REPORT: PROJECT 5. VEHICLE DETECTION AND TRACKING (NEURAL NETWORK CLASSIFIER)

Author: Markus Buchholz

Project consist of following files:

1. Project5_SDCar_markus_buchholz. pdf => project report
2. Project5_markus_buchholz.ipynb => project code in jupyter
3. output video:
    a. project_video_detected_combined_cnn_10.mp4

1. Introduction

This report discusses the approach of vehicle detection and tracking based on computer vision (OpenCV) and Neural Network (NN) which was used to build a classifier (TensorFlow on GPU). Final detector includes the lane detector (described in details in Project 4) and vehicle detector. Detailed information about the lane detector was given in project 4 report therefore following report focuses only on vehicle detector. Presented solution consist of 2 classifier: SVM presented during the classes and NN (in Keras). Both classifiers fit the data but only classifier based on NN is used for class prediction (test images and videos were classified by NN).

The discussed approach consists of following steps:

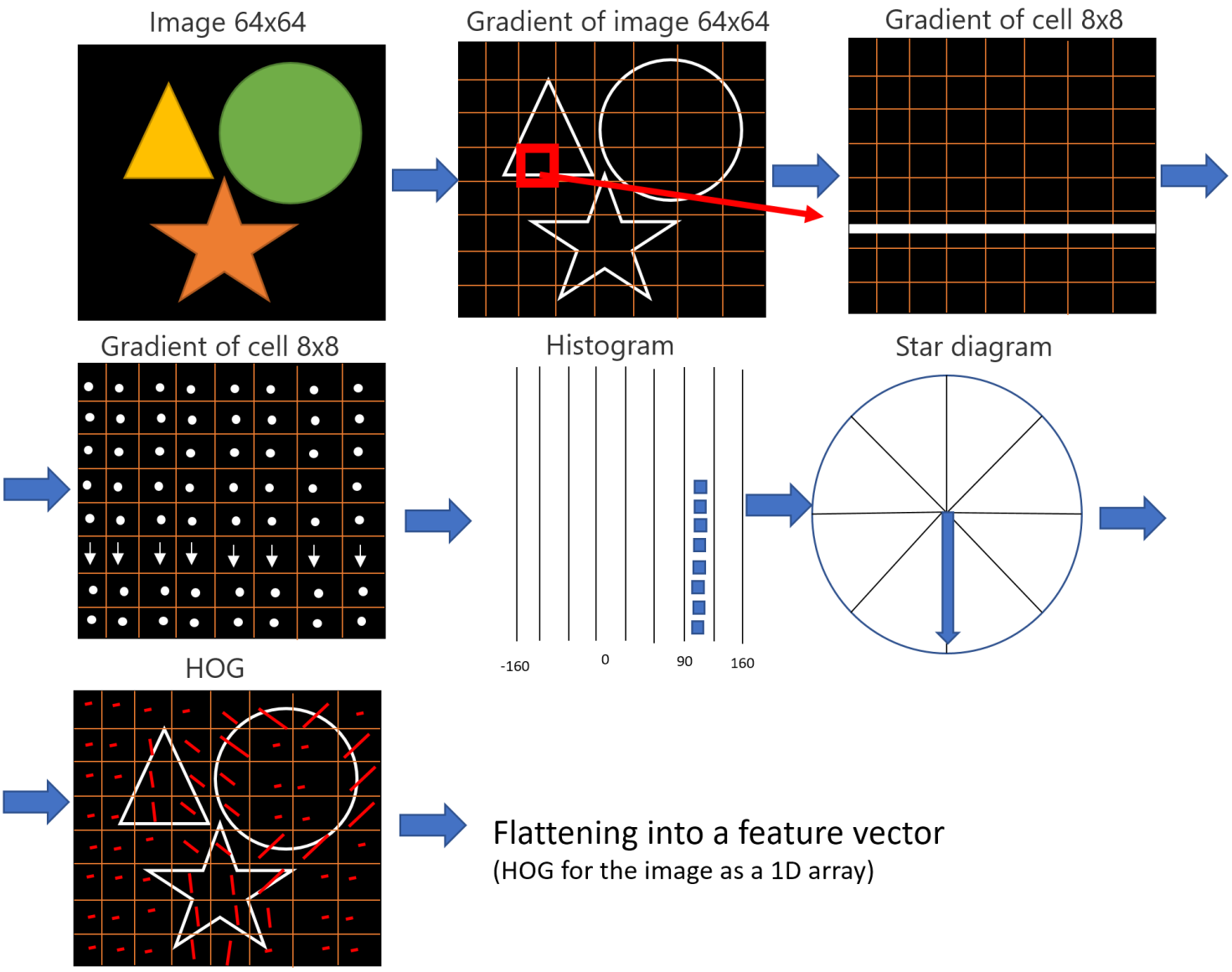
Covered by project 4	• Compute the camera calibration matrix and distortion coefficients given a set of chessboard images. • Apply a distortion correction to raw images. • Use color transforms, gradients, etc., to create a thresholded binary image. • Apply a perspective transform to rectify binary image ("birds-eye view"). • Detect lane pixels and fit to find the lane boundary. • Determine the curvature of the lane and vehicle position with respect to center. • Warp the detected lane boundaries back onto the original image. • Apply sanity check of estimated polynomials (left and right). • Apply buffer in order to smooth over the last *n* frames of video and obtain a cleaner result.
Covered by project 5	• Elaboration of the number and size of windows for the image. The image it “divided” by sliding window which further is taken for processing and prediction. • Color conversion in order to boost the separation class possibility (vehicle images and non-vehicles) – for sliding window. • Spatial binning of color – for sliding window image. • Computation of color histogram – for sliding window image. • Computation of histogram gradient orientation HOG – for sliding window image. • Concatenate results of sliding window image processing. • Perform prediction for window image (the window image is classified). Classifier is build previously based on provided data. Data is processed in the same way as described above, except applying the sliding window step. • For each sliding window classified as vehicle the window box is estimated. • Each pixel in estimated box is amplified so the heat map for the each given image can be computed. • Applying the threshold in order to filter the position of detected vehicle (filtering the heat map). • For the heat map applying the label() function which returns the number of detects (her vehicles) and positions. • Output image is: visual display of the lane boundaries, numerical estimation of lane curvature, driven vehicle position against the track, drawn boxes over detected cars on the road.

