Marie Guertin (260870552)
Luka Loignon (260871296)
Christian Martel (260867191)
Arian Omidi (260835976)
Dina Shoham (260823582)

Detecting, Localizing, and Tracking Vehicles
Group 7

ECSE 415 – Introduction to Computer Vision
Presented to Prof. Tal Arbel
Department of Electrical and Computer Engineering

Monday, April 11th, 2022
McGill University

Abstract – *Vehicle detection, localization, and tracking has many meaningful applications in intelligent transportation, traffic management, autonomous vehicles, and more. In this paper, a complete pipeline for this process is outlined, including alternate methods for each stage in the pipeline and an analysis of their effectiveness.*

1. Introduction

For this project, a complete pipeline for the detection, localization, and tracking of cars was designed and implemented. First, the dataset of vehicle and non-vehicle images was established using both provided samples and an additional dataset, and tuned by varying a number of parameters. Next, an SVM classifier was trained to extract features and ultimately determine whether or not an image contains a vehicle. The classifier was later used in the detection process to classify the contents of windows generated by a number of different implementations of the sliding window method. Finally, the detected vehicles were tracked across frames of videos to complete the pipeline.

2. Dataset

2.1. Dataset Composition

The classification dataset is composed of:

a) Generated vehicle samples from the provided KITTI training video sequence 0000, 0001 and 0002 provided ground truth bounding boxes. These images were transformed with small random scaling and translations to make the model more robust to outliers and non-perfect positive samples.

b) Generated non-vehicle samples from the provided KITTI training video sequences. These were extracted using the sliding window algorithms from the detection/localization section to add samples that were going to match the patches captured by the sliding window. The *minimum_intersection_ratio* parameter is used to define the maximal intersection threshold that a patch can have with a ground truth vehicle bounding box to be considered a negative sample.

c) Vehicle and non-vehicle samples from the Audacity dataset which has more than 8000 vehicle and non-vehicle images[1].

In **Fig. 1**, 5 extracted non-vehicle and vehicle samples from the video sequences are shown. It is observed that some of the samples contain vehicles but are labeled as non-vehicle since the vehicle portion of the image is below the minimal threshold.

Fig. 1: Vehicle and non-vehicle samples extracted from the KITTI video sequences.

In **Fig. 2**, 5 extracted non-vehicle and vehicle samples from the Udacity dataset are shown. All samples from this dataset have the same 64 x 64 size.

Fig. 2: Vehicles and non-vehicles samples from Udacity dataset [1]

2.2 Parameters

We defined three dataset parameters to tune our classification dataset composition:

a) **Positive-to-Negative Ratio:** The number of vehicle samples per non-vehicle sample. For instance, a ratio of 0.5 means that for each vehicle sample, we have 2 non-vehicles samples. We ran cross-validation on randomly generated datasets with increasing positive-to-negative ratios. We observed that increasing the positive-to-negative ratio tends to increase the overall recall and decrease the overall precision and vice-versa. Our detection algorithms were having trouble with false positives. To minimize the number of false positives, we decided to use a positive-to-negative ratio of 0.8 which allowed to achieve respectively high recall and very high precision.

a) **The minimum intersection threshold:** It is the minimum threshold for a generated patch in relation to a positive bounding box to be considered a vehicle. As the threshold is increased, we get larger vehicle parts in our non-vehicle samples. From plot observations, we determined that a good vehicleénon-vehicle distinction was achieved with a ratio of 0.75.

b) **The total number of training samples:** It sets the total number of vehicle/non-vehicle samples to extract from our dataset. The positive-to-negative ratio found in a) is used to guide the extraction. It is observed that, as more samples are added, the accuracy increases because the model takes into consideration more outliers or in other words, its variance increases and its bias decreases. However, the inference latency and training time increases linearly as more samples are added to the dataset.

2.3 Statistics (Refer to Appendix A)

The class distribution of the images extracted from the video sequences using a positive-to-negative ratio of 0.8 and extracting 1000 samples per sequence can be observed in **Fig. 1** of **Appendix A**. The width and height distribution of the samples can be seen in **Fig. 2**. The aspect ratio of the images can be seen in **Fig. 3**. The main takeaway is that images usually have larger width than height (aspect ratio between 0 and 1) as vehicles are.

3. Classification - SVM

We first trained a SVM classifier in order to achieve the task of detecting if an input image is a vehicle or not. The inputs of the classifier are feature vectors extracted from the RGB images provided by the dataset module. The output are prediction labels for each feature vector. '0' for non-vehicles and '1' for vehicles.

To obtain an accurate classifier, we must first preprocess the images obtained from the dataset module to extract the feature vectors using tuned preprocessing parameters. Then, we must train a classifier using the generated feature vectors and corresponding labels.

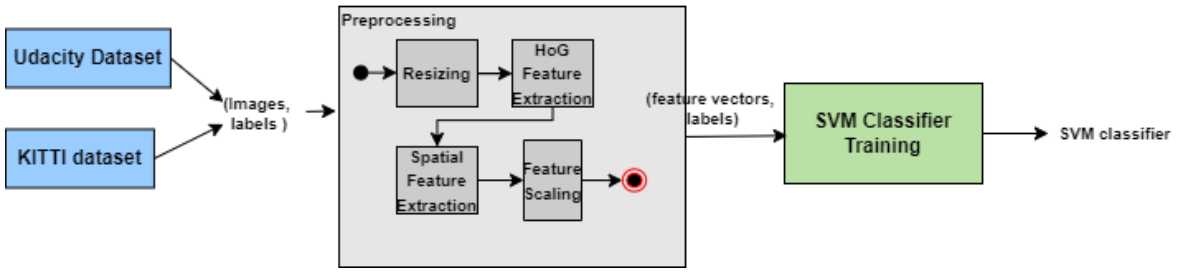

Fig. 3: Overview of the SVM classifier training pipeline

3.1 Preprocessing

From the image samples extracted from our dataset, we perform a series of preprocessing steps to generate the feature vector to input in our SVM classifier.

