Using high-performance deep learning platform to accelerate object detection

S O Stepanenko[1], P Y Yakimov[1,2]

[1]Samara National Research University, Moskovskoe Shosse 34A, Samara, Russia, 443086
[2]Image Processing Systems Institute of RAS - Branch of the FSRC "Crystallography and Photonics" RAS, Molodogvardejskaya street 151, Samara, Russia, 443001

e-mail: serega.stepanenko.97@gmail.com

Abstract. Object classification with use of neural networks is extremely current today. YOLO is one of the most often used frameworks for object classification. It produces high accuracy but the processing speed is not high enough especially in conditions of limited performance of a computer. This article researches use of a framework called NVIDIA TensorRT to optimize YOLO with the aim of increasing the image processing speed. Saving efficiency and quality of the neural network work TensorRT allows us to increase the processing speed using an optimization of the architecture and an optimization of calculations on a GPU.

1. Introduction

Object detection is becoming more and more popular [1]. It has become possible with the development of new powerful computational devices and the use of neural networks, which can find objects in an image having high accuracy. A system that is based on an artificial neural network is not a big problem to be created because there is a large number of different frameworks which simplify creating of a neural network reducing the network development to functions call. The object detection problem requires high computational power, and in real tasks, for example processing of a video stream, powerful equipment is required [2]. For example, FPS of YOLO work on NVIDIA GTX Titan X is about 40 [3], FPS of SSD on NVIDIA GTX Titan X is 19 [4], FPS of FasterR-CNN on Tesla k40 is 5 [5], FPS of Fast R-CNN is 0.5 [3]. All those algorithms except for YOLO have FPS less than a common camera frame rate.

Nowadays there are many solutions for object detection [6]. All of them use different algorithms to detect, can detect with different accuracy and can have different speed of processing [7]. The most existing solutions use CUDA [8] to process data in parallel. Via CUDA, we can increase the processing speed but there are other ways to increase the processing speed as well. An optimization of the neural network architecture can be used to make the processing faster and to remain the accuracy at the same level. But it's not always easy to make especially if the network has a very complex architecture. There is a way to increase the neural network processing speed not spending much time to change the program.

There is a platform which is able to increase the neural network processing speed using algorithms to optimize an architecture and using abilities of NVIDIA GPUs to increase calculations as well. This platform is called TensorRT [9]. TensorRT provides an API for creating of neural networks and allows us to optimize models of many popular frameworks as well. It makes that convenient to use in many cases because it's possible to accelerate the program not spending many resources to change the code.

2. Convolutional neural networks inference technologies

The word inference means receiving the result of work of the neural network which was trained on some data set. This article considers a use of the platform TensorRT to accelerate an algorithm for object detection that is called YOLO [9].

2.1. YOLO

YOLO [10] – is an algorithm for object classification and detection using convolutional neural networks to do that. Pros of convolutional neural networks for tasks of this type are that convolutional neural networks can process images having more simple architecture than standard neural networks. There are many implementations of YOLO based on different frameworks and written in different programming languages. The standard implementation is based on the neural darknet which is written in the programming language C. The work of YOLO starts from changing of the input image. It becomes 448x448x3, where 448x448 is the image size, 3 is color channels amount. At first the image is passed through the modified net GoogleNet. It's the 1st 20 layers of the network. The output of this part of the network is 1024 feature maps with size of 14x14. Then the images are passed through a sequence of convolutional layers and a sequence of pooling layers. At the moment of getting into a fully connected layer there are 1024 feature maps with size of 7x7. After the images have been passed through 2 fully connected layers the network provides prediction of some class belonging and provides the position of an object in the image [11].

To define the object bounds in YOLO algorithm at first a grid with size of SxS is imposed. Then object prediction is done for each cell. A vector with size of 5*B+C is created for each grid element, where B is bound amount which are predicted by a grid element, C is class amount which the network can predict, 5 defines object amount which can be found. 1st 5*B values of the vector show coordinates of the center of the bound inside the grid cell, height and width and probability that the bound has been defined correctly. Other C values show probability that the object center is at the center of this cell. As a result, there are S*S*B bounds of objects with class probabilities. Then the vector is sorted descending and the algorithm Non maximal suppression is used. It repeats for every class. As a result, all bounds are viewed. The max probability of classes is considered for every bound and if it is positive then the bound is put on the image [3].

