RefineDet++: Single-Shot Refinement Neural Network for Object Detection

Shifeng Zhang, Longyin Wen, Zhen Lei*, *Senior Member, IEEE,* and Stan Z. Li, *Fellow, IEEE*

***Abstract*—Convolutional neural network based methods have dominated object detection in recent years, which can be divided into the one-stage approach and the two-stage approach. In general, the two-stage approach (*e.g.*, Faster R-CNN) achieves high accuracy, while the one-stage approach (*e.g.*, SSD) has the advantage of high efficiency. To inherit the merits of both while overcoming their disadvantages, we propose a novel single-shot based detector, namely RefineDet++, which achieves better accuracy than two-stage methods and maintains comparable efficiency of one-stage methods. The proposed RefineDet++ consists of two inter-connected modules: the anchor refinement module and the alignment detection module. Specifically, the former module aims to (1) filter out negative anchors to reduce search space for the subsequent classifier, and (2) coarsely adjust the locations and sizes of anchors to provide better initialization for the subsequent regressor. The latter module takes (1) the refined anchors as the input from the former module with (2) a newly designed alignment convolution operation to further improve the regression accuracy and predict multi-class label. Meanwhile, we design a transfer connection block to transfer the features in the anchor refinement module to predict locations, sizes and class labels of objects in the object detection module. The multi-task loss function enables us to train the whole network in an end-to-end way. Extensive experiments on PASCAL VOC and MS COCO demonstrate that RefineDet++ achieves state-of-the-art detection accuracy with high efficiency.**

***Index Terms*—Object detection, one-stage, refinement network.**

I. Introduction

OBJECT detection is a hot topic with a long history [1]–[5], which aims to detect object instances of the predefined categories. Accurate object detection would have far reaching impact on various applications, such as image understanding, image retrieval and video surveillance. In recent years, object detection has achieved significant advances with the deep Convolutional Neural Network (CNN), which is generally divided into two categories: the two-stage approach [2], [4], [6]–[9] and the one-stage approach [3], [10]–[13].

The two-stage approach relies on two steps: (1) generating a sparse set of candidate object proposals, and then (2) regressing and classifying the object proposals for detection, which achieves state-of-the-art performances on some challenging datasets (*e.g.*, PASCAL VOC [14] and MS COCO [15]). The one-stage approach, also called single-shot approach, directly uses convolution layers to predict objects based on preset anchors of different scales and aspect ratios over the input images, whose main advantage is the fast detection speed. However, the detection accuracy of the one-stage method usually lags behind that of the two-stage method, and one of the main reasons is the problem of class imbalance [10]. Some of recent works in the one-stage approach aim to solve the class imbalance problem to improve detection accuracy. RON [12] significantly reduces the search space of objects via using the objectness prior constraint on convolutional feature maps. RetinaNet [10] reshapes the standard cross entropy loss to focus training on a sparse set of hard examples and down-weights the loss assigned to well-classified examples to address the class imbalance issue. S^3FD [16] reduces false positives resulting from class imbalance by designing a max-out labeling mechanism.

According to our view, the existing state-of-the-art two-stage methods, such as FPN [8], R-FCN [7] and Faster R-CNN [4], have the following four advantages over the one-stage methods: (1) using two-step structure with sampling heuristics to handle the class imbalance problem; (2) using two-step regression to predict the locations and sizes of the object boxes; (3) using two-step features to describe the objects; (4) using the RoI-wise operation (*e.g.*, RoIPooling [6] and RoIAlign [17]) to align features used for detection. In this paper, we present a novel object detection method namely RefineDet++ to inherit the advantages of the aforementioned two approaches (*i.e.*, two-stage and one-stage approaches) and overcome their shortcomings. It improves the architecture of the one-stage approach via designing two inter-connected modules, called the Anchor Refinement Module (ARM) and the Alignment Detection Module (ADM), as shown in Fig. 1. To be specific, the former ARM aims to (1) reduce the search space for the subsequent classifier by removing easy negative anchors, and (2) provide better initialization for the subsequent regressor via roughly adjusting the locations and sizes of anchors. The latter ADM uses (1) the refined anchors from the former module as the input with (2) a newly designed alignment convolution (AlignConv) operation instead of the vanilla convolution layer to predict multi-class labels and further improve the regression. As illustrated in Fig. 1, these two inter-connected modules emulate the two-stage structure so as to inherit the aforementioned four advantages to efficiently produce accurate detection results. Moreover, we introduce a Transfer Connection Block (TCB) to transfer the features in the ARM to predict multi-class labels, locations

Shifeng Zhang and Zhen Lei are with the Center for Biometric and Security Research (CBSR), National Laboratory of Pattern Recognition (NLPR), Institute of Automation Chinese Academy of Sciences (CASIA) and the School of Artificial Intelligence, University of Chinese Academy of Sciences (UCAS), Beijing, China (e-mail: {shifeng.zhang, zlei}@nlpr.ia.ac.cn).

Longyin Wen is with the JD Digits, Mountain View, CA, USA (e-mail: lywen.cv.workbox@gmail.com).

Stan Z. Li is with the Westlake University, Hangzhou, China (e-mail: Stan.ZQ.Li@westlake.edu.cn).

* Corresponding author.

and sizes of objects in the ADM. The multi-task loss function allows us to train the whole network in an end-to-end manner.

Extensive experiments conducted on PASCAL VOC 2007, PASCAL VOC 2012 and MS COCO datasets demonstrate that the proposed RefineDet++ achieves remarkable results in comparison to the state-of-the-art methods. In particular, it obtains 84.2% mAP on VOC 2007 and 83.8% mAP on VOC 2012 based on the VGG-16 network, and obtains 47.2% AP on MS COCO `test-dev` based on the ResNeXt-152 network. These results advance the state-of-the-arts with 0.2% and 2.0% mAPs on VOC 2007 and 2012, and outperform all previously published one-stage and two-stage methods on MS COCO. Notably, our RefineDet++ can run 27.8 FPS with the 320×320 input size and VGG-16 in inference on a NVIDIA Titan X (Maxwell) GPU. To summarize, the main contributions of this work are in three-fold as follows: 1) Introducing a novel one-stage framework for object detection, consisting of two interconnected modules: the ARM and the ADM. This makes it to outperform the two-stage approach while maintaining high efficiency of the one-stage approach. 2) Designing the TCB to transfer the features in the ARM to handle more challenging tasks, *i.e.*, predict accurate object locations, sizes and class labels, in the ADM. 3) Achieving the state-of-the-art results on generic object detection benchmarks.

Preliminary results of this work have been published in [18]. The current work has been improved and extended from the conference version in several important aspects. (1) We design a new alignment convolution (AlignConv) operation instead of the vanilla convolution layer with the fixed kernel size to extract features for accurate detection. (2) We noticeably improve the accuracy of the detector in our previous work with slightly additional overhead. (3) All sections are rewritten with more details, more references and more experiments to have a more elaborate presentation.

II. Related Work

A. Classical Object Detectors

Early object detectors are based on the sliding-window paradigm with hand-crafted features and classifiers. Viola and Jones [1] use Haar feature and AdaBoost to train a series of cascaded classifiers for face detection, achieving satisfactory accuracy with high efficiency. DPM [19] uses mixtures of multi-scale deformable part models to represent highly variable object classes. However, with the advent of CNN, the object detection task is soon dominated by the CNN-based detectors, which can be broadly divided into two categories: the one-stage approach and two-stage approach.

B. One-Stage Approach

Due to the high efficiency, the one-stage approach has attracted much more attention in recent years. OverFeat [20] performs detection from raw pixels using a ConvNet trained end-to-end. YOLO [11], [21] directly detects objects using a single feed-forward network and achieves extremely fast detection speed. SSD [3] spreads out default boxes on multi-scale layers within a ConvNet to predict object category and box offsets. RON [12] and DSSD [22] introduce additional large-scale context in object detection to improve detection accuracy by using deconvolution/upsampling layers. DSOD [23] and ScratchDet [24] point out several principles to train object detectors from scratch based on SSD. RetinaNet [10] addresses the class imbalance problem by the focal loss that focuses learning on hard examples and down-weight the numerous easy negatives. STDN [25] designs the scale-transfer layer to explicitly explore the inter-scale consistency nature across multiple detection scales. DES [26] uses a semantic segmentation branch and a location-agnostic module to enrich the semantics of object detection features. DFP [27] introduces an architecture comprised of global attention and local reconfigurations to reformulate the feature pyramid construction as feature reconfiguration process. RFB [28] proposes a receptive field block to enhance the feature discriminability and robustness. PFPNet [29] propose a parallel feature pyramid by widening the network width instead of increasing the network depth to improve the detection performance.

