CNN-based object detection solutions for embedded heterogeneous multi-core SoCs

Cheng Wang, Ying Wang, Yinhe Han, Lili Song, Zhenyu Quan, Jiajun Li and Xiaowei Li
State Key Laboratory of Computer Architecture
Institute of Computing Technology, Chinese Academy of Sciences, Beijing, P.R. China
{wangcheng, wangying2009, yinhes, songlili, quanzhenyu, lijiajun, lxw}@ict.ac.cn

Abstract - This paper surveys how to use Convolutional Neural Networks (CNN) to hypothesize object location and categorization from images or videos in mobile heterogeneous SoCs. Recently a variety of CNN-based object detection frameworks have demonstrated both increasing accuracy and speed. Though they are making fast progress in high quality image recognition, state-of-the-art CNN-based detection frameworks seldom discuss their hardware-depended aspects and the cost-effectiveness of real-time image analysis in off-the-shelf low-power devices. As the focus of deep learning and convolutional neural nets is shifting to the embedded or mobile applications with limited power and computational resources, scaling down object detection framework and CNNs is becoming a new and important direction. In this work we conduct a comprehensive comparative study of state-of-the-art real-time object detection frameworks about their performance, cost-effectiveness/energy-efficiency (in the metric of mAP/Wh) in off-the-shelf mobile GPU devices. Based on the analysis results and observation in investigation, we propose to adjust the design parameters of such frameworks and employ a design space exploration procedure to maximize the energy-efficiency (mAP/Wh) of real-time object detection solution in mobile GPUs. As shown in the benchmarking result, we successfully boost the energy-efficiency of multiple popular CNN-based detection solutions by maximizing the utility of computation resources of SoC and trading-off between prediction accuracy and energy cost. In the second Low-Power Image Recognition Challenge (LPIRC), our system achieved the best result measured in mAP/Energy on the embedded Jetson TX1 CPU+GPU SoC.

I Introduction

Recently, the explosive progress in of deep learning becomes an important momentum for the development of precise object detection for complex real life images and video, and it is beginning to enable intelligent image recognition accuracy surpassing the level of human-being.

Since 2012, convolutional neural networks (CNNs) based image recognition solutions are showing increasingly higher image classification accuracy on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [1, 2]. However, with the growth of depth and width of convolutional neural networks, the increasing computation and storage overhead it takes to deploy large scale deep neural networks (DNN) and convolutional neural networks (CNN) models significantly limits the performance and feasibility of such detection solutions in mobile devices whose primary concern is energy-efficiency due to the resource and power constraint.

There is a growing concern in deep learning area about how to provide powerful and cost-effective vision recognition capability in low-end mobile, embedded or even cutting-edge IoT devices with thrifty power budget. In this sense, low power recognition task with mobile devices is different from the conventional image recognition solutions focused on accuracy, which often runs in server-end or desktop level computers with constant power supply. Comparatively, for lightweight mobile devices that have limited computation strength and power budget, how to choose an appropriate CNN-based detection software framework, adjust their critical parameters and exploit the full potential of hardware with balanced task-partitioning and mapping are very critical to create the most efficient object detection solution suitable for this scenario.

In this paper, instead of focusing solely on the localization and categorization accuracy, we investigate and compare several state-of-the-art CNN based frameworks for object detection and their performance/efficiency in popular heterogeneous multi-core SoC. Jetson TX1, and we also investigate several possible methods to adjust their critical parameters to achieve the maximum level of energy-efficiency (defined as **mAP/Wh**).

To understand the resilience of CNN-based object detection and their potential in providing embedded intelligent vision, a design parameter exploration method is conducted to customize the surveyed CNN-based object-detection solution that yields the best efficiency in Jetson TX1. For example, we tried CNN hyper-parameter tuning, dimension reduction techniques and other input-level approximation methods, and open up a huge space of design parameters to devise an object detector based on CNN. With sufficiently large design space, we can search for proper design points in order to fulfill different tasks with varied performance constraint, corresponding to different tracks in ILSVRC.

