Received: 14 September 2020 | Revised: 4 February 2021 | Accepted: 9 March 2021

DOI: 10.1002/cpe.6307

RESEARCH ARTICLE

WILEY

Accelerating real-time object detection in high-resolution video surveillance

Yuefeng Wang[1] | Kuang Mao[1] | Tong Chen[2] | Yanglong Yin[1] | Shuibing He[3] | Gang Chen[3]

[1]Research Center for Intelligent Computing Systems, Zhejiang Laboratory, Hangzhou, China

[2]Information and Communication Branch, State Grid Zhejiang Electric Power Co., Ltd., Hangzhou, China

[3]College of Computer Science and Technology, Zhejiang University, Hangzhou, China

Correspondence
Kuang Mao, Research Center for Intelligent Computing Systems, Zhejiang Laboratory, Building 10, Artificial Intelligence Town, 1818 Wenyi Xi Lu, Yuhang, Hangzhou, Zhejiang, China.
Email: maok@zhejianglab.com

Funding information
Key Research and Development Program of Zhejiang Province, Grant/Award Number: No.2018C01004; Key Scientific Technological Innovation Research Project by Ministry of Education in China; National Key Research and Development Plan of China, Grant/Award Number: No.2018AAA0101900; Zhejiang Lab Research Project, Grant/Award Number: No. 2020KC0AC01

Summary

As various algorithms and models spring up, object detection-based deep learning for image analysis has become an established technology in the field of computer vision. Furthermore, object detection for high-resolution videos derived from ubiquitous surveillance cameras draws many researchers' attention due to its practical significance and the challenge on detecting accuracy and speed. Existing object detection methods mainly resize and compress each frame in the video and then apply the detection algorithm, but with the method, the detection for small objects in dense scenes is far from satisfactory in terms of accuracy because small objects are reduced to pixel-level sizes in compressed frame images. In order to fix this issue, we propose AROD, a parallel detection framework based on adaptive image cropping. Unlike the traditional first-compression-and-then-detection methods, AROD adopts adaptive image cropping for distributed parallel detection. In this way, AROD significantly improves the accuracy of small object detection in dense scenes with acceptable overhead. In our experimental evaluation, YOLOv2-tiny model equipped with AROD reaches 29.29 FPS along with high detection accuracy. Experimental results verify that AROD ensures the high throughput of real-time video analytics while maintaining high detection accuracy.

KEYWORDS

distributed parallelism, high-resolution, small object detection

1 | INTRODUCTION

With the development of artificial intelligence, real-time object detection-based video surveillance has been increasingly applied in monitoring, tracking, and recognition. Especially during the breakout of COVID-19, real-time monitoring and recognition in the densely populated areas can provide help for epidemiological investigations and finding the close contacts. However, this technology requires high camera image resolution in order to capture more detailed information especially for distant objects, which is of great significance for scene monitoring.

Thanks to the convolutional neural network, there come more and more relevant applications such as face recognition,[1] video surveillance,[2] autonomous vehicle,[3] and so on. As for the state-of-the-art object detection methods, such as Faster R-CNN[4] and YOLO,[5] they generally train the deep neural networks with common datasets like ImageNet.[6] Images will be compressed first in terms of resolution, and then the corresponding model will be used for inference. Scale of this compression is dependent on the specific model. For example, YOLOv3-tiny[7] compresses the images to 320 pixels by 320 pixels while SSD512[8] compresses them to 512 by 512. There are two main reasons for this compression. First, it can greatly reduce

the calculation work for convolutional neural networks and thus the inference can be much faster to meet the actual application requirements. Moreover, the size of the target object is relatively large against the entire picture, in common training datasets such as ImageNet and Cifar100,[9] where compression will not make the large objects hard to recognize.

For the detections where objects are densely populated or are far from the camera, the detection scene is quite different from the traditional ones. A camera on railway stations, squares, and subway stations has a larger viewing angle, which is able to contain more objects to be detected but of smaller areas, especially the distant ones. This makes the traditional object detection algorithm fail to meet the requirements in terms of accuracy or speed.

In summary, existing algorithms face the following problems when aiming to comprehensively detect objects of interest in such dense scenes. First, algorithms that requires images to be compressed before detection, have low accuracy. For high-resolution videos of 1080P, compressing images to 320 by 320 pixels using traditional methods will cause too much loss of image details, thus large number of small objects will be missed. As shown in Figure 1, the accuracy of tiny distant objects is very low with YOLOv2-tiny,[10] which is far from satisfactory. Second, for the specific algorithm of small object detection, the speed of inference on high-resolution videos is slow, resulting in failure to apply it in real-time production scenarios. For example, Faster R-CNN has a good detection accuracy for small objects and can improve the detection accuracy to a certain extent, but its huge inference delay (7 frames per second (FPS)) cannot meet the needs of real-time tasks, as shown in Figure 2.

To solve these problems of object detection in dense scenes, we propose AROD, a distributed parallel detection framework. It generates suitable image cropping templates for the scene offline, uses the idea of data parallelism and makes full use of the GPU to accelerate the inference process. When each detection is completed, the results are quickly filtered to produce the final answers. Our framework can greatly increase the detection accuracy of high-resolution videos while only introducing a small overhead, especially for the tiny objects. Compared with the specialized small-object detection algorithm, the AROD framework equipped with the ordinary algorithm can obtain higher accuracy and faster detection speed. There are two main challenges during the implementation of our AROD framework.