C. Two-Stage Approach

The two-stage approach is composed by a proposal generator (*e.g.*, Selective Search [30], EdgeBoxes [31], DeepMask [32], [33]) and a region-wise prediction subnetwork. The former one generates a sparse set of candidate object proposals, and the latter one uses a separate convolutional network to determine the corresponding class labels and the accurate object regions. It is worth noting that the two-stage approach (*e.g.*, R-CNN [2], SPPnet [34], Fast R-CNN [6] to Faster R-CNN [4]) achieves dominated performance on several challenging benchmarks including PASCAL VOC [14] and MS COCO [15]. Since then, many effective techniques have been proposed to further improve performance, such as architecture diagrams [7], [35]–[38], training strategies [39]–[42], contextual reasoning [43]–[46] and multiple layers exploiting [8], [47]–[49].

D. Anchor-Free Approach

Recent anchor-free detectors have attracted much more attention. It directly finds objects without preset anchors in two different ways. One way [50]–[54] is to first locate several predefined or self-learned keypoints and then bound the spatial extent of objects. Another way [55]–[57] is to use the center point or region of objects to define positives and then predict the four distances from positives to the object boundary. Without anchors' hyperparameters, these methods are more potential in terms of generalization ability.

E. Related Topic

There are some related topics to fully-supervised object detection, such as weakly-supervised object detection [58], [59] that uses only image-level annotations to train object detectors via a three step pipeline, instance segmentation [17], [60], [61] that requires the correct detection of all objects in an image while also precisely segmenting each instance. All of them are of growing importance and deserve further study.

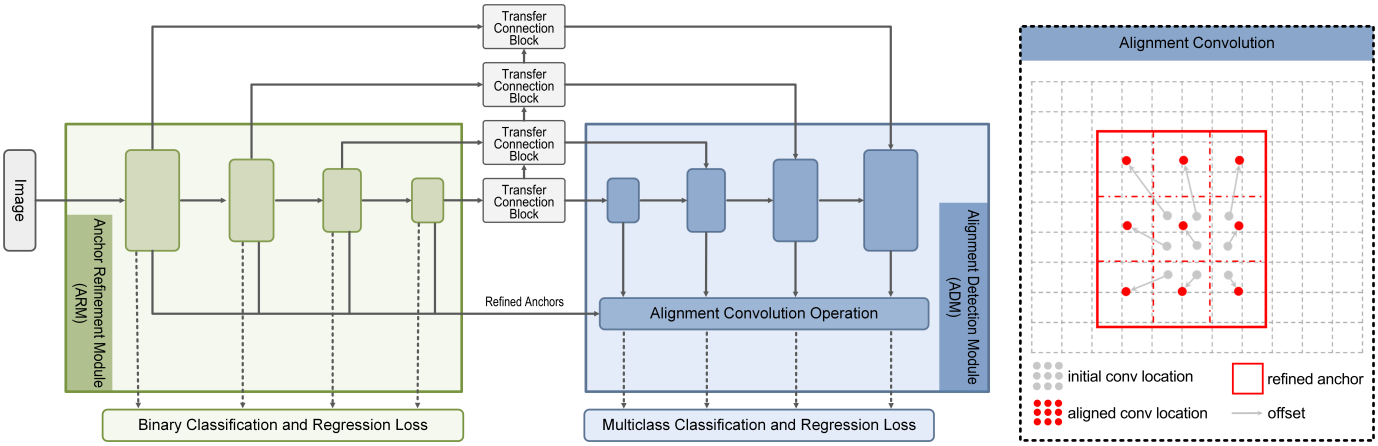


Fig. 1: Structure of RefineDet++. We only draw the layers used for detection for better visualization. The 3×3 AlignConv for one refined anchor is presented on the right side, which takes the guided offsets to complete the convolution operation.

III. RefineDet++

The overall network structure of RefineDet++ is shown in Fig. 1. Similar to SSD [3], our RefineDet++ is composed of a feed-forward convolutional network, which first generates a fixed number of detection bounding boxes with the predicted scores to indicate the presence of different classes of objects in those boxes, then applies the Non-Maximum Suppression (NMS) to output the final results. The proposed RefineDet++ consists of two inter-connected modules namely the ARM and the ADM. The former one is designed to reduce search space for the subsequent classifier via removing negative anchors and also provide better initialization for the subsequent regressor via coarsely adjusting anchors' locations and sizes. While the latter one predictes multi-class labels and further regresses accurate object locations based on the refined anchors from the ARM and the aligned features from the AlignConv. To meet our needs, the ARM is built by removing the classification layers of the backbone networks (such as VGG-16 [62], ResNet [63] and ResNeXt [64] pretrained on ImageNet [65]) and then adding some auxiliary structures. The ADM is formed by the outputs of TCBs and the AlignConv operation to generate the offsets relative to the refined anchor box coordinates and the scores for object classes. The four core components in RefineDet++ are elaborated as follows: (1) transfer connection block: converting the features from the ARM to the ADM for detection; (2) negative anchor filtering: early rejecting easy negative anchors so as to mitigate the imbalance issue; (3) two-step cascaded regression: regressing more accurate locations and sizes of objects; (4) feature alignment convolution: aligning features used for the second stage detection based on refined anchors.

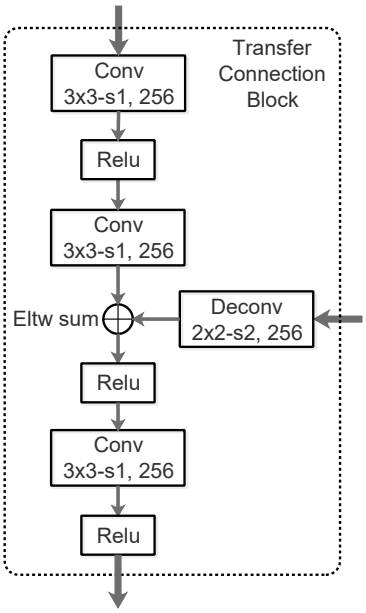


Fig. 2: Detailed structure of TCB. We intend to make TCB have the same structure as the module in FPN [8]. However, due to the incompleteness of Caffe and the unreleased implementation details of FPN, we design TCB according to our own understanding. After the official FPN implementation is released, we find that TCB has three differences: 1) TCB uses one more Conv layer; 2) TCB uses ReLU after Conv; 3) TCB uses DeConv instead of the bilinear upsampling. Comparing to the module in FPN, our TCB improves the mAP improvement by about 0.2% with slight speed drop from 28.9 FPS to 27.8 FPS, we still use TCB in this work.

A. Transfer Connection Block

As shown in Fig. 1, several TCBs are introduced to convert features of different layers in the ARM into the form required by the ADM. In this way, the ADM can share features from the ARM. It's worth noting that only the feature maps associated with anchors are applied the TCBs in the ARM. Another role of the TCBs is to integrate large-scale context [8], [22] from the high-level features to the transferred features for better detection accuracy. Fig. 2 shows the architecture of the TCB. We use the deconvolution operation to enlarge the high-level feature maps to match the dimensions, and then sum them in the element-wise way. After that, we add a convolution layer after the summation to ensure the discriminability of features used for detection.

B. Negative Anchor Filtering

The negative anchor filtering mechanism is designed in the proposed RefineDet++, which aims to early reject well-classified negative anchors in order to alleviate the imbalance problem. To be specific, during the training phase, if the negative confidence of a refined anchor is larger than the empirically preset threshold $\theta = 0.99$, we will discard it when training the subsequent ADM. In other words, only refined hard negative anchors and refined positive anchors are passed to train the subsequent ADM. Besides, in the inference phase, these refined anchors with a predicted negative confidence larger than θ will be discarded in the subsequent ADM.

C. Two-Step Cascaded Regression

Current one-stage methods [3], [12], [22] depend on one-step regression to predict the locations and sizes of objects on various feature layers with different scales, which is quite inaccurate in some challenging situations, especially for the small objects. To this end, we propose a two-step cascaded regression strategy to regress the locations and sizes of objects. Specifically, the ARM is designed to provide better initialization for the regression in the ADM by adjusting the locations and sizes of anchors. That is, we associate N anchor boxes with each regularly divided cell on the feature map. The initial position of each anchor box relative to its corresponding cell is fixed. At each feature map cell, we predict four offsets of the refined anchor boxes relative to the original tiled anchors and two confidence scores indicating the presence of foreground objects in those boxes. Thus, we can yield N refined anchor boxes at each feature map cell.

After obtaining the refined anchor boxes, we pass them to the corresponding feature maps in the ADM to further generate accurate object locations and sizes and predict object categories. The corresponding feature maps in the ARM and the ADM have the same dimension. We calculate C class scores and the four accurate offsets of objects relative to the refined anchor boxes, yielding $C + 4$ outputs for each refined anchor boxes to complete the detection task. This process is similar to the default boxes used in SSD [3]. However, in contrast to SSD [3] directly uses the regularly tiled default boxes for detection, our proposed RefineDet++ detector uses two-step strategy, *i.e.*, the ARM first generates the refined anchor boxes and then the ADM takes the refined anchor boxes as input for further detection, leading to more accurate detection results, especially for the small objects.