As an illustration, the design strategy helps provide the solutions for second Low-Power Image Recognition Challenge (LPIRC). In the contest, we employ two different object detection architectures: BING +FAST-RCNN and Faster-RCNN [3] [4] [5] for the missions of track-1 and track-3 in LPIRC respectively. These two frameworks are all based on convolutional neural networks and performed on the NVIDIA Jetson-TX1 development board. The strategy for building a low-power but powerful object detection system in contest is to trade-off between recognition accuracy and model complexity, and maximize the energy-efficiency (EE) of heterogeneous CPU+GPU SoC through EE-driven model design space exploration.

As shown in the benchmarking result, after thorough design space exploration, we successfully boost the energy-efficiency of both Fast R-CNN and Faster R-CNN by maximizing their utility of computation resources of SoC and trading-off between prediction accuracy and energy cost. Meanwhile, we also drawn conclusions from the experimental results, which are not perfectly consistent with the claims in prior literature. We believe that the findings not only give a user-perspective view of the performance and accuracy of state-of-the-art CNN-based object detection solutions, but also provide useful guidelines for those searching for their best-effort image recognition solutions in mobile GPUs.

The rest of the paper is organized as follows. We introduce the representative object detection methods in Section II, and in Section III present the design parameters that can be adjusted to improve the efficiency of object detection framework on Jetson TX1. Section IV show how we finalize the solutions for the challenge. Section V concludes this paper.

II. SOFTWARE FRAMEWORK OF CNN-BASED OBJECT DETECTION METHODS

A. *Fast R-CNN and Faster R-CNN*

With the advancement of deep convolutional neural networks (CNNs), more and more new detection methods are emerging, such as Fast R-CNN, Faster R-CNN and YOLO. Fast R-CNN and Faster R-CNN are mainly based on the R-CNN framework [1]. As shown in Fig. 1, they include three steps: 1) coarse-grained region proposal extraction; 2) CNN feature extraction and object classification; and 3) fine-grained bounding box regression. Fast R-CNN proposed by Ross et al. is an enhancement over the famous framework of object detection called region-based CNN (R-CNN). An important feature is that Fast R-CNN needs a coarse-grained region proposal extraction method, such as Selective Search, EdgeBox, BING and so on. Among them, BING proposed by Cheng et al. is a very fast method for extracting object proposals. According to the paper, BING generates a small set of category-independent, high quality windows. Such coarse-grained proposals generated by BING or Selective Search extract the possible coordinates for Fast R-CNN to generate region proposals, which are called “bounding box.

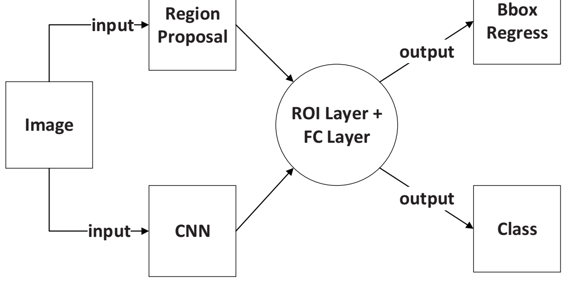

Fig. 1. Framework of Fast R-CNN and Faster R-CNN.

Running Fast R-CNN is complicated by partitioning the detection task into two separate steps: proposal search and R-CNN. However, it also bring an opportunity to better exploit the heterogeneous multi-core SoCs with both CPU cores and GPU cores. For example, Selective Search or Bing can be scheduled on CPU while the task of Fast R-CNN can be mapped to GPU, so that we can pipeline the two tasks and hide the performance overhead of proposal searching with CNN inference.

