

# RefineDetLite: A Lightweight One-stage Object Detection Framework for CPU-only Devices

Chen Chen<sup>1</sup>, Mengyuan Liu<sup>1</sup>, Xiandong Meng<sup>2</sup>, Wanpeng Xiao<sup>1</sup>, Qi Ju<sup>1</sup>

<sup>1</sup>Tencent TEG AI    <sup>2</sup>The Hong Kong University of Science and Technology

{beckhamchen, mengyuanliu, wanpengxiao, damonju}@tencent.com, xmengab@connect.ust.hk

## Abstract

*Previous state-of-the-art real-time object detectors have been reported on GPUs which are extremely expensive for processing massive data and in resource-restricted scenarios. Therefore, high efficiency object detectors on CPU-only devices are urgently-needed in industry. The floating-point operations (FLOPs<sup>1</sup>) of networks are not strictly proportional to the running speed on CPU devices, which inspires the design of an exactly “fast” and “accurate” object detector. After investigating the concern gaps between classification networks and detection backbones, and following the design principles of efficient networks, we propose a lightweight residual-like backbone with large receptive fields and wide dimensions for low-level features, which are crucial for detection tasks. Correspondingly, we also design a light-head detection part to match the backbone capability. Furthermore, by analyzing the drawbacks of current one-stage detector training strategies, we also propose three orthogonal training strategies—IOU-guided loss, classes-aware weighting method and balanced multi-task training approach. Without bells and whistles, our proposed RefineDetLite achieves 26.8 mAP on the MSCOCO benchmark at a speed of 130 ms/pic on a single-thread CPU. The detection accuracy can be further increased to 29.6 mAP by integrating all the proposed training strategies, without apparent speed drop.*

## 1. Introduction

Object detection is a fundamental technology in the computer vision society and is also a crucial component for many high-level artificial intelligence tasks, *e.g.*, object tracking [59], vision-language transferring [9, 13], surveillance, autonomous driving [58] and robotics. Benefited from the rapid development of deep learning, the accuracy of object detection has been greatly improved. However, with the explosive growth of social media informa-

tion, the high computational complexity seriously hinders the wide applications of object detection algorithms. Therefore, much attention has been paid to the study of how to make trade-off between detection accuracy and implementation complexity. Thanks to the powerful parallel processing ability of GPUs, many researchers claimed they have achieved real-time detection. However, GPUs are still extremely high cost in terms of dealing with massive data. Consequently, research into fast object detection pipelines on computationally constrained devices (*e.g.*, CPU-only computers and mobile devices) is extremely urgent.

Inspired by the pioneering deep-learning-based R-CNN serials ([20, 19, 47]), most state-of-the-art detectors are inclined to exploit classical classification networks [22, 53] as the backbone part. Obviously, the computational complexity of backbone networks is the important bottleneck that affects the running efficiency of the whole detector, and hence many lightweight algorithms employ famous efficient convolution networks [25, 49, 24, 62, 41, 27, 18] instead. However, as pointed out in [35], there exists gaps between the design principles of classification and detection networks. For instance, the larger receptive fields and wider feature vectors of early stages are crucial for improving localization ability, while classification networks care only about the feature representation ability of the last layer. Therefore, directly employing classification networks as the backbones maybe not the optimal strategy. Additionally, another important issue that must be recognized is that the number of FLOPs is not strictly proportional to the running time since many other factors (*e.g.*, memory access cost and degree of parallelism) impact the practical network latency [41]. Therefore, how to design an actually “**fast**” detection backbone network running on a CPU is a critical demand in industrial practice.

Typically, CNN-based object detectors are categorized into either two-stage detectors or one-stage detectors based on different processings of the detection part. Two-stage [47] detectors usually contain a region proposal network (RPN), a ROI warping module and a localization and classification subnet. More elegantly, one-stage de-

<sup>1</sup>Here, FLOPs means the number of multiply-adds following [41].

tectors [44, 45, 46, 40] directly output bounding boxes and classification probabilities through only once network forward pass. In general, two-stage detectors are usually considered to be more accurate on detection because of the bounding boxes “**refinement**” operation during the second stage, but are more time-consuming as compared to one-stage detectors. Intuitively, in the past few years, the majority of researchers have been dedicated to studying lightweight detection structures of one-stage detectors [2, 56, 32, 49, 58]. But, the low detection accuracy cannot satisfy the practical requirements because of the coarse localization and classification through only single stage prediction. Therefore, other works were encouraged to develop or even automatically search lightweight detection architectures based on two-stage detectors [34, 43, 17], which have achieved a relatively higher detection accuracy under low computational complexity.

To inherit the merits of both two-stage (high accuracy) and one-stage (high efficiency) detectors, Zhang *et al.* proposed a single-stage refinement network (RefineDet [61]), which can be viewed as a pseudo two-stage detector. By adding a lightweight FPN-like feature refinement stage, RefineDet simulates the anchor refining process during the second stage of the two-stage detector. Since RefineDet optimizes the anchor refinement loss and the object detection loss simultaneously, it can achieve high detection accuracy and efficiency at the same time. Inspired by this effective architecture and following the design principles of fast convolution networks suitable for detection, we propose a lightweight version of RefineDet, named RefineDetLite.

Besides the network architecture design, we further analyze the drawbacks of current one-stage detectors. With the aim of improving the detection accuracy without increasing any inference computational load, we concentrate on optimizing the loss functions and training strategies. First, we focus on the training data imbalance problem among different classes. Second, we propose an intersection-over-union (IOU)-guided loss to overcome the inconsistency between localization and classification confidences during training. Finally, going deep into the essence of object detection, we classify it as a multi-task training problem. In order to keep the balance of localization and detection losses during training, we introduce some trainable balancing coefficients and derive the final loss formulation in detail following the theoretical guidance of [28].

In summary, the main contributions of our proposed efficient object detection framework are as follows:

- Based on the RefineDet, analyzing the design key points of efficient convolutional networks, we propose a lightweight backbone and detection head specifically designed for the detection task. We call the entire network structure RefineDetLite.

- Without introducing any extra computational cost during inference, we propose some general training strategies (IOU-guided loss, classes-weighted loss and balanced multi-task training) to further improve the detection accuracy.
- RefineDetLite surpasses many state-of-the-art lightweight one-stage and two-stage detectors with faster running speed on CPUs and higher detection accuracy on the MSCOCO benchmark. It can achieve **29.6 mAP** on MSCOCO online test-dev at a running speed of **131 ms/pic** on a single thread CPU<sup>2</sup>.

## 2. Related Works

### 2.1. Deep-learning-based object detectors

The well known R-CNN [20] is the pioneer of the deep-learning-based object detectors, which creatively utilizes convolutional networks to predict object regions and class labels based on a sparse set of pre-extracted candidate region proposals. Encouraged by the R-CNN prototype, numerous successors boosted the performance of CNN-based detectors by optimizing candidate extracting approaches [19, 47, 30], feature fusion methods [37, 12, 52], training strategies [42, 51, 55] and contextual reasoning [10, 14, 26].

In addition to the complex two-stage diagram, another fast-growing pipeline is the one-stage detectors, which directly predict object bounding boxes and category probabilities through a single forward pass and end-to-end training. Considered to be more straightforward and efficient, in the past few years, following YOLO [44, 45, 46] and SSD [40], a large number of researches have paid more attention to bridging the detection accuracy gap between two-stage and one-stage detectors. DSSD [15], FSSD [36] and DSOD [50] exploit different feature fusion methods to ameliorate the weak representation ability of low-level feature. Taking a step further, RefineDet [61] introduces an extra loss refinement stage to considerably improve small-object detection accuracy without significantly increasing the complexity.

