MegDet: A Large Mini-batch Object Detector

Chao Peng* Tete Xiao[1*] Zeming Li[2*] Yuning Jiang Xiangyu Zhang Kai Jia Gang Yu Jian Sun
Megvii (Face++) Inc., {pengchao, jyn, zhangxiangyu, jiakai, yugang, sunjian}@megvii.com
[1]School of EECS, Peking University, jasonhsiao97@pku.edu.cn
[2]School of Software, Tsinghua University, lizm15@mails.tsinghua.edu.cn

Abstract

The development of object detection in the era of deep learning, from R-CNN, Fast R-CNN and Faster R-CNN, to more recent Mask R-CNN and RetinaNet, mainly comes from novel network, framework, or loss design. However, mini-batch size, a milestone technique for the training of deep neural networks, has not been well studied for object detection. In this paper, we present the Large Mini-batch Object Detector (MegDet), which enables training with a much larger mini-batch size up to 256. To address the optimization issues when trained with such a batch-size, we use a warm-up policy and propose a Cross-GPU Batch Normalization operation. With these techniques, we can effectively utilize multiple GPUs (up to 128 in our experiments) to significantly reduce the training time from 33 hours to 4 hours on our self-implemented large scale deep learning platform, while achieve even better performance. The MegDet is the backbone of our submission to MSCOCO 2017 Challenge, where it ranks the 1^{st} place in Ojbect Detection task.

1. Introduction

Tremendous progresses have been made on CNN-based object detection, from pioneering R-CNN [8], Fast/Faster R-CNN series [7, 28], to more recent works such as Mask RCNN [11] and RetinaNet [21]. The improvements are mainly driven by better back-bone networks [33, 35, 13, 16], new detection framework [28, 11], novel loss designs [21] and so on.

A recent trend in image classification demonstrates the possibility to use very large mini-batch size to significantly speed up the training procedure. In particular, the training of ResNet-50 [13] can be accomplished in an hour [10] or even in 14 minutes [38], by scaling the training to multiple servers with mini-batch sizes of $8,192$ or $32,768$, respectively, while suffers little drop in terms of performance. In contrast, the mini-batch size remains very small in object detection. For example, a mini-batch involves only 2 images in prevalent frameworks such as Faster R-CNN and R-FCN [4], though the mini-batch size is increased to 16 in state-of-the-art frameworks such as Mask R-CNN and RetinaNet, which is still quite small compared with it in image classification. Our goal in this paper is to explore the impact of mini-batch size in object detection and develop a practical solution for the training of a large mini-batch size object detector.

The main drawback associated with small mini-batch size is the overly lengthy training time. Since the improvements in performance are often coupled with larger datasets and architectures that require more computational resources, the training time is inevitably boosted to a level

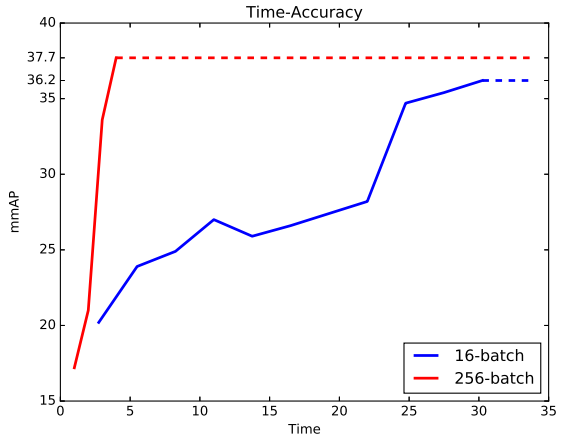

Figure 1: **MSCOCO validation mmAP v.s. the mini-batch size.** We present a Large Mini-batch Object Detector based on FPN, which allows an almost order-of-magnitude acceleration, and reaches better performance. The version that uses a mini-batch size of 256 on 128 GPUs can be trained in $\sim$4 hours, a significant shorter period compared with its counterpart with a mini-batch size of 16 on 8 GPUs.

*: Equal contribution. This work was done when Tete Xiao and Zeming Li were interns at Megvii Research.

that computer vision researchers cannot bare. For example, the training of ResNet-152 on MSCOCO dataset [22] takes 3 days, with the mini-bath size as 16 on a server with 8 NVIDIA Titian XP GPUs. The second drawback is that a small mini-batch size fails to estimate batch statistics accurately for batch normalization (BN) [17], an essential technique to train deep neural networks. Such a problem can dramatically downgrade the performance of object detection models. Last but not least, the numbers of positive and negative training samples within a relative small mini-batch are more likely to be imbalanced. This unreasonable distribution may further lead to sub-optimal performance. Details are presented in Section 3.1.

