Vol.2019-ARC-237 No.33
2019/7/26

Real-time Object Perception with Accelerated On-vehicle Edge AI

SHAOQI CHEN[†1] ZHIMING TAN[†1]
KOICHIRO YAMASHITA[†1] LU SHI[†1]

Abstract: We propose a system of detecting objects from on-vehicle camera, calculating their depth, and then showing them on map. Traditional detection requires huge amount of CPU, GPU, and memory resources on server. It is difficult to execute it on the edge embedded CPU with applicable precision and performance. So we implement with an accelerated deep learning model, which runs 3 times faster than the state-of-the-art models on an embedded CPU by design of networks and memory usage. Our distributed architecture with edge and server can provide real time information on street for various applications, including map, traffic management, autonomous driving, and logistics, etc.

Keywords: Object detection, Edge AI, Deep Learning, Lightweight Convolutional Neural Network

1. Introduction

Real-time object perception with edge devices, such as on-vehicle CPU or mobile CPU, is a challenging task in computer vision. The real-time perception is important for practical applications such as Advanced Driving Assistant System (ADAS), autonomous driving, logistics, and robot navigation, etc. In the past, the mainstream methods for real-time object detection based on the algorithms designed with handcrafted features, such as shape, texture, color, HOG, and DPM [1, 2]. These methods have sophisticated feature representations and stronger pertinence but lack of robustness. With the upgrading of hardware and development of software framework in recent years, deep Convolutional Neural Network (CNN) based detection methods have been applied to object detection for edge devices, which can get robust and high-level feature representations by learning.

Deep CNN based object detection methods can be grouped into two genes: “two-stage detectors”, such as R-CNN, Fast R-CNN, Faster R-CNN, and FPN [3−6], and “one-stage detectors”, such as YOLO series, SSD, and RetinaNet [7−11]. In the former, the detection part usually consists of Region Proposal Network (RPN) [5] and detection head. These detectors tend to utilize a heavy detection calculation (e.g., over 25G Floating-point Operations Per Second, FLOPS [12, 13, 14, 6, 5]) for good accuracy which is too expensive for edge devices. Most researchers have adopted the later because it has traded off the accuracy and speed for detection target. Currently, one-stage detectors such as SSD [10] and YOLO [7, 8, 9] achieve real-time inference on GPU with competitive accuracy, but still run slowly on devices with embedded CPUs.

Modern state-of-the-art networks require high computational resources, beyond the capabilities of many mobile and embedded processors. Many researchers have designed new network architectures which are specifically tailored for edge devices and resource constrained environments. Mark Sandler et al. [15] introduce a novel layer module named as inverted residual with linear bottleneck to significantly reduce the memory footprint needed during inference. Zhang et al. [16] introduce group pointwise convolution to reduce the amount of computation significantly by restricting the input convolution operation to each group. Ma et al. [17] derive four guidelines for efficient network design and present a lightweight architecture called ShufflenetNet V2.

In this paper, we propose an end-to-end lightweight CNN architecture which can be used for real-time object detection on edge devices. Our network run 3 times faster than the state-of-the-art on the embedded CPU with competitive accuracy.

As most detection tasks base on monocular cameras, it is hard to obtain object depth information. The depth is important for recognizing the location of the object in on-vehicle applications without expensive radar sensors. Such applications include Autonomous Emergency Braking (AEB), Forward Collision Waring (FCW), Danger Forecast System (DFS) [36], dynamic mapping, and so on. Fortunately, some specifically designed CNNs enable depth estimation from a single input image, avoiding from multiple viewpoints [18, 19, 20, 21]. They directly obtain estimated disparity for each pixel, then calculate the depth information by providing the camera parameters [22]. However, if we only estimate depth in this way, it lacks of corresponding object attribute information. In this paper, we propose a method for acquiring depth information [22] for objects detected with our lightweight CNN. If additional GPS module on the edge device is available, we can further calculate the location information of the objects and draw it on the map.

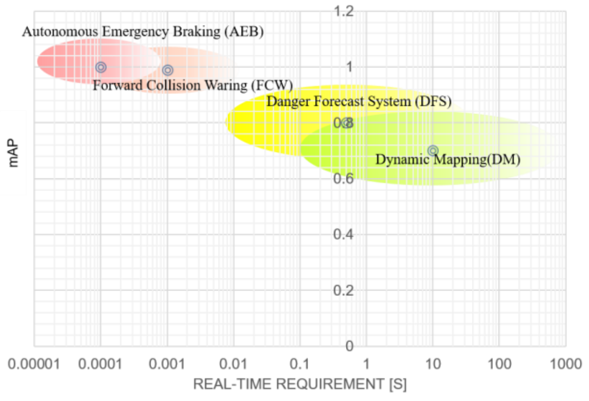

Figure 1. Range of real-time computation for ADAS application.

