Assignment #03

Introduction to Image Analysis

and Machine Learning

Development of a car detector

based on a supervised learning approach.

Group M15

Emanuel Alogna (emal@itu.dk)

Peter Jan Mortimer (pjan@itu.dk)

**1. Introduction**

For our final report, we developed a car detector using a binary classifier. Our goal was to develop a car detector that can reliably detect cars in videos. Our application can be split up into two major components: the car classifier and the car detector.

In the car classifier component, we train a binary classifier based on a dataset of “car” and “non-car” images. In this assignment we experimented with different machine learning algorithms and compared their performance on this task.

In the car detector component, we use different image processing steps to prepare images in the test videos for our car classifier component. We used different techniques to improve the performance of the car detector to increase the detection speed of our application.

**2. Methods**

In this chapter, we introduce the two components of our application: the car classifier and the car detector.

**2.1. Car Classifier**

Our car classifier uses a binary classifier to determine, whether the given image resembles a car or not. There are different image preprocessing steps depending on the chosen classification algorithm.

**2.1.1. Dataset**

The original assignment dataset includes 5 videos of a highway. 375 64x64 images of cars on the highway were manually extracted. Frames from the videos that do not include any cars are used to generate 64x64 image patches as negative samples for the dataset.

We only want to determine whether a given image includes a car or not. Our data only contains the labels , where labels a non-car sample and a car sample.

**2.1.2. Data Augmentation**

We extended our dataset using different data augmentation approaches. To reliably classify car images from different perspectives and lighting conditions requires a larger dataset. This can be done without manually extending the car images, which could be done by manually annotating videos. The dataset is extended by using different image processing techniques on the existing samples.

We increased the number of positive samples by applying a random rotation of degrees to each car image and reflecting the image along its center y-axis (see Fig. 1). We believe this data augmentation step to be necessary to increase the number of samples to train more complex models, like convolutional neural networks.



Figure 1: Data Augmentation applied to one example car image. Image (a) show the original car. In image (b) the image is reflected along the center y-axis. In image (c) the image is randomly rotated and reshaped to the 64x64 input resolution.

**2.1.3. Histogram of oriented Gradients**

The histogram of oriented gradients (short: HOG) has proven to be a reliable descriptor to detect pedestrians in images. The HOG features are based on counts of the image gradient for local patches within the chosen image. For one 64x64 image sample we produce a HOG vector with 1764 features. To analyze the HOG features in full is hard due to the large size. We applied a data transformation technique to reduce the number of dimensions during our data analysis.

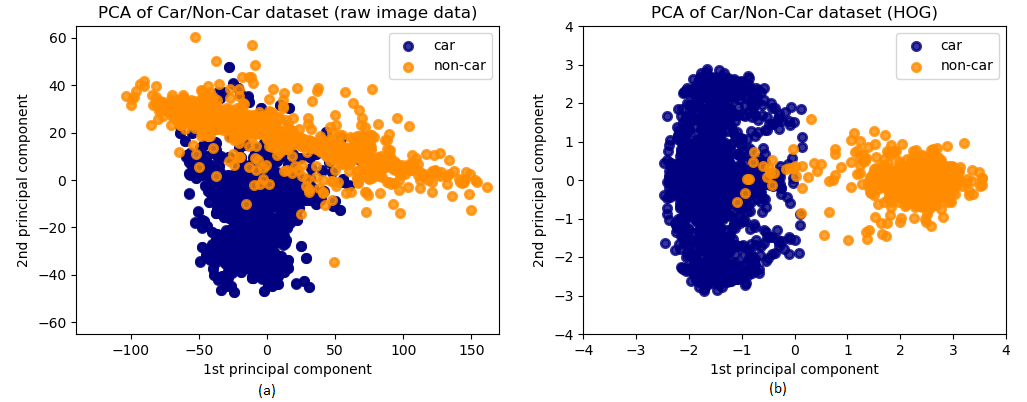
The effectiveness of the HOG features in separating between the two class labels car and non-car can be visualized using principal component analysis (short: PCA). PCA is a method to reduce the dimensionality of our dataset by applying a linear transformation to our dataset. The dataset is transformed to new axes, the principal components. Taking the first two principal components and plotting them can give us a good understanding of the basic structure of our data. In Fig. 2, we calculated the first two principal components of the raw grayscale image from each of our images and calculated the first two principal components of the HOG vector for each of our images. The HOG feature vectors show a better separation of our class labels in 2 dimensions than the raw image data does. This suggests that a model will calculate a better decision boundary for our classes using hog features than just the raw image information.

