Environment perception in racing simulators using Deep Neural Networks

Liviu Marina

Department of Automation
Transilvania University of Brasov
Brasov, Romania 500240
@unitbv.ro

Florin Moldoveanu

Department of Automation
Transilvania University of Brasov
Brasov, Romania 500240
@unitbv.ro

Sorin M. Grigorescu

Department of Automation
Transilvania University of Brasov
Brasov, Romania 500240
s.grigorescu@unitbv.ro

***Abstract*- In this paper, we introduce a real time approach for object recognition, applying the Faster-RCNN concept in an affordable and open source simulator that can be used to train and test Deep Neural Networks (DNN) models. The objective is to provide to the scientific community a framework where the advantages and disadvantages of already published or new designed architectures and concepts for object detection can be validated. The framework proposes two available configurations for implementing DNN algorithms. Our framework provides an interface to the TORCS racing game simulator, where the deployed methods can be evaluated. Various ground truth information can be extracted from TORCS, like the traffic scene image, car position on the track, speed or distances between the traffic participants.**

I. INTRODUCTION

The purpose of an object detection algorithm is to be accurate and fast in real time environments. Even if the available benchmarking datasets are limited [9] compared to the datasets available for classification [10], object detection can be used to detect the features from a common traffic scene (Figure 1).

Object detection is a more complex task when compared to image classification. First, in order to localize the objects accurately, numerous candidates for the object's locations should be generated. In the second step, the candidates are refined to gain the proper localization. Solving this task may lead to issues regarding the processing speed or accuracy.

Nowadays object detection is based on the success of region proposal methods [2], [4] and region-based convolutional neural networks [9], [11]. As originally developed, region-based convolutional networks were computationally expensive. The cost was reduced after the concept of sharing the convolutions across layers was introduced.

In this work, we have developed an object detection method for a free and real time race simulator, widely used in the scientific community, named TORCS. The scope of this research was to provide an open source framework to the scientific community where various neural networks architectures and object detection algorithms can be easily tested.

The Caffe library, developed by Berkeley University of California, was used for training and validating DNN algorithms.

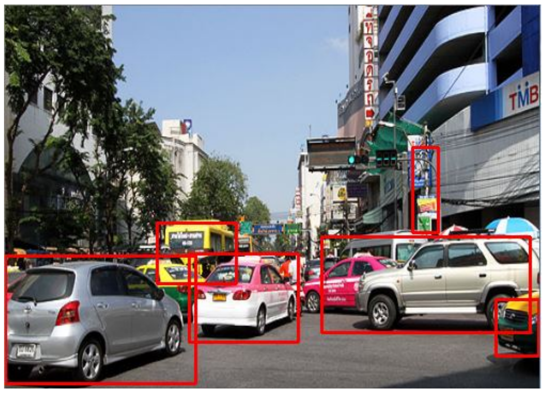

Figure 1: Common traffic scene with detected objects

Caffe is a fully open-source framework that affords clear access to deep neural architectures [12]. The library is written in C++, with CUDA used for GPU computation, supporting bindings for Python/Numpy and MATLAB. It is well-suited for research use, being developed using engineering best practice, providing unit tests for correctness, speed for deployment and code modularity. It also has an expressive architecture that encourages innovation and further enhancements through extensible code base. The most relevant advantage in using this library is the capacity to process aver 60 million images per day, using for example a single NVidia K40 GPU. This property makes Caffe one of the best choices for industry deployment.

A. *Related work*

Object proposals methods were intensively studied. There is a large literature that covers this subject [5], [6], [7]. The most popular methods are those based on grouping super-pixels (e.g. Selective Search [8]) and those based on sliding windows. DNNs are used to classify the proposals regions into object categories or background.

The region-based convolutional neural network method from [3] was able to achieve good accuracy in object detection, with the cost of some notable disadvantages, like an expensive training session in terms of space and time and a slow object detection speed (approx. 50 seconds per image).