†1 Fujitsu R&D Center Co., Ltd.

Fig. 1. shows a real-time performance requirement in each allocation. In this paper, it is considered a possible application of Danger Forecast System (DFS) and Dynamic Mapping (DM) on the practical embedded platform.

The rest of this paper is organized as follows. Section 2 describes the related work on lightweight CNN architectures, monocular depth estimation, and object location. The details of them are presented in Section 3. In Section 4, we will show the experimental results with accuracy and speed performance. Conclusion and future directions are summarized in final section.

2. Related Work

2.1 Lightweight CNN architectures

The one-stage detectors are regarded as the first choice for real-time detection on edge devices. For instance, the YOLO family [7, 8, 9] and SSD [10] can run on GPU in real-time. From YOLO v2 to YOLO v3, taking the input image 608x608 for consideration, the mAP on COCO test-dev dataset is boosted from 48.1% to 57.9%. With the gain of accuracy, YOLO v3 has an increment of computation with 141 GFLOPS compared with 63 GFLOPS from YOLO v2. Tiny version of YOLO has been designed to simplify network architecture and reduce computation, which may lead to decreasing of detection accuracy. The mAP of YOLOv3 tiny on COCO test-dev is 33.1% with 5.56 GFLOPS.

Tuning deep CNN architectures to strike an optimal balance between speed and accuracy has been a hot area of research for the last several years. Andrew G et al. come up with depthwise convolution and pointwise convolution to extract features, which reduced time complexity significantly [23]. Furthermore, based on their work, Mark Sandler et al. [15] introduce two structures: linear bottleneck and inverted residual blocks, which can not only remove the redundant information from high-level, but also avoid the information collapsing from low-level. These structures can extract features more fully and efficiently. Zhang et al. introduce pointwise group convolution and channel shuffle to greatly reduce computation cost [16]. Ma et al. propose that the direct metric of speed depends on other factors such as Memory Access Cost (MAC) but not only considering FLOPS, and present an architecture named ShuffleNet V2 based on the work from Zhang et al. Our lightweight network for object detection is called ShuffleNet-YOLOv3, which combines small backbone network and lightweight one-stage detector.

2.2 Monocular depth estimation

There is large number of work focusing on depth estimation from images, such as using pairs [24], several overlapping images captured from multiple viewpoints [25]. These approaches are only applicable when there are more than one input image available. In our task, we focus on the work related to monocular depth estimation with only single input image.

Learning-based stereo estimation algorithms compute the similarity between each pixel in one image and every other pixel in another image. These methods rely on large amounts of ground truth data of disparity and stereo image pairs at training time. This type of data is hard to obtain from real world. For the supervised method of depth estimation from single image, Liu et al. [26] use CNN to learn depth. Ladicky et al. [18] incorporate semantics into their model to improve their per-pixel depth estimation. Eigen et al. [19, 27] show that it is possible to produce dense pixel depth estimation by using a two-scale deep network trained on images and their corresponding depth values. Unlike most other previous work in depth estimation from single image, they do not rely on hand-crafted features or an initial over-segmentation, and instead learn a representation directly from the raw pixel values. Like the stereo methods, these approaches rely on high quality, pixel aligned ground truth depth at training time.

Godard et al. [22] pose monocular depth estimation as an image reconstruction problem, solving for the disparity field without requiring ground truth depth. They also incorporate the left-right consistency check directly into the network to improve the quality of synthesized depth image. At training time, the model learns to predict the depth information for both images of a stereo pair by processing reference image only. This work represents state-of-the-art for monocular depth estimation currently. Based on their work, we estimate the depth information for specific target combined with our lightweight CNN detection network.

2.3 Object location

Location of object in the real world is important for applications such as autonomous driving, obstacle avoidance, and robot operation, etc. Many works related to object location are based on radar-ranging method [28, 29]. Kishigami T. et al. [30] use millimeter-wave radar to perceive the direction and distance of vehicles and pedestrians in wide areas, while Douillard et al. implement Light Detection and Ranging (LiDAR) to do detection and location work [31]. It is hard to get the object category information from these kinds of sensors. To overcome this problem, De Silva et al. propose fusion of information from LiDAR and camera [32]. However, data streams from these sensors are different in many aspects, such as data format, resolution, and geometric alignment. More importantly, the cost and power consumption of radar and LiDAR are high. In this paper, we propose to use vision-based method to realize target location in the real world with our real-time object detection and monocular depth estimation model.

