Autonomous Car project

Matthieu Paques

28/05/2021

Introduction

State of the art of autonomous driving

The idea of an autonomous driving car has been explored for 30 years giving birth nowadays to functioning prototypes. The development of machine learning has allowed a rapid revolution in the control of autonomous vehicles. The most recent and stunning example relying on Deep Learning was given by the Tesla Autopilot in 2020. However a lot remains to be done to allow complete autonomy as the most recent models stands at level 2 over 5 (partial automation) regarding the U.S. National Highway Traffic Safety Administration norms. Autonomous driving implies four main tasks: Perception, Localization, Planning and Control. The use of machine learning in each of these domain may avoid the recourse to expensive hardware technologies such as lidar, radar or ultrasonic detectors. The current work only concerns perception and control.

Benchmark

The objective of the considered project is to control a miniature car on a realistic circuit using Machine Learning techniques. A forefront camera and a Raspberry Pi card are the only available hardware resources of the car. Thus prediction must entirely relies on a RGB image of the road. The control of the car is limited to the steering angle and the speed. The speed prediction has been reduced to "stop" or "go" with a "go" velocity of 0.2 m/s. The possible angle value ranges between 50 degrees and 130 degrees corresponding respectively to a maximum left turn and a maximum right turn. The car may be facing diverse situations following UK driving rules as straight lines, crossroads, loops and eight-shaped circuit. Different obstacles (person, car, tree, box) and traffic signals (red and green lights, left and right signals) may be present on the circuit.

Dataset

A training dataset of 13798 labelled RGB images was given. Speed and steering angles labels were both normalized such that $speed \in \{0, 1\}$ and $angle \in [0, 1]$. 1020 unlabelled test images were equally given to evaluate the performance of the model in a Kaggle competition. The images were produced in real driving conditions.

Model presentation

Architecture

A strategy of subdivision of tasks was chosen to solve the autonomous driving problem. Instead of having a single end-to-end model to perform predictions we preferred multiple task-specific models. The main motivation was to be able to control each step instead of having a single model acting as a "black box". Five mains tasks were identified to obtain as a final output the speed and the angle (see Figure 1). First object detection appeared as inevitable regarding the relative small training data size and the ambiguity of several classes of object. Indeed, the original dataset comprehends only 102 and 137 green and red lights respectively and both objects share multiple features in common. The same issue can be assumed for the left and right signals (205 left and 339 right signals). The object detection model takes as input the road image and output bounding boxes and classes of the detected object. To discriminate an object as an obstacle the limits of the left-hand road need to be known. A lanes segmentation model is implemented to convert the

