Speed up Object Detection on Gigapixel-level Images with Patch Arrangement

Jiahao Fan[1], Huabin Liu[1], Wenjie Yang[1], John See[2], Aixin Zhang[1,*], Weiyao Lin[1,*]
[1]Shanghai Jiao Tong University, China, [2]Heriot-Watt University, Malaysia
{jiahaofan, huabinliu, 13633491388}@sjtu.edu.cn, J.See@hw.ac.uk, {axzhang, wylin}@sjtu.edu.cn

Abstract

With the appearance of super high-resolution (e.g., gigapixel-level) images, performing efficient object detection on such images becomes an important issue. Most existing works for efficient object detection on high-resolution images focus on generating local patches where objects may exist, and then every patch is detected independently. However, when the image resolution reaches ***gigapixel-level****, they will suffer from a huge time cost for detecting numerous patches. Different from them, we devise a novel patch arrangement framework for fast object detection on* ***gigapixel-level*** *images. Under this framework, a Patch Arrangement Network (PAN) is proposed to accelerate the detection by determining which patches could be packed together into a compact canvas. Specifically, PAN consists of (1) a Patch Filter Module (PFM) (2) a Patch Packing Module (PPM). PFM filters patch candidates by learning to select patches between two granularities. Subsequently, from the remaining patches, PPM determines how to pack these patches together into a smaller number of canvases. Meanwhile, it generates an ideal layout of patches on canvas. These canvases are fed to the detector to get final results. Experiments show that our method could improve the inference speed on gigapixel-level images by 5× while maintaining great performance.*

1. Introduction

With the widespread use of super high-resolution cameras, image resolution has increased rapidly and recently reached the gigapixel level (*e.g.*, 25,000 $\times$ 14,000 pixels) [20]. Therefore, it's a great challenge to analyze such images efficiently. Recently, some object detection methods [8, 23] have been proposed for efficient high-resolution object detection. As shown in Figure 1a, in order to speed-up object detection, they focused on generating local regions (termed as '***patch***') that may contain object candidates from high-resolution images. Then detection is only

[*]Corresponding authors.

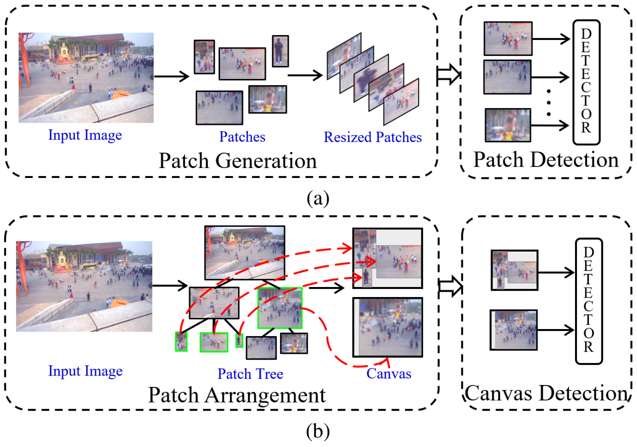


Figure 1. Comparison between pipelines. (a) Previous: it first generates patches where object candidates exist, then detection is conducted on all patches separately. (b) Ours: based on a multi-grained patch tree, certain patches are selected and packed together into compact canvases with a smaller number. Detection is only conducted on canvases.

performed on these resized patches instead of the whole image to achieve the speed-up. Therefore, their time cost is dependent on the number of patches. However, when being applied to **super** high-resolution images (*e.g.*, gigapixel-level images), these methods may require numerous patches to ensure performance. Thus, they may still suffer huge time costs on gigapixel-level images.

In gigapixel-level images, we observed that it is neither necessary nor ideal to perform detection separately on each generated patch. As shown in Figure 1b, some patches can be packed together to form a new and compact one (called '***canvas***'). In this way, processing a **smaller number** of canvases rather than **numerous** patches could significantly speed up the detection.

Based on the above intuition, we propose a novel framework for object detection on gigapixel-level images. Under this detection framework, a Patch Arrangement Network (PAN) arranges patches into compact canvases in a local-to-global view, into compact canvases for final detection. Thereby it significantly speeds up object detection in gigapixel-level images while maintaining ideal performance. ***First***, we build up a multi-grained patch tree for the input gigapixel-level image, which consists of fine-grained patches (leaf layer nodes) and their corresponding coarser-grained patches (middle layer nodes). To embed information of patches and explore the relationship among them, an LSTM-based tree structure encoder is applied. The subsequent arrangement is performed based on the multi-grained patch candidates in this tree. ***Second***, we propose a Patch Filter Module (PFM), which learns an adaptive patch selection between two granularities, i.e, it arranges some neighboring fine-grained patches into their corresponding coarser one. In this way, detection can be accelerated by replacing some patch candidates with fewer coarser patches. ***Third***, we further develop a Patch Packing Module (PPM). It determines how to pack all remaining patches together to form a smaller number of canvases, and generates a compact layout of packed patches in the canvas. PFM performs arrangement over neighboring patches (local view) while PPM takes all the patches into consideration for arrangement (global view). ***Finally***, the whole framework is trained jointly with policy-based reinforcement learning.