3. Proposed System

The object detection based on deep CNN has shown great advantages. In this paper, we propose a system that can achieve real-time object detection on edge device, depth estimation of target objects, and locating them on the map. The overall architecture of our system is shown in Fig. 2.

Edge device, such as driving video recorder, with a camera is mounted on the front windscreen of a vehicle. We get our deep CNN-based object detection module packaged as an application on the edge device. Image captured from the camera will be sent to the module of Object Detection. Then the inferenced results

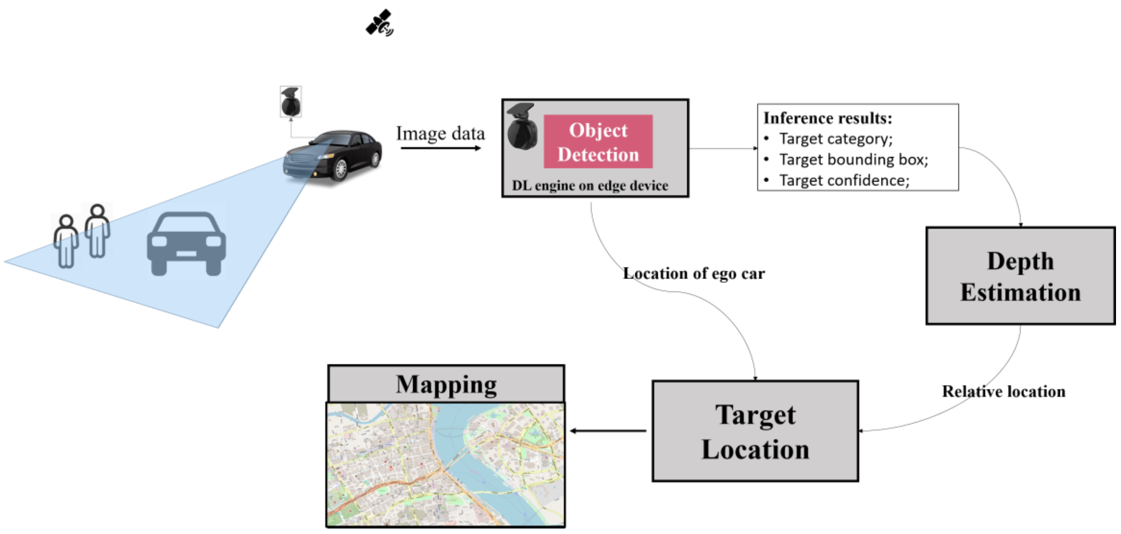

Figure 2. Diagram of system for object detection, monocular depth estimation, location, and mapping with edge AI.

including categories, bounding box of object, and the confidence for each object will be sent to the module of Depth Estimation. In this module, we will obtain 3-dimension information for each object in front of the camera, including the 2D position on the image plane and the depth information. At the same time, the edge device equipped with GPS positioning module will provide location information of ego vehicle. With the 3D information of target and GPS information of ego vehicle, then GPS position of each target object can be calculated. Furthermore, we can locate these target objects on the map. Following we will describe each module in detail.

3.1 Object detection

Recently, there are growing number of researches on deployment of deep learning based task on the edge device, such as object detection, classification, speech recognition, etc. Since the main concern is in constrained computing resources and stricter requirement for power consumption, we should follow the principle of lightweight when designing the network. Here lightweight means smaller calculation load and lower power consumption. Take a panoramic view of the object detection field, YOLO v3 network can get good balance of speed and performance. Although it is one of the one-stage object detection methods, it still needs huge amount of computing resource. YOLO v3 with backbone of DarkNet53 will cost 60+ GFLOPS resource for input image size of 416x416 during one inference. Such huge amount of computation is a burden for edge device with limited computing power, making it more difficult to achieve real-time performance. For instance, PyTorch version YOLO v3 can only run at average 0.3 fps on i5 6200u CPU platform. Looking into the structure of YOLOv3, we can find that the backbone named as DarkNet is composed of conventional residual block like ResNet [14]. Comparing the current mainstream lightweight networks, the residual block is a basic structure of residual network without any optimization for simplifying. In aspects of number and width of structural stacks, quantity and width of blocks are also too complicated. So our design of lightweight network can follow: reducing network width, reducing the number of network layers, and changing residual block to shuffle block (Shuffle Net v2).

Table 1 shows the modification for our lightweight network Shuffle-YOLOv3. Initially, we use shuffle block to replace residual block not only in backbone but also in the yolo detection layers. This modification can lightweight backbone and regression network at the same time. After slimming the network, overall computational operation is reduced by more than half. With the existence of separable convolution and shuffle structure, improvement on speed for deep learning model running on edge device will be even higher.

