

QueryDet: Cascaded Sparse Query for Accelerating High-Resolution Small Object Detection

Chenhongyi Yang*
University of Edinburgh
chenhongyi.yang@ed.ac.uk

Zehao Huang
TuSimple
zehaohuang18@gmail.com

Naiyan Wang
TuSimple
winsty@gmail.com

Abstract

While general object detection with deep learning has achieved great success in the past few years, the performance and efficiency of detecting small objects are far from satisfactory. The most common and effective way to promote small object detection is to use high-resolution images or feature maps. However, both approaches induce costly computation since the computational cost grows squarely as the size of images and features increases. To get the best of two worlds, we propose QueryDet that uses a novel query mechanism to accelerate the inference speed of feature-pyramid based object detectors. The pipeline composes two steps: it first predicts the coarse locations of small objects on low-resolution features and then computes the accurate detection results using high-resolution features sparsely guided by those coarse positions. In this way, we can not only harvest the benefit of high-resolution feature maps but also avoid useless computation for the background area. On the popular COCO dataset, the proposed method improves the detection mAP by 1.0 and mAP-small by 2.0, and the high-resolution inference speed is improved to $3.0\times$ on average. On VisDrone dataset, which contains more small objects, we create a new state-of-the-art while gaining a $2.3\times$ high-resolution acceleration on average. Code is available at <https://github.com/ChenhongyiYang/QueryDet-PyTorch>.

1. Introduction

With the recent advances of deep learning [15, 53], visual object detection has achieved massive improvements in both performance and speed [3, 12, 26, 27, 29, 37, 39, 49]. It has become the foundation for widespread applications, such as autonomous driving and remote sensing. However, detecting small objects is still a challenging problem. There is a large performance gap between small and normal scale objects. Taking RetinaNet [27], one of the state-of-the-art

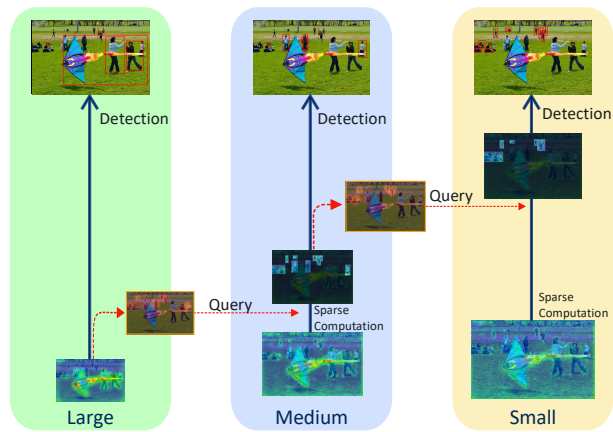


Figure 1. QueryDet achieves highly effective small object detection in high-resolution features. The locations (query keys) where small objects might exist are first predicted in the low-resolution features, and a sparse feature map (query values) is constructed using high-resolution features in those locations. Finally, a sparse detection head is used to output the detected boxes. This paradigm is applied in a cascaded manner, enabling fast and accurate small object detection.

object detectors, as an example, it achieves 44.1 and 51.2 mAP on objects with medium and large sizes but only obtains 24.1 mAP on small objects on COCO [28] test-dev set. Such degradation is mainly caused by three factors: 1) the features that highlight the small objects are extinguished because of the down-sampling operations in the backbone of Convolutional Neural Networks (CNN); hence the features of small objects are often contaminated by noise in the background; 2) the receptive field on low-resolution features may not match the size of small objects as pointed in [25]; 3) localizing small objects is more difficult than large objects because a small perturbation of the bounding box may cause a significant disturbance in the Intersection over Union (IoU) metric.

Small object detection can be improved by scaling the size of input images or reducing the down-sampling rate of CNN to maintain high-resolution features, as they in-

*Work done when working as a full-time research intern at TuSimple.

crease the effective resolution in the resulted feature map. However, merely increasing the resolution of feature maps can incur considerable computation costs. Several works [1, 26, 29] proposed to build a feature pyramid by reusing the multi-scale feature maps from different layers of a CNN to address this issue. Objects with various scales are handled on different levels: large objects tend to be detected on high-level features, while small objects are usually detected on low levels. The feature pyramid paradigm saves the computation cost of maintaining high-resolution feature maps from shallow to deep in the backbone. Nevertheless, the computation complexity of detection heads on low-level features is still enormous. For example, adding an extra pyramid level P_2 into RetinaNet will bring about 300% more computation (FLOPs) and memory cost in the detection head; hence severely lowering down the inference speed from 13.6 FPS to 4.85 FPS on NVIDIA 2080Ti GPU.

In this paper, we propose a simple and effective method, QueryDet, to save the detection head’s computation while promoting the performance of small objects. The motivation comes from two key observations: 1) the computation on low-level features is highly redundant. In most cases, the spatial distribution of small objects is very sparse: they occupy only a few portions of the high-resolution feature maps; hence a large amount of computation is wasted. 2) The feature pyramids are highly structured. Though we cannot accurately detect the small objects in low-resolution feature maps, we can still infer their existence and rough locations with high confidence.

A natural idea to utilize these two observations is that we can only apply the detection head to small objects’ spatial locations. This strategy requires locating the rough location of small objects at a low cost and sparse computation on the desired feature map. In this work, we present QueryDet that is based on a novel query mechanism Cascade Sparse Query (CSQ), as illustrated in Fig. 1. We recursively predict the rough locations of small objects (queries) on lower resolution feature maps and use them to guide the computations in higher resolution feature maps. With the help of sparse convolution [13, 55], we significantly reduce the computation cost of detection heads on low-level features while keeping the detection accuracy for small objects. Note that our approach is designed to save the computation spatially, so it is compatible with other accelerating methods like lightweighted backbones [44], model pruning [16], model quantization [51], and knowledge distillation [5].

