CSPNet: A New Backbone that can Enhance Learning Capability of CNN

Chien-Yao Wang[1], Hong-Yuan Mark Liao[1,2], Yueh-Hua Wu[1,3], Ping-Yang Chen[4], Jun-Wei Hsieh[5], and I-Hau Yeh[6]
[1]Institute of Information Science, Academia Sinica, Taiwan
[2]Department of Computer Science and Information Engineering, Providence University, Taiwan
[3]Department of Computer Science and Information Engineering, National Taiwan University, Taiwan
[4]Department of Computer Science, National Chiao Tung University, Taiwan
[5]College of Artificial Intelligence and Green Energy, National Chiao Tung University, Taiwan
[6]Elan Microelectronics Corporation, Taiwan
kinyiu@iis.sinica.edu.tw, liao@iis.sinica.edu.tw, kriswu@iis.sinica.edu.tw
pingyang.cs08g@nctu.edu.tw, jwhsieh@nctu.edu.tw, and ihyeh@emc.com.tw

Abstract

Neural networks have enabled state-of-the-art approaches to achieve incredible results on computer vision tasks such as object detection. However, such success greatly relies on costly computation resources, which hinders people with cheap devices from appreciating the advanced technology. In this paper, we propose Cross Stage Partial Network (CSPNet) to mitigate the problem that previous works require heavy inference computations from the network architecture perspective. We attribute the problem to the duplicate gradient information within network optimization. The proposed networks respect the variability of the gradients by integrating feature maps from the beginning and the end of a network stage, which, in our experiments, reduces computations by 20% with equivalent or even superior accuracy on the ImageNet dataset, and significantly outperforms state-of-the-art approaches in terms of AP_{50} on the MS COCO object detection dataset. The CSPNet is easy to implement and general enough to cope with architectures based on ResNet, ResNeXt, and DenseNet.

1. Introduction

Neural networks have been shown to be especially powerful when it gets deeper [8, 37, 11] and wider [38]. However, extending the architecture of neural networks usually brings up a lot more computations, which makes computationally heavy tasks such as object detection unaffordable for most people. Light-weight computing has gradually received stronger attention since real-world applications usually require short inference time on small devices, which poses a serious challenge for computer vision algorithms. Although some approaches were designed exclusively for mobile CPU [10, 30, 9, 32, 41, 23], the depth-wise convolution they adopted is usually not compatible with industrial IC design such as Application-Specific Integrated Circuit (ASIC) for edge-computing systems. In this work, we investigate the computational burden in state-of-the-art ap-

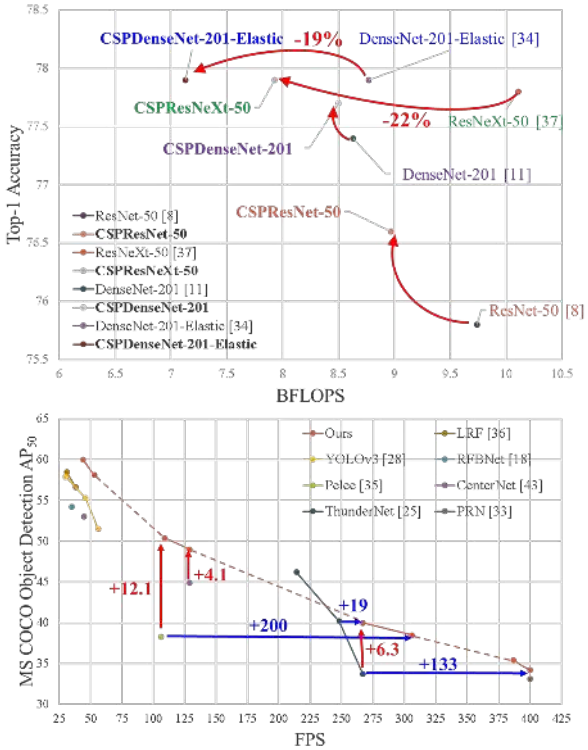

Figure 1: Proposed CSPNet can be applied on ResNet [8], ResNeXt [37], DenseNet [11], etc. It not only reduce computation cost and memory usage of these networks, but also benefit on inference speed and accuracy.

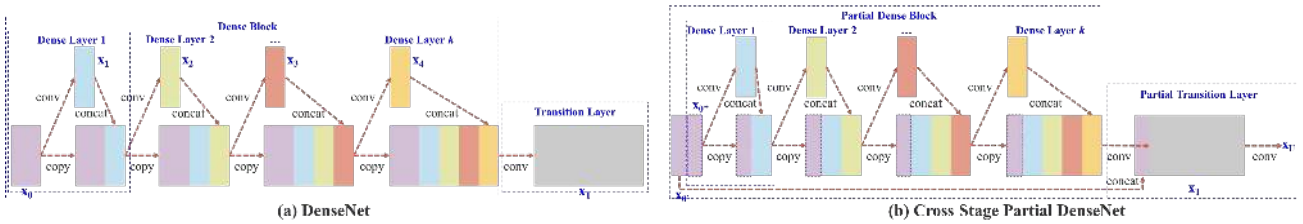

Figure 2: Illustrations of (a) DenseNet and (b) our proposed Cross Stage Partial DenseNet (CSPDenseNet). CSPNet separates feature map of the base layer into two part, one part will go through a dense block and a transition layer; the other one part is then combined with transmitted feature map to the next stage.

proaches such as ResNet, ResNeXt, and DenseNet. We further develop computationally efficient components that enable the mentioned networks to be deployed on both CPUs and mobile GPUs without sacrificing the performance.