Nevertheless, the effort to increase the mini-batch size of object detection is not trivial. The first dilemma is that, according to "the equivalent learning rate rule" [10, 18], which indicates that when the mini-batch size is multiplied by k, the learning rate shall also be multiplied by k, a large mini-batch size requires a large learning rate to reduce the training procedure and match the performance of the model with a small mini-batch size. However, such a large learning rate could lead to inferior results, even the failure of convergence. The second dilemma is the physical limitation of GPU memories, for that the input scale for object detection is normally set as 800×800 to detect small objects, which is much larger than it for image classification. While scaling networks via distributed synchronous SGD is now commonplace for image classification, there still lack of works on such technique for object detection.

To tackle the above-mentioned dilemmas, we propose a solution that enables the training of a large mini-batch object detection. First, we present a new explanation of linear scaling rule and adopt the "warm-up" learning rate policy in [10] to gradually increase the learning rate at the very early stage of training. This increases the chance of convergence. Second, to further address the performance and convergence issues, we propose Cross-GPU Batch Normalization (CGBN), to exploit the fact that we have a large number of samples in a mini-batch. CGBN not only improves the performance but also makes the training much more stable. More importantly, we have observed that we can keep the final accuracy while increasing the mini-batch size with the help of the CGBN.

Our MegDet on MSCOCO with a ResNet-50 as backbone can be trained in $\sim$4 hours using 128 GPUs, compared with the 33-hour training time of its counterpart with regular mini-batch size. The curves of performance v.s. training time are shown in Figure 1. Importantly, not only does our method consume significantly shorter time, but also reaches even better performance. This means that we can enjoy the rapidly increased computational power from industry by speeding up the innovation cycle by nearly an order-of-magnitude. Based on MegDet, we achieve the 1^{st} place of MSCOCO 2017 Detection Challenge. Our contributions can be summarized as follows:

- We provide a new interpretation of linear scaling rule, in the context of object detection, based on an assumption of maintaining equivalent loss variance.
- We are the first to perform BN in the object detection framework. We demonstrated that our Cross-GPU BN not only benefits the accuracy, but also makes the training easy to converge, especially for the large mini-batch size.
- We are the first to finish the COCO training (based on ResNet-50) in 4 hours, using 128 GPUs, and achieving improved accuracy.
- Our MegDet leads to the winning of COCO 2017 Detection Challenge.

2. Related Work

Object detection. With the development of convolutional neural networks (CNNs), great progress has been made in object detection. State-of-the-art object detectors can be roughly divided into two categories: two-stage detectors and one-stage detectors.

Since R-CNN [8] is proposed, two-stage framework has become the *de facto* paradigm for object detection. With the help of the Selective Search algorithm [36], the first stage generates a relatively small set of proposals containing the possible locations of all objects in an image, followed by the second stage which classifies each region into foreground or background. In the R-CNN framework, the second stage is a CNN that takes each region after resize as input then processes separately. To overcome the tremendous computational bottleneck that every proposal needs to be fed as input of CNNs, Spatial Pyramid Pooling (SPP) [12] and RoI Pooling [7] operators are proposed, so that a single image can be forwarded only once with a classification head that warps and classifies each proposal on top of a feature map. In Faster R-CNN [28] the traditional proposal algorithm is replaced by the Region Proposal Network (RPN) which integrates region proposal with classification to form a single end-to-end network. More recent two-stage frameworks involve R-FCN [4], in which a position-sensitive pooling is proposed to get rid of the heavy classification head; FPN [20], in which a generic feature extractor that exploits multi-level feature representations in an inherent and pyramidal hierarchy is proposed; and Mask R-CNN [11], in which a novel feature pooling technique termed RoI Align is proposed, and Faster R-CNN is extended to predict object masks in parallel with bounding boxes.

For one-stage detector, OverFeat [30] is one of the first one-stage detectors built on CNNs. More recent works, such as YOLO [26, 27] and SSD [23], demonstrate great potential in the field of object detection. YOLO uses a fully

Epoch	Batch Size	Ratio(%)
1	16	5.58
	256	**9.82**
6	16	11.77
	256	**16.11**
12	16	16.59
	256	**16.91**

Table 1: The ratio of positive and negative samples in the training at epoch 1, 6, 12. A larger batch size makes the ratio more balanced, especially at the early stage.

convolutional network (FCN) to obtain classification and regression predictions simultaneously. SSD [23] presents a architecture that detects objects with different scales on different feature maps. These frameworks run fast, still, they trail two-stage detectors in terms of performance by a large margin. Recently, RetinaNet is proposed in [21] with a novel focal loss which significantly narrows the gap between two-stage detectors and one-stage ones.

