MaxpoolNMS: Getting Rid of NMS Bottlenecks in Two-Stage Object Detectors

Lile Cai[1] Bin Zhao[2] Zhe Wang[1] Jie Lin[1] Chuan Sheng Foo[1]
Mohamed Sabry Aly[3]
Vijay Chandrasekhar[1]
{[1]Institute for Infocomm Research, [2]Institute of Microelectronics}, A*STAR, Singapore
[3]Nanyang Technological University, Singapore
{caill,wang_zhe,lin-j,foo_chuan_sheng,vijay}@i2r.a-star.edu.sg
zhaobin@ime.a-star.edu.sg, msabry@ntu.edu.sg

Abstract

Modern convolutional object detectors have improved the detection accuracy significantly, which in turn inspired the development of dedicated hardware accelerators to achieve real-time performance by exploiting inherent parallelism in the algorithm. Non-maximum suppression (NMS) is an indispensable operation in object detection. In stark contrast to most operations, the commonly-adopted GreedyNMS algorithm does not foster parallelism, which can be a major performance bottleneck. In this paper, we introduce MaxpoolNMS, a parallelizable alternative to the NMS algorithm, which is based on max-pooling classification score maps. By employing a novel multi-scale multi-channel max-pooling strategy, our method is $20\times$ *faster than GreedyNMS while simultaneously achieves comparable accuracy, when quantified across various benchmarking datasets, i.e., MS COCO, KITTI and PASCAL VOC. Furthermore, our method is better suited for hardware-based acceleration than GreedyNMS.*

1. Introduction

Deep neural networks (DNNs) have caused a major leap in object detection accuracy. State-of-the-art DNN-based object detection algorithms can be broadly classified into one-stage and two-stage methods. One-stage object detectors (e.g., YOLO [24] and SSD [22]) operate in a sliding-window manner that makes prediction for densely sampled locations in the input image. Alternatively, two-stage approaches, such as Faster R-CNN [25] and R-FCN [5], first generate a sparse set of region proposals and then perform a second stage prediction to classify each proposal and refine its location. Two-stage methods consistently achieve higher accuracy than one-stage methods, but are significantly slower [18].

Two-stage object detectors consist of a (first stage) region proposal network (RPN) that hypothesizes candidate object locations and a (second stage) detection network that refines region proposals. The RPN and detection network share the same feature extractor network. Figure 1 displays the typical blocks of two-stage object detectors. Typically a significant portion of execution time is consumed in the feature extractor, region proposal and object detection networks. These layers contain convolution and pooling operations, which in principle can be mapped to a highly-parallel hardware accelerator (e.g., Google TPU [19]) – this is not the case with remaining blocks, i.e., Non-Maximum Suppression (NMS).

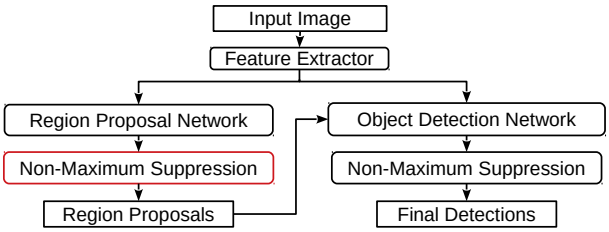


Figure 1. System diagram of two-stage object detectors. This work tackles the NMS in region proposal network (denoted in red box).

NMS is an essential block as it removes duplicate detections, hence reducing false positives. Both the region proposal network and object detection network employ NMS as a post-processing step. The commonly-adopted GreedyNMS [7] is a simple hand-crafted method. When applied in region proposal network, it first sorts all the candidate detection boxes according to their objectness scores, followed by two nested loops to greedily select high score boxes and delete other boxes that overlap significantly with the selected ones. The inner loop is parallelizable, but the outer loop is sequential in nature – without examining the preceding box first, it cannot be decided whether the follow-

ing box should be selected. When applying GreedyNMS in object detection network, the number of processed boxes is much smaller than that for RPN (e.g., 300 vs. 6000), and it is conducted separately for each object class.

NMS can induce a performance bottleneck as it cannot be easily parallelized. This is illustrated in Fig. 2. As GPUs become increasingly powerful, the time spent on convolution operations reduces significantly, while the time spent on NMS is not affected and gradually occupies an increasing portion of the total execution time.

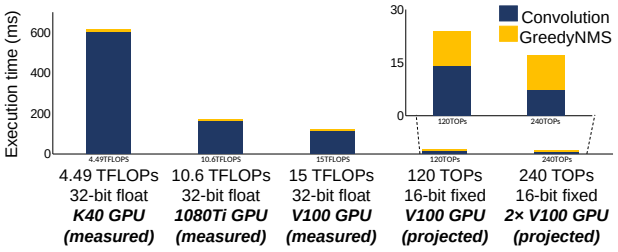


Figure 2. Execution time of convolution and GreedyNMS on different GPU platforms. Time was measured when the GPU was launched to run a Faster R-CNN detection network with ResNet-152-V2 backbone [14]. The last two bars are projected values since 16-bit fixed-point Faster R-CNN is not available[1]. The convolution includes all the operations in ResNet block 0–4, and the GreedyNMS is the one in RPN.

In this paper, we introduce a scalable and parallelizable approach to perform NMS in region proposal network. The key insight is that an object proposal corresponds to a peak in the objectness score map, hence we leverage max-pooling to obtain this peak. The method is thus called *MaxPoolNMS*. By employing a novel multi-scale multi-channel max-pooling strategy, our method simultaneously obtains comparable accuracy and up to 20× speed-up versus GreedyNMS. Our method avoids computing intersection over union (IoU) and solely relies on max-pooling operations, and thus is highly parallelizable.