1. **Grayscaling (optional)**: Convert all samples to grayscale to reduce input size and increase preprocessing latency. Since colors are not taken into account to compute the HoG features, we assumed it would not affect the detection performance

2. **Resizing:** Resize the image samples to a fixed input shape to get a HoG feature vector of the same length for all images.

3. **HoG feature extraction**: We decided to use HoG features to train our SVM linear classifier since HoG features are widely used in computer vision for object detection tasks. HoG features are great to extract the general structure of an object from the magnitude and orientation of the gradient. To extract the HoG feature vector, first the pixels are grouped into cells and cells are grouped into blocks. For each cell in each block, the gradient histogram is computed according to the number of orientation bins.

   As observed in **Fig. 7** and **Table 1**, as we increase the number of pixels per cell or decrease the number of cells per block, the feature vector length decreases significantly and the object shape definition decreases. Having too few pixels per cell or too many cells per block could result in a model too complex which risks overfit our training data. On the other hand, having too many pixels per cell or too few cells per block could result in a simple model with high bias.

   Hence, we chose an in-between parameter combination of (2 x 2) cells per block and (4,4) pixels per cell leading to a HoG vector with 34 596 features.

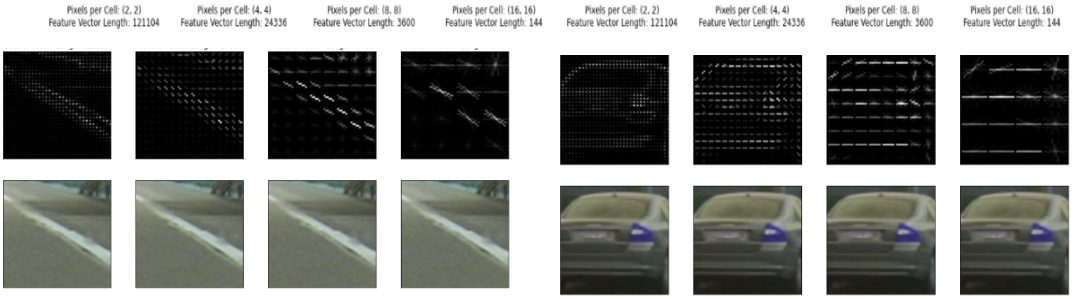

Fig. 7: Effect of varying the number of pixels per cell for a fixed (4,4) cells per block

Table 1: Vector Length for various cells per block used a fixed (4,4) pixels per cell

Number of Cells Per Block	Feature Vector Length
(2,2)	34 596
(4,4)	121 104
(8,8)	360 000
(16,16)	665 856

4. **Spatial feature extraction (optional)**: Take into account the image pixel intensity through space. The image is first resized into spatial bins. The spatial bins values are then flattened as a 1D vector and appended to the HoG feature vector. For example, appending a (16 x16) spatial bins vector would add 768 additional features to the vector. The number of spatial bins must be established in function of the HoG feature vector length and in function of the weight we want to accord to spatial features.

5. **Feature scaling (optional)**: The feature vector is scaled on a 0-1 range in order to ensure that each feature has the same weight.

3.1 Cross-Validation

The classifiers are trained and evaluated using a 3-fold cross validation using the provided KITTI video sequences 0000, 0001, 0002. For each fold, a sequence is chosen as the validation set and the rest of sequences are used as the train set. This way, for each fold, we ensure that our classifier performs detection on unseen data since the validation set images come from a video sequence that was not used for training.

In addition, the samples from external datasets are added to the train set to increase the model variance. The detection performance metrics: accuracy, recall and precision provided in the following experiments correspond to the overall mean and standard deviation on the 3 folds.

3.2 Hyper-Parameters Tuning Experiments (Refer to Appendix B)

Multiple experiments were performed in order to tune the dataset, preprocessing and training hyper-parameters. These experiments were performed using a random dataset generated using the following parameters: A minimum intersection ratio of 0.75, 1800 samples per KITTI video sequence, and 1800 samples from Udacity dataset [1].

1. *Classifier Types*

   The results of running cross-validation for different classifier types can be observed in **Table 1** of **Appendix B**. The SVM classifier with the RBF kernel was chosen since it achieves the best overall detection performance. This is due to the fact that RBF can solve non-linearly separable problems contrary to linear SVM. With the right tuning, RBF SVM should at least perform as good as linear SVM. However, it has the slowest inference latency of the classifiers tested which can be an issue during real-time detection.

2. *Hyper-tuning of C and gamma regularization parameters for RBF SVM classifier*

   Grid-search hyper-tuning was performed to find the best combination of C and gamma regularization parameters for the RBF SVM. With a sufficiently high C value, the RBF classifier could perfectly classify all training samples which would result in poor performance due to overfitting. On the other hand, a too low C value would have poor performance due to underfitting since it would not penalize enough wrongly classified training samples. Hence, it is important to tune the C parameter to find the one resulting in the best cross-validation detection performance metrics.
   After tuning, we ended up choosing a gamma parameter of 0.00001 and a C parameter of 5. An example of C parameter tuning can be observed in **Fig. 1** of **Appendix B**.

