Lessons learned from lightweight CNN based object recognition for mobile robots

Conference Paper · May 2018

DOI: 10.1109/AQTR.2018.8402778

2 authors, including:

Levente Tamas
Universitatea Tehnica Cluj-Napoca
78 PUBLICATIONS **362** CITATIONS

SEE PROFILE

Lessons Learned from Lightweight CNN Based Object Recognition for Mobile Robots

Andrea-Orsolya Fulop and Levente Tamas

***Abstract*— The focus of this paper is on the comparison of multiple neural network frameworks and the their usage in 2D/3D robot perception applications. Numerous frameworks exists for this task including the recent deep learning based ones, which allow us to develop a perception system, with the chosen parameters for object recognition. In this paper we analyzed the possible solutions, including different Convolutional Neural Networks (CNNs) variants. The advantages of 2D CNNs linked with 3D features lead to another approach, which can be extended further. The leading idea is to create a custom object recognition method that takes advantage of a 2D system's precision and speed, but it can efficiently incorporate 3D features. This way, the disturbances specific to each method separately can be minimized. On the other hand, this is a lightweight solution, that is supposed to be tolerated by less powerful processing units as well. By placing 3D bounding boxes around detected objects, the convenience of the 2D detection methods can be integrated in a 3D metric world.**

I. Introduction

A. Motivation

Our target application was an object recognition system based on 2D CNNs [**?**] and 3D pose estimation. The combination of 2D state-of-the-art systems with 3D features carries great possibilities [**?**]. Clearly, in order to use this technique, not only the RGB images are needed, but also their corresponding depth images. We implemented a system, that does not rely entirely on 2D or 3D features, rather combines them and extracts the most important segments. This system is able to provide good results even when the available computational resources are limited. Since our main goal was to use it on mobile robots, the processing power cannot be compared to a computer's GPU, such as in case of dedicated servers.
We also tested how do different neural network frameworks perform when the same, lightweight dataset is used. The comparison highlights the advantages and drawbacks of each, regarding the domain of object recognition for mobile robot applications.

B. Problem description

This system helps a mobile robot to recognize custom objects in its environment. Since the capabilities of such a robot are limited, the solution needs to be constructed in such a way that it is supported by a modest hardware. An example for such a hardware is a Raspberry Pi 3 model B with a Intel Movidius Neural Compute Stick. The stick supports Caffe and TensorFlow, and thanks to its reduced size the usage is comfortable even for lightweight systems, such as mobile robots.
In order to get started, a basic knowledge is essential about the functionality of neural networks, convolution, image processing techniques and mobile robot programming. However, the learning curve for these non-parametric estimation techniques is rather steep, i.e. in relatively short time custom solution can be achieved in developing custom object recognition tasks.

Technical University of Cluj-Napoca, Romania, Robotics and Nonlinear Research Group, contact: Levente.Tamas@aut.utcluj.ro. L. Tamas is partially affiliated with the Department of Electrical Engineering and Information Systems, University of Pannonia, Veszprem, Hungary

C. Related work

The concept of neural networks and deep learning gains more territory each year in computer vision and robotics, thus there is a rich literature on frameworks related to this technology. Placing 2D bounding boxes around detected objects is a common feature for most detection systems. In this paper we focus on three major frameworks, namely Darknet [**?**], Caffe [**?**] and TensorFlow [**?**].
Convolutional Neural Networks are specific feed-forward networks, that perform extremely well on visual recognition tasks. Since object recognition belongs to this field, it is convenient to take advantage of this characteristic. The most representative networks that are worth to mention are Region-based Convolutional Neural Networks (R-CNN) [**?**], Fast R-CNN [**?**], You Only Look Once (YOLO) [**?**], Faster R-CNN [**?**] and Region-based Fully Convolutional Networks (R-FCN) [**?**]. All of the above require a powerful GPU, at least for the training phase. In order to have a decent, preferably real-time performance, it is recommended to use GPU during the deployment process as well.
Using the Pascal VOC 2007 dataset, the performances are compared in Table I.

Method	**mAP**	**Rate**
YOLO	63.4%	45 fps
YOLOv2	76.8%	67 fps
R-CNN	58.5%	3 fps
Fast R-CNN	70.0%	-
Faster R-CNN (VGG-16)	73.2%	5 fps
R-FCN (ResNet50)	77.4%	8-11 fps
R-FCN (ResNet101)	79.5%	5-8 fps

TABLE I: Performance comparison of neural networks for object recognition based on mean average precision (mAP) and frames per second (fps).