Training neural networks with large mini-batch size. Coupled with the prosperity of deep learning, training deep neural networks with distributed SGD on various servers is drawing more attention. With no doubt, such large-scale distributed training can result in a nontrivial growth of mini-batch sizes, which makes it hard to maintain the original performance. In [10], Goyal *et al*. provide empirical insights that address the problems when trained with a large mini-batch size. Specifically, they are able to train a ResNet-50 network on ImageNet dataset [5] with the mini-batch size of $8,192$ on 256 NVIDIA Tesla P100 GPU, and reduce the training time to less one hour, while demonstrate no loss of accuracy. You *et al*. [38] further increase the mini-batch size to $32,768$, and dramatically reduce the training time of a ResNet-50 to 14 minutes on $2,048$ KNLs. Hoffer *et al*. [14] investigate the generalization gap between large batch and small batch, and provide insights to explain and eliminate the gap. However, in spite of the thriving discussion of training neural networks with large mini-batch size on *image classification*, so far there still lacks study for it on *object detection*.

3. Approach

In this section, we present our Large Mini-Batch Detector (MegDet). We start by discussing the problems when training a object detector using small mini-batch; then provide a new interpretation for the "equivalent learning rate rule"; and finally describe several techniques that enable our approach to be trained with much larger mini-batch size with even better performance.

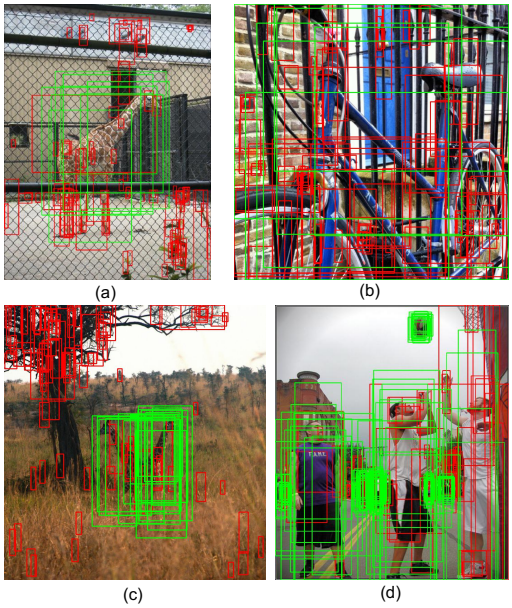

Figure 2: Examples of positive and negative proposals. (a-b) Two examples with imbalanced ratio. (c-d) Two examples with moderate balanced ratio. Note that we sample the negative proposals for clearer visualization.

3.1. Problems with Small Mini-Batch Size

Our motivation behind that we are interested in training object detectors with large mini-batch comes from the following several observations. First, to enjoy the developments of large neural network architectures and datasets is not free, *i.e*., it takes significantly amount of time to train a single network. As shown in Figure 1, it takes more than 30 hours to train a detector based on ResNet-50 with mini-batch size of 16. Even worse, with mini-batch size of 2 it will take more than a week to complete training the same model. Second, with no exception, the BN layers are all fixed in previous works. The reason is that small mini-batch size leads to inaccurate and unstable BN statistics. However, using the BN parameters calculated on ImageNet can lead to sub-optimial performance, and limit the generation ability of models. Third, the numbers of positive and negative samples (foreground/background) during training can be extremely unbalanced for some images. When the mini-batch size is small, the distribution of these samples may be noisy for optimization. Table 1 compares the ratio of positive and negative samples between detectors with small and large mini-batch size at different training epochs. It demonstrates that the ratio is more imbalanced for the detector with small mini-batch size, especially during the early stage of training. Visualization samples are provided in Figure 2.

3.2. Learning Rate for Large Mini-Batch

The learning rate policy is essential for training a network with large mini-batch, since we would like to shorten the training time while maintaining accuracy. Therefore, we first review the loss functions for object detection networks:

$$\begin{aligned} L(x, w) &= \frac{1}{N}\sum_{i=1}^{N} l(x_i, w) + \frac{\lambda}{2}||w||_2^2 \\ &= l(x, w) + l(w), \end{aligned} \tag{1}$$

where N is the mini-batch size, $l(x, w)$ is the task specific loss and $l(w)$ is the regularization loss. For Faster R-CNN [28] framework and its variants [4, 20, 11], $l(x_i, w)$ consists of RPN prediction loss, RPN bounding-box regression loss, R-CNN prediction loss, and R-CNN bounding box regression loss.