2.2. TensorRT

TensorRT is a platform of deep learning by NVIDIA [12]. Nowadays there are 5 versions of TensorRT. Every new version is able to interact with greater number of layer types of a neural network and mathematical operations. TensorRT enables to use implemented parsers for many popular frameworks. It contains: Tensorflow, Caffe2, PyTorch, Mxnet, Microsoft Cognitive Toolkit, Chainer. Tensorflow has built-in TensorRT 3.0 [9]. In case when the network is created on these frameworks it is very simple to use TensorRT. It is enough to use ат implemented parser. The process of the network creating with use of a TensorRT parser is shown in figure 1. If the network is not created on these frameworks, then it's possible to use the API of TensorRT to transfer the network model.

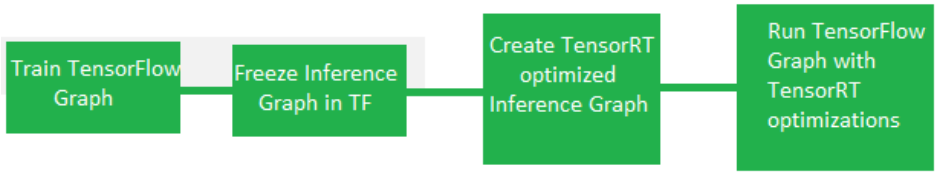

Figure 1. A flow of the network creating with use of TensorFlow on TensorRT.

The advantage of TensorRT using is that this platform is able to accelerate a neural network using an algorithm to simplify the network architecture not changing the network functionality and using abilities of NVIDIA GPUs to accelerate calculations.

To simplify the network architecture TensorRT analyzes a graph that represent the network model. If there are elements in the graph which are repeated, then TensorRT merges them. As a result the network size becomes less.

Acceleration on a GPU is possible due to an ability to use “Tensor Cores”. These cores allow to use half-precision data type float16 for calculations. It is not possible if CUDA is used. CUDA allows to use data type float32. The processing speed increases due to much more fast transfer of data and more fast calculations with this data type. This type of accelerating is possible only with use of a little amount of GPUs which can provide this technology.

3. YOLO implementation

To compare the processing speed implementations of YOLO with use TensorRT platform and without use, with use of one data set and same trained models, have been considered.

3.1. Implementation of YOLO without use of TensorRT

3.1.1. Darknet

To compare performance one of implementations of the YOLO algorithm that is based on the neural network darknet has been considered. YOLO was run on a GPU. To do that CUDA 10.0 and OpenCV were required. YOLOv2 model was used as the model. Before running it’s required to make the project. It can be done via running the command make from the project folder. After installing There will be an executable file which must be run. To type a command with required options is enough to run. The command for running the program is the follow: *./darknet detect path_to_cfg_file path_to_weights_file*. Darknet allows to process a video from a file and from a webcam.

3.1.2. Darkflow

Another implementation of the YOLO algorithm in the language python that uses Tensorflow. It is required to install CUDA 9.0, Tensorflow 1.0, Numpy, OpenCv 3.0 or above to run this program. It is required to have CUDA 10.0 and CUDA 9.0 to run darkflow and other implementations in one PC. To change the CUDA versions it is enough to update the environment variables. Darkflow has an ability to process a video stream. Before running it is required to run an installation script. After installatiom the program can be run via command: flow --model path_to_cfg_file --load path_to_weights_file --imgdir path_to_folder_with_images –gpu percent, where percent – a digit from 0 to 1 that shows the percent ofGPU usage. 0 – o% of usage. 1 – 100% of usage. The processing is on a CPU if –gpu has not been specified. Probably in this case the processing speed is significantly less than in case of processing on a GPU.

