ThunderNet: Towards Real-time Generic Object Detection

Zheng Qin*†[1], Zeming Li*[2], Zhaoning Zhang[1], Yiping Bao[2], Gang Yu[2], Yuxing Peng[1], Jian Sun[2]
[1] National University of Defense Technology [2] Megvii Inc. (Face++)
{qinzheng12, zhangzhaoning, pengyuxing}@nudt.edu.cn {lizeming, baoyiping, yugang, sunjian}@megvii.com

Abstract

Real-time generic object detection on mobile platforms is a crucial but challenging computer vision task. However, previous CNN-based detectors suffer from enormous computational cost, which hinders them from real-time inference in computation-constrained scenarios. In this paper, we investigate the effectiveness of two-stage detectors in real-time generic detection and propose a lightweight two-stage detector named ThunderNet. In the backbone part, we analyze the drawbacks in previous lightweight backbones and present a lightweight backbone designed for object detection. In the detection part, we exploit an extremely efficient RPN and detection head design. To generate more discriminative feature representation, we design two efficient architecture blocks, Context Enhancement Module and Spatial Attention Module. At last, we investigate the balance between the input resolution, the backbone, and the detection head. Compared with lightweight one-stage detectors, ThunderNet achieves superior performance with only 40% of the computational cost on PASCAL VOC and COCO benchmarks. Without bells and whistles, our model runs at 24.1 fps on an ARM-based device. To the best of our knowledge, this is the first real-time detector reported on ARM platforms. Code will be released for paper reproduction.

1. Introduction

Real-time generic object detection on mobile devices is a crucial but challenging task in computer vision. Compared with server-class GPUs, mobile devices are computation-constrained and raise more strict restrictions on the computational cost of detectors. However, modern CNN-based detectors are resource-hungry and require massive computation to achieve ideal detection accuracy, which hinders them from real-time inference in mobile scenarios.

From the perspective of network structure, CNN-based detectors can be divided into the *backbone part* which extracts features for the image and the *detection part* which detects object instances in the image. In the backbone part, state-of-the-art detectors are inclined to exploit huge classification networks (e.g., ResNet-101 [10, 4, 16, 17]) and large input images (e.g., 800×1200 pixels), which requires massive computational cost. Recent progress in lightweight image classification networks [3, 33, 20, 11, 28] has facilitated real-time object detection [11, 28, 14, 20] on GPU. However, there are several differences between image classification and object detection, e.g., object detection needs large receptive field and low-level features to improve the localization ability, which is less crucial for image classification. The gap between the two tasks restricts the performance of these backbones on object detection and obstructs further compression without harming detection accuracy.

In the detection part, CNN-based detectors can be categorized into *two-stage detectors* [27, 4, 16, 14] and *one-stage detectors* [24, 19, 25, 17]. For two-stage detectors, the detection part usually consists of Region Proposal Network (RPN) [27] and the detection head (including RoI warping and R-CNN subnet). RPN first generates RoIs, and then the

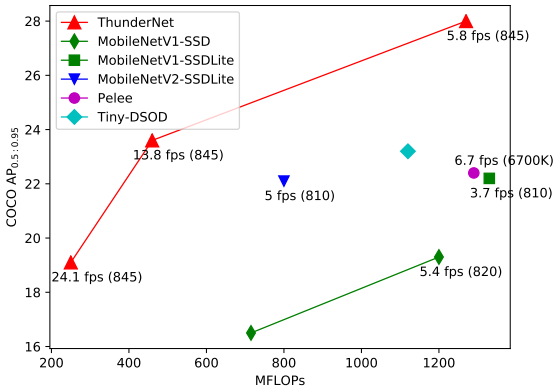

Figure 1. Comparison of ThunderNet and previous lightweight detectors on COCO test-dev[1]. ThunderNet achieves improvements in both accuracy and efficiency.

*Equal contribution.
†This work was done when Zheng Qin was an intern at Megvii Inc.

[1]Speed is evaluated with a single thread on CPU: MobileNet-SSD on Snapdragon 820, MobileNet/MobileNetV2-SSDLite on Snapdragon 810, Pelee on Intel i7-6700K (4.0 GHz), and ThunderNet on Snapdragon 845.