2. Related Work

Convolutional object detectors Modern convolutional object detectors follow either a one-stage, proposal-free paradigm, or a two-stage, proposal-based paradigm. One-stage detectors apply the object classifier and location regressor to a densely sampled set of local windows in different locations, scales and aspect ratios. YOLO [24], SSD [22] and the recently proposed RetinaNet [20] are representative architectures for one-stage object detectors. Two-stage paradigm was popularized by the R-CNN framework [11], and was further improved in terms of speed by Fast R-CNN [10]. Faster R-CNN [25] incorporates the region proposal function into deep learning framework by introducing a region proposal network that shares the same feature extraction network with Fast R-CNN. Mask R-CNN [12] adds a mask prediction branch to Faster R-CNN to obtain pixel level object detections. Cascade R-CNN [2] adds more stages to Faster R-CNN and uses the output of previous stages to train the next stage detector of higher quality. R-FCN [5] is another line of work for two-stage detectors, which aims to share more computation among the proposal boxes in the second stage by employing position-sensitive prediction maps. Huang et al. [18] conduct an extensive study to evaluate the speed/accuracy trade-offs for SSD, Faster R-CNN and R-FCN, and conclude that the most accurate models are consistently achieved by two-stage methods. Recent work on single shot instance segmentation shows that the proposal-based approach can not only achieve better mAP but also runs significantly faster than sliding window-based approach for mask prediction [3].

NMS in object detection NMS has been employed as a post-processing step in several generations of detectors. The de facto algorithm, GreedyNMS, was first demonstrated in [7] to surpass other approaches for human detection. Since then, it has been a standard component in object detection and widely used in one-stage and two-stage detectors. Soft-NMS [1] is a variant of GreedyNMS that decays the score of neighboring detections instead of totally removing it. It has been shown to improve the mAP of object detectors by around 1-2%. However, it was only applied to replace GreedyNMS in the second stage detection network in the paper, and it is not clear whether it can also replace the one in region proposal network. Fitness NMS [26] weights the original classification score by a predicted *fitness* value so that boxes with high IoU with ground truth boxes can have higher scores. Another line of research is to replace GreedyNMS with learnable network architectures so that the model can be trained fully end-to-end. The idea is to predict only one high scoring detection for an object and the key to achieve this is to design features that condition on multiple detections on the same object. Tnet [15] is a convnet for NMS, where the IoU values between a detection and its neighboring boxes are used together with score values to make sparse prediction. Gnet[16] computes pairwise context features between a detection and its neighbors to generate the feature representation. Relation network [17] computes the relation feature for a detection by a weighted sum of appearance features from other detections. Due to pairwise computation and additional network architectures, these methods are more suited to deploy in the second stage detection network where the number of processed boxes is small. Also, all these work focused on improving the ac-

[1]Projected execution time is derived using the total number of convolution operations in the entire Faster R-CNN network, total memory accesses for weights and activations, and reported computing performance and memory-access bandwidth of the examined GPU.

curacy of GreedyNMS, while the speed and parallelism aspects have been left unexplored.

3. Method

Modern convolutional object detector utilizes multi-scale "anchors" to realize scale-invariant object detection. In RPN, a binary classifier (i.e., object/background) and a location regressor are trained for each anchor. Applying the trained classifier to densely sampled locations of the input image (typically in a stride of 16) produces the objectness score maps. Figure 3 displays the 12 objectness score maps for an input image from the KITTI dataset. We use 4 scales $\{64^2, 128^2, 256^2, 512^2\}$ and 3 aspect ratios (width to height) $\{1:2, 1:1, 2:1\}$ for anchor generation. We make the following observations on these maps.

1. Objects correspond to peaks on the map. This is due to the fact that during training, only anchors with a high IoU (e.g., above 0.7) with ground truth boxes are considered as positive samples, and thus only anchors containing objects can have high objectness scores during testing. Anchors in the neighborhood of the peak anchor can also have high response since similar input should produce similar output for a continuous classification function.
2. The score map is scale and aspect-ratio specific and only responds to objects of around that size. This is because small (large) anchor boxes can only be matched to small (large) ground truth objects, and thus can only be trained to detect small (large) objects.
3. An object can have high response on more than one score maps. This is due to the fact that the actual object scales and aspect ratios are continuous, yet the predefined anchor sizes are discretized. An object can have a size that falls between two neighboring anchor sizes and thus has a strong response on both maps. Take the image shown in Fig. 3 as an example. The green car on the right has high response in two neighboring aspect ratios (scores maps (e) and (i)), as well as in two neighboring scales (score maps (e) and (f)).

These observations suggest that max-pooling operations could be sufficient to obtain a meaningful set of object proposals. As an object is a local maximum on the score map, it can be picked up by max-pooling, whose function is to select the maximum value in a local window. The key parameters here are the kernel sizes and strides for max-pooling. Observation 2 provides insights into how to set the parameters effectively. As each score map is focused on detecting objects of a specific size, we set the kernel size and stride of max-pooling to be proportional to the anchor box size. To further reduce false positives, Observation 3 suggests that we can perform multi-channel max-pooling across neighboring score maps to remove duplicate responses for the same object. After obtaining the proposals returned by max-pooling, we sort the proposals by their scores and output a fixed number of proposals for the second stage prediction. We provide details on how to conduct multi-scale and multi-channel max-pooling in the following.

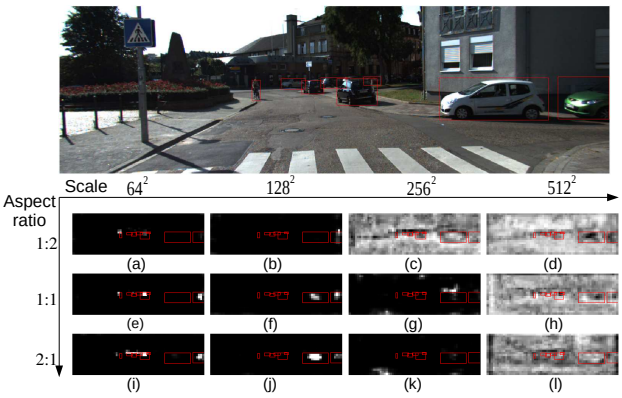


Figure 3. The 12 objectness score maps for an input image from the KITTI dataset.

