International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 11 (2019), pp. 1945-1949
© International Research Publication House. http://www.irphouse.com

DroidDet: Reducing Convolutional Neural Network for Object Detection on Embedded Devices

Trunghai Do and Hyungshin Kim*

Department of Computer Science and Engineering, Chungnam National University, Korea.

** Corresponding author: Hyungshin Kim (ORCID: 0000-0001-9615-1644)*

Abstract

Convolutional neural networks (CNNs) become the center of many computer vision applications to solve a variety of tasks including object detection. However, it is difficult to deploy neural networks effectively on embedded devices with limited hardware resources. In this paper, we propose DroidDet a lightweight fully convolutional neural network that adopts YOLO object detection algorithm for the ARM Mali-T628 GPU of ODROID-XU4 board. Our DroidDet is constructed by strongly considering about the balance between accuracy, detection speed and model size with respect to the constrains in hardware resources.

Keywords: on-device neural network, object detection, ARM GPU, Neural network embedding, CNN reduction

I. INTRODUCTION

In computer vision applying convolutional neural network, there are several approaches for inference. We can infer on the device or through the cloud. However, if there is no internet connection between the device and the cloud, there is no other way than to infer directly on the device. The consequence of including trained model in the application package is that the size of the application will be significantly increased. Furthermore, to update model regularly, model size must be small.

All autonomous systems require efficient real-time object detection algorithms to guarantee control without delay. Therefore, speed is one of the most significant features of object detection tasks. Besides, through ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenges, CNN architectures [1, 2, 3] have been proven that we can achieve high accuracy by stacking more convolutional layers in the network architecture. This fact is also true in object detection. Consequently, the depth is the second important factor needs to be considered as designing the network. However, the very deep neural network applications are only suitable for deploying on back-end equipped with high-end GPUs. It is difficult to move directly those deep neural network to resource-constrained systems without modifying the architecture. For instance, we cannot deploy too large and deep neural network onto embedded devices because we have to keep small model size and fast detection speed while maintaining an acceptable accuracy of the model.

In this paper, we propose DroidDet a lightweight fully convolutional neural network that enables object detection on embedded ARM Mali GPUs. We present some considerations and make balance between the accuracy, detection speed and model size when designing a convolutional neural network architecture for embedded systems. Although there are a number of studies aimed at deploying neural network on embedded devices, their works have just been experimented on high-end NVIDIA GPUs. Different from them, we suggest a simple approach to train our DroidDet on NVIDIA GTX-1080 GPU and deploy the trained model on ARM Mali GPU of ODROID-XU4 board for the object detection task.

II. RELATED WORK

A research [4] relied on Fast R-CNN [5] is made to fit the specific platform and achieves trade-off between speed and accuracy on embedded system. They have taken advantage of the computing power of the Jetson TK1 platform to obtain the best performance in Low-Power Image Recognition Challenge (LPIRC). LCDet [6] is a low-complexity object detector for the purpose of detecting objects on mobile devices. They design and develop an end-to-end TensorFlow-based fully convolutional deep neural network for object detection inspired by YOLO [7]. Although their work is to deploy trained model on Snapdragon 835 including TensorFlow-optimized Hexagon 682 DSP, they present inference speed results on NVIDIA K40. In autonomous driving vehicle, some basic requirements for image object detectors include accuracy, speed, small model size and energy efficiency. An another fully convolutional neural network for object detection named SqueezeDet [8] is based on the YOLO algorithm to address these requirements. They implemented model training, evaluation, error analysis and visualization pipeline using Tensorflow compiled with the cuDNN computation kernel on NVIDIA TITAN X GPU.

All of the works aforementioned target embedded systems, but none of them are tested on embedded devices without equipped with NIVIDA GPU. It is still a long way from research to deploy work in the field. Our study is different from them in training model on high-end NVIDIA GPU but deploying trained model on embedded device named ODROID-XU4.

III. METHOD

We base on YOLO algorithm to design the DroidDet network architecture for the object detection task. We design the architecture, train and deploy the trained model as follows.

III.I Designing network architecture for object detection

CNN-based object detection models are typically fine-tuned from which they have been pre-trained on ILSVRC dataset for the image classification task. Caffe Model Zoo is a collection of pre-trained models tailored to our needs. However, Tiny YOLO and our DroidDet are built based upon Darknet Reference model which is absent from Caffe Model Zoo. Hence, we need to train this model from the scratch. Details for this training will be presented on Section III.3. We show here the benchmark for each model on image classification task first to find out some candidates for the backbone architecture and the threshold for object detection.

