SkyNet: a Hardware-Efficient Method for Object Detection and Tracking on Embedded Systems

Xiaofan Zhang [1] Haoming Lu [1] Cong Hao [1] Jiachen Li [1] Bowen Cheng [1] Yuhong Li [1] Kyle Rupnow [2] Jinjun Xiong [3 1] Thomas Huang [1] Honghui Shi [3 1] Wen-mei Hwu [1] Deming Chen [1 2]

Abstract

Developing object detection and tracking on resource-constrained embedded systems is challenging. While object detection is one of the most compute-intensive tasks from the artificial intelligence domain, it is only allowed to use limited computation and memory resources on embedded devices. In the meanwhile, such resource-constrained implementations are often required to satisfy additional demanding requirements such as real-time response, high-throughput performance, and reliable inference accuracy. To overcome these challenges, we propose SkyNet, a hardware-efficient method to deliver the state-of-the-art detection accuracy and speed for embedded systems. Instead of following the common top-down flow for compact DNN design, SkyNet provides a bottom-up DNN design approach with comprehensive understanding of the hardware constraints at the very beginning to deliver hardware-efficient DNNs. The effectiveness of SkyNet is demonstrated by winning the extremely competitive System Design Contest for low power object detection in the 56th IEEE/ACM Design Automation Conference (DAC-SDC), where our SkyNet significantly outperforms all other 100+ competitors: it delivers 0.731 Intersection over Union (IoU) and 67.33 frames per second (FPS) on a TX2 embedded GPU; and 0.716 IoU and 25.05 FPS on an Ultra96 embedded FPGA. The evaluation of SkyNet is also extended to GOT-10K, a recent large-scale high-diversity benchmark for generic object tracking in the wild. For state-of-the-art object trackers SiamRPN++ and SiamMask, where ResNet-50 is employed as the backbone, implementations using our SkyNet as the backbone DNN are 1.60X and 1.73X faster with better or similar accuracy when running on a 1080Ti GPU, and 37.20X smaller in terms of parameter size for significantly better memory and storage footprint.

1 Introduction [†]

Edge AI applications not only require high inference accuracy from deep neural networks (DNNs), but also ask for aggressive inference speed, throughput, and energy efficiency to meet real-life demands. These applications rely on hardware-efficient DNN design when they are deployed onto embedded systems with extremely limited computation and memory resources. Recently, we have seen intensive studies on DNN accelerators in hardware, which attempt to take advantage of different hardware design styles, such as GPUs, FPGAs and AISCs, to improve the speed and efficiency of DNN inference and training processes (Qiu et al., 2016; Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, 2016; Zhang et al., 2017a; Jouppi et al., 2017; Franklin, 2017; Zhang et al., 2018a).

Although hardware accelerators can be helpful, they are still limited by available resources to handle varied real-life applications, especially for embedded systems since most DNNs are not originally designed to be hardware-efficient. As a result, optimization starts turning to the software side, to compress DNNs for less complexities, lowering computation demands and memory footprints. Recent researches have demonstrated the possibility of using low bit-width data to represent original floating-point parameters, such as using binary and ternary networks (Courbariaux et al., 2016; Rastegari et al., 2016; Li et al., 2016; Tschannen et al., 2018; Wang et al., 2018a; Gope et al., 2019). These solutions are intended to replace the hardware-intensive floating-point multiplications by logical operations, so that DNNs can be more efficient on hardware platforms.

Researchers also investigate the network pruning strategies to reduce the redundancy of DNN structures (Han et al., 2015; 2016; Luo et al., 2017). According to the published pruning strategies, the relatively less important connections between DNN layers are discarded and network retraining is then performed to regain accuracy. Significant reductions can be achieved on the classic DNNs, such as AlexNet (Krizhevsky et al., 2012) and VGG-16 (Simonyan & Zisserman, 2014). Since the major benefit of network compression comes from the fully-connected (FC) layers, to continuously have effective pruning results for latter DNNs (e.g., GoogleNet (Szegedy et al., 2015) and ResNet (He

[1]IBM-Illinois Center for Cognitive Computing Systems Research (C3SR), University of Illinois at Urbana-Champaign, USA [2]Inspirit IoT, Inc, USA [3]IBM Research, USA. Correspondence to: Xiaofan Zhang <xiaofan3@illinois.edu>.

[†]Readers can also find our challenge report for DAC System Design Contest 2019 at (Zhang et al., 2019). Our code is open-sourced here.

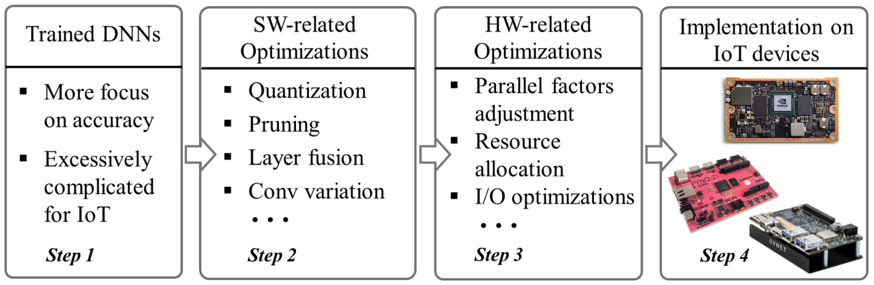


Figure 1. A top-down design flow for hardware-efficient DNN deployment on resource-constrained devices. Challenges appear between *step 2* and *3* where iterative explorations are necessary to balance DNN accuracy and performance on targeted devices.

et al., 2016)) with reduced FC layers, more sophisticated algorithms are required to be integrated in network pruning. Recently published literature adopts evolutionary algorithm (Dai et al., 2019a), alternating direction method of multipliers (ADMM) (Ren et al., 2019), and iterative pruning (Ding et al., 2018) for better compression performance while maintaining DNN accuracy.

As most of the computations happen inside the convolutional (Conv) layers, previous works also attempts to reduce the computation complexity by using depth-wise separable Conv layers for image classification and ubiquitous keyword-spotting applications (Howard et al., 2017; Zhang et al., 2017b). This depth-wise separable structure can efficiently reduce the number of operations and provide more compact DNN designs for resource-constrained hardware. To further improve the DNN deployment on hardware, layer fusion is proposed in (Alwani et al., 2016) to minimize data movements between on-chip and off-chip memory.

In general, a design process of hardware-efficient DNNs can be summarized in Figure 1 with the adoption of above-mentioned technologies. It is a top-down design flow which starts from *step 1*: to select a reference DNN with more concentrations on accuracy. For computer vision applications, the families of VGG (Simonyan & Zisserman, 2014) and ResNet (He et al., 2016) are highly likely to be selected as backbones of desired designs. Such DNNs are excessively complicated for targeted embedded systems, which must be compressed using software and hardware optimizations in *step 2* and *3*, respectively. Since software compression and hardware implementation are typically carried out in two separate steps, *step 2* and *3* are usually performed in an iterative manner to balance DNN accuracy and hardware performance on targeted devices. Network retraining is also required to regain accuracy after compression before *step 4*. Because of the iterative nature of the process, it is very challenging to cover both inference accuracy in software and deployment efficiency in hardware.