As an improvement, Ren and Girshick propose Faster R-CNN based on Fast R-CNN. Faster R-CNN introduce a novel Region Proposal Networks (RPNs) to compute proposals more correctly, which is the only difference to Fast R-CNN. RPN, considered as the highlight of Faster R-CNN, calculates proposals with a deep neural net and shares full-image convolutional features with the detection network to obtain faster speed of generating proposals. Because RPNs can share convolutions with RCNN, the marginal cost for obtaining proposals are claimed to be very small. Thus, Faster R-CNN is thought faster than Fast R-CNN because they get rid of the conventional bounding box searching procedure and embed the function into the network.

B. *You only look once (YOLO) and Single Shot MultiBox Detector (SSD)*

You only look once (YOLO)[6] is a one-stage detection solution based on convolutional net. YOLO reframes the object detection problem as a single regression problem, from image pixels to bounding box coordinates and class probabilities at the same time. A unique feature of YOLO is that it unifies the separate components of object detection into a single neural network. The neural network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image simultaneously. Compared to traditional methods of object detection, YOLO is extremely fast. It claims to run at 45-120 frames per second speed with no batch processing on a Titan X GPU.

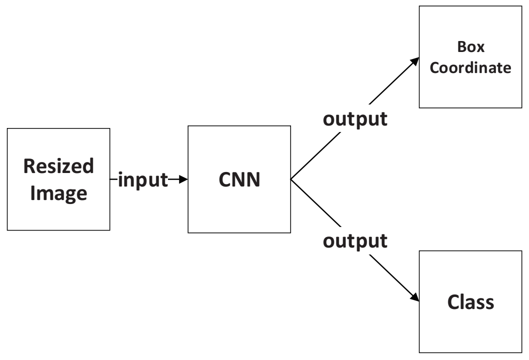

Fig. 2. The framework of YOLO.

The SSD [7] approach is also built based on convolutional neural network that produces a fixed-size set of bounding boxes and scores for the categorization of those boxes. The SSD is a relatively simple methods that finalizes object proposals because it completely eliminates proposal generation and the subsequent feature resampling operations in the network.

C. *The performance of different detection framework on*

high-end GPU devices

Table I shows the comparison between Faster R-CNN, Fast-YOLO and SSD with more powerful GPGPU, K40. SSD300 outperforms Faster R-CNN in both speed and accuracy. Fast-YOLO is the fastest solution that can run at 155 FPS, but it has much lower accuracy than Faster R-CNN and SSD. This table shows the commonly accepted conclusions in high-end GPU devices. It is shown that Faster R-CNN and other frameworks will outperform Fast R-CNN in both accuracy and speed. However, it is to be shown that in different platform, different set-ups and different constraint, the conclusion may be different. The space exploration stage introduced in this work mainly uses Fast R-CNN as illustration, some of the conclusions and techniques can be directly applied to other frameworks like Yolo and SSD, which is proposed only recently.

TABLE I
Results on Pascal VOC2007 test set. Times were measured on an Nvidia K40 GPU [7]

Solution	mAP(%)	FPS	# Boxes
Fast R-CNN(S)	57.1	16	2000
Faster R-CNN(ZF)	62.1	17	300
Fast-YOLO	52.7	155	98
SSD 300	72.1	58	7308

IV. DESIGN SPACE EXPLORATION FOR ENERGY-EFFICIENT IMAGE RECOGNITION

In this work, we aim to boost the energy-efficiency of CNN-based detection solution by maximizing the utility of computation resources of SoC and trading-off between prediction accuracy and energy cost. The metric we use to evaluate energy efficiency is mAP/energy. mAP is the prediction accuracy metric of object detection methods, and energy means the energy spent on the detection task.

Conventional detection methods mainly focus on improving accuracy or reducing complexity on condition that the accuracy is not severely compromised. However, in our stage of design space exploration, the first goal is to create a large enough space with both design points that focus on speed and those focus on accuracy without worrying about the other metric, so that we will have a better chance of obtaining the most energy-efficient solutions from the space. In this section, all experiments runs on Jetson TX1. All results of accuracy are tested on the ILSVRC 2013 validation set unless specified otherwise.