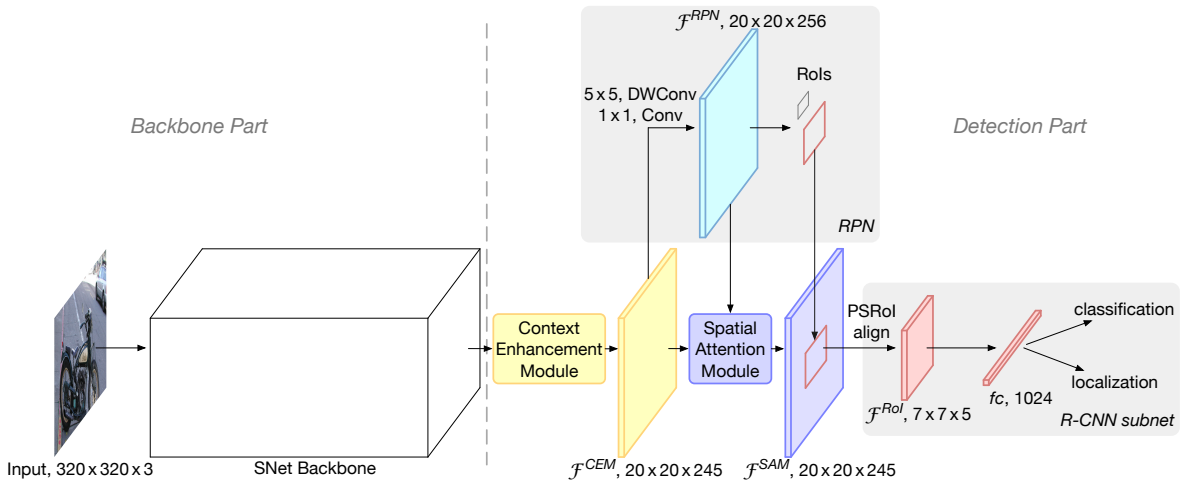

Figure 2. The overall architecture of ThunderNet. ThunderNet uses the input resolution of 320×320 pixels. SNet backbone is based on ShuffleNetV2 and specifically designed for object detection. In the detection part, RPN is compressed, and R-CNN subnet uses a 1024-d *fc* layer for better efficiency. Context Enhancement Module leverages semantic and context information from multiple scales. Spatial Attention Module introduces the information from RPN to refine the feature distribution.

RoIs are further refined through the detection head. State-of-the-art two-stage detectors tend to utilize a heavy detection part (e.g., over 10 GFLOPs [27, 10, 4, 16, 2]) for better accuracy, but it is too expensive for mobile devices. Light-Head R-CNN [14] adopts a lightweight detection head and achieves real-time detection on GPU. However, when coupled with a small backbone, Light-Head R-CNN still spends more computation on the detection part than the backbone, which leads to a mismatch between a weak backbone and a strong detection part. This imbalance not only induces great redundancy but makes the network prone to overfitting.

On the other hand, one-stage detectors directly predict bounding boxes and class probabilities. The detection part of this category is composed of the additional layers to generate predictions, which usually involves little computation. For this reason, one-stage detectors are widely regarded as the key to real-time detection. However, as one-stage detectors do not conduct RoI-wise feature extraction and recognition, their results are coarser than two-stage detectors. The problem is aggravated for lightweight detectors. Prior lightweight one-stage detectors [11, 28, 31, 13] do not obtain an ideal accuracy/speed trade-off: there is a huge accuracy gap between them and the large detectors [19, 25], while they fail to achieve real-time detection on mobile devices. It inspires us to rethink: *can two-stage detectors surpass one-stage detectors in real-time detection?*

In this paper, we propose a lightweight two-stage generic object detector named *ThunderNet*. The design of ThunderNet aims at the computationally expensive structures in state-of-the-art two-stage detectors. In the backbone part, we investigate the drawbacks in previous lightweight backbones, and present a lightweight backbone named *SNet* designed for object detection. In the detection part, we follow the detection head design in Light-Head R-CNN, and further compress RPN and R-CNN subnet. To eliminate the performance degradation induced by small backbones and small feature maps, we design two efficient architecture blocks, *Context Enhancement Module* (CEM) and *Spatial Attention Module* (SAM). CEM combines the feature maps from multiple scales to leverage local and global context information, while SAM uses the information learned in RPN to refine the feature distribution in RoI warping. At last, we investigate the balance between the input resolution, the backbone, and the detection head. Fig. 2 illustrates the overall architecture of ThunderNet.

ThunderNet surpasses prior lightweight one-stage detectors with significantly less computational cost on PASCAL VOC [5] and COCO [18] benchmarks. ThunderNet outperforms Tiny-DSOD [13] with only 42% of the computational cost and obtains gains of 6.5 mAP on VOC and 4.8 AP on COCO under similar complexity. Without bells and whistles, ThunderNet runs in *real time* on ARM (24.1 fps) and x86 (47.3 fps) with MobileNet-SSD level accuracy. To the best of our knowledge, this is the *first* real-time detector and the *fastest* single-thread speed reported on ARM platforms. *These results have demonstrated the effectiveness of two-stage detectors in real-time object detection.*

2. Related Work

CNN-based object detectors. CNN-based object detectors are commonly classified into two-stage detectors and one-stage detectors. In two-stage detectors, R-CNN [8] is among the earliest CNN-based detection systems. Since then, progressive improvements [9, 7] are proposed for better accuracy and efficiency. Faster R-CNN [27] proposes

Region Proposal Network (RPN) to generate regions proposals instead of pre-handled proposals. R-FCN [4] designs a fully convolutional architecture which shares computation on the entire image. On the other hand, one-stage detectors such as SSD [19] and YOLO [24, 25, 26] achieve real-time inference on GPU with very competitive accuracy. RetinaNet [17] proposes focal loss to address the foreground-background class imbalance and achieves significant accuracy improvements. In this work, we present a two-stage detector which focuses on efficiency.

Real-time generic object detection. Real-time object detection is another important problem for CNN-based detectors. Commonly, one-stage detectors are regarded as the key to real-time detection. For instance, YOLO [24, 25, 26] and SSD [19] run in real time on GPU. When coupled with small backbone networks, lightweight one-stage detectors, such as MobileNet-SSD [11], MobileNetV2-SSDLite [28], Pelee [31] and Tiny-DSOD [13], achieve inference on mobile devices at low frame rates. For two-stage detectors, Light-Head R-CNN [14] utilizes a light detection head and runs at over 100 fps on GPU. This raises a question: are two-stage detectors better than one-stage detectors in real-time detection? In this paper, we present the effectiveness of two-stage detectors in real-time detection. Compared with prior lightweight one-stage detectors, ThunderNet achieves a better balance between accuracy and efficiency.