Note that our theoretical speed-up ratio is controllable by setting the maximum amount of patches that can be packed in each canvas. We evaluate our approach on the gigapixel-level PANDA [20] dataset and a wide range of detectors. PAN maintains ideal detection performance while improving the inference speed by $5\times$. The main contributions of our work are three-fold:

- We devise a novel framework for efficient object detection on gigapixel-level images, which adaptively packs patches into a compact canvas and generates an ideal layout for selected patches in this canvas.
- Under this framework, we propose a novel multi-grained patch tree to explore the relationship among patches. Based on this tree, a patch filter module and patch packing module are proposed to arrange patch candidates in a local and global view, respectively.
- Extensive experiments show that PAN can speed up the inference speed of detection on gigapixel images by $5\times$ while maintaining high detection performance.

2. Related Work

Object detection on High-resolution Images With the wide application of high-resolution (HR) cameras, there has been an increasing demand for performing object detection on HR images. Typically, the aerial and remote sensing images own relatively higher resolutions. For example, images in VisDrone [24] dataset can reach $2{,}000\times1{,}500$ pixels. Recently, PANDA [20] is introduced, which is the first **gigapixel-level** ($25{,}000\times14{,}000$) human-centric video&image dataset. It further expands the frontiers of high-resolution image analysis [10, 11] and remains a great challenge for speeding up object detection on such large images.

A major line of accelerating object detection focuses on devising efficient network architecture (*e.g.*, Faster R-CNN [15], YOLO [14], and SSD [12]). However, most of them are developed on general images like MSCOCO. Directly applying them to HR images may still cause a huge time cost. Meanwhile, specific approaches have not been well-studied for HR images before.

For this reason, recently, some works have been proposed to speed up object detection on relatively HR images. [4] proposes an image-level solution, which adaptively selects a resolution for each input image. It's acknowledged that we can only conduct necessary computations on some partial regions instead of the whole HR image. Based on this intuition, most existing works focus on finding local spatial ***patches*** where objects may exist. [16] regards the detected boxes obtained on a low resolution as patches and performs final detection on these patches at a refined resolution. Similarly, CRENet [21] clusters the coarse detection results to form the patches. ClusDet [23] follows the idea of RPN [15] and employs a neural network to estimate accurate patches. AutoFocus [13] and GLSAN [5] generate patches based on the image features. DMNet [8] obtains patches with the guidance of density maps. Besides, reinforcement learning (RL) has also been adopted to find the valuable patches [6, 18]. Nevertheless, the above methods usually require more patches to ensure the performance as the image resolution increases. When the image resolution reaches **gigapixel-level**, the number of patches will be numerous (may exceed 12,000). Therefore, their corresponding time costs are still unbearable in real-world applications.

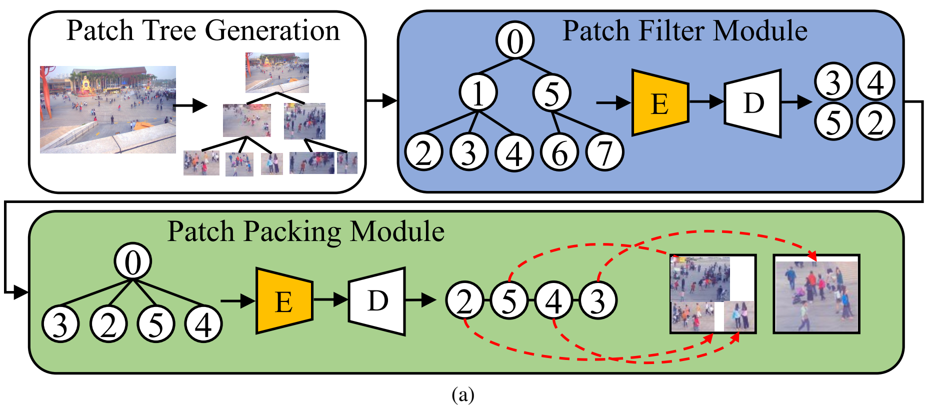
3. Proposed Method

Figure 2a illustrates an overview of our framework. In the following sections, we first describe the construction of our multi-grained patch tree. Based on this tree, we elaborate the main modules involved in our approach: Patch Filter Module (PFM) and Patch Packing Module (PPM). Finally, we describe how to optimize our framework.

3.1. Patch Tree Generation

To learn how to arrange patches originated from the same high-resolution image, we argue that it is critical to represent and learn their intrinsic relationship. Instead of using each patch separately, we fully explore their intrinsic relationships by first constructing a multi-grained patch tree.

Initial patch generation. Following CRENet [21], we first obtain initial patches based on clustering. Specifically,



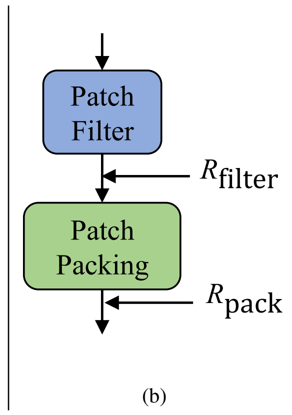


Figure 2. (a) The architecture of the two-stage patch arrangement module. (b) Joint training strategy. The patch arrangement policy is divided into two sub-policies: patch filter policy(blue) and patch packing policy(green). The different rewards are given after each policy finishing.

mean shift [3] is adopted to cluster the object boxes according to their positions, and then we merge these clustered boxes to generate patches. During training, we directly use the ground-truth boxes for clustering. As for inference, a coarse detector is applied to the down-scaled input images to get rough detection results for clustering.