A. Region Proposal method

Fast R-CNN need a coarse-grained region proposal extraction method to obtain enough bounding boxes, such as Selective Search, EdgeBox, BING and so on. In table II and table III we can see that using unmodified Selective Search to scan an image from ImageNet library takes an average of 1 second on Xeon CPU and even up to 50 seconds for the ARM cores in Jetson TX1. It is incompetent in terms of speed for any real-time application, and thought as the major bottleneck when compared to the other detection methods. The speed of EdgeBox is at the same level of Selective Search and they are supposed to provide the “slow-but-accurate” solutions. Thus, other choices, “fast but less accurate”, must be found to create more design possibilities for less powerful Jetson TX1. In contrast to EdgeBox and Selective Search, BING is the fastest region proposal method of all the above methods, however, it is also less accurate than the other counter parts. When the general goal of energy-efficiency is prioritized over the other single-object metric, BING is expected to contribute to more speed-up and energy-efficiency gains as we show in latter sections.

The original BING extracts features of different scale rectangles on the whole image. To make it even faster, we directly crop the image to half size. After that, the speed of BING is almost increased more than 100 percent. Moreover, we try to store the top 50-300 scored proposals for further use in Fast R-CNN. We found that accuracy falls fast at the beginning and then falls slightly as the proposal count decreases (Fig 3). This experiment shows that feeding the deep classifier with more proposals help accuracy about 6 percent. However it could make speed slower about 100 percent. In scenarios when the application is more sensitive to speed, it is to be avoided to pass too many of proposals to the stage of CNN based bounding box classification. In latter section, Fast R-CNN assumes BING is employed to extract region proposal in default unless specified otherwise.

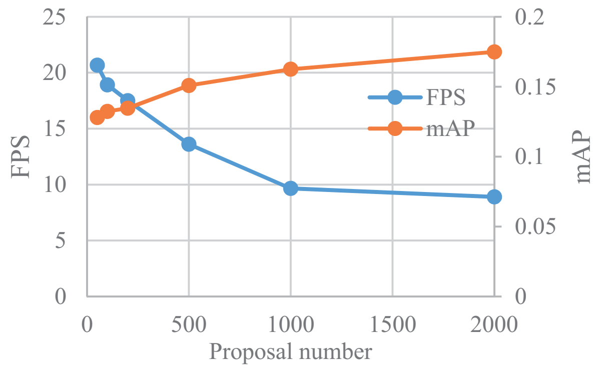

Fig. 3. Impact of proposal number on speed and accuracy with BING.

Faster R-CNN can also be accelerated by reducing the proposals. In contrast to Fast R-CNN, Faster R-CNN does not need a separate region proposal method but relies on Region Proposal Networks (RPNs) to compute proposals more correctly, which contributes to a lot of performance gains by sharing the convolution layers with the R-CNN. This feature is attractive when both region proposal search and R-CNN are running in the same core or close-coupled cores. However, for Fast R-CNN, regional proposal search can be executed in pipeline on the CPU cores of Jetson TX1 and seamlessly communicates with R-CNN on GPGPU through shared memory. In such a case, Fast R-CNN benefit from the heterogeneous parallelization across CPU and GPU. Comparatively, for Faster R-CNN, because RPNs and R-CNN are sharing a lot of network layers, it is not easy to map them separately to CPU and GPU. Therefore, the throughput of Faster R-CNN drops over that of Fast R-CNN in Jetson

TX1 in our set-up.

TABLE II
Timing (ms) on a K40 GPU, except SS proposal is evaluated in a Xeon CPU [5]

model	system	conv	proposal	total	rate
VGG	SS+Fast R-CNN	146	1510	1830	0.5fps
VGG	RPN+Fast R-CNN	141	10	198	5fps
ZF	RPN+Fast R-CNN	31	3	59	17fps