Darknet [**?**] is an open source neural network framework, and it uses YOLO, a state-of-the-art object detection system. Its performance depends heavily on the machine's GPU, but under optimal circumstances it performs real-time, meaning 40-90 fps. It makes use of a single neural network to the whole image, then divides into multiple regions and predicts the locations, using bounding boxes and probabilities. The predictions are influenced by the environment of the detected object. Since it uses only one network evaluation, its speed exceeds multiple times the speed of R-CNNs or even Fast R-CNNs. Recently they introduced YOLOv2[**?**], which provides some additional improvements over the former variant.
Caffe [**?**] is a deep learning framework developed by Berkeley AI Research. It mainly focuses on image processing, and it is not designed for other deep learning application such as text or sound recognition. The previously mentioned networks, namely R-CNN, Fast R-CNN and Faster R-CNN can all be integrated and effectively used with Caffe.
TensorFlow [**?**] is probably the most widely used open source deep learning framework. It was developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization. It is a flexible and adaptable system, that can be useful on many fields of interest.

II. Object recognition

A. Overview

Object recognition allows robots or any kind of AI program to identify objects from a given camera input stream. This input stream is usually provided by a camera, and the optimal outcome is a real-time detection.
The previously mentioned networks [**?**], [**?**], [**?**], [**?**], [**?**], [**?**] have impressive performance on large datasets, but using only 2D images has its own drawbacks. There is a constant battle between speed and precision, so in order to achieve a satisfying result compromises are needed. Speed has to be sacrificed for a more precise detection and vice versa. Furthermore, it provides coordinates only in 2D. Robots usually need 3D coordinates in order to interact with an object or recognize and locate them on a map.

B. 2D

The first objective is the preparation of a sufficiently rich dataset, meaning that it contains satisfactory amount of data, in this case images. The resolution of the pictures should be adequate, neither too high, nor too low, and it is essential that the overall quality is acceptable.
Usually a preliminary processing and filtering is recommended in order to emphasize the contour of the objects and to reduce the images' size. Labels need to be assigned to each class, so the identity of the detectable objects from the images can be "learned". The dataset is divided in two parts: a bigger one for training and a smaller for validation. The structure of the neural network is then determined. Since CNNs achieved the best performance in this domain, their usage is highly recommended. The desired amount of convolutional, pooling and fully-connected layers, respectively the activation functions need to be specified.
The network treats the images as matrices filled with pixel values. During the training process filters are applied on the images, that are updated after each iteration. As the error decreases, the filters become more and more accurate. Once the training process is finished, the filters can be used on new images or even on a camera stream.

C. 3D

3D state-of-the-art detection methods have an increased runtime and need way more processing power than their 2D counterparts. Using point clouds acquired from RGB-D sensors, semantic segmentation and training is a costly process, wearing down even powerful GPUs.

A major challenge in the 3D object detection part relies in the part segmentation: the separation of the background and region of interest [**?**]. For this a useful technique is the use of the 2D bounding boxes as hints in the 3D point cloud segmentation part, thus speeding up and making more robust the recognition pipeline.

This approach assumes that the color and depth cameras are relatively calibrated. In our case we used a Kinect like camera, for which this constraint is satisfied. According to our previous investigations on the feature 3D based object recognition robustness [**?**] we chose the viewpoint feature histogram (VFH) based variant for its speed and robustness against sampling noise. The disadvantage of this approach, i.e. the sensitivity to the boundary segmentation was compensated with the use of pre-segmentation based on the 2D bounding boxes determined by the 2D detection algorithm.

D. Experimental result

Test conditions: The tests were conducted on a Dell laptop, with Nvidia GeForce GTX 1060 6GB graphics card and Intel Core i7-7700HQ CPU @ 2.80GHz x 8 processor, using Ubuntu 16.04 LTS, 64-bit OS, CUDA 9.0, cuDNN 7.0.5, TensorFLow 1.6.0, Caffe 1.0.
Our experiments began by the comparison of three frameworks, and several networks. In order to achieve the best accuracy, the same custom dataset was used for each of them. This dataset was created by us, using a Kinect camera and it contains 4 object classes, including pictures about objects from several angles.
In the case of Darknet, the modified YOLOv2 network was used. For this, the data preparation takes more time, since the images must be labeled. This label contains the class number, and the position of the object relative to the image. This ensures not only the recognition, but also the right placement of the bounding boxes during detection. These bounding boxes are weighted by the predicted probabilities. The configuration files include the structure of the network and the parameters that need to be set. The training executed until the average loss was stagnating around 0.08. After 30.000 iterations, the model reached its present form. Figure 1 presents the visual results from the recognition process. CPU is adequate only if pictures are used in order to

test the model, but for real-time performance CUDA is recommended.