3.1. Multi-scale max-pooling

As score maps are generated by multi-scale anchors, it is natural to use multi-scale kernel sizes for different score maps when conducting max-pooling. Given a score map for an anchor of size $h \times w$, an object of size $h \times w$ will be roughly projected to a $\frac{h}{s} \times \frac{w}{s}$ area on the map, where s is the stride of the map. Suppose the centers of two objects are αw apart on the image, they will correspond to two peaks that are $\frac{\alpha w}{s}$ apart on the score map. In order to not miss either peak for object proposal, the kernel size and stride of the max-pooling operation along the x dimension can be set as:

$$ksize_x, stride_x = \max(1, \text{round}(\frac{\alpha w}{s})). \quad (1)$$

Similarly for y dimension, we set the parameters as:

$$ksize_y, stride_y = \max(1, \text{round}(\frac{\alpha h}{s})). \quad (2)$$

Table 1 presents the kernel sizes and strides for the 12 score maps computed by Eq. 1 and Eq. 2 with $\alpha = 0.25$ and $s = 16$. The algorithm to perform multi-scale max-pooling is summarized in Algorithm 1.

Table 1. The kernel sizes and strides for the max-pooling operation on the 12 score maps when $\alpha = 0.25$ and $s = 16$.

	64^2	128^2	256^2	512^2
1:2	1x1	1x3	3x6	6x11
1:1	1x1	2x2	4x4	8x8
2:1	1x1	3x1	6x3	11x6

3.2. Multi-channel max-pooling

An object can produce multiple peaks on neighboring score maps. This property can be utilized to reduce false

Algorithm 1 Multi-Scale Max-Pooling

input:
M^{score}: array of $n_{ar} \times n_{scl}$ score maps, where n_{ar} is the number of aspect ratios and n_{scl} is the number of scales
output:
M^{max}: array of $n_{ar} \times n_{scl}$ maps of the same size as M^{score} that record the maximum value within a pooling window
parameters:
k^x, k^y: array of $n_{ar} \times n_{scl}$ kernel sizes computed by Eq. 1 and Eq. 2
s^x, s^y: array of $n_{ar} \times n_{scl}$ strides computed by Eq. 1 and Eq. 2
functions:
maxpool($x, ksizes, strides$): perform max-pooling on input x with the given pooling sizes and strides. $ksizes$ and $strides$ are 3-D vectors specifying the pooling parameters along the x, y and channel dimension.
procedure
  $M^{max} \leftarrow M^{score}$
  for $i \in [0, 1, \cdots, n_{ar} - 1]$ **do**
    for $j \in [0, 1, \cdots, n_{scl} - 1]$ **do**
      maxpool($M^{max}_{i,j}, [k^x_{i,j}, k^y_{i,j}, 1], [s^x_{i,j}, s^y_{i,j}, 1]$)
    end for
  end for
end procedure

positives. Specifically, we apply a 2-channel max-pooling on the two neighboring score maps. Two maps are considered to be "neighbors" if they are horizontally or vertically connected in the scale-aspect ratio grid shown in Fig. 3. To perform 2-channel max-pooling across aspect ratios (Algorithm 2), we first perform max-pooling on $M^{score}_{i,j}$, then concatenate the max-pooling results of $M^{score}_{i,j}$ with $M^{score}_{i+1,j}$ in the channel dimension, and then perform 2-channel max-pooling on the concatenated map. Max-pooling across scales (Algorithm 3) is conducted in a similar manner. The difference between Algorithm 2 and Algorithm 3 is the choice of kernel sizes and strides for 2-channel max-pooling. For max-pooling across scales, we find it necessary to use the smaller one of the two maps' kernel sizes and strides, otherwise peaks on the smaller scale score map are incorrectly removed.

Algorithm 2 Multi-Channel Max-Pooling Across Aspect Ratios

procedure
  $M^{max} \leftarrow M^{score}$
  for $j \in [0, 1, \cdots, n_{scl} - 1]$ **do**
    for $i \in [0, 1, \cdots, n_{ar} - 2]$ **do**
      if $i == 0$ **then**
        maxpool($M^{max}_{i,j}, [k^x_{i,j}, k^y_{i,j}, 1], [s^x_{i,j}, s^y_{i,j}, 1]$)
      end if
      $M^{concat} \leftarrow$ concatenate($M^{max}_{i,j}, M^{max}_{i+1,j}$)
      maxpool($M^{concat}, [k^x_{i+1,j}, k^y_{i+1,j}, 2], [s^x_{i+1,j}, s^y_{i+1,j}, 2]$)
    end for
  end for
end procedure

Algorithm 3 Multi-Channel Max-Pooling Across Scales

procedure
  $M^{max} \leftarrow M^{score}$
  for $i \in [0, 1, \cdots, n_{ar} - 1]$ **do**
    for $j \in [0, 1, \cdots, n_{scl} - 2]$ **do**
      $M^{concat} \leftarrow$ concatenate($M^{max}_{i,j}, M^{max}_{i,j+1}$)
      maxpool($M^{concat}, [k^x_{i,j}, k^y_{i,j}, 2], [s^x_{i,j}, s^y_{i,j}, 2]$)
      if $j == n_{scl} - 2$ **then**
        maxpool($M^{max}_{i,j+1}, [k^x_{i,j+1}, k^y_{i,j+1}, 1], [s^x_{i,j+1}, s^y_{i,j+1}, 1]$)
      end if
    end for
  end for
end procedure

4. Experimental Results

4.1. Accuracy benchmarking

In this section, we report the detection accuracy of MaxpoolNMS on three benchmarking datasets, i.e., MS COCO [21], KITTI [9] and PASCAL VOC [8]. The pooling strategy in Algorithm 1, 2 and 3 is denoted as MS, MC-AR and MC-SCL, respectively. The α used in the experiments is 0.25 for all datasets. We will investigate the sensitivity of α on accuracy in Section 5. Our experiment is based on the Tensorflow object detection API [18]. For anchor generation, we use 4 scales $\{64^2, 128^2, 256^2, 512^2\}$ and 3 aspect ratios (width to height) $\{1:2, 1:1, 2:1\}$ in all the experiments. The number of proposals is 300.