3. *Effect of 0-1 Normalization on Detection Performance*

   It is observed in **Table 2** of **Appendix B** that applying normalization to our vector helped to significantly increase the overall recall and accuracy of the classifier. Hence, this preprocessing parameter was kept enabled in the preprocessing pipeline.

4. *Effect of Gray Scaling on Detection Performance*

   It is observed in **Table 3** from **Appendix B** that applying gray scaling to our vector does not improve the performance of the HoG SVM classifier. This might be due to the fact that there exists a correlation between color intensity in space and vehicle or non-vehicle samples. Hence, this parameter was disabled from the preprocessing pipeline.

5. *Effect of Spatial Feature Extraction on Detection Performance*

   It is observed in **Table 4** from **Appendix B** that spatial feature extraction helped to improve the overall detection performance. Hence, it seems like there exists a correlation between color intensity in space and vehicle/non-vehicle detection. The best overall detection performance is obtained by extracting (16,16) spatial bins.

6. *Adding More Vehicle Samples to the Dataset*

   The video sequences from the KITTI dataset (the major part of our training set) do not contain a lot of vehicles. Even though random patches are generated around the vehicle ground truth bounding boxes, the small number of vehicles in that dataset does not contain high vehicle variance. Hence, this is why the model infers a high rate of false negatives (lower recall). On the other hand, the precision is high since the KITTI dataset contains a high variance of non-vehicle images. A solution to combat this problem is to add more vehicle samples from the Udacity dataset to the train set to compensate for the low variance of vehicle images in the KITTI dataset.

   The effect of increasing the number of vehicle samples in the dataset can be observed in **Fig. 2** of **Appendix B**. Since more vehicle samples are added, the overall recall metric increases. However, the overall precision metric decreases since adding more vehicles in the training increases the rate of false positives.

3.3 Evaluation

Upon completion of parameter hyper-tuning, we trained a classifier using the best combination of parameters we found. Fold 1, 2, and 3

Table 2: Accuracy performance of optimal SVM classifier on validation and test sets.

Accuracy (%)					
	Fold 1	Fold 2	Fold 3	Overall	Test
RBF SVM (C = 5,Gamma =10^{-5})	90.32	89.65	85.51	88.49 ± 2.13	93.44

Table 3: Recall performance of optimal SVM classifier on validation and test sets.

Recall (%)					
	Fold 1	Fold 2	Fold 3	Overall	Test
RBF SVM (C = 5, ,Gamma = 10^{-5})	87.19	85.78	80.07	84.35 ± 3.07	90.44

Table 4: Precision performance of optimal SVM classifier on validation and test sets.

Precision (%)					
	Fold 1	Fold 2	Fold 3	Overall	Test
RBF SVM (C = 5,Gamma = 10^{-5})	91.95	91.83	88.24	90.67 ± 1.72	95.47

In **Tables 2,3,4,** the detection performance metrics obtained during cross-validation are observed. One take-away is that the overall precision (90.67±1.72%) is significantly higher than the overall recall (84.37±3.07). As explained previously, this is due to the low vehicle samples variance in our dataset. A solution would be to add much more vehicle samples of different shapes, angles, different occlusions. This would help increase our recall, but would necessarily affect our precision.

Finally, in **Appendix C**, the results of successful and unsuccessful predictions on the test set can be observed for both the non-vehicle and vehicle classes.

4. Localization and Detection

For detection, we implemented the sliding window method, where we would run the Linear SVM classifier trained in Section 3 on sampled windows to determine whether or not the window contains a vehicle. Furthermore, we localize the vehicles by removing redundant predictions and merging the remaining bounding boxes. We tested our detection and localization on a dataset of three 1242x375 video sequences, taken using a car-mounted camera while driving within various urban environments. Furthermore, this dataset consists of a total of 771 images, split into 3 sequences, each with hand-crafted bounding box labels for each vehicle present in the image. These bounding boxes vary in shape and size as the perspective of the vehicles in each image are different.

4.1 Sliding Window Generation

Given that the image may contain vehicles of varying sizes, we would need to sample these windows at variable scales. Initially, we generated windows using an image pyramid method [2], where we would sample 64x64 pixel windows from an image at variable scales. However, as can be seen in **Figure 8**, when running our classifier on the sampled windows we obtained a large number of false positives, particularly with windows located in the sky or in trees. We hypothesize that this is due to a low number of these samples in our classifier training dataset, thus decreasing our models' confidence in such windows.

Fig. 8: Detection using Images Pyramid (*top*) compared to the Variable Scale (*bottom*).

Next we implemented a variable scale window generation, seen in **Figure 9**,which leverages the camera position and information about the scene to sample windows more intelligently. Firstly, we observed that vehicles only appeared below the horizon line. Thus, the top half of the image does not need to be sampled. We also observed that due to perspective, vehicles located closer to the bottom of the image are larger. Therefore, we sample windows starting from the horizon line and increase the size of the window as we approach the bottom of the image. As can be observed in **Figure 8**, the detection results are much improved and the rate of false positives reduced drastically using this method.