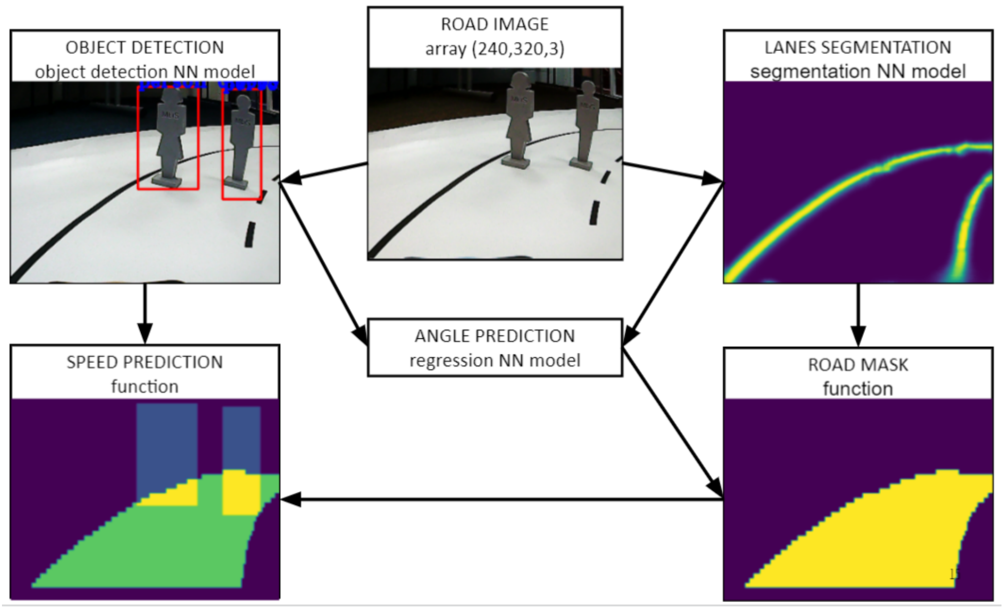


Figure 1: Overview of the different models and functions

road image to a binary mask of the lanes. Then the road mask is built by filling the area within the lanes. Knowing the direction angle proved to be useful to select the desired side of the road. The angle prediction depends both on the orientation of the lanes and the presence of left and right signals. A neural network model taking the aforementioned variables as input is implemented to solve the angle regression problem.

Object Detection

Model selection

Object detection is a multidimensional task implying object classification and localisation within an image. The localisation of the object can be made whether by assigning a class to each pixel of the image (semantic segmentation) or by delimiting the object boundaries as a rectangular box. Models producing bounding boxes reveals to be easier to train regarding loss convergence and production of training data.

Two main families of model exist for object detection: R-CNN and YOLO. The R-CNN was first described in (Girshick et al., 2014) from UC Berkeley. It proved to be one of the first satisfying applications of convolutional neural networks to the problem of object detection. R-CNN shown good performance on several benchmark dataset (20-class VOC-2012, 200-class ILSVRC-2013). Three main modules composed this model: Region proposal, Feature extractor and Feature classifier. The RetinaNet network (Lin et al., 2017) combine R-CNN principles with two major innovations. First, convolutional layers are organised in FPN (Feature Pyramid Network). In FPN, features are extracted by successive down-sampling layers allowing a raw to fine details description. Second, RetinaNet introduced the Focal Loss to replace the Cross-Entropy Loss. At each pyramid layers, thousands of anchor boxes are produced whilst only a small minority could be assigned to a ground-truth object. To increase prediction accuracy, "easy" negative samples (detections with high probabilities) will contribute to a smaller loss than "hard" samples (detections with low probabilities). RetinaNet architecture is composed of a backbone model for feature extraction (commonly ResNet+FPN), a sub-model for classification and a sub-model for bonding-box regression (see Figure 2). RetinaNet shown to outperform its competitors' accuracy on the COCO dataset with a mean Average Precision (mAP) of 40.8.

The YOLO model is a real-time object detector first described in (Redmon et al., 2016). This single neural network is an end-to-end trained model predicting bounding boxes and class labels for each bounding box directly (see Figure 3). While Mask-RCNN needs thousands of occurrence to cover the whole image, YOLO passes the image once without focusing on a specific area (You Only Look Once). Instead, the image is divided in a grid of cells. Then a decimation process called non-maximal suppression removes both boxes with low probability of containing an object and boxes sharing large areas with other boxes. This technique offers the advantage of a very fast prediction operating at 45 frames per second making YOLO suitable for

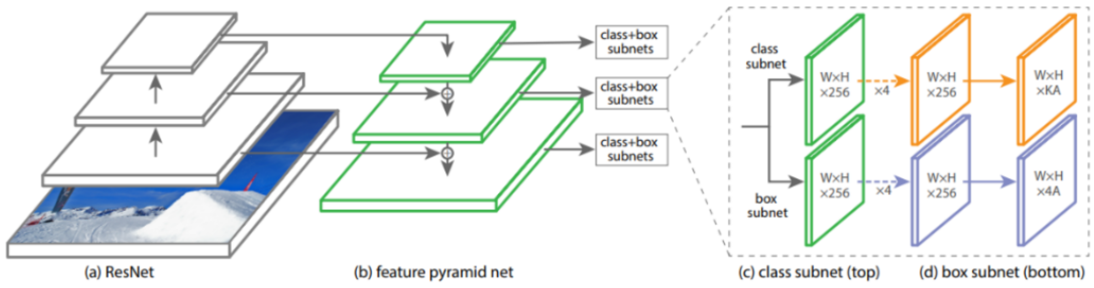


Figure 2: RetinaNet architecture. Retrieved from (Lin et al., 2017)

