YOLO-based Object Detection on ARM Mali GPU

Trunghai Do, Jemin Lee, Hyungshin Kim*

(Dept. of Computer Science and Engineering, Chungnam National University)

Abstract: Nowaday, convolutional neural networks become the heart of many computer vision solutions to solve a wide range of tasks. In this paper, we present YOLO implementation on ARM Mali-T628 MP6 GPU of ODROID-XU4. Original YOLO algorithm was implemented by Darknet framework dedicated for NVIDIA GPU. Unlike Darknet, we use CK-Caffe as a deep learning backend to run YOLO on ARM-based GPUs. Additionally, we replace all fully connected layers which act as detection layers in YOLO with convolutional layers to reduce the model size. We train our GoogLeNet-based and Fast YOLO-based models on canonical PASCAL VOC2007 + VOC2012 "trainval" set on NVIDIA GTX-1080 and experiment on PASCAL VOC2007 "test" set on ODROID-XU4. As a result, GoogLeNet takes 7.5 seconds per image and Fast YOLO takes 1.6 seconds per image at inference time.

Keywords : Real-time Object detection, Embedded system, Convolutional neural networks.

I. Introduction

Recently, convolutional neural networks (CNNs or ConvNets) are widely used in object detection applications. Object detection must not only classify object's category but also localize object's position in an image. Moreover, it is known that object detection in real-time requires more computation and memory than image classification. Those challenges encourage most of the effort to concentrate on improving performance of object detection algorithms.

We can train network models on high-end NVIDIA GPU to save development time. However, these trained models should be run on a wide range of target devices including high-end GPUs and embedded devices. For example, autonomous system such as self-driving vehicles, drones or robots, all of them need efficient real-time object detection algorithms to guarantee control without delay. Therefore, speed is one of the most important features of the object detection task. Besides, through ILSVRC challenges, CNN architectures have been proven that the higher accuracy can be obtained by deepening the network architecture. Thus, the depth of network is as important as the detection speed. Nevertheless, deeper and wider neural network-based solutions always come along with power-hungry and are difficult to migrate to resource-constrained systems compared to high-end GPUs. To be able to deploy real-time object detection models on embedded devices, the models should preserve the accuracy, should be fast and small size. The requirements are challenges that need to be addressed as we deploy deep learning algorithms on embedded systems.

In our work, we use YOLO[1] which is a real-time object detection algorithm among state-of-the-art detection approaches. This algorithm was originally implemented by employing Darknet framework [2] which takes advantage of powerful CUDA library dedicated

* Corresponding Author
Hyungshin Kim : Dept. of Computer Science and Engineering, Chungnam National University,

for NVIDIA GPU except for a slow solution on CPU. Therefore, original YOLO can only be deployed on NVIDIA GPU or CPU. To overcome this, we leverage caffe-yolo [3] which is a re-implemented version of YOLO on Caffe framework [4]. We employ GoogLeNet [5] and Fast YOLO as our backbone models for feature extraction. We also replace all fully connected layers corresponding for detection with convolutional layers called ConvDet that helps reduce the model size. We train models on NVIDIA GTX-1080, our work focuses on deploying trained models on embedded system named ODROID-XU4 which has Mali -T628 MP6 GPU. Our models run on CK-Caffe [6] deep learning framework supported by OpenCL. GoogLeNet based model takes 7.5 seconds per image and Fast YOLO based model takes 1.6 seconds per image at test time on ARM Mali-T628 MP6 embedded GPU. When we tested image classification by using shallow network architecture named AlexNet[7] on ARM Mali-T628 MP6 GPU-0, it takes approximately 803ms to classify one image. Having a large number of parameters that need to be evaluated, object detection requires more computation than image classification. Thus, our detection time on ARM Mali-T628 MP6 GPU-0 is acceptable with deeper and more complex architecture than AlexNet

II. Implementation

1. Deep learning framework

In this work, we choose CK-Caffe which is a deep learning framework based on Caffe. Unlike Caffe, CK-Caffe runs on OpenCL API supporting variety of architectures: CPUs, GPUs, DSPs, FPGAs. At training phase, we can easily define Caffe model architecture as config files with the Protocol Buffer language and train the model on powerful NVIDIA GTX-1080 GPU. Once the CNN model is trained, it can also easily be deployed on ARM Mali-T628 MP6 GPU which is an embedded GPU on ODROID-XU4.

