A deep approach for an adaptive cruise control system

Alessandro Crescenzi, Giovanni Lambertini, Alvaro Gjepali
University of Modena and Reggio Emilia
{241790, 255321, 258974}@studenti.unimore.it

Abstract

Nowadays, the development of ADAS systems as driver supports is increasingly gaining ground. It has been shown that the installation of these systems in vehicles has greatly improved the quality of driving, reduced the number of accidents and helped to reduce driver stress, allowing longer and safer journeys to be supported. For this reason, we present a deep approach for an adaptive cruise control system. Our aim is to start from vehicle point-of-view images and perform the instance segmentation of the principal urban environment elements and in particular the recognition of the main speed limit traffic signs. In addition, we make in parallel a vehicle-to-vehicle distance estimation system to calculate the distance from the previous vehicle and improve our adaptive cruise control system. Our code is available here

1. Introduction

Adaptive cruise control (ACC) has become increasingly important in today's fast-paced world. Adaptive cruise control systems enable vehicles to maintain a safe and consistent distance from the vehicle ahead while automatically adjusting the speed as needed. One of the primary advantages of adaptive cruise control is its contribution to road safety. The system's ability to monitor the traffic environment and react to changing conditions helps prevent accidents caused by driver inattention or delayed reactions. Furthermore, adaptive cruise control can improve fuel efficiency by optimizing acceleration and deceleration, leading to smoother driving and reduced fuel consumption. Our work is divided into four phases.

1.1. Data acquisition & Preprocessing

We've recorded both low and high quality camera-car videos with a dash-cam placed behind the rearview mirror of our cars.

The preprocessing part of our pipeline is composed by distortion corretion and noise removal for the entire video frames and reflection removal for the traffic signs recognition part.

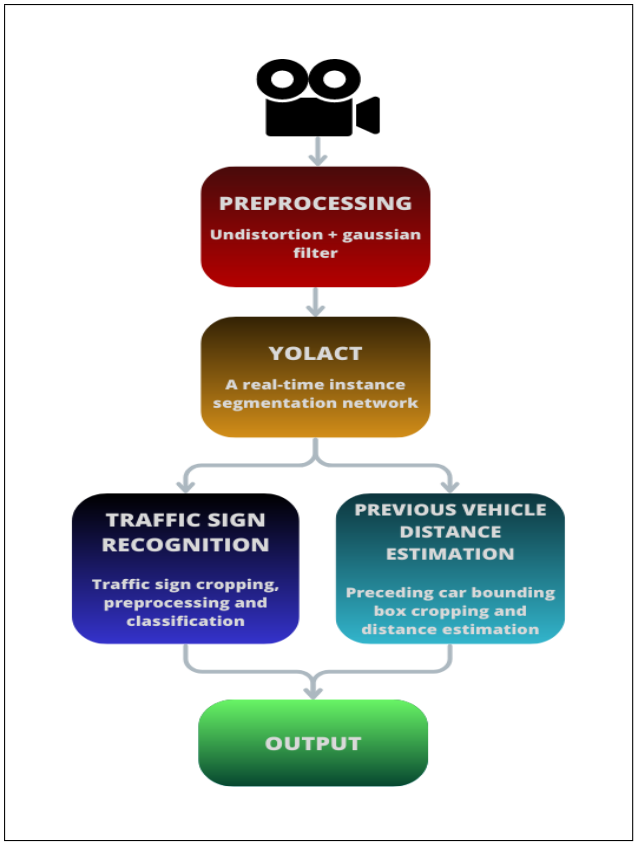


Figure 1. Complete pipeline of our ACC system.

1.2. Instance segmentation

As for the neural network, we've decided to use Yolact, short for "You Only Look At CoefficientTs" [2], which is a simple, fully convolutional model for real-time instance segmentation. The dataset used for training the network was the Cityscapes Dataset, a popular large-scale urban scene understanding dataset that provides pixel-level annotations for various urban street scenes.

1.3. Traffic sign recognition

We followed a similar approach to the "German Traffic Sign Recognition Benchmark" focusing on a smaller set of traffic signs, in particular: speed limits, end of speed limits, yield and stops signs.