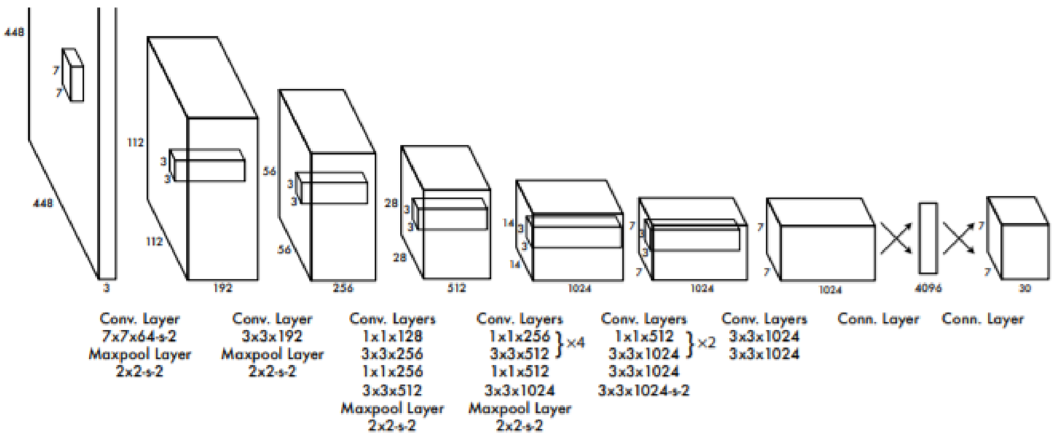

448
7
7
448
3
112
3
3
112
192
56
3
3
56
256
28
3
3
28
512
14
3
3
14
1024
7
3
3
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 ×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 ×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 3: YOLO architecture. Retrieved from (Redmon et al., 2016)

real-time application. Its accuracy is satisfying but still behind the RetinaNet with a mAP of 33.0.

Training

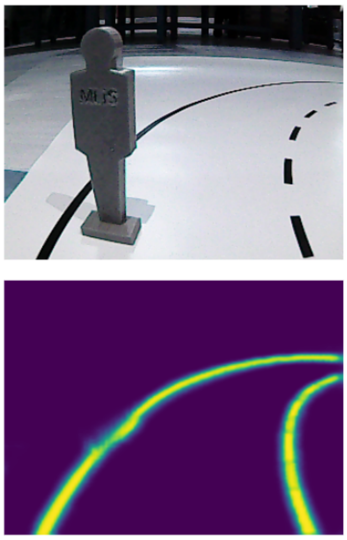
Two models were taken on to implement object detection. The RetinaNet model was first attempted for its classification and localization performances. However, the inference time constraints enforced by a real-time prediction problem motivate an abandonment of RetinaNet to the benefit of the Yolo network. The Yolo network also revealed to give better performances while being trained on a small dataset. We considered 8 classes of object from the given subset 'box','car', 'green light', 'left signal', 'person', 'red light', 'right signal', 'tree'. Training images with their associated bounding boxes were produced using the python tool LabelImg. 1020 images were first generated by hand with a particular care given to an equal representation of each class beyond the dataset. Data were augmented to reach the size of 3060 applying a random combination of crop, shear, hue, exposure, blur and rotation on the online software Roboflow. Object detection is a perfect application for Transfer Learning. Numerous existing models have been trained on large dataset and the classes we are considering belongs to daily objects. In our particular case, source and target domains are close and source and target tasks are similar. For the YOLO network, weights were obtained from (Sharma, 2020) trained on the VOC2012 dataset. The VOC2012 is composed of 5717 training data, 5823 validation data and 20 classes from common objects (car, person, bus, person,...). The RetinaNet model was equally trained by transfer learning from MS COCO dataset weights retrieved from (Gaiser, 2020).