Based on the mini-batch stochastic gradient descent (SGD), a system needs to compute the gradients with respect to weight w, and update them after every iteration. When the size of mini-batch changes, such as $\hat{N} \leftarrow k \cdot N$, we expect that the learning rate r is adapted to maintain the efficiency of training. Previous works [18, 10, 38] use *Linear Scaling Rule*, which changes the new learning rate to $\hat{r} \leftarrow k \cdot r$. Since one iteration with a large mini-batch $\hat{N}$ should match k consistent iterations with a small mini-batch N, the learning rate r shall also be multiplied by the same ratio k to counteract the scaling factor N in the loss function. This is based on the *gradient equivalence* assumption [10] during the SGD process. This rule has proven to be effective for image classification, and we empirically find that this is still applicable for object detection.

Nevertheless, for image classification, each image has only one label and $l(x, w)$ is a cross-entropy loss. For object detection, each image has different number of bounding box annotations, resulting in various ground-truth distributions. Therefore, the assumption of gradient equivalence in [10] may be less likely to hold in object detection, even when the gradient is stable. Here, we provide an alternative interpretation in the following.

Variance equivalence. Different from the gradient equivalence assumption, we assume that the variance of gradient remain the same during k steps. Given the mini-batch size N, if the gradient of each sample $\nabla l(x_i, w)$ obeying i.i.d., the variance of gradient on $l(x, w)$ is:

$$\begin{aligned} \text{Var}(\nabla l(x, w_t)) &= \frac{1}{N^2}\sum_{i=1}^{N} \text{Var}(\frac{\partial l(x_i, w_t)}{\partial w_t}) \\ &= \frac{1}{N^2} \times \left(N \cdot \sigma_l^2\right) \\ &= \frac{1}{N}\sigma_l^2. \end{aligned} \tag{2}$$

Similarly, for the large mini-batch $\hat{N} = k \cdot N$, we can get the following expression:

$$\text{Var}(\nabla l_{\hat{N}}(x, w_t)) = \frac{1}{kN}\sigma_l^2. \tag{3}$$

Instead of expecting equivalence on weight update, here we want to *maintain the variance of one update in large mini-batch $\hat{N}$ equal to k accumulative steps in small mini-batch N*. To achieve this, we have:

$$\begin{aligned} \text{Var}(r \cdot \sum_{t=1}^{k}(\nabla l_N^t(x, w))) &= r^2 \cdot k \cdot \text{Var}(\nabla l_N(x, w)) \\ &\approx \hat{r}^2 \text{Var}(\nabla l_{\hat{N}}(x, w)) \end{aligned} \tag{4}$$

With Equation (2) and (3), the above equality holds if and only if $\hat{r} = k \cdot r$, which gives the same linear scaling rule for $\hat{r}$.

Although the final scaling rule is the same, our *variance equivalence* assumption Equation (4) is weaker because we just expect that the large mini-batch training can maintain equivalent statistics on the gradients. We hope the variance analysis here can shed light on deeper understanding of learning rate in wider applications.

Warm-up strategy. As discussed in [10], the linear scaling rule may not be applicable at the initial stage of the training because the weights of networks change dramatically. To address this problem, we adopt *Linear Gradual Warm-up* technique. That is, we begin with a small learning rate (*e.g.*, r) that allows the training to be activated. Then we increase the learning rate linearly after every iteration until it reaches $\hat{r}$.

The linear gradual warm-up strategy helps the convergence of training. However, as demonstrated in Section 4, we empirically find that such a strategy is still insufficient for much larger mini-batch size, *e.g.*, when it exceeds 128. Next, we propose the Cross-GPU Batch Normalization, which is the last stepping-stone towards training object detectors with large mini-batch size.

3.3. Cross-GPU Batch Normalization

Batch Normalization [17] is an important technique that enables the training of deep convolutional neural networks. Without batch normalization, training such a deep network will consume much more time or even fail to converge. However, previous object detection frameworks, such as FPN [20] and Mask R-CNN [11], are initializes with ImageNet pre-trained models, after which the batch normalization layers are fixed during the whole fine-tuning procedure. In this work, we make an attempt to perform batch normalization for object detection.