2. Histogram of Oriented Gradient (HOG)

For each pixel in training image (size of 64 by 64) the gradient, magnitude and direction is computed. Subsequently this picture(64x64) is divided by cells (8x8 pixel each). For each cell then, the histogram of gradient orientation is computed. The gradient sample is distributed among 8 bins creating then the resulting histogram of cell. Afterwards, the bins of the histogram are transferred to the star diagram, where each bin reflect the vector. Vector length reflect the size of the bin (more pixels in the discussed cell were classified to the regarding bin so the longer vector is).

HOG was computed with use of scikit-image package. The inbuild function gives the possibility to adjust the orientation bins, pixels per cell and cell per block.

HOG algorithm can be represented as follows.

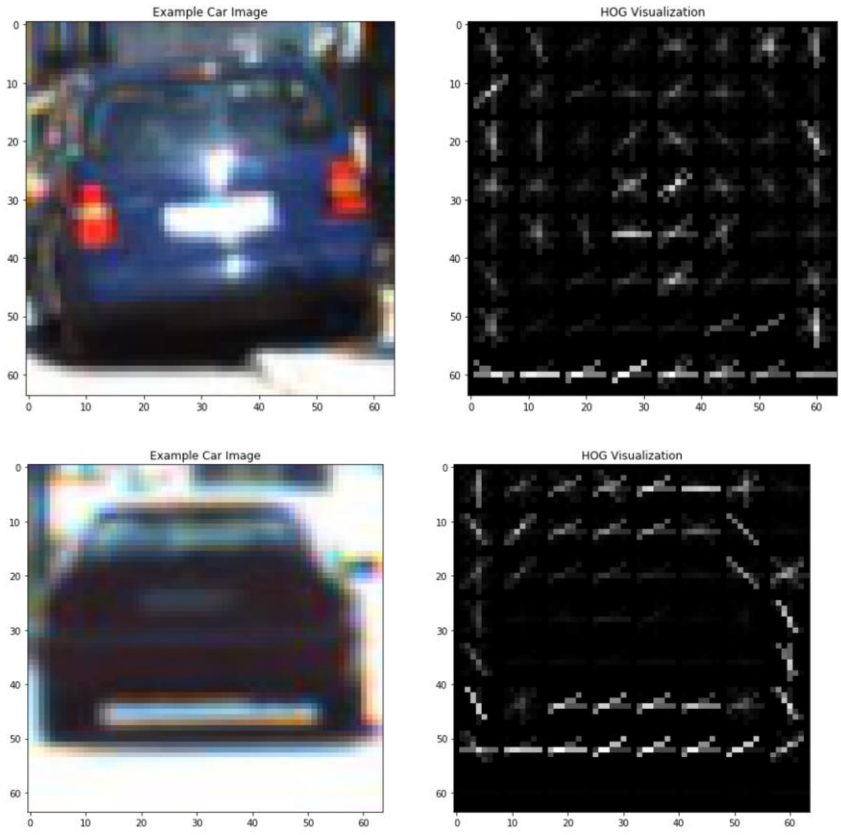


3. Adjusting of HOG parameters

The HOG parameters were chosen/checked during the iteration process. It was noticed that the setup given during the lessons give the best results. For HOG it was used 8 orientations (also other variants were checked like 10 and 12), 8 pixels per cell and 2 cells per block. The HOG was applied for all the color channels. The total number of features in final (return from hog() function is: the total number of block positions multiplied by the number of cells per block, times the number of orientations, or in the case shown above: 7×7×2×2×8=1568. Python code for HOG implementation is given below.