In this paper, we address the hardware-efficient DNN design problem by proposing SkyNet, a bottom-up DNN design approach with comprehensive awareness of hardware constraints. SkyNet has been demonstrated on a low power object detection task, which can deliver the state-of-the-art results for both DNN accuracy and hardware efficiency. The main contributions of this paper are summarized as follows:

- We summarize the latest low power object detectors for embedded systems and locate the potential obstacles of using top-down DNN design flows, which may prevent improved DNN accuracy and hardware efficiency.
- We propose a bottom-up design strategy of hardware-efficient DNNs for both embedded GPU and embedded FPGA; using such a design method, we propose SkyNet, which has comprehensive awareness of hardware limitations to overcome the challenges of top-down design flow.
- We demonstrate SkyNet in DAC-SDC'19 using both TX2 GPU and Ultra96 FPGA with the stat-of-the-art accuracy. SkyNet achieved the highest overall score regarding accuracy, throughput, and energy-efficiency, and won the first place winner award for both GPU and FPGA tracks.
- We extend SkyNet for object tracking. By using SkyNet as the backbone DNN, SiamRPN++ and SiamMask obtain 1.60X and 1.73X speedup with better or similar accuracy, and 37.20X smaller parameter size compared to using the original ResNet-50 backbone when running on a 1080Ti GPU.

2 Related Work

Recent state-of-the-art object detectors feature DNN backbones to extract input features. Researchers initially propose a two-stage approach: the first stage outputs multiple region proposals for object candidates and the second stage generates more accurate regions with corresponding class labels (Dai et al., 2016; Lin et al., 2017a; He et al., 2017; Cheng et al., 2018b;a; Cai & Vasconcelos, 2019; Li et al., 2019b). Since the two-stage detectors have long latency, some one-stage approaches are proposed to simultaneously regress object locations and classes to reduce latency (Sermanet et al., 2014; Redmon et al., 2016; Liu et al., 2016; Lin et al., 2017b; Law & Deng, 2018; Shen et al., 2019; Zhou et al., 2019; Tian et al., 2019). Object tracking also relies on the features extracted from powerful DNN backbones, and we

Table 1. DAC-SDC winning entries from both GPU and FPGA tracks. They follow a top-down approach, from choosing reference DNNs to applying optimization strategies on software and hardware sides, so that they compress DNNs with improved hardware efficiency. Optimizations include: ① input resizing, ② network pruning, ③ data quantization, and ④ TensorRT (Vanholder, 2016) on software, and ⑤ CPU-FPGA task partition, ⑥ double-pumped DSP, ⑦ fine-grained pipeline, ⑧ clock gating, and ⑨ multithreading on hardware.

Rank	GPU-Track	Reference DNN	Optimizations	
'19 2nd	Thinker (Xiong et al., 2019a)	ShuffleNet + RetinaNet	① ② ③	⑨
'19 3rd	DeepZS (Deng et al., 2019)	Tiny YOLO	Not clear	⑨
'18 1st	ICT-CAS (Lu et al., 2018)	Tiny YOLO	① ② ③ ④	Not clear
'18 2nd	DeepZ (Deng & Zhuo, 2018)	Tiny YOLO	Not clear	⑨
'18 3rd	SDU-Legend (Zang et al., 2018)	YOLOv2	① ② ③	⑨
Rank	**FPGA-Track**	**Reference DNN**	**Optimizations**	
'19 2nd	XJTU Tripler (Zhao et al., 2019)	ShuffleNetV2 + YOLO	② ③	⑤ ⑥ ⑧
'19 3rd	SystemsETHZ (Kara & Alonso, 2019)	SqueezeNet + YOLO	① ② ③	⑦
'18 1st	TGIIF (Zeng et al., 2018)	SSD	① ② ③	⑤ ⑥
'18 2nd	SystemsETHZ (Kara et al., 2018)	SqueezeNet + YOLO	① ② ③	⑦
'18 3rd	iSmart2 (Hao et al., 2018)	MobileNet + YOLO	① ② ③	⑤ ⑦

have seen recent Siamese network based trackers formulate trackers as feature between the exemplar image and search region (Tao et al., 2016; Valmadre et al., 2017; Wang et al., 2018b; Li et al., 2019a; Wang et al., 2019). These state-of-the-art methods make real-time object detection and tracking possible using desktop GPUs but still need aggressive compression before deploying onto embedded systems.

2.1 Low-Power Object Detectors

Nowadays, much attention has been paid to delivering hardware-efficient designs for object detection instead of simply pursuing higher inference quality. To address the design difficulties of real-life applications, a low power object detection challenge in DAC-SDC is proposed to target unmanned aerial vehicle (UAV) applications using embedded platforms, such as NVIDIA TX2 GPU, Ultra96 FPGA, and Xilinx Pynq-Z1 FPGA (Xu et al., 2019). By examining the winning entries, we notice that all of them share similar top-down DNN design approaches as shown in Figure 1.

All teams listed in Table 1 adopt one-stage detectors. Most of them start from well-established hardware-efficient DNNs, such as ShuffleNet (Zhang et al., 2018b), SqueezeNet (Iandola et al., 2016), and MobileNet (Howard et al., 2017), and replace the image classifier with YOLO (Redmon et al., 2016; Redmon & Farhadi, 2017) or RetinaNet (Lin et al., 2017b) back-end for object detection. Other solutions directly adopt the object detection algorithms, such as SSD (Liu et al., 2016) and YOLO. To deliver hardware-efficient DNNs, they employ input resizing and network pruning to lower the network complexity. Some of the GPU entries use half-precision data format (16-bit) and TensorRT for improved throughput. More aggressive compression is necessary for FPGA designs because of even tighter resource budgets. DNN parameters are quantized to around 8 bits or even down to 1 or 2 bits. The FPGA teams also cover task partitioning (between host CPU and FPGA), double-pumped DSP (with doubled working frequency in DSP units), tailored pipeline, and multithreading to boost hardware performance. One of the teams apply clock gating for even better energy-efficiency in the embedded system.

2.2 Hardware-Aware Neural Network Search

To deliver DNNs for edge devices, there has been growing interests in using neural architecture search (NAS) (Tan et al., 2019; Wu et al., 2019; Dai et al., 2019b; Howard et al., 2019; Xiong et al., 2019b; Stamoulis et al., 2019; Dong et al., 2018; Cai et al., 2018) to automatically find resource constrained convolutional neural networks (CNNs) targeting edge-platforms. Tan *et al.* (Tan et al., 2019) are one of the first to use NAS for efficient CNN by adding latency into the optimization constraint and use reinforcement learning (Zoph & Le, 2016; Zoph et al., 2018) to maximize the reward (high accuracy and low latency). To find the efficient networks for a specific platform, (Tan et al., 2019) uses real-time latency by running models on the targeted device instead of latency proxy. Limited by the number of available physical hardwares, (Wu et al., 2019; Cai et al., 2018) use look-up-table (LUT) to approximate the run-time of models on a specific device. To incorporate human knowledge, (Howard et al., 2019) uses platform-aware NAS to search CNNs for a platform and manually adjust some part of the structure to make it more efficient. Compared to previous hardware-aware NAS methods that target a specific platform, SkyNet can target both embedded GPU and embedded FPGA platforms and capture hardware limitations by using the realistic hardware performance feedbacks instead of using LUT approximation.