Table II summarizes the running speed of the entire object detection system. Selective search takes 1-2 seconds and Fast R-CNN with VGG-16 takes 320ms on 2000 SS proposals. Faster R-CNN with VGG-16 takes in total 198ms for both proposal and detection. RPN only takes 10ms computing the additional layers. But all these experiments are run on a K40 GPU. When we run it on Nvidia Jetson TX1, the conclusion is very different in table III. Because of the slow GPU and CPU in TX1, the RPN alone consumes a large proportion of execution time. The difference of the speed for RPN between Xeon+K40 and TX1 is mainly because RPN not only needs lots of GPU computation power, but also induces lots of operations on CPU to generate the proposals. The wimpy ARM cores in TX1 spend a plenty of time in processing the results of RPN and becomes the bottleneck of system. Therefore, the advantages of Faster R-CNN will not be that evident in TX1 when compared to Fast R-CNN.

For Fast R-CNN, if we use BING as the region proposal method, we can see in Table III, it only takes about 15 percent of total time when we run BING on CPU and run Fast R-CNN on GPU at the same. In contrast, RPN of Faster R-CNN is sharing some convolution layers with R-CNN and it is supposed to run serially on GPU cores. Thus, RPN cannot run with R-CNN at the same time. Otherwise, we has to put it to CPU cores for pipelined execution, the speed will become even less attractive because of RPN is closely coupled with the other part of network.

TABLE III
Decomposition of execution time (ms) on Nvidia Jetson TX1

model	system	conv	Proposal time	Total time	rate
Caffe Net	BING+Fast R-CNN	18	8	53	19fps
	SS+Fast R-CNN	18	49000	49044	0.000 02fps
	RPN+Fast R-CNN	19	176	220	4fps

B. *Different CNN implementations*

Different CNNs models can be fitted into Both Fast R-CNN and Faster R-CNN for the procedure after proposal search. In this work, we assume the hyper-parameter of CNNs, i.e. depth, kernel size, activation functions and other features, as an adjustable parameters in the stage of design space exploration for energy-efficient object detector on TX1. Replacing the feature extraction CNNs with customized networks can also create a large space of design points with varied detection accuracy and detection speed. Thus, in order to find the most energy-efficient object detection solution in TX1, it has to adjust the hyper-parameters of CNNs, estimate their efficiency accordingly and stop searching until the result of highest accuracy/watt is met.

Instead of enumerating all the tested networks, three representative models are chosen to evaluate Fast R-CNN: CaffeNet [10] (Fast R-CNN(C)), CaffeNet-pruned (Fast R-CNN(P)) and CaffeNet-half (Fast R-CNN(H)).

Fast R-CNN(C) uses original CaffeNet with 5 convolution layer and 3 full connect layer. The structure Fast R-CNN(P) is: K11S4C64-K5S1C128-K3S1C192-K3S1C192-K3S1C128-K3S1C128-K3S1C96-Fc2048-Fc2048-Fc1000, where “K” is for “kernel size”, “S” is for “stride” and “C” is for “channels”. Fast R-CNN(P) is trained repetitively to deliberately remove the connections with little influence on the final result so that it has a more compact weight set. In experiment with TX1, Fast R-CNN(P) is fed with ILSVRC 1000-class images from its test set, and the top-5 accuracy of this net is 0.784. The structure of Fast R-CNN(H) is K11S4C64-K5S1C128-K3S1C192-K3S1C192-K3S1C128-K3S1C128-K3S1C128-Fc512-Fc512-Fc1000. Its top-5 classification accuracy of 0.715. For both networks, we fine-tuned them using the training set of ILSVRC 2013.

The design of two networks are following the guidelines of [8]. The results have shown the priority of depth for improving accuracy. With the trade-offs, we can have a much deeper model if we further decrease width/filter sizes and increase depth. We mainly decrease the width of full connect layers and slightly increase the number of convolution layers. The impact of the three CNNs on speed and accuracy of Fast R-CNN are shown in Fig. 4.