1. *Diversity issue.* Due to the specific perspective of each camera, the sizes of the target objects in video frames are quite different in various scenes. A simple image cropping method[11] is proposed for the object detection in different scenes, where the original frames are sliced with a fixed size.

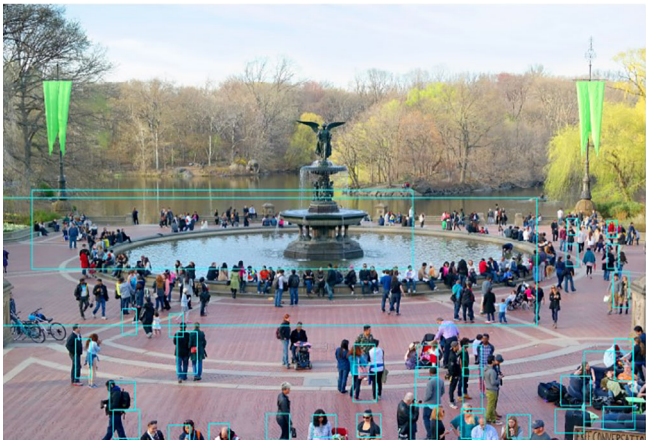


FIGURE 1 YOLOv2-tiny detection effect

FIGURE 2 Faster R-CNN detection effect

This design does not suit every scenario. For example, when the object of interest is generally larger than the fixed size, we will not get the correct detection results from the segmented parts but waste the computing resources. Therefore, it is necessary to study an adaptive image cropping method, which has the following benefits: (1) specify the cropping size of the best detection performance according to the objects in the videos; (2) applicable to any scenario without manual determination; and (3) support the scenario where objects are unevenly distributed, that is, areas of tiny and dense objects will be cropped several times while those of large and sparse ones with fewer.

2. *Duplication issue.* Detection results are extremely repetitive because of the segmentation, and deduplication will introduce a lot of overhead. We need to design an algorithm to reduce the overhead to the millisecond level. Existing methods such as non-maximum suppression (NMS) algorithm[12] encounter the problems of serial execution and poor scalability, resulting in a huge overhead for the detections of tiny objects in the dense scenario from high-resolution videos. Therefore, it is necessary to design a highly parallelized and scalable method, which reduces the time delay of the deduplication without introducing additional overhead even when the recognition bounding boxes increase greatly.

Contributions. First, we propose AROD, a distributed parallel detection framework, which can be applied to any object detection method. Second, our adaptive image cropping strategy significantly improves the accuracy of object detection for high resolution videos with dense scenarios, especially when the objects are tiny ones. Third, we realize a parallel version of NMS algorithm, where the deduplication of detection results derived from multiple tiles is parallelized and costs only several milliseconds. Forth, our framework makes it possible to deploy object detection algorithms that have relative low accuracy but have fast speed, achieving a well trade-off between processing speed and accuracy.

Article organization. The rest of this article is organized as follows. In Section2, we briefly introduce some relevant work and analyze their limitations in real-time and dense scenarios. We then give the workflow of AROD in Section 3 and experimental results in Section 4. Finally, we put forward some solutions to the problems that AROD may encounter in the future and conclude this article in Section 5.

2 | RELATED WORK

There have been many detection algorithms in computer vision, of which the performance is reaching new heights. In this section, we make review of two kinds of object detection algorithms and a traditional deduplication algorithms for the detection results.

2.1 | Algorithms of high accuracy

R-CNN[13] is such a representative, whose workflow is descried as follows. It first generates 1000 to 2000 candidate regions for a picture, and uses deep network to extract features for each region. The features are then sent to all the SVM classifiers to determine whether a region belongs to a certain category. Finally, regressors are used to fine-tune the positions of the bounding boxes for the objects in each frame. On one hand, detections of a large number of candidate regions give the R-CNN series algorithms an advantage in accuracy, especially for small object detection in dense scenes, but on the other hand, it also brings a huge inference delay.

R-CNN has been followed up with some acceleration optimizations. Considering the possible large overlaps between candidate bounding boxes, which leads to redundancy for feature extraction, Fast R-CNN[14] normalizes the entire image before sending it into the deep neural network. Further, it adds the bounding boxes in the last pooling layer rather than extracts the features in the convolutional layer, which solves the problem that R-CNN is slow in training and inference. When it comes to Faster R-CNN, it uses the region proposal network (RPN)[4] as a substitute of selective search[15] in Fast R-CNN, which further accelerates the processing of candidate bounding boxes.

Faster R-CNN is the state-of-the-art method with mAP reaching 73.2. However, it can only provide a processing rate of 7 frames per second for tiny object detection in dense scenes with high-resolution videos, failing to meet the requirements of real-time scenes in terms of high detection precision and low inference delay.

2.2 | Algorithms of high processing speed

YOLO treats the object detection task as a regression problem, thus directly obtaining the bounding box, classification and confidence. As its name, you only need to process each image once to get all the information above. Compared with the previous methods, YOLO has several advantages: (1) The whole process of YOLO is simple and quick without complicated detection. (2) Unlike other methods that apply sliding windows or Region Proposal, YOLO uses the entire image during training and testing, where the context is made good use of and it is much easier to make the right prediction on the background information. (3) YOLO is also able to learn the generalized features of objects.