1.4. Distance estimation

In order to estimate the distance from the vehicle in front we used the triangles similarity approach. It was useful to detect if the driver was respecting the safety distance or not.

2. Related Work

We now present a brief introduction to other approaches and where our work fits in. We have divided this part into three subsections, each related to its scope: real-time instance segmentation, traffic sign recognition and vehicle distance estimation.

2.1. Real-time instance segmentation

As for the part of real-time instance segmentation, there are several studies on the subject that in recent years have tried to improve this computer vision task, both in terms of accuracy and speed, trying to achieve performance comparable to real-time. There are many projects to increase the performance of object detection models, both in speed and accuracy, as YOLO [16], SSD [13], Faster R-CNN [17]. And also for semantic segmentation models there are many projects to upgrade using different techniques as SegNet [1] or SegFormer [19]. In the instance segmentation field, instead, the reference network is still Mask-R-CNN [7] (that generates regions-of-interest (ROIs) candidates in a first step and subsequently, in a second step, it classifies and segments them). This as meaning that few approaches have proven to make significant improvements in both accuracy and speed. A first approach implemented by MaskLab [4], was to make parallel the two steps of detection and segmentation. Although we have gone from a two-stage approach to a single-stage approach, to achieve performance accuracy at least comparable to those of state-of-art models, it is necessary to perform non-trivial operations (extra convolutional preprocessing) that affects the speed. Another approach, which is also the one from which we started to develop our network is the one used by Bolya et al. in YOLACT [2]. Also in this case a single-stage model has been used by parallelizing two tasks: the generation of a set of masks of non-local prototypes on the whole image and the prediction of a set of linear combination coefficients per instance. Then, for each instance, linearly combines the prototypes using the corresponding predicted coefficients and then crops with a predicted bounding box. They show that, doing segmentation in this manner, the model learns to localize instance masks on its own. Thanks to its highly parallel structure, where the only overhead is due to the backbone network, this network is fast and it generates high quality masks.

2.2. Traffic sign recognition

There are many papers and projects in the task of traffic sign recognition, as well as there are dozens of datasets for training. The most famous dataset, also for historical reasons, is the German Traffic Sign Recognition Benchmark (GTSRB) [9], it is a various light conditions and rich backgrounds collection of 39,209 training images and 12,630 test images belonging to 43 classes of traffic signs. This is the dataset that we used for the implementation and training of our network. For what concern the models used for the traffic sign recognition task, Shimtuber et al. [5] proposed a model obtained combining various DNNs trained on differently preprocessed data, making the system insensitive also to variations in contrast and illumination. The outputs of each DNN are averaged to obtain a final prediction. They realized 4 types of preprocessing and subsequently 5 nets have been trained for each of the 5 datasets, so 25 nets were trained in total. Although the size of each of the networks is very small and not very heavy from the computational point of view, it did not seem to us the best approach to use, given the important constraints of real-time. Another defect of the previous approach is the invariance to translation. Given the high dynamism of our working environment, we have used an invariant approach to translation, contrast and brightness. We used a Spatial Transformer Network [10], to create our own network to perform the task of traffic sign recognition in a light, fast and accurate way.

2.3. Vehicle distance estimation

The calculation of the distance from the preceding vehicle with monocular camera is often dealt with a geometric approach, given the difficulty of obtaining a reliable and well-annotated dataset. Usually, in most of the existing works relating to the calculation of the distance from the preceding vehicle, there are two types of objects that are taken into account: the vehicle or its plate. After detecting and localizing one of these two objects in our image, we calculate the size in pixels and then, the distance from the object is computed. For example, [15] makes massive use of preprocessing to locate the license plate edges of a vehicle mainly via Canny edge detectors [3] and Probabilistic Hough Transform Method [14]. [18] instead exploits the fact that the background color of a Chinese license plate is blue for large vehicles and yellow for smaller ones. Taking advantage of these features, they acquire the correct angles of the plate and from there, its size in pixels. These approaches are the most accurate because they are based on the size of the plate that is known a priori for most countries, but they rely on strong assumptions: the size of the