We evaluate our QueryDet on the COCO detection benchmark [28] and a challenging dataset, VisDrone [59], that contains a large amount of small objects. We show our method can significantly accelerate inference while improving the detection performance. In summary, we make two main contributions:

- We propose QueryDet, in which a simple and effective

Cascade Sparse Query (CSQ) mechanism is designed. It can reduce the computation costs of all feature pyramid based object detectors. Our method can improve the detection performance for small objects by effectively utilizing high-resolution features while keeping fast inference speed.

- On COCO, QueryDet improves the RetinaNet baseline by 1.1 AP and 2.0 AP_S by utilizing high-resolution features, and the high-resolution detection speed is improved by $3.0\times$ on average when CSQ is adopted. On VisDrone, we advance the state-of-the-art results in terms of the detection mAP and enhance the high-resolution detectopm speed by $2.3\times$ on average.

2. Related Works

Object Detection. Deep Learning based object detection can be mainly divided into two streams: the two-stage detectors [2, 11, 12, 26, 39] and the one-stage detectors [17, 29, 35–37, 58] pioneered by YOLO. Generally speaking, two-stage methods tend to be more accurate than one-stage methods because they use the RoIAlign operation [14] to align an object’s features explicitly. However, the performance gap between these two streams is narrowed recently. RetinaNet [27] is the first one-stage anchor-based detector that matches the performance of two-stage detectors. It uses feature pyramid network (FPN) [26] for multi-scale detections and proposes FocalLoss to handle the foreground-background imbalance problem in dense training. Recently, one-stage anchor-free detectors [7, 7, 21, 23, 45, 56] have attracted academic attentions because of their simplicity. In this paper, we implement our QueryDet based on RetinaNet and FCOS [45] to show its effectiveness and generalization ability.

Small Object Recognition. Small object recognition, like detection and segmentation, is a challenging computer vision task because of low-resolution features. To tackle this problem, a large amount of works have been proposed. These methods can be mainly categorized into four types: 1) increasing the resolution of input features [1, 10, 22, 24, 26, 29, 41, 48]; 2) oversampling and strong data augmentation [20, 29, 60]; 3) incorporating context information [4, 6, 57], and 4) scale-aware training [25, 26, 42, 43].

Spatial Redundancy. Several methods have used sparse computation to utilize the spatial redundancy of CNNs in different ways to save computation costs. Perforated-CNN [9] generates masks with different deterministic sampling methods. Dynamic Convolution [47] uses a small gating network to predict pixel masks, and [54] proposes a stochastic sampling and interpolation network. Both of

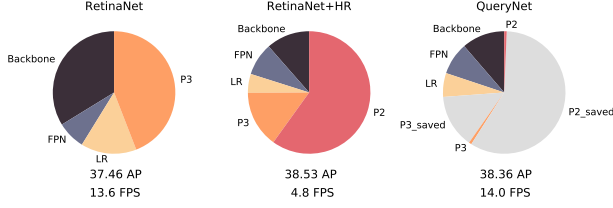


Figure 2. The FLOPs distribution of different module when using ResNet-50 backbone. In RetinaNet, the computational cost on the high-resolution P_3 accounts for 43% of the total cost; when the higher-resolution P_2 is added, they together account for 74% of the total cost. Our QueryDet can effectively reduce the computation on those features by 99%, leading to a fast inference speed and keeping a high detection accuracy. Note LR stands for the low-resolution P_4 to P_7 .

them adopt Gumbel-Softmax [18] and a sparsity loss for the training of sparse masks. On the other hand, the Spatially Adaptive Computation Time (SACT) [8] predicts a halting score for each spatial position that is supervised by a proposed ponder cost and the task-specific loss function. SB-Net [38] adopts an offline road map or a mask to filter out ignored region. Unlike these methods, our QueryDet focuses on objects’ scale variation and simply adopts the provided ground-truth bounding box for supervision. Another stream of works adopts a two-stage framework: glance and focus for adaptive inference. [50] selects small regions from the original input image by reinforcement learning and processes these regions with a dynamic decision process. [46] adopts a similar idea on object detection task. One similar work to our QueryDet is AutoFocus [33]. AutoFocus first predicts and crop region of interest in coarse scales, then scaled to a larger resolution for final predictions. Compared with AutoFocus, our QueryDet is more efficient since the “focus” operation is conducted on feature pyramids other than image pyramids, which reduces the redundant computation in the backbone.

3. Methods

In this section, we describe our QueryDet for accurate and fast small object detection. We illustrate our approach based on RetinaNet [27], a popular anchor-based dense detector. Note that our approach is not limited to RetinaNet, as it can be applied to any one-stage detectors and the region proposal network (RPN) in two-stage detectors with FPN. We will first revisit RetinaNet and analyze the computational cost distribution of different components. Then we will introduce how we use the proposed Cascade Sparse Query to save computation costs during inference. Finally, the training details will be presented.

3.1. Revisiting RetinaNet

RetinaNet has two parts: a backbone network with FPN that outputs multi-scale feature maps and two detection heads for classification and regression. When the size of input image is $H \times W$, the sizes of FPN features are $\mathcal{P} = \{P_l \in \mathbb{R}^{H' \times W' \times C}\}$. Here l indicates the pyramid level and (H', W') is usually equals to $(\lfloor \frac{H}{2^l} \rfloor, \lfloor \frac{W}{2^l} \rfloor)$ in a typical FPN implementation. The detection heads consist of four 3×3 convolution layers, followed by an extra 3×3 convolution layer for final prediction. For parameter efficiency, different feature levels share the same detection heads (parameters). However, the computation costs are highly imbalanced across different layers: the FLOPs of detection heads from P_7 to P_3 increases in quadratic order by the scaling of feature resolutions. As shown in Figure 2, the P_3 head occupies nearly half FLOPs while the cost of low-resolution features P_4 to P_7 only accounts for 15%. Thus, if we want to extend the FPN to P_2 for better small object performance, the cost is unaffordable: high-resolution P_2 and P_3 will occupy 75% of the overall cost. In the following, we describe how our QueryDet reduce the computation on high-resolution features and promote the inference speed of RetinaNet, even with an extra high-resolution P_2 .