	Darknet 53-YOLOv3	ShuffleNet v2	**Shuffle-YOLO v3**
Repeat Layers	1-2-8-8-4-yolo(3-3-3)	1-4-8-4-1-fc	1-2-6-6-4-yolo(3-3-3)
Layers width	64-128-256-512-1024-yolo(1024-512-256)	24-116-232-464-1024-fc(1000)	64-128-192-256-384-yolo(384-256-192)
Blocks	Residual	Shuffle	Shuffle
Total computation	60 GFLOPS	-	**25 GFLOPS**

Table 1: The structure of original version of YOLOv3, ShuffleNet v2 and our redesigned network: Shuffle-YOLO v3.

Detailed structure of our network is shown in Fig. 3. As mentioned above, we appropriately modify structure of ShuffleNet v2 and delete some layers. Conversely, due to three YOLO detection layers extracting features from three layers of backbone, we should retain basic structure like Darknet53-YOLOv3 in order to keep enough scale difference in different YOLO layers. So we only reduce four layers. In addition, we make small change inside the Shuffle block, by extending the deep layers' convolution kernel size to 5x5 and changing activation function.

The basic structure of the shuffle block is shown in Fig. 4. It is redesigned based on ShuffleNet v2. We can see that different strides has different channel structures. We relax the restriction of convolution kernel size, so as to adjust shuffle structure

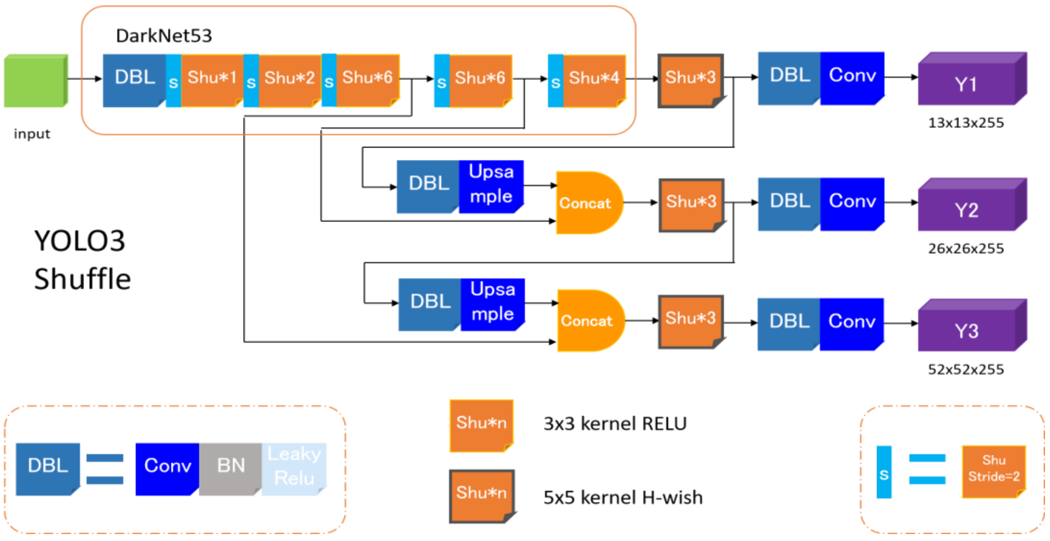

Figure 3. Architecture of Shuffle-YOLOv3

freedom.

Using larger convolution kernels can help network to enhance the capability of deep network features. On the other side, we use h-swish activation function to replace Relu activation function in the deep layers. It is helpful to improve network accuracy, too.

Among all, our optimization points can be concluded as follow. First, we modify the number of ShuffleNet layers. By referring to DarkNet 53, we reduce four layers in middle part of the backbone. Second, we reduce kernel number in each block so as to reduce the amount of calculation. Third, compared with original ShuffleNet, we introduce 5x5 kernel for deep level feature in deep layers. Because 5x5 kernel can remain more receptive field information, it is meaningful for detecting small objects. So we add it in serial deep blocks. Fourth, we replace some Relu activation function by H-swish activation function as MobileNet v3 [33].