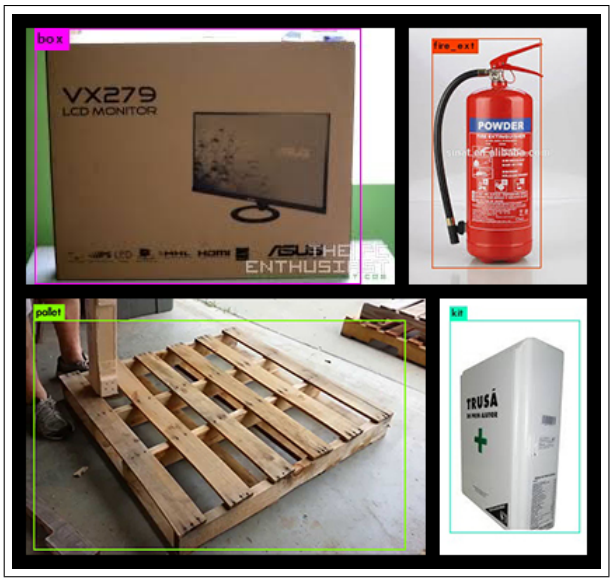

Fig. 1: Examples of detection of the objects of interest, after applying the trained model, using Darknet [**?**] and the modified YOLOv2 [**?**] network.

For Caffe, the labels for each class were assigned in the program. The network used is a modified Alexnet [**?**]. The process started with histogram equalization, resizing, division in two parts then storage in LMDB database. With the help of already implemented Caffe tools, the mean image of the training data can be generated. The structure of the network is defined in a prototxt file, specific to Caffe. The solver, responsible for model optimization, contains the base learning rate, the learning rate policy, step size, maximum number of iterations and more advanced parameters, that can be chosen individually. The switch between CPU and GPU works seamlessly, but the process used with CPU is almost ineffective. During training with GPU, the loss quickly stabilized around 0.38. Figure 2 shows the evolution of the accuracy and loss after the first 10000 iterations.

TensorFlow was the only framework that showed a decent performance even when only CPU was used. In order to have more common points for comparison, the network is a modified Alexnet in this case too. With respect to the length of the code, this solution included the largest amount of it. The labels were assigned in the program and the configuration of the parameters does not require a separate script. Since the entire purpose of TensorFlow is the usage of computational graphs, it can be executed much more efficiently than simply in Python. Once the training process reached the maximum given number of iterations it stopped, and the average loss stabilized around 0.39. The test shows the results in Figure 3 after 10000 iterations.

The results for each method were exported, compared and

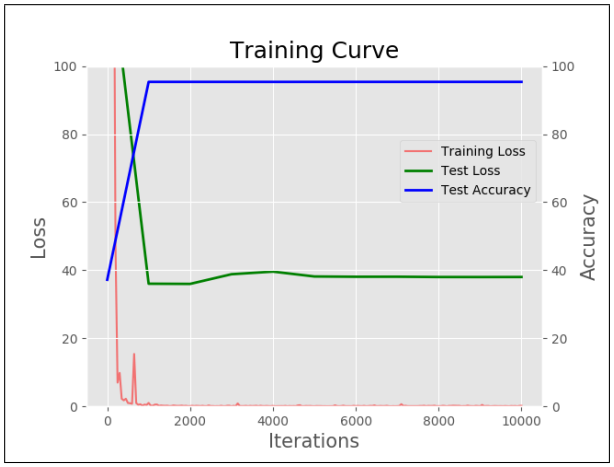

Fig. 2: The evolution of the training parameters during the first 10000 iterations. Red - training loss, green - test loss, blue - test accuracy (best viewed in color).

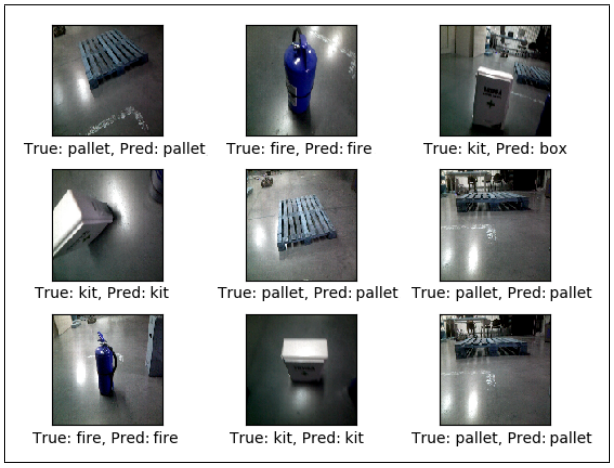

Fig. 3: The correct and predicted labels of nine, randomly chosen objects from the list. True - real label, Pred - predicted label.

the final comparison can be found in Table II. It presents the performance recorded on the test set. Each model was used on the same dataset, and the precision is recorded for each class separately, then the average is calculated. The results are based on a small test set, that contains a total of 40 images, 10 for each object class. Each percentage shows the number of correct detections over the total number of images for each object class separately.