Typically, inference time for the object detection task cannot be faster than the classification task, and the size of an object detection model cannot be smaller than the size of the backbone image classification model. From Table 1, we can find useful information for selecting the backbone model. AlexNet and CaffeNet all have low memory requirements and fast inference speed, but their model size is too large. If we stack more convolutional layers on these networks for detection, the model size becomes larger, and thus we do not choose these models. MobileNet version 1 and version 2 [9] all achieve high accuracy, but the memory required for data is too large and the models take long time to predict the image category. Other networks including GoogLeNet_v1, SqueezeNet_v1.1 and Darknet Reference have acceptable balances between model size, memory required for data, execution time and accuracy. Therefore, we take these models to experiment our proposed method.

In the object detection tasks, images are typically fed through CNN network to produce feature maps before passing detection layers. Both Base YOLO and Tiny YOLO networks use fully connected layers for detection. We will analyze the number of parameters in detection layers of these networks as follows.

Suppose that the input feature map is of size (W_f,, H_f, Ch_f) where W_f and H_f are the width and height of the corresponding feature map, and Ch_f is the number of input channels to the detection layer. Base YOLO detection layer consists of two fully connected layers. Assume the number of output from the first fully connected layer (fc1) is F_{fc1}, then the number of parameters in the fc1 layer is $W_f \times H_f \times Ch_f \times F_{fc1}$. The second fully connected layer generates C conditional class probabilities and (B × 5) bounding box coordinates and confidence scores for each $W_o \times H_o$ grids. Therefore, the number of parameters in the fc2 layer is $F_{fc1} \times W_o \times H_o \times (B \times 5 + C)$. The total number of parameters in detection layers of Base YOLO is:

$$F_{fc1} \times (W_f \times H_f \times Ch_f + W_o \times H_o \times (B \times 5 + C)) \quad (1)$$

Tiny YOLO employs a neural network with fewer convolutional layers and fewer filters than Base YOLO and uses one fully connected layer for detection. Denote the shape of the input feature map as (W_f, H_f, Ch_f) and the output for detection layer is F_{fc1}, the number of parameters required by detection layer is:

$$F_{fc1} \times W_f \times H_f \times Ch_f \quad (2)$$

Instead of using the fully connected layer for detection, we replace all of them with a convolutional layer and named it *ConvDet*. Denote the width of ConvDet is F_w and the height is F_h. We maintain the output shape of ConvDet as the spatial dimension of the input feature map and compute (B × 5 + C) outputs for each grid, and hence the number of parameters required by the ConvDet layer is:

$$F_w \times F_h \times Ch_f \times (B \times 5 + C) \quad (3)$$

The number of parameters are specified in Table 2.

Table 1. Benchmarking models on the image classification task on ODROID-XU4 GPU-0.

Model name	Input image size (pixelxpixel)	Model size (MB)	Memory required for data (MB)	Execution time (Seconds)	Top-1 Accuracy (%)
AlextNet	227x227	243.9	8.3	1.87	56.9
CaffeNet	227x227	243.9	6.8	1.83	57.1
GoogLeNet_v1	224x224	53.5	55.2	3.58	68.7
MobileNet_v1	224x224	17	81.3	10.43	70.8
MobileNet_v2	227x227	14.2	144.5	15.67	71.9
SqueezeNet_v1.0	227x227	5.0	53.1	2.22	57.5
SqueezeNet_v1.1	227x227	5.0	32.3	1.28	57.5
Darknet Reference	224x224	29.3	27.8	1.37	56.2

International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 11 (2019), pp. 1945-1949
© International Research Publication House. http://www.irphouse.com

