VIC Kaggle Challenge : Pipeline overview
- Report -

https://github.com/Anouar7768/VIC-Car-detection

Anouar Oussalah

CentraleSupélec, Université Paris-Saclay

Our approach for the car detection Kaggle challenge involved using machine learning. Specifically, we chose to extract relevant features from the training dataset and subsequently train a machine learning model for the classification task. Further details regarding our methodology will be expounded upon below.

1 Features

1.1 Histogram of Oriented Gradients (HOG) features

HOG is a popular feature descriptor in computer vision for object detection, including car detection. It captures local changes in intensity and direction by analyzing image gradients. HOG divides the image into small cells, computes a histogram of gradient orientations within each cell, and combines neighboring cells to form larger blocks. This generates a feature vector that is highly discriminative for object detection tasks. HOG is particularly useful for car detection due to its ability to handle changes in lighting conditions and distinguish cars from other objects in cluttered scenes. Additionally, HOG features can be efficiently computed using convolution operations, making them well-suited for real-time applications.

We chose HOG features for our car detection project and used `skimage.feature` library for computation, with manual hyperparameter tuning.

We used the library skimage to compute HOG features, and we tuned the hyperparameters manually.

1.2 Features extraction

To train our model, we needed both positive samples (samples of cars) and negative samples (samples of non-car objects). To ensure that our classifier receives features of consistent size, we decided to use two fixed-sized frames:

- 64*64 frames, which can capture smaller details of the cars, particularly those in the background.
- 128*128 frames, which are better suited for cars in the foreground and closer to the camera.

These two frame sizes were chosen based on their ability to extract the necessary details from both foreground and background cars.

Positive samples : To ensure consistency in our training data, we used the `cv2.resize` library to resize our car frames. We only included frames larger than 64*64, as smaller sizes can made it difficult to extract relevant information. Additionally, the test set consisted of high-resolution images, so using low-resolution images in training could introduce unrelevant information to our model. For frames with resolution between 64*64 and 128*128, we resized them to 64*64 frames, while frames larger than 128*128 were resized to 128*128 frames.

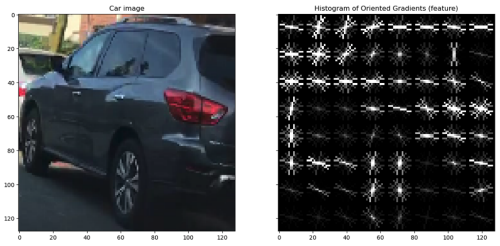

Fig. 1. Example of a positive sample, and its HOG feature

Negative samples : To sample non-car frames for both 64*64 and 128*128 resolutions, we randomly selected frames that did not overlap with any car frames.

We also added an external dataset of positive and negative samples, for the 64*64 frames (see Kaggle dataset).

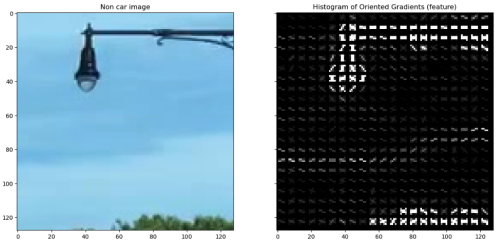

Fig. 2. Example of a negative sample, and its HOG feature

2 Model selection

To select a suitable model, we first used Naive Bayes as our baseline model due to its fast training and prediction times. However, after conducting further research, we discovered that SVMs are effective for object detection tasks. We tested SVMs and obtained surprisingly high accuracies for both 128*128 and 64*64 frames (see Table 1). The SVM model outperformed the Naive Bayes model, so we used it for our task.

Model	Time to train	f1-score
baseline	1 s	75%
SVM	125 s	96%

Table 1. Comparison of performance of the SVM and baseline model for 64*64 frames

3 Model training

At the end of our data labeling process, we obtained approximately

- 9500 labeled features for the 128*128 model, consisting of 4000 positive samples and 5500 negative samples
- for the 64*64 model, 20000 labeled features, with an equal distribution of 50% negative and 50% positive samples.

.

Using these labeled features, we trained two models: one for the 64*64 frames and the other for the 128*128 frames. Both models were trained in under 5 minutes.

4 Classification of the test set

In order to detect cars on the test images using our trained models, we employed a sliding window approach. For each image, we scanned it using windows of size 64*64 or 128*128 (depending on the model being used), and we moved the window with a sliding step of 16 or 32 (depending on the window size). For each window, we extracted the corresponding HOG features and fed them into the appropriate SVM model (either the 64*64 or the 128*128 model) to make a prediction of whether the window contains a car or not. In addition, we added a confidence threshold to eliminate uncertain predictions by computing the distance between the boundary and the extracted features. This helped to improve the accuracy of the predictions and reduce false positives.

The computational performance of the models was as follows:

- The 128*128 model took around 10 minutes to classify the test
- The 64*64 model took approximately 1 hour to classify the test set

Overall, the whole pipeline (feature extraction, training, and classifcation) took less than 1 hour and 20 minutes to run, considering that it was executed locally.

4.1 Post-processing

The model output resulted in many redundant windows, which can be observed in Figure 3.

Fig. 3. Example of an output of the model

We attempted to use the min-max suppression technique to obtain a single bounding box for each car but found that it did not improve the accuracy of the prediction. Therefore, we decided to use the raw results provided by the model.