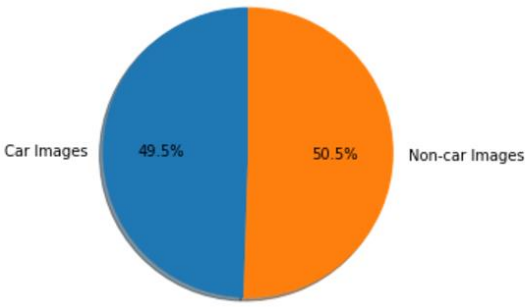
```
# HOG
def get_hog_features(img, orient, pix_per_cell, cell_per_block, vis=True, feature_vec=True):

    if vis == True:
        features, hog_image = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                                  cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=False,
                                  visualise=True, feature_vector=False)
        #testing the HOG
        #plt.imshow(hog_image)
        #plt.show()
        #print(features)
        return features, hog_image
    else:
        features = hog(img, orientations=orient, pixels_per_cell=(pix_per_cell, pix_per_cell),
                       cells_per_block=(cell_per_block, cell_per_block), transform_sqrt=False,
                       visualise=False, feature_vector=feature_vec)


        return features
```

Below pictures depict examples of HOG image processing for training images64x64 pixels.

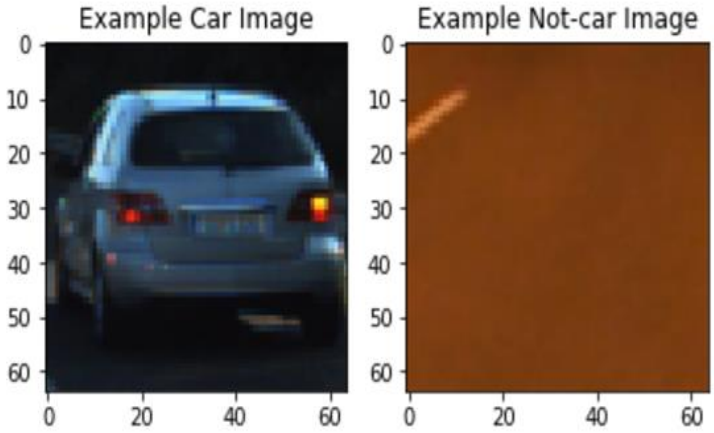


4. Training of classifier

In this project, the classifier was based on NN (Keras). In first attempt classifier was base on SVM but during the project research (experimental work) it was decided to apply the other classifier, which are more robust and offers more hyperparameters to be tuned. Main advantage for NN is that architecture can be adjusted by engineer reflecting actual challenge to solve. SVM classifier is still included in this project code but the predictions are performed by NN.
In order to train a classifier, the provided data (by Udacity class links; exploration of provided data was given below) had to be adjusted to the same data format (concatenate feature data) which is used in main pipeline of the project (for analyzing incoming camera data frames). So the training data went through color conversion, special color binning, computation of color histogram and HOG. After performing concatenating of data features (for each training image separately) the sample (flatten data array) had to be standardized (StandardScaler() was used) by removing the mean and scaling to unit variance. Having the data set, the data was splited to training set (80%) and testing set (20%). Test data set was used to evaluate the accuracy of classifier. In this project classifier was build by use of neural network. Keras high-level API was used (the same as in project 3). Applied architecture was based on example given on Keras website.




```
Number of pictures: CAR : 8792
Number of pictures: NON-CAR : 8968
image size (64, 64, 3)
```



Architecture of applied neural network is given below:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 4096)              22810624
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_3 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_3 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 4097
=================================================================
Total params: 56,377,345
Trainable params: 56,377,345
Non-trainable params: 0
```

SVM classifier (computed but NOT TAKEN TO PREDICT IMAGE CLASS)

```
###PAMETERS
color_space = 'YCrCb' # Can be RGB, HSV, LUV, HLS, YUV, YCrCb
orient = 8  # HOG orientations
pix_per_cell = 8 # HOG pixels per cell
cell_per_block = 2 # HOG cells per block
hog_channel = -1 # Can be 0, 1, 2, or "ALL"
spatial_size = (16, 16) # Spatial binning dimensions
hist_bins = 32    # 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
y_start_stop = [None, None] # Min and max in y to search in slide_window()


############## DATA FITED TO CALSSIFIER SVM AND NN => HOG FEATURES
car_features = extract_features(cars, cspace=color_space, spatial_size=spatial_size,
                        hist_bins=hist_bins, hist_range=(0, 256), orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel)
notcar_features = extract_features(notcars, cspace=color_space, spatial_size=spatial_size,
                        hist_bins=hist_bins, hist_range=(0, 256), orient=orient,
                        pix_per_cell=pix_per_cell, cell_per_block=cell_per_block,
                        hog_channel=hog_channel)

X = np.vstack((car_features, notcar_features)).astype(np.float64)
# Fit a per-column scaler
X_scaler = StandardScaler().fit(X)
# Apply the scaler to X
scaled_X = X_scaler.transform(X)

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


print('spliting...')
# Split 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)

print('DONE...')
print('Using:',orient,'orientations',pix_per_cell,
    'pixels per cell and', cell_per_block,'cells per block')
print('Feature vector length:', len(X_train[0]))
print('Total training images:', len(X_train))
print('Total test images:', len(X_test))
############## SVM CLASSIFIER#################################################################
# SVM IS COMPUTED BUT NOT USED IN PROJECT PREDICTION => APPLIED ONLY AS REFERANCE
##############################################################################################
svc = LinearSVC()
# Check the training time for the SVC
t=time.time()
# Check the training time for the SVC
t=time.time()
print('training...')
svc.fit(X_train, y_train)
print('DONE...')

t2 = time.time()
print(round(t2-t, 2), 'Seconds to train SVC...')
# ACCURACY FOR THE SVC - TEST DATA
print('Test Accuracy of SVC = ', round(svc.score(X_test, y_test), 4))

t=time.time()
```

NN CLASSIFIER (predicting image classes)

