This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2957724, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Energy Efficient Real-Time UAV Object Detection on Embedded Platforms

Jianing Deng, Zhiguo Shi, *Senior Member, IEEE*, and Cheng Zhuo, *Senior Member, IEEE*

***Abstract*—The recent technology advancement on unmanned aerial vehicle (UAV) has enabled diverse applications in vision related outdoor tasks. Visual object detection is a crucial task among them. However, it is difficult to actually deploy detectors on embedded devices due to the challenges among energy consumption, accuracy, and speed. In this paper, we address a few key challenges from platform, application to system, and propose an energy efficient system for real-time UAV object detection on an embedded platform. The proposed system can achieve speed of 28.5 FPS and 2.7 FPS/W energy efficiency on the data set from 2018 Low Power Object Detection Challenges (LPODC).**

I. Introduction

Visual object detection is a crucial task in computer vision to support various applications, *e.g.*, surveillance camera, UAV, self-driving vehicle. Due to the recent breakthroughs in deep neural network (DNN), especially convolutional neural network (CNN), the performance of CNN-based detectors has been steadily improved, which not only defeats human visual systems in occluded and tiny object detection, but also exceeds the conventional algorithms in detection accuracy.

For example, R-CNN [1] first combined a region proposal method with CNN features for detection and outperformed the conventional detectors by 29% accuracy improvement. The follow-up work in [2] introduced the region of interest (RoI) pooling strategy to extract regional features from the CNN backbone, thereby forming a two-stage detection framework. Moreover, one-stage detectors have also been proposed, such as YOLO [3] and SSD [4]. They can further reduce the redundant computations for speed-up by densely predicting possible locations to improve the RoI pooling strategy.

On the other hand, the recent technology advancement on UAV has enabled itself as a full-fledged workhouse, which integrates various computing devices, peripherals and sensors. With its diverse applications in vision related outdoor tasks, it is apparently an appealing idea to implement low power object detection on an embedded device for UAV applications. However, to actually deploy such detectors on an embedded device, algorithm performance is not the only concern that needs to be addressed. Instead the system needs to evaluate the underlying platform and environment constraints, and then utilize the available computation resources to achieve the trade-off among energy consumption, accuracy and speed.

Apparently such trade-off is not a trivial task. We need to tackle a few key challenges from platform, application to system in order to realize UAV object detection on an embedded platform:

J. Deng, Z. Shi and C. Zhuo (Email: czhuo@zju.edu.cn) are with the College of Information Science and Electronic Engineering, Zhejiang University.

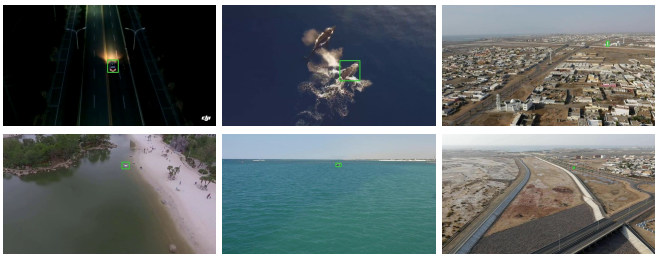

Fig. 1. Selected images and objects from the LPODC data set with 95 sub-categories. Top row: examples with distraction (context or similar objects) to the target objects. Bottom row: examples with tiny objects.

- **Platform level resource utilization:** Unlike many prior algorithms designed for desktop or server, we need to address the aforementioned compatibility and resource constraints on an embedded platform.
- **Application level data uniqueness:** The view angle of UAV camera is distinctly different from a traditional ground perspective. Due to the shooting distance between the camera and the target, the task is further complicated by tiny, distracted and diverse objects (Fig. 1).
- **System level energy efficiency and real-time constraints:** While it is highly demanded to have real-time capability for UAV, energy efficiency is the Achilles heel we cannot ignore for embedded applications. A trade-off between the two is crucial for the successful deployment.