Multi-grained patch tree. Based on initial patches, we build a multi-grained patch tree $G = (V, E)$ from leaves to root as illustrated in Figure 3). ***First***, the initial patches form leaf nodes in the patch tree; they are regarded as *fine-grained* patches. ***Second***, we cluster these leaf patches by mean shift [3] to obtain new *coarse-grained* patches in the middle layer of tree. ***Finally***, the root node is represented by the complete image. It is connected with all nodes in V_{mid}. In this way, the spatial inclusion relationship among multi-grained patches is reflected by the parent-children relationship between nodes in the tree. The nodes set V is split into leaf nodes set V_{leaf} and middle nodes set V_{mid}. Each node $v_i \in V$ corresponds to a patch, node v_i and v_j are connected when patch v_i covers patch v_j spatially. All patch nodes in V_{leaf} and V_{mid} constitute the **patch candidates** $\{v_i\}_{i=1}^N$ for the following arrangement, where $N = |V_{\text{leaf}}| + |V_{\text{mid}}|$.

3.2. Tree Structure Encoder

To explore relationships among patch nodes in the tree, we introduce a novel tree structure encoder to embed (1) object information of each patch node, and (2) the inter- and intra-layer relationship among patch nodes in the tree.

Specifically, we first represent each patch node v_i by $\mathbf{v}_i = (x_i, y_i, w_i, h_i, r_i, a_i, o_i, n_i)$, where x_i and y_i are the center coordinates of the patch, w_i, h_i, r_i represent its width, height, and aspect ratio, respectively. a_i is area of this patch. o_i, n_i denotes the average object area in this patch, and number of objects in the patch, respectively. We can estimate o_i and n_i with the coarse detection results (used in the initial patch generation). Next, this 8-d feature is embedded to a high-dimensional representation by a learnable fully-connected layer.

To explore relationships among patch nodes in the tree, we apply a LSTM-based model consisting of Tree-LSTM [17] and Chain-LSTM (standard LSTM).

Firstly, a Child-Sum Tree-LSTM [17] is leveraged to encode the ***inter-layer*** relationship in the tree. Similar to standard LSTM, each Tree-LSTM unit (indexed by j) has its hidden state $\boldsymbol{h}_j$. Under the Child-Sum style, Tree-LSTM conditions $\boldsymbol{h}_j$ on the sum of its child hidden states:

$$\tilde{\boldsymbol{h}}_j = \sum_{k \in C(j)} \boldsymbol{h}_k \tag{1}$$

where $C(j)$ denotes the set of children of node v_j. In this way, information across different layers in the tree could be well aggregated in a bottom-up manner (indicated by the solid line in Figure 3).

To further learn the ***intra-layer*** relationship, we first unfold the hidden states of Tree-LSTM in a chain-like style. For flexible implementation, we construct the nodes chain in the *pre-order traversal* manner (indicated by the dotted line in Fig. 3). Then, the unfolded hidden states are sequentially fed into the Chain-LSTM.

The overall tree structure encoder can be formulated as:

$$z_i = LSTM_{\text{Chain}}(LSTM_{\text{Tree}}(FC(\mathbf{v}_i))) \tag{2}$$

where z_i is the encoded representation of patch v_i.

3.3. Patch Filter Module

In the multi-grained patch tree, one coarse-grained patch covers its corresponding fine-grained patches spatially (as shown in Figure 3). Intuitively, we can only leave the coarse-grained patches to significantly reduce the number

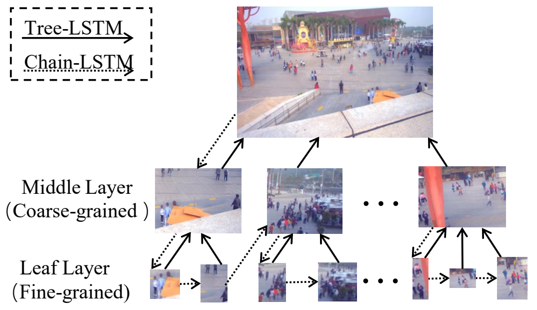


Figure 3. An example of multi-grained patch tree and the encoding data flow. The solid line shows the data flow of Tree-LSTM. The dashed line shows the data flow of Chain-LSTM.

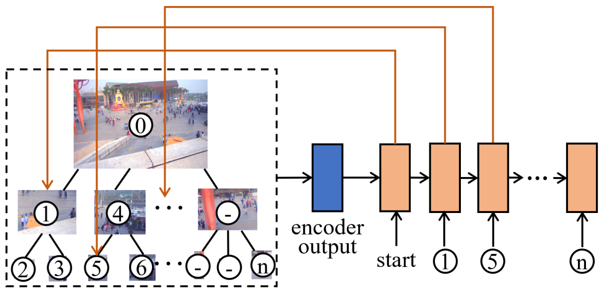


Figure 4. The mechanism of filter decoder. The decoder takes the output of tree structure encoder as input and points to a specific patch on the tree at each time step.

of patch candidates for detection. However, considering the variance in object scale, not all objects can be well detected at a coarser granularity. Based on this motivation, we introduce the patch filter module (PFM), which learns to adaptively select patch candidates between two granularities (as shown at the top of Figure 2a). We formulate such process as a *tree-to-set* problem. Since the tree is well-encoded, we apply a filter decoder to select patches from the tree.

