MAOD: An Efficient Anchor-Free Object Detector Based on MobileDet

DONG CHEN AND HAO SHEN

Communication Strategy of China Co-Innovation Center, Communication University of China, Beijing 100024, China
Corresponding author: Dong Chen (haohaoxueilema@163.com)

ABSTRACT For real-time object detectors, accuracy and efficiency are two important considerations. In this paper, we propose a lightweight anchor-free detector, MAOD, to better balance efficiency and accuracy. Our object detector contains three components: an efficient backbone network (MobileDet), a lightweight feature pyramid structure (L-FPN) and an anchor-free per-pixel prediction method. MobileDet and L-FPN provide more accurate and faster multi-scale feature extraction. Our anchor-free per-pixel prediction method achieves efficient classification and location regression tasks. On the benchmark MS-COCO dataset, MAOD achieves 46.1% AP at the speed of 68 FPS with the input size 512 × 512. When the input size is 800 × 800, MAOD achieves 47.1% AP at the speed of 43 FPS. The fast version of MAOD (320 × 320 input size) can run at 91 FPS with 43.3% AP. Compared with other state-of-the-art object detectors, our detector has similar accuracy while maintaining extremely fast inference speed. MAOD achieves an optimal efficiency-accuracy tradeoff.

INDEX TERMS Lightweight real-time detector, anchor-free object detection, MobileDet backbone, lightweight feature pyramid.
MAOD is a smaller, faster, and more robust real-time object detector.

II. RELATED WORK
A. ANCHOR-FREE DETECTORS

Recently, anchor-free designs become increasingly popular in object detection, and achieve better performance than anchor-based approaches by removing anchor boxes. The design ideas of anchor-free detectors generally fall into two categories, namely, key point and anchor point [6].

Anchor-point approaches have higher efficiency and simpler network architecture than keypoint-based detectors. YOLOv1 [7] uses an $S \times S$ grid on each image to predict objects and their bounding boxes. However, YOLOv1 only uses the center of the object as a positive sample, so its accuracy is very low. Unlike YOLOv1, FoveaBox [8] introduces FPN [9] and adopts a larger central area as a positive sample. As a result, FoveaBox obtains good results. Since all pixels in the ground-truth box are positive samples, FCOS can apply more samples to train and achieve better accuracy. FSAF [10] adds an anchor-free branch to each feature level of RetinaNet [11], so that each object can be assigned to the best feature levels. However, FSAF does not completely abandon anchor-based design ideas. SAPD [6] improves FSAF with two new optimized designs: soft-selected pyramid levels and soft-weighted anchor points.

Keypoint-based detector converts the object detection task into a keypoint detection problem. By detecting the center or corner of the bounding box, they obtain better accuracy [1]. CornerNet [12] achieves object detection by detecting two corner points of the bounding box, but speed is its main drawback [13]. CenterNet [14] adopts the idea of center point detection. The object center is defined as the local peak point in the heat map. Moreover, CenterNet does not require post-processing operations (such as NMS), so it is more efficient. Another CenterNet [15] uses both corner and center points of the bounding box as key points.

B. ACCURACY OR SPEED

Achieving higher accuracy is the primary goal of object detection at present. Many two-stage detectors have remarkable accuracy, including TridentNet [16], PANet [17], RPDet [18], Libra R-CNN [19], SNIP [20]. However, the improvement in accuracy brings higher computational costs. Model efficiency is also important. In order to increase model speed, a growing number of lightweight models have appeared. YOLO-Lite [21] and YOLO-Nano [22] achieve extremely fast inference speed by using a lightweight backbone network. ThunderNet [23] employs a two-stage architecture to achieve fast detection on the ARM platform and obtains a good result. Since small backbone networks have limited representation power, lightweight detectors usually do not have good accuracy. In this work, we want to strike a better balance between efficiency and accuracy.