2. Object detection algorithm

YOLO adopts a single-pass detection pipeline combining bounding box localization and classification by a single network. YOLO system divides the input image into a $S \times S$ grid. Each grid cell predicts B bounding boxes and confidence scores for each box. Each bounding box consists of 5 predictions: x, y, w, h, and confidence. Each grid cell also predict C conditional class probabilities $\Pr(Class_i|Object)$. YOLO prediction system is encoded as an $S \times S \times (B \times 5 + C)$ tensor. For evaluating on PASCAL VOC, we also use S=7, B=2, PASCAL VOC has 20 labelled classes hence C=20. The final prediction is a $7 \times 7 \times 30$ tensor.

3. Network architectures

a) GoogLeNet model. The Inception module of GoogLeNet was designed to work well even under strict constrains on memory and computational budget. We stack two more convolutional layers to obtain rich feature before forwarding these feature maps through detection layer. Suppose that the input feature map is of size (W_f, H_f, Ch_f) where W_f and H_f are width and height of feature map and Ch_f is the number of input channels to detection layer. Base YOLO detection layer is comprised of two fully connected layers. Assuming the number of outputs of fc1 is F_{fc1}, then the number of params in the fc1 layer is $W_f \times H_f \times Ch_f \times F_{fc1}$. The second fully connected layer generates C class probabilities and $B \times 5$ bounding box coordinates and confidence scores for each of $W_o \times H_o$ grids. Therefore, the number of params in the fc2 layer is $F_{fc1} \times W_o \times H_o \times (B \times 5 + C)$. The total number of params in detection layer of base YOLO is $F_{fc1} \times (W_f \times H_f \times Ch_f + W_o \times H_o \times (B \times 5 + C))$. In our work, we use convolutional layer for detection (shortly name it ConvDet). Denote

ConvDet's filter width as F_w and F_h. We maintain the output shape of ConvDet as spatial dimension of input feature map and compute $(B\times5+C)$ outputs for each grid, thus the number of params required by the ConvDet layer is $F_w\times F_h\times Ch_f\times(B\times5+C)$. The number of params reduce from 212×10^6 to 0.28×10^6

b) Fast YOLO model. This model employs a neural network with fewer convolutional layers and fewer filters than base YOLO and still uses a fully connected layer as detection layer. Assuming the acronym is as same as above, the number of params required by detection layer is $W_f\times H_f\times Ch_f\times F_{fc1}$. The same strategy has been applied, thus the params in ConvDet is $F_w\times F_h\times Ch_f\times(B\times5+C)$. The number of params reduce from 73×10^6 to 0.28×10^6.

4. Training

We train both GoogLeNet+ConvDet and Fast YOLO+ConvDet networks based on original YOLO lost function via open source Caffe implementation named caffe-yolo.

a) GoogLeNet+ConvDet model. We use GoogLeNet model from Caffe's Model Zoo for feature extraction and then apply fine tuning strategy to the pre-trained network. We have trained the detection model for 32000 iterations on PASCAL VOC 2007 and 2012 "trainval" set with input images are of size 448×448. Throughout training, we use a batch size of 16, a momentum of 0.9 and a decay of 0.0005.

b) Fast YOLO+ConvDet model. There is no Fast YOLO model pre-trained on ILSVRC2012 dataset using Caffe framework and therefore we have trained it from scratch on NVIDIA GTX-1080 for one week until it obtains a top-1 accuracy 56.128% and a top-5 accuracy 80.1% on validation set as in Fig. 1. Finally, we append ConvDet layer into pre-trained model and then fine tuning the network on PASCAL VOC 2007 and 2012 "trainval" set for the object detection task.

