0]
    ch2 = ctrans_tosearch[:, :, 1]
    ch3 = ctrans_tosearch[:, :, 2]

    # Define blocks and steps as above
    nxblocks = (ch1.shape[1] // pix_per_cell) - cell_per_block + 1
    nyblocks = (ch1.shape[0] // pix_per_cell) - cell_per_block + 1
    nfeat_per_block = orient * cell_per_block ** 2

    # 64 was the original sampling rate, with 8 cells and 8 pix per cell
    window = 64
    nblocks_per_window = (window // pix_per_cell) - cell_per_block + 1
    cells_per_step = 2  # Instead of overlap, define how many cells to
step
    nxsteps = (nxblocks - nblocks_per_window) // cells_per_step
    nysteps = (nyblocks - nblocks_per_window) // cells_per_step

    # Compute individual channel HOG features for the entire image
    hog1 = get_hog_features(ch1, orient, pix_per_cell, cell_per_block,
feature_vec=False)
    hog2 = get_hog_features(ch2, orient, pix_per_cell, cell_per_block,
feature_vec=False)
    hog3 = get_hog_features(ch3, orient, pix_per_cell, cell_per_block,
feature_vec=False)

    box_list = []

    for xb in range(nxsteps):
        for yb in range(nysteps):
            ypos = yb * cells_per_step
            xpos = xb * cells_per_step
            # Extract HOG for this patch
            hog_feat1 = hog1[ypos:ypos + nblocks_per_window, xpos:xpos +
nblocks_per_window].ravel()
            hog_feat2 = hog2[ypos:ypos + nblocks_per_window, xpos:xpos +
nblocks_per_window].ravel()
            hog_feat3 = hog3[ypos:ypos + nblocks_per_window, xpos:xpos +
nblocks_per_window].ravel()
            hog_features = np.hstack((hog_feat1, hog_feat2, hog_feat3))

            xleft = xpos * pix_per_cell
            ytop = ypos * pix_per_cell

            # Extract the image patch
            subimg = cv2.resize(ctrans_tosearch[ytop:ytop + window,
xleft:xleft + window], (64, 64))

            # Get color features
```

```
            spatial_features = bin_spatial(subimg, size=spatial_size)
            hist_features = color_hist(subimg, nbins=hist_bins)

            # Scale features and make a prediction
            #test_features = X_scaler.transform(
            unscaled_features = np.hstack((spatial_features,
hist_features, hog_features)).reshape(1, -1)
            #unscaled_features = np.hstack((spatial_features,
hog_features)).reshape(1, -1)
            test_features = X_scaler.transform(unscaled_features)
            # test_features = X_scaler.transform(np.hstack((shape_feat,
hist_feat)).reshape(1, -1))
            test_prediction = svc.predict(test_features)

            if test_prediction == 1:
                xbox_left = np.int(xleft * scale)
                ytop_draw = np.int(ytop * scale)
                win_draw = np.int(window * scale)

                box_list.append(((xbox_left, ytop_draw +
ystart),(xbox_left + win_draw, ytop_draw + win_draw + ystart)))
                #cv2.rectangle(draw_img, (xbox_left, ytop_draw + ystart),
                #              (xbox_left + win_draw, ytop_draw + win_draw
+ ystart), (0, 0, 255), 6)

    return box_list, hog1
```

and

```
def find_cars_boxes_cnn(img, model, ystart=None, ystop=None, scale=1,
color_space='YUV', filter=False):

    if ystart is None:
        ystart = int(img.shape[0]/2)

    if ystop is None:
        ystop = int(img.shape[0])

    img_tosearch = img[ystart:ystop, :, :]
    ctrans_tosearch = convert_color(img_tosearch, conv=color_space)
    ctrans_tosearch = (ctrans_tosearch - 128) / 255

    if scale != 1:
        imshape = ctrans_tosearch.shape
        ctrans_tosearch = cv2.resize(ctrans_tosearch, (np.int(imshape[1] /
scale), np.int(imshape[0] / scale)))