It is worth noting that the input image of classification network is 224×224 or 299×299, so a single NVIDIA

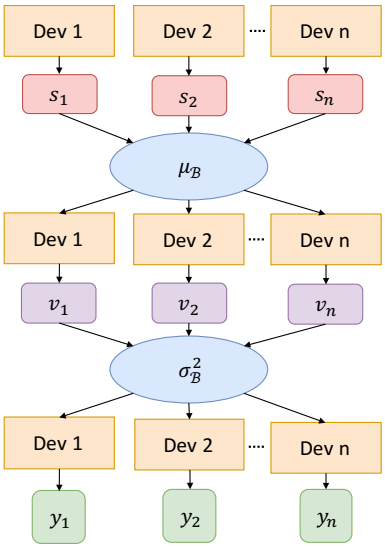

Figure 3: Implementation of Cross-GPU Batch Normalization. The gray ellipse depicts the synchronization over devices, while the rounded boxes represent paralleled computation of multiple devices.

TITAN Xp GPU with 12 Gigabytes memory is enough for 32 or more images. In this way, batch normalization can be computed on each device alone. However, for object detection, a detector needs to detect objects of various scales, thus higher resolution images are needed as input. In [20], input of size 800×800 is used, which limits the number of possible samples on one device. Thus, we have to perform batch normalization crossing multiple GPUs to compute sufficient statistics from more training samples.

To implement batch normalization across multiple GPUs, we need to compute the aggregated mean/variance statistics over all devices. Most existing deep learning frameworks utilize the batch normalization implementation in cuDNN [3] that only provides a high-level API without permitting modification of internal statistics. Therefore we need to implement batch normalization in terms of preliminary mathematical expressions and use an "All-Reduce" operation to aggregate the statistics. These fine-grained expressions usually cause significant run-time overhead and the All-Reduce operation is missing in most frameworks.

Our implementation of Cross-GPU Batch Normalization is sketched in Figure 3. Given n GPU devices in total, sum value s_k is first computed based on the training examples assigned to the device k. By averaging the sum values from all devices, we obtain the mean value $\mu_{\mathcal{B}}$ for current mini-batch. This step requires an All-Reduce operation. Then we calculate the variance for each device and get $\sigma^2_{\mathcal{B}}$. After broadcasting $\sigma^2_{\mathcal{B}}$ to each device, we can perform the standard normalization by $y = \gamma \frac{x-\mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}} + \beta$. Algorithm 1 gives the detailed flow. In our implementation, we use NVIDIA Collective Communication Library (NCCL) to efficiently perform All-Reduce operation for receiving and broadcasting.

Input: Values of input x on multiple devices in a minibatch: $\mathcal{B} = \bigcup_{i=1}^{n} \mathcal{B}_i$, $\mathcal{B}_i = \{x_{i_1 \ldots i_n}\}$
BN parameters: γ, β
Output: $y = \text{CGBN}(x)$

for $i = 1, \ldots, n$ **do**
  compute the device sum s_i over set B_i
end for
reduce the set $s_{1,\ldots,n}$ to minibatch mean $\mu_{\mathcal{B}}$
broadcast $\mu_{\mathcal{B}}$ to each device
for $i = 1, \ldots, n$ **do**
  compute the device variance sum v_i over set B_i
end for
reduce the set $v_{1,\ldots,n}$ to minibatch variance $\sigma^2_{\mathcal{B}}$
broadcast $\sigma^2_{\mathcal{B}}$ to each device
compute the output: $y = \gamma \frac{x-\mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}}+\epsilon}} + \beta$ over devices

Algorithm 1: Cross-GPU Batch Normalization over a mini-batch $\mathcal{B}$.

Note that we only perform batch normalization across GPUs on the same machine. So, we can calculate batch normalization statistics on 16 images if each GPU can hold 2 images. To perform batch normalization on 32 or 64 images, we apply *sub-linear memory* [2] to save the GPU memory consumption by slightly compromising the training speed.

In next section, our experimental results will demonstrate the positive impacts of CGBN on both convergence and performance.

4. Experiments

We conduct experiments on the challenging COCO Detection Dataset [22], which is split into train, validation, and test, containing 80 categories and over $250,000$ images. We use ResNet-50 [13] pre-trained on ImageNet [5] as the backbone network and Feature Pyramid Network (FPN) [20] as the detection framework. We train the detectors over $118,000$ training images and evaluate on $5,000$ validation images. We use the SGD optimizer with a momentum as 0.9 and a weight decay as 0.0001. The base learning rate for batch size 16 is 0.02. For other mini-batch size settings, the linear scaling rule described in Section 3.2 is applied. For larger mini-batch size, we use the sublinear memory technique [2] and distributed training to remedy the GPU memory constraints.