In this paper, we address the aforementioned challenges and present an energy efficient system for real-time UAV object detection on an embedded platform (Nvidia Jetson TX2 Board). The contributions are summarized as below:

- We analyze the data uniqueness for UAV-captured image data set and then propose a basic framework for single object detection. Based on the framework, we design an energy-efficient UAV-based object detector architecture with real-time detection capability.
- We manage to enhance the localization capability of the detector with various optimization policies. The optimization can overcome the aforementioned resource utilization problem, and maintain the real-time processing speed.
- We perform design space exploration to enable trade-off among energy efficiency, accuracy and speed. We then implement an optimized UAV object detection system on Jetson TX2, which can outperform the winner in 2018 LPODC for Design Automation Conference System Design Contest (DACSDC) in various aspects.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2957724, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

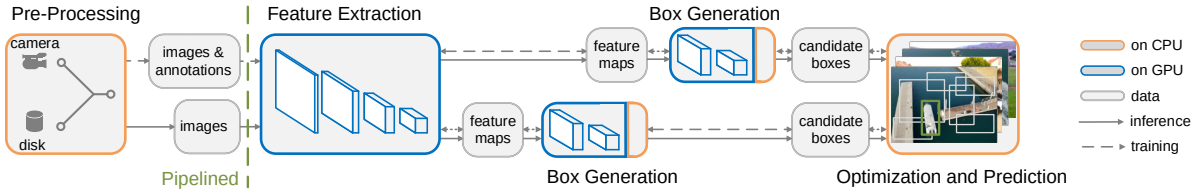

Fig. 2. Proposed framework and system for UAV single object detection.

II. UAV Object Detection Data Set

As low power optimization is the most concerning topic for embedded devices, the LPODC at DACSDC'18 places its focus on UAV single object detection with not only stringent accuracy requirement, but also constraints for real-time and power consumption [5], [6]. The objective of LPODC is to accurately localize the objects of interest in UAV-view images with axis-aligned bounding boxes, given that each image contains only one object of interest. The data set that LPODC provides includes a total of 150k UAV-view images from 12 different categories and 95 sub-categories. Around 70% of data set are publicly released as training set, while the rest are reserved for testing. The images are diverse in all the aspects, which are captured by cameras of different specifications, at different time (morning, noon, evening), under different contexts (road, sea, city, urban, *etc.*), with different view angles and shooting distances. As shown in Fig. 1, most objects only occupy 1-2% of the entire image and are often occluded or distracted. Unlike general computer vision challenges such as ImageNet [7] or COCO [8] solely focusing on accuracy, LPODC comprehensively considers accuracy, speed and energy for the system implementation, which hence requires software and hardware co-design. The evaluation balances among accuracy, energy and speed. The accuracy is measured by the mean Intersection over Union (mIoU) in Eq. 1:

$$mIoU = \frac{\sum_{i=1,2,...,N} \frac{B_i \cap B_i^*}{B_i \cup B_i^*}}{N} \tag{1}$$

where B_i and B_i^* are the areas of the i-th predicted and ground-truth bounding boxes in the total of N bounding boxes. Average processing time and energy per image are also accounted in the final score when evaluating the system [5].

III. Framework for UAV Object Detection

A. Overview of the Proposed Framework

Fig. 2 provides an overview of the proposed 4-stage framework for UAV object detection on the embedded platform:

- **Pre-Processing:** Data (images & annotations) are loaded, augmented and transformed to the proper format.
- **Feature Extraction:** Extract high-level semantic features from the transformed images through the selected CNN.
- **Box Generation:** Predict candidate bounding boxes (with objectness scores) using the extracted features.
- **Optimization and Prediction:** During training, this stage calculates the loss function and *optimizes* network parameters. During inference, this stage filters out the candidate bounding boxes and conducts object location *prediction*.