```
learning_rate= 0.0001
MODEL_NAME = 'vehicle_detection-{}-{}.model'.format(learning_rate, '2conv-basic')
# SAME DATA AS FOR SVM USED TO TRAIN NN
x_train = X_train
y_train = y_train
x_test = X_test
y_test = y_test


#DEFINITION OF NN ARCHITECTURE
model = Sequential()
#
model.add(Dense(4096, input_dim=5568, activation='relu'))

model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# FIT THE DATA
model.compile (optimizer = Adam(learning_rate), loss = "mse", metrics=['accuracy'])
history = model.fit(x_train, y_train,
          epochs=5,
          batch_size=128,verbose = 1)
score = model.evaluate(x_test, y_test, batch_size=128)
```

5. Sliding windows

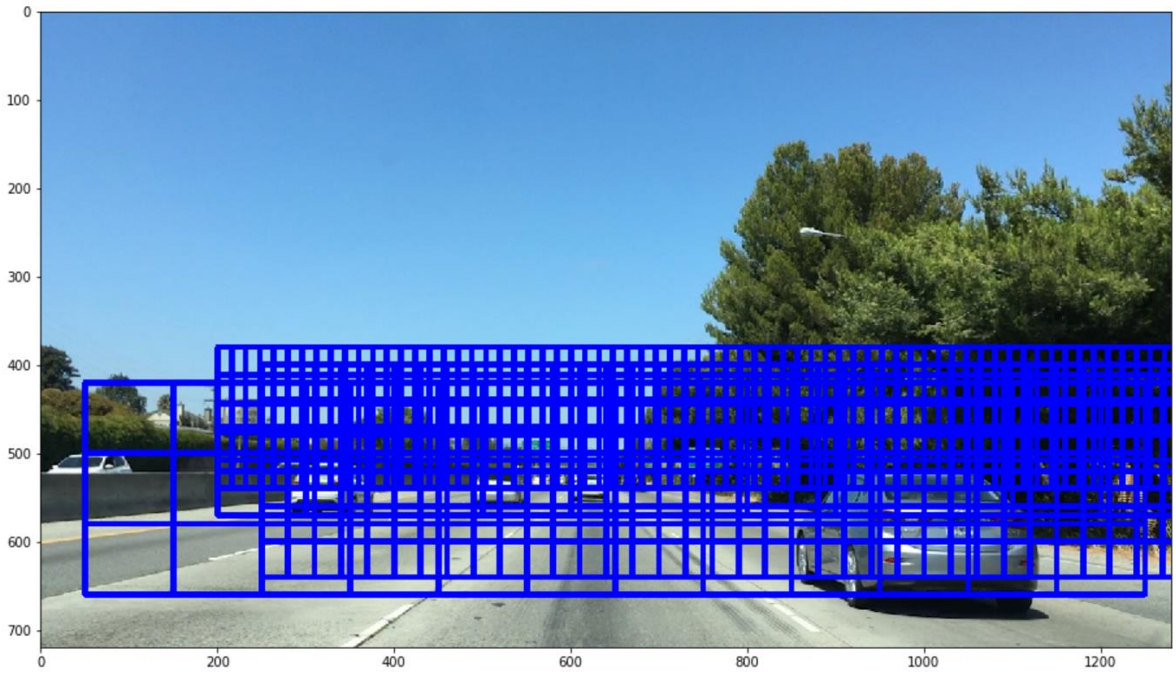
During the research it was noticed that the size and choice how the window is sliding plays predominant role in project performance. Applying correct size of window influences directly for car finding. The choice of search method influences on "speed" of the pipeline (how fast the image in processed). Here the optimization