Backbone networks for detection. Modern CNN-based detectors typically adopt image classification networks [30, 10, 32, 12] as the backbones. FPN [16] exploits the inherent multi-scale, pyramidal hierarchy of CNNs to construct feature pyramids. Lightweight detectors also benefit from the recent progress in small networks, such as MobileNet [11, 28] and ShuffleNet [33, 20]. However, image classification and object detection require different properties of networks. Therefore, simply transferring classification networks to object detection is not optimal. For this reason, DetNet [15] designs a backbone specifically for object detection. Recent lightweight detectors [31, 13] also design specialized backbones. However, this area is still not well studied. In this work, we investigate the drawbacks of prior lightweight backbones and present a lightweight backbone for real-time detection task.

3. ThunderNet

In this section, we present the details of ThunderNet. Our design mainly focuses on efficiency, but our model still achieves superior accuracy.

3.1. Backbone Part

Input Resolution. The input resolution of two-stage detectors is usually very large, e.g., FPN [16] uses input images of 800× pixels. It brings several advantages but involves enormous computational cost as well. To improve the inference speed, ThunderNet utilizes the input resolution of 320×320 pixels. Moreover, in practice, we observe that *the input resolution should match the capability of the backbone*. A small backbone with large inputs and a large backbone with small inputs are both not optimal. Details are discussed in Sec. 4.4.1.

Stage	Output Size	Layer		
		SNet49	SNet146	SNet535
Input	224×224	image		
Conv1	112×112	3×3, 24, s2	3×3, 24, s2	3×3, 48, s2
Pool	56×56	3×3 maxpool, s2		
Stage2	28×28 28×28	[60, s2] [60, s1]×3	[132, s2] [132, s1]×3	[248, s2] [248, s1]×3
Stage3	14×14 14×14	[120, s2] [120, s1]×7	[264, s2] [264, s1]×7	[496, s2] [496, s1] ×7
Stage4	7×7 7×7	[240, s2] [240, s1]×3	[528, s2] [528, s1]×3	[992, s2] [992, s1]×3
Conv5	7×7	1×1, 512	-	-
Pool	1×1	global avg pool		
FC		1000-d fc		
FLOPs		49M	146M	535M

Table 1. Architecture of the SNet backbone networks. SNet uses ShuffleNetV2 basic blocks but replaces all 3×3 depthwise convolutions with 5×5 depthwise convolutions.

Backbone Networks. Backbone networks provide basic feature representation of the input image and have great influence on both accuracy and efficiency. CNN-based detectors usually use classification networks transferred from ImageNet classification as the backbone. However, as image classification and object detection require different properties from the backbone, simply transferring classification networks to object detection is not optimal.

Receptive field: The receptive field size plays an important role in CNN models. CNNs can only capture information inside the receptive field. Thus, a large receptive field can leverage more context information and encode long-range relationship between pixels more effectively. This is crucial for the localization subtask, especially for the localization of large objects. Previous works [23, 14] have also demonstrated the effectiveness of the large receptive field in semantic segmentation and object detection.

Early-stage and late-stage features: In the backbone, early-stage feature maps are larger with low-level features which describe spatial details, while late-stage feature maps are smaller with high-level features which are more discriminative. Generally, localization is sensitive to low-level features while high-level features are crucial for classification. In practice, we observe that *localization is more difficult than classification for larger backbones*, which indicates that early-stage features are more important. And *the weak representation power restricts the accuracy in both subtasks for extremely tiny backbones*, suggesting that both early-stage and late-stage features are crucial at this level.

The designs of prior lightweight backbones violate the aforementioned factors: ShuffleNetV1/V2 [33, 20] have restricted receptive field (121 pixels vs. 320 pixels of input),

ShuffleNetV2 [20] and MobileNetV2 [28] lack early-stage features, and Xception [3] suffer from the insufficient high-level features under small computational budgets.

Based on these insights, we start from ShuffleNetV2, and build a lightweight backbone named *SNet* for real-time detection. We present three SNet backbones: *SNet49* for faster inference, *SNet535* for better accuracy, and *SNet146* for a better speed/accuracy trade-off. First, we replace all 3×3 depthwise convolutions in ShuffleNetV2 with 5×5 depthwise convolutions. In practice, 5×5 depthwise convolutions provide similar runtime speed to 3×3 counterparts while effectively enlarging the receptive field (from 121 to 193 pixels). In SNet146 and SNet535, we remove Conv5 and add more channels in early stages. This design generates more low-level features without additional computational cost. In SNet49, we compress Conv5 to 512 channels instead of removing it and increase the channels in the early stages for a better balance between low-level and high-level features. If we remove Conv5, the backbone cannot encode adequate information. But if the 1024-d Conv5 layer is preserved, the backbone suffers from limited low-level features. Table 1 shows the overall architecture of the backbones. Besides, the last output feature maps of Stage3 and Stage4 (Conv5 for SNet49) are denoted as C_4 and C_5.

3.2. Detection Part

Compressing RPN and Detection Head. Two-stage detectors usually adopt large RPN and a heavy detection head. Although Light-Head R-CNN [14] uses a lightweight detection head, it is still too heavy when coupled with small backbones and induces imbalance between the backbone and the detection part. This imbalance not only leads to redundant computation but increases the risk of overfitting.

