SPEED-UP OF OBJECT DETECTION NEURAL NETWORK WITH GPU

Takuya Fukagai[⋆] *Kyosuke Maeda*[†] *Satoshi Tanabe*[‡]
Koichi Shirahata[⋆] *Yasumoto Tomita*[⋆] *Atsushi Ike*[⋆] *Akira Nakagawa*[⋆]

[⋆] Fujitsu Laboratories Limited
[†] Fujitsu Software Technologies Limited
[‡] Fujitsu Limited

ABSTRACT

We realized a speed-up of an object detection neural network with GPU. We improved the object detection speed of faster R-CNN [1], which is one of the most commonly used detection networks [2].

The speed of the original faster R-CNN (py-faster-rcnn [3]) was 72.4ms per image on our GPU server [1]. We accelerated the detection speed by implementing our new algorithms that are suitable for GPUs. The speed-up is realized without sacrificing the object detection accuracy (mAP). Our GPU-accelerated faster R-CNN can detect objects with 55.8ms per image. This is nearly 30% speed-up.

In detection networks, the processes of building scored candidate regions, sorting and non-maximum-suppression (nms) are commonly used. In faster R-CNN, these processes are executed in proposal layer. We reduced the processing time of the proposal layer from 5.6ms to 2.2ms. This is 2.5 times as fast as the original one.

We also evaluated the detection speed with larger batch sizes. By applying batch size 16, it is accelerated to 44.9ms per image. This is 1.6 times as fast as the original faster R-CNN (py-faster-rcnn).

Since we realized a speed-up of common basic methods for detection networks, our speed-up methods are also applicable to other detection networks such as R-FCN [4], YOLO [5] [6] and SSD [7].

Index Terms— deep learning, object detection, neural network, faster R-CNN, GPU

1. INTRODUCTION

Object detection is one of the most useful and basic applications of deep neural networks. The methods with deep neural networks got the highest scores in the object detection competitions such as "ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" and "COCO: Common Objects in Context Detection Challenge".

[1]OS: Ubuntu 14.04.5 LTS (GNU/Linux 4.2.0-42-generic x86_64), CPU: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, GPU: GPU (Tesla P100-PCIE-16GB) x 1, Libraries: MKL, CUDA 8.0, cuDNN v5.1.5

Various kinds of detection networks have been reported. Faster R-CNN is one of the most commonly used detection networks. It achieves object detection with high mean Average Precision (mAP), compared with other high-speed detection networks [2].

Major object detection networks are based on Convolutional Neural Networks (CNN) and many methods have been proposed to speed-up the calculation of convolution and full connection layers which are the basic components of CNN [8] [9] [10] [11] [12] [13].

In detection networks, however, not only convolution and full connection but also other modules require fair amount of time to process. We speeded up the following common processes of the detection networks.

image preprocessing : Conversion of inputs from image inputs to network inputs.

proposal layer : A neural network layer which proposes rectangular regions where candidate objects are likely to exist. Proposal layer has following three processes. 1) Building scored candidate regions by combining inputs from several neural network modules. 2) Sorting scored candidate regions. 3) Selecting high score candidate regions in turn and suppress other overlapping regions by nms.

postprocessing : Conversion of outputs from neural network outputs to object detection results.

In this paper, we propose the methods to speed up the above processes with GPU. In section 2, we introduce the latest researches on detection networks as related works. In section 3, we explain the flow of faster R-CNN. In section 4, we show the results of the speed-up. In section 5, we explain how we realized the speed-up of faster R-CNN. In section 6, we summarize our research.

2. RELATED WORKS

In this section, we introduce 4 major kinds of detection networks, faster R-CNN, R-FCN, YOLO and SSD.

They are categorized into 2 groups, the models with Region Proposal Network (RPN layer) and the models without it. Faster R-CNN and R-FCN are the models with RPN layer. YOLO and SSD are the models without it. YOLO and SSD are also called single shot detection models.

Although we only speeded up faster R-CNN with GPU, our speed-up methods are also applicable to R-FCN, YOLO and SSD.