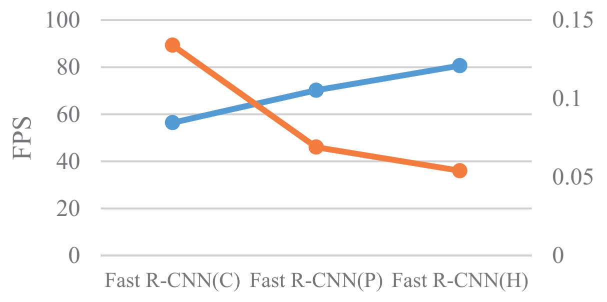

Fig. 4. Impact of different CNNs on speed and accuracy.

C. *Different batch size*

GPU have a large amount of stream cores and consequently a large throughput. Its highly parallel structure makes them more efficient on large blocks of data processing. Jetson TX1 is built based on Maxwell architecture with 256 CUDA cores delivering over 1 TeraFLOPs of performance in theory. We calculate the theoretical computing times of every layer of CNN, and we find that for CNNs of particular size using small input batch size cannot exploit the peak performance on Nvidia Jetson TX1. The best batch size for high throughput processing is different depending on the size of CNN model. For example, regarding Fast R-CNN(P), we found that processing about 30-50 images simultaneously in a batch helps to reach the possibly highest throughput on TX1. We also test the impact of batch size (from 50 to 1) input of CNN in experiments. In Fig. 5, we can see the curve is rising until batch size is 30 and after it the curve is slightly fluttered. So we can input 30 or more images at the same time and get a better performance on GPU.

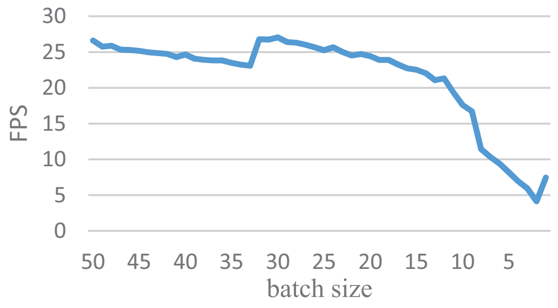

Fig. 5. Impact of different batch size on speed.

D. *Input-level approximation*

A large batch size is beneficial to performance before it quickly saturated. However, in order to support batch image input, it is better to crop all input image to a fix size. We test three different size: 224*224, 300*300 and 600*600, and show the results in Fig. 6. Besides, down-sizing the input image can directly reduce the quantity of operations in the inference of a network. The speed-up effect is obvious because the feature map size will decrease and the volume of convolutional operations drops immediately at the same ratio as image size shrinks. We can see that when image size changes from 600*600 to 300*300, the inference speed almost increase 3x and the detection accuracy of Fast R-CNN(P) only decrease about 3 percent.

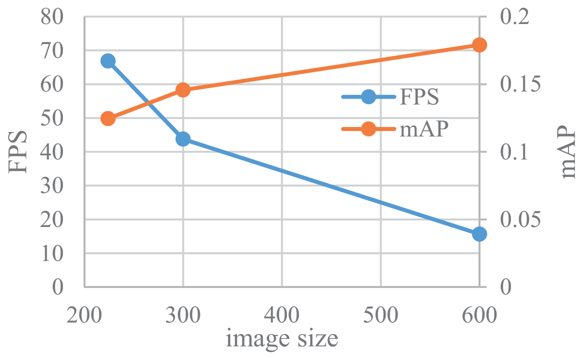

Fig. 6. Impact of different image size on speed and accuracy.

E. *Half precision and truncated singular value decomposition*

TX1 adds support for the stream cores to accelerate matrix multiplication of half-precision, and its computation throughput can be doubled in best case when full-precision operation is replaced with half-precision operation [11], so we use half-precision operation instead of full-precision operation to increase the computation throughput and storage overhead in TX1. In experiments, CaffeNet implementation with half-precision floating-point data (CaffeNet-FP16) is about 1.5x faster than CaffeNet implementation with single-precision floating-point.