To address this issue, we compress RPN by replacing the original 256-channel 3×3 convolution with a 5×5 depthwise convolution and a 256-channel 1×1 convolution. We increase the kernel size to enlarge the receptive field and encode more context information. Five scales $\{32^2, 64^2, 128^2, 256^2, 512^2\}$ and five aspect ratios $\{1{:}2, 3{:}4, 1{:}1, 4{:}3, 2{:}1\}$ are used to generate anchor boxes. Other hyperparameters remain the same as in [14].

In the detection head, Light-Head R-CNN generates a thin feature map with $\alpha \times p \times p$ channels before RoI warping, where $p = 7$ is the pooling size and $\alpha = 10$. As the backbones and the input images are smaller in ThunderNet, we further narrow the feature map by halving α to 5 to eliminate redundant computation. For RoI warping, we opt for PSRoI align as it squeezes the number of channels to α.

As the RoI feature from PSRoI align is merely 245-d, we apply a 1024-d fully-connected (*fc*) layer in R-CNN subnet. As demonstrated in Sec. 4.4.3, this design further reduces the computational cost of R-CNN subnet without sacrificing accuracy. Besides, due to the small feature maps, we reduce the number of RoIs for testing as discussed in Sec. 4.1.

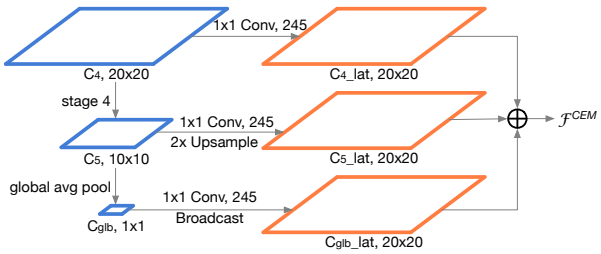

Figure 3. Structure of Context Enhancement Module (CEM). CEM combines feature maps from three scales and encodes more context information. It enlarges the receptive field and generates more discriminative features.

Context Enhancement Module. Light-Head R-CNN applies Global Convolutional Network (GCN) [23] to generate the thin feature map. It significantly increases the receptive field but involves enormous computational cost. Coupled with SNet146, GCN requires 2× the FLOPs needed by the backbone (596M vs. 298M). For this reason, we decide to abandon this design in ThunderNet.

However, the network suffers from the small receptive field and fails to encode sufficient context information without GCN. A common technique to address this issue is Feature Pyramid Network (FPN) [16]. However, prior FPN structures [16, 6, 13, 26] involve many extra convolutions and multiple detection branches, which increases the computational cost and induces enormous runtime latency.

For this reason, we design an efficient *Context Enhancement Module* (CEM) to enlarge the receptive field. The key idea of CEM is to aggregate multi-scale *local* context information and *global* context information to generate more discriminative features. In CEM, the feature maps from three scales are merged: C_4, C_5 and C_{glb}. C_{glb} is the global context feature vector by applying a global average pooling on C_5. We then apply a 1×1 convolution on each feature map to squeeze the number of channels to $\alpha \times p \times p = 245$. Afterwards, C_5 is upsampled by 2× and C_{glb} is broadcast so that the spatial dimensions of the three feature maps are equal. At last, the three generated feature maps are aggregated. By leveraging both local and global context, CEM effectively enlarges the receptive field and refines the representation ability of the thin feature map. Compared with prior FPN structures, CEM involves only two 1×1 convolutions and a *fc* layer, which is more computation-friendly. Fig. 3 illustrates the structure of this module.

Spatial Attention Module. During RoI warping, we expect the features in the background regions to be small and the foreground counterparts to be high. However, compared with large models, as ThunderNet utilizes lightweight backbones and small input images, it is more difficult for the network itself to learn a proper feature distribution.

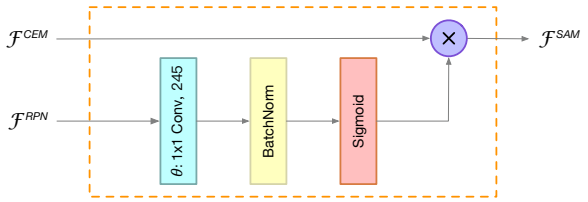

Figure 4. Structure of Spatial Attention Module (SAM). SAM leverages the information learned in RPN to refine the feature distribution of the feature map from Context Enhancement Module. The feature map is then used for RoI warping.

For this reason, we design a computation-friendly *Spatial Attention Module* (SAM) to explicitly re-weight the feature map before RoI warping over the spatial dimensions. The key idea of SAM is to use the knowledge from RPN to refine the feature distribution of the feature map. RPN is trained to recognize foreground regions under the supervision of ground truths. Therefore, the intermediate features in RPN can be used to distinguish foreground features from background features. SAM accepts two inputs: the intermediate feature map from RPN $\mathcal{F}^{RPN}$ and the thin feature map from CEM $\mathcal{F}^{CEM}$. The output of SAM $\mathcal{F}^{SAM}$ is defined as:

$$\mathcal{F}^{SAM} = \mathcal{F}^{CEM} \cdot sigmoid(\theta(\mathcal{F}^{RPN})). \quad (1)$$

Here $\theta(\cdot)$ is a dimension transformation to match the number of channels in both feature maps. The sigmoid function is used to constrain the values within [0, 1]. At last, $\mathcal{F}^{CEM}$ is re-weighted by the generated feature map for better feature distribution. For computational efficiency, we simply apply a 1×1 convolution as $\theta(\cdot)$, so the computational cost of CEM is negligible. Fig. 4 shows the structure of SAM.