YOLO can process the videos with high rate as it compresses the frame before detection. Because of that, YOLO fails to extract the features of tiny objects in dense scenes from the compressed images with missing details, and the accuracy is far from satisfactory. In our testing, although the processing rate of YOLO is up to 45 FPS, its detection accuracy is very low.

2.3 | Deduplication algorithms

NMS is common in object detection algorithms to remove the repeated bounding boxes, that is, to suppress the elements that are not maximum. Object detection is different from image classification in that the latter has only one output while the number of outputs of object detection is unknown. Except for ground-truth training, the model can never be sure of how many objects it is to predict on an image, so it will propose much more bounding boxes than the actual number. As a result, the number of bounding boxes is much larger, and they seriously overlap with each other. Therefore, we need NMS to pick the best one from all the candidates. Recent works[12,13,16] all rely on NMS algorithm.

The main idea of NMS is to sort all the bounding boxes with score, unconditionally reserve the one with the highest score, and then check all the remaining boxes to calculate their overlap area with the chosen one, where intersection over union (IOU) is often used for such quantification. If the value of IOU exceeds the threshold, the corresponding bounding box will be discarded. Every time another box with a high score is found, the above steps will be repeated to discard the boxes whose IOUs are greater than the threshold. This loop continues until all the boxes are processed.

We can find that the traditional NMS algorithm is heavily dependent on previous and subsequent steps, which is hard to be parallelized. When the number of bounding boxes increases, it shows a certain exponential growth trend. As for the dense scenes where the number of objects is generally large (more than 100), the traditional NMS algorithm will bring a lot of time overhead and cannot meet the requirements of real-time applications.

3 | DESIGN OF AROD

3.1 | Framework overview

First, we will give a formal definition of the problem that we have studied in this article. Given a high-resolution video, our objective is to greatly improve the detection accuracy of tiny objects without reducing the resolution of each video frame. Meanwhile, the processing rate is ensured to be over 25 FPS so as to meet the requirements of real-time applications.

Therefore, we propose AROD, a distributed parallel detection framework with adaptive image cropping strategy. Our framework supports any object detection algorithms, and meets the requirements of high precision and low latency for object detection in dense scenes while introducing a small overhead.

Workflow of AROD is shown in Figure 3. Modules in AROD are enclosed by boxes with dashed lines, which are divided into two parts as online and offline. In the offline section is the module named *adaptive cropping template generator*. The online section contains two modules, image cropping and *parallel NMS*. Implementation is described as follows:

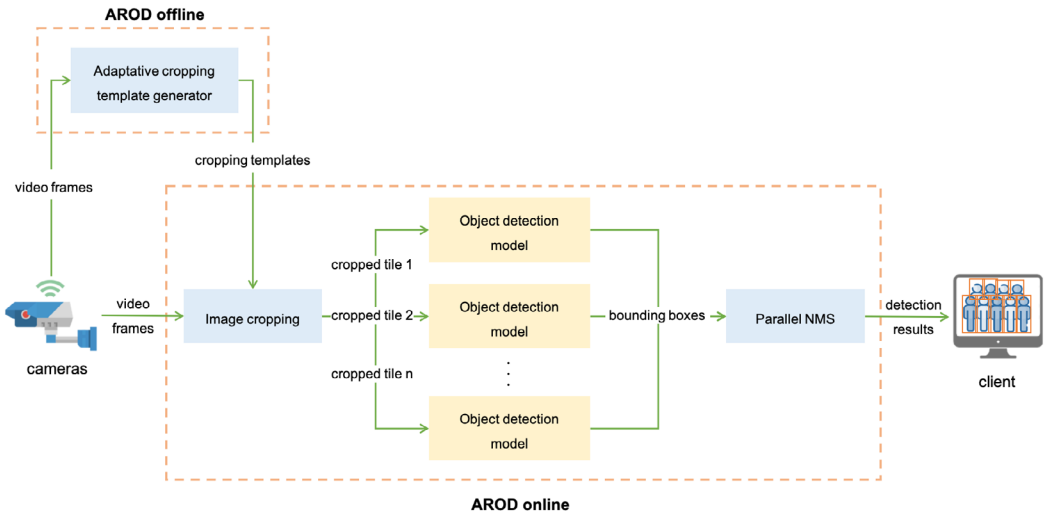


FIGURE 3 AROD image detection process

Step1: generate image cropping templates with the offline generator.

Step2: crop video frames with the above templates and run object detection algorithm on each tile in parallel.

Step3: merge the detection results from each cropped tile by the parallel NMS module.

In the rest of this section, we introduce the *adaptive cropping template generator* (Section 3.2), *image cropping* (Section 3.3), and *parallel NMS* (Section 3.4) in detail.

3.2 | Adaptive cropping template generator

For high-resolution video frames to be detected, we cannot predict the actual sizes of the objects in them. If the target object is too large, there may be too many cropped tiles as well, which will bring difficulty for the detection. In contrast, if the object is tiny and the number of cropped tiles is also small, then it is likely that this tiny object will be missed by the detector. In general, the best cropping plan is unknown so it is necessary to find an adaptive strategy. Our *adaptive cropping template generator* is thus proposed to automatically generate suitable cropping templates for the objects of interest in the video frames. This stage can be seen as the learning process of the HD cameras for a certain scene. Although it is time-consuming, it is a one-time thing whose cost can be amortized. For scenes where the distribution of objects changes frequently, new cropping plans are dynamically generated to adapt to the new distribution characteristics. Moreover, this generator has nothing to do with the main flow so it works asynchronously.