3.2. Implementation of YOLO with use of TensorRT

This article presents an implementation of YOLO with TensorRT 5.0 [13]. Before launching the program it’s required to install all dependencies. To make the program runnable CUDA 10.0, TensorRT 5.0,

OpenCV 3.4.0 are required. Files which contain trained model weights and the network configuration are required to run the program. They can be found on the official web site of the YOLO developers. A trained model YOLOv2 is used for research. This model is able to detect 80 classes of objects. At first it's required to install the project using make. Then it's required to set up the project typing paths to all dependencies and to weights and configuration files. Then the data type that will be used must be chosen. It's possible to choose Float32, Float16 and Int8. In case when the GPU doesn't support tensor cores the program can be run with use Float32 only. There is a possibility to process not only single images and batches of images. Video processing is possible when Deepstream SDK is used in addition. Deepstream SDK is developed by NVIDIA to process data in streams. It uses TensorRT, CUDA, Video Codec SDK. Today the last version of Deepstream SDK is 3.0. It's possible to process video without use of Deepstream SDK when the source code is changed to make it possible to extract frames from video streams. Such capability is provided by OpenCV. To run the program, it's required to type the following command:

trt-yolo-app

The following options are available for this command:

- Batch_size – Images amount which are processed at the moment
- Decode – Input is either True or False. It is for decoding of images. True by default.
- Seed – A parameter for the random digit generator.

After the program work has been finished files which contain processed images are saved to a folder. To process a video, it's possible to use OpenCV which extracts frames from the video stream. There is another way to process a video to use deepstream a library by NVIDIA to process streams. Deepstream uses libraries for accelerating stream processing and uses TensorRT and CUDA as well.

Also Darkflow was modified in order to be run on TensorRT.

4. Experiment researches

2 implementations of the YOLO algorithm were used with use of the one trained model YOLOv2 for experimental research. 2416 images were used as input. Output images which objects were found on were saved to a folder. Processing time of every image were written to a file for the implementation without TensorRT. Processing time of every image wasn't calculated and the average time was calculated. All experiments were done on a PC with characteristics which are presented in table 1.

Table 1. Main characteristics of the PC.

GPU	CPU	Memory
NVIDIA GeForce GT 710	AMD FX-4300	4 GB
NVIDIA GeForce GTX 950	Intel Core i5-6500	8 GB

Average FPS of the image set processing by Darkflow implementation is presented in table 2.

Table 2. FPS of Darkflow work.

GPU	FPS
NVIDIA GeForce GT 710	1.31
NVIDIA GeForce GTX 950	10.53
Tesla p100	120
NVIDIA GeForce GTX 2080 TI	170

Average FPS of the image set processing by Darknet is presented in table 3.

Table 3. FPS of Darknet work.

GPU	FPS
NVIDIA GeForce GT 710	1.2
NVIDIA GeForce GTX 950	6.25

Darknet processes the images slower than Darkflow. Average FPS of the image set processing by an implementation of YOLO in TensorRT API is presented in table 4.

Table 4. FPS of an implementation of YOLO in TensorRT.

GPU	FPS
NVIDIA GeForce GT 710	5
NVIDIA GeForce GTX 950	50

YOLO in TensorRT works faster than Darkflow and darknet. The implementation is written in C++ with use of API of TensorRT.

The time of work with different size of a batch was compared for the implementation with use of TensorRT. Batch size was from 1 to 16. It was not possible to allocate the GPU memory if the batch size was more than 16. Time of work with use of different batch size is presented in figures 2 and 3 for 2 different GPUs.