SAM has two functions. The first one is to refine the feature distribution by strengthening foreground features and suppressing background features. The second one is to stabilize the training of RPN as SAM enables extra gradient flow from R-CNN subnet to RPN:

$$\frac{\partial \mathcal{L}}{\partial \mathcal{F}_i^{RPN}} = \frac{\partial \mathcal{L}^{RPN}}{\partial \mathcal{F}_i^{RPN}} + \sum\nolimits_{\forall j} \frac{\partial \mathcal{L}^{R\text{-}CNN}}{\partial \mathcal{F}_j^{SAM}} \cdot \frac{\partial \mathcal{F}_j^{SAM}}{\partial \mathcal{F}_i^{RPN}}. \quad (2)$$

As a result, RPN receives additional supervision from R-CNN subnet, which helps the training of RPN.

4. Experiments

In this section, we evaluate the effectiveness of ThunderNet on PASCAL VOC [5] and COCO [18] benchmarks. Then we conduct ablation studies to evaluate our design.

4.1. Implementation Details

Our detectors are trained end-to-end on 4 GPUs using synchronized SGD with a weight decay of 0.0001 and a momentum of 0.9. The batch size is set to 16 images per GPU. Each image has 2000/200 RoIs for training/testing. For efficiency, the input resolution of 320×320 pixels is used instead of $600\times$ or $800\times$ pixels in common large two-stage detectors. Multi-scale training with $\{240, 320, 480\}$ pixels is adopted. As the input resolution is small, we use heavy data augmentation [19]. The networks are trained for 62.5K iterations on VOC dataset and 375K iterations on COCO dataset. The learning rate starts from 0.01 and decays by a factor of 0.1 at 50% and 75% of the total iterations. Online hard example mining [29] is adopted and Soft-NMS [1] is used for post-processing. Cross-GPU Batch Normalization (CGBN) [22] is used to learn batch normalization statistics.

4.2. Results on PASCAL VOC

PASCAL VOC dataset consists of natural images drawn from 20 classes. The networks are trained on the union set of VOC 2007 trainval and VOC 2012 trainval, and we report single-model results on VOC 2007 test. The results are exhibited in Table 2.

ThunderNet surpasses prior state-of-the-art lightweight one-stage detectors. ThunderNet with SNet49 outperforms MobileNet-SSD with merely 21% of the FLOPs, while the SNet146-based model surpasses Tiny-DSOD by 2.9 mAP with about 43% of the FLOPs. Moreover, ThunderNet with SNet146 performs better than Tiny-DSOD by 6.5 mAP under similar computational cost.

Furthermore, ThunderNet achieves superior results to state-of-the-art large object detectors such as YOLOv2 [25], SSD300* [19], SSD321 [19] and R-FCN [4], and is on a par with DSSD321 [6], but reduces the computational cost by orders of magnitude. We note that the backbone of ThunderNet is significantly weaker and smaller than the large detectors. It demonstrates that ThunderNet achieves a much better trade-off between accuracy and efficiency.

4.3. Results on MS COCO

MS COCO dataset consists of natural images from 80 object categories. Following common practice [16, 14], we use trainval35k for training, minival for validation, and report single-model results on test-dev.

As shown in Table 3, ThunderNet with SNet49 achieves MobileNet-SSD level accuracy with 22% of the FLOPs. ThunderNet with SNet146 surpasses MobileNet-SSD [11], MobileNet-SSDLite [28], and Pelee [31] with less than 40% of the computational cost. It is noteworthy that our approach achieves considerably better AP_{75}, which suggests our model performs better in localization. This is consistent with our initial motivation to design two-stage real-time detectors. Compared with Tiny-DSOD [13], ThunderNet achieves better AP but worse AP_{50} with 42% of the FLOPs. We conjecture that deep supervision and feature pyramid in Tiny-DSOD contribute to better classification accuracy. However, ThunderNet is still better in localization.

Model	Backbone	Input	MFLOPs	mAP
YOLOv2 [25]	Darknet-19	416 × 416	17400	76.8
SSD300* [19]	VGG-16	300 × 300	31750	77.5
SSD321 [6]	ResNet-101	321 × 321	15400	77.1
DSSD321 [6]	ResNet-101 + FPN	321 × 321	21200	78.6
R-FCN [4]	ResNet-50	600 × 1000	58900	77.4
Tiny-YOLO [25]	Tiny Darknet	416 × 416	3490	57.1
D-YOLO [21]	Tiny Darknet	416 × 416	2090	67.6
MobileNet-SSD [31]	MobileNet	300 × 300	1150	68.0
Pelee [31]	PeleeNet	304 × 304	1210	70.9
Tiny-DSOD [13]	DDB-Net + D-FPN	300 × 300	1060	72.1
ThunderNet (*ours*)	SNet49	320 × 320	**250**	70.1
ThunderNet (*ours*)	SNet146	320 × 320	**461**	**75.1**
ThunderNet (*ours*)	SNet535	320 × 320	1287	**78.6**

Table 2. Evaluation results on VOC 2007 test. ThunderNet surpasses competing models with significantly less computational cost.