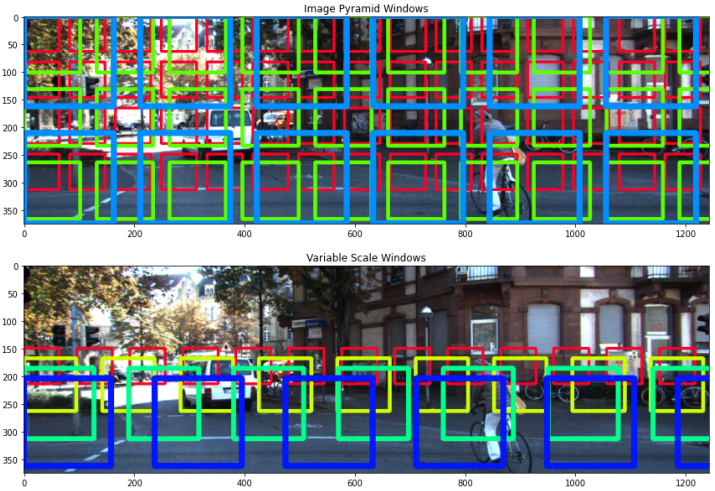

Fig. 9: Window generation of Image Pyramid method (*top*) compared to the Variable Scale method (*bottom)*.

4.2 Detection of Redundant Bounding Boxes

We implemented three different methods to detect and remove redundant bounding boxes. Their relative performance was measured using the average Intersection over Union (IoU) over a 3-fold cross validation on the three video sequences. The results of each method can be seen in Table 5.

Table 5: Detection evaluation results using the Linear SVM model across 3-fold cross validation.

Mean IoU (%)				
	Fold 1	Fold 2	Fold 3	**Average**
Non-Maxima Suppression	18.38	18.97	20.94	19.43
OpenCV Group Rectangles	19.81	20.43	19.05	19.76
Heatmaps	28.61	26.82	17.01	24.15

Firstly, we used non-maxima suppression to detect redundant bounding boxes by selecting the prediction with the highest confidence and suppressing all other predictions with an overlap greater than a certain threshold. This method performed the worst of the three methods used, achieving an average IoU of 19.43%. Since our classifier occasionally misclassifies windows and these are not filtered out by non-maxima suppression, these incorrect predictions drive down the average IoU.

Next we used the OpenCV groupRectangles function [3] to merge bounding boxes. This method produced slightly better results compared to non-maxima suppression, yielding an average IoU of 19.76%, however, it is also plagued with the same problem of detecting false positives. Despite the similar IoU results, the predicted bounding boxes from this method are visually better than those from non-maxima suppression as it reduces the number of redundant bounding boxes in a more intelligent way.

To combat the problem of false positives we implemented a third approach to merge the bounding boxes called heatmaps [4]. This method begins with a black image the same size of the original image. Then for every bounding box detected, a constant is added for each pixel covered by the bounding box. Therefore, areas with many overlapping boxes have the highest values and are where we are most confident there exists a car. The heatmap is then thresholded to remove false positives or other noise and finally new bounding boxes are drawn up from the resulting heatmap. Using this method we obtained the best results at 24.15% average IoU, 4.39% higher than the groupRectangles. Although by thresholding the heatmap we eliminate some correctly identified vehicles, the removal of false positives prove to be much more valuable. The results of the detection using heat maps can be observed in **Figure 10**.

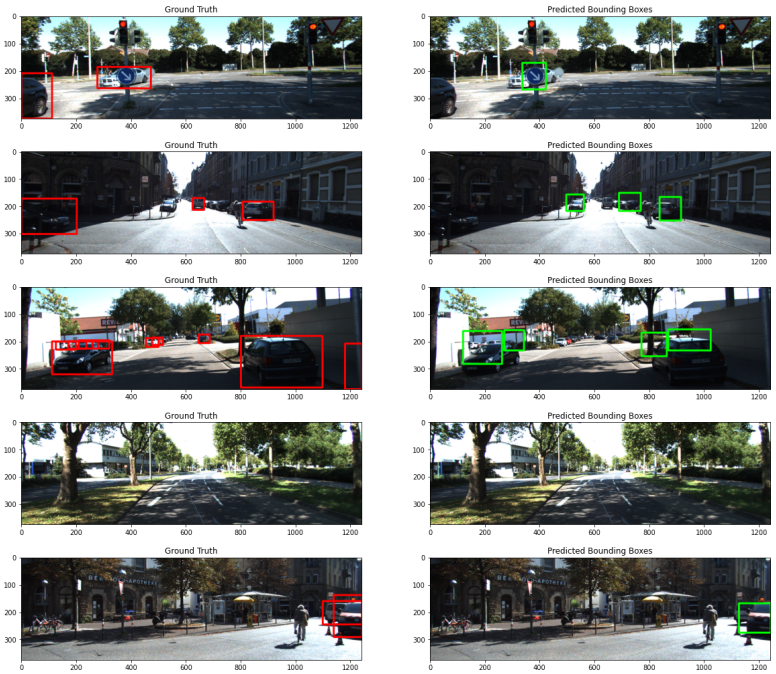

Fig. 10: Ground truth compared to predicted bounding boxes of randomly selected images using the Linear SVM classifier and Heatmap merging method.

5. Tracking

5.1 Lucas-Kanade Dense Optical Flow Tracking

In our first attempt at tracking, we implemented a tracking method based on the Lucas-Kanade Dense Optical Flow method which estimates the optical flow between two nearby frames of a video with the assumptions the flow is small and essentially constant in the neighborhood of the pixel into consideration.

The first step is to initialize the tracker by using our detection CNN model. We iterate through the frames until we make at least one detection. Next, we iterate through the subsequent frames and perform tracking by computing the dense optical flow within each of the bounding boxes. The mean of the optical flows for each box is calculated and used to displace the bounding boxes in the next frame. To make our tracking method more robust, we perform a backward check and compute the difference between the original points (p0) and the retrieved points (p0r). If the difference between these points is larger than 1, we discard those points because they are inconsistent. Finally, every 5 frames we perform detection again to reduce tracking error which grows with each new frame.