Figure 2: Here are the first two principal components plotted for the raw grayscale image data (a) and for the calculated HOG features (b). The HOG feature vectors show a greater separation between the image classes. This suggests that a classifier model will be better at determining a good decision boundary when learning using the hog features than the raw image data.

A closer analysis of our HOG feature vector plot shows an overlap for some non-car instances with the cluster of car instances (see Fig.3a). These overlapping non-car instances are images of the road surface marking or other street signs (see Fig.3b). These types of images we expected to resurface as possible false positives when evaluating our car classifier.

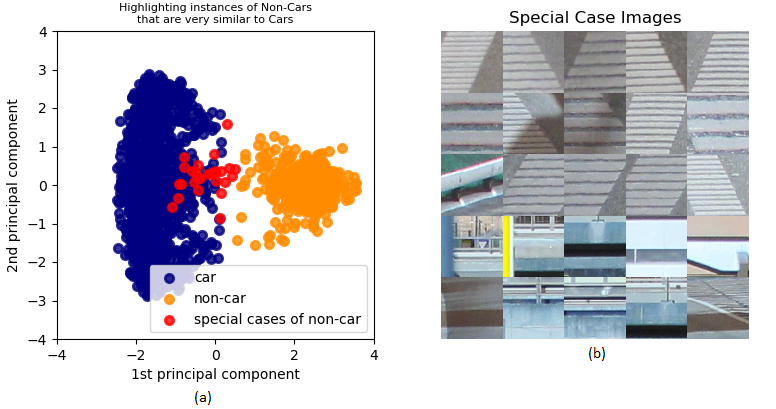


Figure 3: The points marked red in the plot of image (a) are non-car instances, that show similar HOG vector properties as car instances. In image (b) we display 25 of these special non-car instances and see that most of these images resemble road surface markings or street signs. We expect our classifiers to more likely detect false positives around road surface markings based on this observation.

**2.1.4. Support Vector Machine**

The first model we trained was a support vector machine (short: SVM) model based on the hog features of the images. The SVM model tries to find the decision boundary with the largest margin between both classes. We used a kernel-SVM with a radial basis function (short: rbf) kernel to allow our model to detect possible nonlinear decision boundaries. We consider this our approach of creating a car classifier as using a simple model, the SVM, and the complex preprocessing step, the HOG feature descriptor, to create a good classifier.

**2.1.5. Convolutional Neural Network**

[still have to write and visualize our model architecture here]

**2.2. Car Detector**

Our car detector component takes a frame from the given input video and uses different image processing techniques to determine regions of interest (short: ROI), where possible cars could be. These ROIs are then given to the car classifier component to classify whether a given image patch contains a car or not. In the following we will introduce the different techniques we used to create our performant car detector.

**2.2.1. Foreground Detector**

The first step of our car detector component is the foreground detector. All car videos include a static camera, which we can leverage since the road and other non-car objects will most likely not move during the recording. The only moving objects that make up our foreground are cars then. We apply a background mixture model, that models each pixel as a mixture of gaussians. If there are drastic intensity changes for a pixel, then it is classified as a foreground pixel. Apart from some noise, most larger areas of the detected foreground are cars (see Fig.4b). We then use a contour detector (as in assignment 2 for the pupil detector) to recognize the cars in the binary image. The bounding boxes of the detected contours are then passed as regions of interest to the next processing step (see Fig.4c).