between number of windows to search and speed of the image processing had to be performed. The optimization was carried out during the iteration process. At the beginning of project the same size of sliding window was applied (across the image). First choice of project methodology (same size of sliding window) influenced that processing of each image took too many CPU cycles. In order to speed up whole process, the Udacity recommendation were applied.

- The image window sliding area was reduced by analyzing the area of possible vehicle occurrence (y: from 380 to 700; x: 50 to the size of coming image).

- Different size of sliding windows were applied (four different size) – depending directly on image perspective (four different perspective).

- The size of sliding image overlap was chosen by verifying the classifier output. For all project image perspectives there was applied 0.75, 0.5 overlap

Sliding windows applied in the project:

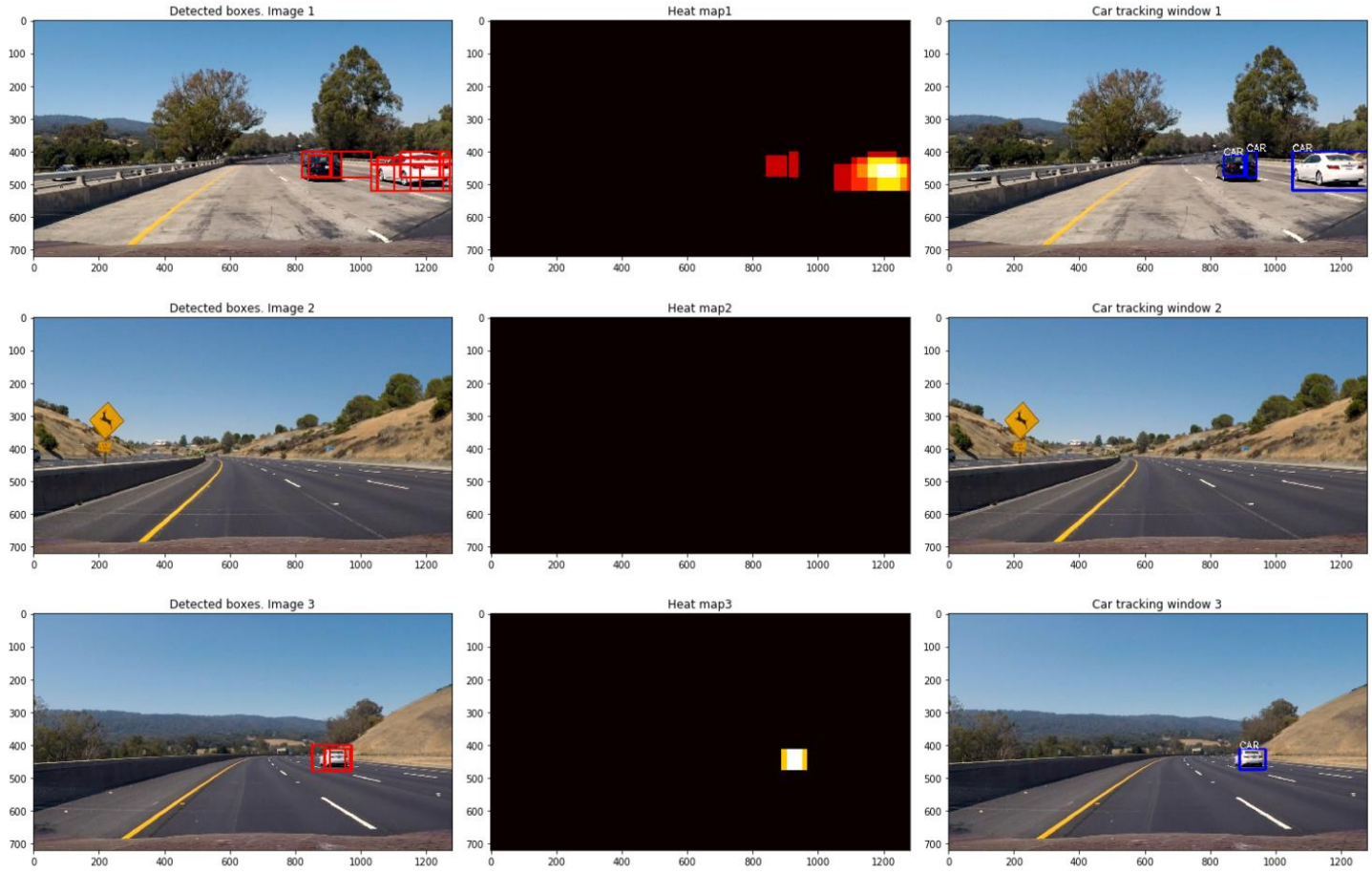


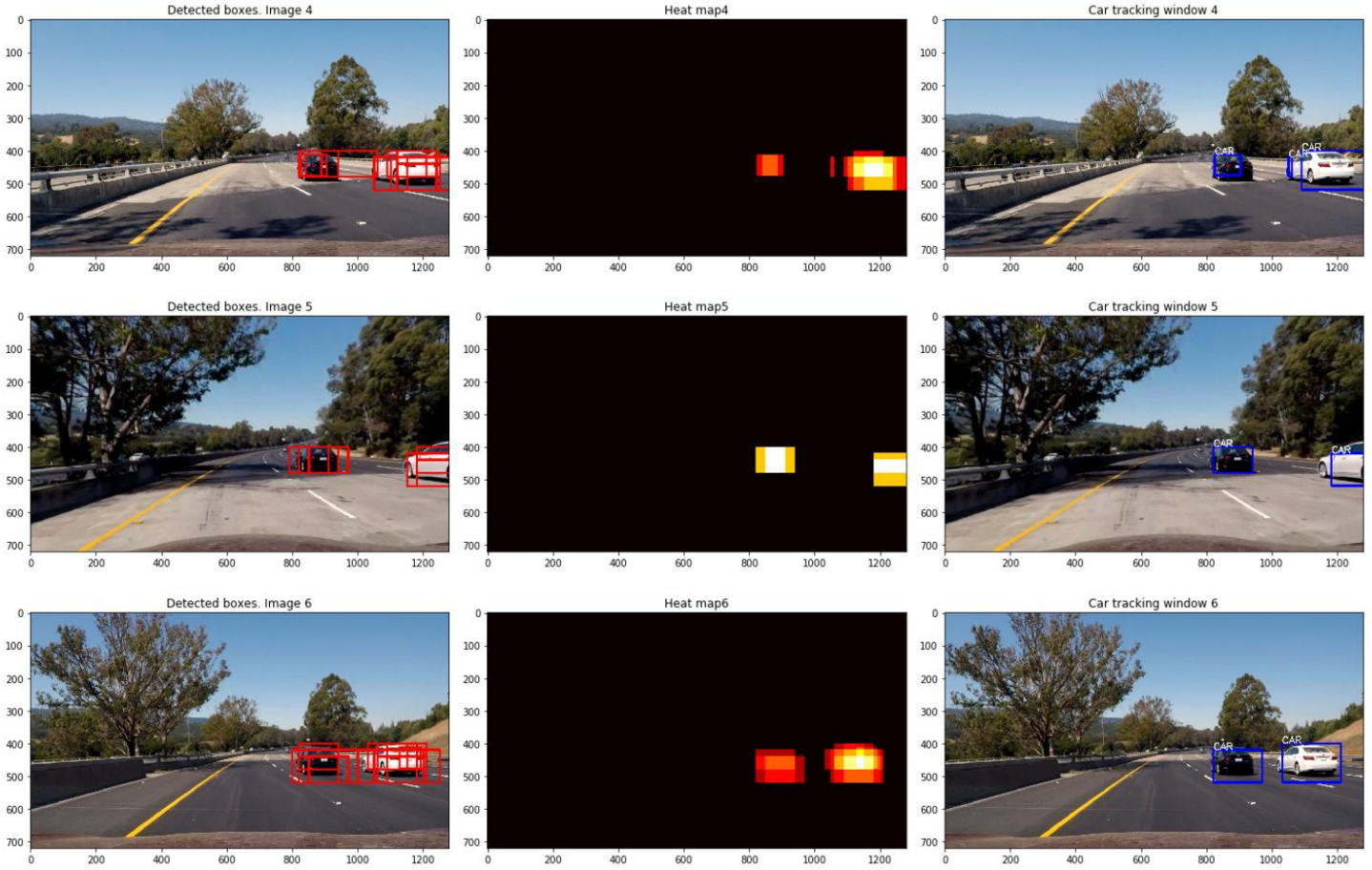