Details of the algorithm are shown in Algorithm 3. To simplify, we do not show the overlaps among the cropped tiles, which exist in the generation step. The overall process is descried as follows and illustrated in Figure 4:

Step1: For the first frame, we directly run the object detector on the entire image. Then we partition the image into two parts by Algorithm 1, *A*0 and *B*0, and count the number of bounding boxes contained in every region, denoted as *n_A*0 and *n_B*0.

Step2: Respectively run object detector on *n_A*0′ and *n_B*0′ and record their number of bounding boxes as *n_A*0′ and *n_B*0′. Compare *n_A*0 with *n_A*0′. If *n_A*0′>*n_A*0×F % (F is manfully set), that means we need to further partition *A*0′ into *A*1 and *C*1, detail in Algorithm 2, and update the cropping template. Such process is the same for *B*0 but we avoid repeating for simplicity.

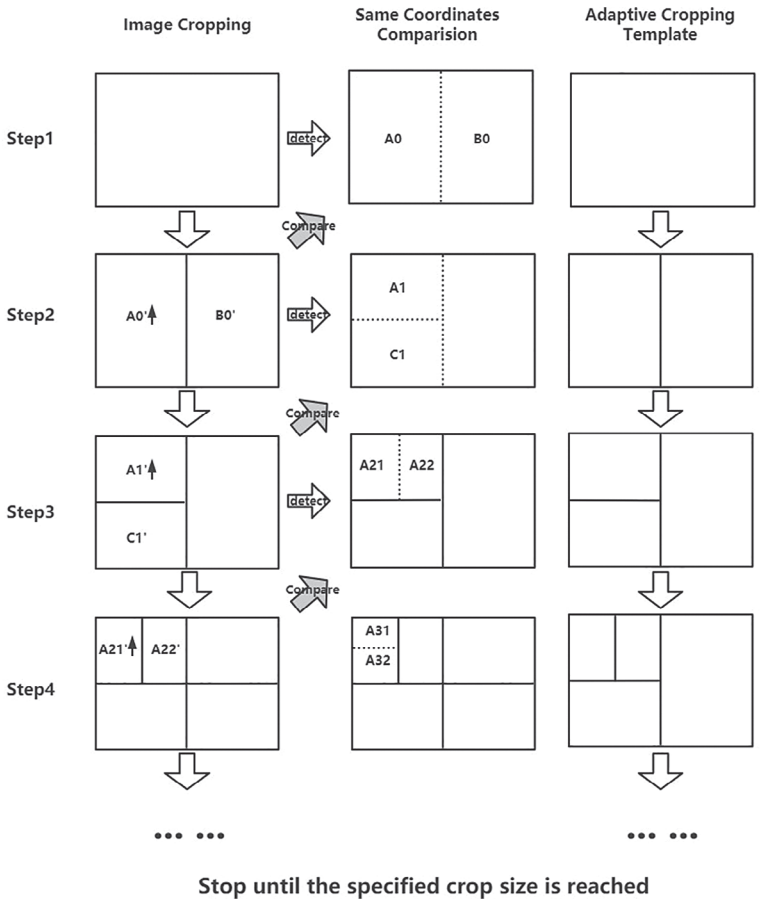


FIGURE 4 Adaptive cropping template generator

Step3: Iteratively repeat the above steps until the size of the minimum tile in the generated cropping template is smaller than the expected value set in advance. At this time, this template can be applied in the online cropping tasks.

As can be seen, we need to partition the same image and do the inference several times when generating the template. In actual, however, it does not take a lot of time. First, because of the parallelization of the AROD framework, if there are sufficient computing resources, the detection and inference time of 1 tile and 16 tiles are nearly the same. Further, taking YOLOv3-tiny as an example, where the input image size is 320 by 320, we can only generate 80 cropped tiles even for the 4K image. Under the current circumstance shown in Figure 4, we only need 3–4 steps in common cases and 7 steps even in the worst case. We can also speed up the partition rate if needed, such as quartering a certain region in each step rather than dichotomy. Moreover, this operation can be completed asynchronously with the online inference of real-time applications.

Algorithm 1. Next cropping

Input: *crops*
Output: *crops*
for *crop* ∈ *crops* **do**
    split *crop* in defined way;
end for
return *crops*;

Algorithm 2. Check validation of cutting

Input: *bboxes_counts S*, *next_bboxes_counts S′*
next_crops ← {};
is_validate ← *False*;
for *i* ← 0 **to** *len*(S) **do**
    if *S′*[*i*] > *S*[*i*] ∗ (1+F%) **then**
        is_validate ← *True*;
        next_crops.append(*crop*);
    end if
end for
if *is_validate* = *True* **then**
    return *next_crops*;
else
    return *False*;
end if

3.3 | Image cropping

When the generated cropping template is used to crop the image, because the input is a video stream, we can use *selection detection* (Section 3.3.1) to reduce repeated detection. To avoid large object being split, we also need to use *combinative detection* (Section 3.3.2) to accommodate misidentification.