Another succesfull approach is Fast R-CNN [2]. It almost achieves real-time rates using large scale deep networks, while ignoring the time spent on region proposals. The proposals were identified as a bottleneck in the detection systems. An advantage of the Fast R-CNN method is that it can run on GPUs, which is the solution to accelerate the proposals computation.

The latest solution for object detection, known as Faster R-CNN [4], introduced a region proposal network (RPN) that shares full-image convolutional features with the detection network. In this way, the time to calculate the proposals was significantly reduced. RPN is a fully, end-to-end trained, convolutional network that simultaneously predicts the object localization and the objects scores. RPN is merged with Fast R-CNN into a single network, sharing their convolutional features. Region proposal network will decide if there is an object or not in the image, and also propose a box location. Even if Faster R-CNN achieves real time rates, until now it was used only to analyze multiple image streams, not being used into a real-time simulator, where the advantages and the weakness of this concept can be highlighted.

The main contribution of the work presented in this paper can be summarized as follows:

- Development of a real-time framework where object detection methods can be applied and tested.
- Improve the exiting techniques by optimizing the computational speed of the region proposal network.

The rest of the paper is organized as follows. In Section II, the formulation of the challenge and the proposed solution are stated. The algorithm to interface the game simulator framework upon the detection system and the mathematical model of the proposed solution is given in Section III. Section IV presents experimental results and the conclusions are stated in Section V.

II. PROBLEM FORMULATION

The goal of this work is to apply object detection methods in a real time simulator were deep neural network algorithms can be validated and improved.

An open-source framework, simulating a car racing environment, is developed and provided to the scientific community to be used for designing new DNN architectures or to test the state-of-the-art achievements in object detection technics. To train and validate our object detection algorithm, both GPU and CPU configurations were used and a comparison between the obtained performances was made.

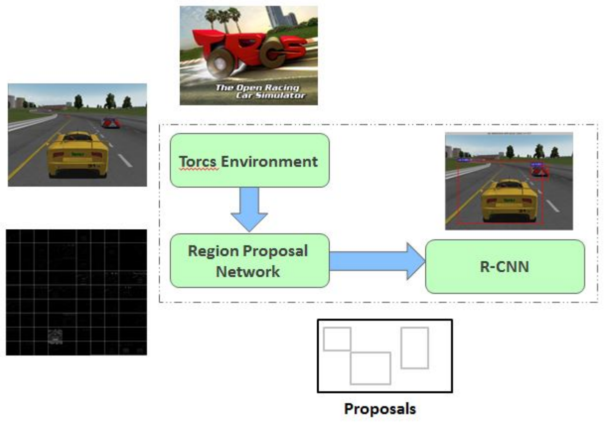

Figure 2: Block diagram of the proposed open-source framework

III. METHODOLOGY

The object detection method that developed in our open-source framework is the combination between a Region Proposal Network (RPN) and the concept of Fast R-CNN (Fast Region-based Convolutional Network) [2] [4]. One advantage of using this architecture is sharing the computation between RPN and the Fast R-CNN network, using a common set of shared convolutional layers. RPN is a fully convolutional network [13] which is designed to predict region proposals with a wide range of scales and aspect ratios. It takes as input an image and provides a set of rectangular objects as output, each with a matching score. The novel “anchor” boxes, which serves as a reference for proposals, was introduced [4]. RPN provides a sliding window that classifies if an object is present or not. This window slides on the output of the last shared convolutional layer of a pre-trained network, named the output feature map, and provides to RPN an *n x n* spatial window with an input convolutional feature map data. Fixed anchor boxes are created and classified as been objects or not for every position of the sliding window.

Fast Region-based Convolutional Network [2] takes as input a set of region proposals and the entire image, which is processed with several convolutional and max pooling layers to produce a feature map. After this step, for each proposed object, a region of interest pooling layer (RoI pooling layer) extracts from the feature map a feature vector. As it was described in [2], a RoI is a rectangular window in a convolutional feature map, being defined by a four-tuple (*r, c, h, w*) that represents the RoI top corner (*r, c*), height (*h*), and width (*w*). The RoI layer is a special-case of the spatial pyramid pooling layer [15].