We ran the tracking on dataset 3 using the CNN model for detection and the NMS method to remove redundant bounding boxes. Our tracking method performed as we expected given the IoU results of our CNN model using NMS (see table 8). Indeed, we achieved a mean IoU of 37.64%.

Table 6: Mean and Median IoU results for tracking performed on dataset 3 using NMS

Lucas-Kanade Tracker	
Mean IoU (%)	Median IoU (%)
37.64	38.68

5.2 Other Method

For our second tracking method, we followed Tutorial 12 in order to try out different tracking methods from the OpenCV library. We determined that the Multiple Instance Learning (MIL) tracker would be the best one for our application, as it is fairly accurate and runs quickly compared to the other trackers available through OpenCV. MIL uses an online classifier to determine whether kernels around the initial bounding box in subsequent frames - determined using the existing detection code - belong to the object or the background.

6. Deep Learning Implementation

6.1 Model Selection

Due to the subpar performance of our vehicle detection and localization, we decided to implement a deep learning classifier to perform this task. We ultimately chose to use a Residual Neural Network (ResNet) architecture because of its excellent performance in image classification tasks, boasting better performance compared to other architectures such as VGG16 and traditional CNN's [9]. ResNet accomplishes this increased performance by utilizing skip connections, which avoids the problem of vanishing gradients enabling deeper architectures and faster training times.

Ultimately, since binary classification is a relatively simple task for deep neural networks, we chose to implement the smaller ResNet18 architecture to avoid overfitting our training data. As the name indicates, this model has 18 convolutional layers and a fully connected output layer with a sigmoid activation function. Therefore, the model is inputted a 64x64 RGB image and outputs a probability on whether the input contains a vehicle or not.

6.2 Training

The same dataset as described in **Section 2** was used to train the ResNet model. All images within this dataset were resized to 64x64. This dataset was then split into training, validation, and test datasets with a 80%-15%-5% split respectively, as seen in **Figure 11**.

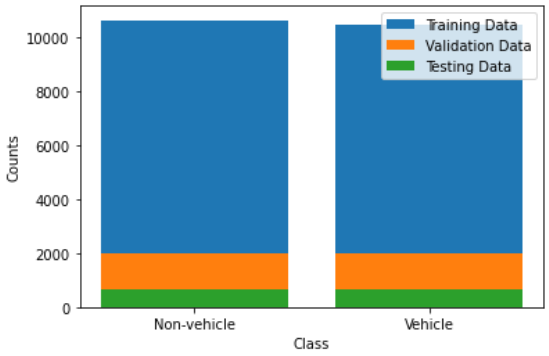

Fig.11: Distribution of dataset for neural network classification

The model was trained on the training dataset for 20 epochs. This was empirically determined as after 15 epochs, we observed that the training accuracy remained stagnant and the validation accuracy decreased as we overfit the training data. The model was then tested on the validation set after each epoch. The model that achieved the highest prediction accuracy was selected as our final model.

Table 7: Classification accuracy of ResNet18 model trained for 20 epochs on a 80%-15%-5% training-validation-test split.

Accuracy (%)		
Train	Validation	Test
99.71	98.74	98.41

6.3 Evaluation and Comparison

The same sliding window and bounding box merging methods that were used in **Section 4.1** were used with the ResNet model. Initially, the model produced a few false positives on each input image. To counteract this, window samples were only classified as a vehicle if the model was over 99.99% certain the window contained an image. As can be observed in **Figure 12,** this change drastically reduced the number of false positives, however, this also caused the model to fail to detect vehicles which were extremely occluded or distant. Overall, this change resulted in an 11.21% average increase in mean IoU during our cross validation trials.

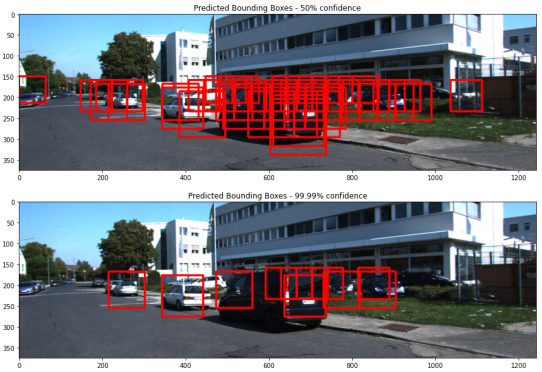

Fig.12: ResNet18 predicted bounding boxes with different confidence thresholds.

Again the worst performing merging method is non-maxima suppression which results in an 36.86% average IoU. Regardless, the deep learning detection still out performed the best Linear SVM implementation by 12.71% average IoU. Furthermore, the OpenCV Group Rectangles bounding box merging was once again in the middle of the pack, with an 41.36% average IoU.

It can be observed from **Table 8**, that similar to the detection with the Linear SVM classifier, the best localization results were obtained when using heatmaps to merge the bounding boxes, due to its ability to filter out false positives and create final bounding boxes of differing sizes. This deep learning implementation of detection boasts a 21.05% increase in average IoU compared to the Linear SVM with heatmap implementation. The results of this detection pipeline using ResNet18 and heat maps can be observed in **Figure 13**.