COCO For COCO dataset, we directly use the pretrained detection models with ResNet-101-V1 [13] backbone provided in the Tensorflow detection model zoo and replace GreedyNMS with MaxpoolNMS in inference. Image sizes are scaled to lie within [600, 1024]. The model is evaluated on COCO test-dev2017 that contains 20,288 images and the results are reported in Table 2. It can be seen that MaxpoolNMS is able to match the accuracy of GreedyNMS for different matching IoUs and object sizes.

KITTI For KITTI dataset, we randomly split 1870 images from the training set as the validation subset and train on the rest 5611 images. We use the ResNet-50-V2 [14] as the backbone network. The optimizer is SGD with momentum 0.9 and random horizontal flips is used for data augmentation. GreedyNMS is still employed in finetuning and is replaced with MaxpoolNMS in inference. For Faster R-CNN, the network was trained for 100k iterations with a batch size of 1. The initial learning rate is 0.001, and decayed by a factor of 0.1 at 75k iterations. For R-FCN, we trained for 150k iterations and other settings are the same as training Faster R-CNN. Input images are resized so that the maximum dimension is 1024. The results are reported in Table 3. MaxpoolNMS can achieve comparable mean average precision (mAP) as GreedyNMS for testing samples at various difficulty levels from easy to hard.

Table 2. Evaluation results on COCO test-dev2017.

	mAP@IoU			mAP@area			Recall@maxDets			Recall@area		
	0.50:0.95	0.50	0.75	S	M	L	1	10	100	S	M	L
Faster R-CNN+ResNet-101-V1												
GreedyNMS	**30.1**	48.0	**32.3**	10.4	32.8	**46.2**	**26.5**	37.3	37.8	**12.9**	40.5	**59.7**
MS	**30.1**	**48.2**	**32.3**	**10.5**	**32.9**	46.1	**26.5**	**37.4**	**37.9**	**12.9**	**40.8**	59.5
MC-AR	**30.1**	48.1	**32.3**	**10.5**	32.8	45.8	26.4	37.1	37.5	**12.9**	40.7	58.7
MC-SCL	29.7	47.6	31.9	**10.5**	32.5	44.6	26.2	36.6	37.1	**12.9**	40.3	56.9
R-FCN+ResNet-101-V1												
GreedyNMS	27.0	45.2	28.7	9.3	30.0	39.2	23.8	33.3	33.9	11.3	36.8	**51.3**
MS	**27.3**	**45.5**	**28.9**	**9.6**	**30.3**	**39.4**	**24.0**	**33.5**	**34.0**	**11.5**	**37.0**	51.1
MC-AR	27.2	45.3	28.8	9.1	30.1	39.3	23.8	33.2	33.7	11.4	36.9	50.5
MC-SCL	26.9	44.9	28.5	9.2	29.8	38.2	23.7	33.0	33.4	**11.5**	36.6	49.4

Table 3. Evaluation results on KITTI val.

		Car			Pedestrian			Cyclist		
	mAP	Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
Faster R-CNN+ResNet-50-V2										
GreedyNMS	81.29	98.68	94.56	88.43	86.92	81.09	75.36	86.92	82.57	80.09
MS	81.35	98.83	**95.17**	88.51	86.96	80.88	75.11	**87.27**	82.74	80.42
MC-AR	81.27	98.67	95.06	**88.65**	87.21	81.14	75.61	86.25	81.98	79.56
MC-SCL	**81.78**	**98.84**	94.51	88.63	**87.51**	**81.72**	**76.14**	87.12	**83.09**	**80.58**
R-FCN+ResNet-50-V2										
GreedyNMS	80.51	96.39	93.26	86.76	88.38	81.43	76.01	**86.33**	81.42	78.76
MS	80.42	96.85	93.49	87.21	88.28	81.16	75.58	86.28	81.36	78.48
MC-AR	80.29	96.80	93.40	86.80	88.17	81.36	75.68	85.69	81.19	78.40
MC-SCL	**81.03**	**96.97**	**93.65**	**87.39**	**88.81**	**82.02**	**76.65**	85.79	**81.82**	**79.05**

VOC We trained the network on the union set of VOC2007 trainval and VOC2012 trainval (07+12), which contains a total of 16,551 (5011 + 11540) images. The network was trained with a batch size of 4 on 4 GTX 1080Ti GPUs. For Faster R-CNN, the number of iterations and learning rate setting are the same as KITTI. For R-FCN, we trained for 150k iterations. The initial learning rate is 0.003, decayed by 0.1 at 80k and 120k iterations. Images are resized so that the minimum dimension is 600. The model is then evaluated on the VOC2007 test that contains 4952 images. The results are shown in Table 4 . It can be seen that MC-AR and MS both achieve comparable mAP as GreedyNMS, yet there is some noticeable drop (2-5%) in mAP for MC-SCL. We will investigate why MC-SCL caused the mAP drop in Section 5.