```
def scanning_windows(image, max_w = 10, inter_1 = 5, inter_2 = 5, min_w=3):

#DEFINE WINDOWS IN 4 PERSPECTIVES
scan_windows = []

#PERSPECTIVE MAX
windows_max = slide_window(image, x_start_stop=[200, None], y_start_stop=[380, 550], \
                    xy_window=(64,64), xy_overlap=(0.75, 0.5))
for i in range (max_w):

    scan_windows.append(windows_max[i])


#PERSPECTIVE INTERMEDIATE 1
windows_intermediate_1 = slide_window(image, x_start_stop=[250, None], y_start_stop=[400, 700], \
                    xy_window=(120,80), xy_overlap=(0.75, 0.5))
for i in range (inter_1):

    scan_windows.append(windows_intermediate_1[i])


#PERSPECTIVE INTERMEDIATE 2
windows_intermediate_2 = slide_window(image, x_start_stop=[200, None], y_start_stop=[420, 700], \
                    xy_window=(200,100), xy_overlap=(0.75, 0.5))
for i in range (inter_2):

    scan_windows.append(windows_intermediate_2[i])

#PERSPECTIVE MIN
windows_min = slide_window(image, x_start_stop=[50, None], y_start_stop=[420, 700], \
                    xy_window=(200,160), xy_overlap=(0.5, 0.5))
for i in range (min_w):

    scan_windows.append(windows_min[i])
```

6. False positives - Heat Maps

For all detected windows (where the prediction is True) the box is created (return from processing_image() function. In order to intensify the detected area (showing the locations of repeated detections) the pixel for each detected area is intensified (function add_heat()). Afterwards to make visible only one window (point out the exact position of detected vehicle, the heating_map() function is applied. Here, defined number of heat map are connected together (in the for loop) to make visible common part (yellow color). Finally, the threshold is applied (threshold is tuned), detected object is highlighted and other objects wrongly classified filtered. Thresholding is performed in apply_threshold() function – if the pixel value is below the threshold, so the pixel is dimmed (switch off).

Samples of box detection, heat map and tracking window.




Detected boxes. Image 4
Heat map4
Car tracking window 4
Detected boxes. Image 5
Heat map5
Car tracking window 5
Detected boxes. Image 6
Heat map6
Car tracking window 6
CAR

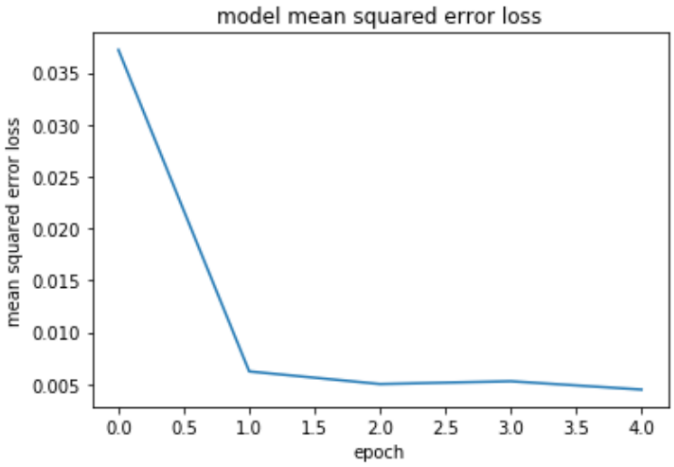
The Python code where the processing of heat map is performed:

```
def add_heat(heatmap, bbox_list):
    # Iterate through list of bboxes
    for box in bbox_list:
        # Add += 1 for all pixels inside each bbox
        # Assuming each "box" takes the form ((x1, y1), (x2, y2))
        heatmap[box[0][1]:box[1][1], box[0][0]:box[1][0]] += 1
    # Return updated heatmap
    return heatmap# Iterate through list of bboxes

def apply_threshold(heatmap, threshold):
    # Zero out pixels below the threshold
    heatmap[heatmap <= threshold] = 0
    # Return thresholded map
    return heatmap


def draw_labeled_bboxes(img, labels):
    # Iterate through all detected cars
    for car_number in range(1, labels[1]+1):
        # Find pixels with each car_number label value
        nonzero = (labels[0] == car_number).nonzero()
        # Identify x and y values of those pixels
        nonzeroy = np.array(nonzero[0])
        nonzerox = np.array(nonzero[1])
        # Define a bounding box based on min/max x and y
        bbox = ((np.min(nonzerox), np.min(nonzeroy)), (np.max(nonzerox), np.max(nonzeroy)))
        # Draw the box on the image
        cv2.rectangle(img, bbox[0], bbox[1], (0,0,255), 6)
        cv2.putText(img, 'CAR', bbox[0], cv2.FONT_HERSHEY_SIMPLEX,  1,(255,255,255),2)
    # Return the image
    return img

def heating_map(image, number_frames = 0, threshold=2):
    heat = np.zeros_like(image[:,:,0]).astype(np.float)


    for boxlist in image_bboxes[-number_frames:]: # how many frames taken to common map
        heat = add_heat(heat, boxlist)



