Type of Presentation: *Oral*

Topic: Measurement in Robotics

Dense convolutional neural network for the line shape prediction in line following robot application

M. Chovanec [1], **K.Jasenčáková** [1]
[1] University of Zilina, Faculty of management science and informatics,
Univerzitná 8215/1, 01026 Žilina, Slovakia
E-mail: michal.chovanec@fri.uniza.sk, katarina.jasencakova@fri.uniza.sk

Summary: In this paper, we present dense convolutional neural network implementation for real-time sensors processing data on a line-following robot. We trained network for line shape prediction. The network is estimating the optimal speed for the line following robot – braking in curves, accelerating on the straight line. Our implementation can run with limited resources, as small microcontrollers ARM Cortex M without FPU. A convolutional neural network with 5 layers and 8x8 input can run in real time, with response 2..3[ms] on ARM Cortex M4 72MHz processor.

Keywords: deep learning, robotics, DenseNet, line follower, ARM Cortex, STM32, AIoT

1. Introduction

Growing deep learning applications is not the only domain of computing on GPU (PC) [1]. In recent years, we can see trends in using machine learning on small devices such as smart phones or tablets. Nowadays microcontrollers (ARM Cortex M0 .. M7) have enough computing performance to run convolutional neural networks [2][3].

The recent solutions depend on a platform, e.g. current solutions for embedded neural networks from main microcontrollers manufacturers [4][5].

From deep neural networks, there are often only obsolete architectures supported such as AlexNet and computational inefficient kernels 5x5 or 7x7 [6].

We decided to implement platform-independent framework, supporting state of the art convolutional neural networks – DenseNet [7]. It requires fewer kernels than older architectures.

Optimization of this neural network consists of several steps, for instance 8 bits discretization and kernels with sizes 1x1, 3x3 and 3x1. We also implemented just 2x2 and 2x1 poolings and only ReLU activation function. These ideas were presented in the VGG network [8].

This network can process data directly on the edge (or embedded) device and provide new functionality for IoT and embedded applications [9] – Artificial Intelligence of Things (AioT).

To demonstrate the functionality of the network, we chose the application of the line following robot (**Fig. 1**). The neural network is predicting curve shape and determining maximal permissible velocity in this application. The robot can go faster on a straight line and brakes on curves.

The network can run in real time, with response 2..3[ms] on ARM Cortex M4 processor (STM32F303, 72MHz [10]). For high speeds of the robot, it is necessary to sample line with a period at least 5[ms].

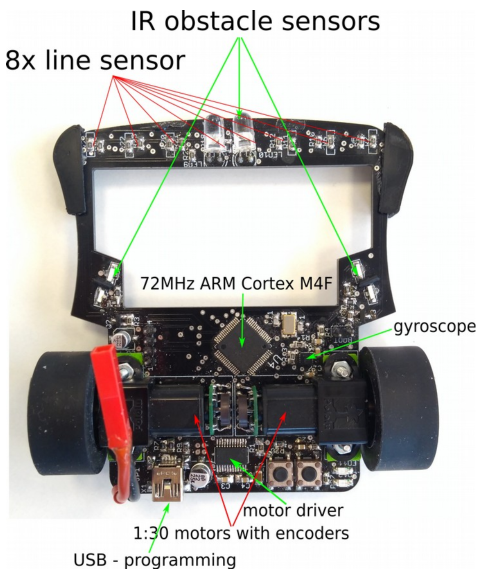

Fig. 1. Line following robot

2. Methodology

The robot is equipped with 8 line sensors (phototransistors with maximum sensitivity at 550nm). Input into the neural network consists of last 8 sensors measurements – sampled in equidistant distance 10mm. The network sees matrix 8x8, as you can see on **Fig. 2**.

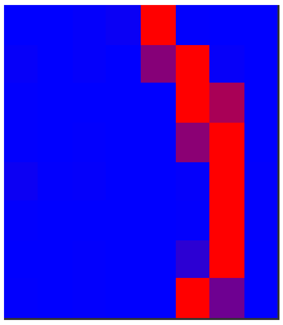

Fig. 2. Neural network input example

Applied microcontroller uses 12bits AD converter. To perform neural network optimization, we normalized data into the interval <0, 127>. During training process on GPU is used 32 float and inputs are normalized into the interval <0, 1>.

There are 5 outputs from neural network for different curves shapes classification: sharp left, left, straight, right, sharp right. Outputs are normalized in the same way as inputs.

We used 40 000 samples for training, and 5000 for testing. In this application, it is non trivial to collect and annotate real data. However it is easy to create artificial data, corresponding to sensors characteristics and including all necessary curves shapes. We also added two types of noise into the data. White noise is from interval the <-0.2, 0.2>, which simulates surface defects and sensors noise. Another noise is luma noise. It is random constant offset for all sensors to simulate different light conditions. This noise is from the interval <0, 1>.