3. FLOW OF FASTER R-CNN

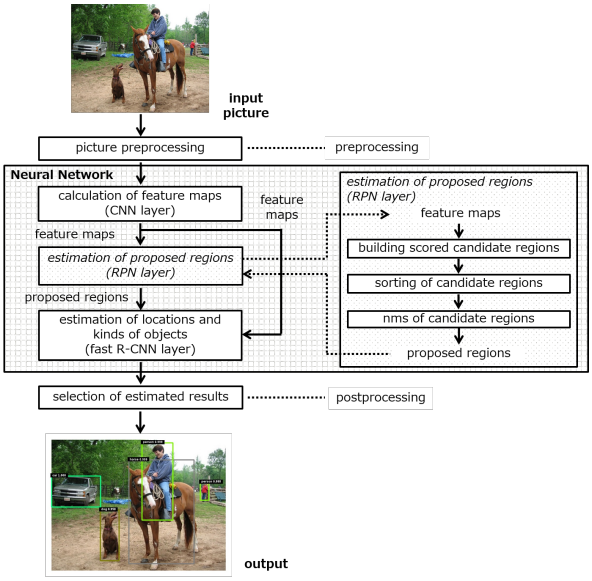

Fig. 1. Flowchart of faster R-CNN

Fig. 1 is a flowchart of faster R-CNN.

First, faster R-CNN preprocesses input pictures. During preprocessing, it resizes input pictures and subtract average RGB values from them.

Second, it calculates feature maps from preprocessed images with CNN layer. We used VGG16 [14] as a CNN layer. Third, it estimates proposed regions with RPN layer. Proposed regions are rectangular areas where target objects are likely to exist. During the process of generating proposed regions, we calculate regions like Fig. 2. By applying sorting and nms to these regions, it reduces the number of proposed regions. Fourth, fast R-CNN [15] layer estimates locations and categories of objects. Object detection areas are indicated as rectangular areas.

Finally, the postprocessing module outputs the object detection results. It receives the inputs from several neural network modules and builds the candidates of detected results. After building the object detection candidates, nms is applied again. It removes low score candidate regions that are overlapping with high score regions.

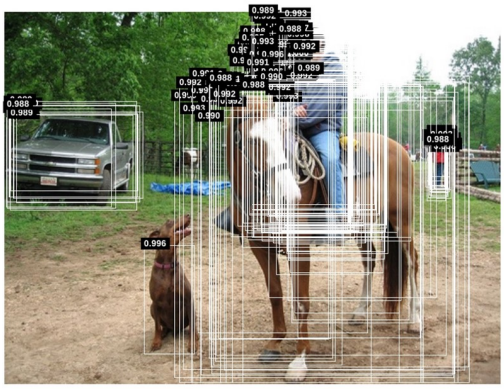

Fig. 2. Example of proposed regions before applying nms

4. RESULTS OF SPEED-UP

4.1. Results of object detection speed-up

Fig. 3 is a comparison of detection speed between original faster R-CNN (py-faster-rcnn), PVANet [16] [17] and our GPU-accelerated faster R-CNN. PVANet has the open source GPU-accelerated proposal layer. Our methods can detect objects with 55.8ms per image. We realized nearly 30% speed-up without sacrificing the object detection accuracy (mAP). It is also faster than PVANet. We evaluated the detection speed with larger batch sizes. By applying batch size 16, it is accelerated to 44.9ms per image.

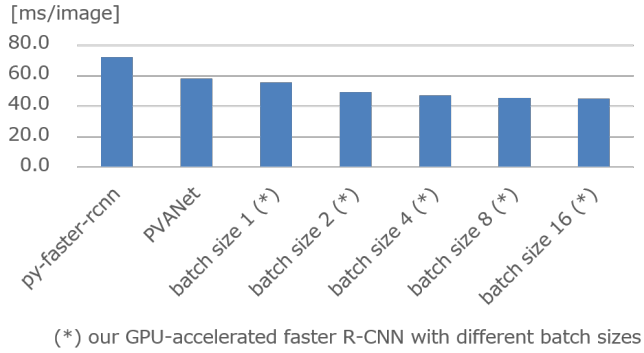

Fig. 3. Comparison of object detection speed with minibatch

4.2. Results of proposal layer speed-up

Fig. 4 shows the processing time of proposal layer. We compared the speed of the original faster R-CNN (py-faster-rcnn) and our GPU-accelerated faster R-CNN. We also showed the speed of PVANet. We realized 2.5 times speed-up of the proposal layer and reduced the time from 5.6ms to 2.2ms. When we applied the batch size larger than 1, it was further accelerated. The speed-up of proposal layer was more than 3 times with batch size 2 or more.