B. Pre-Processing

To achieve real-time detection, we use single-scale image processing instead of multi-scale (*e.g.*, image pyramid [9]) in the proposed framework, sacrificing detection accuracy while greatly improving the processing speed. As a result, the selection of image scale may easily impact the detector performance. Given an $H \times W$ image, the total floating-point multiplication and addition operations (FLOPs) for a one-stage detector can be calculated by:

$$FLOPs \approx \sum_l 2 \cdot \frac{K_l^2 \cdot H \cdot W \cdot C_{l-1} \cdot C_l}{S_l^2} \propto H \cdot W \tag{2}$$

where K_l and C_l are the size and the number of the kernels; S_l is the overall strides in the l_{th} convolutional layer. Apparently a smaller image scale helps improve the speed at the cost of detection accuracy [10]. Since the images in LPODC have 640×360 resolution, scaling to a square image, as in the prior one-stage detectors [3], [4], may actually distort the objects and lose the details to be learned. Thus, in our work we balance between speed and accuracy with a 640×384 image scale [1].

During training, we use the standard data augmentation as in [10] to pre-process all the raw images, and then reshape them to 640×384 with bilinear interpolation. For inference, we just transform the input images to 640×384 scale.

C. Feature Extraction

At feature extraction stage, an extraction network extracts the image features in a hierarchical manner, the quality of which is critical to the detection accuracy. Given the significant computing resources consumed by the extraction network, it is crucial to design a powerful but light-weight network. We investigated two backbone network candidates for feature extraction: 1) Darknet7, a tiny CNN with only 7 convolutional layers [10], and 2) Resnet18, an 18-layer residual CNN [11]. While Darknet7 is featured with much simpler structure, Resnet18 has deeper network and hence is capable to provide more powerful representations of the image at the cost of higher computation load. As the original Resnet18 is still too slow to meet the real-time requirements, we further halve the number of channels in all of its convolutional layers. The modified Resnet18 is then deployed as the backbone network in our framework for its lighter computation load as well as its higher accuracy than Darknet7. While there are many works customizing network architectures for mobile applications

[1] 384 is the closest number to 360 that can be divided by 32.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2957724, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

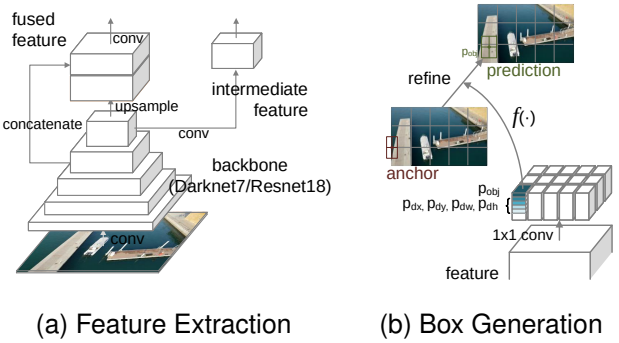

Fig. 3. Flows for (a) feature extraction and (b) box generation in the proposed detection framework.

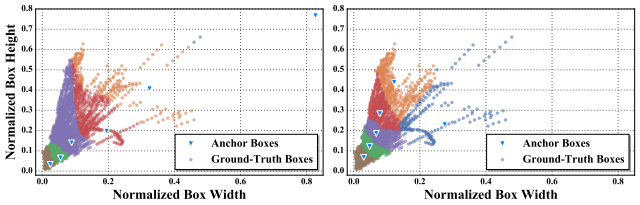

Fig. 4. Distributions of the normalized shape of the anchors and ground-truth boxes in training set under different anchor selection approaches. Left: approach (1) with 0.52 mIoU and 0.885 accuracy. Right: approach (2) with 0.64 mIoU and 0.883 accuracy.

(*e.g.*, MobileNet [12], ShuffleNet [13], Fire SSD [14]), in this work, we only investigated two representative candidates of DarkNet and ResNet, which have simpler architectures and hence can be easily implemented on various frameworks.