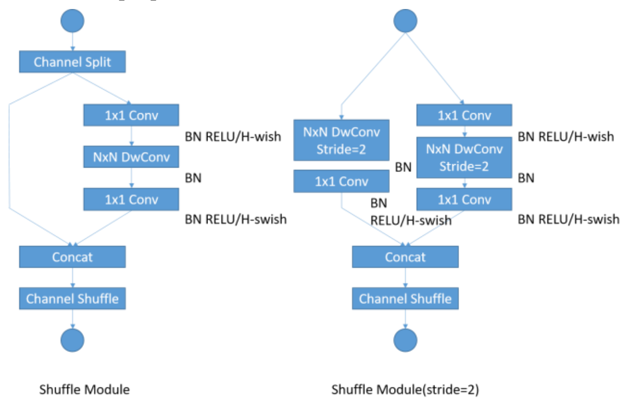

Figure 4. Analysis of Shuffle structure

3.2 Depth estimation

Depth estimation from only one input image is mainly based on the monocular depth estimation network from Godard et al. They pose the depth estimation as an image reconstruction problem during training. The network learns a function that is able to reconstruct one image from the other, then learns about the 3D shape of the scene. The prediction is given in terms of image disparity – a scalar value for each pixel.

According to the results of object detection module, we can locate the targets in the image plane with a bounding box (bbox) closely around them. As shown in Fig.5, the center point "*p*" of bbox for "*person*" is used as reference point to calculate the depth from camera to this "*person*" with formula (1).

$$depth = \alpha * b * f / disp \qquad (1)$$

where "α" is a coefficient for adjustment, "b" is the baseline distance between two cameras, "f" is focal length of the camera, and "*disp*" is disparity value for reference point.

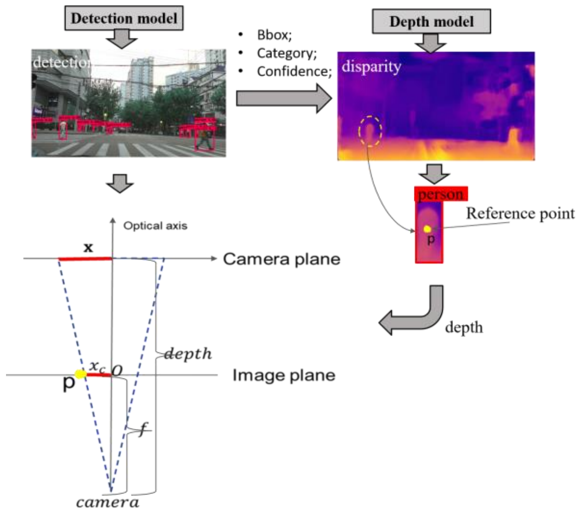

Figure 5: Target depth estimation.

According to the principle of imaging, optical axis will have an intersection point "*O*" with the image plane, and the ideal point "*O*" will be located in the center of image plane. Actually it will be off center, with offset values in two direction, which can be calculated after calibration. Assuming that the intersection point between optical axis and image plane after rectification as point "*O*", we can obtain the transverse distance "*x*" between optical axis and target in camera coordinate system with formula (2).

$$x = (x_c / f) * depth \qquad (2)$$

Where "x_c" is transverse distance between point "O" and reference point "p". By now, we can obtain depth and direction information of the target.

3.3 Target location and mapping

This module aims to obtain the GPS position information of the target objects (e.g. vehicle, pedestrian) detected with detection module and draw them on the map. Position module on the edge device will update GPS position information, namely longitude, latitude, and altitude at a regular intervals. The GPS module used in our edge device is updated every second. We assume that the motion of the ego vehicle with edge device is linear during the interval, so GPS information within this interval can be obtained by linear interpolation. Combined with transverse distance and longitudinal distance information of target object from camera, GPS information of target can be estimated.

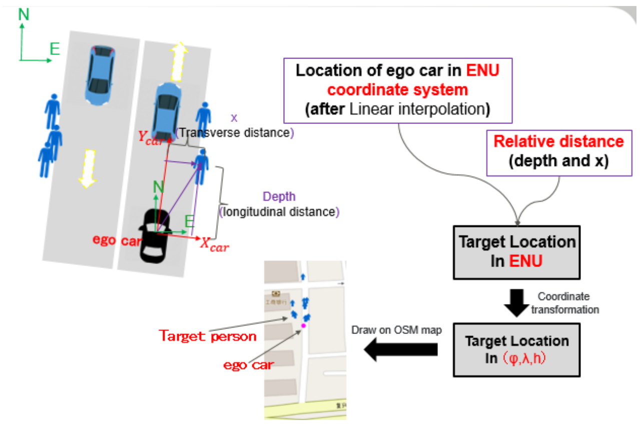
N
E
Location of ego car in ENU coordinate system (after Linear interpolation)
x
Y_{car}
(Transverse distance)
Relative distance (depth and x)
N
E
Depth (longitudinal distance)
ego car
X_{car}
Target Location In ENU
Coordinate transformation
Target person
ego car
Draw on OSM map
Target Location In (φ,λ,h)

Figure 6: Target location and mapping. Edge device with camera and GPS module is mounted on the ego car. "*x*" is transverse distance. "*depth*" is longitudinal distance.