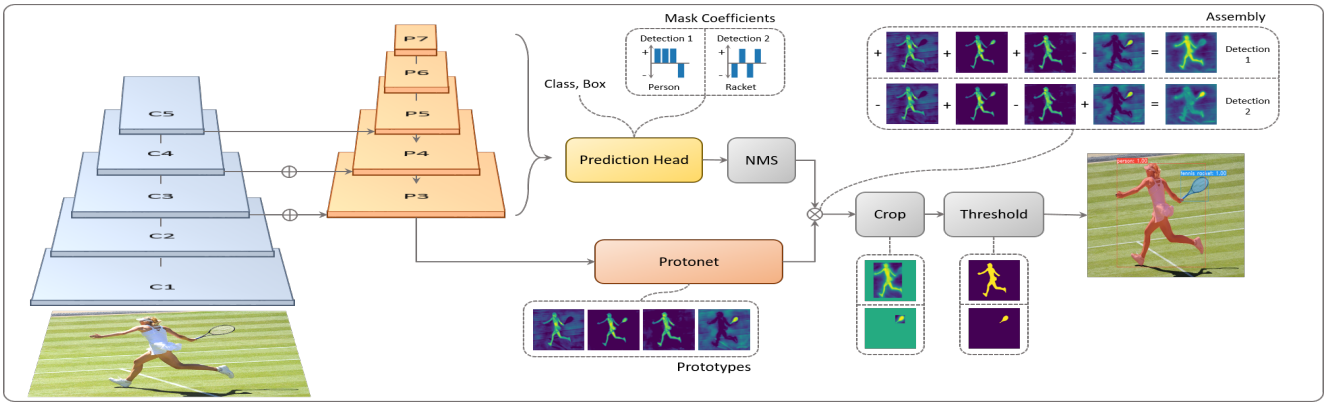


Figure 2. A complete pipeline of YOLACT. The blue blocks represent the backbone net. The orange one represent the FPN part. As it is possible to see, the prononet branch and the mask coefficient one can be execute in parallel. Their respective outputs are combined together. This image is taken from [2].

plate is fixed, but actually it may vary slightly from country to country, it is taken for granted that the method returns exactly the bounding box of the plate and moreover these methods do not work if the previous vehicle is very far. On the other hand, NVIDIA's DRIVE Labs was able to create a model [1] of deep neural networks trained on automatically labeled radar and LiDAR data for estimating distances from objects. This is definitely a more innovative approach, but from a computational point of view it is definitely more expensive than all other types of geometric approaches. Since we wanted a real-time system and at the same time usable in all contexts, we preferred to use a system that is based on the calculation of the distance from the preceding vehicle, starting from its bounding box.

3. Our Model

Our proposed model is composed following a pipeline that is illustrated in Figure 1. First of all we discuss about the video acquisition, calibration and undistortion. Then we introduce how we used YOLACT [2] to perform instance segmentation task in an urban and suburban environment. Finally, in the last section we will show how we implemented our custom network for the traffic sign recognition and after that we will explain how we decided to calculate the distance from the previous vehicle.

3.1. Video acquisition and preprocessing

The dataset we used to test our model consists of 100+ car-videos recorded by a dash-cam placed behind the rear view mirror. It contains short clips of three minutes taken in different urban scenarios. We captured both low quality (640x480) and high quality videos (1920x1440). We used the firsts to have a real time pipeline and the seconds to obtain better results in terms of quality. To preprocess our videos we followed these three steps:

[1] https://blogs.nvidia.com/blog/2019/06/19/drive-labs-distance-to-object-detection/

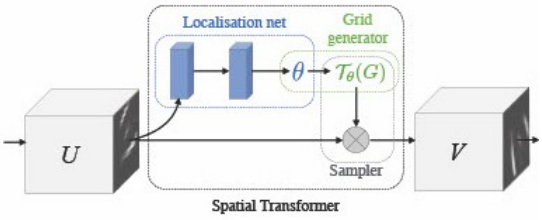
(a) Before (b) After

Figure 3. The difference before and after the undistortion process.