D. Feature Alignment Convolution

As shown in Fig. 3(a), the ARM uses regularly tiled initial anchors with the corresponding initial features to generate refined anchors. If the ADM utilizes the vanilla convolution operation, the features extracted for the second stage detection are misaligned. This means the refined anchors are predicted still based on the initial features (see Fig. 3(b)), which fails to extract accurate features to the regions of refined anchors and prevents further improvements in performance.

To this end, we design an alignment convolution operation (AlignConv), which uses the aligned feature from the refined anchors to predict multi-class labels and regress accurate object locations, as shown in Fig. 3(c). Specifically, the newly designed AlignConv operation conducts convolution operation based on computed offsets from the refined anchors. Denoting each refined anchor with a four-tuple (x, y, h, w) that specifies the top-left corner (x, y) and height and width (h, w), the AlignConv is conducted as follows. First, after taking the refined anchors from ARM, we equally divide the regions of the refined anchors into $K \times K$ parts, where K is the kernel size of convolution operation. The center of each part is computed as: for the part at i-th row and j-th column, the center location is $(x + \frac{(2i-1)\cdot w}{2K}, y + \frac{(2j-1)\cdot h}{2K})$. Second, we multiply the feature values at the $K \times K$ part centers in refined anchors with the corresponding parameters of the convolution filter, see Fig. 1. In this way, we successfully extract more accurate features that are aligned to the refined anchors for object detection. In contrast to existing deformable convolution methods [66]–[68] that learn the offsets by convolution operation with extra parameters, our AlignConv conducts the convolution with the guidance from the refined anchors of the ARM, which is more suitable for RefineDet++ and produces better performance.

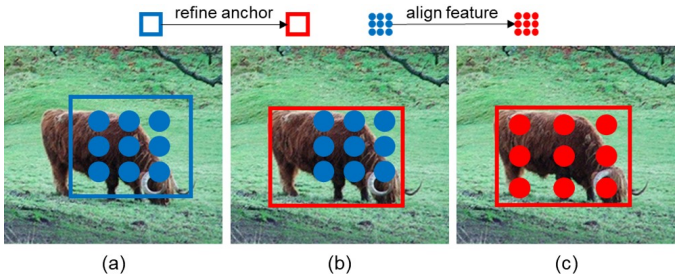


Fig. 3: Illustration of anchor refinement and feature alignment. (a) Initial anchor with the corresponding initial features. (b) Refined anchor still with the initial features. (c) Refined anchor with the aligned features.

IV. Implementation Detail

A. Data Augmentation

To construct a robust model to various objects, a series of data augmentation strategies introduced in SSD [3] are applied. Specifically, we randomly crop, expand and flip the original training images with additional random photometric distortion [69] to output the final training images. For more details, please refer to the original work [3].

B. Backbone Network

Any popular networks can be applied as the backbone for RefineDet++, such as Inception [70], [71], MobileNet [72] and ShuffleNet [73]. For fair comparison, we use VGG-16 [62], ResNet [63] and ResNeXt [64] as the backbones, which are pretrained on the ILSVRC CLS-LOC dataset [65]. Following DeepLab-LargeFOV [74], we use the reduced version of VGG-16 that converts the fc6 and fc7 layers to the convolution layers conv_fc6 and conv_fc7 by subsampling parameters. Similar to SSD [3], L2 normalization [75] is used to initially scale the feature norms in conv4_3 and conv5_3 to 10 and 8,

which will be updated during back-propagation in the training phase. These scales of the feature norms are critical to balance the feature scales between different detection layers. Besides, two extra convolution layers namely conv6_1 and conv6_2 are added to the end of the truncated VGG-16, while one extra residual block namely res6 is appended to the end of the truncated ResNet and ResNeXt. These newly added layers are used to capture high-level information and drive object detection at multiple scales.

C. Anchor Design

In our RefineDet++, four convolution layers with the total stride sizes 8, 16, 32 and 64 pixels are selected as the detection layers to handle different scales of objects. Specifically, the detection layers in the VGG-16 backbone are conv4_3, conv5_3, conv_fc7 and conv6_2, while in the ResNet backbone and the ResNeXt backbone are the final layer of each block. Each detection layer is associated with several different anchors to detect different objects. To ensure that different scales of anchors are tiled with the same density [76]–[79] on image, we use the anchor scale design strategy in [16], [80]. Specifically, we only associate one specific scale of anchors (*i.e.*, $4S$, where S represents the downsampling factor of the detection layer) and three aspect ratios (*i.e.*, 0.5, 1.0 and 2.0) at each location of the detection layer.

D. Anchor Matching

During the training phase, we divide the tiled anchors into positive and negative samples based on the Jaccard overlap [81]. To be specific, we first associate each ground-truth bounding box to an anchor with the best intersection-over-union (IoU), and then associate the anchors to other ground-truth boxes with the overlap larger than θ_p. Finally, we associate the anchor boxes to background if their IoU with any ground truth box are smaller than θ_n[1]. These anchors whose IoU with any ground truth box falls in $[\theta_n, \theta_p)$ are ignored during training. Empirically, we set $\theta_n = 0.5$ and $\theta_p = 0.5$ for the ARM and the ADM in our experiments.

E. Hard Negative Mining

As most of the anchors are assigned as negative examples after the above matching step, even for the ADM where some easy negative anchors are rejected by the ARM. This leads to extremely class imbalance of the training samples. Using all negative anchors or randomly selecting some of them will make the training process slow and unstable. To mitigate this issue, we apply hard negative mining in SSD [3] to select some negative anchors with top loss values and make the ratio between the negative and positive anchors below $3:1$.

F. Loss Function

There are two parts in the loss function of RefineDet++, *i.e.*, the loss from the ARM and the loss from the ADM. For the ARM, we regress the size and location as well as assign a binary class label (being an object or not) for each anchor to obtain the refined anchors. Then, we pass these refined anchors with the negative confidence less than the threshold to the ADM to further predict accurate object locations, sizes and categories. The loss function can be defined with these definitions as following:

$$\begin{aligned}\mathcal{L} = &\frac{1}{N_{\text{arm}}}\Big(\sum_i \mathcal{L}_{\text{b}}(p_i, [l_i^* \geq 1]) + \sum_i [l_i^* \geq 1]\mathcal{L}_{\text{r}}(x_i, g_i^*)\Big) \\ &+\frac{1}{N_{\text{adm}}}\Big(\sum_i \mathcal{L}_{\text{m}}(c_i, l_i^*) + \sum_i [l_i^* \geq 1]\mathcal{L}_{\text{r}}(t_i, g_i^*)\Big)\end{aligned} \tag{1}$$

where i is the index of anchor in a mini-batch, p_i and x_i are the predicted score for being an object and the refined coordinates of the anchor i in the ARM, c_i and t_i are the predicted object class and coordinates of the bounding box in the ADM, l_i^* and g_i^* are the ground truth class label and regression location and size of anchor i, N_{arm} and N_{adm} are the numbers of positive anchors in the ARM and the ADM. The binary classification loss $\mathcal{L}_{\text{b}}$ is the cross-entropy/log loss over two classes (object *vs.* not object), and the multi-class classification loss $\mathcal{L}_{\text{m}}$ is the softmax loss over multiple classes. Following Fast R-CNN [6], the smooth L1 loss is used as the regression loss L_{r}. The Iverson bracket indicator function $[l_i^* \geq 1]$ outputs 1 when the condition is true, *i.e.*, $l_i^* \geq 1$ (the anchor is not the negative), otherwise 0. Hence $[l_i^* \geq 1]\mathcal{L}_{\text{r}}$ means that the regression loss is ignored for negative anchors. Notably, if $N_{\text{arm}} = 0$, we set $\mathcal{L}_{\text{b}}(p_i, [l_i^* \geq 1]) = 0$ and $\mathcal{L}_{\text{r}}(x_i, g_i^*) = 0$; and if $N_{\text{adm}} = 0$, we set $\mathcal{L}_{\text{m}}(c_i, l_i^*) = 0$ and $\mathcal{L}_{\text{r}}(t_i, g_i^*) = 0$ accordingly.

G. Optimization

For the backbones used in our RefineDet++ method, such as VGG-16, ResNet and ResNeXt, we use the ILSVRC CLS-LOC pretrained models [65] to initialize them. The parameters in the newly added layers of VGG-16 are randomly initialized using the "xavier" method [82]. The newly added blocks of ResNet and ResNeXt are initialized by the parameters from a zero-mean Gaussian distribution with standard deviation 0.01. The whole network is fine-tuned using the Stochastic Gradient Descent algorithm (SGD) with 0.9 momentum, 0.0005 weight decay and 32 batch size. We use the warmup strategy [83] to gradually ramp up the learning rate from 3.125×10^{-4} to 4×10^{-3} at the first epoch. After that, it switches to the regular learning rate schedule. Different datasets have slightly different policies of the learning rate decay, which will be described later in the corresponding subsection.