The processing procedure of this module is shown in Fig. 6. The ENU coordinate system denotes to the East-North-Up local coordinate system, where the U axis is outward, and *Xcar-Ycar* is the camera coordinate system, assuming that the optical axis of the camera is along the driving direction of the car. Firstly, the position (φ, λ, h) of ego car in geodetic coordinate system is converted to ENU coordinate system, where "φ", "λ" and "h" represent longitude, latitude, and altitude in WGS84 (World Geodetic Coordinate System 1984), respectively. Then we estimate the positions of targets in ENU coordinate system. After that, their positions are transferred back to geodetic coordinate system with form of (φ, λ, h). We can draw out the targets on some open source maps with their estimated GPS location.

3.4 Optimal memory usage

In addition to CPU calculation load, it's also necessary to consider minimization of memory footprint to implement on the edge devices. If the model runs out of memory, the app would get terminated by operating system. The memory usage also has an influence on the computing speed and the power consumption, and affects how quickly the battery will be drained or makes the edge device too hot.

One way to speed up the model is to simplify the computation it does. We typically count this with FLOPS or Multiply-Accumulate Operations (MACCs). The number of computations, whether count with FLOPS or MACCs, takes only part of the runtime. The memory usage is another part which may seem even more important.

We only do inference operation on edge device, commonly without training operation. In one layer, the device need to read the input feature map from memory, then compute the dot products by reading the layer's parameters from memory, and finally write the results as new feature map back to memory. The huge amount of memory reading and writing will have a big impact on the speed. Memory requirements mainly come from two aspects. The first is the memory occupied by model parameters, and the second is the memory used for layer outputs. For the former, parameters of convolution operation which need to be trained can be calculated as $C_{in}*C_{out}*K^2+C_{out}$, where C_{in} and C_{out} are the channel numbers of input and output feature maps respectively, and K is kernel size. For the later, the memory footprint of each output layer is calculated as $C_{out}*H*W$, where $H*W$ is output shape.

To reduce the memory used by network parameters, we bring in bottleneck block to the shuffle module in backbone, using 1x1 convolution to decrease the number of channels from input feature map. Then followed with NxN (normally set to 3x3) depthwise convolution and 1x1 pointwise convolution, this bottleneck block will greatly reduce model parameters and FLOPS, as shown in Fig. 4. To reduce memory occupied by intermediate layer output, we appropriately reduce the repeating layers and layers width as not to affect the feature extraction effect, as shown in Table 1. For our network with the inference on one image with input size 416x416, the memory footprint is around 220MB, taking about 48.8% of the original YOLOv3.

4. Experiment and Result

In this section, we evaluate the effectiveness of our modified ShuffleNet-YOLOv3 on PASCAL VOC [34, 35] benchmarks. Then we show some comparison results with several lightweight objection networks currently used on mobile edge devices.

Our detectors are trained end-to-end on one 1080Ti GPU. The input resolution is 416x416. Multi-scale training with pixels in the range {320-608} is adopted. We use heavy data augmentation for training.We report our detection results on two datasets. The first one is traffic dataset made by ourselves, with samples from autopilot and traffic surveillance. It mingles BDD dataset, cityscape dataset, and our own images. The second dataset is the open dataset: VOC. The results are given in Table 2 and Table 3.

Model	Backbone	Input	Model Size(MB)	mAP	FPS (CPU)
YOLOv3[9]	Darknet-53	416x416	243	71.8	1.25
Shuffle-YOLOv3	Modified shuffle	416x416	**45.5**	**70.8**	**4**

Table 2. Evaluation results on our traffic dataset (6 classes).

Model	Backbone	Input	VOC2012 IOU=0.5	VOC2007 IOU=0.5	COCO IOU=0.5
YOLOv3[9]	Darknet-53	416x416	-	85.5(07+12+coco)*	55.3
Shuffle-YOLOv3	Modified shuffle	416x416	**81.8(07+12)**	**82.6(07+12)**	**52.0**
SSD	VGG19	321x321	74.9(07+12)	76.8(07+12)	45.4

Table 3. Evaluation results (mAP) on VOC and COCO (YOLOv3 VOC testing result comes from third-party)

Our modified ShuffleNet-YOLOv3 surpasses prior state-of-the-art one-stage detectors. ShuffleNet-YOLOv3 with 416x416 is only 18.7% of the model size of YOLOv3, while the mean Average Precision (mAP) is just reduced about 1% (71.8% -> 70.8%). Moreover, our model performs faster than YOLOv3 by nearly 4x.