3.3.1 | Selective detection

Considering that the input is a video stream, where the objects will not change much among neighboring frames, we can make use of the detection results of the previous frames as the reference for which tiles to be detected in the next few frames. Note that some cropped tiles may not contain any object of interest, as illustrated in Figure 5. In these scenarios, it is wise to avoid those regions containing no objects of interest to reduce the calculation burden. Therefore, as shown in Figure 6, the four tiles in the top row are ignored within the next *N* frames because they contain no objects. Until *N* frames later, all the 12 tiles of the entire image will be detected again.

Algorithm 3. Adaptive cropping template generation

Input: *image*
$C, T \leftarrow [[0, 0, w, h]]$;
cropping_template ← [];
$S \leftarrow w * h$;
input_area ← defined model input area;
while $S >$ *input_area* **do**
  $B \leftarrow$ detect every *image* in C in parallel;
  $N\_C \leftarrow$ call *next_cropping*(C); (see Algorithm 1)
  $B\_C \leftarrow$ count B in N_C;
  $N\_B \leftarrow$ detect every *image* in N_C in parallel;
  $N\_B\_C \leftarrow$ count N_B in N_C;
  res ← call *check_valid_cutting*(B_C, N_B_C); (see Algorithm 2)
  if *res* = *False* **then**
    return T;
  else
    $C \leftarrow$ *res*;
    record cropping template T;
  end if
  update S;
end while
return *cropping_template*;

FIGURE 5 Cropping template

3.3.2 | Combinative detection

It is likely that some objects are larger than the cropping size and fall into several tiles. For those tiles that only contains a part of such a large object, the detection results are not credible. Take Figure 6 for further optimization, some people on the frame are too large to be contained in a single tile. Therefore, we run the object detector on not only cropped tiles but also the entire image, and then combine the results as Figure 7. After that, parallel NMS algorithm 3.4 is applied for the deduplication of bounding boxes to obtain the right results.

3.4 | Parallel NMS

We need to do two things when merging the detection results of all cropped tiles. First, map the coordinates of bounding boxes back to the original frames. Second, even if every frame has an overlap part, it is likely that multiple tiles contain some parts of it. This may give rise to the problem that an object is repeatedly labeled. NMS algorithm is applied to address this problem. However, the traditional NMS algorithm has high time complexity and

FIGURE 6 Selective detection

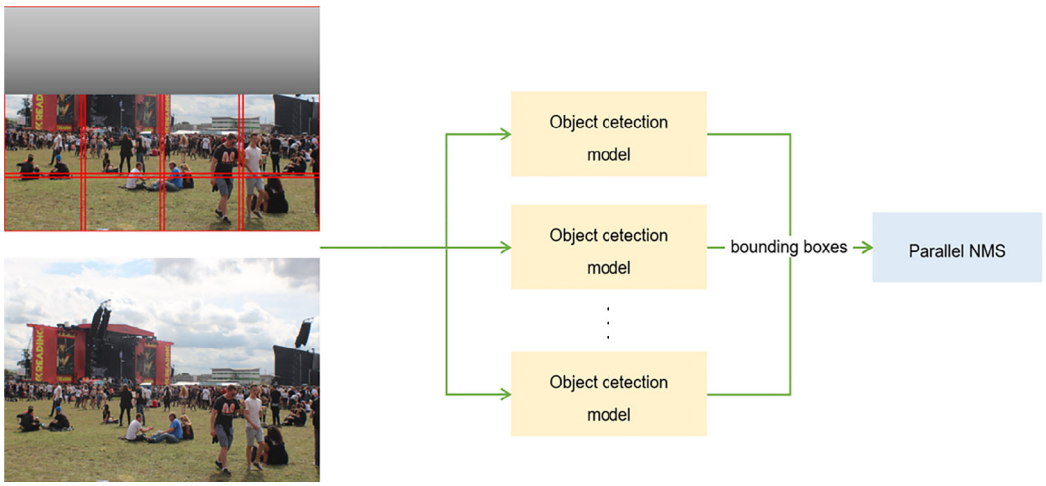


FIGURE 7 Combinative detection

is not suitable for real-time applications. Especially in the dense scenario, the number of bounding boxes will be very large. Therefore, we propose a highly parallel NMS algorithm that uses GPU multi-threading to achieve low-latency deduplication, detailed in Algorithm 4. Main idea is described as follows:

Step1: Map all the bounding boxes of every cropped tile that returned from the object detector back to the corresponding positions in the original image, denoted as set ***B***. Meanwhile, initialize an array named ***retain*** for set ***B*** where every element is tentatively ***True*** and corresponds to a bounding box in set ***B***.

Step2: Let each thread on the GPU process two bounding boxes in set ***B*** and denote them as ***box_i*** and ***box_j***.

Step3: If the overlap area of ***box_i*** and ***box_j*** exceeds the threshold θ and the confidence of ***box_i*** is greater than that of ***box_j***, we will then change the value in set ***B*** that corresponds to ***box_j*** from ***True*** to ***False***.