Overall, not only was the ResNet classifier much more accurate than the Linear SVM classifier, it was also faster. The localization 3-fold cross validation implemented with the ResNet took only 1 minute and 36 seconds, while the same task run with the Linear SVM classifier took 17 minutes and 38 seconds. This 11 time speedup makes the deep learning localization much more suited for real time vehicle detection.

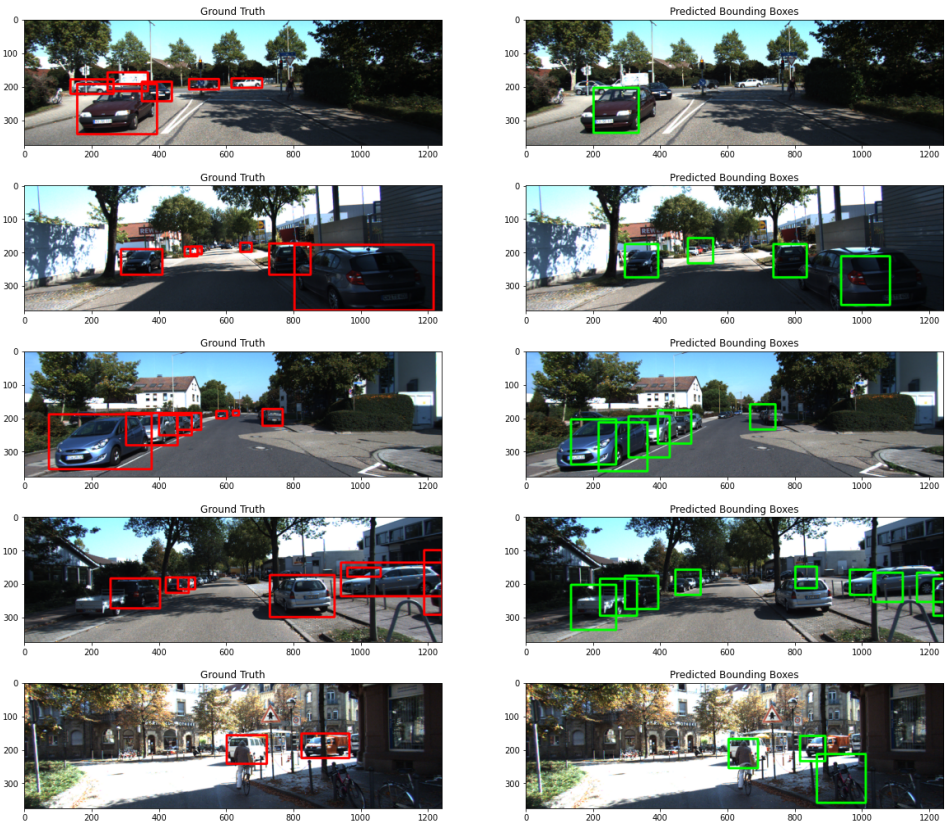

Fig.13: Ground truth compared to predicted bounding boxes of randomly selected images using the ResNet18 classifier and Heatmap merging method.

Table 8: Detection evaluation results using the ResNet18 model across 3-fold cross validation.

Mean IoU (%)				
	Fold 1	**Fold 2**	**Fold 3**	**Average**
Non-Maxima Suppression	32.13	37.89	40.56	36.86
OpenCV Group Rectangles	40.36	38.86	44.87	41.36
Heatmaps	42.57	43.37	49.67	45.20

7. Conclusion

Overall, after trying many different libraries and methods, adjusting parameters, and modifying the dataset, we managed to implement a functional pipeline for vehicle detection, localization, and tracking.

In terms of the dataset, using vehicle samples from external dataset helped considerably increase the variance of vehicle samples of our model, which in turn made it more robust.
For classification, considerable improvements on the RBF SVM classifier were observed by applying the following preprocessing steps to the images: extracting HoG features as well as (16x16) spatial color features, and applying 0-1 normalization. The best overall classification accuracy using SVM classifiers was achieved using a RBF SVM with C=5 and gamma=10^{-5}. This performance was largely outperformed by the ResNet CNN model.

The sliding window method was used alongside the trained classifiers to detect vehicles in a given image. By leveraging the information about the camera location, the variable scale window generation not only greatly reduced the number of false positives detected compared to the image pyramid approach, but the same search area could be covered in fewer windows, resulting in more efficient vehicle detection. Furthermore, for both classifiers, the highest average IoU was obtained when using heatmaps to merge and remove redundant bounding boxes, as this method was the best at filtering out false detections. Overall, the deep learning localization was far superior to the Linear SVM localization in both computation time and detection accuracy, and therefore deep learning models are more suitable for real-time and safety critical systems.

For tracking, we had success with the Lucas-Kanade Optical Flow method when combined with the CNN model for detection and the NMS method, achieving a mean and median IOU of almost 40%.

8. References

[1] https://github.com/udacity/CarND-Vehicle-Detection

[2] https://pyimagesearch.com/2015/03/23/sliding-windows-for-object-detection-with-python-and-opencv

[3] https://docs.opencv.org/3.4/d5/d54/group__objdetect.html#ga3dba897ade8aa8227edda66508e16ab9

[4] https://nrsyed.com/2018/05/06/hog-based-svm-for-detecting-vehicles-in-a-video-part-2/

[5] https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

[6] https://github.com/opencv/opencv/blob/3.4/samples/python/lk_track.py

[7] https://learnopencv.com/object-tracking-using-opencv-cpp-python/

[8] https://learnopencv.com/multitracker-multiple-object-tracking-using-opencv-c-python/