Furthermore, ShuffleNet-YOLOv3 achieves superior results to state-of-the-art large object detectors such as SSD300*[10], which has 31.75 GFLOPS. We reduce the computational cost by orders of magnitude. The backbone of our model is significantly smaller than the large detectors.

Fig. 7 visualizes several examples on VOC test-dev. ShuffleNet-YOLOv3 achieves a much better trade-off between accuracy and efficiency, which is not only efficient but highly accurate.

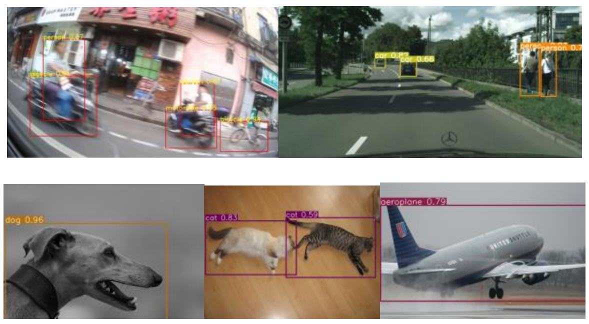

Figure 7. Examples visualization on traffic dataset and VOC test-dev.

At last, we evaluate the inference speed of ShuffleNet-YOLOv3 on Cortex A72 1.8GHz, Intel I7 3.0 GHz, and GeForce 1080Ti (GPU). On A72 and I7, the inference is executed with a single thread. The results are shown in Table 4. We achieve faster detection on both A72 and I7 at 2 fps and 4 fps respectively, compared with Darknet-YOLOv3. On GPU, our model can run at over 60 fps.

Fig. 8(b) shows the disparity image obtained by the depth estimation module. This module runs at 28 fps on Titan X with input resolution of 512x256. The error of estimated depth is around 5%. We follow Godard's training work on KITTI training datasets, where the baseline b is 0.54m, and the focal length f is 1012. There are some difference between our camera parameters and camera parameters for KITTI. As inference with model trained with KITTI dataset may exist bias for our camera, so coefficient " a " is used for adjustment. In our experiments, " a " is set to 1367, which may differ from different cameras. The camera used in our experiment has a configuration of resolution 1980x1080, CMOS sensor, horizon FOV 120°.

Model	ARM (Cortex A72 1.8GHz)	CPU (Intel I7 3.0GHz)	GPU (NVIDIA 1080Ti)
Darknet-YOLOv3	1500ms	800ms	33ms
Shuffle-YOLOv3	500ms	250ms	16.7ms

Table 4. Inference on ARM, CPU, GPU.

Fig.8(d) shows two top-view maps drawn with detected targets (vehicles and pedestrians) after location estimation.

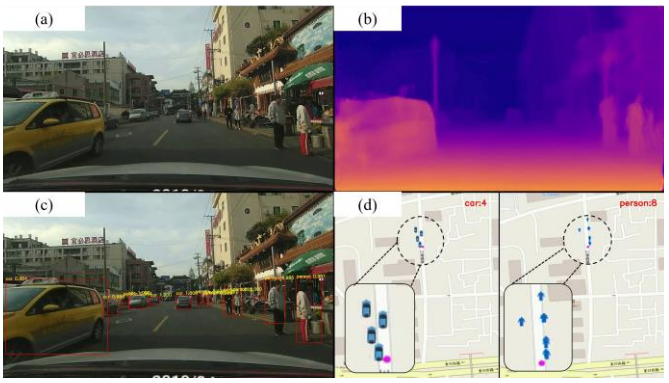

Figure 8. Results on images captured from the driving video recorder. (a) Original input image. (b) Disparity image. (c) Object detection results. (d) Detected targets (e.g. pedestrian and vehicle) are painted as blue icon on the map with estimated location, pink point on (d) refer to the ego car with edge device.

5. Conclusion

In this work, we propose a system of detecting objects from on-vehicle camera, estimating their depth, and drawing them on the map. For the object detection part, we design a lightweight network architecture, and achieve faster object detection on edge device with balance between the operation speed and detection accuracy. It runs in 3 times faster than the state-of-art on embedded CPU by arranging networks and memory. For our depth estimation module, we further get the location information for each object from object detection module. It runs with 28 FPS on GPU, the distance error within 40 meters does not exceed 5%. The distributed architecture, which is implemented by edge and server, can provide real-time information on street for various potential application, including map, V2X, autonomous driving, and logistics, etc.

Reference

[1] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1. IEEE, 2005, pp. 886-893.