3.2. Accelerating Inference by Sparse Query

In the design of modern FPN based detectors, small objects tend to be detected from high-resolution low-level feature maps. However, as the small objects are usually sparsely populated in space, the dense computation paradigm on high-resolution feature maps is highly inefficient. Inspired by this observation, we propose a coarse-to-fine approach to reduce the computation cost of low-level pyramids: First, the rough locations of small objects are predicted on coarse feature maps, and then the corresponding locations on fine feature maps are intensively computed. This process can be viewed as a query process: the rough locations are query keys, and the high-resolution features used to detect small objects are query values; thus we call our approach **QueryDet**. The whole pipeline of our method is presented in Figure 3.

To predict the coarse locations of small objects, we add a query head that is parallel to the classification and regression heads. The query head receives feature map P_l with stride 2^l as input, and output a heatmap $V_l \in \mathbb{R}^{H' \times W'}$ with $V_l^{i,j}$ indicating the probability of that the grid (i, j) contains a small object. During training, we define small objects on each level as objects whose scale is smaller than a pre-defined threshold s_l . Here we set s_l to the minimum anchor scale on P_l for simplicity, and for anchor-free detectors it is set to the minimum regression range on P_l . For a small object o , we encode the target map for Query Head by computing distance between its center location (x_o, y_o) and every location on the feature map, and set locations whose

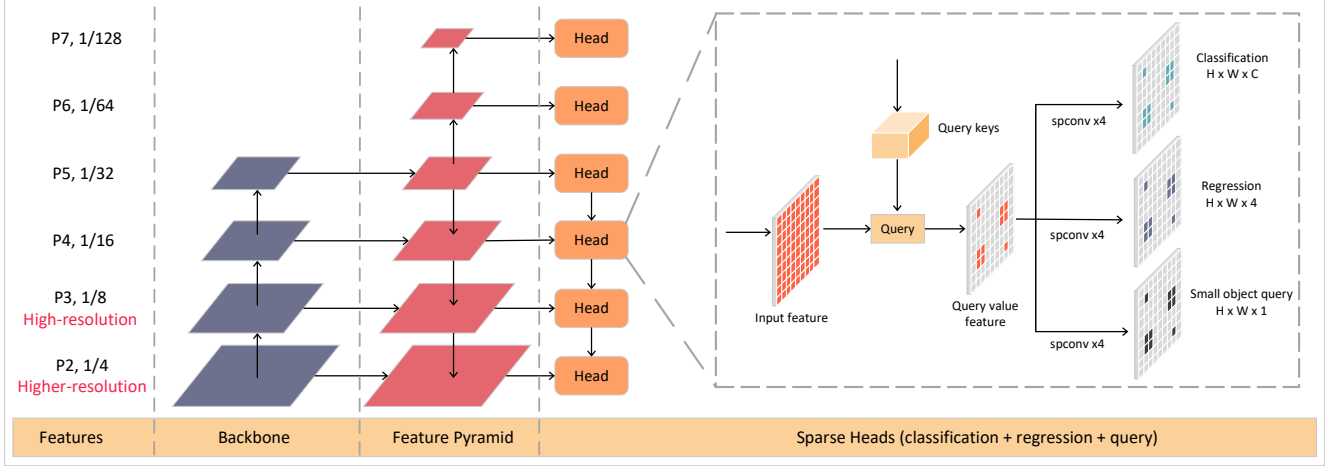


Figure 3. The whole pipeline of the proposed QueryDet. The image is fed into the backbone and Feature Pyramid Network (FPN) to produce a series of feature maps of different resolutions. Beginning from the query start layer (P_5 in this image), each layer receives a set of key positions from previous layer and a query operation is applied to generate the sparse value feature map. Then the sparse detection head and the sparse query head predict the detected boxes of the corresponding scales and key positions for the next layer.

distance is smaller than s_l to 1, otherwise 0. Then the Query Head is trained using FocalLoss [27]. During inference, we choose the locations whose predicted scores are larger than a threshold σ as queries. Then q_l^o will be mapped to its four nearest neighbors on P_{l-1} as key positions $\{k_{l-1}^o\}$:

$$\{k_{l-1}^o\} = \{(2x_l^o + i, 2y_l^o + j), \forall i, j \in \{0, 1\}\}. \quad (1)$$

All $\{k_{l-1}^o\}$ on P_{l-1} are collected to form the key position set $\{k_{l-1}\}$. Then the three heads will only process those positions to detect objects and compute next level's queries. Specifically, we extract features from P_{l-1} using $\{k_{l-1}\}$ as indices to construct a sparse tensor P_{l-1}^v that we call value features. Then the sparse convolution (spconv) [13] kernels are built using weights of the 4-conv dense heads to compute results on layer $l-1$.

To maximize the inference speed, we apply the queries in a cascade manner. In particular, the queries for P_{l-2} would only be generated from $\{k_{l-1}\}$. We name this paradigm as **Cascade Sparse Query (CSQ)** as illustrated in Figure 1. The benefit of our CSQ is that we can avoid generating the queries $\{q_l\}$ from a single P_l , which leads to exponentially increasing size of corresponding key position k_l during query mapping as l decreases.