H. Inference

During the inference phase, the ARM first filters out a large number of easy negative anchors whose background confidence are larger than 0.99 and then refines the locations and sizes of remaining anchors. After that, the following ADM takes the refined anchors and computes the guided offsets for the AlignConv operation to complete the next stage of detection, which produces top 400 confident detections per image. Finally, we apply NMS with the Jaccard overlap 0.45 for each class to generate the final top 200 detection results for each image.

[1]In general, we set $\theta_n \leq \theta_p$ in training.

TABLE I: Performance of different methods on PASCAL VOC. For evaluation on VOC 2007 `test` set, all methods are trained on VOC 2007 and VOC 2012 `trainval` sets. For evaluation on VOC 2012 `test` set, all methods are trained on VOC 2007 `trainval` and `test` sets plus VOC 2012 `trainval` set. Our best mAP are indicated with bold fonts.

Method	Backbone	Input size	#Boxes	FPS	mAP (%)	
					VOC 2007	VOC 2012
two-stage:						
Fast R-CNN [6]	VGG-16	$\sim$1000 $\times$ 600	$\sim$2000	0.5	70.0	68.4
Faster R-CNN [4]	VGG-16	$\sim$1000 $\times$ 600	300	7	73.2	70.4
OHEM [40]	VGG-16	$\sim$1000 $\times$ 600	300	7	74.6	71.9
SIN [84]	VGG-16	$\sim$1000 $\times$ 600	128	-	76.0	73.1
HyperNet [48]	VGG-16	$\sim$1000 $\times$ 600	100	0.88	76.3	71.4
Faster R-CNN [4]	ResNet-101	$\sim$1000 $\times$ 600	300	2.4	76.4	73.8
ION [43]	VGG-16	$\sim$1000 $\times$ 600	4000	1.25	76.5	76.4
MLKP [85]	VGG-16	$\sim$1000 $\times$ 600	300	10	78.1	75.5
MR-CNN [44]	VGG-16	$\sim$1000 $\times$ 600	250	0.03	78.2	73.9
R-FCN [7]	ResNet-101	$\sim$1000 $\times$ 600	300	9	80.5	77.6
C-FRCNN [86]	ResNet-101	$\sim$1000 $\times$ 600	300	-	82.2	-
CoupleNet [36]	ResNet-101	$\sim$1000 $\times$ 600	300	8.2	82.7	80.4
Zhai et al. [87]	ResNet-101	$\sim$1000 $\times$ 600	300	5.3	82.9	80.5
Revisiting RCNN [88]	ResNet-101+ResNet-152	$\sim$1000 $\times$ 600	-	-	84.0	81.2
one-stage:						
YOLO [21]	GoogleNet [89]	448 $\times$ 448	98	45	63.4	57.9
RON384 [12]	VGG-16	384 $\times$ 384	30600	15	75.4	73.0
SSD321 [22]	ResNet-101	321 $\times$ 321	17080	11.2	77.1	75.4
SSD300 [3]	VGG-16	300 $\times$ 300	8732	46	77.2	75.8
DSOD300 [23]	DS/64-192-48-1	300 $\times$ 300	8732	17.4	77.7	76.3
YOLOv2 [11]	Darknet-19	544 $\times$ 544	1445	40	78.6	73.4
DSSD321 [22]	ResNet-101	321 $\times$ 321	17080	9.5	78.6	76.3
STDN321 [25]	DenseNet-169	321 $\times$ 321	17080	40.1	79.3	-
SSD512 [3]	VGG-16	512 $\times$ 512	24564	19	79.8	78.5
DES300 [26]	VGG-16	300 $\times$ 300	8732	67.8†	79.5	77.0
DFP300 [27]	VGG-16	300 $\times$ 300	8732	39.5	79.6	77.5
RFB300 [28]	VGG-16	300 $\times$ 300	11620	83‡	80.5	-
SSD513 [22]	ResNet-101	513 $\times$ 513	43688	6.8	80.6	79.4
STDN513 [25]	DenseNet-169	513 $\times$ 513	43680	28.6	80.9	-
PFPNet-R320 [29]	VGG-16	320 $\times$ 320	6375	33	80.7	77.7
DSSD513 [22]	ResNet-101	513 $\times$ 513	43688	5.5	81.5	80.0
DES512 [26]	VGG-16	512 $\times$ 512	24564	27.2†	81.6	80.2
RFB512 [28]	VGG-16	512 $\times$ 512	32756	38‡	82.2	-
PFPNet-R512 [29]	VGG-16	512 $\times$ 512	16320	24	82.3	80.3
DFP512 [27]	ResNet-101	512 $\times$ 512	24564	-	82.4	81.1
RefineDet320++	VGG-16	320 $\times$ 320	6375	27.8	81.1	79.0
RefineDet512++	VGG-16	512 $\times$ 512	16320	16.1	82.5	80.9
RefineDet320++*	VGG-16	-	-	-	83.6	83.0
RefineDet512++*	VGG-16	-	-	-	**84.2**	**83.8**

* The performance is evaluated with the multi-scale testing strategy.
† The speed is evaluated with batch size 4 or 8 on an NVIDIA Titan X (Pascal) GPU.
‡ The speed is evaluated with PyTorch-0.3.0 and cuDNN V6, which is 2.6 times faster than the configuration in SSD [3].

V. EXPERIMENT

We conduct experiments on three standard datasets, namely PASCAL VOC 2007, PASCAL VOC 2012 and MS COCO, to validate the performance of the proposed method. The PASCAL VOC and MS COCO datasets include 20 and 80 object classes, respectively, where the 20 classes in PASCAL VOC are the subset of the 80 classes in MS COCO.

A. PASCAL VOC 2007

For PASCAL VOC 2007, we train all models on the union set of the VOC 2007 and 2012 `trainval` sets, and evaluate them on the VOC 2007 `test` set. After one warmup epoch, we train the model with the learning rate 4×10^{-3} for 160 epochs and then decay it to 4×10^{-4} and 4×10^{-5} for training another 40 and 40 epochs, respectively. We only use VGG-16 as the backbone network with the default batch size 32 for all experiments on the PASCAL VOC 2007 and 2012 datasets.

We compare the proposed RefineDet++ with the state-of-the-art detectors in Table I. It can be observed that with small input image, RefineDet320++ produces 81.1% mAP without bells and whistles, which is much better than all one-stage detectors with similar input size. It even performs better than some one-stage and two-stage detectors with larger input size, *e.g.*, STDN513 (80.9% mAP), SSD512 (79.8% mAP) and R-FCN (80.5% mAP). With the 512×512 image size, RefineDet512++ achieves 82.5% mAP that surpasses all one-stage methods, including DFP512 (82.4% mAP), PFPNet-R512 (82.3% mAP), RFB512 (82.2% mAP). Although some two-stage detectors (CoupleNet [36], Zhai et al. [87], Deep-Regionlets [90] and Revisiting RCNN [88]) have better performance, they use larger input size ($\sim 1000 \times 600$) than our RefineDet512++. To reduce the impact of input size, we evaluate RefineDet++ with the multi-scale testing strategy, achieving 84.2% mAP that outperforms all other state-of-the-art methods.

TABLE II: Effectiveness of various designs.

Component	RefineDet320++				
Feature Alignment Convolution?	✓				
Negative Anchor Filtering?	✓	✓			
Two-step Cascaded Regression?	✓	✓	✓		
Transfer Connection Block?	✓	✓	✓	✓	
mAP (%)	81.1	80.0	79.5	77.3	76.2

1) Inference Time: For a comprehensive understanding, we also report the inference time in the fifth column of Table I. The speed of RefineDet++ is evaluated on a NVIDIA Titan X (Maxwell) GPU with batch size 1, CUDA 9.1 and cuDNN v7. The frame-per-second (FPS) is used to evaluate the running speed. With the input sizes 320×320 and 512×512, RefineDet++ achieves 27.8 FPS (35.9ms) and 16.1 FPS (62.1ms), respectively. Among the high efficiency detectors, RefineDet512++ is the most accurate one (82.5% mAP) and RefineDet320++ achieves a faster speed of 27.8 FPS with a little drop in the accuracy (81.1% mAP). As shown in Table I, all two-stage detectors are not able to run in real-time, because they rely on the region-wise subnetwork to perform the second-step prediction that is effective but time-consuming. In contrast, our RefineDet++ discards the region-wise subnetwork but rely on the fully convolution network and the efficient AlignConv to extract accurate features for detection, making it achieve the state-of-the-art results with high efficiency.