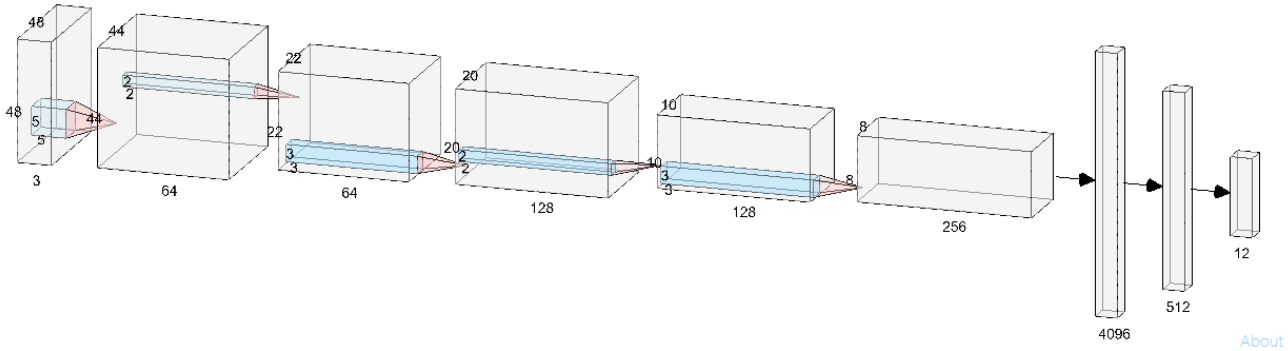
Calibration We used the Zhang method [21] to calibrate our camera and estimate its intrinsic parameters. We used them to calculate distance from previous vehicle together with the bounding boxes extracted in instance segmentation step. Figure 3 shows very well the difference before and after the undistortion.

Gaussian filter We used this very famous filter to reduce the noise and remove artifacts in our videos. After running multiple tests with different parameters, we opted for a kernel size of 5x5 and a sigma of 0 for high quality videos and a kernel size 3x3 and sigma of 0 for low quality ones.

Reflection removal Being positioned behind the rea rview mirror, our camera has generated videos that are affected by an harmful reflection of the dashboard generated by the sun. To try to remove it we found this method [20] that tries to remove the reflection layer starting from one single image with awesome results. It uses partial differential equations with gradient thresholding and Discrete Co-



(a) Spatial Transformer Network.



(b) CNN part of the traffic sign recognition network.

Figure 4

sine Transform, and we tried to optimize the calculations on our high quality images but the processing speed per image wasn't enough for our model to be real-time. Therefore we decided to implement the reflection removal only to our point of interest, the traffic signs.

3.2. Real-time instance segmentation

Since our starting model had been trained on COCO [12], we had to fine-tune the net with Cityscapes [6]. It consists in a large, diverse set of stereo video sequences recorded in streets from 50 different cities. 5000 of these images have high quality pixel-level annotations; 20000 additional images have coarse annotations. To start the training, we had to do some preliminary steps. First of all, for the sake of compatibility with COCO style dataset, we had to convert the Cityscapes annotations to a json file that could be read from the network. Secondly, we modified the last layer related to the prediction of confidence values for each of the dataset classes, passing from 80 classes of COCO to 10 classes of Cityscapes. Actually Cityscapes contains many more classes but we have selected only those of our interest: "person", "bicycle", "car", "motorcycle", "rider", "bus", "train", "truck", "traffic sign", "traffic light". Then we loaded the weights of the network trained with COCO, except the ones related to the confidence values of the classes. Finally, the training was carried out in two phases. In the first phase the network was trained using the dataset of 20000 images containing coarse annotations. In the second phase the network was trained using the dataset of 5000 images containing fine annotations. For what concern the configuration of our network, we used ResNet-50 [8] with FPN [11] as our default feature extractor backbone and a base image size of 700x700.

3.3. Traffic sign recognition

The instance segmentation task returns the masks and bounding boxes of all the road elements of our interest in the scene. Then the bouding boxes of the traffic signs and the previous vehicle are filtered. The latter will be used to calculate the distance, we will discuss about it in the subsection 3.4. Since the class of traffic signs includes shop signs, billboards and real traffic signs, we trained our network on a custom dataset made by the merge of GTSRB [9] and a set of road signs identified by our network of instance segmentation and labelled by hand. We were interested in the classification of speed limits, stop and yeld signs only, so we created an additional class "unknown" to put all the traffic signs that were not of our interest.