3.3. Training

We keep the training of classification and regression heads as same as in the original RetinaNet [27]. For the query head, we train it using FocalLoss [27] with the generated binary target map: Let the ground-truth bounding box of a small object o on P_l be $b_l^o = (x_l^o, y_l^o, w_l^o, h_l^o)$. We first compute the minimum distance map D_l between each feature position (x, y) on P_l and all the small ground-truth

centers $\{(x_l^o, y_l^o)\}$:

$$D_l[x][y] = \min_o \{\sqrt{(x - x_l^o)^2 + (y - y_l^o)^2}\}, \quad (2)$$

Then the ground truth query map V_l^* is defined as

$$V_l^*[x][y] = \begin{cases} 1 & \text{if } D_l[x][y] < s_l \\ 0 & \text{if } D_l[x][y] \geq s_l \end{cases}. \quad (3)$$

For each level P_l , the loss function is defined as following:

$$\mathcal{L}_l(U_l, R_l, V_l) = \mathcal{L}_{FL}(U_l, U_l^*) + \mathcal{L}_r(R_l, R_l^*) + \mathcal{L}_{FL}(V_l, V_l^*) \quad (4)$$

where U_l , R_l , V_l are the classification output, regressor output and the query score output, and U_l^* , R_l^* , and V_l^* are their corresponding ground-truth maps; \mathcal{L}_{FL} is the focal loss and \mathcal{L}_r is the bounding box regression loss, which is smooth l_1 loss [11] in the original RetinaNet. The overall loss is:

$$\mathcal{L}_{all} = \sum_l \beta_l * \mathcal{L}_l. \quad (5)$$

Here we re-balance the loss of each layer by β_l . The reason is that as we add the higher-resolution features like P_2 , the distribution of the training samples has significantly changed. The total number of training samples on P_2 is even larger than the total number of training sample cross P_3 to P_7 . If we don't reduce the weight of it, the training will be dominated by small objects. Thus, we need to re-balance the loss of different layers to make the model simultaneously learn from all layers.

3.4. Relationships with Related Work

Note that though our method bears some similarities with two-stage object detectors using RPN, they differ in

the following aspects: 1), we only compute classification results in the coarse prediction, while RPN computes both classification and regression. 2), RPN is computed on all levels of full feature maps while the computation of our QueryDet is sparse and selective. 3), two-stage methods rely on operations like RoIAlign [14] or RoIPooling [11] to align the features with the first stage proposal. Nevertheless, they are not used in our approach since we do not have box output in the coarse prediction. It is worth noting that our proposed method is compatible with the FPN based RPN, so QueryDet can be incorporated into two-stage detectors to accelerate proposal generation.

Another closely related work is PointRend [19], which computes high-resolution segmentation maps using very few adaptive selected points. The main differences between our QueryDet and PointRend are: 1) how the queries are generated and 2) how sparse computation is applied. For the first difference, PointRend selects the most uncertain regions based on the predicted score at each location, while we directly add an auxiliary loss as supervision. Our experiments show this simple method can generate high recall predictions and improve the final performance. As for the second, PointRend uses a multi-layer perceptron for per-pixel classification. It only requires the features from a single location in high-resolution feature maps, thus can be easily batched for high efficiency. On the other hand, as object detection requires more context information for accurate prediction, we use sparse convolution with 3×3 kernels.

4. Experiments

We conduct quantitative experiments on two object detection datasets: COCO [28] and VisDrone [59]. COCO is the most widely used dataset for general object detection; VisDrone is a dataset specialized to drone-shot image detection, in which small objects dominate the scale distribution.

4.1. Implementation Details

We implement our approach based on PyTorch [34] and the Detectron2 toolkit [52]. All models are trained on 8 NVIDIA 2080Ti GPUs. For COCO, we follow the common training practices: We adopt the standard $1 \times$ schedule and the default data augmentation in Detectron2. Batch size is set to 16 with the initial learning rate of 0.01. The weights β_l used to re-balance the loss between different layers are set to linearly growing from 1 to 3 across P_2 to P_7 . For VisDrone, following [30], we equally split one image into four non-overlapping patches and process them independently during training. We train the network for 50k iterations with an initial learning rate of 0.01, and decay the learning rate by 10 at 30k and 40k iteration. The re-balance weights β_l are set to linearly growing from 1 to 2.6. For both datasets, we freeze all the batch normalization (BN) layers in the backbone network during training and we did

Method	CSQ	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
RetinaNet	-	37.46	56.90	39.94	22.64	41.48	48.04	13.60
RetinaNet (3x)	-	38.76	58.27	41.24	22.89	42.53	50.02	13.83
QueryDet	×	38.53	59.11	41.12	24.64	41.97	49.53	4.85
QueryDet	✓	38.36	58.78	40.99	24.33	41.97	49.53	14.88
QueryDet (3x)	×	39.47	59.93	42.11	25.24	42.37	51.12	4.89
QueryDet (3x)	✓	39.34	59.69	41.98	24.91	42.38	51.12	15.94

Table 1. Comparison of accuracy (AP) and speed (FPS) of our QueryDet and the baseline RetinaNet on COCO *mini-val* set.

Method	CSQ	AP	AP ₅₀	AP ₇₅	AR ₁	AR ₁₀	AR ₁₀₀	AR ₅₀₀	FPS
RetinaNet	-	26.21	44.90	27.10	0.52	5.35	34.63	37.21	2.63
QueryDet	×	28.35	48.21	28.78	0.51	5.96	36.48	39.42	1.16
QueryDet	✓	28.32	48.14	28.75	0.51	5.96	36.45	39.35	2.75

Table 2. Comparison of detection accuracy (AP) and speed (FPS) of our QueryDet and the baseline RetinaNet on VisDrone validation set.

not add BN layers in the detection heads. Mixed precision training [32] is used in all experiments to save GPU memory. The query threshold σ is set to 0.15 and we start query from P_4 . Without specified description, our method is constructed on RetinaNet with ResNet-50 backbone.