III. NETWORK ARCHITECTURE

In this section, we introduce the detail network architecture of MAOD, as shown in Fig. 1. MAOD contains 3 components: a computation-efficient MobileDet network, a lightweight
TABLE 1. MobileDet architecture.

| Input | Kernel | Filter | Exp | LSC | NL | S | DR | HS |
|-------|--------|--------|-----|-----|----|---|----|----|
| 224×224 | dilated bottleneck, 3×3 | 16 | 16 | y | h-swish | 1 | 3 | D1 |
| 224×224 | bottleneck, 3×3 | 16 | 16 | n | relu-6 | 1 | - | B1 |
| 224×224 | bottleneck, 3×3 | 24 | 64 | n | relu-6 | 2 | - | B2 |
| 112×112 | bottleneck, 3×3 | 24 | 72 | n | relu-6 | 1 | - | B3 |
| 112×112 | bottleneck, 5×5 | 40 | 72 | y | relu-6 | 2 | - | B4 |
| 56×56 | bottleneck, 5×5 | 40 | 120 | y | relu-6 | 1 | - | B5 |
| 56×56 | bottleneck, 5×5 | 40 | 120 | y | relu-6 | 1 | - | B6 |
| 56×56 | bottleneck, 3×3 | 80 | 240 | n | h-swish | 2 | - | B7 |
| 28×28 | bottleneck, 3×3 | 80 | 200 | n | h-swish | 1 | - | B8 |
| 28×28 | bottleneck, 3×3 | 80 | 184 | n | h-swish | 1 | - | B9 |
| 28×28 | bottleneck, 3×3 | 80 | 184 | n | h-swish | 1 | - | B10 |
| 28×28 | bottleneck, 3×3 | 112 | 480 | y | h-swish | 1 | - | B11 |
| 28×28 | bottleneck, 3×3 | 112 | 672 | y | h-swish | 1 | - | B12 |
| 28×28 | bottleneck, 3×5 | 160 | 672 | y | h-swish | 2 | - | B13 |
| 14×14 | dilated bottleneck, 3×3 | 160 | 672 | y | h-swish | 1 | 3 | D2 |
| 14×14 | dilated bottleneck, 3×3 | 160 | 960 | y | h-swish | 1 | 3 | D3 |
| 14×14 | dilated bottleneck, 3×3 | 160 | 960 | y | h-swish | 1 | 4 | D4 |
| 14×14 | dilated bottleneck, 3×3 | 160 | 960 | y | h-swish | 1 | 4 | D5 |
| 14×14 | bottleneck, 3×5 | 1280 | 960 | y | h-swish | 1 | - | B14 |
| 14×14 | GlobalAvgPool | - | - | - | - | - | - | A1 |
| 1×1 Conv | Exp | - | - | - | - | - | - | C1 |

Exp denotes the expansion size, NL denotes the type of nonlinearity used. S denotes stride. LSC denotes whether there is an attention module in that block. Here, LSC is our attention module. See Fig. 2 for different residual blocks. DR denotes the dilation rate.

A. BACKBONE

An excellent backbone network is the foundation of object detection. We use MobileNetV3 [24] as a baseline, which is an extremely computation-efficient CNN model. However, compared with other deep neural networks such as ResNet101 [25], ResNext [26] and Inception V4 [27], MobileNetV3 has lower accuracy. We want to improve its accuracy by slightly increasing the computational cost. Inspired by DetNet [28] and CBAM [29], we design MobileDet backbone (see Table 1 and Fig. 2). Our MobileDet is improved from MobileNetV3-Large and has 58 convolutional layers.

1) DILATED BOTTLENECK

In order to obtain a larger receptive field, we use the bottleneck with depthwise dilated convolution as a basic network unit in our backbone [28]. Its basic implementation structure is illustrated in Fig. 2(a). Unlike DetNet, we use depthwise dilated convolution instead of standard dilated convolution to reduce computational complexity. Our dilated bottleneck adopts the linear inverted residual bottleneck [24]. This structure is helpful to reduce the memory requirement, while improving the representation power of nonlinear per-channel transformations [24]. Since element-wise operation is time consuming, we replace the element-wise addition with channel concatenation operation to reduce computational and time cost [30]. In addition, we apply hard-swish [24] and our attention module (LSC) in each bottleneck.

Since dilated convolution takes a considerable amount of time [28], we only apply 5 dilated bottlenecks: D1, D2, D3, D4 and D5 (see Table 1). The first block, D1, is used to replace the first convolutional layer of MobileNetV3. To enlarge the receptive field, we set the dilation rate to not less than 3. Then, we introduce our 2 dilated bottlenecks (with 3×3 kernels) as an efficient replacement for the last two bottlenecks (with 5×5 kernels) of MobileNetV3. This can further increase the valid receptive field while maintaining low complexity. Finally, we add D4 and D5 after D3 to deepen the network. For the other bottlenecks in our backbone, we keep their structure the same as the original MobileNetV3 (the expansion rate of B14 is 0.75), see Fig. 2(b).