4.2. Speed benchmarking

This section compares the execution time of MaxpoolNMS with GreedyNMS. We implemented the entire Faster R-CNN pipeline in C++, and plugged in the GreedyNMS or MaxpoolNMS for the first stage NMS. We trained Faster R-CNN with ResNet-152-V2 backbone network on KITTI and the C++ model loaded weights from the Tensorflow-trained model. The input image size is $1920 * 580$ and timing is averaged over 100 images randomly selected from KITTI.

Letting n denote the average number of boxes before NMS, and m the number of boxes after NMS, we conduct speed benchmarking with different n and m, and provide the timing breakdown in Table 5. It can be seen that our method can achieve significant speed-up ($5\times$–$20\times$) regardless of different n and m values. The time complexity of the two algorithms can be analyzed as follows. For GreedyNMS, the complexity is $O(n\log(n))$ (step 1: sorting) + $O(nm)$ (step 2: 2 nested loops). For MaxpoolNMS, the time complexity is $O(n)$ (step 1: max-pooling) + $O(n\log(n))$ (step 2: sorting). Our method is more efficient than GreedyNMS in two aspects : (1) sorting is performed *after* max-pooling in our method, and thus there are far less elements to sort; (2) max-pooling only involves simple comparison operation, while the 2 nested loops requires intensive computation of IoU which is a much more expensive operation involving division.

5. Discussions

Why does MaxpoolNMS work? GreedyNMS can be viewed as a special type of max-pooling, where the pooling window size and stride are adaptively determined by IoU and objectness scores, while MaxpoolNMS is max-pooling with fixed pooling parameters. Figure 4 compares the boxes selected by the two methods. Comparing Fig. 4 (c) with Fig. 4 (d) and (e), it can be seen that the two methods select an overlapping but not exactly the same set of boxes. Quantitatively, the overlapping percentage (i.e., the percentage of boxes selected by both methods) is around 40% – 50% as shown in Table 6. It may be surprising that given such a low overlap, the two methods can actually achieve comparable mAP. To explain this, we draw the precision vs. recall curves for region proposals and final detections in Fig. 5. It can be seen that the region proposals selected by MaxpoolNMS has lower precision than GreedyNMS for different recall values. However, the precision gap between GreedyNMS and MaxpoolNMS is eliminated after the second stage prediction. We thus conclude that the effectiveness

Table 4. Evaluation results on VOC2007 test.

	mAP	aero	bike	brid	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Faster R-CNN+ResNet-50-V2																					
GreedyNMS	74.28	75.32	78.92	**77.13**	**68.11**	61.60	81.21	**84.18**	84.15	56.85	82.02	62.75	**82.29**	83.66	76.76	80.60	45.80	77.91	**71.69**	**78.85**	75.85
MS	73.85	74.54	79.13	76.68	67.31	59.36	80.81	83.96	84.16	**57.13**	**82.12**	61.99	81.11	83.69	76.96	80.57	46.28	78.07	70.77	77.11	75.24
MC-AR	**74.38**	**75.79**	**80.21**	76.70	67.44	**62.50**	**81.52**	84.05	**84.23**	56.11	82.07	**63.32**	81.49	83.62	**77.18**	**80.71**	**46.81**	**79.16**	70.67	77.44	**76.53**
MC-SCL	69.16	70.87	71.02	69.77	57.51	56.32	75.09	80.26	78.37	50.67	79.30	59.45	77.39	**84.25**	71.44	78.06	41.17	75.81	67.40	68.17	70.79
R-FCN+ResNet-50-V2																					
GreedyNMS	70.93	**78.84**	77.93	71.41	**62.08**	55.85	77.58	82.17	80.97	52.53	77.74	57.77	77.94	**81.85**	75.56	77.79	**42.82**	73.56	**66.41**	74.86	**73.02**
MS	71.01	77.96	79.15	72.18	60.83	55.95	78.99	82.21	81.56	**52.76**	**78.31**	57.32	**78.45**	81.27	**76.38**	**77.83**	42.81	73.53	65.39	74.74	72.69
MC-AR	**71.31**	78.14	**79.30**	**72.67**	60.70	**56.75**	**79.72**	**82.77**	**82.25**	51.58	78.01	**59.05**	78.36	81.30	75.76	77.52	41.96	**74.99**	66.35	**77.35**	71.57
MC-SCL	69.03	77.32	76.66	67.69	58.40	52.64	75.31	81.50	79.44	51.08	78.07	57.38	76.53	81.63	74.12	76.58	41.07	70.77	63.24	71.55	69.66

Table 5. Timing breakdown (in ms) for GreedyNMS (step 1: sorting, step 2: 2 nested loops) and MaxpoolNMS (step 1: max-pooling, step 2: sorting). Timing is reported as *total timing (sorting/the other step)*. We use CPU implementation for both methods.

Score Threshold	0 (n=53280)	0.001 (n=9745)	0.05 (n=5408)
GreedyNMS (m=300)	31.69(18.19/13.50)	16.30(2.83/13.46)	13.19(1.47/11.72)
GreedyNMS (m=100)	22.33(18.93/3.40)	6.13(2.87/3.26)	4.87(1.56/3.31)
MS	6.67(6.22/0.45)	0.82(0.39/0.43)	0.51(0.07/0.44)
MC-AR	3.44(2.75/0.69)	0.86(0.23/0.64)	0.66(0.04/0.64)
MC-SCL	4.27(3.30/0.97)	1.10(0.26/0.84)	0.86(0.06/0.80)

of MaxpoolNMS is attributed to the following two factors: 1) MaxpoolNMS can select a different region proposal than GreedyNMS that is equivalently effective to detect an object, as illustrated by an example in Fig. 6; 2) The second stage GreedyNMS (which we do not replace with MaxpoolNMS) can remove duplicates for final detections.