4.2. Effectiveness of Our Approach

In Table 1, we compare the mean average precision (mAP) and average frame per second (FPS) between our methods and the baseline RetinaNet on COCO. The baseline runs at 13.6 FPS, and gets 37.46 overall AP and 22.64 AP_S for small objects, which is slightly higher than the results in the original paper [27]. With the help of high-resolution features, our approach achieves 38.53 AP and 24.64 AP_S, improving the AP and AP_S by 1.1 and 2.0. The results reveal the importance of using high-resolution features when detecting small objects. However, incorporating such a high-resolution feature map significantly decrease the inference speed to 4.85 FPS. When adopting our Cascade Sparse Query (CSQ), the inference speed is enhanced to 14.88 FPS, becoming even faster than the baseline RetinaNet that does not use the higher-resolution P_2 , while the performance loss is negligible. Additionally, Figure 2 shows how our CSQ save the computational cost. Compared with the RetinaNet with higher-resolution P_2 , in which P_3 and P_2 account for 74% of the total FLOPs, our CSQ successfully reduce those costs to around 1%. The reason is that in QueryDetall computations on high-resolution P_3 and P_2 are carried out on locations around the sparsely distributed small objects. These results sufficiently demonstrate the effectiveness of our method. We also show the results of $3 \times$ training schedule in Table 1. The stronger baseline does not weaken our improvement but brings more significant acceleration. We owe it to the stronger Query Head as the small object estimation becomes more accurate.

In VisDrone, as illustrated in Table 2, the discoveries are

HR	RB	QH	CSQ	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
				37.46	56.90	39.94	22.64	41.48	48.04	13.60
✓				36.10	56.39	38.17	21.94	39.91	45.25	4.83
	✓			37.66	57.57	40.37	22.03	41.86	49.10	13.60
✓	✓			38.11	58.48	40.85	23.06	41.53	49.36	4.83
✓	✓	✓		38.53	59.11	41.12	24.64	41.97	49.53	4.85
✓	✓	✓	✓	38.36	58.78	40.99	24.33	41.97	49.53	14.88

Table 3. Ablation studies on COCO *mini-val* set. **HR** stands for using of high-resolution features; **RB** stands for the loss re-balance between FPN layers; **QH** stands for whether to add QueryHead that provides extra objectiveness supervision.

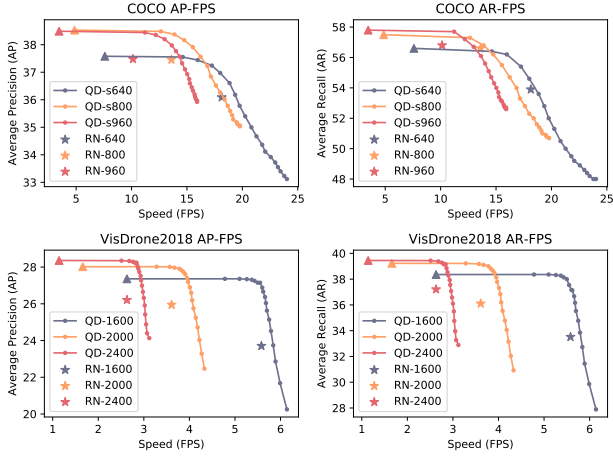


Figure 4. The speed and accuracy (AP and AR) trade-off with input images with different sizes on COCO and VisDrone. The trade-off is controlled by the query threshold σ . The leftmost marker (the \blacktriangle marker) of each curve stands for the result when Cascade Sparse Query is not applied. QD stands for QueryDet and RN stands for RetinaNet.

Start Layer	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
No Query	38.53	59.11	41.12	24.64	41.97	49.53	4.86
P ₆	37.91	57.98	40.51	23.18	42.02	49.53	13.42
P ₅	38.22	58.55	40.86	23.65	42.00	49.53	13.92
P ₄	38.36	58.78	40.99	24.33	41.97	49.53	14.88
P ₃	38.45	58.94	41.07	24.50	41.93	49.52	11.51

Table 4. Investigation of the best starting layer of our CSQ on MS-COCO *mini-val* set.

similar, but the results are even more significant. We improve the overall AP by 2.1 and AP₅₀ by 3.2 on this small objects oriented dataset. The inference speed is improved to $2.3\times$ from 1.16 FPS from 2.75 FPS.

4.3. Ablation Studies

We conduct ablation studies on COCO *mini-val* set to analyze how each component affects the detection accuracy and speed in Table 3. Our retrained RetinaNet achieves 37.46 AP. When we add the high-resolution P_2 , the AP dra-

matically drops by 1.34. As we discussed in Section 3.3, this problem is caused by the distribution shift in the training samples after adding P_2 . Then we re-balance the loss of those layers. The result is improved to 38.11, mostly addressing this problem. Interestingly, the re-balancing strategy only gives us a minor AP enhancement (0.2) when adopting on the original baseline, suggesting that the loss re-balance is more critical in the high-resolution scenario. Then we add our Query Head into the network, through which we get a further performance gain of 0.42 AP and 1.58 AP_S, pushing the total AP and AP_S to 38.53 and 24.64, verifying the effectiveness of the extra objectiveness supervision. Finally, with CSQ, the detection speed is largely improved to 14.88 FPS from 4.85 FPS, and the 0.17 loss in detection AP is negligible.

4.4. Discussions

Influence of the Query Threshold. Here we investigate the accuracy-speed trade-off in our Cascade Sparse Query. We measure the detection accuracy (AP) and detection speed (FPS) under different query thresholds σ whose role is to determine if a grid (low-resolution feature location) in the input image contains small objects. Intuitively, increasing this threshold will decrease the recall of small objects but accelerate the inference since fewer locations are considered. The accuracy-speed trade-off with different input sizes are presented in Figure 4. We increase σ by 0.05 sequentially for adjacent data markers in one curve, and the leftmost marker denotes the performance when CSQ is not applied. We observe that even a very low threshold (0.05) can bring us a massive speed improvement. This observation validates the effectiveness of our approach. Another observation is about the gap between the AP upper bound and lower bound of different input resolutions. This gap is small for large size images, but huge for small size images, which indicates that for higher-resolution input our CSQ can guarantee a good AP lower bound even if the query threshold is set to high.

Which layer to start query? In our Cascade Sparse Query, we need to decide the starting layer, above which we run conventional convolutions to get the detection results for

Query Method	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
No Query	38.53	59.11	41.12	24.64	41.97	49.53	4.86
CQ	38.31	58.73	40.98	24.25	41.98	49.53	10.49
CCQ	38.32	58.75	40.98	24.26	41.98	49.53	8.73
CSQ (ours)	38.36	58.78	40.99	24.33	41.97	49.53	14.88

Table 5. Comparison of different query methods on COCO *mini-val* set. We compare our CSQ and Crop Query (CQ) and Complete Convolution Query (CCQ).