Moreover, since the overall stride of the network is 32, the feature map size of the last layer is then 32× smaller than the input image. Such coarse feature maps are incapable to accurately match the location of a specific object, especially for the tiny and occluded objects as in Fig. 1. Thus, as shown in Fig. 3a, in our extraction network, we propose to upsample the high-level feature maps and combine that with the low-level ones through channel concatenation. Unlike element-wise addition in [15], channel concatenation provides a learnable way to combine feature maps at different level at the cost of ∼1.0 GFLOPs computation load increase. The feature fusion then enriches semantic information and detailed location information to localize the object of interest.

D. Box Generation

At the box generation stage, the extracted feature maps from the last stage are used to densely generate candidate bounding boxes over locations. Instead of computing candidate boxes from scratch, we only predict 4 offsets (center coordinates, width and height) and combine them with the pre-calculated anchor boxes to create candidate bounding boxes. Then, unlike [3] that learns complex regression functions, our algorithm only learns a function that can select a proper anchor among all anchors, and then fine-tunes the selected anchor with the predicted offsets to achieve a better bounding box. Fig. 3b describes the five offsets p_{obj}, p_{dx}, p_{dy}, p_{dw}, p_{dh} that are predicted with a 1×1 convolution, where p_{obj} represents the objectness of the anchor, p_{dx} and p_{dy} adjust the central location of the anchor, p_{dw} and p_{dh} adjust the shape of the anchor.

There are two approaches to decide the original anchors: (1) Adopt the anchors designed for other data sets (*e.g.*, COCO [8]) and adjust them accordingly for LPODC data set; (2) Select the representative cluster centroids by clustering the provided annotation boxes of objects in LPODC data set. Fig. 4 presents the distribution of the anchors and ground-truth boxes in training set for the two approaches with 6 anchors. Each ground-truth box is assigned to its nearest anchor. Intuitively, the clustering based approach may achieve better detection accuracy. However, as shown in Fig. 4, both approaches have almost the same accuracy. This simply indicates that our detector can learn to achieve decent predictions even with coarser anchors. Moreover, since the accuracy does not have dramatic change with more anchors, we adopt the first approach for simplicity and use 6 anchors in our experiments.

With the selected anchors, in order to improve the robustness for objects with different scales, we then propose to use the intermediate feature maps of the extraction network for multi-scale prediction. As shown in Fig. 2, two box generation processes with different scales of input feature maps and anchors are managed in parallel. The finer feature maps obtained after feature fusion are combined with the smaller anchors to localize small objects, while the coarser feature maps are used to generate larger candidate boxes.

E. Optimization and Prediction

This stage diverges to two branches for training and inference. During training, we use stochastic gradient descent to jointly optimize the objectness loss function L_{obj} and the localization loss function L_{loc} as below:

$$L(p_{obj}, p_{loc}) = L_{obj}(p_{obj}, p^*_{obj}) + L_{loc}(p_{loc}, p^*_{loc}, p^*_{obj}) \quad (3)$$

where p_{obj} is the predicted objectness, $p^*_{obj} = 1$ when the corresponding anchor matches the ground-truth box otherwise $p^*_{obj} = 0$, p_{loc} represents the predicted offsets of the box localization and shape, p^*_{loc} is the corresponding ground-truth.

Unlike [10] using $L2$ loss function for both L_{obj} and L_{loc} to detect multiple objects, for the challenge in LPODC, the image contains only one object. Thus, L_{obj} is dominated by a large number of negative examples (*i.e.*, bounding boxes that do not contain object of interest), which actually slows down the training convergence and reduces the localization accuracy. To tackle this problem, we propose to deploy the focal loss function [16] and combine it with a binary cross entropy function as our objectness loss function:

$$L_{focal}(p, p^*) = \begin{cases} \alpha(1-p)^\gamma \log(\sigma(p)) & p^* = 1 \\ (1-\alpha)p^\gamma \log(1-\sigma(p)) & p^* = 0 \end{cases} \quad (4)$$

where α is the weight to balance the positive and negative examples; γ is used to control the decay. We choose $\alpha = 0.75$ and $\gamma = 2$ in our experiments. The implication behind such a formulation is that while focal loss can automatically filter out easy examples and increase the weight of hard examples, the cross-entropy is more suitable as a classification loss function.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2957724, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