An important property of Fast R-CNN is that it trains all the network weights with the back-propagation algorithm, using a more efficient method that shares features during training.

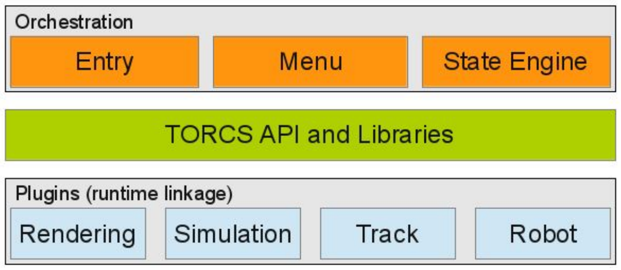

Figure 3: Torcs Architecture Overview

The loss function for an image used for training our object detection algorithm is the following:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \quad (1)$$

where i is an anchor index, p_i is the predicted probability of an anchor i to be an object or not, p_i^* is the ground-truth label, which is 1 if the anchor is positive and 0 if it is negative, t_i is the parametrized coordinates vector of the predicted bounding box, t_i^* is the ground-truth box which is associated with an anchor with a label that equals 1. L_{cls} is the classification loss (logarithmic loss for two classes, object and non-object) and L_{reg} is the regression loss. The outputs of the regression and classification layers are normalized using N_{cls} and N_{reg}. Parameter λ is used to balance the weights computation.

In Figure 2 is illustrated how the game simulator and the object detection system are merged into a single framework. An image with a traffic scene is provided as input for the region proposal network. RPN is trained end-to-end to generate accurate object proposals that are used for detection by a region-based convolutional network.

The image received from Torcs is first converted into an OpenCV array of 500x375 dimensions and further sent to RPN for processing. We developed a multithreading environment to speed up the proposals computation.

After the proposals are being generated, they become an input for the Fast R-CNN algorithm. The objects with the highest matching score are detected and marked on the image.

Due to the fact that Faster R-CNN is developed using C++ (to integrate Caffe) and Python (to develop the layers used for object detection) our framework is also based on a C++ implementation combined with Embedded Python for an easier and faster communication with the Python scripts.

Deep neural network DNN was trained using the approximate joint training solution [4], merging together Fast R-CNN and RPN networks. During each iteration, generated region proposals are forwarded, which are treated like as pre-computed proposals during Fast R-CNN training. Backward propagation combines, for the shared layers, the propagated loss signals. For training purposes, already existing datasets can be used (i.e. Pascal VOC2012 [9] or YOLO [11])

A simplified version our proposed DNN architecture is presented in Figure 6, the feature maps are extracted from the last convolutional layer of a pre-trained network (in our case an ImageNet [14] convolutional neural network model). The region proposal network is trained to decide if there is an object or not in the RoI. The results are sent to a custom Python layer and then the proposals are send to a RoI pooling layer. All the proposals are resized to a fixed size and processed one by one to determine the best matching score for a specific object.

IV. Experimental Results

To evaluate our developed real-time framework for object detection, validation data from the TORCS racing game simulator was used to measure the performance capabilities for the DNN technologies. TORCS is an open-source car racing simulator that will be described in more details in the next section. A multithreading environment was used to perform tests regarding the possible use of the designed framework in both CPU and GPU configurations.

A. TORCS Environment

Our research is using an open source framework that can become a state of the art for deep learning technology. TORCS environment is widely used in the scientific community, being a highly portable multi-platform car racing simulator. It runs on Linux (all architectures, 32 and 64 bit, little and big endian), FreeBSD, OpenSolaris, MacOSX and Windows (32 and 64 bit). It features many different cars, tracks and opponents to race against. From the game engine, we can collect images for object detection and also critical indicators for driving, such as the speed of the car, the ego-car's relative position to the road's central line, or the distance to the preceding cars [17].