Context	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
No Query	38.53	59.11	41.12	24.64	41.97	49.53	4.86
1x1	38.25	58.60	40.87	23.88	41.97	49.53	14.09
3x3	38.30	58.66	40.94	24.14	41.97	49.53	14.06
5x5	38.36	58.72	40.98	24.18	41.97	49.53	14.00
7x7	38.37	58.73	40.98	24.30	41.97	49.53	13.77
9x9	38.37	58.73	40.98	24.30	41.97	49.53	13.42
11x11	38.38	58.755	40.99	24.33	41.97	49.53	13.11

Table 6. Comparison of detection AP and speed when using different amount of context information on MS-COCO *mini-val* set. The context is defined as a patch with various size around the queried position.

Backbone	Model	CSQ	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
MobileNet V2	RN	-	26.72	43.17	28.17	15.27	29.28	34.51	17.75
	QD	×	29.16	46.20	30.95	16.14	31.26	38.66	5.31
	QD	✓	28.94	45.79	30.71	15.74	31.26	38.66	21.66
ShuffleNet V2	RN	-	23.04	38.32	23.75	12.01	25.50	35.16	17.45
	QD	×	26.07	42.34	27.30	13.20	28.03	36.23	5.26
	QD	✓	25.85	41.96	27.08	12.81	28.05	36.23	20.02

Table 7. Results on different backbone networks. **RN** and **QD** stand for RetinaNet and QueryDet, respectively.

	CSQ	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
FCOS	-	38.37	57.63	41.03	22.34	41.95	48.96	17.06
QueryDet (FCOS)	×	40.05	58.69	43.46	25.52	43.43	50.69	7.92
QueryDet (FCOS)	✓	39.49	57.97	42.82	24.81	43.45	50.69	14.40

Table 8. Performance and speed of our QueryDet (FCOS) and its baseline model on COCO *mini-val* set.

large objects. The reason we do not start our CSQ from the lowest resolution layer are in two folds: 1) The normal convolution operation is very fast for the low-resolution features, thus the time saved by CSQ cannot compensate the time needed to construct the sparse feature map; 2) It is hard to distinguish small objects on feature maps with very low resolution. The results are presented in Table 4. We find that the layer that gets the highest inference speed is P_4 , which validates that querying from very high-level layers such as P_5 and P_6 would cause loss of speed. We observe that the AP loss gradually increases as the starting layer becomes higher, suggesting the difficulty for the network to find small objects in very low-resolution layers.

What is the best way to use queries? We demonstrate the high efficiency of our Cascade Sparse Query. We propose

CSQ	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L	FPS
×	38.47	59.44	41.73	22.98	41.90	49.55	17.57
✓	38.20	58.88	41.50	22.23	41.91	49.55	19.03

Table 9. Performance and speed of using our CSQ in Faster R-CNN on COCO *mini-val* set.

two alternative query operations for comparison. The first Crop Query (CQ), in which the corresponding regions indicated by queries are cropped from the high-resolution features for subsequent computations. Note this type of query is similar to the AutoFocus [33] approach. Another one is Complete Convolution Query (CCQ) where we use regular convolutions to compute the full feature map for each layer, but only extract results from queried positions for post-processing. For CQ, we crop a 11×11 patch from the feature map, which is chosen to fit the receptive field of the five 3×3 consecutive convolutions in the detection heads. We present the results in Table 5. Generally speaking, all three methods can successfully accelerate the inference with negligible AP loss. Among them, our CSQ can achieve the fastest inference speed.

How much context do we need? To apply our CSQ, we need to construct a sparse feature map where only the positions of small objects are activated. We also need to activate the context area around the small objects to avoid decreasing accuracy. However, in practice, we found that too much context cannot improve the detection AP but only slow down the detection speed; on the other hand, too little context would severely decrease the detection AP. In this section, we explore how much context do we need to balance the speed-accuracy trade-off. Here, the context is defined as a patch with various sizes around the queried position, where our sparse detection head would also process the features within the patch. The result is reported in Table 6. From it we conclude that a 5×5 patch can brings us enough context to detect a small object. Although more context brings a small AP improvement, the accelerating effect of our CSQ is negatively affected, while fewer context cannot guarantee a high detection AP.

Results on Light-weight Backbones. As we claim in Section 1, our method can be incorporated with light-weighted backbones to gain more speed improvement. Also, as our CSQ aims to accelerate the computation in the detection head, so the overall acceleration is more obvious when using such backbones, because the inference time for backbone network becomes less. We report the results with different light-weight backbones in Table 7. Specifically, the speed is on average improved to $4.1 \times$ for high-resolution detection with MobileNet V2 [40] and $3.8 \times$ with Shuf-