We use two training policies: 1) *normal*, where the learn-

Mini-Batch size	mmAP	Time (h)
16	36.2	33.2
32	**36.4**	**15.1**
64	failed	–
64 (half learning rate)	36.0	7.5
128 (half learning rate)	failed	–

Table 2: Comparisons of different mini-batch sizes, without BN.

ing rate is decreased at epoch 8 and 10 by multiplying 0.1, and the training ends at epoch 11; 2) *long*, where the learning rate is decreased at epoch 11 and 14 by multiplying 0.1, and halved at epoch 17, and the training ends at epoch 18. We use the normal policy unless specified.

4.1. Large Mini-batch Size w/o BN

We start our study on different mini-batch sizes without batch normalization. More specifically, the mini-batch sizes are set as 16, 32, 64, and 128. For mini-batch sizes of 32, we observe that the training fails to converge with a certain possibility, even when we use the warm-up strategy. For mini-batch size of 64, the training fails every time even with the warm-up strategy. In order to make the training converge, we have to lower the learning rate by half. For mini-batch size of 128, the training fails with both warm-up strategy and a learning rate which is halved. The results of these experiments on COCO validation set are shown in Table 2. We can see that: 1) Training with thge mini-batch size of 32 achieves almost linear acceleration without loss of accuracy, compared with the baseline using the mini-batch size of 16; 2) Training with a half learning rate and a mini-batch size of 64 results in a drop of performance; 3) Training is harder or impossible when the mini-batch size and the learning rate are larger, even with the warm-up strategy.

4.2. Large Mini-batch Size w/ CGBN

We also conduct experiments in which we train the models with batch normalization. Our first key finding is that *all experiments are more easily to converge when we use warm-up strategy and CGBN*, no matter of the mini-batch size. This is remarkable because we do not have to worry about the possible loss of accuracy caused by using a smaller learning rate.

The main results are summarized in Table 3. We have the following observations.

First, with the growth of mini-batch size, the accuracy remains the same, which is consistently better that the baseline (16-base). Meanwhile, training with a larger mini-batch size always leads to a shorter training procedure. For instance, the experiment with a mini-batch size of 256 on 128 GPUs can be trained in only 4.1 hours, which is a $8\times$ acceleration compared with the 33.2 hours baseline.

Batch size	BN size	# of GPUs	mmAP	Time(h)
16-base	0	8	36.2	33.2
2	2	2	31.5	131.2
4	4	4	34.9	91.4
8	8	8	35.9	71.5
16	2	8	31.0	45.6
16	16	8	37.0	39.5
32	32	8	37.3	45.5
64	64	8	35.3	40.9
64	32	16	37.1	19.6
64	16	32	37.1	11.2
128	32	32	37.1	11.3
128	16	64	37.0	6.5
256	32	64	37.1	7.2
256	16	128	37.1	**4.1**
16 (long)	16	8	37.7	65.2
32 (long)	32	8	**37.8**	60.3
64 (long)	32	16	37.6	30.1
128 (long)	32	32	37.6	15.8
256 (long)	32	64	37.7	9.4
256 (long)	16	128	37.7	5.4

Table 3: Comparisons of training with different mini-batch sizes, BN sizes (the number of images used for calculating statistics), GPU numbers, and training policies. "long" means that we apply the long training policy. When the BN size ≥ 32, the sublinear memory is applied and thus slightly reduces training speed. Overall, the large mini-batch size with BN not only speeds up the training, but also improves the accuracy.

Second, the best batch normalization scale, *i.e.*, the number of images for estimating batch normalization statistics, is 32. With too few training samples, *e.g.*, less or equal to 8 samples, the batch normalization statistics are highly inaccurate, resulting in a drop in performance. However, when we increase the batch normalization scale to 64, the performance also drops. We conjure that this is due to the mismatch between image classification and object detection tasks.

Third, we find that when a model is trained longer, the performance is slightly boosted. The results are shown in the bottom of Table 3. For example, "32 (long)" is better that its counterpart (37.8 v.s. 37.3). When the mini-batch size is larger than 16, the final results are almost identical, which is a sign of good convergence.