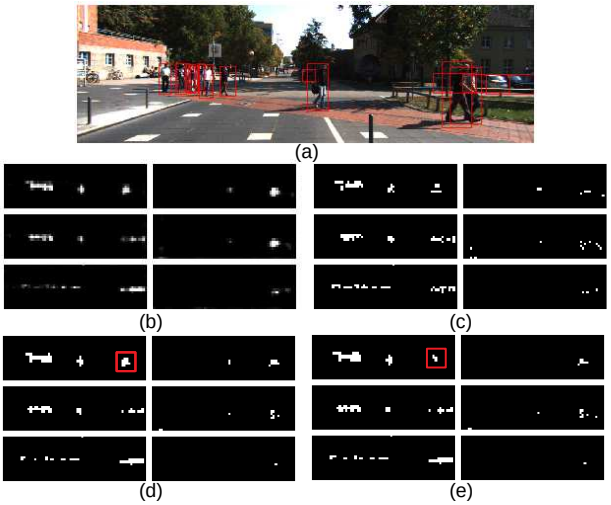


Figure 4. Comparison of the bounding boxes (i.e., region proposals) selected by GreedyNMS and MaxpoolNMS. (a) An input image from KITTI. (b) The objectness score maps for scale 64^2 and 128^2 (from left to right) and aspect ratios $\{1:2, 1:1, 2:1\}$ (from top to bottom). (c) Bounding boxes selected by GreedyNMS (a white pixel indicates that the box corresponding to the anchor at that location is selected). (d) Bounding boxes selected by MS. (e) Bounding boxes selected by MC-SCL. The red boxes in (d) and (e) are to demonstrate how cross-scale max-pooling can help to remove duplicates for the same object.

We have found that MaxpoolNMS is not working well for one-stage object detectors and second stage NMS. The reason is that, instead of selecting 50 – 300 boxes as region proposals, we need to select few boxes (5 – 10) as the final detections in such cases, which will require more sophisticated methods to design the pooling sizes and strides in order to accurately remove duplicates. Fine-tuning is not helping here, as MaxpoolNMS itself does not have learnable parameters and the way it affects training is by selecting different boxes (vs. GreedyNMS) for second stage prediction. However, different selecting strategies will not affect second stage training significantly [4]. One solution may be to borrow the idea of deformable convolutional networks [6] to design a *deformable max-pooling*, with additional modules to learn an offset field for the max-pooling kernel sizes and strides.

Table 6. Overlapping percentage of the boxes selected by GreedyNMS and MaxpoolNMS on KITTI val with Faster R-CNN+ResNet-50-V2 detector.

No. of proposals	MS	MC-AR	MC-SCL
300	49.61%	43.14%	50.84%
200	41.30%	37.67%	41.70%
100	39.01%	46.45%	39.29%
50	39.60%	50.84%	40.02%

Sensitivity of hyper parameter α MaxpoolNMS has one hyper-parameter, i.e., α in Eq. 1 and Eq. 2, which determines the pooling kernel sizes and strides and thus control the trade-off between precision and recall. A large kernel size can help to remove more duplicates (high precision) at the risk of missing detections (low recall). To investigate its effects on detection accuracy, we vary α from 0.1 to 1.0 and run evaluation on COCO val2017 with Faster R-CNN detector. Also, we vary the proposal number from 50 to 300. Results are presented in Fig. 7. It can be seen that for MS and MC-SCL, mAP is stable and can match GreedyNMS when α is in the range of [0.2, 0.8] for proposal number of 200 and 300, and in the range of [0.5, 1.0] for proposal number of 50 and 100, i.e., a small proposal number requires larger pooling size and strides. This is intuitive – in order to select a more compact set of proposals, we need to consider a larger area for comparison. MC-AR tends to be more sensitive to α, implying that proposals in neighboring

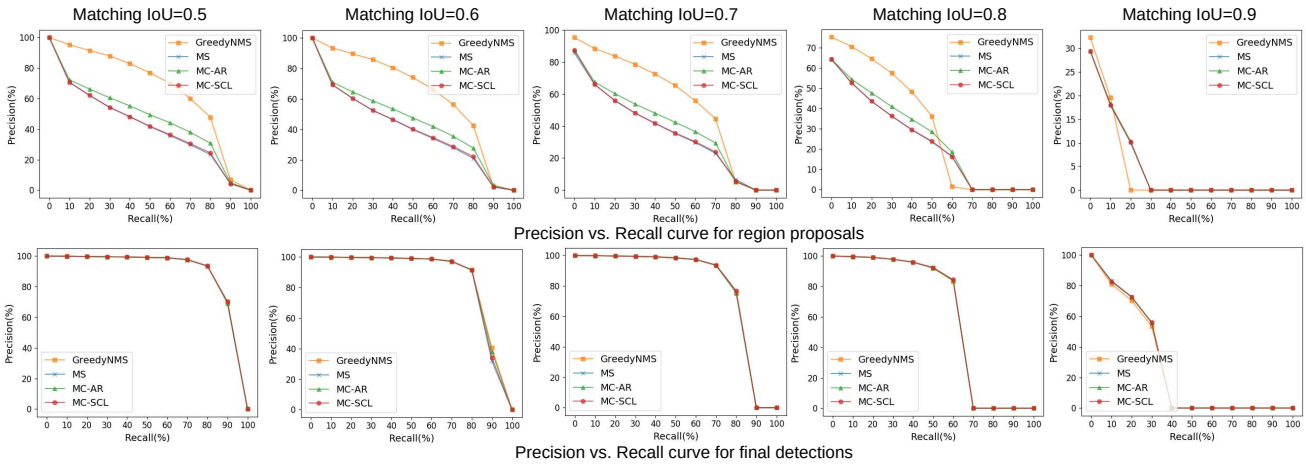
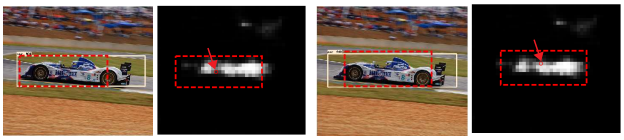


Figure 5. Precision vs. recall curves for region proposals (top row) and final detections (bottom row) at different matching IoUs. The curves are draw on KITTI val with Faster R-CNN+ResNet-50-V2 detector. The precision gap in region proposals between MaxpoolNMS and GreedyNMS is eliminated in final detections.