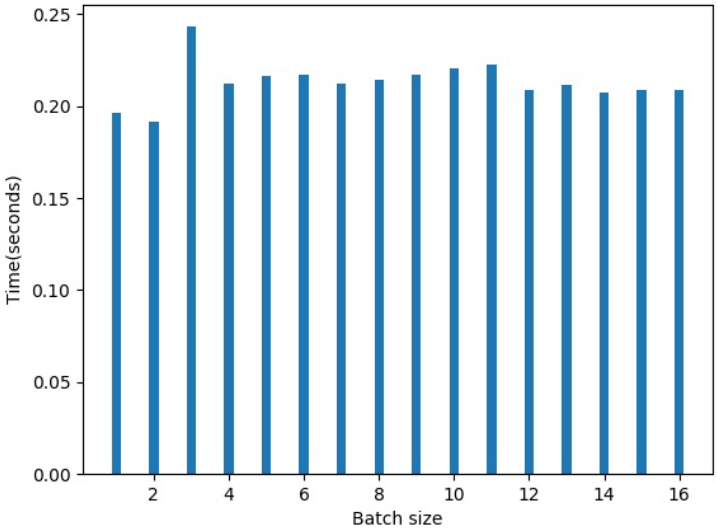

Figure 2. Time of the algorithm work with use of different batch size for NVIDIA GeForce GT 710.

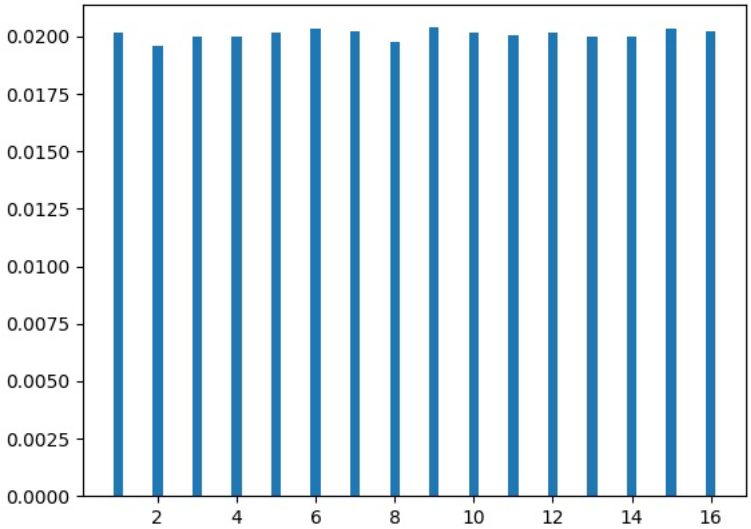

Figure 3. Time of the algorithm work with use of different batch size for NVIDIA GeForce GTX 950.

Difference between work time on 2 GPUs is not significant. The difference between the worst and the best results was about 1.27 times. This could be related to different causes and it's difficult to define the optimal size in advance. It should be done experimental.

Average FPS of Darkflow with TensorRT is presented in table 5.

Table 5. FPS of Darkflow with TensorRT.

GPU	FPS
NVIDIA GeForce GT 710	1.78
NVIDIA GeForce GTX 950	14.41
NVIDIA GeForce GTX 2080 TI	200

Darkflow with TensorRT works faster than Darkflow about in 1.36 times on NVIDIA GeForce GT 710, 1.37 times on NVIDIA GeForce GTX 950, 1.18 times on NVIDIA GeForce GTX 2080 TI. FPS of all used implementation is presented in figure 4.

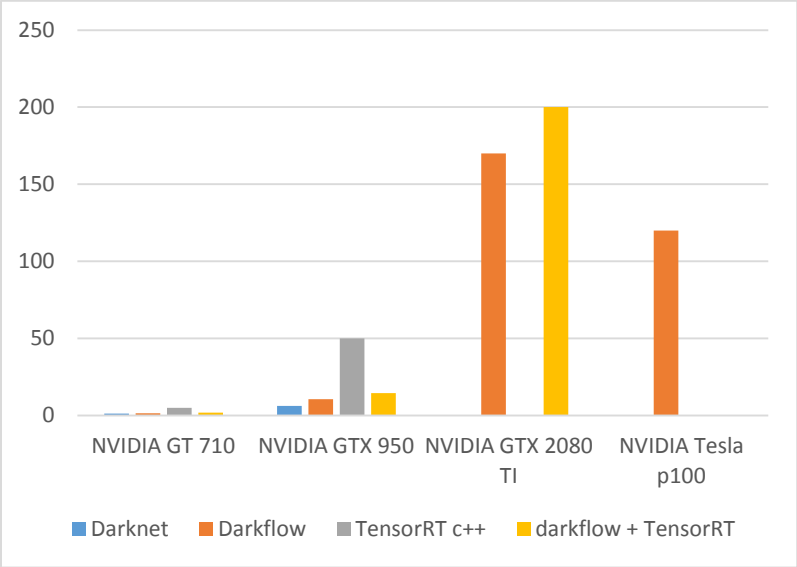

Figure 4. FPS of all used implementations.