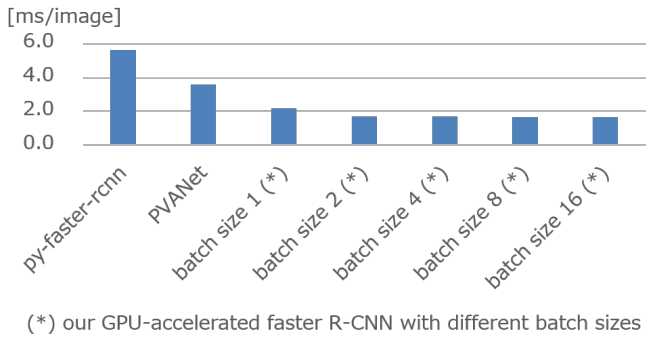

Fig. 4. Processing time of proposal layer with minibatch

4.3. Details of speed-up

Fig. 5 shows the details of the speed-up. The first item, "outside the neural network" includes preprocessing and postprocessing. The second item, "proposal layer" has the processes of building candidate regions, sorting and nms. The third and fourth items are the total time spent for the calculation of convolution and full connection layers per image. The fifth item, "other neural network layers" consists of ReLu, Pooling, Softmax, Reshape, Dropout and ROIPooling layers. We speeded up the first 2 items in Fig. 5. They have the common basic processes in detection networks.

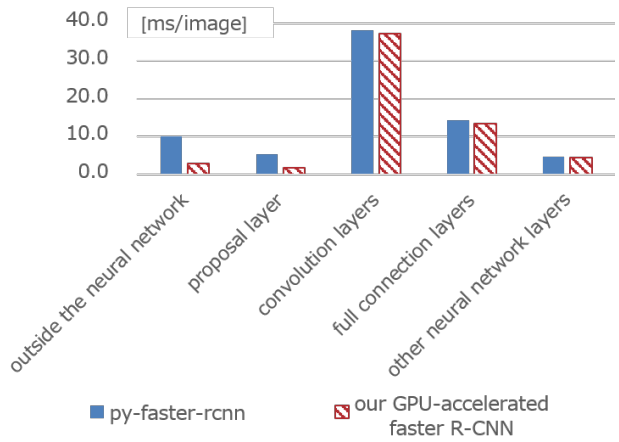

Fig. 5. Details of the speed-up

5. METHODS OF SPEED-UP

5.1. Speed-up of the process outside the neural network

5.1.1. Speed-up of preprocessing

The preprocessing resizes input pictures and subtract average RGB values from them. We assigned a thread for each output pixel. The number of threads is the same as the number of output pixels. Each thread executes bilinear interpolation by referring to the values of the input pixels. Next, each thread subtracts average RGB values from the interpolated output values. It also changes the order of the input multidimensional array from $H(height) \times W(width) \times C(channel)$ to $C \times H \times W$. Each thread is executed in parallel on GPU.

5.1.2. Speed-up of postprocessing

The postprocessing module builds scored candidates of detected results from network outputs. It assigns a thread for each candidate of detected results. It calculates these object detection candidates in parallel on GPU. After estimating scored candidates of detected results, it executes nms. The nms is also executed in parallel on GPU.

5.2. Speed-up of proposal layer

5.2.1. Speed-up of building scored candidate regions

Scored candidate regions are calculated by adding the outputs from other neural network layers to the fixed anchor regions. Each anchor generates a single scored candidate region. In the process of building proposed regions, we assign a thread to each scored candidate region. The scores and locations of candidate regions are calculated in parallel on GPU.

5.2.2. Speed-up of sort process

We designed and implemented a high speed parallel sorting algorithm for modern GPUs. A lot of research works have been reported in this field [18] [19] [20] [21] [22]. Our parallel sorting method is based on merge sort. It sorts about 10,000 scored candidate regions fast. This method is especially useful for getting top 1,024 sorted regions among some 10,000 scored regions. This is a typical number for the proposed regions in faster R-CNN.

We can apply this method to other object detection neural networks like R-FCN, YOLO and SSD with slight modification since this sorting algorithm is relatively simple.

Fig. 6 shows an overview of our fast GPU sorting algorithm. Our fast GPU sorting algorithm has 2 basic parallel sorting algorithms as its basic components. One is [parallel sort 1] and the other is [parallel sort 2] in the top of Fig. 6.

Both [parallel sort 1] and [parallel sort 2] assign a thread for each element in their groups. In [parallel sort 1], each thread counts the number of elements whose values are higher than each target element and moves it to the counted position. In [parallel sort 2], it merges 2 sets of sorted elements and gets a set of sorted elements. Each thread calculates the number of elements whose values are higher than the target element in the other set with binary search and decide the new position by adding this number to the original position.