Model	Backbone	Input	MFLOPs	AP	AP_{50}	AP_{75}
YOLOv2 [25]	Darknet-19	416 × 416	17500	21.6	44.0	19.2
SSD300* [19]	VGG-16	300 × 300	35200	25.1	43.1	25.8
SSD321 [6]	ResNet-101	321 × 321	16700	28.0	45.4	29.3
DSSD321 [6]	ResNet-101 + FPN	321 × 321	22300	28.0	46.1	29.2
Light-Head R-CNN [20]	ShuffleNetV2*	800 × 1200	5650	23.7	-	-
MobileNet-SSD [11]	MobileNet	300 × 300	1200	19.3	-	-
MobileNet-SSDLite [28]	MobileNet	320 × 320	1300	22.2	-	-
MobileNetV2-SSDLite [28]	MobileNetV2	320 × 320	800	22.1	-	-
Pelee [31]	PeleeNet	304 × 304	1290	22.4	38.3	22.9
Tiny-DSOD [13]	DDB-Net + D-FPN	300 × 300	1120	23.2	40.4	22.8
ThunderNet (*ours*)	SNet49	320 × 320	**262**	19.1	33.7	19.6
ThunderNet (*ours*)	SNet146	320 × 320	**473**	**23.6**	40.2	**24.5**
ThunderNet (*ours*)	SNet535	320 × 320	1300	**28.0**	**46.2**	**29.5**

Table 3. Evaluation results on COCO test-dev. ThunderNet with SNet49 achieves MobileNet-SSD level accuracy with 22% of the FLOPs. ThunderNet with SNet146 achieves superior accuracy to prior lightweight one-stage detectors with merely 40% of the FLOPs. ThunderNet with SNet535 rivals large detectors with significantly less computational cost.

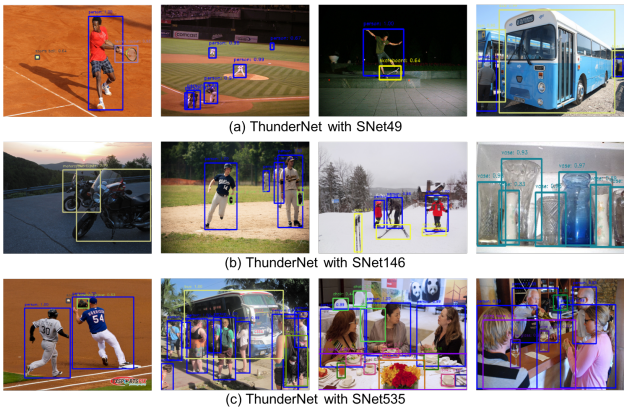

Figure 5. Examples visualization on COCO test-dev.

ThunderNet with SNet535 achieves significantly better detection accuracy under comparable computational cost. As shown in Table 3, ThunderNet surpasses other one-stage counterparts by at least 4.8 AP, 5.8 AP_{50} and 6.7 AP_{75}. The gap in AP_{75} is larger than the gap in AP_{50}, which means our model provides more accurate bounding boxes than other detectors. This further demonstrates that two-stage detectors are prior to one-stage detectors in real-time detection task. Fig. 5 visualizes several examples on COCO test-dev.

We also compare ThunderNet with large one-stage detectors. ThunderNet with SNet146 surpasses YOLOv2 [25] with 37× fewer FLOPs. And ThunderNet with SNet535 significantly outperforms YOLOv2 and SSD300 [19], and rivals SSD321 [6] and DSSD321 [6]. It suggests that ThunderNet is not only efficient but highly accurate.

4.4. Ablation Experiments

4.4.1 Input Resolution

We first explore the relationship between the input resolution and the backbone. Table 4 reveals that large backbones with small images and small backbones with large images are both not optimal. There is a trade-off between the two factors. On the one hand, small images lead to low-resolution feature maps and induce severe loss of detail features. It is hard to be remedied by simply increasing the capacity of the backbones. On the other hand, small backbones are too weak to encode sufficient information from large images. The backbone and the input images should match for a better balance between the representation ability and the resolution of the feature maps.

4.4.2 Backbone Networks

We then evaluate the design of the backbones. SNet146 and SNet49 are used as the baselines. SNet146 achieves 32.5% top-1 error on ImageNet classification and 23.6 AP on COCO test-dev (Table 5(a)), while SNet49 achieves 39.7% top-1 error and 19.1 AP (Table 5(e)).

5×5 Depthwise Convolutions. We evaluate the effectiveness of 5×5 depthwise convolutions on SNet146. We first

Backbone	Input	MFLOPs	AP
SNet49	320 × 320	262	**19.1**
SNet146	224 × 224	267	18.7
SNet535	128 × 128	265	13.2
SNet49	480 × 480	506	22.0
SNet146	320 × 320	473	**23.6**
SNet535	192 × 192	512	20.2

Table 4. Evaluation of different input resolutions on COCO test-dev. Large backbones with small images and small backbones with large images are both not optimal.

replace all 5×5 depthwise convolutions with 3×3 depthwise convolutions. For fair comparison, the channels from Stage2 to Stage4 are slightly increased to maintain the computational cost unchanged. This model performs worse on both image classification (by 0.2%) and object detection (by 0.9 AP) (Table 5(b)). Compared with 3×3 depthwise convolutions, 5×5 depthwise convolutions considerably increase the receptive fields, which helps in both tasks.

We then add another 3×3 depthwise convolution before the first 1×1 convolution in all building blocks as in ShuffleNetV2* [20]. The number of channels is kept unchanged as the baseline. This model is comparable on image classification, but slightly worse on object detection (by 0.3 AP) (Table 5(c)). As this model and SNet146 have the same receptive fields theoretically, we conjecture that 5×5 depthwise convolutions can provide larger valid receptive fields, which is especially crucial in object detection.