    # Apply threshold to help remove false positives
    heat = apply_threshold(heat,threshold)

    # Visualize the heatmap when displaying
    heatmap = np.clip(heat, 0, 255)

    # Find final boxes from heatmap using label function
    labels = label(heatmap)

    draw_img = draw_labeled_bboxes(np.copy(image), labels)
    return draw_img, heatmap
```

7. Pipeline and classifier optimization

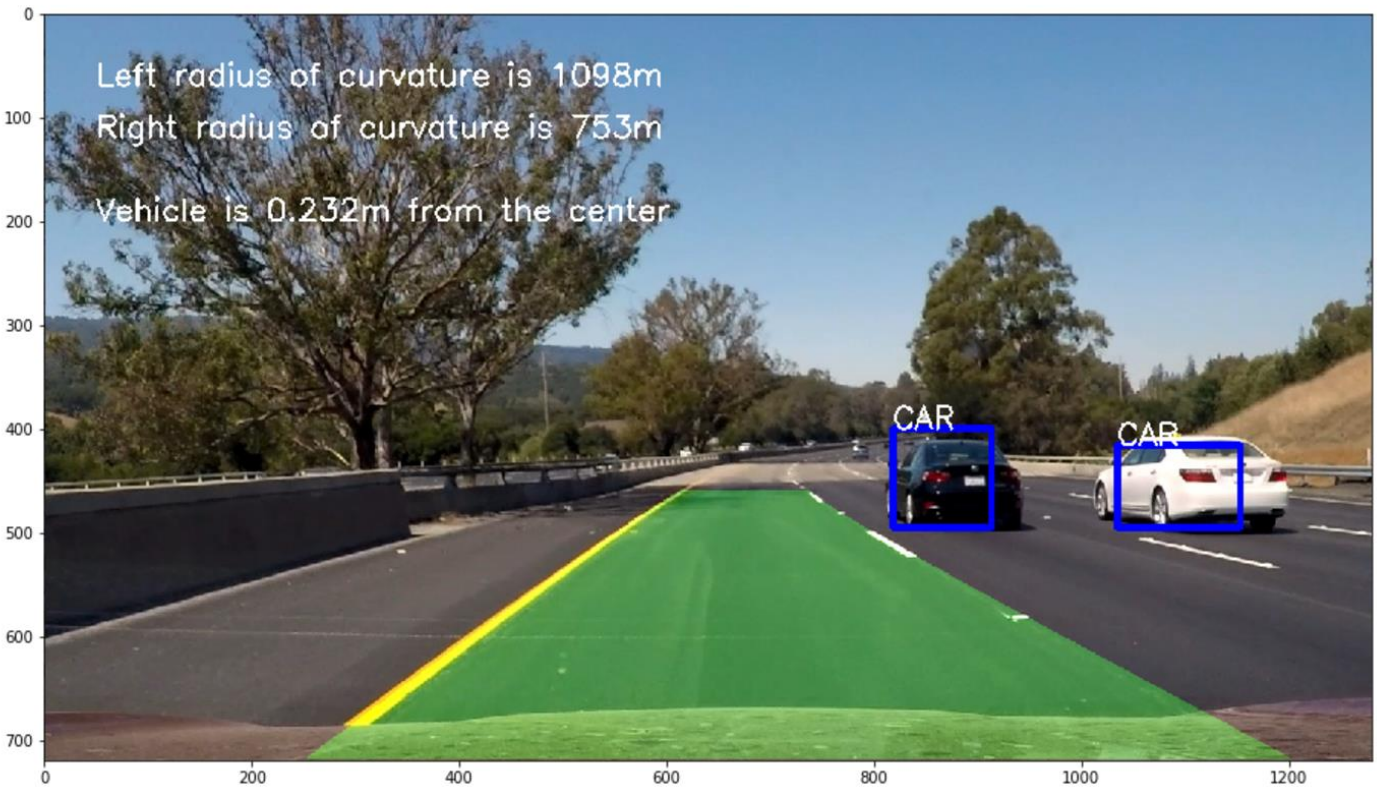
Pipeline detects both road lines and cars. Below it was depicted the test results who the pipeline works. Optimizer was adjusted during the iteration process (hyperparameters of neural network were tuned). Modification regarding size of the network, learning rate, type of optimizer, number of epoch and batch size were figured out during research, bearing in mind the accuracy of test data set. For NN classifier after 5 epochs is was achieved similar test result **(NN: 0.9948; SVM:0.987)** Hyperparameters of project approach can be summarize in following table:

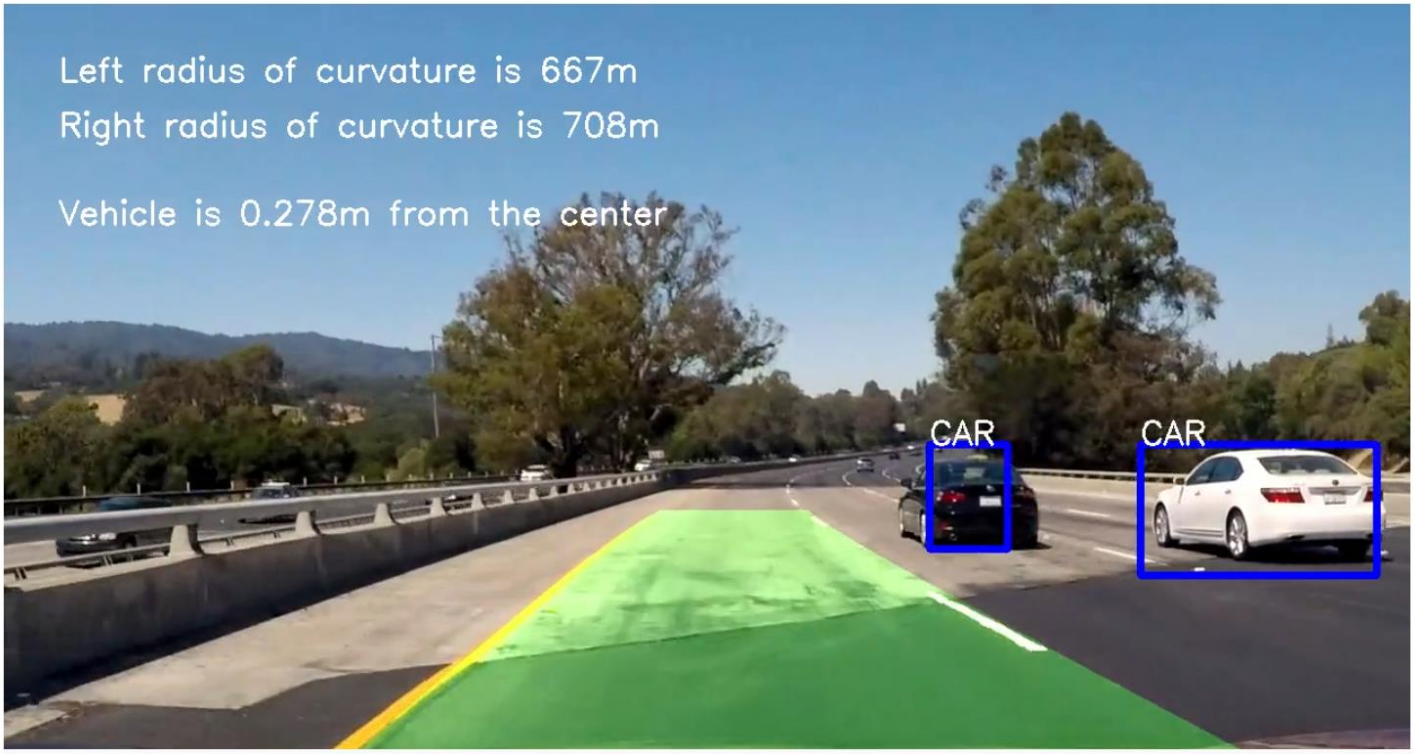
Parameter	Value
Learning rate	0.0001
Optimizer	Adam
Number of epoch	5
Batch size	128




```
Using: 8 orientations 8 pixels per cell and 2 cells per block
Feature vector length: 5568
Total training images: 14208
Total test images: 3552
training...
DONE...
14.19 Seconds to train SVC...
Test Accuracy of SVC =  0.987
```

Samples of pipeline performance:





8. Discussion

During the implementation of car detector the main problem which was faced was connected with finding correctly sliding windows size and methodology for winnows search (in order to reduce processing time and build properly heat map – detect the car). The problem with adjusting the proper threshold was also can be also classified as project issues. In addition, applying the classifier on neural network forced to investigate the values of hyperparameters and network architecture.

New solution approach can incorporated the classifier base on convolutional neural network in order to make the classifier more robust. Here as it was discussed in project 3, NVIDIA architecture or similar can be incorporated. Improvement can be also applied on windows searching methodology to make the pipeline more optimized (faster image processing and better filtering).