4 | EXPERIMENT

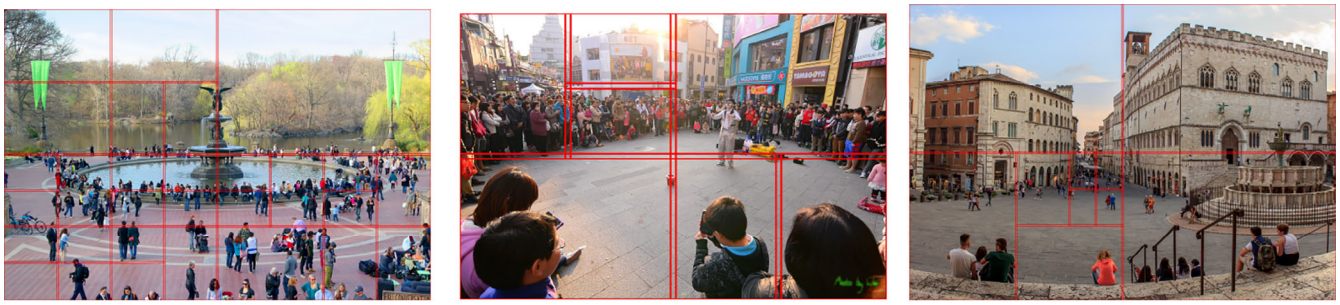
In this section, we mainly conduct three experiments: first, we taking four 4K images as samples, optimize YOLOv2-tiny with AROD, and compare it against the YOLOv2-tiny baseline in terms of processing rate and detection accuracy; second, aiming in open source high-resolution dense dataset *CrowdHuman*,[17] we calculate the *AP* values of YOLOv2-tiny equipped with AROD and compare it against that for YOLOv2-tiny; thirdly, we evaluate the performance and scalability of parallel NMS with a TensorFlow version[18] as baseline.

4.1 | Setup

Each worker node is equipped with 8 Intel Xeon(R) Gold 6126 CPU cores with 32GB of memory and a V100 GPU. Client node is the same as worker node in CPU and memory but has no V100 GPU. The master node is used to receive image, crop the image into tiles, deliver them to every worker

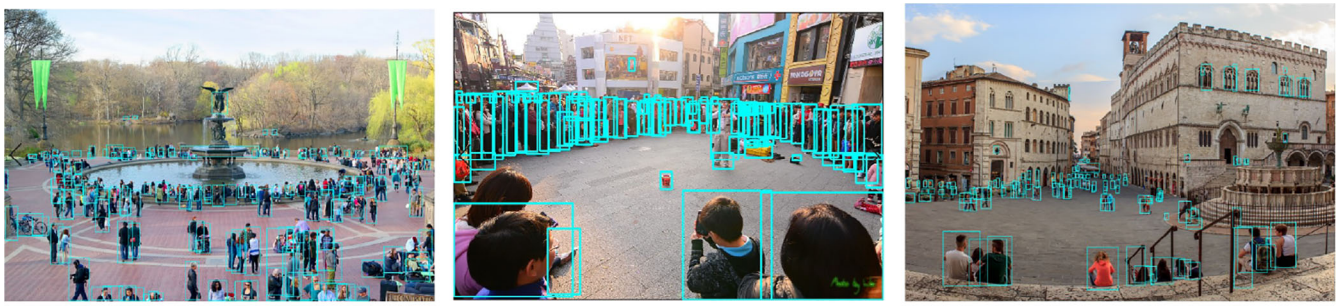
Algorithm 4. Parallel NMS

Input: $B = b_1, b_2, \ldots, b_N, retain = [True, True, \ldots, True]$
//B and *retain* have same length
$i \leftarrow blockId.x * blockDim.y + threadIdx.x$
$j \leftarrow blockId.y * blockDim.x + threadIdx.y$
if $B[i].s < B[j].s$ **then**
  $iou \leftarrow IOUcalc(B[i], B[j])$
  if $iou > \theta$ **then**
    $retain[i] \leftarrow False$
  end if
end if
return *retain*



(A) (B) (C)

FIGURE 8 Cropping template generated by *adaptive cropping template generation*



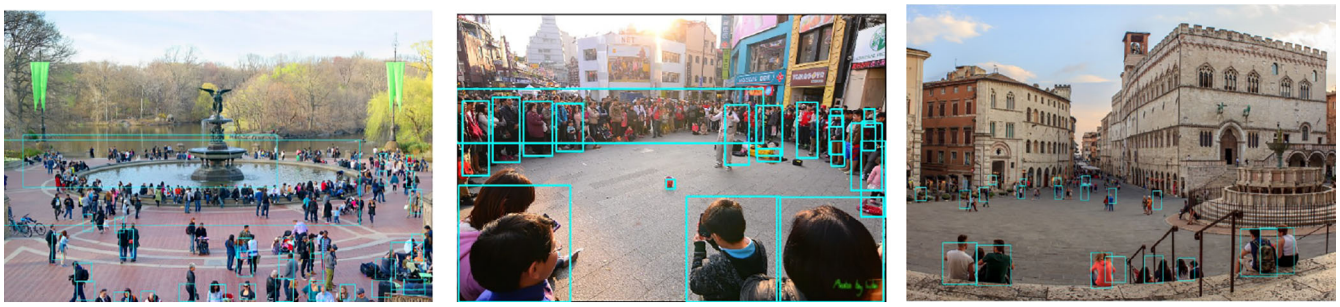
(A) (B) (C)

FIGURE 9 YOLOv2-tiny-AROD

node, merge all the detection results from each tile and finally rendering them back to the original image. Each worker node is used to receive the cropped tiles from the client node, run the object detector, and return the bounding boxes back to the client node. There are 1 master node and 2 worker nodes in our experiment and the bandwidth between them is 200Mb/s. Considering that the number of tiles derived from 1080P images is not so large and can be well responded by only two workers.