2) ATTENTION MODULE

Attention mechanism is an effective tool to strengthen the representational power of network. MobileNetV3 introduces squeeze-and-excitation (SE) unit [31] to improve accuracy. However, SE only focuses on the relationship between channels, missing the spatial features. Spatial information is also important. Inspired by CBAM [29], a better attention module than SE, we propose an efficient attention module to replace SE.

A Light Spatial-Channel (LSC) attention module is a lighter and faster computational unit than CBAM. Our module learns the weight distribution on the feature map by focusing on both spatial and channel information. Similar to CBAM, LSC consists of a spatial sub-module and a channel sub-module (see Fig. 3). For any given feature map $x \in \mathbb{R}^{H \times W \times C}$, we can obtain a spatial feature tensor $y_s = f_s(x) \in \mathbb{R}^{H \times W \times 1}$ and a channel feature tensor $y_c = g_c(x) \in \mathbb{R}^{1 \times 1 \times C}$ by using LSC.

Different from CBAM, our attention module adopts a spatial-first order. Compared with spatial information,
channel information has a greater impact on model accuracy. If we use a channel-first order, the spatial attention module may suppress the channel-refined feature. The overall process can be expressed as:

\[
\begin{align*}
y_s &= f_s(x), \quad y_c = g_c(x) \\
Y_S &= x \otimes f_s(x) = x \otimes y_s \\
Y_{SC} &= Y_S \otimes g_c(Y_S) = Y_S \otimes y_c
\end{align*}
\]

Here \(\otimes\) denotes element-wise multiplication. \(Y_S\) denotes the spatial-refined feature and \(Y_{SC}\) refers to the final output.

\[a:\text{ SPATIAL ATTENTION}\]

Since spatial attention only focuses on spatial features, we first compress channel information by max-pooling and average-pooling operations, and obtain two spatial feature tensors:

\[
x_{\text{max}} \in R^{H \times W \times 1} \quad \text{and} \quad x_{\text{avg}} \in R^{H \times W \times 1}
\]

Then, these tensors are concatenated together to produce a 3-dimensional tensor \(t_c\). Finally, we apply convolution operations for this tensor \(t_c\). To reduce parameters and computation, we use a 1 \(\times\) 7 convolution followed by a 7 \(\times\) 1 convolution as an efficient alternative to the 7 \(\times\) 7 convolution of CBAM [32]. Meanwhile, we replace the sigmoid function with hard-sigmoid to increase the computational efficiency [24]. This process can be expressed as:

\[
y_s = \sigma(\text{Conv}_1 \times 7(\sigma(\text{Conv}_7 \times 1(\text{MaxPool}_c(x) + \text{AvgPool}_c(x))))) \\
y_s = \sigma(\sigma(x_{\text{max}} + x_{\text{avg}})_{7 \times 7} 1 \times 7 = \sigma(\sigma(t_c)_{7 \times 7} 1 \times 7.
\]

Here \(\sigma\) denotes the hard-sigmoid function and \(+\) refers to concatenation. The spatial attention sub-module is illustrated in Fig. 4.

\[b:\text{ CHANNEL ATTENTION}\]

Our channel attention module is a simplified SE unit. Both average-pooling and max-pooling operations are used to squeeze spatial features. Since the ReLU function only has a linear transformation for non-negative inputs, we replace it with the hard-sigmoid. The process of channel attention can be expressed as:

\[
y_c = \sigma(\text{MaxPool}_c(x)) \oplus \sigma(\text{AvgPool}_c(x)) \\
y_c = \sigma(x_{\text{max}}) \oplus \sigma(x_{\text{avg}}).
\]

Here \(\sigma\) denotes the hard-sigmoid function and \(\oplus\) refers to element-wise addition. The channel attention sub-module is shown in Fig. 5.

**B. LIGHTWEIGHT FEATURE PYRAMID NETWORK**

L-FPN is an efficient multi-scale feature pyramid structure which is built on the MobileDet backbone network (see Fig. 6). In order to obtain faster speed and better accuracy, we redesign FPN [9], and introduce depthwise separable convolution and our attention module. L-FPN contains 3 parts: Base feature, Feature pyramid and Prediction module.