[9] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

Appendix A: Dataset Statistics

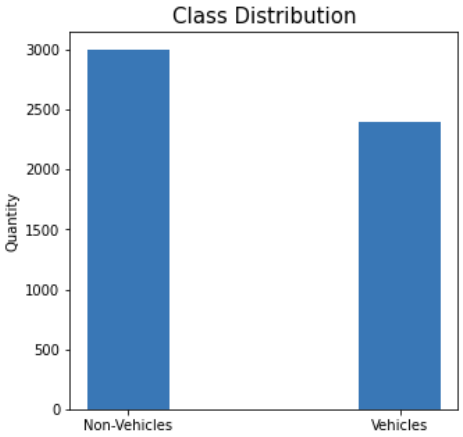

Fig. 1: Class distribution of the dataset

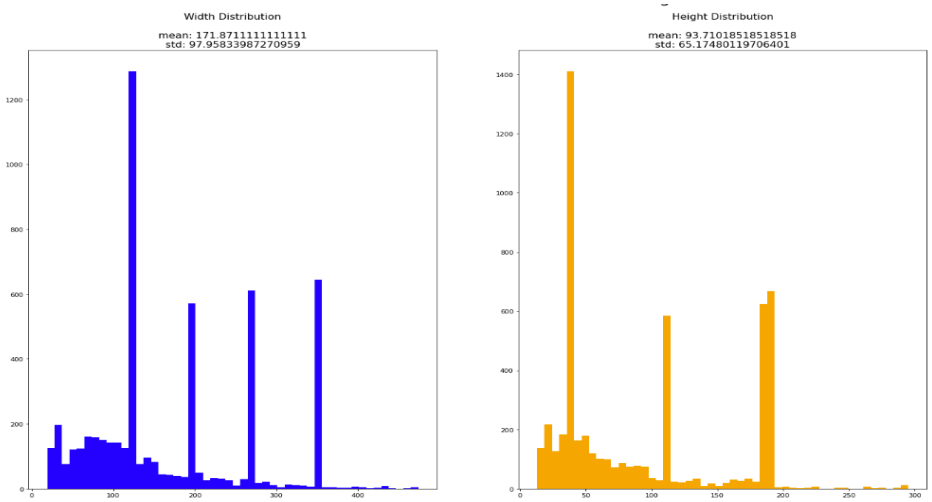

Fig. 2: Width and height distributions for the dataset samples

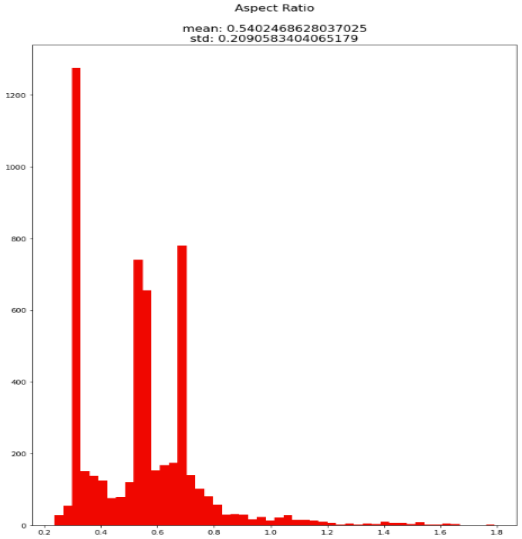

Fig. 3: Aspect ratio distribution for the dataset samples

Appendix B: SVM Classifier

Table 1: Vehicle detection performance resulting from cross-validation performed on different classifier types.

	Accuracy (%)	Recall (%)	Precision (%)	Inference Speed (ms)
Linear SVM (C=1000)	81.52 ± 2.09	77.63 ± 2.11	80.18 ± 2.58	0.012 ± 0.0022
RBF SVM (C=10, gamma = 0.0001)	88.96 ± 2.03	82.71 ± 2.57	91.79 ± 3.79	29.66 ± 1.1
Random Forest (500 estimators)	85.13 ± 4.43	80.00 ± 3.98	85.77 ± 6.38	0.04 ± 0.0012

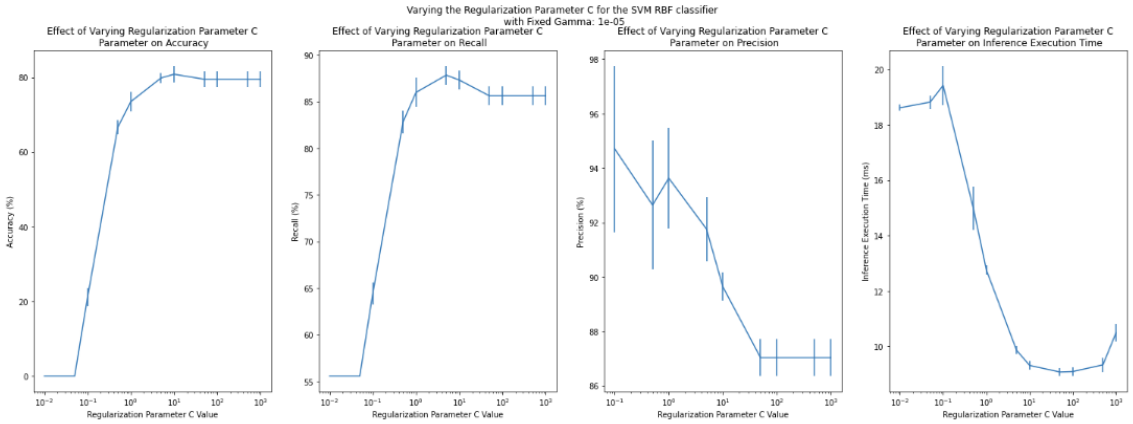

Fig. 1: Effect of increasing C logarithmically on detection performance using a fixed gamma of 10^{-5}.