In total we created 12 classes:

- "Speed Limit 20Kph",
- "Speed Limit 30Kph",
- "Speed Limit 50Kph",
- "Speed Limit 60Kph",
- "Speed Limit 70Kph",

- ”Speed Limit 80Kph”,
- ”Speed Limit 100Kph”,
- ”Speed Limit 120Kph”,
- ”Yield”,
- ”Stop”,
- ”End of Speed Limits”,
- ”Unknown”

For the creation of our custom network we used two main components sequentially: first a Spatial Transformer Network (STN) [10] and then a classic convolutive neural network. In our videos every traffic sign was seen at different scale values and from multiple views. For the training of our network we decided to use a fixed size of 48x48 and to apply a network that would allow us to be invariant not only to contrast and brightness, but also to affine transformations. The structure of an STN can be seen in Figure 4a, in particular it takes an image and returns an output map of the same size, sampled from the input image and the grid points (derived from a localization network, a grid generator and also the input). The transformation provided by the STN is performed on the entire image and can include scaling, cropping, rotations, as well as non-rigid deformations. This to allow networks to transform the most relevant regions in a canonical and expected pose, to simplify recognition. To maximally generalize the transformation of our image we apply an affine transformation:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{1}$$

which has been initialized as an identity transformation. Since it is differentiable with respect to the parameters, it allows gradients to be backpropagated from the sample points to the localization network output. The output of the STN was always the size of 48x48 and was sent in input to a CNN, the 3D structure of which is illustrated in the image 4b. A more detailed view of all network blocks is depicted in Figure 5, in which you can see the whole sequence of convolutions, poolings, non-linerities and fully connected layers.

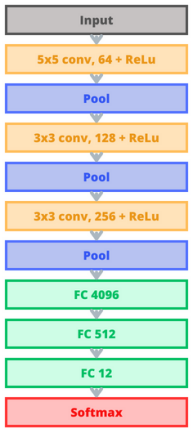


Figure 5. Graphic illustration of all the blocks that make up the convolutional part of our network.

3.4. Vehicle distance estimation

In order to have an effective adaptive cruise control we had to compute the distance from the vehicle in front. So, our purpose was to compute the distance between the camera and an object using the image from a single camera. To do this, considering the Pinhole camera model, it is possible to exploit the similarity among triangles approach. Knowing the focal length of the camera and the real height of the object from which we want to compute the distance, it is easy to compute the distance between the object and the camera using the formula 2:

$$Distance = \frac{focal\_length \cdot real\_vehicle\_height}{image\_vehicle\_height} \tag{2}$$

Of course, we couldn't know exactly the height of the vehicle in front. So, we computed an estimation of it, based on the type of vehicle. To make a good estimation of the average height of a vehicle, we used the cityscapes3d dataset. It provides hundreds of images with 3d labels for trucks and cars. The labels include the 3d position of the center of the vehicle, with respect to the camera, and the 3 dimensions (width, height, and depth) expressed in meters. This dataset was obtained using stereo camera images. We made a python script to compute the average height of cars and trucks, then we computed the percentage error between the distance obtained from the labeled 3d coordinates and the distance computed using the triangular similarity (Cityscapes also provides the parameters of the camera). We obtained a mean percentage error of about 11%. We also tried to do the same thing with the width of the vehicle, but the results were much worse. To detect the bounding box of the vehicle in front, and so to compute the height in

pixel of it, we considered the nearest vehicle to the x coordinate of the principal point.

Using the same approach we used for the estimation of the distance from the vehicle in front, we computed the distance from stop and yield traffic signs and we used this information to print a warning in the bottom left corner of the frames, as you can see in figures 7e, 7f.