Our fast GPU sorting algorithm consists of 2 steps as in Fig. 6. Step 1 divides scored elements into blocks. Each

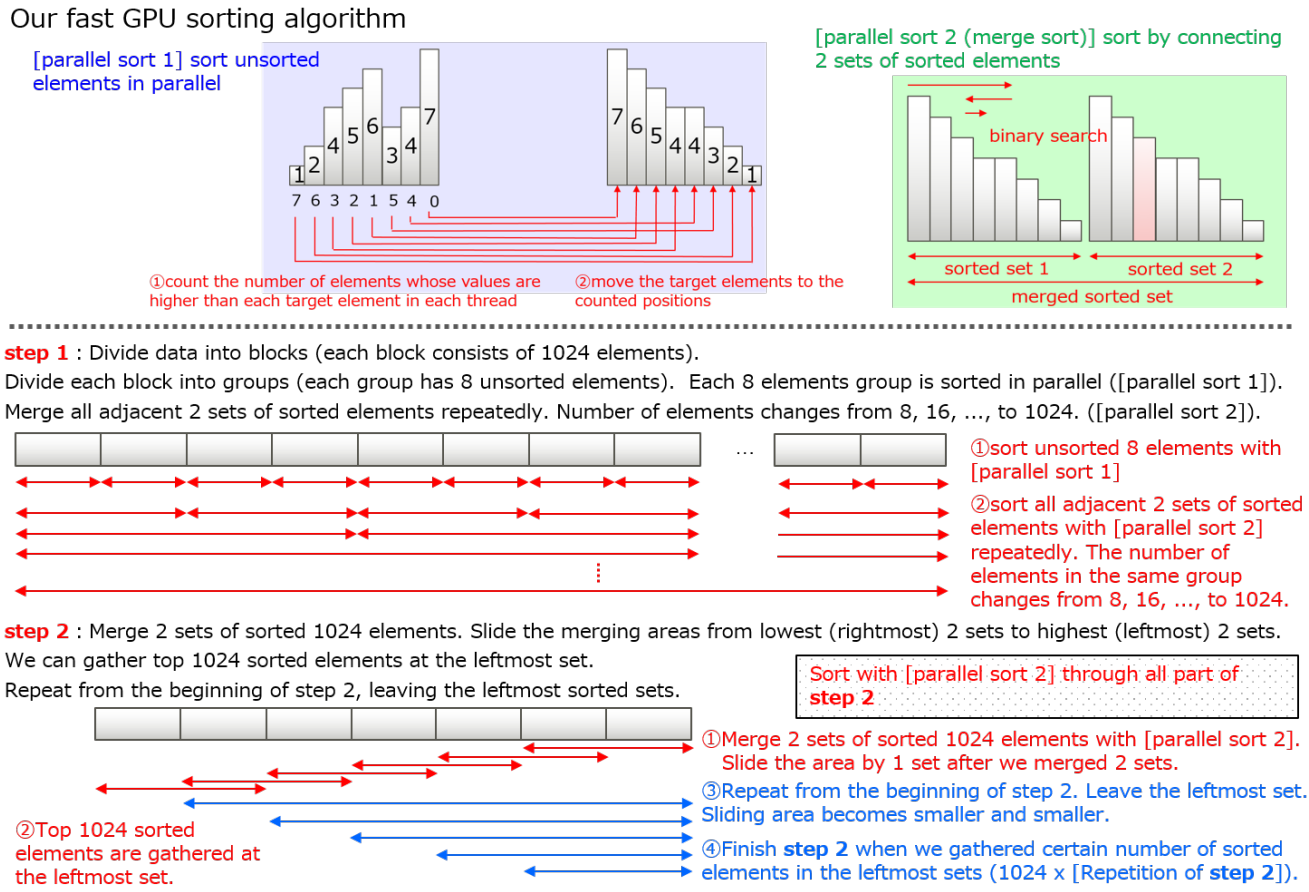

Fig. 6. Overview of our fast GPU sorting algorithm

block consists of 1,024 elements. Then it sorts them in parallel efficiently. 1,024 is the maximum number of threads in a CUDA block. Step 2 sorts the elements of several blocks efficiently by merging 2 sets of sorted 1,024 elements in adjacent blocks repeatedly. Details are described in Fig. 6.

When we slide 2 adjacent 1,024 sorted elements in step 2, we keep top 1,024 of sorted 2,048 elements in the shared memory and copy next 1,024 elements from the global memory. We can extend this method so that we can execute more than 2 merge processes in parallel.

5.2.3. Speed-up of non-maximum-suppression (nms) process

We speeded up the nms process with GPUs. Our nms implementation is based on the one in PVANet. We modified nms in PVANet and made it faster.