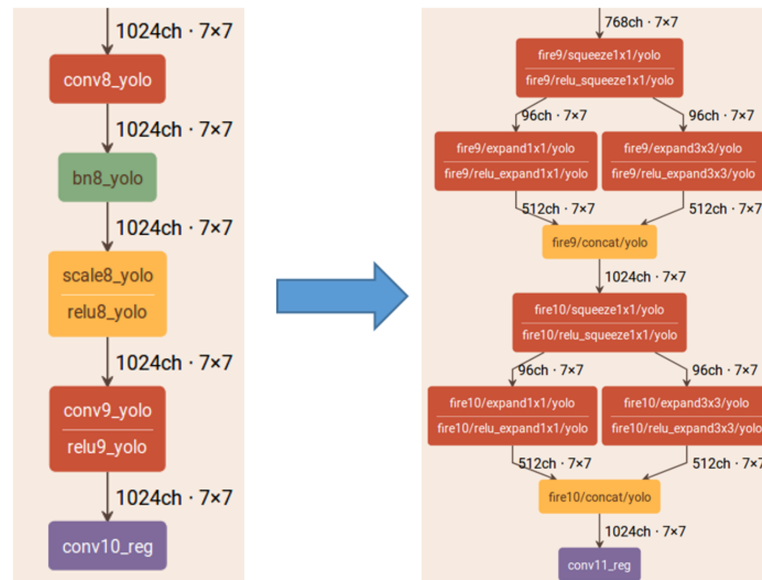

Fig 1. Modification for convolutional layers at the back of the network

In addition, we employ Darknet Reference as our backbone model. However, convolutional layers at the back of Darknet Reference network typically have more parameters than the previous layers. If we add one or more layers based on its network architecture design methodology, the model size will be significantly increased. Therefore, we have applied the Fire module [10] from SqueezeNet in this case. As illustrated in Figure 1, although the shape of a output feature map (1024 x 7 x 7) after each Fire module equal to the shape of the output feature map, the number of parameters is decreased significantly from (18.8 x 10^6) in the original convolutional layers to (1.2 x 10^6) in the Fire modules.

Table 2. Parameters in detection layers

	# of parameters
Base YOLO	212 x 10^6
Tiny YOLO	73 x 10^6
ConvDet	0.28 x 10^6

By applying this method, our DroidDet model size drops dramatically from 101 MB to 14.9 MB without sacrificing accuracy, but the execution time increases by 0.1 seconds. The Fire module is effective for reducing the model size but ineffective in the inference speed. We will show the detection time for SqueezeDet model using Fire module for all layers in Table 2. That is why we only apply the Fire modules for the heaviest layers in our network architecture.

III.II Deep learning framework

Most of the well-known deep learning frameworks like Caffe, Caffe2, TensorFlow and Torch take advantage of the CUDA library. As such, NVIDIA GPUs become irreplaceable parallel computing hardware for deep learning applications. However, at the time of inference, we need to deploy our trained models to a variety of target devices that are not well supported by high-end NVIDIA GPUs, especially in embedded instances. In this work, we choose CK-Caffe [11] as a deep learning framework based on Caffe. Unlike pure Caffe, CK-Caffe runs on the OpenCL API supporting many different architectures: CPU, GPU, DSP, FPGA, and custom accelerator. At the training stage, we can easily define the Caffe model architecture as configuration files with the Protocol Buffer language and train the model on powerful NVIDIA GTX-1080 GPU. Once a CNN model is trained, it can also easily be deployed on the embedded ARM Mali-T628 MP6 GPU of ODROID-XU4.

III.III Training object detection models

Note that GoogLeNet+ConvDet is the GoogLeNet-based object detection network using ConvDet layer for detection and Tiny YOLO+ConvDet is the Tiny YOLO-based object detection network using ConvDet for detection layer. In this section, we will cover training phase for GoogLeNet+ConvDet, Tiny YOLO+ConvDet, SqueezeDet and our DroidDet network. We fine-tune the pre-trained models on the PASCAL VOC 2007 and 2012 "trainval" dataset. Every image is resized to a fixed image size of 448 x 448 before passing through the network.

III.III.I GoogleLeNet+ConvDet model

We use the GoogLeNet model from the Caffe Model Zoo. We employ this model for feature extraction and apply ConvDet layer then fine-tune the pre-trained network for object detection. We have trained the detection model for 32,000 iterations. During the training phase we use a batch size of 16.