We also illustrate epoch-by-epoch mmAP curves of the model 16 (long) and 256 (long) in Figure 4. 256 (long) is worse at early epochs but catches up 16 (long) at the last

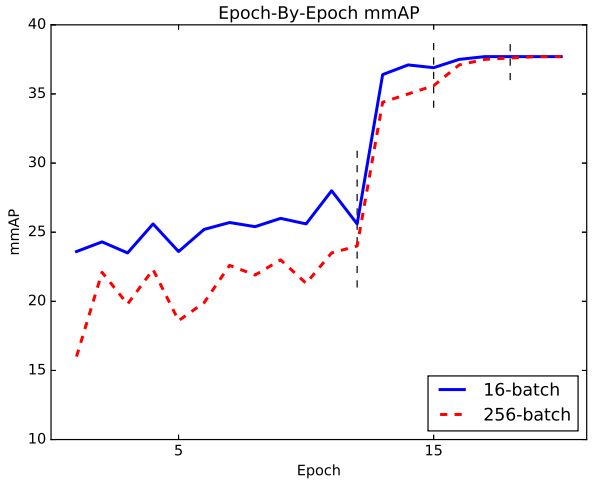

Figure 4: Validation accuracy of 16 (long) and 256 (long) detectors, using the long training policy. The BN sizes are the same in two detectors. The vertical dashed lines indicate the moments of learning rate decay.

stage (after second learning rate decay). This observation is different from those in image classification [10, 38], where both the accuracy curves and convergent scores are very close between different batch size settings. We leave the understanding of this phenomenon as the future work.

name	mmAP	mmAR
DANet	45.7	62.7
Trimps-Soushen+QINIU	48.0	65.4
bharat_umd	48.1	64.8
FAIR Mask R-CNN [11]	50.3	66.1
MSRA	50.4	69.0
UCenter	51.0	67.9
MegDet (Ensemble)	**52.5**	**69.0**

Table 4: Result of (enhanced) MegDet on test-dev of COCO dataset.

5. Concluding Remarks

We have presented a large mini-batch size detector, which achieved better accuracy in much shorter time. This is remarkable because our research cycle has been greatly accelerated. As a result, we have obtained 1st place of COCO 2017 detection challenge. The details are in Appendix.

Appendix

Based on our MegDet, we integrate the techniques including OHEM [32], atrous convolution [39, 1], stronger base models [37, 15], large kernel [25], segmentation supervision [24, 31], diverse network structure [9, 29, 34], contextual modules [19, 6], ROIAlign [11] and multi-scale training and testing for COCO 2017 Object Detection Challenge. We obtained **50.5** mmAP on validation set, and **50.6** mmAP on the test-dev. The ensemble of four detectors finally achieved **52.5**. Table 4 summarizes the entries from the leaderboard of COCO 2017 Challenge. Figure 5 gives some exemplar results.

References

[1] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016. 7

[2] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016. 5

[3] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 5

[4] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016. 1, 2, 4

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. 3, 5

[6] S. Gidaris and N. Komodakis. Object detection via a multi-region and semantic segmentation-aware cnn model. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1134–1142, 2015. 7

[7] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 1, 2

[8] R. Girshick, J. Donahue, T. Darrell, and J. 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. 1, 2

[9] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *International Conference on Machine Learning*, pages 1319–1327, 2013. 7