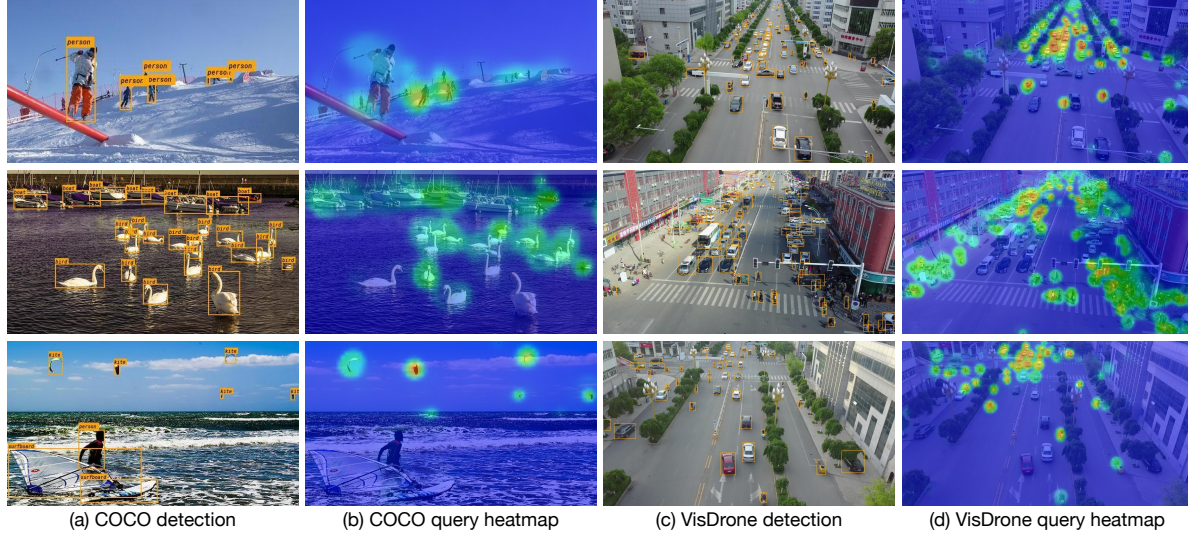


Figure 5. Visualization of the detection results and the query heatmap for small objects of our QueryDet on MS-COCO and VisDrone2018 datasets. We remove class labels for VisDrone2018 to better distinguish the small bounding boxes.

fleNet V2 [31], which validates that our approach is ready to deploy on edge devices for real-time applications such as autonomous driving vehicles for effective small object detections.

Results on Anchor-Free Detectors. QueryDet can be applied to any FPN based detector to accelerate high-resolution detection. Thus, we apply QueryDet on FCOS, a state-of-the-art anchor-free detector, and report the COCO results in Table 8. It can be concluded that QueryDet improves APs with the help of high-resolution features, and when Cascade Sparse Query (CSQ) is adopted, the high-resolution speed is improved by $1.8\times$ on average, validating the universality of the proposed approach.

Effectiveness on Two-stage Detectors Our CSQ can also be applied to FPN based two-stage detectors to reduce computation cost in the high-resolution layers in RPN. To verify this claim, we apply CSQ to the Faster R-CNN detector [39]. In our implementation, the inputs to RPN are from P_2 to P_6 and we start query from P_4 . We modify the RPN structure to let it have 3 *conv* layers instead of 1 layer in the normal implementation, which is followed by 3 branches for objectiveness classification, bounding box regression and query key computation. The former two branches are trained following common practice [39], and the query branch is trained by Focal Loss with $\gamma = 1.2$ and $\alpha = 0.25$. During inference, we set the query threshold to 0.15. As shown in Table 9, our Faster R-CNN achieves 38.47 overall AP and 22.98 AP_s with 17.57 FPS. When CSQ is utilized, the inference speed is improved to 19.03 FPS with a minor loss in AP_s . The results verify the effectiveness of our approach in accelerating two stage detectors.

Note that in two-stage detectors our CSQ can not only save time for the dense computation on in RPN, it can reduce the number of RoIs that are fed into the second stage.

4.5. Visualization and Failure Cases

In Figure 5, we visualize the detection results and the query heatmaps for small objects on COCO and VisDrone. From the heatmaps, it can be seen that our query head can successfully find the coarse positions of the small objects, enabling our CSQ to detect them effectively. Additionally, through incorporating high-resolution features, our method can detect small objects very accurately.

We also show two typical failure cases of our approach: 1) Even if the coarse position of small objects is correctly extracted by the query head, the detection head may fail to localize them (the second image of VisDrone); 2) Positions of large objects is falsely activated, causing the detection head to process useless positions and hence slowing down the speed (the first image of COCO).

5. Conclusion

We propose QueryDet that uses a novel query mechanism Cascade Sparse Query (CSQ) to accelerate the inference of feature pyramid-based dense object detectors. QueryDet enables object detectors the ability to detect small objects at low cost and easily deploy, making it practical to deploy them on real-time applications such as autonomous driving. For future work, we plan to extend QueryDet to the more challenging 3D object detection task that takes LiDAR point clouds as input, where the 3D space is generally sparser than 2D image, and computational resources are more intense for the costly 3D convolution operations.