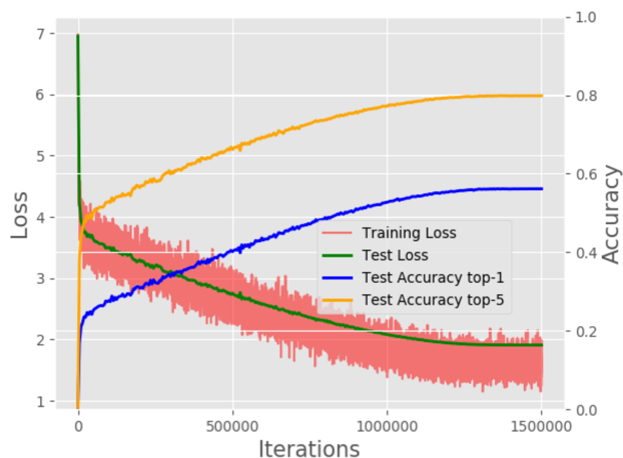

Fig. 2. Pre-trained Darknet Reference on ILSVRC2012 “trainval” set

III.III.II SqueezeDet model

We also experiment on the SqueezeDet model. This model is built based on the SqueezeNet network. We have trained the SqueezeDet model for 100,000 iterations with the base learning rate of 0.001 using "poly" learning policy.

III.III.III Tiny YOLO+ConvDet model and DroidDet model

Both Tiny YOLO+ConvDet and our DroidDet network architecture are built on Darknet Reference network. However, there is no Darknet Reference pre-trained weight on the Caffe Model Zoo, and thus we have to train it from scratch. We have pre-trained Darknet Reference model for the image classification task on ILSVRC 2012 dataset.

We employ the Caffe framework and train model on NVIDIA GTX-1080 for about a week until we obtain a Top-1 accuracy of 56.1% and a Top-5 accuracy of 80.1% on validation set as illustrated in Figure 2. After obtaining pre-trained Darknet Reference model, we use it as a backbone model to train Tiny YOLO+ConvDet and our DroidDet model as follows:

(i) Tiny YOLO+ConvDet model: First, we add 2 more convolutional layers, each with 3 x 3 kernels to produce 1024 output channels and then add a ConvDet layer for detection.

(ii) Our DroidDet model: First, instead of adding 2 more convolutional layers, we add 2 more Fire modules as mentioned above, and then add a ConvDet layer which is used to detect objects.

We fine-tune both models for 100,000 iterations with a base learning rate of 0.001 and the "poly" learning policy on PASCAL VOC dataset for the object detection task.

IV. EXPERIENTS AND RESULTS

In this section, we report the model size, the accuracy and the detection time for each model. All experiments in this section are performed on the ODROID-XU4 board.

IV.I Inference Accuracy

We evaluate our trained object detection models on PASCAL VOC 2007 "test" set. Each network predicts 7x7x2 = 98 bounding boxes per image and 20 conditional class probabilities for each grid cell. After passing through the network, there are multiple detections for one objects. To remedy this situation, we applied Non-Maximum Suppression (NMS) algorithm for every detected box. We use mAP (mean Average Precision) as standard metric to measure the object detection accuracy. From the mAP results illustrated in Table 3, we observe that we can achieve comparable accuracy with Base YOLO(63.4%) by employing GoogLeNet as the backbone architecture. Meanwhile, other models and our DroidDet model have mAP relatively lower than original Tiny YOLO(52.7%). Poor mAP results on our DroidDet model may be due to the simplicity of preprocessing images before passing through the network. We just resize the images to a fixed size of 448 x 448 and subtract the mean image from each pixel.

IV.II Inference time

The ODROID-XU4 is equipped with four big cores (ARM Cortex-A15 up to 2.0 GHz) and four small cores (ARM Cortex-A7 up to 1.4 GHz).

Table 3. The accuracy, inference time and model size for each model testing on PASCAL VOC 2007 “test” set on ODROID-XU4

	Inference time (seconds/image)				
Model name	CPU	GPU-0	GPU-1	mAP (%)	Model size (MB)
GoogLeNet+ConvDet	7.447	7.519	14.654	58.06	100.5
SqueezeDet	0.994	3.825	7.488	42.26	10.5
Tiny YOLO+ConvDet	1.559	1.605	2.924	39.15	101.8
DroidDet	1.359	1.722	3.209	39.42	14.9

International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 11 (2019), pp. 1945-1949
© International Research Publication House. http://www.irphouse.com