Filter Decoder Specifically, the filter module aims to generate a subset $S = \{v_j\}_{i=0}^{M}$ from the patch nodes candidates $\{v_i\}_{i=1}^{N}$, where $|S| = M{<}N$. The filter decoder is constructed by a standard LSTM. As illustrated in Fig. 4, at every time step j, the filter decoder points to a specific patch node on the input tree (as illustrated in Fig.2). Then the pointed node is used as the input of the $j + 1$ step. For implementation, we adopt the *point mechanism* proposed in PointerNet [19], which empowers the module to point to a specific node in the tree. Besides, to avoid duplicated selection, we introduce a mutually excluded mechanism between fine-grained and corresponding coarse-grained patches. For example, in Figure 4, if node 1 is selected, its corresponding fine-grained patch nodes 2&3 will be masked. Vice versa, node 1 will be masked if either node 2 or 3 is selected.

3.4. Patch Packing Module

Given remaining patches, the patch packing module (PPM) aims to pack certain patches into canvases and initialize the layout of patches in the canvas. Different from PFM, which focuses on arranging patches in a local and neighboring view, the PPM performs patch arrangement at the global view, *i.e.*, all patches are considered during packing regardless of their positions in the whole image.

Specifically, given the output patch node set $S = \{v_j\}_{j=0}^{M}$ of PFM, we firstly re-organize it to a new patch tree G_s of two layers. As shown in the Figure 2a, the root node represents the whole image in a global context while the patch node set comprises of nodes of the leaf layer. We then re-use the earlier tree structure encoder used in PPM to fully utilize the learned knowledge of the tree structure.

Packing Decoder. The packing decoder works by solving a *tree-to-sequence* problem: it receives the encoded tree as input, then outputs a sequence that represents the order of packing patches. Following the order, patches are packed into one canvas in a sequential manner. The network structure of packing decoder is similar to the filter decoder, *point mechanism* is also adopted to point to a specific node in the input tree at every step.

Layout Generation During packing, the decoder also initializes a layout of patches in canvas. Specifically, a greedy algorithm is employed. Every patch would first occupy the bottom space of the canvas. Then, we will create a new canvas for subsequent patches when the space is insufficient for placing the next patch. The packing process stops when all patches have been placed into canvases. It is worth mentioning that our theoretical speed-up ratio is controllable by setting maximum capacity of each canvas.

3.5. Overall Optimization

RL-based Joint Optimization Intuitively, given an input tree, either filter decoder or packing decoder point to one node at each time step. Take patch filter module as an example, it factorizes the probability of generating set $p(S|G)$ by a chain rule:

$$p(S|G) = \prod_{i=1}^{m} p(S(i)|S(< i), G) \tag{3}$$

where $S(< i)$ denotes patches that have been selected before the i-th time step. We aim to learn the parameters of the policy $p(S|G)$ to assign high probabilities to the set that can balance the detection speed and accuracy. Inspired by the success of Reinforcement Learning (RL) in solving neural combination problems [1], we adopt policy-based reinforcement learning to optimize the two main modules of our framework: PFM and PPM.

First, for PFM and PPM, we design two distinct rewards R_{filter} and R_{pack} to guide their learning process. Specifically,

R_{filter} considers the number of filtered patches, which is defined as

$$R_{\text{filter}} = 1 - \frac{|S| - |V_{\text{mid}}|}{|V_{\text{leaf}}| - |V_{\text{mid}}|} \tag{4}$$

where $|\cdot|$ denotes the size of a node set. It encourages the PFM to select coarser patches to replace fine-grained patches covered by it. Note that since PFM outputs a set, the R_{filter} is designed to be order-irrelevant to fit the learning process. As for R_{pack}, we utilize the detection performance as its guidance:

$$R_{\text{pack}} = AP_{\text{detector}} \tag{5}$$

Since the patch filter results would affect the policy of patch packing, we use the accumulate reward following the policy to guide the policy. The training objectived of patch filter policy can be updated by:

$$R_{\text{filter}} = \lambda R_{\text{filter}} + R_{\text{pack}} \tag{6}$$

where the weight factor λ is utilize to balance the detection speed and performance during optimization.

Succinctly, the training objective of PFM is defined by the objective:

$$J(\boldsymbol{\theta}|G) = \mathbb{E}_{S\sim p_{\boldsymbol{\theta}(.|G)}} R(S|G) \tag{7}$$

where $\boldsymbol{\theta}$ denotes the parameters of PFM. We adopt policy gradient and stochastic gradient descent for optimization. Following the well-known REINFORCE algorithm [22], the above gradient can be formulated as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}|G) = \mathbb{E}_{S\sim p_{\boldsymbol{\theta}(.|G)}}[(R(S|G) - b(G))\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(S|G)] \tag{8}$$

where $b(S)$ denotes a baseline value irrelevant with S – it is used to reduce the variance of the gradients. We draw B samples $G_1, G_2, ..., G_B$ as a batch. According to Monte Carlo sampling, the gradient in Equation (8) can be approximated as follows:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \frac{1}{B}\sum_{i=1}^{B}(R(S_i|G_i) - b(G_i))\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(S_i|G_i) \tag{9}$$

We use the mean of the batch reward as the baseline value and update it progressively during training. Meanwhile, PPM is optimized in a similar way.

4. Experimental Settings

4.1. Dataset and Evaluation Metrics

We conduct experiments on the public detection benchmark PANDA [20] to evaluate our method. PANDA [20] is a gigapixel-level human centric dataset. It provides 18 scenarios (13 for training, 5 for testing) and each scenario contains about 30 images. The image resolution is about 25,000×14,000. We evaluate our approach on full body detection and the AP for *small* (<96×96 pixels), *middle* (96×96 to 288×288 pixels), and *large* (>288×288 pixels) objects are reported. On top of that, we also report **#Pass**, which denotes the the number of runs of the detector, to reflect the inference cost on gigapixel images.