The TORCS software architecture identifies three major components, as illustrated in Figure 3: Orchestration for controlling the major program, TORCS API and Libraries for interfaces and Plugins for modules loaded during runtime. The State Engine component controls the execution of the race configuration, setup, run and shutdown. Understanding this component can make some task, like implementing a new artificial intelligent agent in the form of a car, very simple.

The plugins play also an important role in TORCS. Rendering, the graphic module interface, is responsible for displaying the current sate based on OpenGL. Simulation module interface is responsible for progressing the situation by a given time step. Another major plugin is Track (Track Loader Module Interface) which is responsible for loading tracks into Torcs. The game simulator provides also the possibility to create tracks for specific purposes. Also new developed track will be loaded by the Track plugin.

The cars are driven into the simulation using the Robot (Robot Module Interface) plugin. Torcs can load multiple robots at the same time, which can drive multiple cars. One robot supports up to 10 cars at once. Various drivers implementation are already shipped into the simulator.

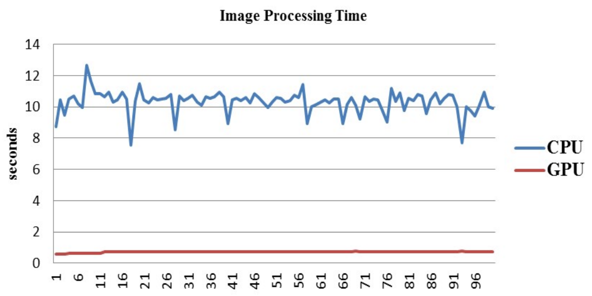

Figure 4: Performance comparison between the CPU configuration and GPU configuration for 100 image samples

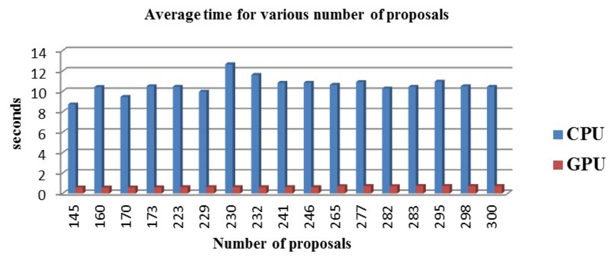

Figure 5: Necessary time for image processing using CPU and GPU configuration for various number on object proposals

Also a “human driver”, which takes input from user to control the car, is present in the simulator.

Deep neural networks technics can be used inside Torcs for image classification [1] or as we did, for object detection. There are some advantages in using this game simulator from which can be highlighted the multitude types of objects that can be detected, an easy way to construct new tracks for specific purposes, the possibility to construct a new artificial intelligent car where new algorithms for path or trajectory planning can be applied and another advantage is also the possibility to easily provide training and validation data to another projects interfaces.

B. *Framework validation*

To validate the proposed framework a multithreading system was developed. Even if the game simulator is multiplatform, the development was done only in Windows environment.

Traffic scenes representing the environment state, are captured from Torcs and are processed in a separate thread to speed up the object detection process. Faster R-CNN method, that was chosen to develop our framework, is based on layers written in a scripting programming language, Python/Numpy.

To validate our work, an interface between Python and the game simulator, which includes also Caffe library and all the necessary dependencies, was developed.

The interface was implemented using Embedded Python language, which assures a manageable binding with Python library. Embedding a scripting language into applications offers many benefits, such as the ability to use functionality from an embedded language that would otherwise be a difficult process.

The computation of region proposals and object detection was made using both CPU and an NVidia GPU, Quadro K1100M. This GPU is supporting CUDA but is coming with a low performance configuration (Core Speed is 705 MHz, 2 GB of shared memory and 384 CUDA Cores) and cannot be compared with the GPUs communally used in research purposes, like Tesla K40 Graphic Card, which provides 12 GB of memory, or NVidia Titan X, which is driven by 3584 NVIDIA CUDA cores running at 1.5GHz. The most powerful NVidia GPU is Quadro P6000, on which 3840 CUDA cores are enabled and provides 24 GB of memory.