[10] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. 1, 2, 3, 4, 7

[11] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2, 4, 7

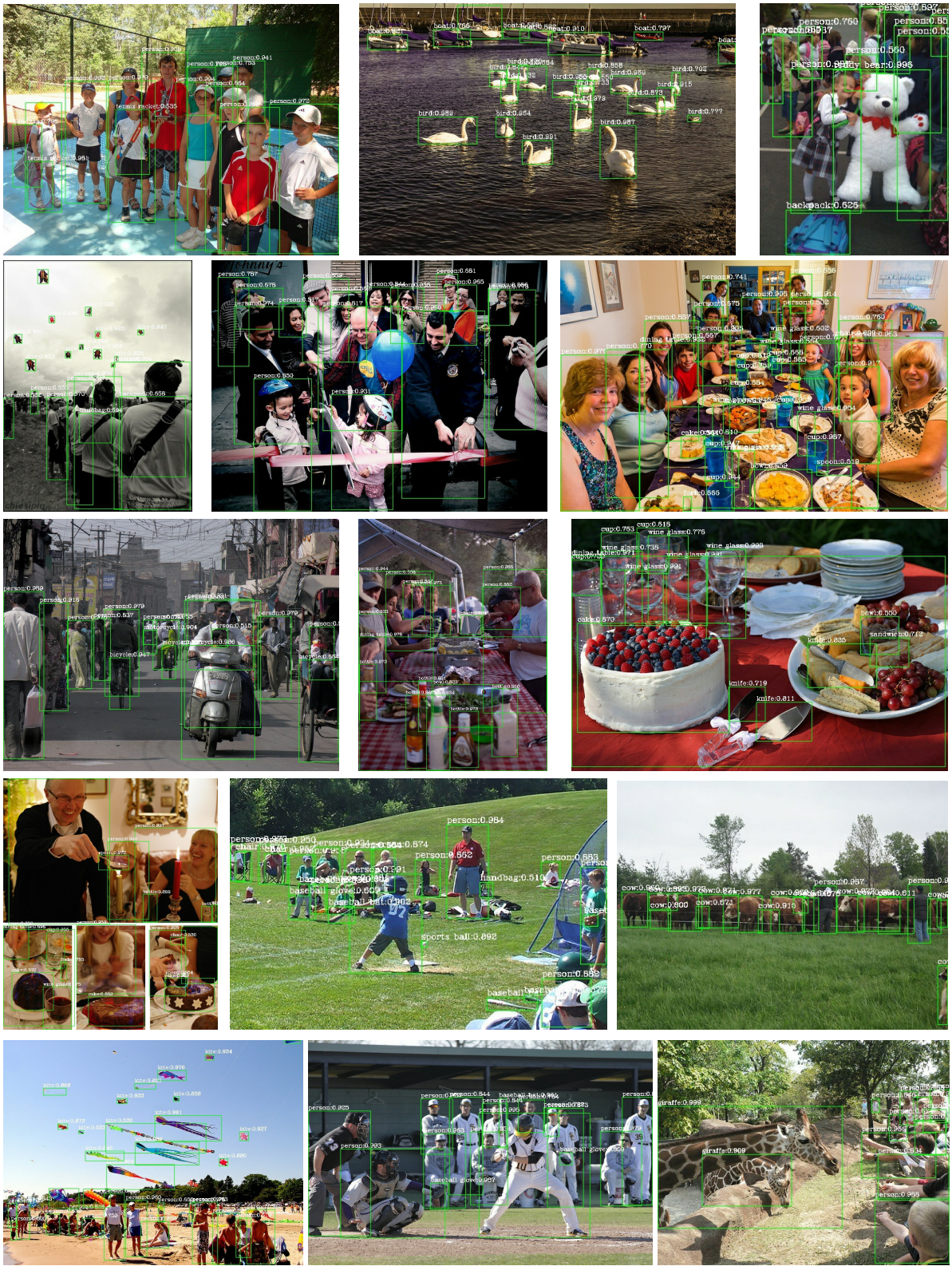

Figure 5: Illustrative examples for our MegDet on COCO dataset.

[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision*, pages 346–361. Springer, 2014. 2

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 5

[14] E. Hoffer, I. Hubara, and D. Soudry. Train longer, generalize better: closing the generalization gap in large batch training of neural networks. In *Advances in Neural Information Processing Systems*, pages 1729–1739, 2017. 3

[15] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 7

[16] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017. 1

[17] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015. 2, 4

[18] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014. 2, 4

[19] J. Li, Y. Wei, X. Liang, J. Dong, T. Xu, J. Feng, and S. Yan. Attentive contexts for object detection. *IEEE Transactions on Multimedia*, 19(5):944–954, 2017. 7

[20] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 2, 4, 5

[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 3

[22] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. 2, 5

[23] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016. 2, 3

[24] J. Mao, T. Xiao, Y. Jiang, and Z. Cao. What can help pedestrian detection? In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[25] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel matters – improve semantic segmentation by global convolutional network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[26] J. Redmon, S. Divvala, R. Girshick, and A. 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

[27] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint arXiv:1612.08242*, 2016. 2

[28] S. Ren, K. He, R. Girshick, and J. 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, 4

[29] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. *IEEE transactions on pattern analysis and machine intelligence*, 39(7):1476–1481, 2017. 7

[30] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 2

[31] A. Shrivastava and A. Gupta. Contextual priming and feedback for faster r-cnn. In *European Conference on Computer Vision*, pages 330–348. Springer, 2016. 7

[32] A. Shrivastava, A. Gupta, and R. Girshick. Training region-based object detectors with online hard example mining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 761–769, 2016. 7

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015. 1

[34] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, pages 4278–4284, 2017. 7

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 1

[36] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013. 2

[37] S. Xie, R. Girshick, P. Dollar, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 7

[38] Y. You, Z. Zhang, C.-J. Hsieh, J. Demmel, and K. Keutzer. Imagenet training in minutes. *arXiv preprint arXiv:1709.05011*, 2017. 1, 3, 4, 7

[39] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 7