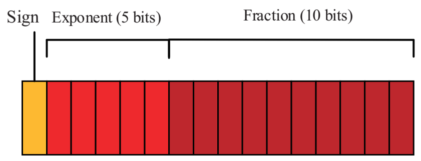

Fig. 7. The IEEE 754 half-precision floating-point format with 16 binary bits

As we observed in experiments, the influence of full connection layer on the performance of object detection is more significant than expected. It is well-known that convolutional layer dominates the propagation time in typical CNNs, but for CNN based object detection, full connection layers also plays an important role because the number of localized RoIs to process is very large and account for nearly half of the inference time when the proposal number and number of object class are huge. Fortunately, fully connected layers are easily accelerated by compressing them with truncated SVD. Depending on the proposal number, truncated SVD can reduce detection time by 30%-70% with only a small (0.3 percentage point) drop in mAP and without additional fine-tuning after model compression. In our experiment, CaffeNet with SVD is about 1.5x faster than CaffeNet without SVD.

V. Case study

A. *Task-1: High-speed object detection with server-provided images*

For track-1 of 2nd Low-Power Image Recognition Challenge (LPIRC) [12], we prepare two solutions, which are designed based on Fast R-CNN. Track-1 have 10000 images test set provided by a local server, TX-1 can fetch the image through Ethernet and process them in a total of 10 minutes.

The evaluation metric of track-1 is mAP/Wh. We formalized this task as an optimization problem whose object is mAP/Wh. *mAP* is the accuracy metric of object detection methods, and it is ratioed against the energy consumption it takes to accomplish the detection task. The accuracy is constrained by the accuracy of detection framework and it takes a lot of opportunity cost to pursue if we use powerful but large CNN models to improve it. However, the energy consumption the task needs can be easily reduced by speeding-up Fast R-CNN and finding a design point in our configuration space. In short, if we start from a powerful object detector, we can keep on accelerate it and compromise its accuracy with the method introduced in last sections. The accuracy is sacrificed until the marginal benefits of enhancing detector accuracy and relaxing detector accuracy, $\Delta mAP/\Delta Wh$ and $\Delta Wh/\Delta mAP$, are equalized.

In addition, the constraint of such a problem is that multi-threaded BING executed onto ARM cores has the same speed as the flowing Fast R-CNN, and also the image fetching time through the Ethernet, so that the task can be parallelized without bottleneck and the resources/energy on TX1 are fully utilized

Finally, we choose the fastest configuration, p is 50, s is 224, b is 50, t is 256. For H parameter, we try two networks, Fast R-CNN(C) and Fast R-CNN(P). The results of track 1 shows in table IV. In the competition, the Fast R-CNN(P) solution was ranked the 1st in all 9 solutions. 20120 images from ILSVRC 2013 training dataset were used for fine-tuning and the accuracy was calculated by mAP. The gap between the offline test results (higher mAP) and the competition results is because the result of contest are discounted by a factor.

TABLE IV
Results in track-1 of 2nd LPIRC with TX1

Solution	Accuracy	Energy(WH)	Score
Fast R-CNN(C)	0.0347	0.79	0.044
Fast R-CNN(P)	0.0260	0.94	0.028

B. *Real-time video detection with camera*

Track-3 requires the solution to use a camera to capture the screen and detect the object in it. After detection, the screen is refreshed immediately once a refresh request is sent from TX1. The sample picture captured from screen is shown is Fig. 8. The makers on the boundary of screen shows the ID of this image, detector should decode the ID, truncate it to obtain the image in center, detect it and return the categorization and proposal coordinates together with the image ID to image server. The detail can be found on www.lpirc.net.

TABLE V
Results in track-3 of 2nd LPIRC with TX1 2016 track 3

Image-size	600*600			300*300		
Proposal Number	300	100	50	300	100	50
mAP(%)	59.4	57.07	52.69	50.8	50.77	49.11
FPS	0.975	1.263	1.403	1.261	1.923	2.273

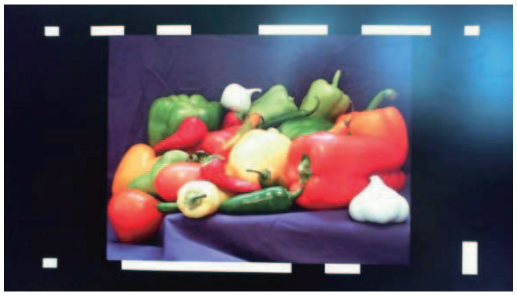
Fig. 8. Sample picture of track 3.