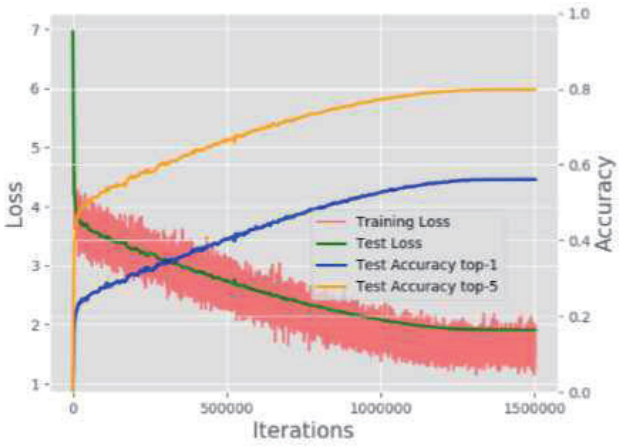

Fig. 1. Pre-trained Fast YOLO on ILSVRC2012 "trainval" set

III. Results

1. Inference accuracy

Average Precision (AP) and mean Average Precision (mAP) are standard accuracy measurement methods for object detection performance. Table 1 shows AP for each category and mAP. From mAP results, we can obtain comparable accuracy with base YOLO (63.4%) by employing GoogLeNet as backbone architecture. Meanwhile, our Fast YOLO architecture has relatively low mAP compared to original Fast YOLO (52.7%). This poor mAP can be explained by low accuracy on feature extraction layers trained on ILSVRC2012 "trainval" set.

2. Execution time.

From the result, Table 2 shows that deploying models on GPU-0 is more efficient than CPU or GPU-1. GPU-0 has twice as many shader cores as GPU-1 (4 shader cores for GPU-0 and 2 shader cores for GPU-1) .Thus, GPU-0 has twice the speed of comparison for GPU-1. Additionally, CK-Caffe takes advantage of OpenCL API which helps deep learning framework can be run more efficiently on GPU than CPU.

Table 1. Per-class AP and mAP results on PASCAL VOC2007 “test” set

	GoogLeNet + ConvDet(%)	Fast YOLO + ConvDet(%)
aeroplane	64.65	50.59
bicycle	67.14	48.59
bird	56.28	27.70
boat	44.94	23.25
bottle	28.52	12.43
bus	68.51	44.25
car	66.72	48.61
cat	76.88	58.99
chair	35.09	20.14
cow	51.39	30.79
diningtable	57.48	39.59
dog	69.00	52.71
horse	73.20	57.14
motorbike	60.75	47.95
person	56.17	40.19
pottedplant	30.65	14.54
sheep	54.05	32.06
sofa	61.89	43.95
train	77.23	53.71
tvmonitor	60.72	35.51
Mean AP	58.06	39.15

Table 2. Execution time for each image on PASCAL VOC2007 “test” set.

	GoogLeNet + ConvDet (secs)	Fast YOLO + ConvDet (secs)
CPU	7.651	6.157
GPU-0	7.519	1.605
GPU-1	14.654	2.924

IV. Conclusions

In our work, by leveraging YOLO algorithm, we have trained GoogLeNet and Fast YOLO-based models and deployed the trained models on ARM Mali-T628 MP6 GPU of ODROID-XU4. We have experimented and found that the real-time object detection can be deployed on resource-constrained ARM-based GPU. In the future, we will investigate YOLO9000[8] to improve our results.

References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Rarha°ł. 2015. You Only Look Once: Unified, Real-Time Object Detection. (2015). arXiv:arXiv:1506.02640

[2] https://pjreddie.com/darknet/

[3] https://github.com/yeahkun/caffe-yolo/

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Janathan Long, Foss Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. (2014). arXiv:arXiv:1408.5093

[5] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2014. Going Deeper with Convolutions. (2014). arXiv:arXiv:1409.4842

[6] Anton Lokhmotov and Grigori Fursin. 2016. Optimizing Convolutional Neural Networks on Embedded Platforms with OpenCL. In Proceedings of the 4th International Workshop on OpenCL (IWOCL‘16), ACM, New York, NY, USA, Article 10, 4 pages. https://doi.org/10.1145/2909437.2909449

[7] Alex Krizhevsky, Ilya Sutsskever, and Geoffrey E.Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS’12). Curran Associates Inc.,USA, 1097-1105. http://dl.acm.org/citation.cfm?id=2999134.2999257

[8] Joseph Redmon and Ali Farhadi. 2016. YOLO9000: Better, Faster, Stronger. (2016). arXiv:arXiv:1612.08242