### 2.2. Efficient object detection

Although many stat-of-the-art object detectors claim that they have achieved real-time detection speed, most of them were experimented on GPUs, which are still a heavy burden for personal users or industrial massive-data scenarios. Therefore, research on lightweight detection frameworks are prevalent in the object detection community. Noting the advantages described above, ideas about how to simplify one-stage detection architectures are overwhelming. SSDLite [49], Tiny-SSD [57] and Tiny-YOLO [2] intuitively

<sup>2</sup>The CPU type is Intel i7-6700@3.40GHz

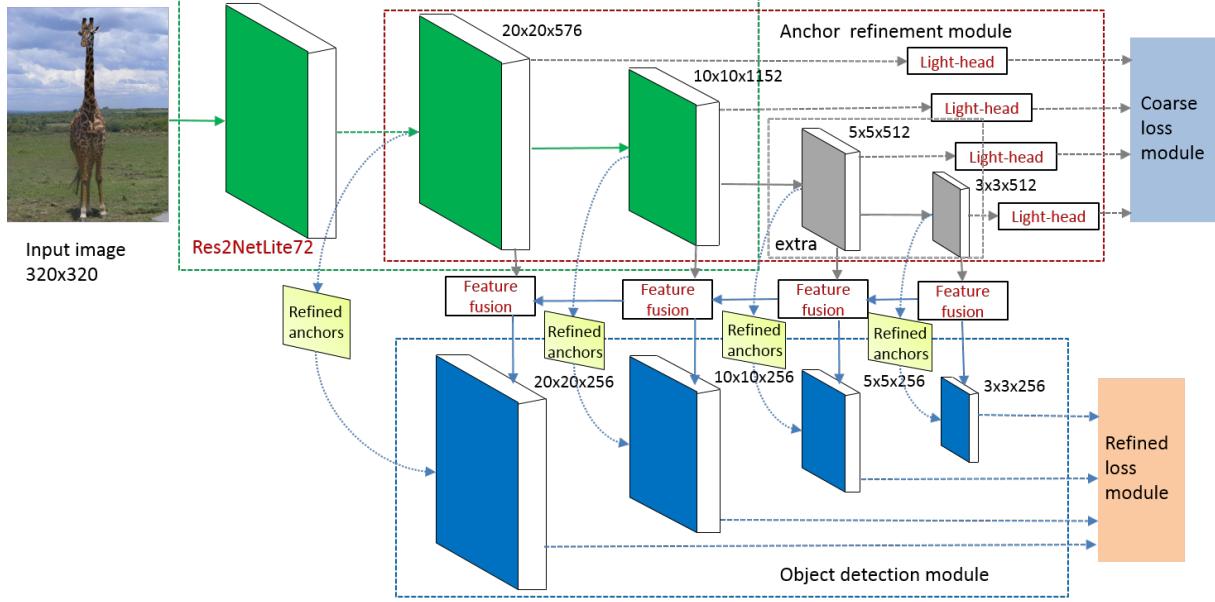


Figure 1. The overall framework of our proposed RefineDetLite.

employ lightweight backbones and detection heads to replace original components of YOLO or SSD. To improve such naive thoughts, Tiny-DSOD [33] and Pelee [54] propose more effective network simplification schemes based on advanced versions of SSD. However, even achieved speed boost, they perform poorly in detection accuracy. Alternatively, another group of researchers attempted to improve the detection efficiency by re-designing network structures manually [43] or through neural architecture searching (NAS) [17] based on two-stage paradigms.

### 2.3. Training and inference strategy optimization

In addition to making great efforts to optimize the network structure, many researchers have been dedicated to distilling some general training or inference strategies to further improve the detection accuracy for existing detectors, with little increasing of computational complexity.

Inheriting the standard non-maximum suppression (NMS), soft-nms [11] slightly modifies this process to achieve a remarkable mAP increase. Taking this a step further, softer-nms [23] adds a small portion of parameters during training and inference by taking into account the uncertainty of bounding box regression. Goldman *et al.* [21] further proposed a soft IOU layer and EM-Merger unit to reduce the bounding box prediction uncertainty.

To alleviate the well-known imbalance problem between positive and negative samples for one-stage detectors, online hard example mining (OHEM [51]), focal loss [38] and the gradient harmonizing mechanism (GHM) have been successively proposed.

Additionally, Yu *et al.* [60] noticed a long-time neglected

misalignment problem—minimizing bounding box offsets does not strictly equal to maximizing IOU, the evaluation metric for regression accuracy. Hence, an IOU loss was invented to be directly used as a regression loss. Successively, GIOU [48], DIOU and CIoU [63] were proposed to further improve the IOU-based evaluation metrics. Another easily overlooked misalignment phenomenon for one-stage detectors is the inconsistency between the training hypothesis and inference configurations. Kong *et al.* [29] proposed a consistent optimization strategy to bridge the gap.

## 3. RefineDetLite

In this section, we present in detail the RefineDetLite network architecture. Following the design principles of efficient convolutional networks on CPUs, our proposed detection network keeps the balance between efficiency and accuracy to the maximum degree.

### 3.1. Overall framework

Fig. 1 elaborately illustrates the overall framework of RefineDetLite, which inherits the design idea of RefineDet [61]. It first extracts pyramidal features to predict coarse bounding boxes and decides whether the anchor is foreground or background, which is called the anchor refinement module (ARM). Then, the skillfully contrived object detection module (ODM) fuses the pyramidal features reversely and employs the refined anchors to further predict precise bounding boxes and exact object classes. Since the network outputs box regions and class probabilities through only one forward pass, it is still a one-stage object detector. However, the ingenious structure achieves high efficiency

Table 1. Overview of Res2NetLite architecture

Stage	Layer	Output Size
	input	224×224×3
Stage0	BatchNorm	224×224×3
	Convolution	112×112×32
Stage1	Maxpooling	56×56×32
	Downsample	Bottleneck
Stage2	Feature enhancement	Res2Blocks×3
	Downsample	Bottleneck
Stage3	Feature enhancement	Res2Blocks×7
	Downsample	Bottleneck
Stage4	Feature enhancement	Res2Blocks×3
	Average pooling	7×7, stride=1
	Convolution	1×1, stride=1
		1000-d

and detection accuracy simultaneously.

As demonstrated in [43], the input resolution should match the capability of the backbone network, so we fixed the input resolution as  $320 \times 320$  for the high efficiency backbone. Then, two groups of pyramidal features with the resolutions  $\{20 \times 20, 10 \times 10, 5 \times 5, 3 \times 3\}$  are used to calculate coarse losses and refined losses, respectively.

Since the original RefineDet is not concerned about the network efficiency on CPUs, we are enlightened to improve this structure by introducing a new efficient backbone, re-designing a light-head detection part and a lightweight feature fusion module.

### 3.2. Backbone

High efficiency with strong feature representation ability is fundamental to lightweight accurate object detectors. As mentioned in the introduction, although many state-of-the-art algorithms borrow classification networks transferred directly from ImageNet pre-trained models as the backbone, there are obvious gaps between classification networks and the detection backbone. To avoid resulting in suboptimal architectures, we propose a new ResNet-like lightweight backbone network based on design principles for detection networks. Table 1 lists in detail the structure of our proposed backbone, *Res2NetLite*.

In “Stage0”, Res2NetLite first quickly down-samples the input resolution to  $1/4 \times 1/4$  and expands the feature dimension to 32 through a simple batchnorm-convolution-maxpooling combination. Then, three stages are concatenated to form the main part of the network. In each stage, following one bottleneck module (Fig. 2(a)), several repeated Res2Blocks [16] (3, 7 and 3 for Stage 1, 2 and 3, respectively) enhance the feature representation ability gradually. After each stage, the feature resolution is halved but the dimension is doubled. The  $c$  is a feature dimension hyper-parameter to control the trade-off between network efficiency and accuracy. Finally, “Stage4” is just a conventional global pooling together with a  $1 \times 1$  convolution layer.