[2] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in

Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. IEEE, 2008, pp. 1-8.
[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for object detection and semantic segmentation," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2014, pp. 580-587.
[4] R. Girshick, “Fast r-cnn,” in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1440-1448.
[5] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in Advances in neural information processing system, 2015, pp. 91-99.
[6] Lin, Tsung Yi , et al. "Feature Pyramid Networks for Object Detection." 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) IEEE Computer Society, 2017.
[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 779-788.
[8] J. Redmon, A. Farhadi, YOLO9000: Better, Faster, Stronger. arXiv:1612.08242.
[9] J. Redmon, A. Farhadi, “Yolov3: An incremental improvement,” arxiv preprint arXiv:1804.02767, 2018.
[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, “Ssd: Single shot multibox detector,” in European conference on computer vision. Springer, 2016, pp. 21-37.
[11] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal loss for dense object detection,” IEEE transactions on pattern analysis and machine intelligence, 2018.
[12] Z. Cai and N. Vasconcelos. Cascade r-cnn: Delving into high quality object detection. arXiv preprint arXiv: 1712.00726, 2017.
[13] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In Advances in neural information processing systems, pages 379-387.
[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770-778, 2016.
[15] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. –C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4510-4520, 2018.
[16] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6848-6856, 2018.
[17] N. Ma, X. Zhang, H.-T. Zhang, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European Conference on Computer Vision (ECCV), pages 116-131, 2018.
[18] L. Ladicky, J. Shi, and M. Pollefeys, “Pulling things out of perspective,” in CVPR, 2014, pp. 89–96.
[19] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” in Advances in neural information processing systems, 2014, pp. 2366–2374.
[20] F. Liu, C. Shen, G. Lin, and I. Reid, “Learning depth from single monocular images using deep convolutional neural fields,” IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 38, no. 10, pp. 2024–2039, 2016.
[21] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, “Unsupervised learning of depth and ego-motion from video,” in CVPR, vol. 2, no. 6, 2017, p. 7.
[22] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in CVPR, vol. 2, no. 6, 2017, p. 7.
[23] Howard, A. G. , Zhu, M. , Chen, B. , Kalenichenko, D. , Wang, W. , & Weyand, T. , et al. (2017). Mobilenets: efficient convolutional neural networks for mobile vision applications.
[24] D. Scharstein and R. Szeliski. A taxonomy and evalution of dense two-frame stereo correspondence algorithms. IJCV, 2002. 2
[25] Furukawa Y , Hernάndez, Carlos. Multi-View Stereo: A Tutorial[J]. Foundations and Trends? in Computer Graphics and Vision, 2015, 9(1-2):1-148.
[26] Liu F , Shen C , Lin G , et al. Learning Depth from Single Monocular Images Using Deep Convolutional Neural Fields[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2015, 38(10):2024-2039.
[27] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In ICCV, 2015. 2
[28] Pohl N , Gerding M , Will B , et al. High Precision Radar Distance Measurements in Overmoded Circular Waveguides[J]. IEEE Transactions on Microwave Theory and Techniques, 2007, 55(6):1374-1381.
[29] Denicke E , Armbrecht G , Rolfes I . Radar distance measurements in circular waveguides involving intermodal dispersion effects[J]. International Journal of Microwave and Wireless Technologies, 2010, 2(3-4):409-417.
[30] Kishigami T , Morita T , Mukai H , et al. Advanced Millimeter-Wave Radar System to Detect Pedestrians and Vehicles by Using Coded Pulse Compression and Adaptive Array[J]. IEICE Transactions on Communications, 2013, E96.B(9):2313-2322.
[31] Douillard, Bertrand, et al. "On the segmentation of 3D LIDAR point clouds." Robotics and Automation (ICRA), 2011 IEEE International Conference on. IEEE, 2011.
[32] De Silva V , Roche J , Kondoz A . Fusion of LiDAR and Camera Sensor Data for Environment Sensing in Driverless Vehicles[J]. 2017.
[33] Howard, A., Sandler, M., Chu, et al. Searching for MobileNetV3, arXiv:1905.02244. 2019.
[34] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” International journal of computer vision, vol. 88, no. 2, pp. 303-338, 2010.
[35] M. Everingham, S.A. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes challenge: A retrospective,” International journal of computer vision, vol. 111, no. 1, pp. 98-136, 2015.
[36] Kouhei Hashitmo, Yutaro Ishida, Ryutaro Ichise, Hiroaki Wagatsuma and Hakaru Tamukoh, “On-Vehicle Danger Forecast System based on Knowledge-based Artifical Intelligence”, SCI’17, Vol.32, No.5, pp.191-201, 2018

Acknowledgments Thanks for the members effort to our system, also the authors mentioned in the cited paper who shared their results.