Figure 5. A processed frame.

YOLO in TensorRT API has the best acceleration. The acceleration is about 10 times. Darkflow with TensorRT has an acceleration but it is much more less.

After using of TensorRT accuracy of YOLO work has not been reduced. An example of an image that is processed by YOLO is presented in figure 5.

5. Conclusion

The article considered 3 implementations of the YOLO algorithm to compare performance. One of these implementations uses the TensorRT platform. Another implementation was modified in order to work with TensorRT. The platform is able to accelerate the algorithm producing the same accuracy. This ability can be used on practice in video stream processing where processing speed is an important value. Using TensorRT the processing time reduced about by 4 times on NVIDIA GT 710 and about by 8 times on NVIDIA GTX 950 in comparison with the standard implementation of the algorithm if an ability of GPUs to do calculations with use of tensor cores was not used because the GPU could not do such calculations. Darkflow that was modified worked faster in 1.36 times on NVIDIA GT 710, 1.37 times on NVIDIA GeForce GT 950, 1.18 times on NVIDIA GTX 2080 TI.

6. References

[1] Bibikov S A, Kazanskiy N L and Fursov V A 2018 Vegetation type recognition in hyperspectral images using a conjugacy indicator *Computer Optics* **42(5)** 846-854 DOI: 10.18287/2412- 6179-2018-42-5-846-854
[2] Shatalin R A, Fidelman V R and Ovchinnikov P E 2017 Abnormal behavior detection method for video surveillance applications *Computer Optics* **41(1)** 37-45 DOI: 10.18287/2412- 6179-2017-41-1-37-45
[3] Redmon J, Farhadi A 2017 *YOLO9000: Better, Faster, Stronger* (University of Washington, Allen Institute for AI) p 9
[4] Wei L 2016 SSD: Single Shot MultiBox Detector *ECCV: Computer Vision* 21-37
[5] Ren Sh, He K, Girshick R and Sun J 2017 Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks *IEEE Transactions on Pattern Analysis and Machine Intelligence* **39(6)** 1137-1149 DOI: 10.1109/TPAMI.2016.2577031
[6] Amosov O S, Ivanov Y S and Zhiganov S V 2017 Human localiztion in video frames using a growing neural gas algorithm and fuzzy inference *Computer Optics* **41(1)** 46-58 DOI: 10.18287/2412-6179-2017-41-1-46-58
[7] Shustanov A, Yakimov P 2017 CNN Design for Real-Time Traffic Sign Recognition *Procedia Engineering* **201** 718-725 DOI: 10.1016/j.proeng.2017.09.594
[8] CUDA URL: https://developer.nvidia.com/cuda-gpus (01.11.2018)
[9] Official site of TensorRT URL: https://developer.nvidia.com/tensorrt (01.11.2018)
[10] YOLO: Real-Time Object Detection URL: https://pjreddie.com/darknet/yolo/ (01.11.2018)
[11] Redmon J, Divvala S, Girshick R, Farhadi A 2015 You Only Look Once: Unified, Real-Time Object Detection *You Look Only Once* p 10
[12] TensorRT integration speeds up tensorflow inference URL: https://devblogs.nvidia.com/tensorrt-integration-speeds-tensorflow-inference/ (01.11.2018)
[13] Implementation of YOLO with TensorRT URL: https://github.com/vat-nvidia/deepstream-plugins/ (01.11.2018)

Acknowledgements

This work was partly funded by the Russian Foundation for Basic Research – Project # 17-29-03112 ofi_m and the Russian Federation Ministry of Science and Higher Education within a state contract with the "Crystallography and Photonics" Research Center of the RAS under agreement 007-ГЗ/Ч3363/26.