4.2 | Results

As shown in Figure 8, the *adaptive cropping template generator* generated different cropping templates for different images. And Figure 9 demonstrates the output of YOLOv2-tiny-*AROD* strategy while Figure 10 shows that of YOLOv2-tiny baseline. As can be seen clearly, *AROD* brings a significant improvement in accuracy that a large number of dense tiny objects are detected.



(A) (B) (C)

FIGURE 10 YOLOv2-tiny

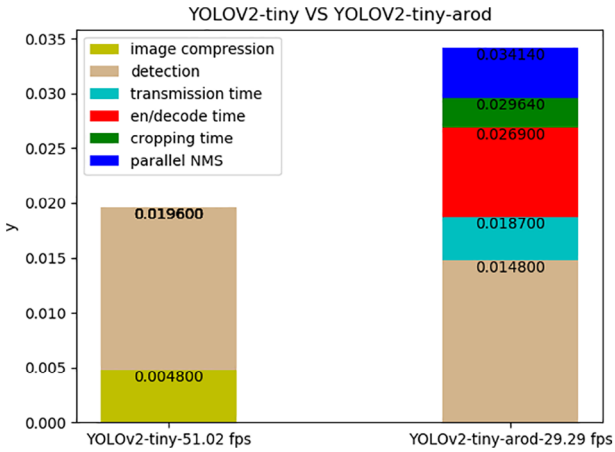


FIGURE 11 Performance comparison

Multi-Row	Avg. Precision (%), Iou: 0.5	0.75	0.8	0.85	0.9	0.95
YOLOv2-tiny	27.22	22.52	20.05	16.21	9.84	1.53
YOLOv2-tiny-AROD	29.39	24.95	22.57	18.97	12.87	3.51

TABLE 1 The accuracy achieved by YOLOv2-tiny with/without AROD

More details about the performance comparison between the YOLOv2-tiny baseline and YOLOv2-tiny-*AROD* are shown in Figure 11. *Transfer* (gray) refers to the time of data transmission between the client node and the worker node. *En/Decode* (yellow) time is also taken into account because that data transmission between nodes is based on TCP/IP protocol, where binary stream conversion is inevitable. We can see that although YOLOv2-tiny-*AROD* introduces additional overhead and the processing rate decrease to 29.29 FPS from 51.02 FPS, it can still fully meet the needs of real-time applications.

Alternatively, we can deploy multiple GPUs in a single machine to play the role of multiple workers, thus avoiding the *Transfer* and *En/Decode* time to reach higher processing rate. According to the experimental results of Figure 11, it can theoretically reach 45.37 FPS. This single-machine-multiple-GPUs approach can reach a higher FPS while the distributed approach allows us to build a parallel platform with multiple servers of lower performance.

CrowdHuman is a baseline dataset used to evaluate detectors in crowd scenarios. The subset was trained and verified with 470K instances, and there were 23 people in each image. Various occlusion was found in the dataset. In fact, the CrowdHuman dataset is by far the most suitable dataset we have found for our use, although it is far from perfect. In most images, the size of the object is still large and the resolution is not as high as the image shown above, which greatly reduces the advantages brought by *AROD*. As shown in Table 1, we can see the improvement of accuracy brought by *AROD*. Given an open-source dataset that matches the scenario described in this article, as shown above, the increase in accuracy is even more significant.

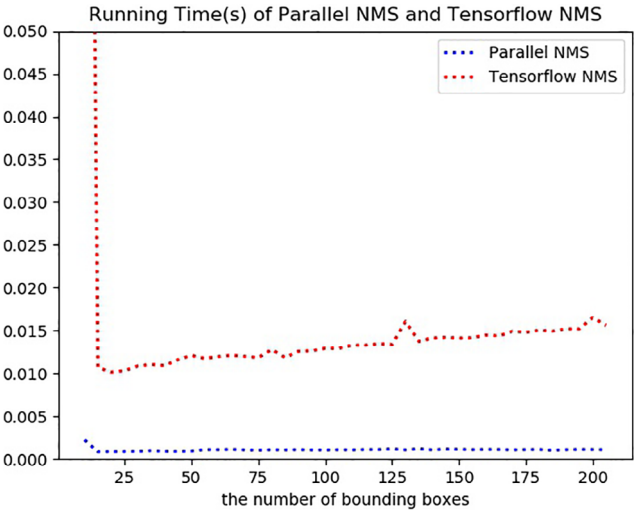


FIGURE 12 NMS performance comparison

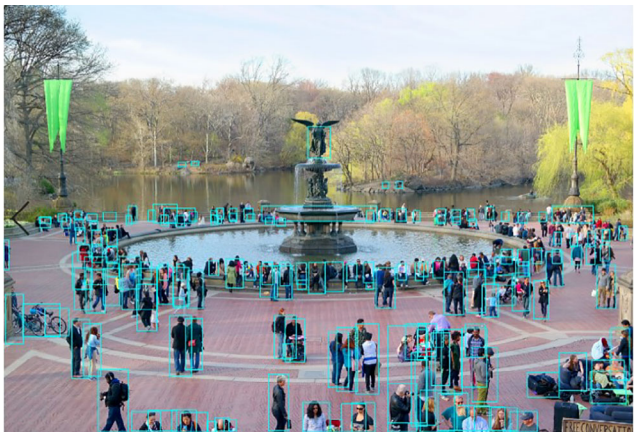


FIGURE 13 Parallel NMS

FIGURE 14 Tensorflow NMS

Referring to Figure 12, we can see that parallel NMS has good performance and strong scalability: as the number of bounding boxes increases, its time complexity still maintains $O(1)$. In contrast, there exists linear growth in TensorFlow NMS, which also takes more time than our parallel NMS. Deduplication results of both methods are shown in Figures 13 and 14, and we find that their effects are similar.