Method	#Pass	AP_{50}				FPS
		Total	Small	Middle	Large	
DS+SW(FR) [20]	-	-	0.190	0.552	0.744	0.07
DS+SW(CR) [20]	-	-	0.227	0.579	0.765	0.05
DS+SW(RN) [20]	-	-	0.221	0.561	0.740	0.1
ClusDet* [23]	7,871	0.718	0.219	0.696	0.782	0.12
DMNet* [6]	1,934	0.540	0.119	0.371	0.714	0.51
DS+SW(FR)* [20]	13,620	0.705	0.203	0.712	0.760	0.07
PAN (Ours) (4×)	3,671	0.715	0.256	0.719	0.768	0.23
PAN (Ours) (6×)	2,565	0.702	0.216	0.700	0.763	0.37
ClusDet+PAN (4×)	3,683	0.713	0.262	0.715	0.767	0.23

Table 1. Comparison with SOTA on PANDA dataset. FR, CR, and RN denote Faster R-CNN [15], Cascade R-CNN [2], and RetinaNet [9]. '#Pass' denotes number of detection runs, '*' denotes our implementation. FPS = 1/(runtime per image). '4×' and '6×' mean the theoretical speed up ratio is set to 4 and 6 in PAN.

4.2. Implementation Details

To facilitate the patch generation for testing images, we use Faster R-CNN [15] with ResNet50 [7] backbone as the coarse detector to get initial rough results. To prepare the data for training the coarse detector, we downsample the original gigapixel images by a factor of 4. Then, a sliding window of 2,048×1,024 pixels is used to decompose the down-sampled image. When evaluating the detection results of our method, we use the same detector and sliding window set up (as mentioned above) but with a downsample factor of 2. For the encoder in the patch filter and patch packing module, we use a fully-connected layer of size 64, while the hidden layers of Tree-LSTM and Chain-LSTM are of size 128. For both filter decoder and packing decoder, we apply an LSTM with hidden size of 128. Our approach is implemented with the PyTorch library. For training, we apply Adam optimizer with an initial learning rate of 0.001. Batch size is 64 for Monte Carlo sampling. All experiments were conducted using a single GeForce GTX 1080Ti GPU.

5. Results and Discussion

5.1. Results on gigapixel dataset PANDA

Compared Methods. We compare our proposed framework with three strong baselines on PANDA. All of them follow the same pipeline but adopt different methods for generating patches.

- DS+SW [20]: It is the combination of two strategies: down-sampling (DS) and sliding window (SW).

Filter			Packing	#Pass	AP_{50}				FPS
Coarse	Fine	Multi			Total	Small	Middle	Large	
✓	-	-	✗	8,979	0.722	0.270	0.711	0.783	0.11
✓	-	-	✓	2,838	0.687	0.144	0.641	0.788	0.34
-	✓	-	✗	15,019	0.715	0.388	0.766	0.716	0.07
-	✓	-	✓	3,862	0.717	0.259	0.731	0.766	0.23
-	-	✓	✗	14,193	0.717	0.390	0.766	0.725	0.07
-	-	✓	✓	3,671	0.715	0.256	0.719	0.768	0.23

Table 2. An ablation study of different modules in our framework. Note that FPS = 1/(runtime per image).

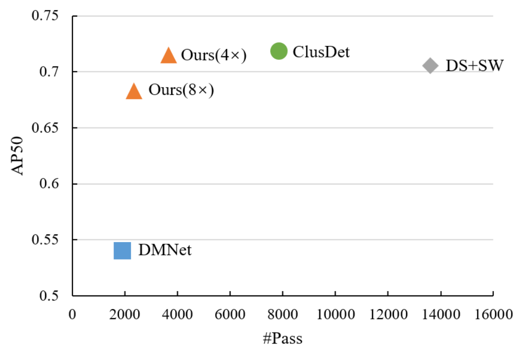


Figure 5. Comparisons of time-performance trade-off among different state-of-the-art approaches.

It firstly down-samples the image and then divides the whole image into regular patches with a fixed-size sliding window.

- ClusDet [23]: It employs a neural network to estimate accurate patches in high-resolution images.
- DMNet [8]: It first predicts an object density map with a trained CNN model. Then it merges the eight-neighbor connected region in the density map into a large candidate region. Finally, it refers the circumscribed rectangle of the candidate region as a patch.

Quantitative Results. The quantitative results are presented in Table 1 and Fig. 5. For a fair comparison, we compare their performance with ours under the same detector Faster R-CNN with ResNet50 backbone. Generally, we observe that our method strikes a great balance between speed and accuracy on gigapixel-level image object detection. Compared to DS+SW (FR), PAN (6×) could boost the inference speed by 5× while maintaining the detection performance. PAN is able to detect one gigapixel-level image in about 2.7 seconds (i.e., FPS=0.37). When applying DS+SW and DMNet to gigapixel-level images, there are huge time costs for object detection (even exceeding 13,000 #Pass with DS+SW). Although DMNet has proposed a density-map-guided method to generate patches, it cannot handle such extreme high resolutions well. DMNet utilizes the object density map to obtain patch candidates. However, the density map based scheme possesses severe trade-off limitations in gigapixel-level images. In their method, objects are more likely to be connected in density maps, and the connected regions in turn, will form larger patches. Thus, although it can reach a faster speed with fewer patches, its performance drop rapidly since many objects cannot be well-detected in too large a patch. Fig. 5 further demonstrates that our PAN strikes a good balance between speed and performance for gigapixel-image object detection.