    # 64 was the original sampling rate, with 8 cells and 8 pix per cell
    window = 64
    pixels_per_step = 16

    cells_per_step = 2  # Instead of overlap, define how many cells to
step
    nxsteps =  (ctrans_tosearch.shape[1]-
window+pixels_per_step)//pixels_per_step
    nysteps =  (ctrans_tosearch.shape[0]-
window+pixels_per_step)//pixels_per_step

    box_list = []
```

```
    all_box_list = []
    image_list = []
    for xb in range(nxsteps):
        for yb in range(nysteps):
            ypos = yb * pixels_per_step
            xpos = xb * pixels_per_step

            xbox_left = np.int(xpos * scale)
            ytop_draw = np.int(ypos * scale)
            win_draw = np.int(window * scale)

            all_box_list.append(((xbox_left, ytop_draw + ystart),
(xbox_left + win_draw, ytop_draw + win_draw + ystart)))

            subimg = ctrans_tosearch[ypos:ypos + window, xpos:xpos +
window]
            image_list.append(subimg)

    x_data = np.array(image_list)

    x_pred = model.predict(x_data)

    for result, box in zip(x_pred, all_box_list):
        value = np.argmax(result)
        if value == 0:
             box_list.append(box)

    return box_list
```

Results are passed to the recognize_cars function in find_cars.py which manages the scales and the heatmap and returns a list of cars.

I finally used scales 1 and 1.5 as it was already too slow for my taste and it didn't seem to improve with more scales.

## Heatmap filter

Once I had the possible boxes for cars I did a heatmap algorithm for combining overlapping windows. The heatmap threshold seems a very important parameter.

I also recentered the boxes around a weighted average of the heatmap values to get better center.

All this is done in the recogze_cars function in find_cars.py and add_heat and find_car functions in utilities.py.

```
def recognize_cars(classifier, model, scaler, image,
                   false_positives_dir=None,
                   n=0,
                   threshold=3,
                   scales=[1],
                   ystart=400,
                   ystop=700,
                   input_boxes=None,
                   filter=False):

    heatmap = np.zeros_like(image[:, :, 0]).astype(np.float)
```

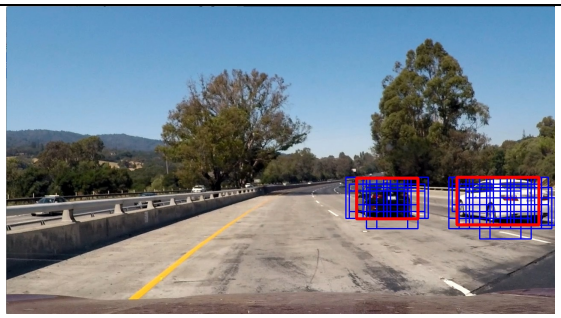
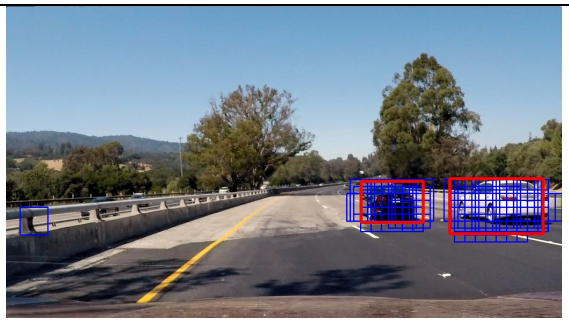
```
    all_boxes = []

    if input_boxes is None:
        for i_scale in scales:

            if classifier == 'cnn':
                box_list = find_cars_boxes_cnn(image, model,
                                               ystart=ystart, #400
                                               ystop=ystop,
                                               scale=i_scale,
#np.power(1.05, float(i_scale)),
                                               color_space='YUV',
filter=filter)
            else:
                box_list = find_cars_boxes_svm(image, model, scaler,
                                               ystart=ystart, #400
                                               ystop=ystop,
                                               scale=i_scale,
#np.power(1.05, float(i_scale)),
                                               color_space='YUV',
filter=filter)


            all_boxes += box_list
            add_heat(heatmap, box_list)
    else:

        all_boxes += input_boxes
        add_heat(heatmap, input_boxes)

    car_boxes = find_car(heatmap,  threshold=threshold)

    # Try to get a list of cars from positions

    cars = []   # List of Car objects
    # Get car image
    for car_box in car_boxes:
        car_image = bbox_subimage(image, car_box)
        color = average_color(car_image)
        position = bbox_center(car_box)
        size = bbox_size(car_box)
        car = Car(position, size, color, image)
        cars.append(car)


    if false_positives_dir is not None:
        for box in all_boxes:
            isnotcar = True
            for car_box in car_boxes:
                if intersect(box, car_box): # Is a car,
                    isnotcar = False
                    break

            if isnotcar:
                n += 1
                cv2.imwrite(false_positives_dir+"/"+str(n)+".png",
image[box[0][1]:box[1][1], box[0][0]:box[1][0]])
```