Table 2: Overall vehicle detection performance resulting from cross-validations with and without feature normalization.

	Accuracy (%)	Recall (%)	Precision (%)
With Normalization (RBF SVM, C=10, gamma = 10^{-1})	89.00 ± 1.82	82.63 ± 1.97	91.90 ± 3.56
Without Normalization (RBF SVM, C=10, gamma = 10^{-1})	85.37 ± 1.28	73.17 ± 5.58	92.65 ± 3.13

Table 3: Overall vehicle detection performance resulting from cross-validations with and without feature gray scaling.

	Accuracy (%)	**Recall (%)**	**Precision (%)**
With Grayscaling (RBF SVM, C=10, gamma = 10^{-1})	89.00 ± 1.82	82.63 ± 1.97	91.90 ± 3.56
Without Grayscaling (RBF SVM, C=10, gamma = 10^{-1})	90.83 ± 1.29	85.04 ± 2.33	93.82 ± 2.29

Table 4: Overall vehicle detection performance resulting from cross-validations with different spatial features extraction settings.

	Accuracy (%)	**Recall (%)**	**Precision (%)**
Without Spatial Features Extraction (RBF SVM, C=10, gamma = 10^{-1})	89.00 ± 1.82	82.63 ± 1.97	91.90 ± 3.56
With 8 x 8 Spatial Features Extraction (RBF SVM, C=10, gamma = 10^{-1})	91.019 ± 1.99	85.67 ± 3.88	94.01 ± 2.24
With 16 x 16 Spatial Features Extraction (RBF SVM, C=10, gamma = 10^{-1})	91.33 ± 1.79	85.33 ± 3.53	94.25 ± 2.81
With 32 x 32 Spatial Features Extraction (RBF SVM, C=10, gamma = 10^{-1})	91.15 ± 1.91	85.75 ± 3.97	93.92 ± 289

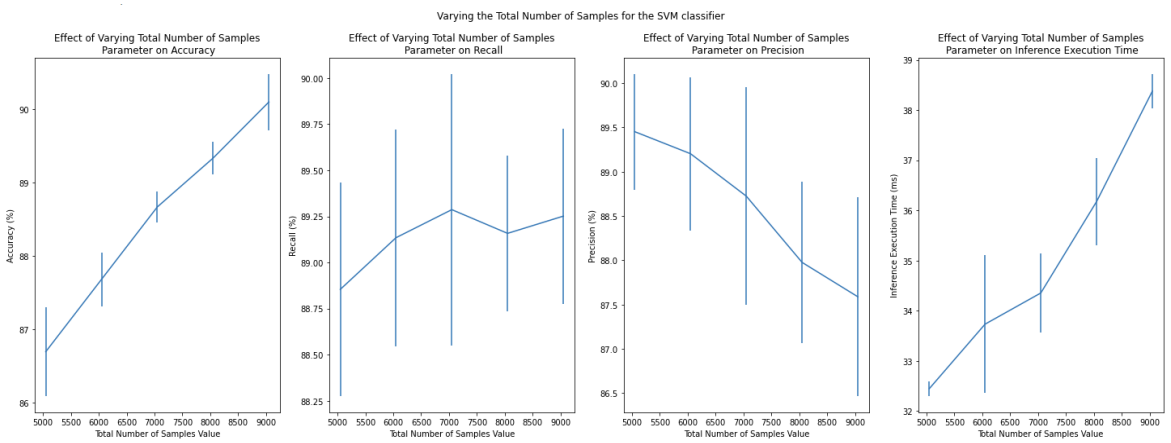

Fig 2: Overall vehicle detection performance resulting from cross-validations with increasing number of vehicle samples

Appendix C: Prediction Results on the Test Set

Fig. 1: Successful non-vehicle predictions on test set

Fig. 2: Successful vehicle predictions on test set

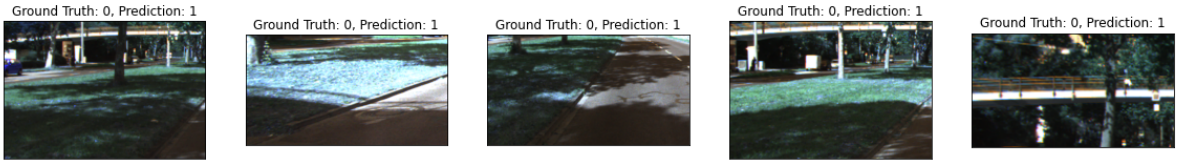

Fig. 3: Unsuccessful non-vehicle predictions on test set

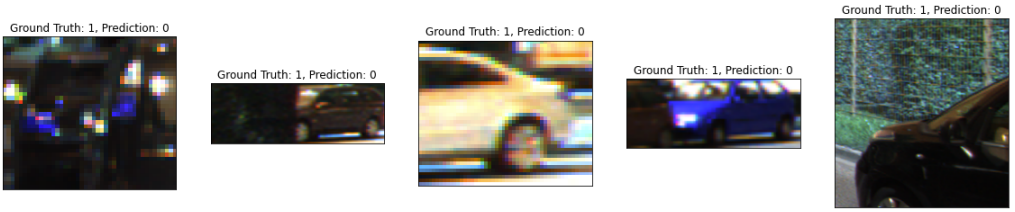

Fig. 4: Unsuccessful vehicle predictions on test set