3 Motivations

To deliver an even better solution compared to the winning designs listed in Table 1, we investigate the potential obstacles in the top-down design flow (Figure 1) which may

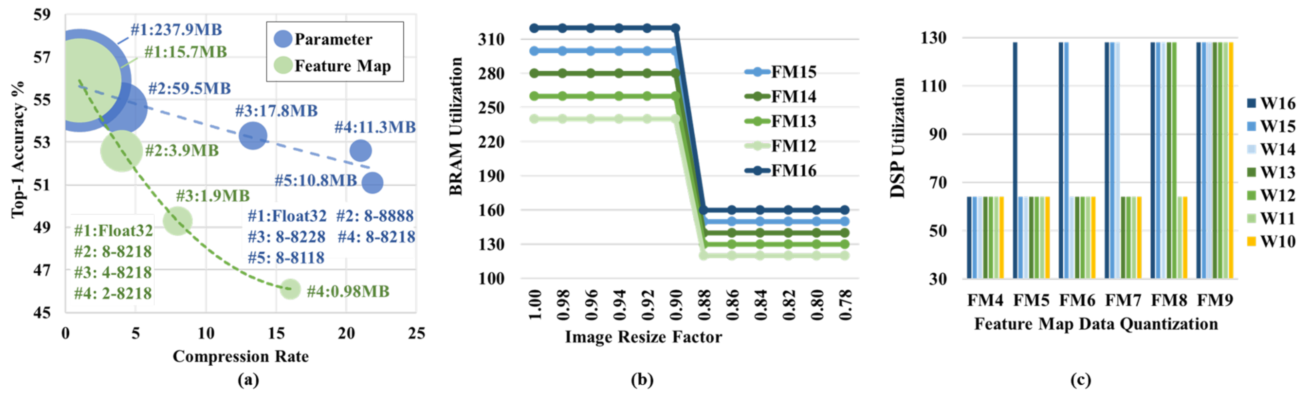


Figure 2. (a) Accuracy results for the same AlexNet under different compression schemes: blue for parameter compression and green for FM compression. The legend shows the quantization details for each design. Each model (#1∼#5) is denoted as precision p_1 for FMs across all layers, p_2 for 1st Conv parameters, p_3 for the parameters in 2nd∼5th Convs, p_4 for the parameters in 1st∼2nd FCs, and p_5 for parameters in the 3rd FC in p_1-$p_2p_3p_4p_5$ format. (b) BRAM usages of accelerators with the same architecture but 12∼16-bit quantization for feature maps (FM12∼FM16) and different image resize factors. (c) DSP utilization of accelerator using different quantizations between weights (W) and feature maps (FMs) with the numbers indicating bits allocated.

hinder further improvements on DNN accuracy and efficiency. We summarize two challenges as follow:

1) It is difficult to balance the sensitivities of DNN configurations on software and hardware during model compression following the top-down approach.
2) It is difficult to select the appropriate reference DNNs at the very beginning of the top-down flow as the uncertain accuracy variations for a given real-life task.

The first challenge causes tedious iterative explorations between software and hardware optimizations. With the similar hardware performance (e.g., throughput and latency), DNN candidates may have different accuracy results as the compression technologies are applied to different network components. We take data quantization as an example. As shown in Figure 2 (a), the accuracy results vary significantly for quantizing parameters and intermediate feature maps (FMs). In this figure, the coordinates of the bubble center represent accuracy and model compression ratio, while the area of a bubble shows data size in megabyte (MB). We scale-up the FM bubble for better graphic effect. By compressing the model from float32 to fixed point representation, we reduce 22X parameter size (237.9MB→10.8MB) and 16X FM size (15.7MB→0.98MB), respectively. Results show that the inference accuracy is more sensitive to the precision of FM.

On the other hand, DNN models with similar accuracy may result in different hardware efficiency. To provide a quantitative analysis, we implement DNNs with the same architecture but different configurations in a FPGA and examine their impacts on hardware. Figure 2 (b) shows the BRAM (on-chip memory in FPGA) usages with different input resize factors and configurations of FM quantization. By reducing the resize factor from 1.00 to 0.78, we can maintain nearly the same DNN accuracy (<1.0% drop), but save half memory when the factor is smaller than 0.9. Similarly, Figure 2 (c) indicates small changes may lead to diverse DSP utilization. By taking the 6-bit FM (FM16) as an example, the required DSPs reduce from 128 to 64 when weights are changed from 15-bit (W15) to 14-bit (W14).

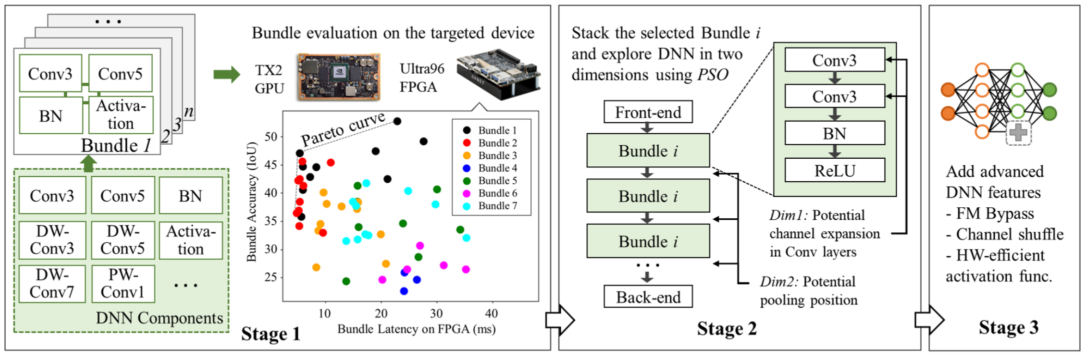
Table 2. Accuracy comparison on DAC-SDC dataset with ResNet (He et al., 2016), VGG (Simonyan & Zisserman, 2014), and SkyNet backbones and the same back-end for object detection.

Backbone DNN	# of Parameter	IoU
ResNet-18	11.18M	0.61
ResNet-34	21.28M	0.26
ResNet-50	23.51M	0.32
VGG-16	14.71M	0.25
SkyNet (ours)	**0.44M**	**0.73**

For the second challenge, it is difficult to select a reference DNN with relatively high accuracy upper bound on a given task. The DNNs with impressive accuracy on published datasets (e.g., CIFAR-10/100 and ImageNet) may not be always suitable. We evaluate the accuracy of popular reference DNNs on DAC-SDC object detection dataset and list the results in Table 2. With the fixed back-end bounding box regression part, these reference DNNs show no clear clues regarding their parameter size and inference accuracy after adequate training. Thus, it is not easy to select a promising reference model for a given task.

4 A Bottom-Up Design Approach

Motivated by the discussed challenges in Section 3, we propose a bottom-up approach to leverage the hardware-efficient DNN design for embedded systems. It is a three-stage approach as shown in Figure 3.



Conv3
Conv5
BN
Activa-tion
Bundle 1
DNN Components
DW-Conv3
DW-Conv5
DW-Conv7
PW-Conv1
Bundle evaluation on the targeted device
TX2 GPU
Ultra96 FPGA
Pareto curve
Bundle Accuracy (IoU)
Bundle Latency on FPGA (ms)
Stage 1
Stack the selected Bundle i and explore DNN in two dimensions using PSO
Front-end
Bundle i
Back-end
ReLU
Dim1: Potential channel expansion in Conv layers
Dim2: Potential pooling position
Stage 2
Add advanced DNN features
- FM Bypass
- Channel shuffle
- HW-efficient activation func.
Stage 3

Figure 3. The proposed bottom-up DNN design flow to deliver hardware-efficient DNNs for embedded systems in three stages.

4.1 Stage 1: Bundle Selection and Evaluation

This flow starts with building the hardware-aware basic blocks, called Bundles. From a software perspective, a Bundle is a set of sequential DNN layers, which can be repeatedly stacked and construct DNNs. While from a hardware perspective, a Bundle is a set of IPs which need to be implemented on hardware. To capture the hardware constraints, Bundles need to be evaluated on targeted embedded systems for collecting realistic latency (for both FPGA and GPU) and resource utilization (for FPGA) results.