Safety Distance Estimation To compute the safety distance, considering the speed of the vehicle expressed in m/s, we used the formula 3:

$$Safety\_distance = reaction\_time \cdot car\_speed \quad (3)$$

We considered a reaction time of one second and to compute the speed of the car we used GPS information provided by the camera. In our videos, we printed the estimated distance from the vehicle in front with a green background if the safety distance was respected and with a red background otherwise, as you can see in Figure 7.

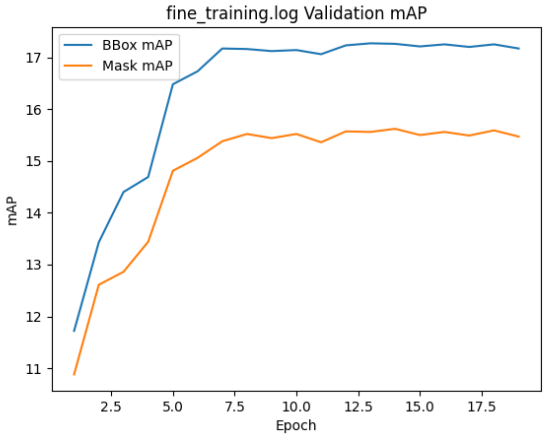
4. Implementation & Results

In this section we will explain further details about the implementation of our networks and systems in general, and we will show some results about them.

4.1. Real-time instance segmentation

We trained our model with batch size 8 on one GPU Tesla T4 with 16GB VRAM. As already mentioned in the subsection 3.2 we performed our training into two phases. In the first one, we had a dataset consisting of 20000 images with coarse annotations. We did several experiments and we saw that the best results were obtained by doing the first training for 2 epochs only and it lasted about 3 hours. Being a dataset composed of very coarse annotations, our goal was to make the network learn only a general idea of the classes of our interest, going to modify the weights related to the COCO classes that we had loaded before starting the training. In the second phase, the most important, we had only 3475 images available with fine annotations (2975 images for training and 500 images for validation). These images were used to train our model for 20 epochs for a total of 12 hours. As you can see from the graphs in Figure 6b, although the mAP after the seventh epoch tends to increase by small values at a time, we took the weights related to the epoch 14 where we got the maximum mAP value for both bbox and masks. For training we took into account 4 type of loss, as it is possible to see in Figure 6a. Bounding box and mask losses were the most difficult to lower, due to the fact that we had few fine annotated images

(a) Values of the four losses used in training.



(b) Values for mAP for BBox and Mask.

Figure 6

Method	$AP(\%)$	$AP_{50}(\%)$	FPS	$Training$
Mask R-CNN	26.2	49.9	1.8	fine
Ours	15.56	28.45	12.7	coarse+fine

Table 1. Results about YOLACT traing on Cityscapes, you can see differences in terms of mAP and FPS with respect to Mask R-CNN.

on which to do training. In Table 1 you can see the comparison between the results in terms of mAP for our YOLACT training, performed on Cityscapes and the results obtained by Mask R-CNN on the same dataset. As you can see our map value is much lower than the state-of-the-art but we got a considerable gain in terms of fps, going from 2fps (R-CNN Mask) to 13fps. In Figure 7a it is possible to see an example of instance segmentation depicting various type of masks.

Figure 7. Some results about our complete pipeline processing.

Class	Accuracy	Precision	Recall
0	1	1	1
1	1	0.99	1
2	0.99	0.99	0.99
3	0.96	0.99	0.96
4	0.98	1	0.98
5	0.99	0.97	0.99
6	0.96	0.97	0.96
7	0.98	0.99	0.98
8	1	0.96	1
9	0.99	0.99	0.99
10	1	0.97	1
11	1	0.96	1
12	0.99	0.99	0.99
Ours	0.9907	0.9861	0.9906
MCDNN	0.995	-	-
CDNN	0.985	-	-

Table 2. Values of Accuracy, Precision and Recall for each classes and a global value. In the last row there are the performances of [5].