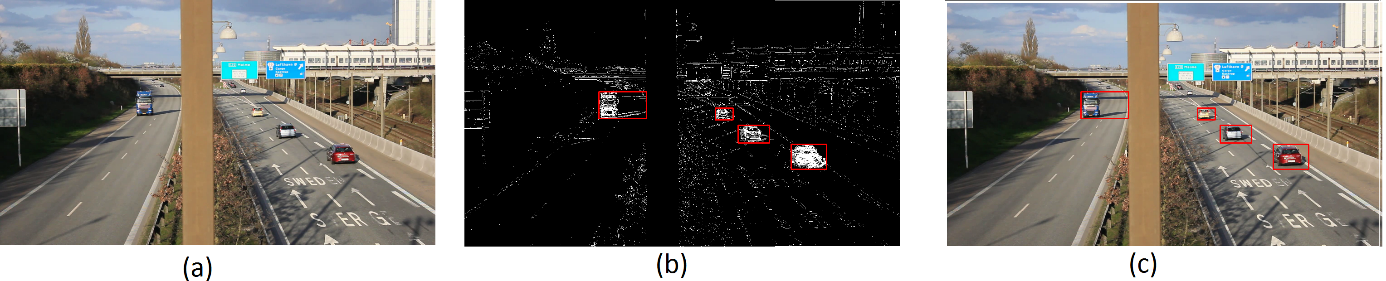


Figure 4: Here the foreground detector is applied to an example frame. The original image (a) is passed through our foreground detector. We then detect the contours in the binary image and take the bounding boxes of each contour, which are marked in red in image (b). We then take these regions of interest and pass them for further processing in the original image, as seen in image (c). The bounding boxes capture most cars, that are large enough in the frame and greatly reduce the area of that must be analyzed later in the car detector pipeline.

**2.2.2. Image Pyramids and Sliding Window**

The regions of interest passed by the foreground detector are then further processed for the car classifier. Both our SVM model and our CNN model require images with the dimension 64x64 pixels as input. We apply image pyramids and sliding windows to find different 64x64 image patches for our car classifier within each region of interest. For the image pyramid we downscale the regions of interest to allow us to find 64x64 image patches that include the full car in cases where the car is too large in the initially detected image frame (see Fig.5). The sliding window uses a windows size of 64x64 pixels to match with the classifier input requirements. The sliding window moves across the given regions of interest at different scales of the image pyramid to find many fitting input images for our car classifier. The sliding window also allows our car detector to correctly find multiple cars within one large region of interest that includes multiple cars.



Figure 5: Here are two example regions of interest. The red rectangle shows the size of a 64x64 pixel window for the sliding windows. In both image (a) and image (b) the initial region of interest is too large to fully fit the car within the window. After downscaling the image (by a factor of 1.5) the window can easily cover the full car, allowing for a higher probability of a successful classification.

**2.2.3. Overlapping Bounding Box**

Our car classifier usually detects multiple instances of the same car within the region of interest when applying the sliding window. This leads to many overlapping bounding boxes in the same area around a car (see Fig.6a). This is a common issue in object detection algorithms and a solution to this is non-maximum suppression[[1]](#footnote-1) (short: NMS). In NMS we compare the bounding boxes and remove bounding boxes that overlap a lot with other bounding boxes. NMS performs very well for our car detector and reduces the number of overlapping bounding boxes, without discarding previously detected cars (see Fig.6b).