In the first stage, we enumerate the DNN components (such as Conv, pooling, activation layers, etc.) and assemble them into Bundle $1 \sim n$. Since our design for DAC-SDC needs to target both GPU and FPGA, we use the resource constraints from FPGA (more restrictive compared to GPU) to evaluate the hardware performance (latency and resource utilization) for each Bundle. To get each Bundle's potential accuracy contribution, we build a DNN sketch with fixed front- and back-end structures, and insert one type of Bundle (with replications) in the middle. In our case, the front-end is a input resizing unit while the back-end is a bounding box regression. Each DNN sketch is quickly trained for 20 epochs to get its accuracy. The most promising Bundles located in the Pareto curve are selected for the next stage.

4.2 Stage 2: Hardware-Aware DNN Search

During DNN search, the inputs include the target application (e.g., image classification or object detection), software and hardware metrics (e.g., DNN accuracy and throughput performance), and target hardware platforms. The outputs are DNN candidates which meet the software and hardware requirements running on targeted embedded platforms.

To solve such a multi-objective optimization problem, we propose a group-based particle swarm optimization (PSO) evolutionary algorithm to discover proper DNN candidates. In this algorithm, each individual DNN is regarded as a particle, and all active DNNs during the search contribute to the swarm. Since we only use one type of Bundle in one DNN, DNNs composed by the same type of Bundle are considered as a particle group. In order to maintain evolution stability, a DNN only evolves within its own group. In our group-based PSO, we label the global optimal DNN as P_{glob} and a group optimal DNN as P^{i}_{group} within the i-th group. We denote a DNN particle within group i as n^{i}_{j} and each n^{i}_{j} has two tunable dimensions: dim^{1}_{j} represents the number of channels of each Bundle replication; and dim^{2}_{j} describes the pooling position between Bundles. Both dimensions affect accuracy and hardware performance. We propose the search algorithm in Algorithm 1 with the following major components:

Population generation. An initial network population P is generated, with $\mathcal{M}$ groups and $\mathcal{N}$ networks for each group. The search is conducted for $\mathcal{I}$ iterations. Within the itr-th iteration, all networks will be trained for e_{itr} epochs, where e_{itr} increases with itr.

Latency estimation. We perform platform-specific latency estimation. For GPUs, we directly measure the inference latency on the training GPU, and scale latency to the target GPU for deployment if the target GPU is different from the training one. For FPGAs, we follow the FPGA implementation template in (Hao et al., 2019), which is an IP based mapping strategy. Given a configurable IP pool used for DNN implementation, in order to save FPGA resources, all DNN layers of the same type share the same hardware computational IP. To maximize the FPGA implementation performance, we configure the IPs to be as large as possible within the available FPGA resources. For each IP under different configurations, such as computation parallelism and buffer size, we collect its hardware resource usage and latency from high level synthesis tool. Based on individual IP performance, we adopt the DNN performance modeling from (Hao et al., 2019), to get the end-to-end latency and resource usage for a DNN.

Fitness value calculation. After each iteration of training and latency estimation, we calculate the fitness value for each network n_j as:

$$Fit_j = Acc_j + \alpha \cdot \sum_{h \in \{HW\}} \beta_h \cdot |Est_h(n_j) - Req_h| \quad (1)$$

Algorithm 1 The bottom-up DNN design with PSO

```
$P \leftarrow$ Initial_population($\mathcal{M}$, $\mathcal{N}$)
while $itr < \mathcal{I}$ do
    Fast_training($P$, $e_{itr}$)
    Performance_estimation($P$)
    for each group $i$ do
        $P^i_{group} \leftarrow$ Group_best($i$)
        Get $D_p =< dim^1_p, dim^2_p, \cdots >$ from $P^i_{group}$
        for each network $n_j$ do
            Get $D_j =< dim^1_j, dim^2_j, \cdots >$ from $n_j$
            for each dimension $dim^k_j \in D_j$ do
                $dim^k_j \leftarrow V_k(dim^k_j, dim^1_p, rand)$
                Update $D_j \leftarrow dim^k_j$
            end
            $n_j \leftarrow$ Evolve($n_j$, $D_j$)
        end
    end
    $P_{glob} \leftarrow$ Global_best()
end
```

where Acc_j is the validation accuracy of n_j, h represents each targeted hardware from all candidates HW; $Est_h(n_j)$ is the estimated latency on hardware h; and $Targ_h$ is the required hardware latency on h. Parameters β_h are used to balance the penalty across different platforms, and α is used to balance between network accuracy and hardware latency. Since FPGA latency is more strictly constrained by its resource budget, we set the FPGA platform factor larger than GPU to prioritize FPGA implementation.

Velocity calculation and particle update. In standard PSO algorithm, a velocity for each dimension of a particle is calculated based on the global best particle, and the particle position is updated by the velocity vector. DNNs in the same group are updated based on the group best G^i_{group} particle and have two tunable dimensions. To determine the number of channels (dim^1_j), we first compute the per-layer difference between the current network and the group best; then we update the number of channels of the current network by a random percentage of the difference to approach the group best. Similarly, to ensure the best pooling positions (dim^2_j), we compare the current network with the global best, and change a random number of pooling positions to approach the group best.

4.3 Stage 3: Feature Addition

We manually add more advanced DNN design features if hardware resources/constraints allow. For DAC-SDC, since most objects are very small, we add a bypass directly from low-level features to high-level features along with feature map reordering (Redmon & Farhadi, 2017) to improve small object detection. To enhance the hardware efficiency, we replace ReLU with ReLU6 (Sandler et al., 2018). More discussions are provided in the next section.

5 SkyNet

5.1 SkyNet Architecture

During the design process, the best Bundle is selected as a combination of 3×3 depth-wise Conv layer (DW-Conv3 (Howard et al., 2017)), 1×1 point-wise Conv layer (PW-Conv1), batch normalization layer (BN (Ioffe & Szegedy, 2015)), and rectified linear unit 6 (ReLU6 (Sandler et al., 2018)). By repeatedly stacking this Bundle, we generate three backbone networks in Table 3 for DAC-SDC. These networks share the same chain structure and bounding box regression function but with different configurations of feature map bypass. For model A, no bypass is included; while for the model B and C, output feature maps of Bundle #3 are fed in the Bundle #6. To handle the requirement of DAC-SDC object detection, SkyNet adapts the YOLO detector head by removing the classification output and use two anchors for bounding box regression.

5.2 Feature Map Bypass, Reordering, and ReLU6

By examining the DAC-SDC competition training data, we keep a record of the size ratio between the output bounding box and the input image and present a distribution diagram in Figure 6. It clearly shows that 91% of the objects to be detected in DAC-SDC dataset are less than 9% of the original input image size and 31% of them are even smaller than 1% of the input image size. It means the majority of objects inside this dataset can be considered as small objects and we need to propose a DNN accordingly.

We add feature map bypass and reordering (Redmon & Farhadi, 2017) to enhance the ability of detecting small object (model B and C). The bypass helps to keep small object features in the later part (closer to the output layer) of the DNN by adding low-level high-resolution feature maps. Also, it is beneficial to have multiple feature maps (from different layers) before generating the bounding boxes. Since the bypass crosses a pooling layer (highlighted in Figure 4), we use reordering (shown in Figure 5) to align the size of original feature map (generated by the Bundle #5) and the low-level feature without losing information.

The other feature to improve hardware efficiency is the ReLU6 (Sandler et al., 2018). It is an activation function which clips output range to $[0, 6]$. Since ReLU6 generates much smaller data range compared to the original ReLU ($[0, +\infty)$), less bits are required to represent intermediate FMs. It also helps to better implement lower-precision floating point in embedded GPUs and fixed-point data format in embedded FPGAs.