2) Ablation Study: We construct five variants to validate the effectiveness of different components of RefineDet++ on the VOC 2007 `test` set shown in Table II. For a fair comparison, all experiments are conducted with the same settings except for the specific declaration. All models are trained on VOC 2007 and VOC 2012 `trainval` sets, and tested on VOC 2007 `test` set.

Feature Alignment Convolution. The proposed AlignConv in ADM is designed to align the features based on the refined anchors accordingly. As shown in Table II, using aligned features for detection can boost the mAP from 80.0% to 81.1%, demonstrating the effectiveness of the proposed alignment convolution. We also replace AlignConv with the deformable convolution and obtain a worse result (80.5% mAP), indicating that the AlignConv is more appropriate than the deformable convolution in our RefineDet++ method. Besides, replacing the vanilla convolution with the alignment convolution reduces the speed from 40 FPS to 27.8 FPS, because the alignment convolution makes the convolution operation inconsistent. Specifically, for vanilla convolution with 3×3 kernel size, the convolution offset of each location is exactly the same, which can be used for acceleration at both hardware and software levels. However, for alignment convolution with 3×3 kernel size, the convolution offset of each location is completely different, and the software and hardware support for this inconsistent operation is not friendly enough. As shown in Table III, using larger 5×5 kernel size slightly improves the mAP by 0.2% but significantly reduces the speed from 27.8 FPS to 22.1 FPS. Moreover, using too large kernel size (*e.g.*, 7×7) even slightly decreases the mAP performance because too much contextual information is introduced. Thus, we select the common 3×3 kernel size as default in the introduced alignment convolution.

TABLE III: Analysis of different kernel sizes of feature alignment convolution. mAP is evaluated on VOC 2007.

Kernel Size	3×3	5×5	7×7
mAP (%)	81.1	81.3	81.2
FPS	27.8	22.1	16.3

TABLE IV: Analysis of different θ for negative anchor filtering. mAP is evaluated on VOC 2007.

θ	1	0.99	0.98	0.95	0.9	0.8
mAP (%)	79.5	80.0	80.0	79.6	78.1	75.2

Negative Anchor Filtering. To verify the effectiveness of the negative anchor filtering, we set the confidence threshold θ of the anchors to be negative to 1.0 in both training and testing. In this case, all refined anchors will be sent to the ADM for detection. Other parts of RefineDet++ remain unchanged. Removing negative anchor filtering leads to 0.5% drop in mAP (*i.e.*, 80.0% *vs.* 79.5%). The reason is that most of these well-classified negative anchors will be filtered out during training, which solves the class imbalance issue to some extent. Besides, we conduct some experiments with different $\theta = 1, 0.99, 0.98, 0.95, 0.9, 0.8$ to analyze this hyper-parameter. As shown in Table IV, large values of θ (*e.g.*, 0.99 and 0.98) improve the performance, while small values of θ (*e.g.*, 0.9 and 0.8) decrease the performance. The reason is that too small values of θ make the first step ARM filter out too many negative anchors (even positive anchors), causing the second step ADM to fail to be trained sufficiently. We also only filter negative anchors in the training phase using $\theta = 0.99$, the performance slightly decreases from 81.1% to 80.8%, because the first step ARM can filter out some false positives that cannot be filtered by the second step ADM. If the first step does not perform the filtering operation during the inference phase, some false positives will be output, causing slight performance degradation.

Two-Step Cascaded Regression. To validate the effectiveness of the two-step cascaded regression, we redesign the network structure by directly using the regularly paved anchors instead of the refined ones from the ARM (see the fourth column in Table II). As shown in Table II, we find that mAP is reduced from 79.5% to 77.3%. This sharp decline (*i.e.*, 2.2%) demonstrates that the two-step anchor cascaded regression significantly help promote the performance.

Transfer Connection Block. We construct a network by cutting the TCBs in RefineDet++ and redefining the loss function in the ARM to directly detect multi-class of objects, just like SSD, to demonstrate the effect of the TCB. The detection accuracy of the model is presented in the fifth column in Table II. We compare the results in the fourth and fifth columns in Table II (77.3% *vs.* 76.2%) and find that the TCB improves the mAP by 1.1%. The main reason is that the model can inherit the discriminative features from the ARM, and integrate large-scale context information to improve the detection accuracy by using the TCB.

TABLE V: Performance of different methods on MS COCO `test-dev` subset. Our best results are indicated with bold fonts.