Early-stage and Late-stage Features. To investigate the trade-off between early-stage and late-stage features, we first add a 1024-channel Conv5 in SNet146. The channels in the early stages are reduced accordingly. This model slightly improves the top-1 error, but reduces AP by 0.4 (Table 5(d)). A wide Conv5 generates more discriminative features, which improves the classification accuracy. However, object detection focuses on both classification and localization. Increasing the channels in early stages encodes more detail information, which is beneficial for localization.

For SNet49, we first remove Conv5 in SNet49 and increase the channels from Stage2 to Stage4. Table 5(f) shows that both the classification and the detection performance suffer from severe degradation. Removing Conv5 cuts the output channels of the backbone by half, which hinders the model from learning adequate information.

We then extend Conv5 to 1024 channels as in the original ShuffleNetV2. The early-stage channels are compressed to maintain the same overall computational cost. This model surpasses SNet49 on image classification by 0.8%, but performs worse on object detection (Table 5(g)). By leveraging a wide Conv5, this model benefits from more high-level features in image classification. However, it suffers from the lack of low-level features in object detection. It further demonstrates the differences between image classification and object detection.

Backbone	MFLOPs	Top-1 Err.	AP
(a) SNet146	146	32.5	**23.6**
(b) SNet146 + 3×3 DWConv	145	32.7	22.7
(c) SNet146 + double 3×3 DWConv	143	32.4	23.3
(d) SNet146 + 1024-d Conv5	147	**32.3**	23.2
(e) SNet49	49	39.7	**19.1**
(f) SNet49 + No Conv5	49	40.8	18.2
(g) SNet49 + 1024-d Conv5	49	**38.9**	18.8

Table 5. Evaluation of different backbones on ImageNet classification and COCO test-dev. **DWConv**: depthwise convolution.

Backbone	MFLOPs	Top-1 Err.	AP
ShuffleNetV1 [33]	137	34.8	20.8
ShuffleNetV2 [20]	147	**31.4**	22.7
ShuffleNetV2* [20]	145	32.2	23.2
Xception [3]	145	34.1	23.0
MobileNetV2 [28]	145	32.9	22.7
SNet146	146	32.5	**23.6**

Table 6. Evaluation of lightweight backbones on COCO test-dev. SNet146 achieves better detection results though the classification accuracy is lower.

Comparison with Lightweight Backbones. At last, we further compare SNet with other lightweight backbones. Table 6 shows that SNet146 outperforms Xception [3], MobileNetV2 [28], and ShuffleNetV1/V2/V2* [33, 20] on object detection under similar computational cost. These results further demonstrate the effectiveness of our design.

4.4.3 Detection Part

We also investigate the effectiveness of the design of the detection part in ThunderNet. Table 7 describes the comparison of the model variants in the experiments.

Baseline. We choose a compressed Light-Head R-CNN [14] with SNet146 as the baseline. C_5 is upsampled by $2\times$ to obtain the same downsampling rate. C_4 and C_5 are then squeezed to 245 channels and sent to RPN and RoI warping respectively. We use a 256-channel 3×3 convolution in RPN and a 2048-d *fc* layer in R-CNN subnet. This model requires 703 MFLOPs and achieves 21.9 AP (Table 7(a)). Besides, we would mention that multi-scale training, CGBN [22], and Soft-NMS [1] gradually improve the baseline by 1.4 AP (from 20.5 to 21.9 AP).

RPN and R-CNN subnet. We first replace the 3×3 convolution in RPN with a 5×5 depthwise convolution and a 1×1 convolution. The number of output channels remains unchanged. This design reduces the computational cost by 28% without harming the accuracy (Table 7(b)). We then halve the number of outputs of the *fc* layer in R-CNN subnet to 1024, which achieves a further 13% compression on the FLOPs with a marginal decrease of 0.2 AP. (Table 7(c)). These results demonstrate that heavy RPN and R-CNN subnet introduce great redundancy for lightweight detectors. More details will be discussed in Sec. 4.4.4.

Context Enhancement Module. We then insert Context Enhancement Module (CEM) after the backbone. The output feature map of CEM is used for both RPN and RoI

	BL	SRPN	SRCN	CEM	SAM	AP	AP_{50}	AP_{75}	MFLOPs
(a)	✓					21.9	37.6	22.5	714
(b)		✓				21.8	37.5	22.4	516
(c)		✓	✓			21.6	37.4	22.2	**448**
(d)		✓	✓	✓		23.3	39.9	24.0	449
(e)		✓	✓		✓	22.9	39.0	23.8	473
(f)		✓	✓	✓	✓	**23.6**	**40.2**	**24.5**	473

Table 7. Ablation studies on the detection part on COCO test-dev. We use a compressed Light-Head R-CNN with SNet146 as the baseline (BL), and gradually add small RPN (SRPN), small R-CNN (SRCN), Context Enhancement Module (CEM) and Spatial Attention Module (SAM) for ablation studies.

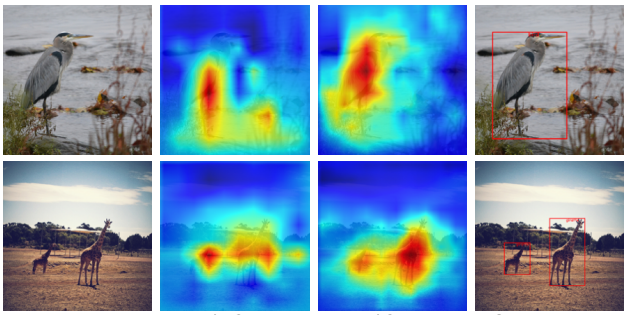

Figure 6. Visualization of the feature map before RoI warping. Spatial Attention Module (SAM) enhances the features in the foreground regions and weakens those in the background regions.

warping. CEM achieves thorough improvements of 1.7 AP, 2.5 AP_{50} and 1.8 AP_{75} with negligible increase on FLOPs (Table 7(d)). The combination of the multi-scale feature maps introduces semantic and context information of different levels, which improves the representation ability.