Tree-LSTM	Chain-LSTM	#Pass	AP_{50}			
			Total	Small	Middle	Large
✓	✗	3,283	0.692	0.137	0.678	0.778
✗	✓	3,561	0.690	0.140	0.678	0.769
✓	✓	3,671	0.715	0.256	0.719	0.768

Table 3. Effect of tree encoder.

Extension to Other Methods. It is flexible for other methods to be ensembled into our proposed framework. To validate it, we extend our framework to the existing work ClusDet. Specifically, the patches generated by ClusDet can be taken as the initial patches. We further filter and pack these patches using our strategy to achieve higher speed. The results are presented at the bottom row of Table 1. It can be observed that compared to the original version of ClusDet, our framework can boost the speed by almost 2 times while maintaining its detection performance.

Visualization Results. To further illustrate the effectiveness of our method, we present the detection result on a gigapixel-level image in Figure 9.

5.2. Ablation Study on PAN

Design of Tree Encoder. Since the relationship among patches is critical towards learning a good arrangement, we first analyze the design of the tree structure encoder. The results of separately employing the Tree-LSTM and/or Chain-LSTM as encoder are shown in Table 3. The three rows shown in Table 3 represent the Tree-LSTM encoder, Chain-LSTM encoder, and Mixed-LSTM encoder, from top to bottom. We can observe that, compared with Chain-LSTM encoder, the Tree-LSTM encoder uses less patches to achieve comparable results. We note that the Tree-LSTM encodes the relationship between coarse-grained patches and their

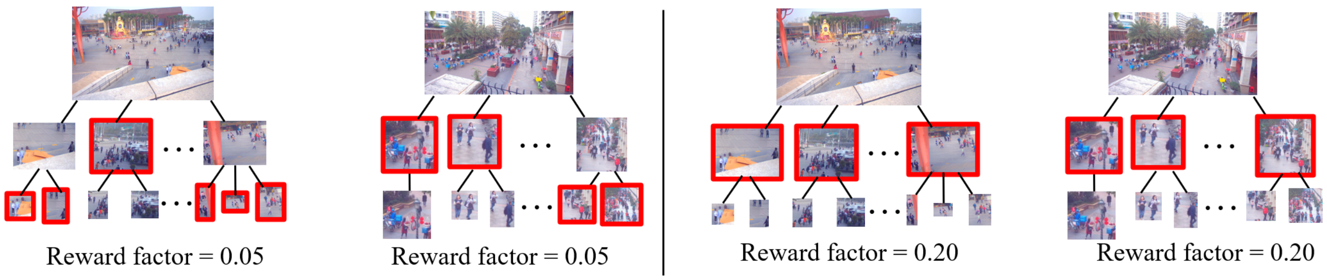


Figure 6. Two examples from PANDA dataset. The red boxes indicate the patches filtered by the network.

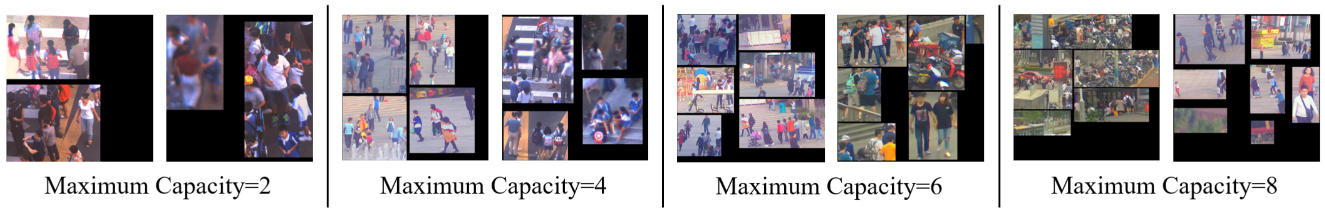


Figure 7. Examples of patch packing results using different capacity.

Detector	DS+SW	Ours (6×)
YOLOv3 [14]	0.518	2.115
SSD [12]	0.515	2.088

Table 4. Extension to different detectors. FPS (*i.e.,* 1/(runtime per image)) of using different object detection algorithms are given.

λ	#Pass	AP_{50}			
		Total	Small	Middle	Large
0.05	3892	0.690	0.146	0.683	0.767
0.10	3671	0.715	0.256	0.719	0.768
0.20	3127	0.672	0.102	0.626	0.785

Table 5. An ablation study on using different values of λ.

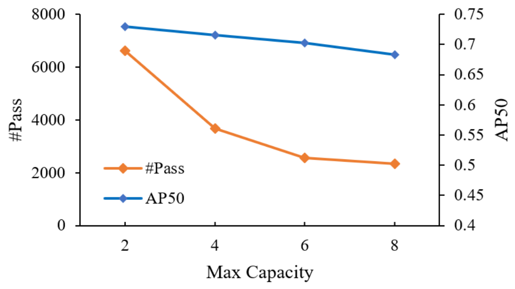


Figure 8. An ablation study on different maximum capacities.

corresponding fine-grained patches (*i.e.* inter-layer relationship), which is not exemplified by the Chain-LSTM. Thus, it can utilize such relationships to encourage more fine-grained patches to be replaced and filtered by coarser patches to achieve higher speed. However, even with the relatively higher speed compared to Mixed-LSTM encoder, the detection performance of Tree-LSTM encoder drops in the absence of the crucial intra-layer relationship. As such, by considering both inter- and intra-layer relationships (*i.e.* Mixed-LSTM), we can get a better trade-off between speed and performance.