References

- [1] Zhaowei Cai, Quanfu Fan, Rogerio S Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV*. Springer, 2016. [2](#)
- [2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018. [2](#)
- [3] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. *ECCV*, 2020. [1](#)
- [4] Chenyi Chen, Ming-Yu Liu, Oncel Tuzel, and Jianxiong Xiao. R-CNN for small object detection. In *ACCV*. Springer, 2016. [2](#)
- [5] Guobin Chen, Wongun Choi, Xiang Yu, Tony Han, and Manmohan Chandraker. Learning efficient object detection models with knowledge distillation. In *NeurIPS*, 2017. [2](#)
- [6] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4), 2017. [2](#)
- [7] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *ICCV*, 2019. [2](#)
- [8] Michael Figurnov, Maxwell D Collins, Yukun Zhu, Li Zhang, Jonathan Huang, Dmitry Vetrov, and Ruslan Salakhutdinov. Spatially adaptive computation time for residual networks. In *CVPR*, 2017. [3](#)
- [9] Mikhail Figurnov, Aizhan Ibrahimova, Dmitry P Vetrov, and Pushmeet Kohli. Perforatedcnns: Acceleration through elimination of redundant convolutions. In *NeurIPS*, 2016. [2](#)
- [10] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Amrith Tyagi, and Alexander C Berg. DSSD: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017. [2](#)
- [11] Ross Girshick. Fast r-cnn. In *ICCV*, 2015. [2](#), [4](#), [5](#)
- [12] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. [1](#), [2](#)
- [13] Benjamin Graham and Laurens van der Maaten. Submanifold sparse convolutional networks. *arXiv preprint arXiv:1706.01307*, 2017. [2](#), [4](#)
- [14] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *ICCV*, 2017. [2](#), [5](#)
- [15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. [1](#)
- [16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017. [2](#)
- [17] Lichao Huang, Yi Yang, Yafeng Deng, and Yinan Yu. DenseBox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015. [2](#)
- [18] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *ICLR*, 2017. [3](#)
- [19] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *CVPR*, 2020. [5](#)
- [20] Mate Kisantal, Zbigniew Wojna, Jakub Murawski, Jacek Naruniec, and Kyunghyun Cho. Augmentation for small object detection. *arXiv preprint arXiv:1902.07296*, 2019. [2](#)
- [21] Tao Kong, Fuchun Sun, Huaping Liu, Yuning Jiang, Lei Li, and Jianbo Shi. FoveaBox: Beyond anchor-based object detection. *IEEE Transactions on Image Processing*, 29, 2020. [2](#)
- [22] Tao Kong, Anbang Yao, Yurong Chen, and Fuchun Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *CVPR*, 2016. [2](#)
- [23] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *ECCV*, 2018. [2](#)
- [24] Jianan Li, Xiaodan Liang, Yunchao Wei, Tingfa Xu, Jiashi Feng, and Shuicheng Yan. Perceptual generative adversarial networks for small object detection. In *CVPR*, 2017. [2](#)
- [25] Yanghao Li, Yuntao Chen, Naiyan Wang, and Zhaoxiang Zhang. Scale-aware trident networks for object detection. In *ICCV*, 2019. [1](#), [2](#)
- [26] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *CVPR*, 2017. [1](#), [2](#)
- [27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017. [1](#), [2](#), [3](#), [4](#), [5](#)
- [28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014. [1](#), [2](#), [5](#)
- [29] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*. Springer, 2016. [1](#), [2](#)
- [30] Ziming Liu, Guangyu Gao, Lin Sun, and Zhiyuan Fang. HRDNet: High-resolution Detection Network for Small Objects. *arXiv preprint arXiv:2006.07607*, 2020. [5](#)
- [31] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical Guidelines for Efficient CNN Architecture Design. In *ECCV*, 2018. [8](#)
- [32] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. *ICLR*, 2018. [5](#)
- [33] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. In *ICCV*, 2019. [3](#), [7](#)
- [34] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*, 2019. [5](#)
- [35] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016. [2](#)
- [36] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *CVPR*, 2017. [2](#)
- [37] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. [1](#), [2](#)

- [38] Mengye Ren, Andrei Pokrovsky, Bin Yang, and Raquel Urtasun. SBNNet: Sparse Blocks Network for Fast Inference. In *CVPR*, June 2018. 3
- [39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NeurIPS*, 2015. 1, 2, 8
- [40] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018. 7
- [41] Abhinav Shrivastava, Rahul Sukthankar, Jitendra Malik, and Abhinav Gupta. Beyond skip connections: Top-down modulation for object detection. *arXiv preprint arXiv:1612.06851*, 2016. 2
- [42] Bharat Singh and Larry S Davis. An analysis of scale invariance in object detection snip. In *CVPR*, 2018. 2
- [43] Bharat Singh, Mahyar Najibi, and Larry S Davis. Sniper: Efficient multi-scale training. In *NeurIPS*, 2018. 2
- [44] Mingxing Tan, Ruoming Pang, and Quoc V Le. EfficientDet: Scalable and efficient object detection. In *CVPR*, 2020. 2
- [45] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019. 2
- [46] Burak Uzkent, Christopher Yeh, and Stefano Ermon. Efficient object detection in large images using deep reinforcement learning. In *WACV*, 2020. 3
- [47] Thomas Verelst and Tinne Tuytelaars. Dynamic Convolutions: Exploiting Spatial Sparsity for Faster Inference. In *CVPR*, June 2020. 2
- [48] Jingdong Wang, Ke Sun, Tianheng Cheng, Borui Jiang, Chaorui Deng, Yang Zhao, Dong Liu, Yadong Mu, Mingkui Tan, Xinggang Wang, et al. Deep high-resolution representation learning for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2020. 2
- [49] Xinjiang Wang, Shilong Zhang, Zhuoran Yu, Litong Feng, and Wayne Zhang. Scale-equalizing Pyramid Convolution for Object Detection. In *CVPR*, 2020. 1
- [50] Yulin Wang, Kangchen Lv, Rui Huang, Shiji Song, Le Yang, and Gao Huang. Glance and Focus: a Dynamic Approach to Reducing Spatial Redundancy in Image Classification. *NeurIPS*, 2020. 3
- [51] Yi Wei, Xinyu Pan, Hongwei Qin, Wanli Ouyang, and Junjie Yan. Quantization mimic: Towards very tiny cnn for object detection. In *ECCV*, 2018. 2
- [52] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. <https://github.com/facebookresearch/detectron2>, 2019. 5
- [53] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. In *CVPR*, 2017. 1
- [54] Zhenda Xie, Zheng Zhang, Xizhou Zhu, Gao Huang, and Stephen Lin. Spatially Adaptive Inference with Stochastic Feature Sampling and Interpolation. *ECCV*, 2020. 2
- [55] Yan Yan, Yuxing Mao, and Bo Li. Second: Sparsely embedded convolutional detection. *Sensors*, 18(10), 2018. 2
- [56] Ze Yang, Shaohui Liu, Han Hu, Liwei Wang, and Stephen Lin. Reppoints: Point set representation for object detection. In *ICCV*, 2019. 2
- [57] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 2
- [58] Shifeng Zhang, Cheng Chi, Yongqiang Yao, Zhen Lei, and Stan Z Li. Bridging the gap between anchor-based and anchor-free detection via adaptive training sample selection. In *CVPR*, 2020. 2
- [59] Pengfei Zhu, Longyin Wen, Dawei Du, Xiao Bian, Haibin Ling, Qinghua Hu, Qinqin Nie, Hao Cheng, Chenfeng Liu, Xiaoyu Liu, et al. Visdrone-det2018: The vision meets drone object detection in image challenge results. In *ECCV*, 2018. 2, 5
- [60] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. *arXiv preprint arXiv:1906.11172*, 2019. 2