```
    return cars, car_boxes, heatmap, all_boxes, n
```

This function encapsulates cars in specific objects (Car) and also provides the functionality of storing images of the false positives windows.

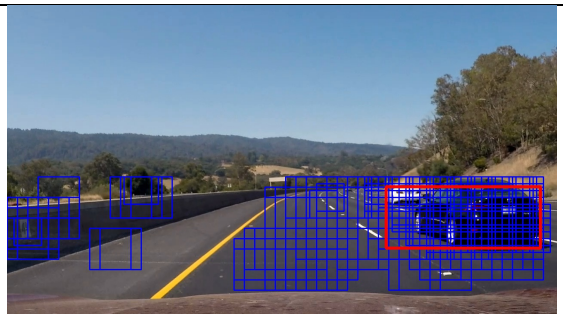
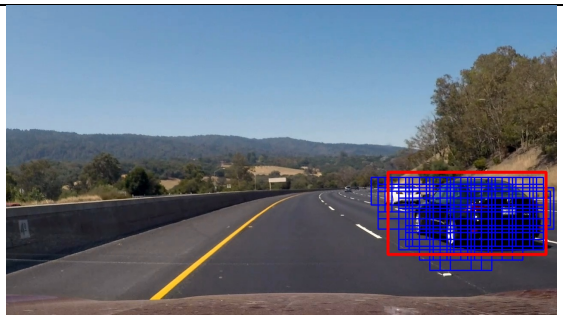
## Examples of the Pipeline

Here you have of the results of applying the SVM and the CNN classifiers. Red boxes are car recognized and blue boxes are boxes fed to the heatmap algorithm.

| SVC rbf, c=5, gamma=5E-5 | CNN |
| --- | --- |
|  |  |
|  |  |
|  |  |

As you may see, the CNN is much more selective but the two give similar positions for the cars.

Of course there are some problems as shown in the following pictures, both algorithms join the two cars into one. I have find this problem very difficult to solve.

## Video Implementation

### 1.- Provide a link to your final video output

I did two videos, first one with the SVM is here, second one with the CNN is here.

In the two videos cars are clearly marked with a red box. A number in the box is an estimate of how many meters to the right (+) or left(-) is the car. That is not the distance but the distance to the right or left.

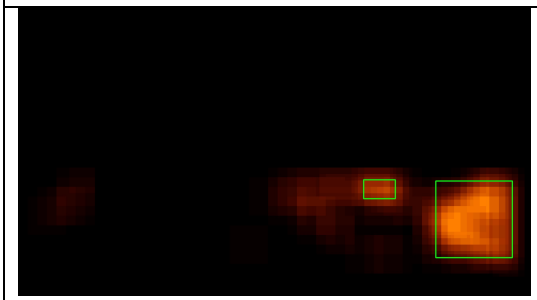
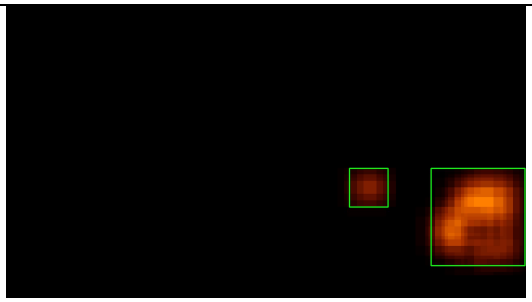
In the SVM (svc) video at minute 43 there is a detection in the left  corresponding to a car driving in opposite direction.

SVM(Svc) video is at https://youtu.be/HxpgUr-o45I

CNN video is at https://youtu.be/WaY5TVmJFxE

### 2.- 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.

As explained before I use a Heatmap algorithm to detect the cars from the positive boxes. The result of applying it is shown in the following images corresponding to the svm and cnn :

| Svm | Cnn |
| --- | --- |
|  |  |
|  |  |

Once applied the excess in boxes in the svm algorithm is solved without problems.

## 3.- Tracking the cars

For the video I did an additional pass so I don't have false recognitions:

- I maintain a list of actual cars. Each car has position, size, color and speed information and two important variables
- Number of frames seen consecutively
- Number of frames unseen

- Once I detect a possible car I create an entry in the list but is not marked as "really a car" until Number of frames seen consecutively is greater than 5 (that value should be adjusted). That removes most false positives.

- If a car is not seen for more than 5 frames it is removed from the list. That crates some problems in occlusions as it forgets the occluded car.

All this is implemented in the Car and CarCollection in the find_cars.py program:

```
def draw(self, image, font):

    thick = 2

    if self.frames_seen < 5 or self.frames_unseen > self.frames_seen:
        return

    if self.frames_seen > 2:
        thick = 6

    if self.frames_unseen > 1:
        thick = 2

    b = self.car_box()
    color = self.drawing_color
    cv2.rectangle(image, b[0], b[1], color, thickness=thick)
```

and

```
def prepare_for_update(self):

    new_cars = []
    for car in self.cars:
        if car.frames_unseen > car.frames_seen or car.frames_unseen > 5:
            # Remove car
            print("Removing car {}".format(car.name))
        else:
            car.frames_unseen += 1  # If seen will be set to 0
            new_cars.append(car)


    self.cars = new_cars
```

The other problem is finding cars. We have a list of cars and a list of recognized cars in the image and must match them someway. The algorithm I use is in method find_car of Car_Collection:

```
def find_car(self, a_car, delta):

    results = []

    for car in self.cars:
        if car.distance(a_car.position) < delta*(car.frames_unseen+1):
            results.append(car)

        elif included(car.car_box(), a_car.car_box()):
            results.append(car)

        elif intersect(car.car_box(), a_car.car_box()): # No ho tinc tan
clar
            results.append(car)

    return results
```

and

```
def new_observation(self, car):

    # Look if there is a car that exists

    found_cars = self.find_car(car, 40)

    ## No cars, add it

    if len(found_cars) == 0:
        self.add_car(car)
        print("Added car", car.car_info())

    ## Found a car
    elif len(found_cars) == 1:
        the_car = found_cars[0]
        the_car.update_data(car)
        d = color_difference(the_car.color, car.color)
        print("Car {} with color difference {}".format(the_car.name, d))

    else:   # More than one car, try to select one

        the_car = None
        color_diff = 999999999999999
        area = 0

        for a_car in found_cars:
            if a_car.area() > area:
                d = color_difference(a_car.color, car.color)
                the_car = a_car
                color_diff = d
                area = a_car.area()

                #if d < color_diff:
                #the_car = a_car
```

```
                #color_diff = d

        # great, I have a car
        the_car.update_data(car)
        print("Selected car {} with color difference
{}".format(the_car.name, color_diff))
```

To break ties I have used different ways as getting the bigger car of the colour but clearly It needs more work.

I also implemented first a Kalman like system for following positions and speed but didn't get it working very well. Finally I use averages over some frames and they seem give a better result.

## Discussion

1. 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?

Main problem I have found is boxes are not tight against the car. Probably I need more training or an algorithm to tight the boxes. I have tested some ideas as using a flood fill with the colour of the car (works very well with the white car but not with the grey one) but have not arrived to final conclusions.

I also worked to get 3D positions and from here try to get better boxes. Also not a final result although it seems promising. I got reasonably numbers for lateral distance from our car and with some more work may give data about distance from previous car.

Biggest problem of all was speed. It is very slow and it is annoying to wait between 40" to 1' for frame. Surely there must be a better solution than the sliding windows system because if not we will need quite fast hardware.

I am more satisfied with the CNN classifier as it gives me better details but it may be because I need much more time to tune SVM parameters.

Some ideas to get better data :

- Calibrate distance to front car
- Add a bird view of my cars and all cars
- Include the lane recognition code
- Get better car tracking, know which car is which and work speeds

## Calling the program

Some switches enable you to run the find_cars program with different options

Python find_cars.py <input video or foto folder> …

-video switch to process video. If not it is photo processing

-o <output video>

-c <svc|cnn> is the classifier

-m <model name> either .hf5 or .pkl

-sc <scaler name> in case of svc

-fp <directory name> directory where to write false positives

-log <directory name> directory where to store each frame of the video

-first <number> first video frame to analyse

-last <number> last video frame to analyse

-d <filename> name of a file where to dump the recognized window boxes of every frame

-l <filename> instead analysing video frames use the boxes in <filename>

-interactive show the video frames in a window. If not works silently

## Program files

- Utilities.zip has most of utilities and car recognizing code
- Find_cars.py is the main program and implements car tracking
- Fast_training.py is used to train svc's
- Learn_cars.py is the trainer for the cnn's
- Relearn.py is for continuing training an already existing cnn model
- Extract.py is used for convert the files in object-detection-crowdai in Udacity dataset to our 64x64 images for training the cnn's
- Extract_samples.py is for scanning a car or not car image and generating 64x64 images for training.