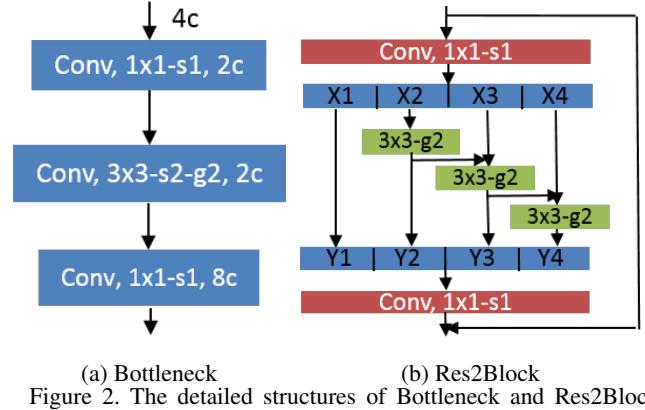
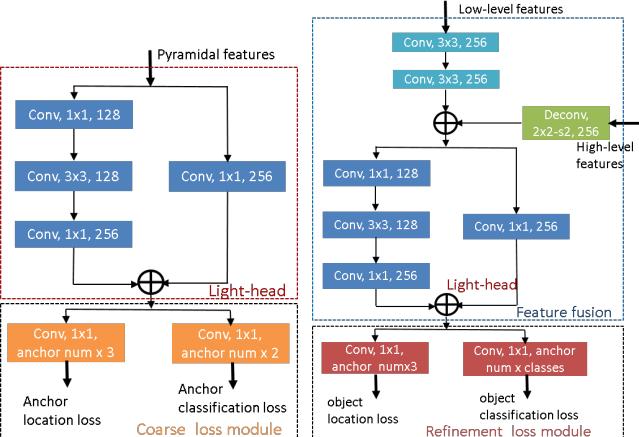


Figure 2. The detailed structures of Bottleneck and Res2Block used in Res2NetLite.

**Suitable for detection.** Different from classification networks, who are only concerned about the feature representation ability of the last convolution layer, pyramidal feature detection backbones care more about that of early stage features. So based on the basic consensus, we broaden the dimensions of low-level features deliberately as much as possible. For instance, in terms of Res2NetLite72 ( $c=72$ ), the dimensions of the four pyramidal features used for anchor refinement in Fig. 1 are  $\{576, 1152, 512, 512\}$  (the last convolution layers of “Stage2” and “Stage3”, as well as two extra bottleneck modules as shown in Fig. 2(a)). Another important issue is that detection backbones require large receptive fields to capture sufficient contextual information for effective localization and classification of large objects in the early stage. Therefore, based on the above two considerations, the key point of our proposed backbone is the exploitation of Res2Block [16]. As depicted in Fig. 2(b), Res2Block splits features into several groups and constructs hierarchical feature connection in a single residual block. Compared with the well-known ResBlock [22], Res2Block achieves multi-scale feature fusion and receptive field expansion but has less computational complexity.

**High efficiency.** In [41], Ma *et al.* derived several practical guidelines for an effective network architecture. In addition to the computational complexity of the network, two other factors—memory access cost (MAC) and degree of parallelism play the crucial role in affecting the exact running speed, especially on CPUs. Following the guidance, we strictly ensure the input and output channels are the same for Res2Blocks, which minimizes the MAC. On the other hand, we employ maximal  $g = 2$  (as shown in Fig. 2) for all group convolutions in Bottlenecks and Res2Blocks since deep-wise convolutions are still not well optimized in CPU devices. Finally, we do not add any fragment structure, like inception or squeeze-and-excitation modules, in Res2NetLite. Generally speaking, our proposed backbone is a kind of straightforward and elegant network without any bypass branches. We enhance the feature representa-



(a) Light-head detection part

Figure 3. lightweight detection head for coarse losses and refined losses modules. The key component is the Light-head module.

tion ability simply by broadening feature dimensions and fusing multi-scale features inside each single block.

### 3.3. Lightweight detection head and feature fusion

To match the backbone capability, we propose the corresponding lightweight detection head and feature fusion module as illustrated in Fig. 3. The key component is the residual-like *Light-head* module as shown in Fig. 3(a). Instead of original plain  $3 \times 3$  convolution layers, the Light-head module further fuses high-level features with different receptive fields in a multi-scale two-pass way (a bottleneck structure and a  $1 \times 1$  convolution layer). Then, this module also enables us to utilize only  $1 \times 1$  convolutions to directly output location and class predictions.

## 4. Training strategy improvements

### 4.1. Loss function

The overall loss  $L$  of RefineDetLite consists of four parts: location and classification losses for both the ARM (coarse losses) and ODM (refined losses), as shown in Eq. 1.

$$L = \lambda_{loc}^{arm} L_{loc}^{arm} + \lambda_{cls}^{arm} L_{cls}^{arm} + \lambda_{loc}^{odm} L_{loc}^{odm} + \lambda_{cls}^{odm} L_{cls}^{odm}, \quad (1)$$

where the  $\lambda$ s are the weighted coefficients and typically fixed to 1, empirically. In most state-of-the-art papers, the location loss is approximated by the smooth L1 loss. But recently, Rezatofighi *et al.* proved that directly optimizing IOU metric is a better approach and accordingly proposed a GIOU loss [48]. In terms of the classification loss, cross entropy loss is the most widely used method. In this paper, we propose to adopt a weighted KL-divergence loss as the

classification loss for ODM as follows:

$$\begin{aligned} L_{cls}^{odm}(\mathbf{X}, \mathbf{Y}) &= \frac{1}{N} \sum_{j=0}^{N-1} \eta_j \cdot L_{cls}(\mathbf{x}_j, \mathbf{y}_j) \\ &= \frac{1}{N} \sum_{j=0}^{N-1} \eta_j \cdot \left( \sum_{i=0}^K w_i \cdot y_{ji} \log \frac{y_{ji}}{\text{softmax}(x_{ji})} \right), \end{aligned} \quad (2)$$

where  $N$  is the mini-batch size,  $j$  is the index of the anchor in the mini-batch,  $\mathbf{x}_j$  is the network output predictions,  $\mathbf{y}_j = (y_{j0}, y_{j1}, \dots, y_{ji}, \dots, y_{jK})$  is the soft ground-truth label for anchor  $j$ ,  $K$  is the number of classes ( $K - 1$  foreground classes together with one background class),  $w_i$  is the class weight for class  $i$  and  $\eta_j$  is the anchor sample weight.

### 4.2. IOU-guided loss

Apparently, if the ground-truth label  $\mathbf{y}_j$  is the traditional one-hot vector, Eq. 2 is exactly the weighted sum of binary cross entropy losses. So what we concerned with is to reform the soft ground-truth label  $\mathbf{y}_j$  to improve the final network detection accuracy.

In almost all one-stage detectors, an anchor will be assigned a hard label with the probability 1 if the maximal IOU between the anchor and a ground-truth box is above a threshold (0.5 for instance). But unfortunately, the detector cannot always optimize the candidate anchor perfectly, which means the final IOU can not be exactly equal to 1 in most situations. So, this easily-neglected discrepancy may cause inconsistency when jointly optimizing localization and classification losses. Finally, it will lead to a phenomenon that some badly localized bounding boxes are classified as foreground objects with very high confidence.

To alleviate this inconsistency, we propose a so-called IOU-guided loss. Based on the above analysis, we are motivated to replace the one-hot vector  $\mathbf{y}_j$  with soft labels. Denote the hard label for anchor  $j$  as  $t_j$  ( $t_j = \{0, 1, \dots, K\}$  (assuming that  $t_j = 0$  means negative sample)). The key item  $y_{jt_j}$  in vector  $\mathbf{y}_j$  is calculated as a function of  $\widehat{\text{IOU}}_j$  as

$$y_{jt_j} = \begin{cases} 1 - \alpha(1 - \widehat{\text{IOU}}_j), & t_j > 0 \text{ (positive sample)} \\ 1 - \alpha\widehat{\text{IOU}}_j, & t_j = 0 \text{ (negative sample)} \end{cases}, \quad (3)$$

where  $\alpha$  is a hyper-parameter. After adding the predicted offsets onto the pre-defined anchor  $j$ , we can re-calculate a new IOU value ( $\widehat{\text{IOU}}_j$ ) between the refined anchor  $j$  and the pre-assigned ground-truth box. Notice that  $\widehat{\text{IOU}}_j \in [0, 1]$ , based on the bounding box prediction accuracy. Therefore, for a positive sample, if the post-calculated  $\widehat{\text{IOU}}_j$  is small, the ground-truth label value  $y_{jt_j}$  decreases accordingly.

To normalize the soft label  $\mathbf{y}_j$ , the other items except  $y_{jt_j}$  are formulated as  $y_{ji} = \frac{1 - y_{jt_j}}{K-1}$ . We also weight the

anchor samples according to  $\widehat{\text{IOU}}_j$  as

$$\eta_j = \frac{1}{1 + \beta(\widehat{\text{IOU}}_j - 1) \cdot \mathbf{1}(t_j > 0)}, \quad (4)$$

where  $\beta$  is another hyper-parameter. The physical meaning of Eq. 4 is that harder positive samples (lower post-calculated IOU value) will be assigned higher weights.