5.3. Speed-up as a whole

We rewrote all of Python scripts in py-faster-rcnn into C++. Then we speeded up the inference phase in faster R-CNN with GPU implementation. As we described in section 1, we accelerated the common basic processes of the detection networks, image preprocessing, proposal layer (building candidate regions, sort and nms) and postprocessing.

We extended our GPU-accelerated faster R-CNN so that it allows minibatch size greater than 1.

6. SUMMARY AND CONCLUSIONS

We speeded up faster R-CNN with GPU. Since we realized a speed-up of the common basic calculation of detection networks, our speed-up methods are also applicable to other major detection networks like R-FCN, YOLO and SSD.

We evaluated the speed-up of faster R-CNN with the default parameters of py-faster-rcnn whose CNN layer is VGG16. As Fig. 5 shows, convolution and full connection layers still take a fair amount of time to process. We expect that we will observe more significant speed-up if we apply our methods to the network which spends less time to compute convolution and full connection layers.

7. REFERENCES

[1] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[2] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, and Kevin Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "py-faster-rcnn," in *https://github.com/rbgirshick/py-faster-rcnn*, Accessed: 2016-09-27.

[4] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Neural Information Processing Systems (NIPS)*, 2016.

[5] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi, "You only look once: Unified, real-time object detection," in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[6] Joseph Redmon and Ali Farhadi, "Yolo9000: Better, faster, stronger," in *Computer Vision and Pattern Recognition (CVPR)*, 2017.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, 2016.

[8] Shmuel Winograd, "Arithmetic complexity of computations," in *Siam vol. 33*, 1980.

[9] Andrew Lavin and Scott Gray, "Fast algorithms for convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[10] Michael Mathieu, Mikael Henaff, and Yann LeCun, "Fast training of convolutional networks through ffts," in *International Conference on Learning Representations (ICLR)*, 2014.

[11] Nicolas Vasilache, Jeff Johnson, Soumith Chintala, Serkan Piantino, and Yann LeCun, "Fast convolutional nets with fbfft: A gpu performance evaluation," in *International Conference on Learning Representations (ICLR)*, 2015.

[12] Liuan Wang, Wei Fan, Jun Sun, Yasumoto Tomita, and Atsushi Ike, "Column weight pruning for accelerating dnn inferences," in *GTC (GPU Technology Conference)*, 2017.

[13] Akihiko Kasagi, Tsuguchika Tabaru, and Hirotaka Tamura, "Fast algorithm using summed area tables with unified layer performing convolution and average pooling," in *IEEE 27th International Workshop on Machine Learning for Signal Processing (MLSP)*, 2017.

[14] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations (ICLR)*, 2015.

[15] Ross Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[16] Sanghoon Hong, Byungseok Roh, Kye-Hyeon Kim, Yeongjae Cheon, and Minje Park, "Pvanet: Lightweight deep neural networks for real-time object detection," in *NIPS 2016 Workshop on Efficient Methods for Deep Neural Networks (EMDNN)*, 2016.

[17] Sanghoon Hong, Byungseok Roh, Kye-Hyeon Kim, Yeongjae Cheon, and Minje Park, "pva-faster-rcnn," in *https://github.com/sanghoon/pva-faster-rcnn*, Accessed: 2017-06-26.

[18] Nadathur Satish, Mark Harris, and Michael Garland, "Designing efficient sorting algorithms for manycore gpus," in *In Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, May 2009.

[19] Nikolaj Leischner, Vitaly Osipov, and Peter Sanders, "Gpu sample sort," in *IEEE 24th International Parallel and Distributed Processing Symposium*, 2010, pp. 1–10.

[20] Nadathur Satish, Changkyu Kim, Jatin Chhugani, Anthony D. Nguyen, Victor W. Lee, Daehyun Kim, and Pradeep Dubey, "Fast sort on cpus and gpus: A case for bandwidth oblivious simd sort," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, 2010, pp. 351–362.

[21] Andrew Davidson, David Tarjan, Michael Garland, and John D. Owens, "Efficient parallel merge sort for fixed and variable length keys," in *In Proceedings of the 23rd IEEE International Parallel and Distributed Processing Symposium*, May 2012, pp. 131–139.

[22] Hagen Peters, Ole Schulz-Hildebrandt, and Norbert Luttenberger, "A novel sorting algorithm for many-core architectures based on adaptive bitonic sort," in *IEEE 26th International Parallel and Distributed Processing Symposium*, 2012, pp. 227–237.