Spatial Attention Module. Adopting Spatial Attention Module (SAM) without CEM (Table 7(e)) improves AP by 1.3 with merely 5% extra computational cost compared with Table 7(c). Fig. 6 visualizes the feature maps before RoI warping in Table 7(c) and Table 7(e). It is clear that SAM effectively refines the feature distribution with foreground feature enhanced and background features weakened.

At last, we adopt both CEM and SAM to compose the complete ThunderNet (Table 7(f)). This setting improves AP by 1.7, AP_{50} by 2.6, and AP_{75} by 2.0 over the baseline while reducing the computational cost by 34%. These results have demonstrated the effectiveness of our design.

4.4.4 Balance between Backbone and Detection Head

We further explore the relationship between the backbone and the detection head. Two models are used in the experiments: a *large-backbone-small-head* model and a *small-backbone-large-head* model. The large-backbone-small-head model is ThunderNet with SNet146. While the small-backbone-large-head model uses SNet49 and a heavier head: α in the thin feature map is 10, and a 2048-d *fc* layer is used in R-CNN subnet. As shown in Table 8, the

Model	Backbone	RPN	Head	Total	AP
large-backbone-small-head	338	43	92	473	23.6
small-backbone-large-head	154	70	286	510	20.2

Table 8. MFLOPs and AP of different detection head designs on COCO test-dev. The large-backbone-small-head model outperforms the small-backbone-large-head model with less FLOPs.

Model	ARM	CPU	GPU
Thunder w/ SNet49	24.1	47.3	267
Thunder w/ SNet146	13.8	32.3	248
Thunder w/ SNet535	5.8	15.3	214

Table 9. Inference speed in fps on Snapdragon 845 (ARM), Xeon E5-2682v4 (CPU) and GeForce 1080Ti (GPU).

large-backbone-small-head model outperforms the small-backbone-large-head one by 3.4 AP even with less computational cost. It suggests that *the large-backbone-small-head design is better than the small-backbone-large-head design for lightweight two-stage detectors*. We conjecture that the capability of the backbone and the detection head should match. In the small-backbone-large-head design, the features from the backbone are relatively weak, which makes the powerful detection head redundant.

4.5. Inference Speed

At last, we evaluate the inference speed of ThunderNet on Snapdragon 845 (ARM), Xeon E5-2682v4 (CPU) and GeForce 1080Ti (GPU). On ARM and CPU, the inference is executed with a *single thread*. The batch normalization layers are merged with the preceding convolutions for faster inference speed. The results are shown in Table 9. ThunderNet with SNet49 achieves real-time detection on both ARM and CPU at 24.1 and 47.3 fps, respectively. To the best of our knowledge, this is the *first* real-time detector and the *fastest* single-thread speed on ARM platforms ever reported. ThunderNet with SNet146 runs at 13.8 fps on ARM and runs in real-time on CPU at 32.3 fps. All three models run at over 200 fps on GPU. These results suggest that ThunderNet is highly efficient in real-world applications.

5. Conclusion

We investigate the effectiveness of two-stage detectors in real-time generic object detection and propose a lightweight two-stage detector named ThunderNet. In the backbone part, we analyze the drawbacks in prior lightweight backbones and present a lightweight backbone designed for object detection. In the detection part, we adopt an extremely efficient design in the detection head and RPN. Context Enhancement Module and Spatial Attention Module are designed to improve the feature representation. At last, we investigate the balance between the input resolution, the backbone, and the detection head. ThunderNet achieves superior detection accuracy to prior one-stage detectors with significantly less computational cost. To the best of our knowl-

edge, ThunderNet achieves the first real-time detector and the fastest single-thread speed reported on ARM platforms.

References

[1] N. Bodla, B. Singh, R. Chellappa, and L. S. Davis. Soft-nms–improving object detection with one line of code. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5561–5569, 2017.

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

[3] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[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.

[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[6] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

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

[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.

[9] 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.

[10] 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.

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[12] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[13] Y. Li, J. Li, W. Lin, and J. Li. Tiny-dsod: Lightweight object detection for resource-restricted usages. *arXiv preprint arXiv:1807.11013*, 2018.

[14] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Light-head r-cnn: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017.

[15] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun. Detnet: Design backbone for object detection. In *The European Conference on Computer Vision (ECCV)*, 2018.

[16] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[17] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[18] 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.

[19] 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.

[20] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[21] R. Mehta and C. Ozturk. Object detection at 200 frames per second. *arXiv preprint arXiv:1805.06361*, 2018.

[22] C. Peng, T. Xiao, Z. Li, Y. Jiang, X. Zhang, K. Jia, G. Yu, and J. Sun. Megdet: A large mini-batch object detector. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6181–6189, 2018.

[23] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[24] 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.

[25] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[26] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[27] 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.

[28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[29] 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.

[30] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[31] R. J. Wang, X. Li, and C. X. Ling. Pelee: a real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems*, pages 1963–1972, 2018.

[32] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1492–1500, 2017.

[33] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.