(a) (b)

Figure 6. Two correct detections of a car that are predicted from two different proposals for a VOC2007 test image. (a) Final detection (in yellow box) and its corresponding proposal (in dashed red box) selected by GreedyNMS. (b) Final detection and its corresponding proposal selected by MaxpoolNMS with MS scheme. Red arrows indicate the anchors that are responsible to generate the proposals. The detector is Faster R-CNN+ResNet-50-V2.

aspect ratios are more likely to contain different objects (so considering them as duplicates and removing them using MC-AR can cause mAP drop).

Influence of large anchor scales We observe that the objectness score maps corresponding to large anchor scales typically have high response in a relatively large area. This can be attributed to several factors: 1) if there are few ground truth objects that are matched to this anchor size, the corresponding score map can not train properly and thus can not make good predictions. This explains for the almost homogeneous score maps (c), (d) , (h) and (l) in Fig. 3; 2) As the effective receptive field is much smaller than the theoretical one [23], the classifier neuron is probably learning to predict an object based on part of it and thus produce a dense prediction for large objects; 3) During training, any anchor with an IoU above some threshold with a ground truth object is considered as positive sample and for large objects, anchors surrounding the object are easier to have high IoUs. These anchors have different receptive fields and thus the classifier is trained to give high response to different areas of the object. Figure 8 displays the mean score value for the 12 score maps shown in Fig. 3. As expected, score maps for small anchors are sparse (very small mean score value) while score maps for large anchors are dense (large mean score value). The presence of dense score maps can cause problem for cross-channel pooling, especially for MC-SCL, as the dense maps may suppress the correctly-selected peaks in neighboring score maps of smaller anchor scales. This is demonstrated by an example in Fig. 9. One solution is to set a threshold to separate sparse maps from dense maps and only conduct cross channel pooling on the sparse maps (with MS for dense maps). As shown in Fig. 10, with proper threshold (i.e., 0 – 0.01), MC-SCL can also match the accuracy of GreedyNMS on VOC2007 test. Note that though KITTI also have dense score maps in the last scale (i.e., 512^2), it is not affected by them as most of its objects are on scale 64^2 and 128^2 (see Fig. 11 for the object size distribution for the three datasets).

Which max-pooling scheme works the best? Based on the above discussions, we conclude that each scheme has its own strength and weakness, which are summarized in Table 7. The best scheme is thus a trade-off between accuracy, implementation complexity and sensitivity given a specific problem.

Table 7. Strength and weakness of the three pooling schemes.

	Strength	Weakness
MS	easiest to implement and can match GreedyNMS with properly selected parameters	the best obtainable accuracy is lower than MC-AR and MC-SCL
MC-AR	can further remove duplicates in neighboring aspect ratios	more sensitive to α
MC-SCL	can further remove duplicates in neighboring scales	easier to be affected by dense score maps

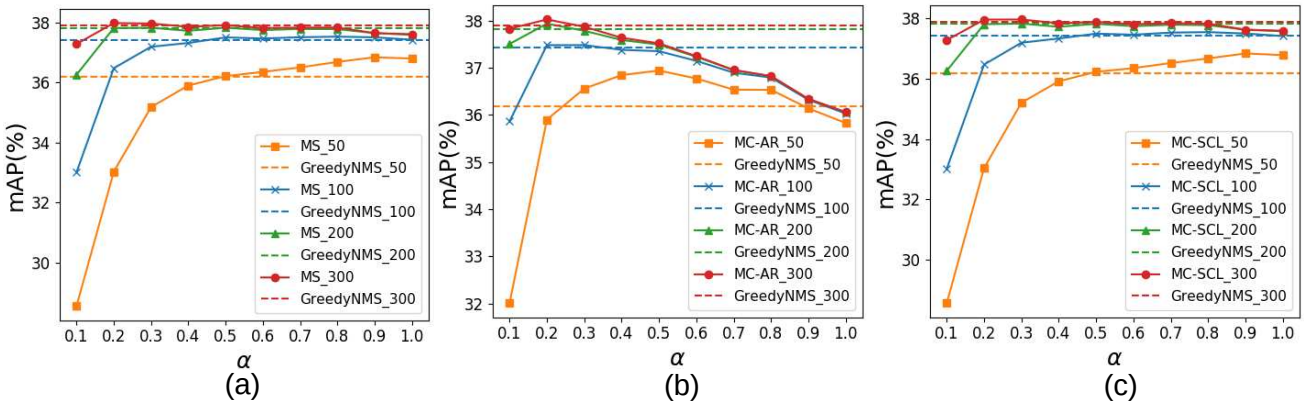


Figure 7. Sensitivity of α on COCO val2017 for different max-pooling schemes and proposal numbers. (a) MS; (b) MC-AR; (c) MC-SCL. The detector is Faster R-CNN+ResNet-101-V1.

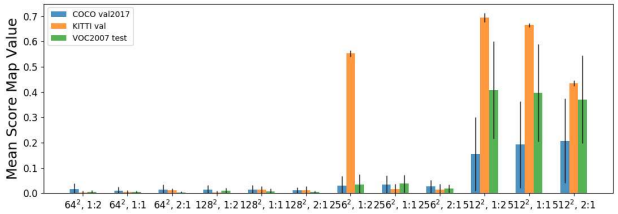


Figure 8. The mean score values for the 12 maps. Vertical black lines indicate standard deviations.