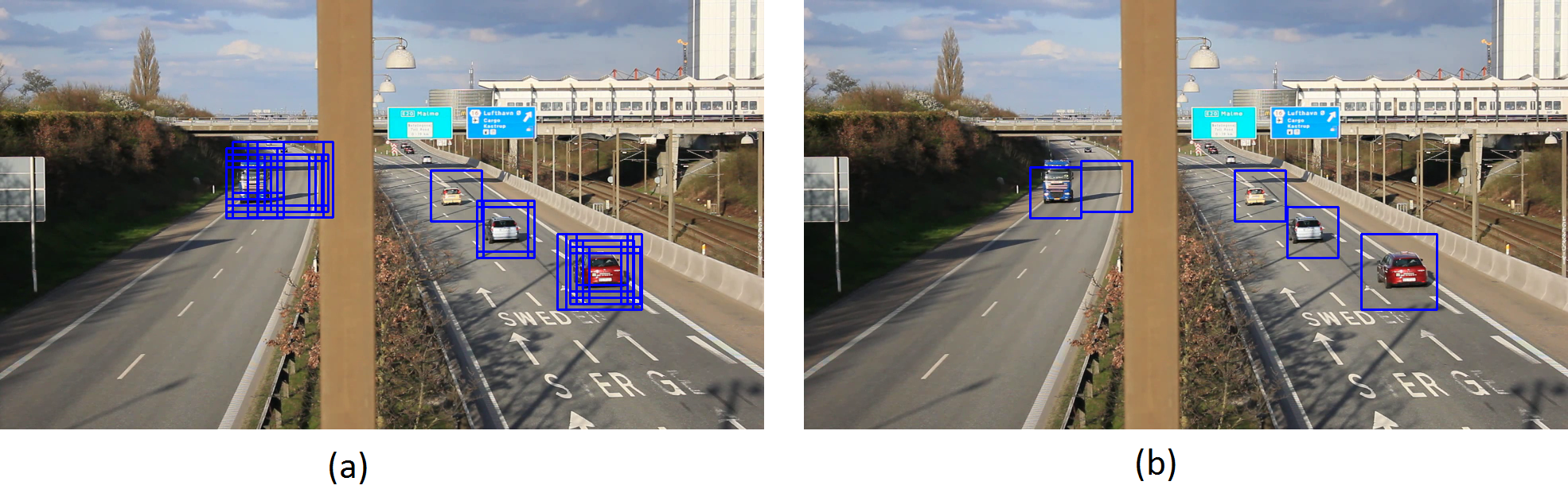


Figure 6: The NMS significantly reduces the number of detected image patches (marked in blue). In image (b) the final car detections are displayed. Unfortunately, the NMS algorithm cannot remove bounding boxes that do not overlap, but do belong to the same car object. This is the case for large vehicles like the truck, which is detected twice in image (b).

**(2.2.4. Predicting Car Destination)**

**3. Results**

We created a classifier both using a SVM model on HOG features and a CNN model on the raw image data. In the following we compared their performance on a separate test set and then compare their performance in the context of a live video in the car detector component.

**3.1. Model Comparison** (SVM vs. CNN)

Our complete dataset used for both models contained 2500 images. Of the 2500 images, 1500 images belonged to the positive car sample set. The other 1000 images belonged to the negative non-car sample set. The dataset has a training/validation/test set split of 85%/15%/15%. The validation set was used for hyperparameter tuning. The test set was used for verifying the overall model performance (see Tab.1).

|  |  |  |
| --- | --- | --- |
|  | SVM (HOG Features) | CNN (Image Data) |
| Test Set Size | 375 | 375 |
| Test Set Accuracy | 99.7% | 98.9% |
| False Positive Rate | 0.4% | 1.7% |
| Confusion Matrix  [non-car|car] |  |  |

Table 1: We compare the performance of the SVM model with HOG features and the CNN model using the raw image data. We measure the performance on their respective prediction results on the test data set (15% of the complete data set). Both models perform very well on our test dataset. All car images are correctly classified, leading to no false negatives for both models. Some non-car images are incorrectly classified as cars, which is reflected in the false positive rate. Overall, both models perform very well and the difference between them are minimal. It cannot be said that one model significantly outperforms the other.

**3.2. Car Detector Performance**

The car detector can reliably detect each car in the frame. The processing time of our car detector does not allow for smooth real-time car detection (see Table 2). There are however certain scenarios, where the car detector performs better than in others.

|  |  |  |
| --- | --- | --- |
| Video file | Processing time per frame  [in seconds] | Difference from real time processing baseline (33ms/frame) |
| Cars\_01.mov | 0.253 | -0.22 |
| Cars\_02.mov | 0.340 | -0.307 |
| Cars\_03.mov | 0.828 | -0.795 |
| Cars\_04.mov | 0.741 | -0.708 |
| Cars\_05.mov | 0.231 | -0.198 |

Table 2: We do a very simple comparison of the average time (wall clock time) our car detector needs to process one frame (using the SVM model as classifier). The time per frame for each video was measured by taking the average processing time for one frame over a 30 second video recording at 30 frames per second (900 frames processed in total). In the second column we compare our average processing times to the optimistic baseline of processing each frame at real time. The baseline for a video recorded at 30 frames per second is 0.033 seconds of processing time for one frame. Our car detector model cannot detect cars in real time, but we can see performance differences depending on the video perspective.

For frames that are have a smaller viewing angle towards the road, the car detector detects multiple non-overlapping car image patches for the same car if the car is close to the camera and relatively large in the overall frame. The region of interest is large in comparison to cars in the distance and the classifier must process more image patches, which can lead to a slower performance (see Fig.7a). This is both the case in the videos Cars\_03.mov and Cars\_04.mov, where the car detector performs worst (see Table 2).

In the two videos with the lowest processing performance (marked red in Table 2), the camera is more subject to shaking due to wind or the vibrations caused by passing cars. This leads the foreground detector to detect contours at seemingly arbitrary image position, which leads to many regions of interest, that must be processed, but do not contain any cars (see Fig.7b). This also slows the performance of the car detector and leads to many false positive detections.

The videos where the car detector performs best (marked green in Table 2) tend to be recorded from a higher viewing angle towards the road, resulting to relatively small car images in the frame, regardless how close the car is to the camera (see Fig.7c). The car detector performs better in these types of videos.