6 Experiment on DAC-SDC

DAC-SDC features a single object detection challenge for embedded systems, which include embedded GPUs (NVIDIA TX2) and FPGAs (Pynq-Z1 and Ultra96) with very low energy consumption. This competition considers

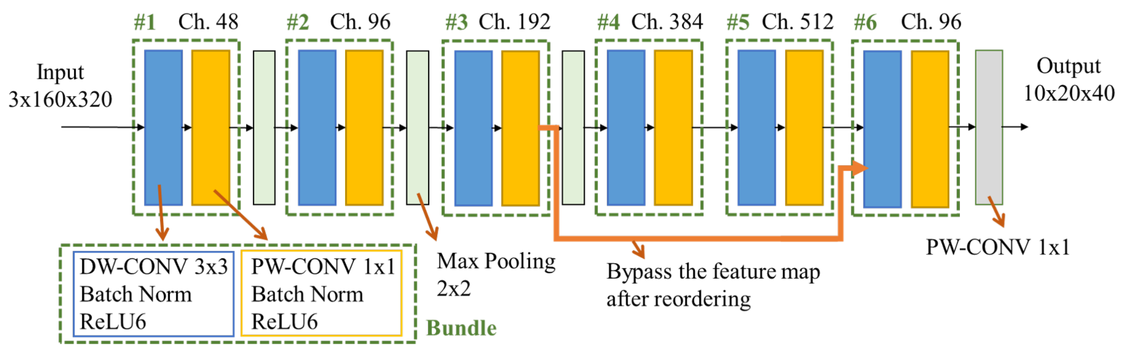


Figure 4. SkyNet architecture (model C in Table 3) generated by stacking six of the selected Bundle (circled by green dashed line) with DNN components as: DW-Conv3, PW-Conv1, BN, and ReLU6. The number of output channels is listed on top of each Bundle denoted as Ch. Three 2×2 pooling layers are inserted. The bypass is highlighted in orange, which passes feature maps generated by the Bundle #3 directly to the last Bundle. The feature map reordering is also performed along with the bypass.

Table 3. The SkyNet architecture with number of channels shown in the bracket. Each convolutional layer except the last one is followed by a BN and a ReLU (omitted for conciseness).

Configurations of SkyNet		
A	B	C
input (3×160×360 color image)		
DW-Conv3 (3) PW-Conv1 (48)		
2×2 max-pooling		
DW-Conv3 (48) PW-Conv1 (96)		
2×2 max-pooling		
DW-Conv3 (96) PW-Conv1 (192) [*Bypass Start*] ***FM Reordering*** (768)		
2×2 max-pooling		
DW-Conv3 (192) PW-Conv1 (384)		
DW-Conv3 (384) PW-Conv1 (512)		
PW-Conv1 (10)	[*Bypass End*] ***FM Concatenated*** DW-Conv3 (512+768) PW-Conv1 (48) PW-Conv1 (10)	[*Bypass End*] ***FM Concatenated*** DW-Conv3 (512+768) PW-Conv1 (96) PW-Conv1 (10)
Back-end for bounding box regression		

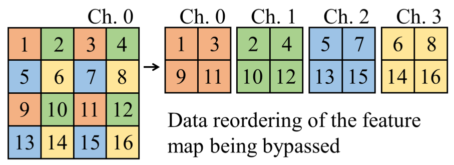


Figure 5. Feature map reordering from $1 \times 4 \times 4$ to $4 \times 2 \times 2$ with shrunken width and height but expanded number of channels. There is no information loss compared to pooling operation. In addition, this reorder pattern also ensures larger receptive field.

the most appropriate needs of UAV applications, such as capability of real-time processing, energy efficiency, and detection accuracy. To better reflect real-life challenges, images of the dataset are captured by UAVs and they are provided by the drone manufacturer called DJI. The whole dataset is divided by two parts: the training dataset with 100,000 images with objects of interest across 12 main categories and 95 sub-categories, and the hidden test set for official evaluation with 50,000 images that only the contest organizers could access (DJI, 2018). Results generated by SkyNet are shown in Figure 7, where 91% of targeted objects are smaller than 9% of the input image size. In DAC-SDC'19, 52 GPU teams and 58 FPGA teams participated worldwide creating a very intense competition. Our SkyNet design has successfully delivered the best inference accuracy and total score for both GPU and FPGA tracks.

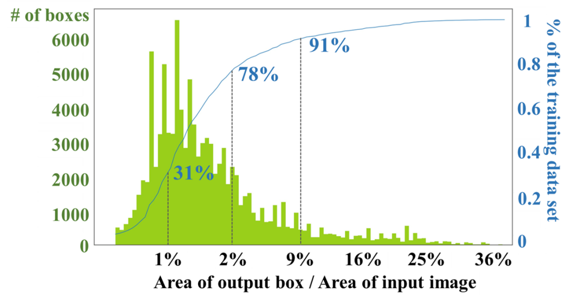


Figure 6. The distribution of bounding box relative size in DAC-SDC training dataset. We capture the bounding box relative size by computing the ratio of output bounding box size divided by the input image size. The green bars show the ratio distribution, and the blue curve shows the corresponding cumulative distribution.

6.1 Ablation Study

We perform an ablation study on DAC-SDC dataset to analyze these three configurations of SkyNet (Model A, B, and C listed in Table 3). By combining two activation functions (ReLU and ReLU6), six configurations of SkyNet are evaluated. We train these models in an end-to-end fashion using multi-scale training with the learning rate starting from 1e-4 to 1e-7. We apply stochastic gradient descent (SGD) to update parameters. To further enrich the training data, we use data augmentations to distort, jitter, crop, and resize inputs

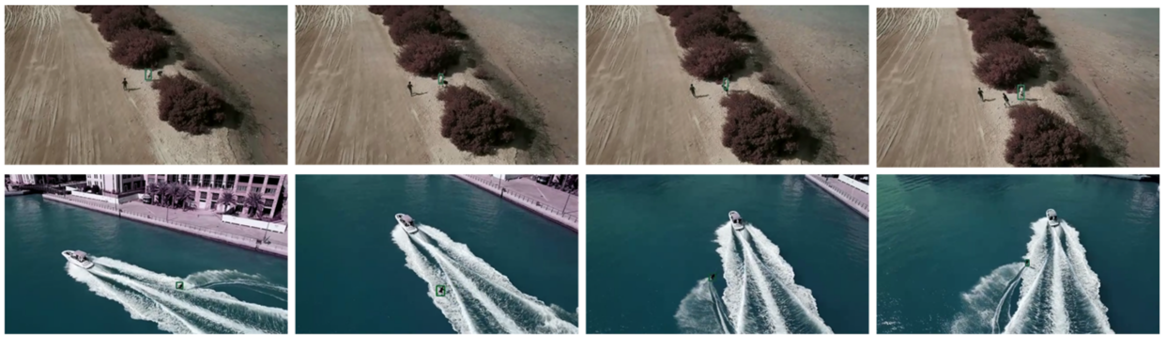


Figure 7. Object detection results generated by SkyNet on DAC-SDC dataset. Challenges include to detect small objects and distinguish multiple similar objects (e.g., images in the first row).

Table 4. Validation accuracy of SkyNet.

DNN Model	Parameter Size	IoU
SkyNet A - ReLU	1.27 MB	0.653
SkyNet A - ReLU6	1.27 MB	0.673
SkyNet B - ReLU	1.57 MB	0.685
SkyNet B - ReLU6	1.57 MB	0.703
SkyNet C - ReLU	1.82 MB	0.713
SkyNet C - ReLU6	1.82 MB	**0.741**

with size 160×320. The accuracy results are presented in Table 4, where SkyNet C - ReLU6 reaches the highest IoU (0.741) on the validation set. Therefore, we use this model as the proposed design for the following experiments.