Though the task is different from track-1, the design space exploration problem is similar: searching for the best parameters that achieve the maximum rate of energy-efficiency. The only difference is that the constraint is a little different from the problem in track-1. The reason is due to the limited sampling speed of the used camera and also refreshing rate of monitor. Because of constraint changing, a different configuration is found for this task. The found object detector implementation cannot detect the images as fast as the one in track-1, which detect images about 50 fps. In addition, because CPU is loaded with the task of camera-shot sampling, image transformation, and ID decoding, it is preferred to use Faster R-CNN to free the CPU cores instead of Fast R-CNN that off-loads the proposal generation task to the CPU cores. The other parameters are also selected to make sure that Faster R-CNN yield the highest energy efficiency. Depending on the switch time of screen and sample time of camera, the implementation of Faster R-CNN runs as faster as the screen refresh to guarantee the CPU and GPU utility on TX1.

Finally, We evaluate three possible solutions based on Faster R-CNN, which are designed via optimization method similar to that of Fast R-CNN. As shown in table, solution 1 is the fastest; solution 3 is the most accurate; solution 2 is an accelerated version of solution 3 by replacing ZF with CaffeNet. After quantitative comparison, solution 2 is used in track 3 on the day of real competition. The results of track 3 shows in table VII.

TABLE VII
Results in LPIRC for track3

Solution	Accuracy	Energy(WH)	Score
Faster R-CNN(C)	0.00251	1.78	0.0014

VI. Conclusions

In this work, we seek to find energy-efficient object detector for mobile devices by exploring the design space of state-of-the-art CNN-based detection solutions and trading-off between prediction accuracy and energy cost. It is shown that changing the hardware platform, optimization goal and constraint has a profound impact on the efficiency of different frameworks. It is encouraged to choose the appropriate one and customize it to achieve the design goal for a particular application scenario.

Acknowledgements

This work was supported in part by National Natural Science Foundation of China under Grant No. 61432017, 61176040, 61504153, 61402146 and 61521092.

References

[1] R. Girshick, et al., “Rich feature hierarchies for accurate object detection and semantic segmentation,” in Proc. of CVPR, 2014.
[2] R. Girshick, et al., “Region-based convolutional networks for accurate object detection and segmentation,” in Trans. on PAMI, 2015.
[3] M. Cheng, et al., “BING: Binarized Normed Gradients for Objectness Estimation at 300fps,” in Proc. of CVPR, 2014.
[4] R. Girshick: Fast R-CNN. arXiv:1504.08083, 2015.
[5] S. Ren, et al., “Faster R-CNN: Towards real-time object detection with region proposal networks,” in Proc. of NIPS, 2015.
[6] J. Redmon, et al., “You only look once: Unified, real-time object detection,” in Proc. of CVPR, 2016.
[7] W. Liu, et al., “SSD: Single Shot MultiBox Detector,” arXiv:1512.02325, 2016.
[8] K. He and J. Sun, “Convolutional Neural Networks at Constrained Time Cost,” in Proc. of CVPR, 2015.
[9] M. Zeiler and R. Fergus, “Visualizing and Understanding Convolutional Networks,” arXiv:1311.2901, 2013.
[10] Y. Jia, et al., “Caffe: Convolutional Architecture for Fast Feature Embedding,” arXiv:1408.5093, 2014.
[12] Y. Wang, et al., “DeepBurning: Automatic Generation of FPGA-based Learning Accelerators for the Neural Network Family,” in Proc. of DAC, 2016.
[11] Y. Lu, et al., “Rebooting Computing and Low-Power Image Recognition Challenge,” in Proc. of ICCAD, 2015