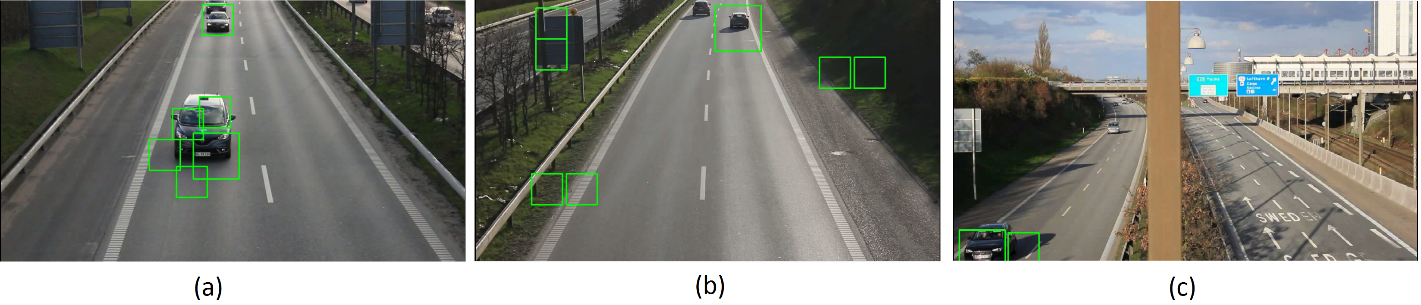


Figure 7: Three video frames, that show different scenarios, that occur depending on the video's viewing angle towards the road. Image (a) is from Cars\_03.mov and shows the multiple, non-overlapping detections made on cars, that appear larger in the frame. We do not downscale the region of interest enough to correctly detect the car fully in an image patch. Image (b) is from Cars\_04.mov and shows that when the camera is shaking, the foreground detector finds many regions of interest. Most of these regions of interest do not include a car and lead to false positive detections. Image (c) is from Cars\_05.mov, where our car detector performs well. Here the viewing angle towards the road does not allow for cars to appear too large in the frame. The car that is very close to the camera in image (c) can still be correctly detected.

**4. Discussion**

The model trained for the car classifier have shown a good performance, but there are still different areas of improvement for both the car classifier component and the car detector component.

**4.1. Improving the Car Classifier**

Even though we got good results on the test set, the classifiers have shown to occasionally detect false positives around street surface markings and street signs (see Fig.7b). This also confirms with our analysis, that among the hog features, these type of non-car images do appear to be similar to car images (see Fig.3). We believe, that searching our negative images for false positives and to add these false positives as hard examples for a second round of training, like in Dalal and Triggs’ paper[[2]](#footnote-2) for the pedestrian detector, could increase the performance of both the SVM and the CNN model.

**4.2. Improving the Car Detector**

There are different areas of improvement for the car detector component. The foreground detector in the car detector component requires a few frames as start up time to determine the current background of the image. The car detector won’t detect any regions of interest in the first frames of the video. Maybe a simpler model to detect the foreground could be used in the first few frames of a video.

The image pyramids parameters should also be adjusted based on the viewing angle of the camera towards the road. If the images were downscaled more in scenarios, where cars can appear very large in the frame, then the car detector could find an image patch that includes the full car for the car classifier. There is a trade off, since increasing the depth of the image pyramid does increase the processing time of the car detector.

**5. Conclusions**

Our goal was to create a reliable car detector based on a supervised learning algorithm. Furthermore, we wanted to compare the use of a simpler machine learning algorithm trained with a more informative feature set with a more complex machine learning algorithm trained on a simpler set of data. In our report, these were the SVM model using HOG features and the CNN model using the raw image data, respectively.

Our results have shown that both models perform well on the car dataset (see Section 3.1.). The HOG features are a good descriptor to distinguish between car and non-car images, allowing the SVM model to easily learn a good decision boundary. The raw image data between car and non-car images is also very different, so that cars do stand out in the image patches used for learning, allowing the CNN model to learn a good classification model after a few epochs of learning.

An image detection problem that would include more subtle differences between the positive and negative images, like detecting a specific car brand among car images, would probably lead to different results. We believe the trade off between good feature engineering and simple learning algorithm or no feature engineering and a complex learning algorithm would be more visible.

1. <https://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-detection-python/> [↑](#footnote-ref-1)
2. [„Histogram of Oriented Gradients for Human Detection“ from N. Dalal and B. Triggs (2005).](https://lear.inrialpes.fr/people/triggs/pubs/Dalal-cvpr05.pdf) [↑](#footnote-ref-2)