The architecture used for deep neural network is provided in Figure 6 and it was described in the Methodology section of this paper. To train the network, Pascal VOC2012 [9] dataset was used, which contains sufficient data to achieve good accuracy for up to 21 different object classes. In this paper we have trained the network for all 21 classes, even if the main focus is to detect the cars present on track during the race.

To be able to make a comparison between CPU and GPU approaches, performance statistics were done using the time needed to detect the objects from an image, for a specific number of proposals, time which was calculated inside the game simulator.

Using a CPU configuration is not possible to achieve real-time performance. Even with a powerful processor (Intel(R) Core(TM i7-4710MQ CPU @ 2.50 GHz, 8CPUs a considerable amount of time is needed to process an image and to detect the objects. In our tests the average time for

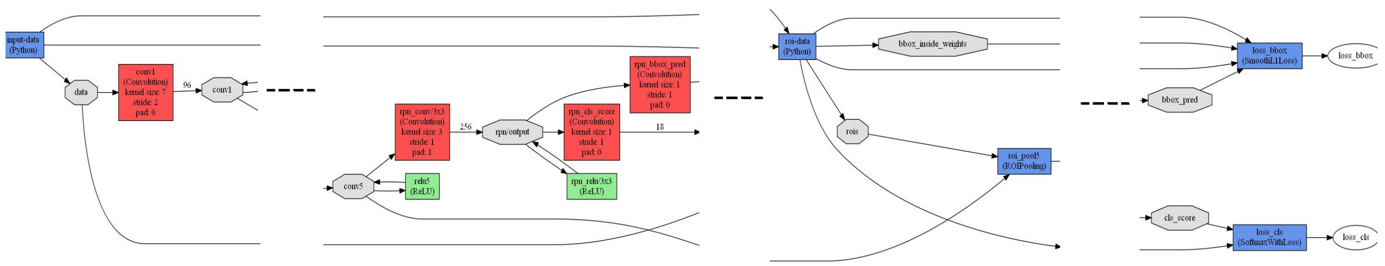
input-data (Python)
data
conv1 (Convolution) kernel size: 7 stride: 2 pad: 0
96
conv1
rpn_conv/3x3 (Convolution) kernel size: 3 stride: 1 pad: 1
256
rpn/output
rpn_cls_score (Convolution) kernel size: 1 stride: 1 pad: 0
rpn_bbox_pred (Convolution) kernel size: 1 stride: 1 pad: 0
18
conv5
relu5 (ReLU)
rpn_relu/3x3 (ReLU)
roi-data (Python)
bbox_inside_weights
rois
roi_pool5 (ROIPooling)
loss_bbox (SmoothL1Loss)
loss_bbox
bbox_pred
cls_score
loss_cls (SoftmaxWithLoss)
loss_cls

Figure 6: Simplified convolutional deep neural network architecture used for object detection.

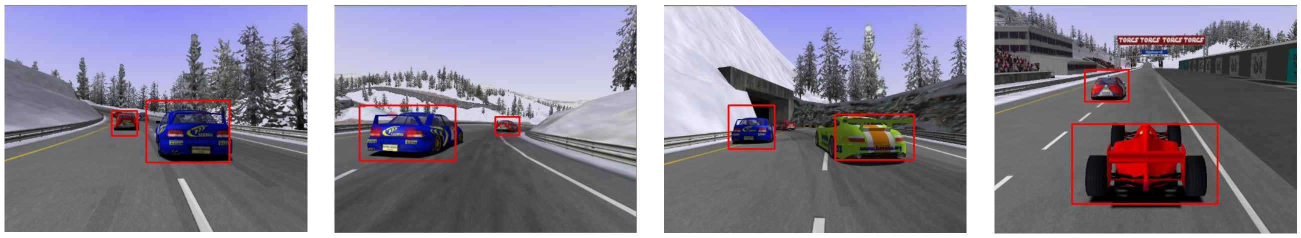
Figure 7: Samples of images with detected objects inside Torcs.

processing was approximately 10 seconds. The time necessary to detect the object is depending also on the number of the proposals. In Fig.5 can be seen that the time value is the smallest when we have less proposals. But the variation is not linear, because, as can be observed, for 230 proposals the necessary time is higher than for the maximum number of proposals (in our case, 300).