6.2 DAC-SDC Evaluation Criteria

Comprehensive evaluations are introduced in DAC-SDC covering detection accuracy (IoU), throughput (FPS), and energy consumption (Xu et al., 2019). To identify the best design, a total score is calculated and shown below.

Assuming there are I registered teams and K images in the test set, the IoU score for team i, denoted as R_{IoU_i}, is calculated as:

$$R_{IoU_i} = \frac{\sum_{k=1}^{K} IoU_{i,k}}{K} \tag{2}$$

For energy, $\bar{E}_I$ is denoted as the average energy consumption of all I entries when performing DNN inference on the test dataset (Equation 3). The energy score of team i (ES_i) is then computed using Equation 4 relating to the ratio between average energy and the energy consumed by this team. x in this equation is set to 2 and 10 for FPGA track and GPU track, respectively.

$$\bar{E}_I = \frac{\sum_{i=1}^{I} E_i}{I} \tag{3}$$

$$ES_i = max\{0, 1 + 0.2 \times log_x \frac{\bar{E}_I}{E_i}\} \tag{4}$$

Eventually, the total score, denoted as TS_i, is calculated in Equation 5 including both inference accuracy (R_{IoU_i}) and energy consumption (ES_i).

$$TS_i = R_{IoU_i} \times (1 + ES_i) \tag{5}$$

6.3 GPU Implementation

For the TX2 GPU implementation, we keep all network parameters using 32-bit float data format to maintain the best inference accuracy. Since most of the compute-intensive part of DNN inference are handled by NVIDIA cuDNN, which leaves little space for handcrafted improvement, we start optimizing our design on a system-level.

The whole procedure of running SkyNet contains four steps as: 1) input fetching from the flash storage in a unit of batch; 2) image pre-processing which includes input resizing and normalization; 3) DNN inference; and 4) post-processing to generate bounding boxes and buffer results in DDR memory. The most straightforward way is to execute these four steps in serial with the cost of low resource utilization and poor throughput performance. In our design, we first merge step 1 and 2 in pre-process and enable multithreading technology to execute these steps in a pipelined fashion as shown in Figure 10. We use NVIDIA System Profiler (L4T) to capture the latency results. In average, the proposed system-level optimizations enable a 3.35X speedup compared to the original series design and help our design reach the highest throughput performance, peaking at 67.33 FPS.

6.4 FPGA Implementation

To implement SkyNet on FPGA, we suffer even scarcer resource budgets, as the peak performance provided by Ultra96 FPGA (144 GOPS @200MHz) is much lower than the TX2 GPU (665 GFLOPS @1300MHz). By using the proposed bottom-up design flow, hardware limitations have already captured by the Bundle design and the Bundle is instantiated on FPGA as a single customized hardware IP. Since SkyNet is structured by the same type of Bundle, this IP can be shared across different SkyNet layers to cope with

Table 5. GPU final results from DAC-SDC'19 and '18 using the hidden test set with 50K images, evaluated by a TX2 GPU.

TEAM NAME	IOU	FPS	POWER(W)	TOTAL SCORE
RESULTS FROM 2019				
SKYNET (OURS)	**0.731**	**67.33**	13.50	**1.504**
THINKER (XIONG ET AL., 2019A)	0.713	28.79	**8.55**	1.442
DEEPZS (DENG ET AL., 2019)	0.723	26.37	15.12	1.422
RESULTS FROM 2018				
ICT-CAS (LU ET AL., 2018)	0.698	24.55	12.58	1.373
DEEPZ (DENG & ZHUO, 2018)	0.691	25.30	13.27	1.359
SDU-LEGEND (ZANG ET AL., 2018)	0.685	23.64	10.31	1.358

Table 6. FPGA final results in DAC-SDC'19 and '18 using the hidden test set with 50K images. Designs in 2019 are evaluated on a Ultra96 FPGA while designs in 2018 use a Pynq-Z1 FPGA.

TEAM NAME	IOU	FPS	POWER (W)	TOTAL SCORE
RESULTS IN 2019				
SKYNET (OURS)	**0.716**	25.05	7.26	**1.526**
XJTU_TRIPLER (ZHAO ET AL., 2019)	0.615	50.91	9.25	1.394
SYSTEMSETHZ (KARA & ALONSO, 2019)	0.553	**55.13**	6.69	1.318
RESULTS IN 2018				
TGIIF (ZENG ET AL., 2018)	0.624	11.96	4.20	1.267
SYSTEMSETHZ (KARA ET AL., 2018)	0.492	25.97	**2.45**	1.179
ISMART2 (HAO ET AL., 2018)	0.573	7.35	2.59	1.164

Table 7. Validation accuracy results regarding different quantization schemes during FPGA implementation

Scheme	Feature Map	Weight	Accuracy (IoU)
0	Float32	Float32	0.741
1	9 bits	11 bits	0.727
2	9 bits	10 bits	0.714
3	8 bits	11 bits	0.690
4	8 bits	10 bits	0.680

the resource constraints. Still, we need more optimization strategies to further enhance the performance.

6.4.1 *Quantization, Batch, and Tiling*

Since fixed-point representation is more favorable in FPGA design, we quantize the FMs and weights from floating point to fixed point and explore different quantization schemes in Table 7. After quantization, the same SkyNet model suffers different levels of accuracy drop from 1.4% to 6.1% in scheme 1 to 4. Since accuracy has higher weight in the total score calculation (Equation 5), we pick scheme 1 as the quantization design for SkyNet.

To exploit data reuse opportunities, input batching is a common technique by increasing the amount of input workload. With larger batch size, the process of network inference asks for larger amount of FPGA on-chip memory (BRAM) to buffer intermediate FMs. Since our implementation is based on an IP-shared structure, buffers instantiated on FPGA are shared by different layers, which means the buffer may not be large enough for the FMs generated by the first few layers while too large for the last few layers as FMs get smaller after pooling. To overcome this problem, we propose a input tiling and batch scheme as shown in Figure 9. Four inputs are stitched to form a larger input which can be processed as an entirety. With the tiling and batch process, it is possible to use one shared buffer across different layers without changing its size. The proposed solution inherits the benefit from batch process to allow better reuse of DNN weights and it eliminates the possible waste of unused buffer space.

6.4.2 *Task partitioning*

To fully utilize the available computational resource, we also implement task partitioning on the Ultra96. The whole design is shown in Figure 10 which is highly similar to our GPU design. Workloads are distributed to both CPU and FPGA and creating a system-level pipeline. With all three tasks (pre-process, SkyNet inference, and post-process) overlapped, our FPGA design can reach 25.05 FPS.

6.5 Result Comparison

After implementing SkyNet on GPU and FPGA following the strategies mentioned in Section 6.3 and 6.4, our designs are evaluated by the DAC-SDC organizers using the hidden test set. As shown in Table 5 and 6, we present the comparison results with the top-3 teams in DAC-SDC'19 and

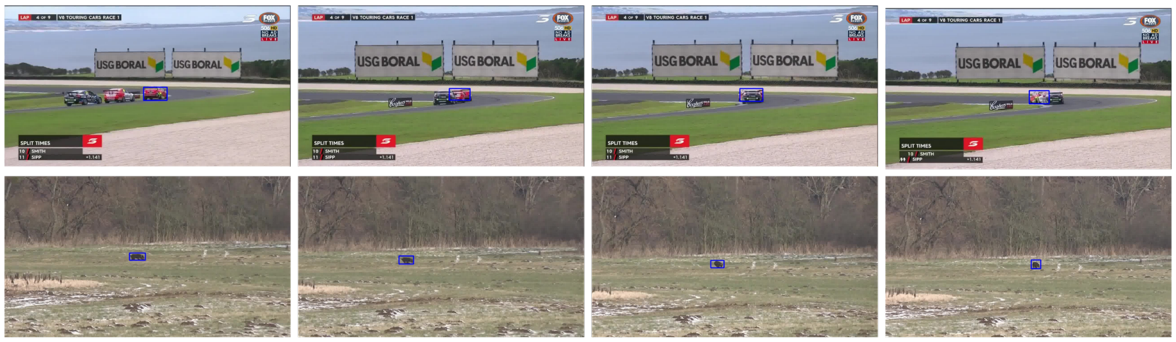


Figure 8. Object tracking results generated by SkyNet on GOT-10K dataset.