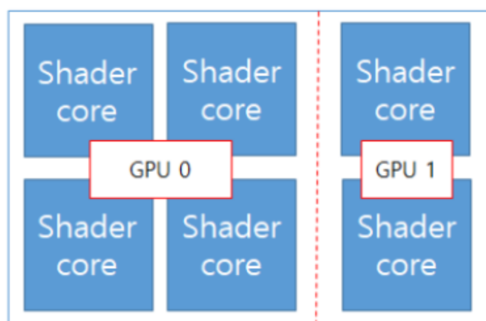

Fig. 3. ARM Mali-T628 MP6 GPU architecture of ODROID-XU4

The ODROID-XU4 is also equipped with the ARM Mali-T628 MP6 GPU supporting OpenCL 1.2 Full Profile. The MP6 version has six clusters/shader cores (4 shader cores for GPU-0 and 2 shader cores for GPU-1 as shown in Figure 3) with a clock rate up to 600 MHz. From the evaluation of the inference time, we see that our DroidDet model can achieve the same inference time as Tiny YOLO+ConvDet model. Compared to the Tiny YOLO+ConvDet model, even though the detection speed of our DroidDet model is slower than 0.1 second, its accuracy is better and its model size has been significantly reduced as shown in Table 3. In addition, Table 3 also shows that deploying models on GPU-0 is more efficient than GPU-1. The GPU-0 has twice as many shader cores as the GPU-1 and thus the GPU-0 is twice as fast as the GPU-1. Although CPU has 8 cores with up to 2.0 GHz, while GPU-0 has only 4 cores with a frequency of 600 MHz, the inference time on GPU-0 is similar to the inference time on CPU.

IV.III Limitations

The first limitation is that our experiment has just been shown on the ODROID-XU4 board while there are many different embedded devices. However, we believe that our approach can be successfully applied to any embedded GPUs supported by the OpenCL framework. The second one is our trained model achieves quite low accuracy comparing with original YOLO implementation. Nevertheless, to achieve the low latency and small model size, we need to sacrifice the accuracy a little bit. The last limitation is the latency in inference, it still has a large gap between object detection on high-end GPUs and embedded GPUs, and therefore, we have not obtained real-time object detection for embedded GPUs.

V. CONCLUSION

In this paper, we have emphasized the necessity of object detection in the real world. We also have identified the problems because we need to deploy the convolutional neural network-based object detection on embedded devices. By recognizing those challenges and considering previous works in designing CNN architecture for embedded systems, we come up with our design method to make the balance between accuracy, latency and model size. Although our study still has some drawbacks, we have proposed DroidDet which is a lightweight fully convolutional neural network for the object detection task. Our model can be trained on high-end NVIDIA GPU but can also be easily deployed on limited ARM Mali GPU. In the future, we plan to apply some other techniques like parameters pruning, quantization and Huffman coding to even more reduce the model size.

ACKNOWLEDGEMENT

This work was supported by the research fund of Chungnam National University.

REFERENCES

[1] R. Girshick et al, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol.38, No.1, (2016), pp.142-158

[2] Szegedy, Christian et al, "Going DeeperWith Convolutions", Proc, of the IEEE Conference on Computer Vision and Pattern Recognition, (2015).

[3] Karen Simonyan and Andrew Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", arXiv preprint, arXiv:1409.1556,(2018).

[4] Mao, Huizi, et al, "Towards Real-Time Object Detection on Embedded Systems", IEEE Transactions on Emerging Topics in Computing, Vol.6, No.3,(2017), pp.417-431.

[5] Girshick and Ross, "Fast R-CNN", Proceedings of The IEEE International Conference on Computer Vision, (2015).

[6] S. Tripathi et al, "Fast R-CNN", Proc. of The IEEE Conference on Computer Vision and Pattern Recognition Workshops, (2017), pp.411-420.

[7] J. Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", Proc. of The IEEE Conference on Computer Vision and Pattern Recognition, (2016), pp.779-788.

[8] BichenWu et al, "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving", Proc. of The IEEE Conference on Computer Vision and Pattern Recognition Workshops, (2017), pp.446-454.

[9] Andrew G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", arXiv preprint, arXiv:1704.04861, (2017).

[10] Forrest N. Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 1MB model size", arXiv preprint arXiv:1602.07360, (2016).

[11] Lokhmotov, Anton and Fursin, Grigori, "Optimizing Convolutional Neural Networks on Embedded Platforms with OpenCL", Proc. of The 4th International Workshop on OpenCL, (2016).