### 4.3. Dataset-aware classes-weighted loss

In the past few years, many researchers focused on solving the imbalance problem between positive and negative samples for one-stage detectors [38, 31], but few works dealt with the imbalance problem among different classes in training data. Consequently, we propose a classes-weighted loss based on the statistics of the training dataset. Assuming the number of labeled boxes for class  $i$  is  $M_i$ , the maximal and minimal number are  $M_{max}$  and  $M_{min}$ , respectively. Therefore, the loss weight for class  $i$  is computed as

$$w_i = \begin{cases} \frac{W - 1}{r_{max}^\gamma - 1} \cdot (r_i^\gamma - 1) + 1, & i > 0 \text{ (foreground)} \\ 1, & i = 0 \text{ (background)} \end{cases}, \quad (5)$$

where  $r_i = \frac{M_{max}}{M_i}$ ,  $r_{max} = \frac{M_{max}}{M_{min}}$ , and  $\gamma$  and  $W$  are two hyper-parameters. Obviously, the loss weight for the most frequent class is 1 while the most rare class is  $W$ .

### 4.4. Balanced multi-task learning

As derived in Eq. 1, the overall loss is actually a weighted sum of different parts, but how to determine the optimal weighted coefficients is a challenge.

In [28], Kendall *et al.* proposed to form a multi-task learning paradigm by considering homoscedastic uncertainty to automatically train loss weights for different computer vision tasks. Inspired by this idea, we can also view the object detection training process as a multi-task learning problem—regression and classification tasks. The advanced version of final loss function<sup>3</sup> is approximated as

$$L = \frac{1}{2\sigma_1^2} L_{loc}^{arm} + \frac{\log \sigma_1^2}{2} + \frac{1}{\sigma_2^2} L_{cls}^{arm} + \frac{\log \sigma_2^2}{2} + \frac{1}{2\sigma_3^2} L_{loc}^{odm} + \frac{\log \sigma_3^2}{2} + \frac{1}{\sigma_4^2} L_{cls}^{odm} + \frac{\log \sigma_4^2}{2}, \quad (6)$$

where  $\{\sigma_1^2, \sigma_2^2, \sigma_3^2, \sigma_4^2\}$  are the trainable uncertainty parameters. In practice, for the steady of overall loss during training, the trainable parameters consist of network parameters  $\phi$  and logarithmic form of the four uncertainty parameters.

## 5. Experiments

In this section, we elaborately present the experimental results and ablation studies on the MSCOCO [39]

<sup>3</sup>For detailed derivation, please refer to [28].

dataset to demonstrate the effectiveness of our proposed RefineDetLite and training strategies.

### 5.1. Training and inference details

**Basic configuration.** To demonstrate the effectiveness of our network architecture, we set a basic configuration for fair comparisons with other state-of-the-art networks. All experiments are conducted on 4 Nvidia P40 GPUs with a batchsize of 128. We choose SGD with a weight decay of 0.0005 and momentum of 0.9 as our optimizer. We set the learning rate to  $4 \times 10^{-3}$  for the first 150 epochs, and decay it to  $4 \times 10^{-4}$  and  $4 \times 10^{-5}$  for training another 50 and 50 epochs, respectively. RefineDetLite72 is adopted as our backbone. For data augmentation, we strictly follow the instruction of SSD [40]. The loss functions for regression and classification are smooth L1 loss and cross-entropy loss, respectively. The weighted coefficients in Eq. 1 are all fixed to 1. We choose OHEM with a conventional positive-negative ratio of 1 : 3 as our training strategy. During inference, we post-process the network outputs by normal nms.

**Advanced configuration.** The advanced configuration is set to evaluate the training strategies proposed in Sec. 4 and improve the detection accuracy without apparent extra time-consumption as far as possible. GIOU loss [48] is chosen as the regression loss function. On the other hand, we adopt the classification loss functions and multi-task training strategies proposed in Sec. 4. Specifically, the hyper-parameters  $\alpha, \beta, \gamma$  and  $W$  are set to 0.25, 0.90, 3/4 and 10 experimentally. Correspondingly, the learning rate is adjusted as  $\{10^{-2}, 10^{-3}, 10^{-4}\}$  for the same epochs described in the basic configuration. Finally, soft-nms [11] is employed as the post-processing approach. The data augmentation and optimizer are kept the same as with the basic configuration. Controlled experiments are presented in Sec. 5.3 to evaluate the effectiveness of each component.

### 5.2. Results on MSCOCO

MSCOCO [39] is the most widely-used object detection evaluation dataset, which consists of 118,287 training samples (trainval35k), 5,000 validation samples (val5k) and 40,670 test samples (test-dev) in its 2017 version. We trained all networks on trainval35k and evaluated on test-dev for fair comparisons.

All models are trained on Pytorch [4] and inferred on Caffe2 converted by ONNX[3]. As discussed in the introduction, since network FLOPs are not proportional to the exact speed due to many factors, the network efficiency measured by FLOPs is not an accurate criterion. What people are really concerned with is the actual network running speed on CPU devices. Therefore, we re-implement some state-of-the-art algorithms and test their time efficiency (ms/pic) by averaging the total running time of 100 images on a single-thread Intel i7-6700@3.40GHz GPU for

Table 2. Detection result comparisons on MSCOCO online test-dev server, where time@CPU1 means running time tested by us based on open-source codes (All SSD and RefineDet-based networks [5], Tiny-DSOS [7], ThunderNet [6], Pelee [1], YOLOV3 [8]) on Intel i7-6700@3.40GHz and time@CPU2 means running time claimed by the original authors on different platforms. Bold fonts indicate our proposed algorithms. **RefineDetLite++** means the model trained on the *advanced configuration*.

Algorithms	input size	Backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>	time@CPU1	time@CPU2
<i>one-stage lightweight:</i>										
SSD [40]	300×300	MobileNet	19.3	-	-	-	-	-	128ms	-
SSDLite [49]	320×320	MobileNet	22.2	-	-	-	-	-	125ms	270ms (Pixel 1)
SSDLite [49]	320×320	MobileNetV2	22.1	-	-	-	-	-	120ms	200ms (Pixel 1)
Pelee [54]	304×304	PeleeNet	22.4	38.3	22.9	-	-	-	140ms	149ms (intel i7)
Tiny-DSOD [33]	300×300	DDB-Net+D-FPN	23.2	40.4	22.8	-	-	-	180ms	-
<i>RefineDet-based lightweight:</i>										
RefineDet [61]	320×320	MobileNet	24.3	43.0	24.4	7.6	26.5	38.7	168ms	-
RefineDet [61]	320×320	MobileNetV2	24.8	42.9	25.4	7.6	26.5	40.4	163ms	-
RefineDet [61]	320×320	MobileNetV3	22.1	40.1	23.3	6.5	23.6	35.7	150ms	-
RefineDet [61]	320×320	ShuffleNetV2	21.1	38.3	21.5	5.8	22.8	34.3	129ms	-
<i>two-stage lightweight:</i>										
FPNLite (@ 64) [17]	320×320	MobileNetV2	22.7	-	-	-	-	-	-	192ms (Pixel 1)
FPNLite (@ 128) [17]	320×320	MobileNetV2	24.3	-	-	-	-	-	-	264ms (Pixel 1)
NAS-FPNLite (3 @ 48) [17]	320×320	MobileNetV2	24.2	-	-	-	-	-	-	210ms (Pixel 1)
NAS-FPNLite (7 @ 64) [17]	320×320	MobileNetV2	25.7	-	-	-	-	-	-	285ms (Pixel 1)
ThunderNet [43]	320×320	SNet535	28.0	46.2	29.5	-	-	-	146ms	172ms (Snapdragon 845)
<i>one-stage classical:</i>										
SSD [40]	300×300	VGG	25.1	43.1	25.8	6.6	25.9	41.4	1250ms	-
SSD [40]	321×321	ResNet101	28.0	45.4	29.3	6.2	28.3	49.3	1000ms	-
YOLOV3 [46]	320×320	DarkNet53	28.2	51.5	-	-	-	-	1300ms	-
RefineDetLite	320×320	Res2NetLite72	<b>26.8</b>	<b>46.6</b>	<b>27.4</b>	<b>7.4</b>	<b>27.7</b>	<b>42.4</b>	<b>130ms</b>	-
RefineDetLite++	320×320	Res2NetLite72	<b>29.6</b>	<b>47.4</b>	<b>31.0</b>	<b>9.1</b>	<b>30.8</b>	<b>45.5</b>	<b>131ms</b>	-

fair comparisons. Additionally, we also show the speed claimed by the authors tested on different platforms as references. The detailed comparisons are listed in Table 2.