1) **BASE FEATURE**

By aggregating deep and shallow features of the backbone, lightweight FPN generates the base feature. First, we obtain different scale features from multiple bottleneck blocks of MobileDet, which are B6, B12, D3 and D5 (see Table 1) respectively. For B6, B12, D3 and D5, we define the outputs of these bottlenecks as \(\{C_1, C_2, C_3, C_4\}\), and they have strides of \(\{4, 8, 16, 16\}\) pixels with respect to the input image. Then, these features are fused to produce the base feature through channel concatenation.

2) **FEATURE PYRAMID**

Our feature pyramid is responsible for providing feature maps of different scales for the final prediction. Based on the base feature, we use convolution and upsampling operations to construct this feature pyramid. As shown in Fig. 6, our feature pyramid has 4 feature levels: F1, F2, F3 and F4. These feature levels have strides 8, 16, 32 and 64, respectively. F4 is obtained by applying an upsampling layer and a 1 \(\times\) 1 convolutional layer. To get richer semantic information, F4 fuses shallow features (C2) after the upsampling operation. We use depthwise convolutions and 1 \(\times\) 1 convolutions to produce other feature levels (F1, F2 and F3). Meanwhile, we employ hard-swish function as non-linear activation after each convolution. In addition, we use channel concatenation operation to fuse each feature level with C4 separately. However, simple channel concatenation operations are not adaptive enough [31]. LSC is introduced to improve the detection performance of L-FPN.

3) **PREDICTION MODULE**

On top of each feature level of L-FPN, we adopt a prediction module to achieve classification and regression tasks. L-FPN shares these prediction modules between different levels, which helps to reduce computation. We apply the linear inverted residual bottleneck [24] in each module. At the detection stage, we add two 1 \(\times\) 1 convolution layers after the bottleneck respectively for classification and regression branches. Moreover, both the classification and regression branches have two sub-branches. We discuss this in detail in Section III-C.

**C. ANCHOR-FREE PER-PIXEL PREDICTION**

Our method is improved from FCOS to achieve more accurate detection tasks. FCOS is a simple and effective anchor-free detection method. By removing the anchor boxes, FCOS uses each pixel on the feature map to directly predict the object and
Its location. However, this method suffers some drawbacks: 1) Ambiguous Samples: Since each pixel can only predict one object, it is difficult for FCOS to detect multiple overlapping objects on each feature level. 2) The large stride (128x) of the final feature maps is harmful for detecting small objects. Moreover, as the anchor-free detectors do not use predefined anchor boxes, the feature maps with 128x downsampling cannot effectively detect large objects.

To improve its performance, we make the following improvements:

First, each pixel on the feature map is responsible for predicting multiple objects (see Fig. 6).

Second, we add an extra confidence sub-branch to the classification branch to achieve foreground and background classification. By eliminating background pixels, the inference time is reduced.

Third, we replace Focal Loss with DR Loss [34] to better alleviate the class-imbalance problem.

Fourth, we adopt Distance-IoU Loss [35] as an efficient replacement for IoU Loss, which helps to improve the performance of the regressor.

Finally, in order to better detect small objects, we removed 128x downsampling in the feature pyramid. Our feature pyramid has strides of [8, 16, 32, 64].

We detail our training and prediction process.

1) PREDICTION

The final output is the output of each prediction module in our L-FPN. Therefore, we can obtain a series of feature maps from L-FPN. We map each pixel (u, v) on the feature map back to the input image, which can be expressed as (u, v) = (us + [s/2], vs + [s/2]). Here, s denote the stride of each feature level in L-FPN. Then, these mapped pixels are responsible for object detection. Since a pixel may belong to multiple different object classes, each pixel needs to correspond to n objects and n bounding boxes, n > 1. To determine the value of parameter n, we conduct a series of experiments (refer to Section IV-C-3)). As shown in Figure 6, the parameter n from different feature levels has different values: \( n_f \in \{n_1 = 3, n_2 = 3, n_3 = 5, n_4 = 5\} \), where \( f \in \{F_1, F_2, F_3, F_4\} \).

\[ a: \text{CLASSIFICATION BRANCH} \]

The classification branch is responsible for predicting the category of each object. This branch has two sub-branches: class and confidence. The class sub-branch outputs a 4-dimensional tensor: [n, W, H, K], where K is the number of object classes (no background class). This tensor contains \( n \times W \times H \) objects.