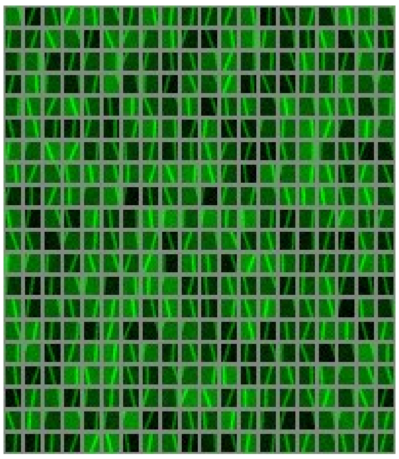

Fig. 3. Training dataset examples

We tested four different neural network architectures, see the **Table 1**. Architectures were chosen as a compromise between performance and quality.

Table 1. Network architectures.

layer	net 0	net 1	net 2	net 3
0	conv 3x3x4	conv 3x3x6	conv 3x3x4	conv 3x3x6
1	max pooling 2x2	max pooling 2x2	max pooling 2x2	max pooling 2x2
2	dense conv 3x3x4	dense conv 3x3x6	dense conv 3x3x4	dense conv 3x3x6
3	FC 5	FC 5	dense conv 3x3x4	dense conv 3x3x6
4			FC 5	FC 5

3. Experimental results

We trained networks with the same hyperparameters : epoch count 10, learning rate 0.001, minibatch size 32 and small 5% dropout on last layer input. We used ADAM solver with Xavier weights initialization and zero bias. Training took only a few minutes on GPU GTX940. We didn't observed a problem with overfitting. Results on training and testing sets differ less than 1%.

In testing, we focused on networks classification accuracy and computational costs. The **Table 2.** represents the classification quality and the computational costs. We chose Network 2, which not only achieves good quality results but also low computing complexity for our robot. Confusion matrix of this network is shown in the **Table 3**.

Table 2. Networks classification results.

network	net 0	net 1	net 2	net 3
success [%]	96.88	97.22	97.32	97.5
FLOPS	7754	13354	13578	25546

Table 3. Net 2 confusion matrix

class	0	1	2	3	4
0	925	14	0	0	0
1	15	1001	17	0	0
2	0	20	949	13	0
3	0	0	19	998	21
4	0	0	0	15	993
class sucess[%]	98.404	96.715	96.345	97.271	97.929
total success[%]	97.32				

The misclassification errors are only on neighboring curves shapes. A video of the working robot is available on [11]. We also provide source code of presented neural network on [12] under GNU GPL licence.

4. Conclusions

In this paper, we demonstrated the ability to run DenseNet neural network on small ARM Cortex M4 microcontroller.

Using discretization to 8 bits (instead of floating point), C++ templates, loops unrolling and SIMD instructions are networks able to run in real time, with response time 2..3[ms].

For line curve prediction application, the quality of classification is around 97% with 13 000 math operations costs.

In the near future, we can expect growing applications of deep learning in embedded devices not only in research and industry domain but also in customers electronic - Artificial Intelligence of Things.

References

[1]. Romuald Josien, GPU COMPUTING TRENDS, 2018, (http://ubee.enseeiht.fr/TSDL/pages/TALKS/TSDL2018-Romuald-Josien.pdf)

[2]. Cotton, Nicholas & Wilamowski, Bogdan & Dundar, G. (2008). A Neural Network Implementation on an Inexpensive Eight Bit Microcontroller. 12th International Conference on Intelligent Engineering Systems - Proceedings, INES 2008. 109 - 114. 10.1109/INES.2008.4481278.

[3]. Jovan Ivković, Analysis of the Performance of the New Generation of 32-bit Microcontrollers for IoT and Big Data Application, 7th International Conference on Information Society and Technology ICIST 2017

[4]. STM32 solutions for Artificial Neural Networks, (https://www.st.com/content/st_com/en/stm32-ann.html)

[5]. Artificial Intelligence and Machine Learning, (https://www.nxp.com/applications/solutions/internet-of-things/ai-machine-learning:MACHINE-LEARNING)

[6]. Alex Krizhevsky, imageNet Classification with Deep Convolutional Neural Networks, 2012, (https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf)

[7]. Gao Huang, Densely Connected Convolutional Networks, 2018, (https://arxiv.org/pdf/1608.06993.pdf|

[8]. Karen Simonyan, Very Deep Convolutional Networks for Large-Scale Image Recognition, 2015, (https://arxiv.org/pdf/1409.1556.pdf)

[9]. W. F. Lawless, Artificial Intelligence for the Internet of Everything, 2018

[10]. STM32F303 datasheet, 2018, (*https://www.st.com/resource/en/datasheet/stm32f303vc.pdf*)

[11]. Video of robot “motoko uprising deep neural network line following robot” (https://www.youtube.com/watch?v=E9FJIDowNmU).

[12]. Neural network source code (https://github.com/michalnand/rysy/tree/master/deployment/libs_network).