Then the combination of the two helps mitigate the imbalance between the single ground truth box and candidate boxes. The overall objectness loss function and localization function consider all the selected candidate boxes as below:

$$L_{obj}(p_{obj}, p^*_{obj}) = \sum_i L_{focal}(p^i_{obj}, p^{i*}_{obj}) \quad (5)$$

$$L_{loc}(p_{loc}, p^*_{loc}, p^*_{obj}) = \sum_i p^{i*}_{obj}(p^i_{loc} - p^{i*}_{loc})^2 \quad (6)$$

Then during inference, we just simply sort all the candidate bounding boxes according to their objectness scores, and pick up the one with the highest score as the final prediction.

IV. System Implementation

In this section, we present our implementation on an Nvidia Jetson TX2 board based on the proposed framework. The TX2 board includes a Pascal GPU with 256 CUDA cores and 6-core CPU and supports five alternative modes for the trade-off between speed and energy consumption.

Fig. 2 illustrates the block diagram and data flow of the complete system. First, trained weights of the detector are transplanted to TX2 board and loaded onto the flash. Then, images captured by the camera or stored on the disk are loaded onto the flash through the data loading module. The loaded images are further reshaped to 640×384 and passed into the feature extraction module. After that, two parallel box generation modules are used to generate candidate boxes according to the different scales of feature maps. The candidate box with highest predicted score is selected as the final prediction and saved back to the disk.

To maximize the efficiency of the detection system, we properly allocate the resources on the TX2 board according to the characteristics of each module. Specifically, data loading module and data saving module are both implemented on CPU due to its logic-complex nature, while the feature extraction module and box generation module are mostly implemented on GPU as CNN is computation-intensive. In order to accelerate inference to meet the requirement of real-time detection, we design a two-stage pipeline scheme for the detection system, where the data loading module is pipelined from the other modules. As shown in Fig. 2, the critical path of the resulting system is in the second stage of the pipeline, thereby reducing the overall processing time. Moreover, for the scenario where detection is conducted with all images captured (*e.g.*, DACSDC'18), batching scheme can be used to make full use of GPUs and further accelerate detection.

V. Experimental Results

In this section, we first present the overall performance of the proposed system with the local test set, and compare that with DroNet [17] (retained over LPODC dataset for fair comparison). After that, ablation experiments are conducted to justify our design choices. Finally, we evaluate the performance of the detection system on the Jetson TX2 Board.

As the official test data set is not public available, we divide the 110k publicly released training set into a new training set and a local test set, which will be used in the following experiments (80% in each sub-category are randomly selected for training with the rest for testing).

TABLE I
Overall performance of the proposed UAV single object detector with different backbones under Jetson TX2 mode2.

Backbone	Accuracy(mIoU)	Speed(FPS)	Power(W)	#Param.
DroNet [17]	0.594	45.007	4.604	0.07 M
Darknet7 [3]	0.879	21.628	10.088	11.11 M
Resnet18 [11]	0.886	28.458	10.630	5.51 M

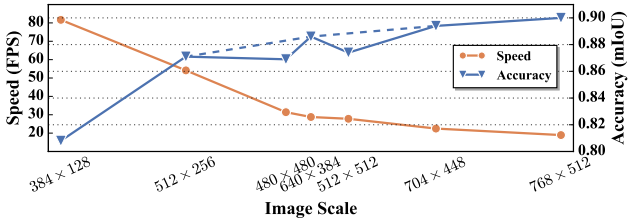

Fig. 5. Comparison of detector (Resnet18 backbone) speed and accuracy for different image scales.

A. Overall Performance

The overall performance of DroNet and the proposed UAV single object detector is presented in Table I. As seen from the table, the model using Resnet18 as the backbone can achieve 0.886 IoU, 0.07 higher than Darknet7, while improving the speed by 19% and reducing the power by 18%, demonstrating the effectiveness of residual connections. On the other hand, although DroNet achieves much higher speed with lower power, its accuracy is much lower than the proposed detector. Thus the proposed system is more applicable to the scenarios requiring higher accuracy.