Method	Data	Backbone	Resolution	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	AR_1	AR_{10}	AR_{100}	AR_S	AR_M	AR_L
two-stage:															
SIN [84]	train	VGG-16	∼600×1000	23.2	44.5	22.0	7.3	24.5	36.3	22.6	31.6	32.0	10.5	34.7	51.3
ION [43]	train	VGG-16	∼600×1000	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
OHEM++ [40]	trainval	VGG-16	∼600×1000	25.5	45.9	26.1	7.4	27.7	40.3	-	-	-	-	-	-
MLKP [85]	trainval35	ResNet-101	∼600×1000	28.6	52.4	31.6	10.8	33.4	45.1	27.0	40.9	41.4	15.8	47.8	62.2
R-FCN [7]	trainval	ResNet-101	∼800×1333	29.9	51.9	-	10.8	32.8	45.0	-	-	-	-	-	-
CoupleNet [36]	trainval	ResNet-101	∼600×1000	34.4	54.8	37.2	13.4	38.1	50.8	30.0	45.0	46.4	20.7	53.1	68.5
Faster R-CNN [63]	trainval	ResNet-101-C4	∼600×1000	34.9	55.7	37.4	15.6	38.7	50.9	-	-	-	-	-	-
FPN [8]	trainval35k	ResNet-101-FPN	∼600×1000	36.2	59.1	39.0	18.2	39.0	48.2	-	-	-	-	-	-
TDM [49]	trainval	Inception-ResNet-v2	∼600×1000	36.8	57.7	39.2	16.2	39.8	52.1	31.6	49.3	51.9	28.1	56.6	71.1
Deform R-FCN [66]	trainval	Aligned-Inception-ResNet	∼600×1000	37.5	58.0	40.8	19.4	40.1	52.5	-	-	-	-	-	-
Hu et al. [91]	trainval35k	ResNet-101	∼800×1333	39.0	58.6	42.9	-	-	-	-	-	-	-	-	-
DeepRegionlets [90]	trainval35k	ResNet-101	∼600×1000	39.3	59.8	-	21.7	43.7	50.9	-	-	-	-	-	-
FitnessNMS [92]	trainval	DeNet-101	∼800×1333	39.5	58.0	42.6	18.9	43.5	54.1	-	-	-	-	-	-
GA-RPN [93]	trainval35k	ResNet-50	∼800×1333	39.8	59.2	43.5	21.8	42.6	50.7	-	-	-	-	-	-
Gu et al. [94]	trainval35k	ResNet-101	∼800×1333	39.9	63.1	43.1	22.2	43.4	51.6	-	-	-	-	-	-
DetNet [95]	trainval35k	DetNet-59	∼800×1333	40.3	62.1	43.8	23.6	42.6	50.0	-	-	-	-	-	-
Soft-NMS [96]	trainval	ResNet-101	∼800×1333	40.9	62.8	-	23.3	43.6	53.3	-	54.7	60.4	-	-	-
SOD-MTGAN [97]	trainval35k	ResNet-101	∼800×1333	41.4	63.2	45.4	24.7	44.2	52.6	-	-	-	-	-	-
G-RMI [98]	trainval35k	Ensemble of Five Models	∼600×1000	41.6	61.9	45.4	23.9	43.5	54.9	-	-	-	-	-	-
C-Mask RCNN [86]	trainval35k	ResNet-101	∼800×1333	42.0	62.9	46.4	23.4	44.7	53.8	-	-	-	-	-	-
Cascade R-CNN [37]	trainval35k	ResNet-101	∼800×1333	42.8	62.1	46.3	23.7	45.5	55.2	-	-	-	-	-	-
Revisiting RCNN [88]	trainval35k	ResNet-101+ResNet-152	-	43.1	66.1	47.3	25.8	45.9	55.3	-	-	-	-	-	-
Grid R-CNN [99]	trainval35k	ResNeXt-101	∼800×1333	43.2	63.0	46.6	25.1	46.5	55.2	-	-	-	-	-	-
SNIP [100]	trainval35k	DPN-98	∼1200×2000	45.7	67.3	51.1	29.3	48.8	57.1	-	-	-	-	-	-
one-stage:															
YOLOv2 [11]	trainval35k	DarkNet-19 [11]	544×544	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
SSD300 [3]	trainval35k	VGG-16	300×300	25.1	43.1	25.8	6.6	25.9	41.4	23.7	35.1	37.2	11.2	40.4	58.4
RON384++ [12]	trainval	VGG-16	384×384	27.4	49.5	27.1	-	-	-	-	-	-	-	-	-
DSSD321 [22]	trainval35k	ResNet-101	321×321	28.0	46.1	29.2	7.4	28.1	47.6	25.5	37.1	39.4	12.7	42.0	62.6
STDN300 [25]	trainval	DenseNet-169	300×300	28.0	45.6	29.4	7.9	29.7	45.1	24.4	36.1	38.7	12.5	42.7	60.1
DES300 [26]	trainval35k	VGG-16	300×300	28.3	47.3	29.4	8.5	29.9	45.2	25.6	38.3	40.7	14.1	44.7	62.0
SSD512 [3]	trainval35k	VGG-16	512×512	28.8	48.5	30.3	10.9	31.8	43.5	26.1	39.5	42.0	16.5	46.6	60.8
RFB300 [28]	trainval35k	VGG-16	300×300	29.9	49.9	31.1	11.9	31.8	44.7	-	-	-	-	-	-
DFP300 [27]	trainval	ResNet-101	300×300	31.3	50.5	32.0	-	-	-	-	-	-	-	-	-
PFPNet-R320 [29]	trainval35k	VGG-16	320×320	31.8	52.9	33.6	12.0	35.5	46.1	-	-	-	-	-	-
STDN513 [25]	trainval	DenseNet-169	513×513	31.8	51.0	33.6	14.4	36.1	43.4	27.0	40.1	41.9	18.3	48.3	57.2
DES512 [26]	trainval35k	VGG-16	512×512	32.8	53.2	34.6	13.9	36.0	47.6	28.4	43.5	46.2	21.6	50.7	64.6
DSSD513 [22]	trainval35k	ResNet-101	513×513	33.2	53.3	35.2	13.0	35.4	51.1	28.9	43.5	46.2	21.8	49.1	66.4
RFB512-E [28]	trainval35k	VGG-16	512×512	34.4	55.7	36.4	17.6	37.0	47.6	-	-	-	-	-	-
DFP512 [27]	trainval	ResNet-101	512×512	34.6	54.3	37.3	-	-	-	-	-	-	-	-	-
PFPNet-R512 [29]	trainval35k	VGG-16	512×512	35.2	57.6	37.9	18.7	38.6	45.9	-	-	-	-	-	-
RetinaNet [10]	trainval35k	ResNet-101	∼800×1333	39.1	59.1	42.3	21.8	42.7	50.2	-	-	-	-	-	-
ExtremeNet [52]	trainval35k	Hourglass-104	511×511	40.2	55.5	43.2	20.4	43.2	53.1	-	-	-	-	-	-
CornerNet511 [50]	trainval35k	Hourglass-104	511×511	40.5	56.5	43.1	19.4	42.7	53.9	35.3	54.3	59.1	37.4	61.9	76.9
CenterNet-HG [53]	trainval35k	Hourglass-104	512×512	42.1	61.1	45.9	24.1	45.5	52.8	-	-	-	-	-	-
FoveaBox [57]	trainval35k	ResNeXt-101	∼800×1333	42.1	61.9	45.2	24.9	46.8	55.6	-	-	-	-	-	-
FSAF [55]	trainval35k	ResNeXt-64x4d-101	∼800×1333	42.9	63.8	46.3	26.6	46.2	52.7	-	-	-	-	-	-
FCOS [56]	trainval35k	ResNeXt-64x4d-101	∼800×1333	43.2	62.8	46.6	26.5	46.2	53.3	-	-	-	-	-	-
CenterNet511 [51]	trainval35k	Hourglass-104	511×511	44.9	62.4	48.1	25.6	47.4	57.4	36.1	58.4	63.3	41.3	67.1	80.2
RepPoints [54]	trainval35k	ResNet-101-DCN	∼800×1333	45.0	66.1	49.0	26.6	48.6	57.5	-	-	-	-	-	-
RefineDet320++*	trainval35k	ResNet-101	320×320	33.2	53.4	35.1	13.1	35.5	51.0	28.3	44.5	47.8	20.9	53.1	70.1
RefineDet512++*	trainval35k	ResNet-101	512×512	37.7	57.9	40.3	18.8	42.4	54.2	31.7	49.4	52.3	30.3	58.1	70.4
RefineDet1024++†	trainval35k	ResNet-101	1024×1024	42.1	62.8	45.8	26.0	47.3	53.1	34.3	53.9	56.2	39.5	61.0	70.7
RefineDet1024++	trainval35k	ResNet-152	1024×1024	43.2	63.3	46.8	26.3	49.3	54.5	34.4	54.9	57.3	39.6	61.9	71.2
RefineDet1024++	trainval35k	ResNeXt-152_32×4d	1024×1024	43.8	64.1	47.7	27.3	50.0	55.2	34.8	54.9	57.4	40.5	62.0	71.3
RefineDet1024++‡	trainval35k	ResNeXt-152_32×4d	1024×1024	**47.2**	**67.8**	**51.4**	**29.3**	**51.1**	**58.6**	**36.7**	**58.9**	**63.8**	**45.2**	**68.4**	**80.7**

* The inference speed of RefineDet320++ and RefineDet512++ based on ResNet-101 are 12.4 FPS and 6.9 FPS, respectively.
† For the 1024 × 1024 training size, we add one more residual block to generate a feature map with total stride size 128 as another detection layer.
‡ The performance is evaluated using the multi-scale testing strategy.

B. PASCAL VOC 2012

Comparing to the VOC 2007 `test` set, the annotations in the VOC 2012 `test` set are held-out and researchers are required to submit the detection results to an online evaluation server to evaluate the performance of their detectors. Following the training and testing protocol, we use all the images in the VOC 2007 and the `trainval` images in VOC 2012 ($21,503$ images) to train RefineDet++, and test it on the VOC 2012 `test` set with $10,991$ images. The same learning rate policy as PASCAL VOC 2007 is used for training.

We report the mAP scores of the proposed RefineDet++ and compare it to the state-of-the-art detectors. As shown in Table I, it can be found that RefineDet320++ achieves the top accuracy (79.0% mAP) among the detectors with the similar input size 320×320, which produces 1.3% higher mAP than the second best algorithm (77.7% of PFPNet-R320). This

result is even better than the majority of the one-stage and two-stage methods with the larger input size, such as SSD512 (78.5%) and R-FCN (77.6%). Zooming in the input size to 512×512, the proposed method can produce 80.9% mAP that is only slightly lower than 81.1% mAP of DFP512 [27]. Comparing to the two-stage detectors, only DeepRegionlets [90] and Revisiting RCNN [88] produce better performance than our detector. However, these two detectors use bells and whistles during training and inference with 1000×600 input size. To have a more fair comparison, we also evaluate RefineDet++ with the multi-scale testing strategy and achieve 83.0% mAP with the input size 320×320 and 83.8% mAP with 512×512, surpassing all published two-stage and one-stage approaches.

C. MS COCO

We also evaluate RefineDet++ on the very challenging MS COCO dataset [15], which includes 80 object classes. MS COCO contains $80k$+ images for training, $40k$+ for validation, and $20k$+ for testing. Following the evaluation protocol [43], we use all images in the `training` set and $35k$+ images in the `validation` set to train our RefineDet++ model, denoted as the `trainval35k` set. The remaining $5,000$ images in the `validation` set are used for hyper-parameter searching. The average precision (AP) over different IoU threshold of all object categories is used as the primary evaluation metric. The ResNet and ResNeXt are used as the backbone network in our RefineDet++. We submit the detection results of RefineDet++ on the `test-dev` set to the evaluation server for evaluation. After one warmup epoch, we start the regular training with the initial learning rate 4×10^{-3} and divide it by 10 at 80 and 100 epochs. The total number of training epochs is 110.

We report the detection results on the MS COCO `test-dev` set in Table V. RefineDet320++ achieves 33.2% AP that is better than numerous methods, including PFPNet-R320 (31.8% AP), DFP300 (31.3% AP), STDN513 (28.0% AP), DES512 (32.8% AP) and R-FCN (29.9% AP). We can further improve the detection accuracy of RefineDet++ to 37.7% by using large input image size 512×512. It surpasses all the one-stage detectors except CornerNet511 [50] (40.5% AP), which detects an object bounding box as a pair of keypoints. Comparing to the two-stage detectors, RefineDet512++ performs better than several methods, such as Faster R-CNN [4], CoupleNet [36] and Deformable R-FCN [66]. However, it still falls behind the top two-stage algorithms regarding to the detection accuracy. Comparing to our RefineDet512++ based on ResNet-101, all of these two-stage algorithms use much larger training and testing images ($\sim 1333 \times 800$), some of them apply much stronger backbones (*e.g.*, ResNet-152, DPN-98 and Inception-ResNet) and some of them utlize extra training and testing strategies (*e.g.*, auxiliary filter and model ensemble). For a fair comparison, we also explore the larger input size and stronger backbones. As shown in Table V, using 1024×1024 input image size improves the AP performance from 37.7% to 42.1% and applying stronger backbone further increases the AP result by 1.1% for ResNet-152 and by 1.7% for ResNeXt-152_32×4d. Finally, with the multi-scale testing strategy, RefineDet1024++ based on ResNeXt-152_32×4d achieves 47.2% AP, which performs better than all the one-stage and two-stage detectors.