Without any bells and whistles, our proposed RefineDetLite trained on the basic configuration achieves 26.8 mAP at a speed of 130 ms/pic, which surpasses state-of-the-art *one-shot lightweight* detectors (SSD [40], SSDLite [49], Pelee [54] and Tiny-DSOD [33]). Specifically, RefineDetLite achieves SSDLite-MobileNet level speed with a significant 4.6 mAP improvement. In addition, it outperforms Pelee and Tiny-DSOD in both accuracy and speed.

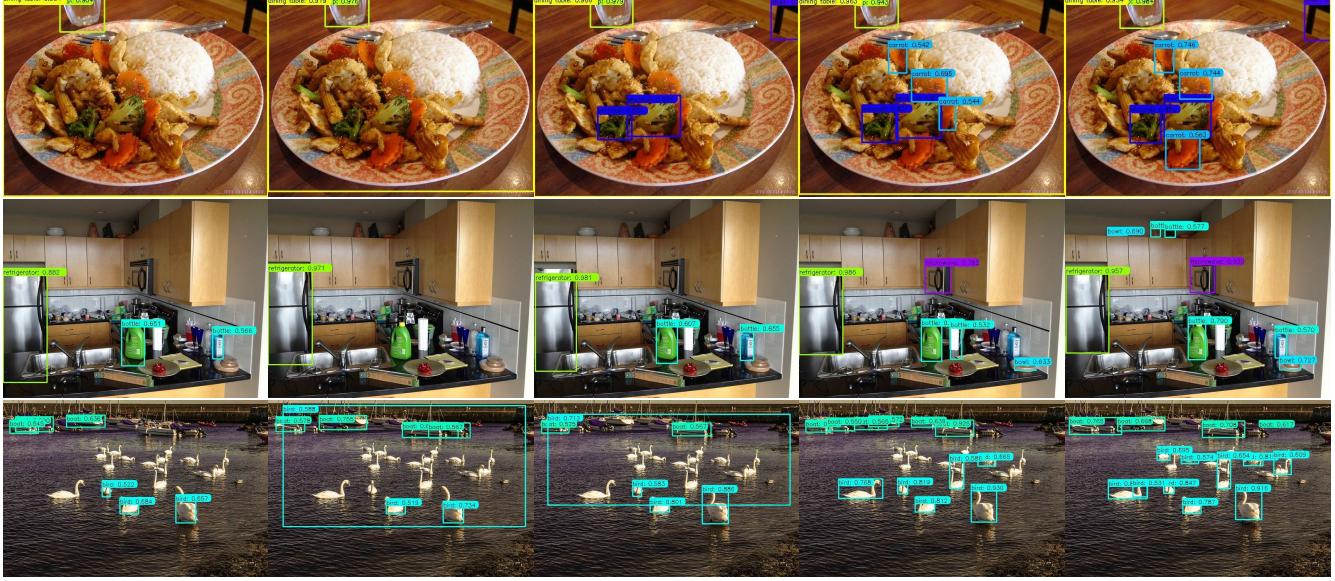
We also conducted comparison experiments on some state-of-the-art *two-stage lightweight* detection algorithms. The most competitive approach is the elaborately designed ThunderNet-SNet535 [43], which achieves 28.0 mAP (which surpasses our basic version 26.8, but is less than the advanced version 29.6). Since no official code was released by the authors, we did the speed comparison experiments based on a third-party re-implementation [6]. Besides this, NAS-FPNLites and FPNLites [17] are lightweight version of two-stage detectors based on NAS-FPN and FPN [37], respectively, but the accuracies are still lower than RefineDetLite. Since no open-released code can be referenced to do speed comparisons, we can only perform indirectly reasoning according to the speed of SSDLites and NAS-FPNLites both on a Pixel 1 CPU.

For further study of the effectiveness of our proposed backbone and light-head structures, we implemented Re-

fineDet with other lightweight backbones (MobileNets serials [25, 49, 24] and ShuffleNetV2 [41]). The statistics in Table 2 reveal that RefineDetLite surpasses all MobileNet-based RefineDet in both accuracy and efficiency. Additionally, at the same running speed level, it beats RefineDet-ShuffleNetV2 by a huge 5.7 mAP gain.

Finally, by adding training and inference strategies (soft-nms, GIOU and the three new strategies proposed in Sec. 4), we can further improve the RefineDetLite detection accuracy from 26.8 to 29.6 with little extra time consumption. One issue that deserves to be mentioned is that RefineDetLite can achieve similar detection accuracy level of some classical one-stage detectors (SSD-VGG, SSD-ResNet101 and YOLOV3) with an 8–10 times speed increase.

Additionally, we visualize the comparison results among different models—SSDLite+MobileNet, RefineDet+ShuffleNetV2, SSD+VGG, RefineDetLite and RefineDetLite++ as in Fig. 4. The visualization results clearly show that the RefineDetLite can detect much more small objects (*e.g.*, carrots, bottles and birds) compared with SSDLite+MobileNet and RefineDet+ShuffleNetV2 at almost the same running speed (130 ms/pic). It also outperforms SSD+VGG with a considerable 10 times of speed increase. Furthermore, by adding the proposed training strategies, the detection accuracy and recall of RefineDetLite++ surpasses RefineDetLite significantly.



(a) SSDLite+MobileNet (b) RefineDet+ShuffleNetV2 (c) SSD+VGG (d) RefineDetLite (e) RefineDetLite++  
Figure 4. Visualization comparisons among different models.

Table 3. Ablation studies on proposed detection modules and training strategies. Bold fonts indicate our proposed modules.

models	Res2Block	Light-head	soft-nms	GIOU	classes weights	IOU guided	Multi-task	AP	AP <sub>50</sub>	AP <sub>75</sub>	time@CPU1
(a) Baseline								24.7	43.0	25.1	150ms
(b)	✓							25.7	43.9	26.7	140ms
<b>(c) RefineDetLite</b>	✓	✓						<b>26.8</b>	<b>46.6</b>	<b>27.4</b>	<b>130ms</b>
(d)	✓	✓	✓					27.2	46.0	28.4	131ms
(e)	✓	✓	✓	✓				27.4	46.2	28.8	131ms
(f)	✓	✓	✓	✓	✓			27.5	46.2	28.9	131ms
(g)	✓	✓	✓	✓				27.7	46.5	29.0	131ms
(h)	✓	✓	✓	✓	✓			28.2	46.4	29.4	131ms
(i)	✓	✓	✓	✓	✓	✓		28.5	46.8	29.8	131ms
(j)	✓	✓	✓	✓	✓			28.6	46.8	29.9	131ms
(k)	✓	✓	✓	✓	✓			28.7	47.0	30.2	131ms
(l)	✓	✓	✓	✓	✓	✓		29.0	47.3	30.1	131ms
(m)	✓	✓	✓	✓	✓		✓	29.3	47.4	30.5	131ms
(n)	✓	✓	✓	✓	✓		✓	29.2	47.4	30.4	131ms
<b>(o) RefineDetLite++</b>	✓	✓	✓	✓	✓	✓	✓	<b>29.6</b>	<b>47.4</b>	<b>31.0</b>	<b>131ms</b>

### 5.3. Ablation study

**Network architectures.** First, we evaluate the effectiveness the two key components (*Res2Block* and *Light-head*) of our proposed RefineDetLite architectures. We set up a baseline architecture, just simply replacing all Res2Blocks in the backbone Res2NetLite by the classical ResBlocks [22] and replacing all Light-head modules in the detection part and feature fusion by simple  $3 \times 3$  convolutions. Experimental results in rows (a)–(c) of Table 3 show that the two components achieve 1.0 and 1.1 mAP improvements, separately. Simultaneously, they save the running time 10ms and 10ms, separately. Therefore, we can draw the conclusion that Res2Block and Light-head make great contributions to improving both the accuracy and efficiency of the detector.