Effect of Patch Filter. We examine how the granularity of patches affects the detection speed and accuracy. Table 2 shows the results on PANDA dataset. We evaluate our method by selecting patches from three granularities: *coarse*-grained, *fine*-grained, and *multi*-grained. Selecting coarse-grained patches means only focusing on the middle layer patches on the tree. We can see that it uses fewer patches, but the detection accuracy degenerates. Selecting fine-grained patches is equivalent to focusing on the leaf layer patches on the tree. We could get higher accuracy, but more patches are needed. By selecting multi-grained patches, we use relatively fewer patches while simultaneously maintaining detection performance. Figure 6 shows two example images from PANDA dataset. The patches selected by PFM are highlighted in red. We can observe that they are distributed in both middle and leaf layers of the tree, which shows that our approach will adaptively choose patches from different granularities.

Effect of Patch Packing. We also examine the effect of patch packing. The default maximum capacity of a canvas C is set as 4. As shown in Table 2, we evaluate our PPM based on three kinds of filtering strategies. The patch packing module can greatly reduce the patch number to achieve high time efficiency. Meanwhile, Figure 7 shows several patch packing results when different values of the maximum capacity of the canvas are used. Under most circumstances, the network would try to reach the maximum capacity of each canvas to reduce the number of canvases present in one image.

Effect of Maximum Capacity C. As we have mentioned in Section 3.4, the theoretical speed-up ratio can be controlled

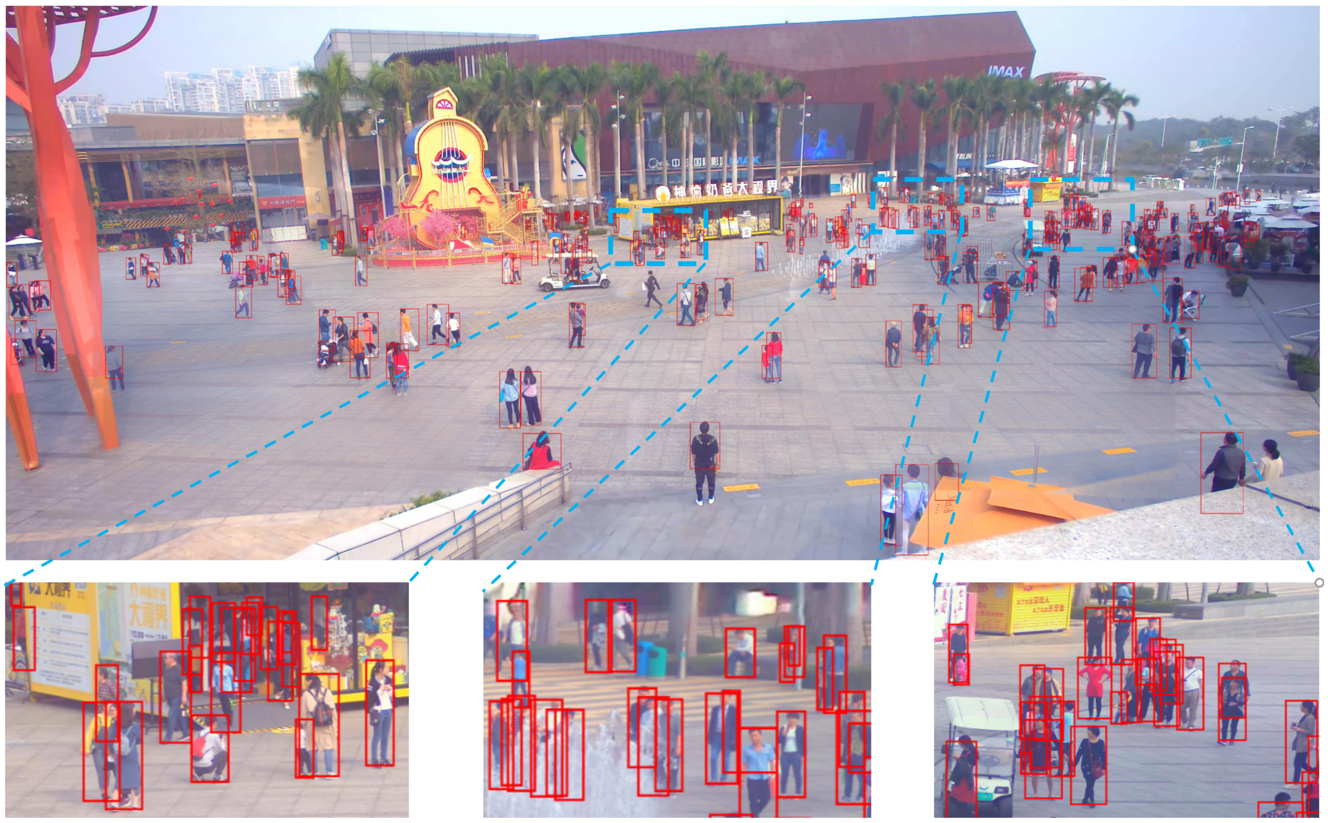


Figure 9. Example of detection results using our method.

by setting the maximum canvas capacity C, which denotes the maximum number of patches a canvas could contain. Figure 8 presents the influence of canvas capacity on the detection speed and accuracy. As shown here, the time cost is significantly reduced while a slight performance decline was observed as expected. This is however, advantageous as the detection speed could be boosted by $8\times$ (*i.e.*, $C = 8$) at minimal impact to the performance, highlighting its feasibility for real-world applications.