Class	**Darknet**	**Caffe**	**TensorFlow**
Box	60%	80%	80%
Fire extinguisher	100%	40%	40%
Pallet	60%	80%	70%
First-aid kit	90%	70%	60%
Total	**77.5%**	**67.5%**	**62.5%**

TABLE II: The results of the final comparison, for each class separately and the average of detection rate.

Layer 1		Layer 2		Layer 3		Layer 4		**Results**		
Size	Activation f.	Size	Activation f.	Size	Activation f.	Size	Activation f.	Class 1	Class 2	Average
2	relu	2	relu	-	-	-	-	92%	94%	93%
2	sigmoid	2	sigmoid	-	-	-	-	85%	57%	71%
2	tanh	2	tanh	-	-	-	-	91%	88%	89.5%
2	relu	4	relu	-	-	-	-	95%	94%	94.5%
2	tanh	4	tanh	-	-	-	-	100%	75%	87.5%
4	relu	4	relu	-	-	-	-	99%	77%	88%
4	relu	8	relu	-	-	-	-	99%	82%	90.5%
4	tanh	8	tanh	-	-	-	-	99%	69%	84%
8	relu	8	relu	-	-	-	-	99%	91%	95%
8	tanh	8	tanh	-	-	-	-	99%	83%	91%
8	relu	16	relu	-	-	-	-	97%	95%	**96%**
16	relu	16	relu	-	-	-	-	98%	94%	**96%**
16	tanh	16	tanh	-	-	-	-	96%	88%	92%
16	relu	32	relu	-	-	-	-	99%	89%	94%
8	relu	8	relu	8	relu	-	-	98%	83%	90.5%
8	tanh	8	tanh	8	tanh	-	-	95%	96%	95.5%
8	relu	8	relu	16	relu	-	-	96%	95%	95.5%
8	tanh	8	tanh	16	tanh	-	-	99%	86%	92.5%
8	relu	16	relu	32	relu	-	-	93%	98%	95.5%
8	tanh	16	tanh	32	tanh	-	-	99%	85%	92%
16	relu	16	relu	16	relu	-	-	98%	95%	**96.5%**
16	tanh	16	tanh	16	tanh	-	-	100%	78%	89%
32	relu	32	relu	32	relu	-	-	98%	88%	93%
32	tanh	32	tanh	32	tanh	-	-	99%	70%	84.5%
32	relu	32	relu	64	relu	-	-	96%	98%	**97%**
32	relu	32	relu	64	relu	-	-	100%	79%	89.5%
32	tanh	64	tanh	32	tanh	-	-	97%	90%	93.5%
2	relu	4	relu	8	relu	16	relu	96%	93%	94.5%
2	tanh	4	tanh	8	tanh	16	tanh	100%	88%	94%
32	relu	64	relu	128	relu	256	relu	96%	94%	95%
32	tanh	64	tanh	128	tanh	256	tanh	97%	95%	**96%**

TABLE III: The most successful network architectures and their accuracy for two object classes.

It is important to mention, that the final conclusion cannot be drawn by taking into consideration only these results. The created code can be modified, and the network structure can be changed in order to achieve a better performance. This comparison puts accent on computer vision and CNNs respectively. Darknet and YOLO were created only for this purpose, thus professionally optimized, including out-of-the-box features. Caffe has some useful built-in tools as well, but since our dataset is a custom one and consists of a limited number of pictures, its accuracy is poorer. The situation is similar to TensorFlow, where a bigger dataset would result in better precision. Also, for the latter everything needs to be written manually. Without a high-level API, the code becomes unnecessarily long and repetitive. On the other hand it is the most flexible and versatile variant. This fact was proved to be convincing enough to continue the research in order to find a more efficient network structure. Using Keras [**?**] with TensorFlow backend, the problem of the long-drawn code is eliminated. We created additional datasets consisting of a higher number of images in order to have a better insight of its influence on the performance. During the process, multiple factors were taken into consideration, and their influence on the results was recorded. The tested activation functions are the three most used ones, namely ”relu”, ”sigmoid” and ”tanh”. The number of layers varies between 2 and 4, and the number of neurons on some layers goes up to 256. The convolutional kernels are the size of 3x3. The most successful combinations are listed in Table III. Since one layer is not sufficient, this possibility is not included in the table. For the sake of simplicity and time management, we used only two object classes in order to find the best architecture. The training phase lasted for 30 epochs and the fully connected layer consists of 500 neurons in each case. Multiple combinations were eliminated, because the model did not converge properly.