\[ b: \text{CONFIDENCE} \]

The classification sub-branch achieves background recognition tasks. This sub-branch outputs a 4-dimensional tensor: [n, W, H, 1]. Confidence reflects the probability that each pixel (u, v) is a foreground or background pixel. During inference, we set a threshold \( t_h \). When confidence < \( t_h \), the pixel is considered as a background pixel. We need to remove all background pixels and apply adaptive-NMS [36] for all foreground pixels. This helps to improve inference efficiency.

\[ c: \text{BOX REGRESSION BRANCH} \]

The regression branch is responsible for the bounding box prediction. This branch also has two sub-branches: box and center-ness. We can obtain a 4-dimensional tensor from the box sub-branch: [n, W, H, 4]. Each bounding box has a 4D coordinate vector: \( B = (u_1, v_1, u_2, v_2) \). Then, we keep the center-ness the same as FCOS, and it also has a 4D vector: \( \text{Cen} = ( [u - u_1], [v - v_1], [u_2 - u], [v_2 - v] ) = (d_l, d_r, d_t, d_b) \).
TABLE 2. MAOD vs. state-of-the-art detectors on COCO test-dev.

| Method   | Backbone   | Input size | FPS  | AP   | AP_{50} | AP_{75} | AP_{S} | AP_{M} | AP_{L} |
|----------|------------|------------|------|------|---------|---------|--------|--------|--------|
| SNIP [20]| DPN-98     | -          | 1.1  | 45.7 | 67.3    | 51.1    | 29.3   | 48.8   | 57.1   |
| RPDNet [18]| Res-101-DCN | -          | 8.0  | 45.0 | 66.1    | 49.0    | 26.6   | 48.6   | 57.5   |
| PAnet [17]| Res-101-DCN | 800 × 1024 | -    | 47.4 | 67.2    | 51.8    | 30.1   | 51.7   | 60.0   |
| Libra R-CNN [19]| ResNeXt-101 | -          | 5.6  | 43.0 | 64.0    | 47.0    | 25.3   | 45.6   | 54.6   |
| TridentNet [16]| Res-101-DCN | 800 × 800 | 0.7  | 48.4 | 69.7    | 53.5    | 31.8   | 51.3   | 60.3   |

Two-stage detectors:

| SSD [39] | VGG-16 | 300 × 300 | 46 | 25.1 | 43.1 | 25.8 | 6.6 | 25.9 | 41.4 |
| DSSD [40] | ResNet-101 | 321 × 321 | 9.5 | 28.0 | 46.1 | 29.2 | 7.4 | 28.1 | 47.6 |
| RefineDet [5] | VGG-16 | 320 × 320 | 38.7 | 29.4 | 49.2 | 31.3 | 10.0 | 32.0 | 44.4 |
| RefineDet [5] | ResNet-101 | 320 × 320 | 38.6 | 29.4 | 49.2 | 31.3 | 10.0 | 32.0 | 44.4 |
| MAOD (our) | MobileDet + L-FPN | 320 × 320 | 43.3 | 63.2 | 46.1 | 24.6 | 46.3 | 55.0 |

One-stage detectors:

| YOLOv3 [41] | Darknet-53 | 608 × 608 | 20 | 33.0 | 57.9 | 34.4 | 18.3 | 35.4 | 41.9 |
| RetinaNet [11] | ResNet-101 | 832 × 500 | 11.1 | 34.4 | 53.1 | 36.8 | 14.7 | 38.5 | 49.1 |
| RefineDet [5] | ResNet-101 | 512 × 512 | 22.3 | 41.8 | 62.9 | 45.7 | 25.6 | 45.1 | 54.1 |
| CornerNet [12] | Hourglass-104 | 511 × 511 | 4.1 | 42.2 | 57.8 | 45.3 | 20.8 | 44.8 | 56.7 |
| CenterNet [15] | Hourglass-104 | 511 × 511 | 4.1 | 42.2 | 57.8 | 45.3 | 20.8 | 44.8 | 56.7 |
| CenterNet [14] | Hourglass-104 | 512 × 512 | 28 | 45.1 | 63.9 | 49.3 | 26.6 | 47.1 | 57.7 |
| MAOD (our) | MobileDet + L-FPN | 512 × 512 | 68 | 46.1 | 65.5 | 50.2 | 26.9 | 49.6 | 58.5 |