D. Discussion

One-Stage *vs.* Two-Stage Approach. The two-stage detectors use the second stage to classify and refine each candidate box via a region-wise subnetwork, which is effective but time-consuming. The one-stage detectors do not have this region-wise stage so they can run fast. In our opinion, the key aspect to distinguish the one-stage and two-stage detector is whether it has the region-wise operation. Our real-time method achieves state-of-the-art performance via the two-step classification and regression, mainly due to without the time-consuming region-wise operation (e.g., RoIPooling). Thus, we consider it as one-stage approach.

Multi-Scale Testing Strategy. We use several scales with horizontal flipping to test the trained model, and then merge all these results via different NMS (*e.g.*, vanilla NMS, soft-NMS [96], bounding box voting [44]). We have released all the related multi-scale testing codes at https://github.com/sfzhang15/RefineDet/blob/master/test/lib/fast_rcnn/test.py.

VI. Conclusion

This paper presents a single-shot object detector based on the refinement neural network with two inter-connected modules. The first module namely ARM is designed to (1) provide better initialization for the subsequent regressor via coarsely adjusting anchors' locations and sizes, and (2) reduce search space for the subsequent classifier by filtering out the negative anchors. The second module namely ADM takes the refined anchors from the former ARM as the input with a newly designed alignment convolution operation to predict the corresponding multi-class labels and regress more accurate locations and sizes. As a consequence, the proposed RefineDet++ method achieves the state-of-the-art detection accuracy with high efficiency on several object detection benchmark datasets including PASCAL VOC 2007, PASCAL VOC 2012 and MS COCO. In the future, we plan to design a lightweight architecture with the help of automatic machine learning (AutoML) methods to make RefineDet++ run in real-time not only on the GPU devices but also on the CPU and embedded devices.

Acknowledgments

This work has been partially supported by the Chinese National Natural Science Foundation Projects #61872367, #61876178, #61806196, #61806203, #61976229. Zhen Lei is the corresponding author.

References

[1] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *CVPR*, 2001, pp. 511–518.
[2] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *CVPR*, 2014.

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," in *ECCV*, 2016.

[4] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *TPAMI*, 2017.

[5] K. Kang, H. Li, J. Yan, X. Zeng, B. Yang, T. Xiao, C. Zhang, Z. Wang, R. Wang, X. Wang, and W. Ouyang, "T-CNN: tubelets with convolutional neural networks for object detection from videos," *TCSVT*, vol. 28, no. 10, pp. 2896–2907, 2018.

[6] R. B. Girshick, "Fast R-CNN," in *ICCV*, 2015.

[7] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: object detection via region-based fully convolutional networks," in *NIPS*, 2016.

[8] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017.

[9] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Occlusion-aware R-CNN: detecting pedestrians in a crowd," in *ECCV*, 2018, pp. 657–674.

[10] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017.

[11] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," in *CVPR*, 2017.

[12] T. Kong, F. Sun, A. Yao, H. Liu, M. Lu, and Y. Chen, "RON: reverse connection with objectness prior networks for object detection," in *CVPR*, 2017.

[13] K. Song, H. Yang, and Z. Yin, "Multi-scale attention deep neural network for fast accurate object detection," *TCSVT*, vol. 29, no. 10, pp. 2972–2985, 2019.

[14] M. Everingham, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[15] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," in *ECCV*, 2014.

[16] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li, "S^3FD: Single shot scale-invariant face detector," in *ICCV*, 2017.

[17] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," in *ICCV*, 2017.

[18] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *CVPR*, 2018.

[19] P. F. Felzenszwalb, R. B. Girshick, D. A. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *TPAMI*, vol. 32, no. 9, pp. 1627–1645, 2010.

[20] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," in *ICLR*, 2014.

[21] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[22] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "DSSD : Deconvolutional single shot detector," *CoRR*, 2017.

[23] Z. Shen, Z. Liu, J. Li, Y. Jiang, Y. Chen, and X. Xue, "DSOD: learning deeply supervised object detectors from scratch," in *ICCV*, 2017.

[24] R. Zhu, S. Zhang, X. Wang, L. Wen, H. Shi, L. Bo, and T. Mei, "Scratchdet: Exploring to train single-shot object detectors from scratch," in *CVPR*, 2019.

[25] P. Zhou, B. Ni, C. Geng, J. Hu, and Y. Xu, "Scale-transferrable object detection," in *CVPR*, 2018.

[26] Z. Zhang, S. Qiao, C. Xie, W. Shen, B. Wang, and A. L. Yuille, "Single-shot object detection with enriched semantics," in *CVPR*, 2018.

[27] T. Kong, F. Sun, W. Huang, and H. Liu, "Deep feature pyramid reconfiguration for object detection," in *ECCV*, 2018.

[28] S. Liu, D. Huang, and Y. Wang, "Receptive field block net for accurate and fast object detection," in *ECCV*, 2018.

[29] S. Kim, H. Kook, J. Sun, M. Kang, and S. Ko, "Parallel feature pyramid network for object detection," in *ECCV*, 2018.

[30] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *IJCV*, vol. 104, no. 2, pp. 154–171, 2013.

[31] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *ECCV*, 2014, pp. 391–405.

[32] P. H. O. Pinheiro, R. Collobert, and P. Dollár, "Learning to segment object candidates," in *NIPS*, 2015, pp. 1990–1998.

[33] P. O. Pinheiro, T. Lin, R. Collobert, and P. Dollár, "Learning to refine object segments," in *ECCV*, 2016, pp. 75–91.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *ECCV*, 2014.

[35] H. Lee, S. Eum, and H. Kwon, "ME R-CNN: multi-expert region-based CNN for object detection," in *ICCV*, 2017.

[36] Y. Zhu, C. Zhao, J. Wang, X. Zhao, Y. Wu, and H. Lu, "Couplenet: Coupling global structure with local parts for object detection," in *ICCV*, 2017.

[37] Z. Cai and N. Vasconcelos, "Cascade R-CNN: delving into high quality object detection," in *CVPR*, 2018.

[38] K. Cheng, Y. Chen, and W. Fang, "Improved object detection with iterative localization refinement in convolutional neural networks," *TCSVT*, vol. 28, no. 9, pp. 2261–2275, 2018.

[39] M. Najibi, M. Rastegari, and L. S. Davis, "G-CNN: an iterative grid based object detector," in *CVPR*, 2016, pp. 2369–2377.

[40] A. Shrivastava, A. Gupta, and R. B. Girshick, "Training region-based object detectors with online hard example mining," in *CVPR*, 2016.

[41] X. Wang, A. Shrivastava, and A. Gupta, "A-fast-rcnn: Hard positive generation via adversary for object detection," in *CVPR*, 2017.

[42] D. Yoo, S. Park, J. Lee, A. S. Paek, and I. Kweon, "Attentionnet: Aggregating weak directions for accurate object detection," in *ICCV*, 2015, pp. 2659–2667.

[43] S. Bell, C. L. Zitnick, K. Bala, and R. B. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in *CVPR*, 2016.

[44] S. Gidaris and N. Komodakis, "Object detection via a multi-region and semantic segmentation-aware CNN model," in *ICCV*, 2015.

[45] A. Shrivastava and A. Gupta, "Contextual priming and feedback for faster R-CNN," in *ECCV*, 2016, pp. 330–348.

[46] X. Zeng, W. Ouyang, B. Yang, J. Yan, and X. Wang, "Gated bi-directional CNN for object detection," in *ECCV*, 2016, pp. 354–369.

[47] Z. Cai, Q. Fan, R. S. Feris, and N. Vasconcelos, "A unified multi-scale deep convolutional neural network for fast object detection," in *ECCV*, 2016, pp. 354–370.

[48] T. Kong, A. Yao, Y. Chen, and F. Sun, "Hypernet: Towards accurate region proposal generation and joint object detection," in *CVPR*, 2016.

[49] A. Shrivastava, R. Sukthankar, J. Malik, and A. Gupta, "Beyond skip connections: Top-down modulation for object detection," *CoRR*, 2016.

[50] H. Law and J. Deng, "Cornernet: Detecting objects as paired keypoints," in *ECCV*, 2018.

[51] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *ICCV*, 2019.

[52] X. Zhou, J. Zhuo, and P. Krähenbühl, "Bottom-up object detection by grouping extreme and center points," in *CVPR*, 2019, pp. 850–859.

[53] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *CoRR*, 2019.

[54] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "Reppoints: Point set representation for object detection," in *ICCV*, 2019.

[55] C. Zhu, Y. He, and M. Savvides, "Feature selective anchor-free module for single-shot object detection," in *CVPR*, 2019, pp. 840–849.