4.2. Traffic sign recognition

We used the same training hardware of the instance segmentation network. We started from GTSRB dataset [9] and we changed it only taking the classes of our interest (see Section 3.3) and put all the rest of the traffic signs in a single class called 'unknown'. To reduce the imbalance due to this change, we took from the classes that were not of interest to us only 30% of the elements, we deleted the other ones. Then we added a vector of weights to send in input to our loss. To augment our dataset we perform some further modifications in terms of contrast, luminosity, saturation, blurriness and other homogeneous transformations. Then we trained our custom network from scratch with our augmented dataset with batch size 128, Adam optimizer and early stopping technique to regularize our training. With this configuration we trained for 80 epochs (almost 9 hours) and we found that the minimum value of validation loss was at the 55th epoch. In Figure 8 it is possible to observe the confusion matrix obtained on the test set composed of 12631 elements. In the first row of Table 2 we displayed values of Accuracy, Precision and Recall for each class. In second and third rows instead it is possible to see a comparison between our global value of Accuracy and the one that reached Schmidthuber et al. in [5]. MCDNN refers to the value obtained from the average of the 25 networks, CDNN to the value from a single net. We obtained a similar value

with a network that is about 15 times smaller.

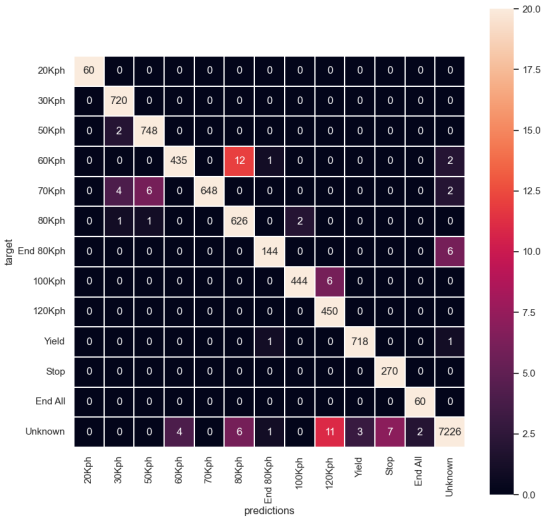


Figure 8. Confusion matrix on the GTSRB test set modified according to our use cases.

5. Conclusions

We developed an adaptive cruise control system joining deep approaches and geometric approach, obtaining good performances in real-time and in accuracy. To process the entire pipeline with one GPU Tesla T4 with 16GB VRAM, we obtained a speed of about 19fps for our low quality videos and of about 7fps for our high quality videos. In the last section we showed some results we obtained, but you can find other material (both good and bad cases) at this link:
A deep approach for an ACC system (material)

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(12):2481–2495, 2017. 2

[2] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact++: Better real-time instance segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020. 1, 2, 3

[3] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. 2

[4] Liang-Chieh Chen, Alexander Hermans, George Papandreou, Florian Schroff, Peng Wang, and Hartwig Adam. Masklab: Instance segmentation by refining object detection with semantic and direction features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 2

[5] Dan Cireşan, Ueli Meier, Jonathan Masci, and Jürgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural networks : the official journal of the International Neural Network Society*, 32:333–8, 02 2012. 2, 7

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 4

[7] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 4

[9] Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, 2013. 2, 4, 7

[10] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and koray kavukcuoglu. Spatial transformer networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 2, 5

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017. 4

[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014. 4

[13] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. 2

[14] Jiri Matas, Charles Galambos, and Josef Kittler. Robust detection of lines using the progressive probabilistic hough transform. *Computer vision and image understanding*, 78(1):119–137, 2000. 2

[15] Purnama Sari Nur Mellinda, Febryanti Sthevanie, and Kurniawan Nur Ramadhani. Detection of vehicle number plate using probabilistic hough transform. *eProceedings of Engineering*, 7(2), 2020. 2

[16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object de-

tection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[17] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. 2

[18] Wenfeng Wang, Shujun Yang, Yong Li, and Weili Ding. A rough vehicle distance measurement method using monocular vision and license plate. In *2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER)*, pages 426–430. IEEE, 2015. 2

[19] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M. Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 12077–12090. Curran Associates, Inc., 2021. 2

[20] Yang Yang, Wenye Ma, Yin Zheng, Jian-Feng Cai, and Weiyu Xu. Fast single image reflection suppression via convex optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 3

[21] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000. 3