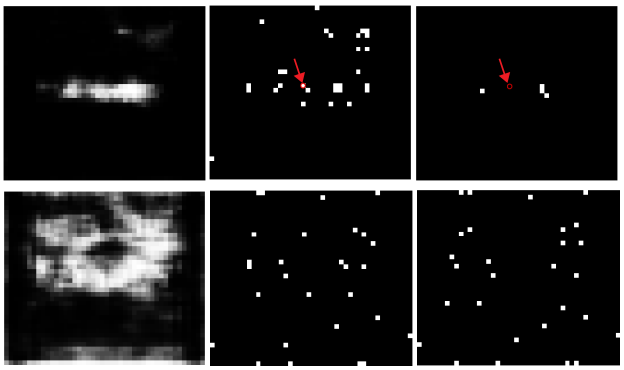


Figure 9. Dense maps may suppress the correctly-selected peaks in neighboring maps. (a) Score maps corresponding to scale 256^2 (top row) and 512^2 (bottom row) for the image shown in Fig. 6. (b) Max-pooling results of MS. (c) Max-pooling results of MC-SCL. Red arrow indicates the correct anchor that should be selected to predict the car in Fig. 6, which is incorrectly removed by cross-scale pooling in (c).

6. Conclusions

GreedyNMS can be a performance bottleneck, particularly with improved computing capabilities for parallel operations. In this paper, we introduced MaxpoolNMS, a parallelizable alternative to NMS for region proposal network. We utilized the fact that an object proposal should correspond to a peak in the objectness score maps and thus can employ multi-scale max-pooling to obtain this peak. We also exploited the fact that an object can produce multiple peaks on neighboring score maps and proposed to use multi-channel pooling across aspect ratios and scales to remove duplicates. Extensive experiments on COCO, KITTI and VOC2007 with popular two-stage object detectors, i.e., Faster R-CNN and R-FCN, demonstrated the effectiveness and robustness of our method. Benchmarking on accuracy and speed shows that MaxpoolNMS can achieve comparable accuracy as GreedyNMS with up to $20\times$ speed-up. Our approach is scalable and parallelizable, making NMS no longer a performance bottleneck in the region proposal network. This paves the way towards high-performance and real-time realization of two-stage object detectors.

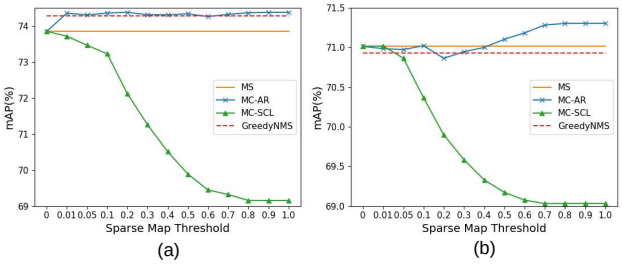


Figure 10. Effects of sparse map threshold on VOC2007 test. (a) Faster R-CNN detector; (b) R-FCN detector.

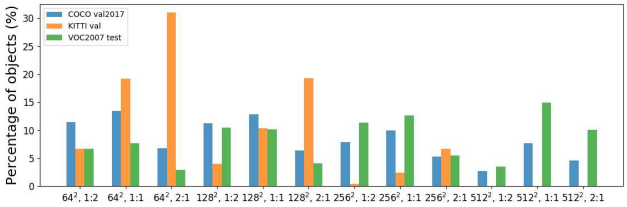


Figure 11. The distribution of object size for COCO, KITTI and VOC.

Acknowledgements

This research is supported by A*STAR under its Hardware-Software Co-optimisation for Deep Learning (Project No.A1892b0026).

References

[1] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S Davis. Improving object detection with one line of code. *arXiv preprint arXiv:1704.04503*, 2017. 2

[2] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. *arXiv preprint arXiv:1712.00726*, 2017. 2

[3] Xinlei Chen, Ross Girshick, Kaiming He, and Piotr Dollár. Tensormask: A foundation for dense object segmentation. *arXiv preprint arXiv:1903.12174*, 2019. 2

[4] Xinlei Chen and Abhinav Gupta. An implementation of faster rcnn with study for region sampling. *arXiv preprint arXiv:1702.02138*, 2017. 6

[5] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-FCN: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 1, 2

[6] Jifeng Dai, Haozhi Qi, Yuwen Xiong, Yi Li, Guodong Zhang, Han Hu, and Yichen Wei. Deformable convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 764–773, 2017. 6

[7] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005. 1, 2

[8] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010. 4

[9] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 4

[10] Ross Girshick. Fast R-CNN. *arXiv preprint arXiv:1504.08083*, 2015. 2

[11] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014. 2

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017. 2

[13] 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. 4

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pages 630–645. Springer, 2016. 2, 4

[15] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. A convnet for non-maximum suppression. In *German Conference on Pattern Recognition*, pages 192–204. Springer, 2016. 2

[16] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *arXiv preprint*, 2017. 2

[17] Han Hu, Jiayuan Gu, Zheng Zhang, Jifeng Dai, and Yichen Wei. Relation networks for object detection. *arXiv preprint arXiv:1711.11575*, 2017. 2

[18] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, 2017. 1, 2, 4

[19] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, pages 1–12. ACM, 2017. 1

[20] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017. 2

[21] 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 *European conference on computer vision*, pages 740–755. Springer, 2014. 4

[22] 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. 1, 2

[23] Wenjie Luo, Yujia Li, Raquel Urtasun, and Richard Zemel. Understanding the effective receptive field in deep convolutional neural networks. In *Advances in neural information processing systems*, pages 4898–4906, 2016. 7

[24] 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. 1, 2

[25] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 1, 2

[26] Lachlan Tychsen-Smith and Lars Petersson. Improving object localization with fitness nms and bounded iou loss. *arXiv preprint arXiv:1711.00164*, 2017. 2