The most efficient networks were then tested on the original dataset of four object classes, and on another dataset consisting of 20 different objects. The results of both tests are shown in Table IV. In the case of the original dataset, the results show a powerful increase in comparison to the previously presented Alexnet structure. However, when more objects are added, a significant decrease in precision can be observed. This can happen due to multiple facts

	Layer 1		Layer 2		Layer 3		Layer 4		4 classes		20 classes	
	Size	Activation f.	Size	Activation f.	Size	Activation f.	Size	Activation f.	30 ep.	100 ep.	30 ep.	100 ep.
1	32	relu	32	relu	64	relu	-	-	77.5%	75%	46.5%	48.1%
2	16	relu	16	relu	16	relu	-	-	72.5%	75%	46.25%	47.1%
3	8	relu	16	relu	-	-	-	-	62.5%	85%	44.8%	45.1%
4	16	relu	16	relu	-	-	-	-	80%	72.5%	46.45%	44.7%
5	32	tanh	64	tanh	128	tanh	256	tanh	35%	64%	52.55%	53.75%

TABLE IV: The results of the multi-class implementation for the most efficient networks taken from Table III, tested on the original dataset (4 classes) and on a more extended one (20 classes).

regarding the relevance of the training and test datasets, the significantly higher number of object classes or other disturbing effects. As a potential deployment use-case, the model was also tested on the Intel Movidius Neural Compute Stick. It is a small scale device, that supports Caffe and TensorFlow and allows the tuning and deploying of CNNs on low-power appliances. The stick comes with a SDK, which offers an API, tools and examples as well. It creates an internal compiled format based on the desired input, which can be used later on the deployment device. The API provides software for connection establishment with the stick, loading the previously created graph and running detection on it. Its performance was proved to be satisfactory for such a small scale device. It is able even for real-time detection with a low power consumption/size constrained applications such as in case of autonomous rovers.

III. Conclusions and future work

In this paper we summarized our lessons learned from the functionality of neural networks, especially CNNs, and the different approaches from three main frameworks in the current state of the art in this research field. The main focus was on the implementation of 2D and 3D object recognition techniques using a Kinect like depth camera for mobile robots in indoor environment. A demo code and video showing the results of this investigation is available on the website of the authors.

In the future we plan to extend the current approach with a depth-net hypothesis verification in the recognition pipeline in order to speed up and run the recognition relying solely on CNN structures. Also the we intend to use a life long learning phase of the algorithm, i.e. to extend the already learned set of features with the new ones during the robot exploration.

Acknowledgement

This work was partially supported by the Romanian National Authority for Scientific Research and Innovation under Grant number PN-III-P2-2.1-BG-2016-0140, PN3-CI125 Grant and Hungarian Research Fund Grant K 120367 and MTA Bolyai Scholarship.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[2] J. Lahoud and B. Ghanem, "2d-driven 3d object detection in rgb-d images," in *2017 IEEE International Conference on Computer Vision (ICCV)*, vol. 00, Oct. 2017, pp. 4632–4640. [Online]. Available: doi.ieeecomputersociety.org/10.1109/ICCV.2017.495

[3] J. Redmon, "Darknet: Open source neural networks in c," http://pjreddie.com/darknet/, 2013–2016.

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[6] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. [Online]. Available: http://arxiv.org/abs/1311.2524

[7] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. [Online]. Available: http://arxiv.org/abs/1504.08083

[8] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[9] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. [Online]. Available: http://arxiv.org/abs/1506.01497

[10] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," *CoRR*, vol. abs/1605.06409, 2016. [Online]. Available: http://arxiv.org/abs/1605.06409

[11] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," *arXiv preprint arXiv:1612.08242*, 2016.

[12] C. Militaru, D. Mezei, and L. Tamas, "Object handling in cluttered indoor environment with a mobile manipulator," in *AQTR 2016: International Conference on Automation, Quality and Testing, Robotics*, May 2016.

[13] L. Tamas and B. Jensen, "Robustness analysis of 3d feature descriptors for object recognition using a time-of-flight camera," in *Control and Automation (MED), 2014 22nd Mediterranean Conference of*. IEEE, 2014, pp. 1020–1025.

[14] F. Chollet *et al.*, "Keras," https://keras.io, 2015.