**Training and inference strategies.** Based on the RefineDetLite architecture trained on the basic configuration,

we further conducted more controlled experiments to evaluate the effectiveness of each training and inference strategy, including the three strategies proposed in Sec. 4—classes-weighted loss, IOU-guided loss and multi-task training and two orthogonal approaches proposed by other researchers: soft-nms and GIOU loss.

Rows (d) and (h) in Table 3 demonstrate the consistent improvement of the well-known soft-nms and GIOU loss, which will be exploited as standard components for future experiments. Rows (e)–(g) reveal that the three proposed strategies—classes-weighted loss, IOU guided loss and multi-task training achieve 0.2, 0.3 and 0.5 mAP gains, respectively, after using soft-nms. Furthermore, after employing GIOU loss, the three components still achieve 0.3, 0.4 and 0.5 mAP improvements (rows (i)–(k)).

Statistics in rows (l)–(o) demonstrate the orthogonality of the three proposed strategies. Experiments show steady accuracy increasing after combinations of any components.

Finally, when integrating all strategies together, we can achieve a 29.6 mAP, which is a state-of-the-art detection accuracy at the 150 ms/pic running speed level.

## 6. Conclusion

In this paper, we proposed a lightweight efficient and accurate one-shot object detection framework. After analyzing the gaps between classification networks and detection backbones, and following the design principles of efficient networks, we designed a residual-like lightweight backbone—Res2NetLite, enlarging the receptive fields and feature dimensions of early stages, together with a corresponding light-head detection parts. In addition, we investigate the weaknesses of current one-stage training process, hence develop three improved training strategies: IOU-guided loss, classes-weighted loss and balanced multi-task training method. Without any bells and whistles, our proposed RefineDetLite achieves 26.8 mAP on MSCOCO test-dev at a speed of 130 ms/pic. By adding the orthogonal training and inference strategies, the advanced version—RefineDetLite++ can further achieve 29.6 mAP with little extra time consumption.

## References

- [1] Official Pelee. <https://github.com/Robert-JunWang/Pelee>. Accessed Nov. 12, 2019.
- [2] Official YOLO serials. <https://pjreddie.com/darknet/yolo/>. Accessed Nov. 12, 2019.
- [3] ONNX. <https://github.com/onnx/onnx>. Accessed Nov. 12, 2019.
- [4] Pytorch. <https://github.com/pytorch/pytorch>. Accessed Nov. 12, 2019.
- [5] Pytorch SSD. <https://github.com/lzx1413/PytorchSSD>. Accessed Nov. 12, 2019.
- [6] Pytorch ThunderNet. [https://github.com/ouyanghuiyu/Thundernet\\_Pytorch](https://github.com/ouyanghuiyu/Thundernet_Pytorch). Accessed Nov. 12, 2019.
- [7] Pytorch Tiny-SSD. [https://github.com/ArcherV/my\\_tiny](https://github.com/ArcherV/my_tiny). Accessed Nov. 12, 2019.
- [8] Pytorch YOLOV3. <https://github.com/eriklindernoren/PyTorch-YOLOv3/>. Accessed Nov. 12, 2019.
- [9] Peter Anderson, Xiaodong He, Chris Buehler, Damien Teney, Mark Johnson, Stephen Gould, and Lei Zhang. Bottom-up and top-down attention for image captioning and visual question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6077–6086, 2018.
- [10] Sean Bell, C Lawrence Zitnick, Kavita Bala, and Ross Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2874–2883, 2016.
- [11] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Soft-NMS—improving object detection with one line of code. In *Proceedings of the IEEE international conference on computer vision*, pages 5561–5569, 2017.
- [12] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *european conference on computer vision*, pages 354–370. Springer, 2016.
- [13] Chen Chen, Shuai Mu, Wanpeng Xiao, Zexiong Ye, Liesi Wu, and Qi Ju. Improving image captioning with conditional generative adversarial nets. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8142–8150, 2019.
- [14] Zhe Chen, Shaoli Huang, and Dacheng Tao. Context refinement for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 71–86, 2018.
- [15] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. DSSD: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.
- [16] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. Res2Net: A new multi-scale backbone architecture. *arXiv preprint arXiv:1904.01169*, 2019.
- [17] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. NAS-FPN: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7036–7045, 2019.
- [18] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-aware neural network design. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1638–1647, 2018.
- [19] Ross Girshick. Fast R-CNN. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.
- [20] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [21] Eran Goldman, Roei Herzig, Aviv Eisenshtat, Jacob Goldberger, and Tal Hassner. Precise detection in densely packed scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5227–5236, 2019.
- [22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [23] Yihui He, Chenchen Zhu, Jianren Wang, Marios Savvides, and Xiangyu Zhang. Bounding box regression with uncertainty for accurate object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2888–2897, 2019.
- [24] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu,

- Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. *arXiv preprint arXiv:1905.02244*, 2019.
- [25] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.
- [26] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3588–3597, 2018.
- [27] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.
- [28] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7482–7491, 2018.
- [29] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, and Jianbo Shi. Consistent optimization for single-shot object detection. *arXiv preprint arXiv:1901.06563*, 2019.
- [30] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 845–853, 2016.
- [31] Buyu Li, Yu Liu, and Xiaogang Wang. Gradient harmonized single-stage detector. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 8577–8584, 2019.
- [32] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-DSOD: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018.
- [33] Yuxi Li, Jiuwei Li, Weiyao Lin, and Jianguo Li. Tiny-DSOD: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018.
- [34] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-head R-CNN: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.
- [35] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Detnet: A backbone network for object detection. *arXiv preprint arXiv:1804.06215*, 2018.
- [36] Zuoxin Li and Fuqiang Zhou. FSSD: feature fusion single shot multibox detector. *arXiv preprint arXiv:1712.00960*, 2017.
- [37] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.
- [38] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.
- [39] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.
- [40] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.
- [41] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.
- [42] Mahyar Najibi, Mohammad Rastegari, and Larry S Davis. G-cnn: an iterative grid based object detector. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2369–2377, 2016.
- [43] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. Thundernet: Towards real-time generic object detection on mobile devices. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6718–6727, 2019.
- [44] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [45] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.
- [46] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.
- [47] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.
- [48] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 658–666, 2019.
- [49] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.
- [50] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. DSOD: Learning deeply supervised object detectors from scratch. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1919–1927, 2017.
- [51] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 761–769, 2016.
- [52] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modu-

- lation for object detection. *arXiv preprint arXiv:1612.06851*, 2016.
- [53] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [54] Robert J Wang, Xiang Li, and Charles X Ling. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*, pages 1963–1972, 2018.
- [55] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2606–2615, 2017.
- [56] Alexander Womg, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 95–101. IEEE, 2018.
- [57] Alexander Womg, Mohammad Javad Shafiee, Francis Li, and Brendan Chwyl. Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection. In *2018 15th Conference on Computer and Robot Vision (CRV)*, pages 95–101. IEEE, 2018.
- [58] Bichen Wu, Forrest Iandola, Peter H Jin, and Kurt Keutzer. Squeezedet: Unified, small, low power fully convolutional neural networks for real-time object detection for autonomous driving. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 129–137, 2017.
- [59] Fengwei Yu, Wenbo Li, Quanquan Li, Yu Liu, Xiaohua Shi, and Junjie Yan. Poi: Multiple object tracking with high performance detection and appearance feature. In *European Conference on Computer Vision*, pages 36–42. Springer, 2016.
- [60] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. Unitbox: An advanced object detection network. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 516–520, 2016.
- [61] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4203–4212, 2018.
- [62] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.
- [63] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression. *arXiv preprint arXiv:1911.08287*, 2019.