Performance

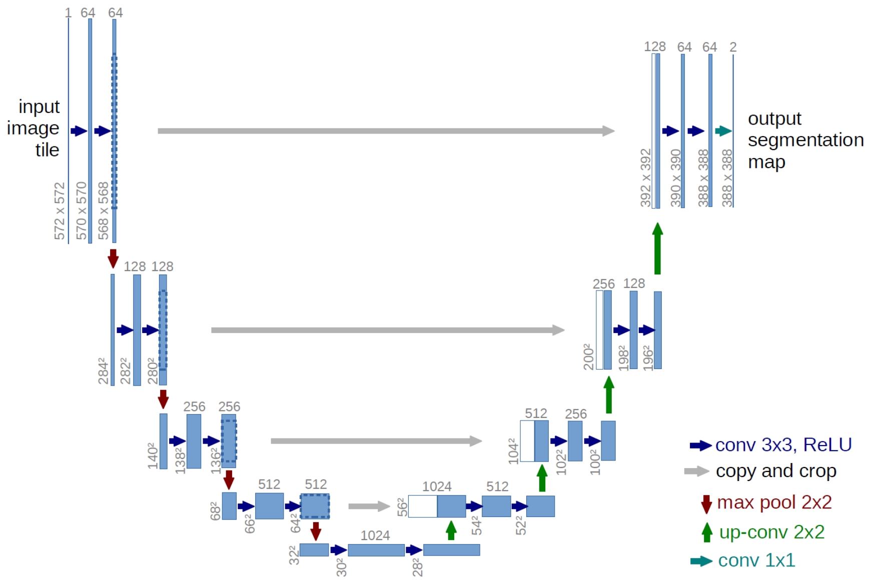
To correctly assess an object detector performances, at least two metrics are required. The IoU score (Intersection over Union) and the accuracy were selected to measure respectively the localisation and the classification losses. The unquestionable supremacy of the YOLO network regarding the localisation and the accuracy motivated its choice for the Kaggle competition (see Table 1). However, the live-test competition had recourse to YOLO-tiny to fit with inference time constraints.

Models comparison				
Models	Trainable parameters	IoU	Accuracy	Inference time (ms)
RetinaNet	12,967,020	0.5623	0.8240	109
YOLO	61,949,149	0.6889	0.9643	92
YOLO-tiny	8,852,366	0.3461	0.8627	30

Table 1: Performance of the object detectors



(a) TOP: Input image. DOWN: Output binary mask



(b) UNet architecture. Retrieved from (Ronneberger et al., 2015).

Figure 4: Lanes segmentation

Lanes Segmentation

Knowing what objects are present isn't sufficient to determine the speed. The limit of the left path remains to be found to sort detected objects as active obstacle or passive element from the setting. The UNet network (Ronneberger, Fischer, & Brox, 2015) appeared as a natural candidate for our semantic segmentation problem. Well documented, fast to train, the UNet has become a reference in medicine in the research of suspicious tumours in lungs or the brain.

UNet

Acting as an auto-encoder, the UNet generates a representation of data in a latent low dimensional space (see Figure 4(a)). The encoder consists in a succession of convolutional and max-pooling layers allowing to down-sample the dimensionality of the image representation. The decoder is built symmetrically to the decoder as a succession of deconvolutional and up-sampling layers. Therefore both input and output share the same dimension. This U-shape architecture of convolutional layers was already found in the FCN network (Long et al., 2015). The UNet solved two main problems of FCN by adding long skipped connections between each down-sampling layer in the encoder to its corresponding up-sampling layer in decoder (see Figure 4(b)). First, an issue inherent to down-sampling is the loss of information. Adding skip connections helps to recover fine grain details when up-sampling giving a better resolution in the prediction. Second, multi-layers architecture networks commonly face the problem of vanishing gradient. Indeed, weights are usually taken in [0,1] to insure the back-propagation stability. However, the successive multiplication of positive numbers smaller than one engender an increasingly smaller update of the weights when back-propagating. Skip connections offer an alternative path for the gradient and guarantees its preservation.

Training

The UNet required binary masks of the lanes to be trained. Three hundreds masks were produced by hand using online tool Apeer. To reduce the number of model parameters images input were reshaped to (128, 128, 3) giving an output mask of shape (128,128). Considering the small numbers of training data, data augmentation appeared as a compelling process. The training dataset was increased using ImageDataGenerator from Keras and by performing the following transformations: shear, rotation, zoom, width and height shift. To prevent over-fitting, early-stopping was applied with a patience of 3 epochs.

Performance

The performance of the model was assessed by the Dice coefficient (Milletari et al., 2016). The Dice coefficient is a common metrics in segmentation problem defined regarding the predicted value of the pixel p and its ground-truth value g.

$$D = \frac{2\sum_{i=1}^{N} p_i g_i}{\sum_{i=1}^{N} p_i^2 + \sum_{i=1}^{N} g_i^2} \tag{1}$$

After training, the model obtained a Dice of 0.7185.