| RetinaNet [11] | ResNet-101 | 1280 × 800 | 5.0 | 39.1 | 59.1 | 42.3 | 21.8 | 42.7 | 50.2 |
| FCOS [2] | ResNeXt-101 | 1333 × 800 | 5.4 | 44.7 | 64.1 | 48.4 | 27.6 | 47.5 | 55.6 |
| FSAF [10] | ResNet-101 | 800 × 800 | 2.7 | 44.6 | 65.2 | 48.6 | 29.7 | 47.1 | 54.6 |
| MAOD (our) | MobileDet + L-FPN | 800 × 800 | 43 | 47.1 | 66.9 | 51.2 | 28.2 | 50.3 | 60.2 |

Here, FPS, AP, AP_{50}, AP_{75}, AP_{S}, AP_{M}, AP_{L} are the performance parameters of object detector. FPS: Frames Per Second. AP: Average Precision. AP_{50} and AP_{75} are AP at IoU = 0.5 and 0.75, respectively. Here, IoU denotes an intersection-over-union between a predicted bounding box and a ground-truth box. AP_{L} is AP for large objects (area > 96 × 96). AP_{S} is AP for small objects (area < 32 × 32). AP_{M} is AP for medium objects (32^2 < area < 96^2).

2) TRAINING
During training, if a pixel \((u, v)\) falls into any ground-truth box \(B_{gt} = (u_{gt}^1, v_{gt}^1, u_{gt}^2, v_{gt}^2)\), it is considered as a foreground (positive) sample \((\text{confidence} = 1)\). In all positive samples, each pixel belongs to \(m\) ground-truth boxes, \(m \geq 1\). We calculate the area of \(m\) ground-truth boxes. As each pixel needs to predict \(n\) objects, we match \(B_{gt}\) and \(B\) by area from small to large. Finally, we use multi-task loss function to train our detector in an end-to-end way.

\(\alpha: LOSS\ FUNCTION\)
Our loss function is multi-task loss function. We adopt DR Loss to train the classification branch and use Diou Loss to train the bounding box sub-branch. The center-ness subbranch also uses the DR Loss function. We define our loss function as follows:

\[
L_{total} = L_{class} + L_{conf} + L_{box} + L_{cen}
\]

\[
= L_{DR} + L_{DR} + L_{Diou} + L_{DR}
\]

(6)

DR Loss treats the classification task as a ranking problem to alleviate the class-imbalance issue [34]. Diou Loss can be formulated as:

\[
L_{Diou} = 1 - IoU + (\rho^2(b, b_{gt})/a^2)
\]

(9)

where \(\rho\) denote the Euclidean distance and \(a\) denote the diagonal length of the smallest enclosing box covering [35] \(B_{gt}\) and \(B\). \(b_{gt}\) and \(b\) represent the centers of \(B_{gt}\) and \(B\), respectively.

IV. EXPERIMENTS
In this section, we train and evaluate our MAOD on the standard benchmarks: MS COCO dataset and ImageNet-1K 2012 classification dataset [25]. ImageNet-1K (1000 classes) is used to train our backbone, and it has 1281K images for training and 50K images for validation. MS-COCO (80 classes) is used to train and test our detector. We implement MAOD in Keras. Our experiments run on a local machine, with 8 NVIDIA GTX 1080Ti GPUs, Intel Core i7-8700K CPU, CUDA 10.0, CUDNN 7.6, and Ubuntu16.04. Fig. 7 shows some detection results.

A. DETECTOR TRAINING AND INFERENCE
1) TRAINING DETAILS
We train MAOD detector with SGD on the MS-COCO trainval35k set (80K images in the training set and 35k images in the validation set). MAOD uses a pre-trained MobileDet backbone. During training, we adopt 3 input sizes: 320 × 320, 512 × 512 and 800 × 800.

Optimization Configuration: Our detector is trained for 16 epochs with SGD (stochastic gradient descent) optimizer. The values of hyper-parameters are: batch size = 64

\[
\text{smooth the loss function and } l \text{ controls the smoothness [34].}
\]

\[
L_{Diou} = 1 - IoU + (\rho^2(b, b_{gt})/a^2)
\]

where \(\rho\) denote the Euclidean distance and \(a\) denote the diagonal length of the smallest enclosing box covering [35] \(B_{gt}\) and \(B\). \(b_{gt}\) and \(b\) represent the centers of \(B_{gt}\) and \(B\), respectively.
FIGURE 7. Some detection results. MAOD can effectively detect small objects and highly overlapping objects.