# Appendices

## A. Example of classes-weighted loss

Here we show the example of classes-aware weights for MSCOCO [39] trainval35k dataset. Fig. 5 illustrates the curves of weights VS number of labeled boxes for all 80 COCO classes. Specifically, for the most frequent class—*person* which contains 273468 labeled boxes, the weight is 1. While for the most rare class—*hair drier* which contains only 209 labeled boxes, the weight is 10. We also test other hyper-parameters ( $\gamma$  and  $W$ ) in Eq. 5 of original paper, but finally we chose  $\gamma = 3/4$  and  $W = 10$  experimentally based on the MSCOCO dataset.

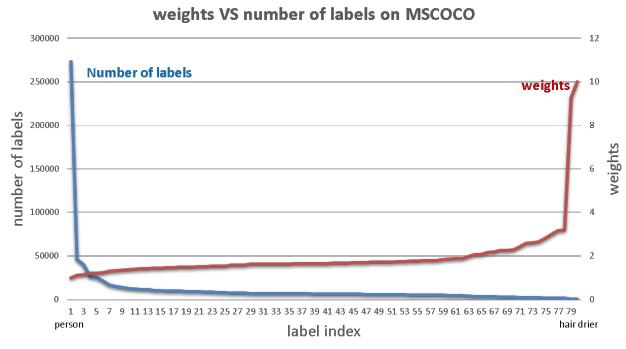


Figure 5. Number of labels VS weights on MSCOCO trainval35k dataset.

## B. Derivation of balanced multi-task training

Without loss of generality, we derive only the object detection loss  $L^{odm}$  and omit the sample index subscript  $j$ . Following the guidance of [28], minimizing the loss equals to minimizing the negative log likelihood as Eq. 7.

In Eq. 7,  $\mathbf{g}, \mathbf{y}$  are the ground-truth bounding boxes and class labels, respectively,  $\mathbf{z}$  is the input image,  $\mathbf{f}^\phi$  is the network with parameters  $\phi$  and  $\{\sigma_3, \sigma_4\}$  are the trainable uncertainty parameters. Based on the theorem of [28], we assume that the regression and classification are two independent optimization processes and the final outputs are two probability distributions with random variables  $\sigma_3^2$  and  $\sigma_4^2$ .

Similarly, based on the derivations of Eq. 7 and 8, we can also derive the loss  $L^{arm}$  for anchor refinement module (ARM). Therefore, the final overall loss for balanced multi-task training is as Eq. 6 in original paper. In practice, for the stable when optimizing the overall loss, the trainable parameters consist of network parameters  $\phi$  and logarithmic form of the four uncertainty parameters, namely  $\{\log \sigma_1^2, \log \sigma_2^2, \log \sigma_3^2, \log \sigma_4^2\}$ .

$$\begin{aligned}
L^{odm} &= -\log P(\mathbf{g}, \mathbf{y} | \mathbf{f}^\phi(\mathbf{z})) \\
&= -\log [P(\mathbf{g} | \mathbf{f}^\phi(\mathbf{z})) \cdot P(\mathbf{y} | \mathbf{f}^\phi(\mathbf{z}))] \quad (\text{suppose the independence of location and classification}) \\
&= -\log N(\mathbf{g}; \mathbf{f}^\phi(\mathbf{z}), \sigma_{loc}^2) + D_{KL}\left(\mathbf{y} \parallel \frac{\mathbf{f}^\phi(\mathbf{z})}{\sigma_{cls}^2}\right) \quad (\text{probability distributions with variances } \sigma_3^2 \text{ and } \sigma_4^2) \\
&= \frac{1}{2\sigma_3^2} \|\mathbf{g} - \mathbf{f}^\phi(\mathbf{z})\| + \log \sigma_3 + \frac{1}{2} \log 2\pi + \sum_i y_i \log y_i - \sum_i y_i \log \text{softmax}\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right) \\
&\propto \frac{1}{2\sigma_3^2} \|\mathbf{g} - \mathbf{f}^\phi(\mathbf{z})\| + \log \sigma_3 - \sum_i y_i \log \text{softmax}\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right) \quad (\frac{1}{2} \log 2\pi + \sum_i y_i \log y_i \text{ is nontrainable}) \\
&\approx \frac{1}{2\sigma_3^2} L_{loc}^{odm} + \frac{\log \sigma_3^2}{2} - \sum_i y_i \log \text{softmax}\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right) \quad (L_{loc}^{odm} \text{ is approximated as L2 norm loss}) \\
&\approx \frac{1}{2\sigma_3^2} L_{loc}^{odm} + \frac{\log \sigma_3^2}{2} + \frac{1}{\sigma_4^2} L_{cls}^{odm} + \frac{\log \sigma_4^2}{2}.
\end{aligned} \tag{7}$$

$$\begin{aligned}
-\sum_i y_i \log \text{softmax}\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right) &= -\sum_i y_i \log \frac{\exp\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right)}{\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)} \\
&= -\frac{1}{\sigma_4^2} \sum_i y_i \log \left[ \frac{\exp\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right)}{\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)} \cdot \frac{\exp\left(\mathbf{f}_c^\phi(\mathbf{z})\right)}{\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)} \right] \\
&= -\frac{1}{\sigma_4^2} \sum_i y_i \left[ \log \frac{\exp\left(\frac{\mathbf{f}_i^\phi(\mathbf{z})}{\sigma_4^2}\right)}{\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)} \right] + \sum_i y_i \log \left[ \frac{\exp\left(\mathbf{f}_c^\phi(\mathbf{z})\right)}{\left(\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)\right)^{\frac{1}{\sigma_4^2}}} \right] \\
&\approx \frac{1}{\sigma_4^2} \left[ \sum_i y_i \log \text{softmax}\left(\mathbf{f}_i^\phi(\mathbf{z})\right) \right] + \log \left[ \frac{\exp\left(\mathbf{f}_c^\phi(\mathbf{z})\right)}{\left(\sum_c \exp\left(\frac{\mathbf{f}_c^\phi(\mathbf{z})}{\sigma_4^2}\right)\right)^{\frac{1}{\sigma_4^2}}} \right] \\
&\approx \frac{1}{\sigma_4^2} L_{cls}^{odm} + \log \sigma_4 \quad (L_{cls}^{odm} \text{ is approximated as cross entropy loss and refer to [28]}) \\
\end{aligned} \tag{8}$$