5 | CONCLUSIONS

AROD realizes the high-accuracy detection of small objects in dense HD image scenes. With YOLOv2-Tiny algorithm, the processing speed reaches more than 29.29 FPS, fully meeting the requirements of real-time tasks. Moreover, AROD is completely independent on the algorithm model used

for detection. With the acceleration of algorithm model detection, AROD can provide higher processing speed. When the detection algorithm model achieves a very high FPS, the overhead of *Transfer*, *En/Decode* introduced by the AROD framework may become noticable. We can use a series of optimizations to reduce the cost of AROD, such as: (1) Use GPU to complete image *En/Decode* tasks. (2) Use RDMA technology to replace the current TCP/IP transmission. In practical applications, multiple GPUs can be deployed on a single node to play the role of multiple workers, to avoid the overhead of transmission and communication between nodes. In this way, higher FPS can be obtained, and AROD is designed to work in this mode seamlessly as well. A distributed multiple-node system can be used if the application grows large in size.

In practice, it is wiser to deploy multiple GPUs on a single node to play the role of multiple workers instead of distributed architecture. In this way, the overhead of communication between nodes can be avoided to reach a higher speed, which is also supported in our AROD. Only when the scale of the application grows to a certain extent shall we take the distributed architecture into account.

ACKNOWLEDGMENTS

This work was supported by the National Key Research and Development Plan of China No.2018AAA0101900 and the Key Scientific Technological Innovation Research Project by Ministry of Education in China and the Zhejiang Lab Research Project No. 2020KC0AC01 and Key Research and Development Program of Zhejiang Province No.2018C01004.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

ORCID

Yuefeng Wang https://orcid.org/0000-0002-8731-4074
Kuang Mao https://orcid.org/0000-0003-1050-2947

REFERENCES

1. Viola PA, Jones M. Robust real-time face detection. *Int Conf Comput Vis*. 2001;57(2):137-154.
2. Venetianer PL, Lipton AJ , Chosak AJ, et al. Video surveillance system. US Patent App. 11/057,154; 2005.
3. Source: Wikipedia. Unmanned vehicles. Chronicle Books; 2011.
4. Ren S, He K, Girshick R.B, Sun J. Faster R-CNN: Towards Real Time Object Detection with Region Proposal Networks. *IEEE Trans. Pattern Anal. Mach. Intell.* 2017;39(6):1137-1149.
5. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. Paper presented at: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV; 2016:779-788.
6. Deng J, Dong W, Socher R, Li LJ, Li K, Fei-Fei L. Imagenet: a large-scale hierarchical image database. Paper presented at: Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition 2009, Miami, Florida; June 2009:248-255; IEEE.
7. YOLOv3-tiny. https://pjreddie.com/darknet/yolo.
8. Liu W, Anguelov D, Erhan D, Szegedy C, Reed S, Fu CY, Berg AC. SSD: single shot multibox detector. Paper presented at: Proceedings of the European Conference on Computer Vision; 2016:21-37; Springer, Cham.
9. Krizhevsky A. Learning multiple layers of features from tiny images; 2009.
10. YOLOv2-tiny. https://pjreddie.com/darknet/yolo.
11. Ozge Unel F, Ozkalayci BO, Cigla C. The power of tiling for small object detection. Paper presented at: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, Long Beach, CA; 2019.
12. Li H, Lin Z, Shen X, Brandt J, Hua G. A convolutional neural network cascade for face detection. Paper presented at: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Boston, MA; 2015:5325-5334.
13. Girshick R, Donahue J, Darrell T, Malik J. Rich feature hierarchies for accurate object detection and semantic segmentation. Paper presented at: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition CVPR, Columbus, OH; 2014.
14. Girshick R. Fast r-cnn. Paper presented at: Proceedings of the IEEE International Conference on Computer Vision, Boston, MA, US; 2015:1440-1448.
15. Uijlings JR, Sande KE, Gevers T, Smeulders AW. Selective search for object recognition. *Int J Comput Vis*. 2013;104(2):154-171.
16. Farfade SS, Saberian MJ, Li LJ. Multi-view face detection using deep convolutional neural networks; 2015. arXiv preprint arXiv:1502.02766.
17. Shao S, Zhao Z, Li B, et al. Crowdhuman: a benchmark for detecting human in a crowd; 2018. arXiv preprint arXiv:1805.00123.
18. TensorFlow NMS. https://www.tensorflow.org/api_docs/python/tf/image/non_max_suppression

How to cite this article: Wang Y, Mao K, Chen T, Yin Y, He S, Chen G. Accelerating real-time object detection in high-resolution video surveillance. *Concurrency Computat Pract Exper*. 2021;e6307. https://doi.org/10.1002/cpe.6307