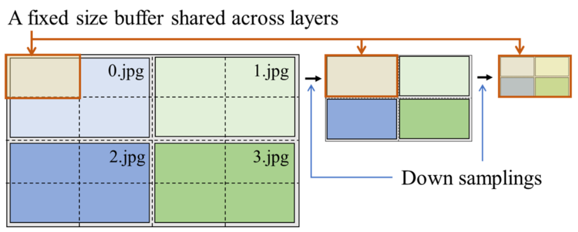


Figure 9. The proposed batch and tiling design to increase the data reuse opportunity and avoid on-chip memory waste.

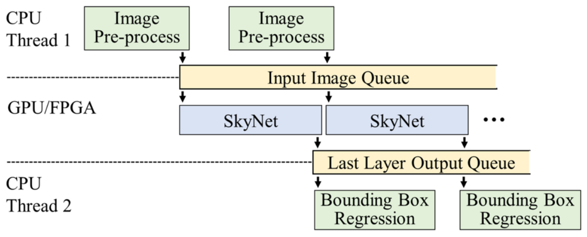


Figure 10. Task partitioning in SkyNet implementation on TX2 GPU and Ultra96 FPGA.

'18. In our GPU design, SkyNet outperforms all other competitors by delivering the best accuracy (0.731), throughput performance (67.33), and total score (1.504). In terms of the FPGA design, SkyNet also reaches the best accuracy and gets the highest total score.

7 SkyNet Extension on GOT-10K

Since SkyNet can deliver real-time object detection on embedded systems, we setup experiments on the GOT-10k benchmark (Huang et al., 2018) to demonstrate its potential on object tracking. GOT-10k is a large high-diversity database for generic object tracking with rich motion trajectory and wide coverage of object classes. Models are evaluated with two metrics in GOT-10k benchmark, average overlap (AO) and success rate (SR). Average overlap is defined as the mean of intersection over union (IoU) between prediction and ground truth bounding boxes, while success rate is defined as the proportion of predictions where the IoU is beyond some threshold. With its open responsive evaluation server, we are able to assess SkyNet with other conventional backbones.

Table 8. Performance of SiamRPN++ networks on GOT-10k with different backbones evaluated on single NVIDIA 1080Ti.

Backbone	AO	$SR_{0.50}$	$SR_{0.75}$	FPS
AlexNet	0.354	0.385	0.101	52.36
ResNet-50	0.365	0.411	0.115	25.90
SkyNet	0.364	0.391	0.116	41.22

Table 9. Performance of SiamMask networks on GOT-10k with different backbones evaluated on single NVIDIA 1080Ti.

Backbone	AO	$SR_{0.50}$	$SR_{0.75}$	FPS
ResNet-50	0.380	0.439	0.153	17.44
SkyNet	0.390	0.442	0.158	30.15

7.1 Evaluation Using SiamRPN++

Siamese trackers are conventional object trackers that locate the object by the correlation between features extracted from the exemplar image and search image, where the quality of feature extractors play an important role. SiamRPN++ (Li et al., 2019a) is the first Siamese tracker that has been proven to profit from backbones with different capacities as long as they are properly trained. To evaluate the performance of different backbones, we trained the networks on GOT-10k with learning rates from 1e-3 to 1e-5, the exemplar and search image to have size 128/127 and 256/255 for SkyNet and other backbones respectively. Results are shown in Table 8.

7.2 Evaluation Using SiamMask

SiamMask (Wang et al., 2019) is another Siamese tracker that outperforms SiamRPN++ under the same configuration. However, segmentation information is required during its training period, so it cannot be directly trained with GOT-10k dataset. Instead, we perform training with Youtube-VOS dataset (Xu et al., 2018) to compare the performance of different backbones under this structure. The networks are trained with learning rates from 1e-3 to 1e-4, examplar size 128/127 and search size 256/255 for SkyNet and ResNet-50 respectively. The results are shown in Table 9.

8 Conclusions

In this paper, we proposed SkyNet, a hardware-efficient method to generate compact DNNs for object detection running on embedded GPUs and embedded FPGAs. SkyNet contains a novel bottom-up DNN design flow which can capture hardware limitations using realistic hardware feedbacks and deliver DNNs with great balance between software and hardware metrics such as DNN inference accuracy and throughput performance. SkyNet was demonstrated on the 56th IEEE/ACM DAC-SDC low power object detection challenge and won the first place winner award for both GPU and FPGA tracks. We also extended SkyNet to handle object tracking task and it delivered 1.60X and 1.73X higher FPS, and 37.20X smaller parameter size with comparable accuracy when compared to Siamese trackers with ResNet-50 backbone.

9 Acknowledgments

This work was partly supported by the IBM-Illinois Center for Cognitive Computing System Research (C^3SR) – a research collaboration as part of IBM AI Horizons Network.

References

Alwani, M., Chen, H., Ferdman, M., and Milder, P. Fused-layer cnn accelerators. In *Proceedings of the International Symposium on Microarchitecture*, 2016.

Cai, H., Zhu, L., and Han, S. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018.

Cai, Z. and Vasconcelos, N. Cascade r-cnn: High quality object detection and instance segmentation, 2019.

Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne. Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks. In *IEEE International Solid-State Circuits Conference (ISSCC)*, 2016.

Cheng, B., Wei, Y., Shi, H., Feris, R., Xiong, J., and Huang, T. Decoupled classification refinement: Hard false positive suppression for object detection, 2018a.

Cheng, B., Wei, Y., Shi, H., Feris, R., Xiong, J., and Huang, T. Revisiting RCNN: On awakening the classification power of faster RCNN. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018b.

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Dai, J., Li, Y., He, K., and Sun, J. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, 2016.

Dai, X., Yin, H., and Jha, N. Nest: A neural network synthesis tool based on a grow-and-prune paradigm. *IEEE Transactions on Computers*, 2019a.

Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., et al. Chamnet: Towards efficient network design through platform-aware model adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019b.

Deng, J. and Zhuo, C. a. DAC-SDC'18 2nd place winner in GPU track. https://github.com/jndeng/DACSDC-DeepZ, 2018. Accessed: 2019-09-01.

Deng, J., Shen, T., Yan, X., Chen, Y., Zhang, H., Wang, R., Zhou, P., and Zhuo, C. DAC-SDC'19 3rd place winner in GPU track, 2019.

Ding, X., Ding, G., Han, J., and Tang, S. Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

DJI. The DAC-SDC dataset for low power object detection. http://www.cse.cuhk.edu.hk/b̃yu/2019-DAC-SDC/index.html, 2018. Accessed: 2019-09-04.

Dong, J.-D., Cheng, A.-C., Juan, D.-C., Wei, W., and Sun, M. Ppp-net: Platform-aware progressive search for pareto-optimal neural architectures. 2018.

Franklin, D. NVIDIA Jetson TX2 delivers twice the intelligence to the edge. *NVIDIA Accelerated Computing—Parallel For all*, 2017.