Angle Prediction

In the present application, the angle depends on the direction of the lanes and in some cases to an additional direction indication. The direction indication is given by a left or a right signal when the car reaches a crossroad. We present here two implementations of a regression model to predict the angle. The second one consists in an update of the first model for a better integration with the remaining of the architecture.

0.0.1 Nvidia

The Nvidia is an end-to-end model performing angle prediction for autonomous driving (Bojarski et al., 2016). This model offers a complete deep learning solution as the lanes segmentation, the path planning, and control are implicitly encoded by the hidden layers. Therefore the Nvidia proposes a minimal model to optimize each driving tasks. The Nvidia architecture is composed of 5 convolutional layers followed by 4 dense layers. The convolution layers extract the images features then angle regression is performed by the dense layers. Two dropout layers were added to prevent over-fitting.
The preprocessing of the images consists in cropping the up half, resizing to shape (200,66), converting from RGB to YUV and normalising pixel values. These different operations permit to remove irrelevant information from the furthest background, to emphasize the edges contrast and to reduce the number of trainable parameters. Once again, the Keras module of data augmentation was applied to increase the training dataset by modifying brightness, blur, sharpness, noise and orientation. This last step appeared to be an error and was abandoned as a two-ways road isn't symmetric. Therefore an inversed image cannot simply be labelled by an inversed angle. The training gave a loss of 0.0229 for the validation data (mean squared error).

0.0.2 Updated NVIDIA

When considering the remaining of the architecture, features extraction through convolution layers appears as a redundant operation. Indeed, lanes segmentation was needed to discriminate a detected object as an obstacle. The segmented mask offers a low dimension representation of the road reducing variability. Therefore, using lanes mask as input for an updated Nvidia network seems a more efficient use of the available resources. In the updated Nvidia, convolutional layers are withdrawn while one extra dense layer is added. The dimension of the input was assumed sufficiently low to need features extraction. Angle prediction was considered as a different task performed by a different model regarding the presence of a direction signal. Indeed, in presence of direction signal the prediction doesn't only depends on the lanes and represent then a slightly different task. The three models were trained on different datasets that we will refer as Regular, Left and Right. The Regular dataset consists in every pictures from the training data with no direction signals. The two others are logically composed of left signal and right signal images. Images were sorted into the three sets using the object detector YOLO.
The Regular model was first trained on the eponymous dataset. The two other models were fine-tuned on

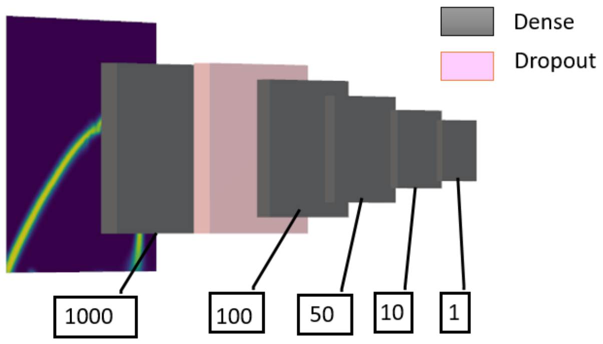


Figure 5: Updated Nvidia architecture.

their respective dataset. The initial weights were taken from the already trained Regular model and the learning rate was decreased from 5e-4 to 1e-5. Table 2 shows strong improvement in the performance when making prediction with the three specialised models instead of a unique one. It has to be noted that the loss was also grandly reduced compared to the previous Nvidia model.

Validation mse		
Domain	Unique model	Specialised models
Regular	0.0077	0.0049
Left	0.0382	0.0154
Right	0.0626	0.0230

Table 2: Performance of the updated Nvidia. Single model trained on a single dataset and Specialised models trained on Regular, Left and Right datasets.

Speed Prediction

Two approaches appeared possible to implement speed prediction: Rules-based or End-to-End. Speed prediction is a rather simple problem regarding the Benchmark. The number of situations determining the speed is limited and the output can only take a binary value. Therefore, the implementation of explicit rules seems a credible approach allowing generalization and small computational cost.