(8 GPUs), weight decay = 4e-5, momentum = 0.9. In order to accelerate convergence, we employ linear warm-up [25] and cosine decay [37] to adjust the learning rate (lr): We run a warm-up training in the first 3 epochs, so that the initial lr linearly increased from 0 to 0.06. Then, by using cosine lr decay, the lr is reduced from 0.06 to 0. These strategies help to reduce training time and improve model accuracy.

2) COMPARISONS WITH STATE-OF-THE-ART DETECTORS
We compare with other state-of-the-art object detectors (including both one-stage and two-stage methods) on COCO test-dev and report their AP (Average Precision) and FPS (Frames Per Second), see Table 2. For fair comparisons, we test on a NVIDIA TITAN X GPU. The standard version MAOD (512 × 512 input size) contains only 12.8M parameters and achieves AP of 46.1% at the speed of 68 FPS.

As shown in Table 2, our detector is much faster than other detectors. When the input size is 320 × 320, the speed of MAOD can reach 91 FPS. Compared with other one-stage object detectors in Table 2, MAOD-800 (800 × 800 input size) has the highest accuracy and it can run at 43 FPS with 47.1% AP. These comparison results suggest that our MAOD achieves the best accuracy-speed trade-off on the MS COCO dataset.

B. BACKBONE TRAINING AND INFERENCE
1) TRAINING DETAILS
We train our MobileDet backbone on the ImageNet-1K dataset by RMSProp optimizer. The values of hyperparameters are: batch size = 128 (8 GPUs), weight decay = 4e-5, momentum = 0.9, initial learning rate = 0.05 which is multiplied by 0.1 after every 10 epochs (exponential learning rate decay) [38]. We test on the ImageNet-1K validation set with the single 224 × 224 crop.

2) COMPARISONS WITH STATE-OF-THE-ART MODELS
To verify the performance of our MobileDet as a feature extractor for object detection [4], we compare it with other state-of-the-art models. Table 3 and Table 4 show the comparison results on ImageNet validation set. MobileDet has 6.9M parameters and achieves 78.0% top-1 accuracy at the cost of 756M FLOPs.

As shown in Table 3, MobileDet has higher accuracy than the other lightweight mobile models. Compared with other backbone networks in Table 4, our backbone has fewer parameters and computations, while maintaining high accuracy. Although DPN-98 has the highest top-1 accuracy (79.8%), MobileDet is 14× smaller than it. Therefore, our network is more suitable as a backbone for object detection than other classification models.

### Table 3. MobileDet vs. mobile models on ImageNet.

| Models             | Top-1 (%) | Parameters (M) | FLOPs |
|--------------------|-----------|----------------|-------|
| MobileNetV2 1.0×   | 71.8      | 3.5            | 300 M |
| MobileNetV3 Large 1.0× | 75.2      | 5.4            | 219 M |
| ShuffleNetV2 1.5×  | 72.6      | 3.5            | 299 M |
| MnasNet-A2 [42]    | 75.6      | 4.8            |       |
| MoGA-A [38]        | 75.9      | 5.1            | 269* M|
| MobileDet(our) 1.0× | 78.0      | 6.9            | 756 M |

Here, * denotes our reimplementation. FLOPs: floating point operations.

### Table 4. MobileDet vs. state-of-the-art classification models on ImageNet.

| Models             | Top-1 (%) | Parameters (M) | FLOPs |
|--------------------|-----------|----------------|-------|
| ResNet-50 [25]     | 75.3      | 25.53          | 3.86 G |
| ResNet-101 [25]    | 76.4      | 47.18*         | 7.60 G |
| ResNet-152 [25]    | 77.8      | 60*            | 11.4 G |
| ResNeXt-50 [26]    | 77.77     | 25.0           | 4.24 G |
| ResNeXt-101 [26]   | 78.89     | 44.3           | 7.99 G |
| DPN-98 [20]        | 79.8*     | 61.6           | 11.7 G |
| Darknet-53 [41]    | 77.2      | 40.62*         | 6.32* G |
| DetNet-59 [28]     | 76.5      | 19.5*          | 4.8 G  |
| MobileDet(our) 1.0×| 78.0      | 6.9            | 0.756 G|