Gope, D., Dasika, G., and Mattina, M. Ternary hybrid neural-tree networks for highly constrained iot applications. 2019.

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, 2015.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. In *Proceedings of International Conference on Learning Representations (ICLR)*, 2016.

Hao, C., Li, Y., Huang, S. H., Zhang, X., Gao, T., Xiong, J., Rupnow, K., Yu, H., Hwu, W.-M., and Chen, D. DAC-SDC'18 3rd place winner in FPGA track. https://github.com/onioncc/iSmartDNN, 2018. Accessed: 2019-09-01.

Hao, C., Zhang, X., Li, Y., Huang, S., Xiong, J., Rupnow, K., Hwu, W.-m., and Chen, D. Fpga/dnn co-design: An efficient design methodology for iot intelligence on the edge.

In *Proceedings of the 56th Annual Design Automation Conference 2019*, pp. 206. ACM, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision (CVPR)*, 2017.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *Proceedings of the International Conference on Computer Vision*, 2019.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Huang, L., Zhao, X., and Huang, K. Got-10k: A large high-diversity benchmark for generic object tracking in the wild. *arXiv preprint arXiv:1810.11981*, 2018.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of International Symposium on Computer Architecture (ISCA)*, 2017.

Kara, K. and Alonso, G. DAC-SDC'19 3rd place winner in FPGA track, 2019.

Kara, K., Zhang, C., and Alonso, G. DAC-SDC'18 2nd place winner in FPGA track. https://github.com/fpgasystems/spooNN, 2018. Accessed: 2019-09-01.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 2012.

Law, H. and Deng, J. Cornernet: Detecting objects as paired keypoints. *Lecture Notes in Computer Science*, pp. 765781, 2018. ISSN 1611-3349. doi: 10.1007/978-3-030-01264-9_45. URL http://dx.doi.org/10.1007/978-3-030-01264-9_45.

Li, B., Wu, W., Wang, Q., Zhang, F., Xing, J., and Yan, J. Siamrpn++: Evolution of siamese visual tracking with very deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2019a.

Li, F., Zhang, B., and Liu, B. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

Li, Y., Chen, Y., Wang, N., and Zhang, Z. Scale-aware trident networks for object detection, 2019b.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017a.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2017b.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. SSD: Single shot multibox detector. In *Proceedings of the European conference on computer vision (ECCV)*, 2016.

Lu, H., Cai, X., Zhao, X., and Wang, Y. DAC-SDC'18 1st place winner in GPU track. https://github.com/lvhao7896/DAC2018, 2018. Accessed: 2019-09-01.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision (ICCV)*, 2017.

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S., et al. Going deeper with embedded FPGA platform for convolutional neural network. In *Proceedings of International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2016.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Proceedings of European Conference on Computer Vision*, 2016.

Redmon, J. and Farhadi, A. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2017.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2016.

Ren, A., Zhang, T., Ye, S., Li, J., Xu, W., Qian, X., Lin, X., and Wang, Y. Admm-nn: An algorithm-hardware co-design framework of dnns using alternating direction methods of multipliers. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. Overfeat: Integrated recognition, localization and detection using convolutional networks. 2014.

Shen, Z., Shi, H., Yu, J., Phan, H., Feris, R., Cao, L., Liu, D., Wang, X., Huang, T., and Savvides, M. Improving object detection from scratch via gated feature reuse. BMVC, 2019.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path nas: Device-aware efficient convnet design. *arXiv preprint arXiv:1905.04159*, 2019.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2015.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Tao, R., Gavves, E., and Smeulders, A. W. M. Siamese instance search for tracking. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.158. URL `http://dx.doi.org/10.1109/cvpr.2016.158`.

Tian, Z., Shen, C., Chen, H., and He, T. Fcos: Fully convolutional one-stage object detection, 2019.

Tschannen, M., Khanna, A., and Anandkumar, A. Strassennets: Deep learning with a multiplication budget. 2018.

Valmadre, J., Bertinetto, L., Henriques, J., Vedaldi, A., and Torr, P. H. S. End-to-end representation learning for correlation filter based tracking. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.531. URL `http://dx.doi.org/10.1109/cvpr.2017.531`.

Vanholder, H. Efficient inference with tensorrt, 2016.

Wang, J., Lou, Q., Zhang, X., Zhu, C., Lin, Y., and Chen, D. Design flow of accelerating hybrid extremely low bit-width neural network in embedded FPGA. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pp. 163–169, 2018a.

Wang, Q., Teng, Z., Xing, J., Gao, J., Hu, W., and Maybank, S. Learning attentions: residual attentional siamese network for high performance online visual tracking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4854–4863, 2018b.

Wang, Q., Zhang, L., Bertinetto, L., Hu, W., and Torr, P. H. Fast online object tracking and segmentation: A unifying approach. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

Wu, B., Dai, X., Zhang, P., Wang, Y., Sun, F., Wu, Y., Tian, Y., Vajda, P., Jia, Y., and Keutzer, K. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

Xiong, F., Yin, S., Fan, Y., and Ouyang, P. DAC-SDC'19 2nd place winner in GPU track, 2019a.

Xiong, Y., Mehta, R., and Singh, V. Resource constrained neural network architecture search. In *Proceedings of the International Conference on Computer Vision*, 2019b.

Xu, N., Yang, L., Fan, Y., Yue, D., Liang, Y., Yang, J., and Huang, T. Youtube-vos: A large-scale video object segmentation benchmark. In *European Conference on Computer Vision (ECCV)*, 2018.

Xu, X., Zhang, X., Yu, B., Hu, X. S., Rowen, C., Hu, J., and Shi, Y. Dac-sdc low power object detection challenge for uav applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

Zang, C., Liu, J., Hao, Y., Li, S., Yu, M., Zhao, Y., Li, M., Xue, P., Qin, X., Ju, L., Li, X., Zhao, M., and Dai, H. DAC-SDC'18 3rd place winner in GPU track. https://github.com/xiaoyuuuuu/dac-hdc-2018-object-detection-in-Jetson-TX2, 2018. Accessed: 2019-09-01.

Zeng, S., Chen, W., Huang, T., Lin, Y., Meng, W., Zhu, Z., and Wang, Y. DAC-SDC'18 1st place winner in FPGA track. https://github.com/hirayaku/DAC2018-TGIIF, 2018. Accessed: 2019-09-01.

Zhang, X., Liu, X., Ramachandran, A., Zhuge, C., Tang, S., Ouyang, P., Cheng, Z., Rupnow, K., and Chen, D. High-performance video content recognition with long-term recurrent convolutional network for FPGA. In *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4, 2017a.

Zhang, X., Wang, J., Zhu, C., Lin, Y., Xiong, J., Hwu, W.-m., and Chen, D. DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In *Proceedings of International Conference on Computer-Aided Design (ICCAD)*, 2018a.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018b.

Zhang, X., Hao, C., Lu, H., Li, J., Li, Y., Fan, Y., Rupnow, K., Xiong, J., Huang, T. S., Shi, H., Hwu, W.-m., and Chen, D. Skynet: A champion model for dac-sdc on low power object detection. *arXiv preprint arXiv:1906.10327*, 2019.

Zhang, Y., Suda, N., Lai, L., and Chandra, V. Hello edge: Keyword spotting on microcontrollers. *arXiv preprint arXiv:1711.07128*, 2017b.

Zhao, B., Zhao, W., Xia, T., Chen, F., Fan, L., Zong, P., Wei, Y., Tu, Z., Zhao, Z., Dong, Z., and Ren, P. DAC-SDC'19 2nd place winner in FPGA track, 2019.

Zhou, X., Wang, D., and Krhenbhl, P. Objects as points, 2019.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.