Rules-based approach

The Rules-based approach relies on explicit rules to determine the speed. Different situations can induce the arrest of the car. The procedure is described by the algorithm 1. First the binary mask of the lanes is obtained by the function *LanesDetection* using the UNet network. Second object detection is performed by the YOLO network returning the bounding boxes and the classes of the object. The detected object are then examined by several tests. In presence of a red light the speed has to be set to zero. If there are left or a right signals the boolean *left* or *right* is respectively set to *True*. In case a "person", a "tree", a "car" or a "box" is detected its bounding box is added to the list *obstacle*. Third, the angle is predicted by the *AnglePrediction* function taking as inputs the *lanes_mask* and the booleans *left* and *right*. The prediction is made by one of the three updated Nvidia models depending on the absence or the presence of left or right signals. The function *RoadMask* returns a binary mask of the left-hand road. Fourth, the *ObsctacleDetection* function calculates the intersection of the *road_mask* and the bounding boxes from the *obstacle* list. If the intersection exceeds 20 percent of the bounding box area the associated object is considered as an obstacle and the speed is set to zero. Finally, the interruption of the road is a condition to stop. This last function verifies the presence of lanes on the left, at the top or on the right of the image depending on the future direction of the car.

Algorithm 1 Prediction

```
procedure RULES BASED PREDICTION(image)
    Initialize $speed$ to 1
    Initialize $left$ and $right$ to $False$
    Initialize $obstacle$ as an empty list
    $lanes\_mask$ = LanesDetection(image)
    $bounding\_boxes$, $names$ = ObjectDetection(image)
    for $bounding\_box$, $name$ in ($bounding\_boxes$, $names$) do
        if $name$ is "red light" then
            $speed = 0$
        end if
        if $name$ is "left signal" then
            $left = True$
        end if
        if $name$ is "right signal" then
            $right = True$
        end if
        if $name$ is "person" or "car" or "tree" or "box" then
            Append $bounding\_box$ to $obstacle$
        end if
    end for
    $angle$ = AnglePrediction($lanes\_mask$, $left$ , $right$)
    $road\_mask$ = RoadMask($lanes\_mask$, $angle$)
    if ObstacleDetection($obstacle$, $road\_mask$ ) is $True$ then
        $speed = 0$
    end if
    if EndRoadDetection($lanes\_mask$, $angle$ ) is $True$ then
        $speed = 0$
    end if
    Return $speed$, $angle$
end procedure
```

Most of the aforementioned functions are trivial. We propose here to have a deeper look at the *RoadMask* which realises a more challenging task (algorithm 2). The function scans the *lanes_mask* from bottom to top prospecting for $x1$ and $x2$ at each step (respectively the left and right lanes of the left-hand path). A lane is found whenever the *lanes_mask* value of the x column and the y lane is bigger than a threshold. To insure a depart from the center of the road *x_start* is first conditioned by the angle of the car. *x_start* is then updated at the next step by taking the center of the segment $[x1, x2]$.

Algorithm 2 Road Mask

```
1:  procedure Road Mask(lanes_mask, angle)
2:      $M$, $N$ = shape($lanes\_mask$)
3:      Initialize $mask$ to a zeros array of shape($M$, $N$)
4:      Initialize $y$ to $M$
5:      Initialize $x\_start$ to $N//2$
6:      if $angle \leq 0.35$ then
7:          $x\_start = N//4$
8:      end if
9:      if $angle \geq 0.65$ then
10:         $x\_start = 3N//4$
11:     end if
12:     while $y \geq step$ do $y = y - step$
13:         Find $x1$ in $[0, x\_start]$
14:         Find $x2$ in $[x\_start, N]$
15:         $mask[y - step{:}y\ , x1{:}x2] = 1$
16:         $x\_start = (x1 + x2)//2$
17:     end while
18:     Return $mask$
19: end procedure
```

The Rules-based approach proved to effectively predict the speed taking into account all the possible situations. The accuracy of speed prediction reached 0.98825.

End-to-end approach

The end-to-end approach relies exclusively on a neural network to predict the speed. The selected network was directly inspired from the updated Nvidia with 5 dense layers of decreasing size and 2 dropout layers. The sigmoid function is chosen as the final activation to output a probability to stop. The input is a combination of the lanes mask and the bounding boxes of potential obstacles (see Rules-based approach). This last model was trained using early stopping on 13798 masks produced from the training data. After training, the accuracy was 0.9772 - slightly worst than the Rules-based approach - explaining the abandonment of this model.