B. Image Scale Selection

Fig. 5 demonstrates the trade-off between speed and accuracy under different scales of image. Note that when the scale is larger than 708×448, the detector can no longer maintain real-time processing. Moreover, we also compared the square and non-square image resolutions under the same scale. The dashed line region in Fig. 5 shows that the non-square image achieves better accuracy as we discussed in section III-B.

C. In-Depth Investigation on the Proposed Framework

Backbone: Table II shows the other optimization techniques have higher boost to the detector using Darknet7 backbone. We argue this is because the modified Resnet18 based backbone in section III-C itself is more powerful. The residual connections makes it easier to train, resulting in less optimization space, as shown in the last three columns of Table II.

Focal loss: Table II also compares the improvements due to focal loss for the overall accuracy of the two detectors. The focal loss accelerates model training in the early stages without incurring additional computation overhead, and alleviates the aforementioned imbalance problem. Thus, the detector can focus more on hard examples in the early stage of training.

Feature fusion and multi-scale prediction: Feature fusion is the most effective optimization method, which improves $\sim$1% overall accuracy for both detector in Table II. Because

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCAD.2019.2957724, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems

TABLE II
DETECTION ACCURACY AND COMPLEXITY OF THE DETECTOR USING DIFFERENT OPTIMIZATION OPTIONS. FOR VANILLA DETECTOR, WE DIRECTLY USE THE BACKBONE AS THE FEATURE EXTRACTION NETWORK WITHOUT ANY OTHER OPTIMIZATION, AND L2 LOSS FOR BOTH L_{obj} AND L_{loc}.

Design Choices	Darknet7				Resnet18			
Vanilla?	✓				✓			
Focal Loss?		✓	✓	✓		✓	✓	✓
Feature Fusion?			✓	✓			✓	✓
Multi-Scale Prediction?				✓				✓
Accuracy (mIoU)	0.856	0.864	0.877	0.879	0.873	0.875	0.883	0.886
GFLOPs	7.640	7.640	8.839	8.946	5.316	5.316	6.798	6.906

TABLE III
PERFORMANCE COMPARISON OF THE PROPOSED SYSTEM (RESNET18 BACKBONE) UNDER DIFFERENT TX2 MODES.

Mode (Name)	f_{CPU} (GHz)	f_{GPU} (GHz)	Speed (FPS)	Power (W)	Efficiency (FPS/W)
0 (Max-N)	2.0	1.30	32.937	13.963	2.359
2 (Max-P)	1.4	1.12	28.458	10.630	2.677

the fused features already include multi-scale information, the additional multi-scale prediction can only provide little accuracy boost. Moreover, as shown in Fig. 6, the combination of feature fusion and multi-scale prediction improves the detection of small objects (scale $< 5\%$).

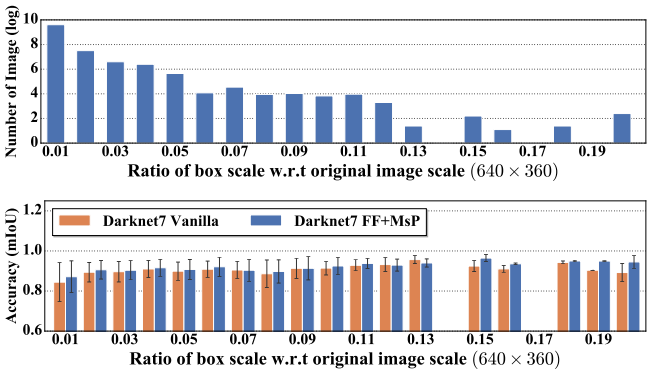

Fig. 6. Distribution of bounding box scale (top) and distribution of detection accuracy (bottom) *w.r.t* different box scales.