## C. More visualizations

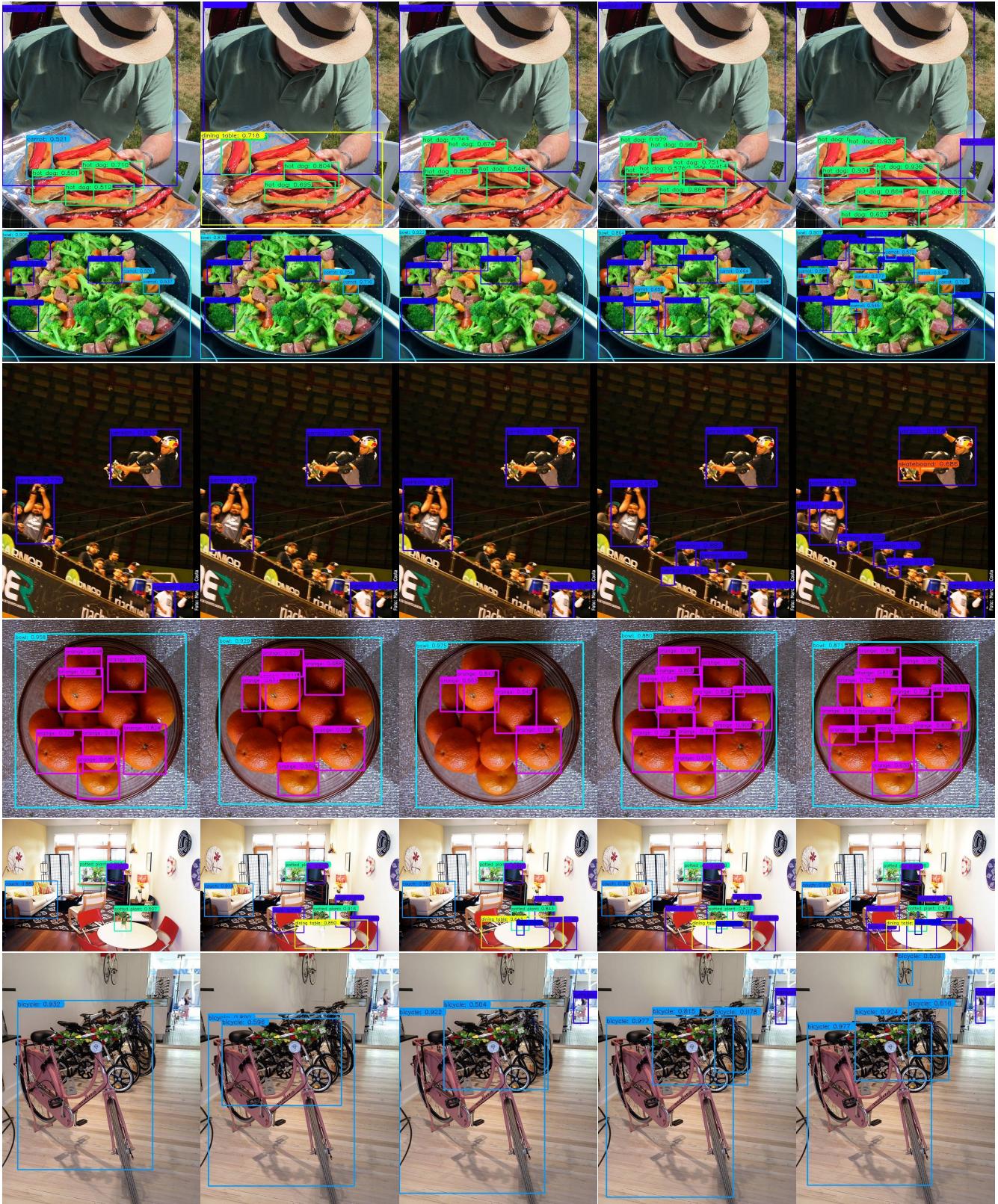
In this section, we provide more visualization cases (Fig. 6–8) of comparisons among different object detection algorithms, including SSDLite+MobileNet, SSD+VGG, RefineDet+ShuffleNetV2 and our proposed RefineDetLite and RefineDetLite++.

First, as demonstrated in Fig. 6, due to the strong refining process and feature representation ability, our proposed RefineDetLite can detect more small objects (e.g., *hot dogs*, *carrots*, *persons*, *oranges*, *chairs* and *bicycles* from the first to the last row in turn). Furthermore, after adding the proposed training strategies, RefineDetLite++ can achieve better accuracy and recall.

Additionally, as shown in Fig. 7, RefineDetLite can also detect more kinds of objects, especially small objects, such as *skateboards*, *chairs*, *skis*, *fir hydrants*, *bottles*, *traffic lights* and *apples*. RefineDetLite can further refine the detection objects by revising some errors such as *sports ball* in the first row and *bird* in the last row.

Thanks to the IOU-guided loss and balanced multi-task training strategies, as compared to RefineDetLite, RefineDetLite++ can locate the objects more accurately. Fig. 8 illustrated the comparisons in detail. The visualization results clearly show that RefineDetLite++ can achieve higher location accuracy such as *motorcycle*, *umbrella*, *cake*, *car*, *train*, *bus*, *horse* and *dog* in each row or images. Therefore, the  $AP_{75}$  value of RefineDetLite++ (31.0) is much higher

than the other lightweight detectors.



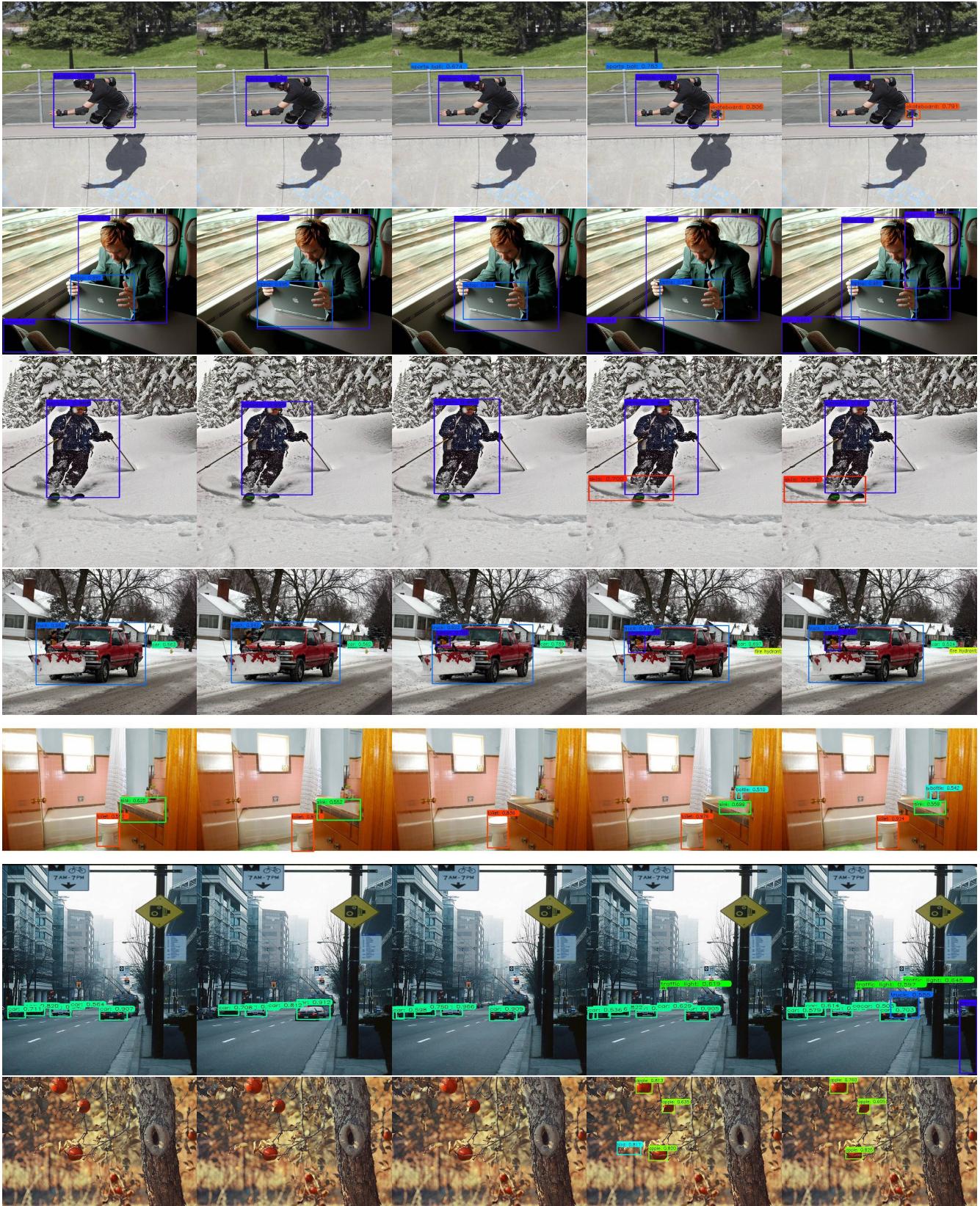
(a) SSDLite+MobileNet

(c) SSD+VGG

(d) RefineDetLite

(e) RefineDetLite++

Figure 6. Visualization comparisons among different models. This groups of images show that RefinDetLite can detect more small objects. Furthermore, RefinDetLite++ can refine the results more accurately. Best viewed by zooming in.



(a) SSDLite+MobileNet (b) RefineDet+ShuffleNetV2

(c) SSD+VGG

(d) RefineDetLite

(e) RefineDetLite++

Figure 7. Visualization comparisons among different models. This groups of images show that RefinDetLite can detect more kinds of objects, especially small objects. Furthermore, RefinDetLite++ can refine the results more accurately. Best viewed by zooming in.



(a) SSDLite+MobileNet

(b) RefineDet+ShuffleNetV2

(c) SSD+VGG

(d) RefineDetLite

(e) RefinDetLite++

Figure 8. Visualization comparisons among different models. This groups of images show that RefinDetLite++ can localize the prediction bounding boxes more accurately as compared to RefineDetLite. Best viewed by zooming in.