Understanding Speed-Performance Trade-off. The factor λ in Equation (6) plays the role of adjusting the balance of detection speed and performance. Table 5 presents the results under various factor settings. With the increase of λ, more fine-grained patches are encouraged to be replaced by coarser patches, and thus #Pass also decreases. The performance degenerates when λ reaches the maximum value of 0.2. Interestingly, the PAN strikes a good balance between detection speed and performance with $\lambda = 0.1$.

Extension to Lightweight Detectors. Since our approach only modifies the input images, it is pluggable to any object detection algorithms besides Fast R-CNN. To validate the acceleration of object detectors under the gigapixel scenario, we further extend our method to two widely used lightweight detectors: YOLOv3 [14] and SSD [12]. The results in Table 4 are conclusive that our method can indeed increase the efficiency of these detectors by about fourfold.

6. Conclusion

This paper introduces a new patch arrangement framework for fast object detection on gigapixel-level images. Under this framework, we devise a Patch Arrangement Network (PAN), which increases the efficiency of detection by learning to arrange patches. Two arrangement modules were proposed: the first, patch filter module (PFM) selects and filters patch candidates across granularities, then a patch packing module (PPM) sequentially packs the remaining patches together into canvases. The overall framework is jointly optimized by policy-based reinforcement learning. The extensive experiments conducted on a gigapixel level image dataset PANDA highlights the benefits of our approach– an improvement of inference speed on gigapixel images by $5\times$, while maintaining an ideal performance.

Acknowledgement. This paper is supported in part by the following grants: National Key Research and Development Program of China Grant (No.2018AAA0100400), National Natural Science Foundation of China (No.U21B2013, 61971277), HWUM JWS 2021 (Project AVALON). We thank the PANDA dataset group for providing the testing set and implementation details of their methods.

References

[1] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016. 4

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018. 5

[3] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995. 3

[4] Ting-Wu Chin, Ruizhou Ding, and Diana Marculescu. Adascale: Towards real-time video object detection using adaptive scaling. *Proceedings of Machine Learning and Systems*, 1:431–441, 2019. 2

[5] Sutao Deng, Shuai Li, Ke Xie, Wenfeng Song, Xiao Liao, Aimin Hao, and Hong Qin. A global-local self-adaptive network for drone-view object detection. *IEEE Transactions on Image Processing*, 30:1556–1569, 2020. 2

[6] Mingfei Gao, Ruichi Yu, Ang Li, Vlad I Morariu, and Larry S Davis. Dynamic zoom-in network for fast object detection in large images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6926–6935, 2018. 2, 5

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 5

[8] Changlin Li, Taojiannan Yang, Sijie Zhu, Chen Chen, and Shanyue Guan. Density map guided object detection in aerial images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 190–191, 2020. 1, 2, 6

[9] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017. 5

[10] Weiyao Lin, Xiaoyi He, Wenrui Dai, John See, Tushar Shinde, Hongkai Xiong, and Lingyu Duan. Key-point sequence lossless compression for intelligent video analysis. *IEEE MultiMedia*, 27(3):12–22, 2020. 2

[11] Weiyao Lin, Huabin Liu, Shizhan Liu, Yuxi Li, Rui Qian, Tao Wang, Ning Xu, Hongkai Xiong, Guo-Jun Qi, and Nicu Sebe. Human in events: A large-scale benchmark for human-centric video analysis in complex events. *arXiv preprint arXiv:2005.04490*, 2020. 2

[12] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2, 7, 8

[13] Mahyar Najibi, Bharat Singh, and Larry S Davis. Autofocus: Efficient multi-scale inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9745–9755, 2019. 2

[14] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016. 2, 7, 8

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015. 2, 5

[16] Vít Ržička and Franz Franchetti. Fast and accurate object detection in high resolution 4k and 8k video using gpus. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7. IEEE, 2018. 2

[17] Kai Sheng Tai, Richard Socher, and Christopher D Manning. Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*, 2015. 3

[18] Burak Uzkent, Christopher Yeh, and Stefano Ermon. Efficient object detection in large images using deep reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1824–1833, 2020. 2

[19] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *arXiv preprint arXiv:1506.03134*, 2015. 4

[20] Xueyang Wang, Xiya Zhang, Yinheng Zhu, Yuchen Guo, Xiaoyun Yuan, Liuyu Xiang, Zerun Wang, Guiguang Ding, David Brady, Qionghai Dai, et al. Panda: A gigapixel-level human-centric video dataset. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 3268–3278, 2020. 1, 2, 5

[21] Yi Wang, Youlong Yang, and Xi Zhao. Object detection using clustering algorithm adaptive searching regions in aerial images. In *European Conference on Computer Vision*, pages 651–664. Springer, 2020. 2

[22] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992. 5

[23] Fan Yang, Heng Fan, Peng Chu, Erik Blasch, and Haibin Ling. Clustered object detection in aerial images. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 8311–8320, 2019. 1, 2, 5, 6

[24] Pengfei Zhu, Longyin Wen, Xiao Bian, Haibin Ling, and Qinghua Hu. Vision meets drones: A challenge. *arXiv preprint arXiv:1804.07437*, 2018. 2