D. System Implementation

We first study the two modes supported by the TX2 board on the optimized system implementation. As shown in Table III, while mode 0 brings higher speed-up, mode 2 may provide higher energy efficiency and fit better with UAV applications. Table IV further compares the performance of different system implementations under mode 2. The system using the modified Resnet18 backbone outperforms the one with Darknet7 in both speed and energy efficiency. Moreover, the pipeline optimization achieves higher speed-up and energy efficiency than the plain implementation. Batching scheme achieves the best results in energy efficiency, proving its effectiveness in particular scenarios. Compared with the winner of the contest, the configuration in the last row achieves comparable IoU accuracy, 18% power saving and 16% speed improvement [5].

VI. CONCLUSIONS

In this paper, we present an energy-efficient detection framework and real time system implementation for UAV single object detection on embedded device. Various optimization techniques have been proposed and studied to achieve both energy efficiency and real time. The proposed system can achieve 28.5 FPS speed with 2.7 FPS/W energy efficiency on a Jetson TX2 platform. In the future, we will explore more network architectures [18] to improve the overall efficiency.

TABLE IV
COMPARISON OF DIFFERENT IMPLEMENTATIONS UNDER TX2 MODE2.

Backbone	Implement	Speed(FPS)	Power(W)	Efficiency(FPS/W)
Darknet7	Serial/1	12.020	7.219	1.665
Darknet7	Pipeline/1	16.132	9.140	1.765
Darknet7	Pipeline/16	21.628	10.088	2.144
Resnet18	Serial/1	14.959	6.700	2.232
Resnet18	Pipeline/1	21.039	8.922	2.358
Resnet18	Pipeline/16	28.458	10.630	2.677

REFERENCES

[1] R. B. Girshick *et al.*, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proc. CVPR*, pp. 580–587, 2014.
[2] S. Ren *et al.*, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE TPAMI*, vol. 39, pp. 1137–1149, 2017.
[3] J. Redmon *et al.*, "You only look once: Unified, real-time object detection," *Proc. CVPR*, pp. 779–788, 2016.
[4] W. Liu *et al.*, "Ssd: Single shot multibox detector," *Proc. ECCV*, pp. 21–37, 2016.
[5] X. Xu *et al.*, "Dac-sdc low power object detection challenge for uav applications," *arXiv:1809.00110*, 2018.
[6] C. Zhuo *et al.*, "From layout to system: Early stage power delivery and architecture co-exploration," *IEEE TCAD*, vol. 38, pp. 1291–1304, 2019.
[7] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, pp. 211–252, 2015.
[8] T. Lin *et al.*, "Microsoft coco: Common objects in context," *Proc. ECCV*, pp. 740–755, 2014.
[9] P. F. Felzenszwalb *et al.*, "Object detection with discriminatively trained part-based models," *IEEE TPAMI*, vol. 32, pp. 1627–1645, 2010.
[10] J. Redmon *et al.*, "Yolo9000: Better, faster, stronger," *Proc. CVPR*, pp. 6517–6525, 2017.
[11] K. He *et al.*, "Deep residual learning for image recognition," *Proc. CVPR*, pp. 770–778, 2016.
[12] A. G. Howard *et al.*, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.
[13] X. Zhang *et al.*, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," *Proc. CVPR*, pp. 6848–6856, 2018.
[14] H. Liau *et al.*, "Fire ssd: Wide fire modules based single shot detector on edge device," *arXiv:1806.05363*, 2018.
[15] T. Lin *et al.*, "Feature pyramid networks for object detection," *Proc. CVPR*, pp. 936–944, 2017.
[16] ——, "Focal loss for dense object detection," *Proc. ICCV*, pp. 2999–3007, 2017.
[17] C. Kyrkou *et al.*, "Dronet: Efficient convolutional neural network detector for real-time uav applications," *Proc. DATE*, pp. 967–972, 2018.
[18] Z. Liu *et al.*, "A multi-level-optimization framework for fpga-based cellular neural network implementation," *ACM JETC*, vol. 14, p. 47, 2018.