[56] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: fully convolutional one-stage object detection," in *ICCV*, 2019.

[57] T. Kong, F. Sun, H. Liu, Y. Jiang, and J. Shi, "Foveabox: Beyond anchor-based object detector," *CoRR*, 2019.

[58] P. Tang, X. Wang, A. Wang, Y. Yan, W. Liu, J. Huang, and A. L. Yuille, "Weakly supervised region proposal network and object detection," in *ECCV*, 2018, pp. 370–386.

[59] P. Tang, X. Wang, S. Bai, W. Shen, X. Bai, W. Liu, and A. L. Yuille, "PCL: proposal cluster learning for weakly supervised object detection," *TPAMI*, vol. 42, no. 1, pp. 176–191, 2020.

[60] Z. Huang, L. Huang, Y. Gong, C. Huang, and X. Wang, "Mask scoring R-CNN," in *CVPR*, 2019, pp. 6409–6418.

[61] J. Yao, Z. Yu, J. Yu, and D. Tao, "Single pixel reconstruction for one-stage instance segmentation," *CoRR*, 2019.

[62] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, 2014.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[64] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *CVPR*, 2017.

[65] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *IJCV*, vol. 115, no. 3, pp. 211–252, 2015.

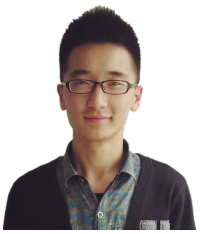
[66] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei, "Deformable convolutional networks," in *ICCV*, 2017.

[67] X. Chen, Z. Wu, and J. Yu, "Dual refinement network for single-shot object detection," *CoRR*, 2018.

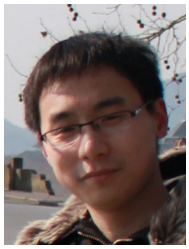
[68] X. Zhu, H. Hu, S. Lin, and J. Dai, "Deformable convnets V2: more deformable, better results," in *CVPR*, 2019, pp. 9308–9316.

[69] A. G. Howard, "Some improvements on deep convolutional neural network based image classification," *CoRR*, 2013.

[70] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *ICML*, 2015.
[71] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," in *AAAI*, 2017, pp. 4278–4284.
[72] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, 2017.
[73] N. Ma, X. Zhang, H. Zheng, and J. Sun, "Shufflenet V2: practical guidelines for efficient CNN architecture design," in *ECCV*, 2018.
[74] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," in *ICLR*, 2015.
[75] W. Liu, A. Rabinovich, and A. C. Berg, "Parsenet: Looking wider to see better," in *ICLR workshop*, 2016.
[76] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li, "Detecting face with densely connected face proposal network," in *CCBR*, 2017.
[77] S. Zhang, X. Zhu, Z. Lei, X. Wang, H. Shi, and S. Z. Li, "Detecting face with densely connected face proposal network," *Neurocomputing*, vol. 284, pp. 119–127, 2018.
[78] S. Zhang, X. Zhu, Z. Lei, H. Shi, X. Wang, and S. Z. Li, "Faceboxes: A CPU real-time face detector with high accuracy," in *IJCB*, 2017.
[79] S. Zhang, X. Wang, Z. Lei, and S. Z. Li, "Faceboxes: A CPU real-time and accurate unconstrained face detector," *Neurocomputing*, vol. 364, pp. 297–309, 2019.
[80] S. Zhang, L. Wen, H. Shi, Z. Lei, S. Lyu, and S. Z. Li, "Single-shot scale-aware network for real-time face detection," *Int. J. Comput. Vis.*, vol. 127, no. 6-7, pp. 537–559, 2019.
[81] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable object detection using deep neural networks," in *CVPR*, 2014, pp. 2155–2162.
[82] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *AISTATS*, 2010, pp. 249–256.
[83] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He, "Accurate, large minibatch SGD: training imagenet in 1 hour," *CoRR*, 2017.
[84] Y. Liu, R. Wang, S. Shan, and X. Chen, "Structure inference net: Object detection using scene-level context and instance-level relationships," in *CVPR*, 2018.
[85] H. Wang, Q. Wang, M. Gao, P. Li, and W. Zuo, "Multi-scale location-aware kernel representation for object detection," in *CVPR*, 2018.
[86] Z. Chen, S. Huang, and D. Tao, "Context refinement for object detection," in *ECCV*, 2018, pp. 74–89.
[87] Y. Zhai, J. Fu, Y. Lu, and H. Li, "Feature selective networks for object detection," in *CVPR*, 2018.
[88] B. Cheng, Y. Wei, H. Shi, R. S. Feris, J. Xiong, and T. S. Huang, "Revisiting RCNN: on awakening the classification power of faster RCNN," in *ECCV*, 2018.
[89] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *CVPR*, 2015, pp. 1–9.
[90] H. Xu, X. Lv, X. Wang, Z. Ren, N. Bodla, and R. Chellappa, "Deep regionlets for object detection," in *ECCV*, 2018.
[91] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *CVPR*, 2018.
[92] L. Tychsen-Smith and L. Petersson, "Improving object localization with fitness NMS and bounded iou loss," in *CVPR*, 2018.
[93] J. Wang, K. Chen, S. Yang, C. C. Loy, and D. Lin, "Region proposal by guided anchoring," in *CVPR*, 2019, pp. 2965–2974.
[94] J. Gu, H. Hu, L. Wang, Y. Wei, and J. Dai, "Learning region features for object detection," in *ECCV*, 2018, pp. 392–406.
[95] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "Detnet: A backbone network for object detection," in *ECCV*, 2018.
[96] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis, "Soft-nms - improving object detection with one line of code," in *ICCV*, 2017.
[97] Y. Bai, Y. Zhang, M. Ding, and B. Ghanem, "SOD-MTGAN: small object detection via multi-task generative adversarial network," in *ECCV*, 2018, pp. 210–226.
[98] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors," in *CVPR*, 2017.
[99] X. Lu, B. Li, Y. Yue, Q. Li, and J. Yan, "Grid R-CNN," in *CVPR*, 2019.
[100] B. Singh and L. S. Davis, "An analysis of scale invariance in object detection - SNIP," in *CVPR*, 2018.



Shifeng Zhang received the B.S. degree from the University of Electronic Science and Technology of China (UESTC), in 2015. Since September 2015, he has been a Ph.D. candidate at the National Laboratory of Pattern Recognition (NLPR), Institute of Automation, Chinese Academy of Science (CASIA). His research interests include computer vision, pattern recognition, especially with a focus on object detection, face detection, pedestrian detection, video detection.



Longyin Wen received the B.Eng. degree in automation from the University of Electronic Science and Technology of China, Chengdu, China, in 2010, and the Ph.D. degree from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2015. He is currently a staff scientist in JD Digits, Mountain View, CA. Before that, He was a computer vision scientist in GE Global Research, Niskayuna, NY. He was a Post-Doctoral Researcher with University at Albany, State University of New York, Albany, NY, USA, from 2015 to 2016. His current research interests include computer vision, pattern recognition, and video analysis.

Zhen Lei received the BS degree in automation from the University of Science and Technology of China, in 2005, and the PhD degree from the Institute of Automation, Chinese Academy of Sciences, in 2010, where he is currently a professor. He has published more than 160 papers in international journals and conferences. His research interests are in computer vision, pattern recognition, image processing, and face recognition in particular. He served as an area chair of the International Joint Conference on Biometrics in 2014, the IAPR/IEEE International Conference on Biometric in 2015, 2016, 2018, and the IEEE International Conference on Automatic Face and Gesture Recognition in 2015. He is the winner of 2019 IAPR YOUNG BIOMETRICS INVESTIGATOR AWARD. He is a senior member of the IEEE.

Stan Z. Li received the BEng degree from Hunan University, China, the MEng degree from National University of Defense Technology, China, and the PhD degree from Surrey University, United Kingdom. He is currently a chair professor in Westlake University and a guest professor of Center for Biometrics and Security Research (CBSR), Institute of Automation, Chinese Academy of Sciences (CASIA). He was with Microsoft Research Asia as a researcher from 2000 to 2004. Prior to that, he was an associate professor in the Nanyang Technological University, Singapore. His research interests include pattern recognition and machine learning, image and vision processing, face recognition, biometrics, and intelligent video surveillance. He has published more than 300 papers in international journals and conferences, and authored and edited eight books. He was an associate editor of the IEEE Transactions on Pattern Analysis and Machine Intelligence and is acting as the editor-in-chief for the Encyclopedia of Biometrics. He served as a program co-chair for the International Conference on Biometrics 2007, 2009, 2013, 2014, 2015, 2016 and 2018, and has been involved in organizing other international conferences and workshops in the fields of his research interest. He was elevated to IEEE fellow for his contributions to the fields of face recognition, pattern recognition and computer vision and he is a member of the IEEE Computer Society.