Here, * denotes our reimplementation. FLOPs: floating point operations.
TABLE 5. Ablation study for the main components.

| Architecture          | AP  | FPS | Parameters (M) |
|-----------------------|-----|-----|----------------|
| ResNet101 + FPN + FCOS| 41.5| 9.3 | 55.2           |
| MobileDet + FPN + FCOS| 42.8| 30  | 15.6           |
| MobileDet + L-FPN – FCOS| 45.6| 39  | 12.8           |
| MobileDet + L-FPN – per-pixel prediction method| 47.1| 43  | 12.8           |

We compare the impact of MobileDet backbone, L-FPN and our per-pixel prediction method. The input size is 800 × 800.

TABLE 6. With or without confidence.

| FPS  | AP  | AP50 | APs50 |
|------|-----|------|-------|
| None | 59  | 44.6 | 64.3  |
| Confidence | 68  | 46.1 | 65.5  |

Here, “None” denotes that no confidence is used. We set the threshold \( t \) to 0.5. The input size is 512 × 512.

C. ABLATION STUDY

In order to further analyze and evaluate the effectiveness of our design choice, we conduct ablation studies on the remaining 5k images in COCO validation set.

1) MAIN COMPONENTS

Since MAOD is composed of MobileDet backbone network, L-FPN and our anchor-free per-pixel prediction method, we need to know how much each of them contributes to performance improvements. In this study, the baseline is FCOS based on ResNet-101 and FPN. Table 5 shows the impact of each component.

We first replace the ResNet-101 backbone network with MobileDet, which significantly improves accuracy (AP) and inference speed (FPS) with fewer parameters. Then, we use our L-FPN instead of the original FPN. We can find that the accuracy is improved by about 3 AP and the speed reaches 39 FPS. Finally, by replacing FCOS with our per-pixel prediction method, we achieve the best result. These results show that our backbone, L-FPN and the per-pixel prediction method are all essential.

2) WITH OR WITHOUT CONFIDENCE

As mentioned before, we add a confidence sub-branch. Confidence reflects the probability that each pixel is a specific object or background. Therefore, we can use confidence to reduce inference time. Table 6 shows the effectiveness of the confidence sub-branch. By using the confidence sub-branch, the speed of our model increases from 59 FPS to 68 FPS, and the AP improves from 44.6% to 46.1%. Therefore, the confidence sub-branch can effectively enhance the performance of MAOD.

3) MULTI-OBJECTIVE PREDICTION

As mentioned above (in Section III-C-1), each pixel is responsible for detecting \( n \) objects and therefore we need to further determine the value of parameter \( n \). We test the value of parameter \( n \) on each feature level. By conducting a series of experiments, we choose \( \{n_1 = 3, n_2 = 3, n_3 = 5, n_4 = 5\} \) as a good tradeoff between model accuracy and complexity (see Figure 8).

4) ATTENTION MECHANISM AND BACKBONE

Since LSC is used to optimize our backbone, we compare it with other attention mechanisms to verify its effectiveness. In this study, we consider 4 options: SE, BAM, CBAM and no attention module, and replace LSC with them. The classification network is our MobileDet. Table 7 shows the comparison results. We can see that the MobileDet network with LSC achieve the highest accuracy on ImageNet-1K. These results suggest that our LSC is a powerful and efficient attention module.

V. CONCLUSION

In this paper, we introduce MAOD which is a computation-efficient one-stage real-time detector. Our MAOD consists of a lightweight MobileDet backbone, a lightweight FPN and an anchor-free per-pixel prediction method. MAOD can achieve a better balance between inference speed and accuracy. In practical application, our detector is more suitable for mobile devices and autonomous driving. In future research, we will further improve the training speed and inference speed of MAOD. Code will be made public.

REFERENCES

[1] X. Wu, D. Sahoo, and S. C. H. Hoi, “Recent advances in deep learning for object detection,” 2019, arXiv:1908.03673. [Online]. Available: http://arxiv.org/abs/1908.03673
[2] Z. Tian, C. Shen, H. Chen, and T. He, “FCOS: Fully convolutional one-stage object detection,” in Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV), Seoul, South Korea, Oct. 2019, pp. 9627–9636.
[3] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in Proc. ECCV, Zurich, Switzerland, 2014, pp. 740–755.
[4] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Salt Lake City, UT, USA, Jun. 2018, pp. 4510–4520.