In Figure 4 is represented the performance difference between GPU and CPU configurations. Using a GPU for object detection will decrease considerably the computation time. Even with a low performance configuration of the GPU, we get a very high detection rate, comparing with the values recorded for CPU mode. The average time to process an image was reduce to an average of 0.7 seconds. Taking in consideration this result, we cannot say that we achieve to make real-time object detection. The performance is limited by our hardware configuration, for train and validation was used an onboard GPU with 2GB of shared memory. If we will take in consideration the performance difference between our GPU and the ones used in common DNN research papers (as it was stated at the beginning of this chapter), we can observe that our GPU performance is minimum 10 times smaller.

V. Conclusion

The proposed open-source framework can be used to validate various types of neural deep network architectures using Caffe as development library. Even if this paper was based on object detection algorithms that were applied on images sampled from Torcs, also the framework can be used to solve classification problems, only by changing the deep neural network architecture. Even in this work no real-time performance was achieved during the tests, due to the hardware configuration limitations, a more powerful GPU will enable the usage of this framework with success in real-time environments. We believe that our work will be very useful for the scientific community which is interested in developing machine learning algorithms, because is providing a complete, Windows based, framework for training and validation. Most of the scientific researches were done under Linux platforms, using only video streams or basic game environments (i.e. Atari games [17]).

Torcs represents a complex car racing environment that is making possible a future industrialization of the algorithms developed inside it, to be applicable in the real world.

Our framework can be also improved to get a higher computational speed, by using C++ language to rewrite the layers and modules that in this moment are implemented in Python language or can be adapted to use a new approach in object detection, that is describe in [16].

Acknowledgment

This paper is

References

[1] C. Chen, A. Seff, A. Kornhauser and J. Xiao, *DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving*, *in IEEE Conference on Computer Vision and Pattern Recognition (CVRP), 2015.*

[2] R. Girshick, "Fast R-CNN," *in IEEE International Conference on Computer Vision (ICCV),* 2015.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation", *IEEE Conference on Computer Vision and Pattern Recognition (CVRP), 2014.*

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *in IEEE Conference on Computer Vision and Pattern Recognition (CVRP), 2015.*

[5] J. Hosang, R. Benenson, and B. Schiele, "How good are detection proposals, really?" in British Machine Vision Conference (BMVC), 2014.

[6] J. Hosang, R. Benenson, P. Dollar, and B. Schiele, "What makes for effective detection proposals?", *in IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 2015.*

[7] N. Chavali, H. Agrawal, A. Mahendru, and D. Batra,"Object-Proposal Evaluation Protocol is 'Gameable',"*arXiv:1505.05836, 2015*

[8] J. R. Uijlings, K. E. van de Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition", *in International Journal of Computer Vision (IJCV), 2013.*

[9] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, *in The pascal visual object classes 9voc0 challenge. International Journal of computer vision, 88(2):303-338, 2010*

[10] B. Thomee, D.A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.J. Li. Yfcc100m: The new data in multimedia research, *in Communications of the ACM, 59(2):64-73, 2016.*

[11] J. Redmon, A. Farhadi. YOLO9000: Better, Faster, Stronger**,** *IEEE Conference on Computer Vision and Pattern Recognition (CVRP)*, 2016.

[12] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, Trevor Darrell. "Caffe: Convolutional Architecture for Fast Feature Embedding"

[13] J. Long, E. Shelhamer, and T.Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015.

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, and L. Fei-Fei, ImageNet : A large-scale hierarchical image database, in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009

[15] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition, *in ECCV, 2014.*

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, “Playing Atari with Deep Reinforcement Learning”, *in NIPS Deep Learning Workshop 2013*

[17] http://torcs.sourceforge.net/