Discussion

Performance

Kaggle Competition

Whist the main objective of this project was to pilot an autonomous car, a competition was also hold on Kaggle to measure the performance of the developed model. During 21 days, one submission was allowed to predict both speeds and steering wheel angles of 1020 test images. The presented model obtained a satisfying result giving a mean squared error of 0.02411 equivalent to a eighth over eighteen position on the leader board.

Live testing

The live testing of the car shown a more disappointed performance. An extenuating circumstance would be the late replacement of the YOLO network by the YOLO-tiny after a suggestion from our professor. Still, the inference time was too long to allow a smooth trajectory on the road. Thus, the complexity of the network has to be questioned. Inopportune arrests were also to be noted. This second issue could be easily solved as the YOLO-tiny score threshold is very likely to be in cause. Indeed, the selection of a very lenient threshold induces the detection of fake objects which may engender undesirable predictions.

Motivations of the model

The presented model subdivided the overall task in multiple explicit steps. Different models were implemented to deal respectively with object detection, lanes segmentation, angle prediction and speed prediction. Whilst naive, this strategy allows to control each individual step of the process and thus gives inputs to improve the final predictions. For example, when the object detector shown difficulties to discriminate the "red light" from the "green light" new data of these two classes were added to perform a new training. Also lanes segmentation allowed to control the selection of the interesting features from the input images. Another advantage of a multi-tasks strategy is the low computational cost to train each model. The longer model to train concerned the YOLO and was achieved within a few minutes on Google Colab using a GPU.

Limits of the model

The main limit inherent to a subdivision of tasks is the difficulty to optimize the different models as an ensemble. Indeed, the strength of Machine Learning lies in its capacity to optimize each calculation to obtain the desired output. If the different steps are explicitly defined, the model might be constraint by an architecture which is unlikely to be optimal. Second, dealing with multiple models increases the risk of having redundant tasks. One can legitimately infer that the YOLO network and the UNet are redundant regarding feature extraction by convolutional layers. Redundancy could be acceptable for static predictions but its causes an obvious problem of optimization of inference time for real-time predictions. With regard to the previous considerations a unique end-to-end model as the Nvidia seems more appropriate to offer a minimal solution to our autonomous driving problem.

Ways of improvement

The present work was constructed to deal with a static problem. Each image is considered independently from the other to realize a prediction. Whilst relevant considering the given training dataset, an autonomous driving application calls naturally for a recurrent model. Indeed, a real-life driving problem require to deal with time dependant images. Thus, if the training data respected the temporal sequence of acquisition a different class of models would have been relevant. A compelling model for recurrent problem is the STFCN. The Spatio-Temporal FCN relies on FCN along with LSTM to do video segmentation. Combining FCN and LSTM allows to capture respectively the spatial and the temporal information.

Conclusion

The present project had as an objective to build a model to control a miniature autonomous car. Instead of an end-to-end approach the task was subdivided between five different neural networks and functions. Object detection allowed to detect potential obstacles and traffic signals using the YOLO network. Lhe lanes mask was extracted by the UNet to determine the left-hand road. The steering angle prediction was performed by a regression model using the lanes mask as an input. Finally, the speed was determined in a rules based approach depending on the object detected and their position on the left-hand road. The described protocol shown good performance in making static prediction but appeared to be too complex to fit real-time reactivity requirements. Instead, a unique end-to-end model should be considered to insure an acceptable inference time.

References

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., & Zieba, K. (2016). End to end learning for self-driving cars.

Gaiser, H. (2020). *Keras retinanet.* https://github.com/fizyr/keras-retinanet. Github.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 580-587.

Lin, T. Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017). Focal loss for dense object detection. *In Proceedings of the IEEE international conference on computer vision*, 2980-2988.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 3431-3440.

Milletari, F., Navab, N., & Ahmadi, S. A. (2016). V-net: Fully convolutional neural networks for volumetric medical image segmentation. *In 2016 fourth international conference on 3D vision (3DV)*, 565-571.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 779-788.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. *In International Conference on Medical image computing and computer-assisted intervention*, 234-241.

Sharma, R. (2020). *Yolov3 implemented in tensorflow 2.0.* https://github.com/zzh8829/yolov3-tf2. Github.