In this study, we introduce Cross Stage Partial Network (CSPNet). The main purpose of designing CSPNet is to enable this architecture to achieve a richer gradient combination while reducing the amount of computation. This aim is achieved by partitioning feature map of the base layer into two parts and then merging them through a proposed cross-stage hierarchy. Our main concept is to make the gradient flow propagate through different network paths by splitting the gradient flow. In this way, we have confirmed that the propagated gradient information can have a large correlation difference by switching concatenation and transition steps. In addition, CSPNet can greatly reduce the amount of computation, and improve inference speed as well as accuracy, as illustrated in Fig 1. The proposed CSPNet-based object detector deals with the following three problems:

1) Strengthening learning ability of a CNN The accuracy of existing CNN is greatly degraded after lightweightening, so we hope to strengthen CNN's learning ability, so that it can maintain sufficient accuracy while being lightweightening. The proposed CSPNet can be easily applied to ResNet, ResNeXt, and DenseNet. After applying CSPNet on the above mentioned networks, the computation effort can be reduced from 10% to 20%, but it outperforms ResNet [8], ResNeXt [37], DenseNet [11], HarDNet [2], Elastic [34], and Res2Net [5], in terms of accuracy, in conducting image classification task on ImageNet [3].

2) Removing computational bottlenecks Too high a computational bottleneck will result in more cycles to complete the inference process, or some arithmetic units will often idle. Therefore, we hope we can evenly distribute the amount of computation at each layer in CNN so that we can effectively upgrade the utilization rate of each computation unit and thus reduce unnecessary energy consumption. It is noted that the proposed CSPNet makes the computational bottlenecks of PeleeNet [35] cut into half. Moreover, in the MS COCO [17] dataset-based object detection experiments, our proposed model can effectively reduce 80% computational bottleneck when tested on YOLOv3-based models.

3) Reducing memory costs The wafer fabrication cost of Dynamic Random-Access Memory (DRAM) is very expensive, and it also takes up a lot of space. If one can effectively reduce the memory cost, he/she will greatly reduce the cost of ASIC. In addition, a small area wafer can be used in a variety of edge computing devices. We adopt cross-channel pooling [6] to compress the feature maps during the feature pyramid generating process. In this way, the proposed CSPNet with the proposed detector can cut down 75% memory usage on PeleeNet when generating feature pyramids.

Since CSPNet is able to promote the learning capability of a CNN, we thus use smaller models to achieve 50% COCO AP_{50} at 109 fps on GTX 1080ti. Since CSPNet can effectively cut down a significant amount of memory traffic, our proposed method can achieve 40% COCO AP_{50} at 52 fps on Intel Core i9-9900K. In addition, since CSPNet can significantly lower down the computational bottleneck and Exact Fusion Model (EFM) can effectively cut down the required memory bandwidth, our proposed method can achieve 42% COCO AP_{50} at 49 fps on Nvidia Jetson TX2.

2. Related work

CNN architectures design. In ResNeXt [37], Xie *et al*. first demonstrate that cardinality can be more effective than the dimensions of width and depth. DenseNet [11] can significantly reduce the number of parameters and computations due to the strategy of adopting a large number of reuse features. And it concatenates the output features of all preceding layers as the next input, which can be considered as the way to maximize cardinality. SparseNet [44] adjusts dense connection to exponentially spaced connection can effectively improve parameter utilization and thus result in better outcomes. Wang *et al*. further explain why high cardinality and sparse connection can improve the learning ability of the network by the concept of gradient combination and developed the partial ResNet (PRN) [33]. For improving the inference speed of CNN, Ma *et al*. [23] introduce four guidelines to be followed and design ShuffleNet-v2. Chao *et al*. [2] proposed a low memory traffic CNN called Harmonic DenseNet (HarDNet) and a metric Convolutional Input/Output (CIO) which is an approximation of DRAM traffic proportional to the real DRAM traffic measurement.

Real-time object detector. The most famous two real-time object detectors are YOLOv3 [28] and SSD [20]. Based on SSD, LRF [36] and RFBNet [18] can achieve state-of-the-art real-time object detection performance on GPU. Recently, anchor-free based object detector [4, 43, 13, 14, 40] has become main-stream object detection system. Two object detectors of this sort are CenterNet [43] and CornerNet-Lite [14], and they both perform very well in terms of efficiency and efficacy. For real-time object detection on CPU or mobile GPU, SSD-based Pelee [35], YOLOv3-based PRN [33], and Light-Head RCNN [16]-based ThunderNet [25] all receive excellent performance on object detection.

3. Method

3.1. Cross Stage Partial Network

Cross Stage Partial Network. The mainstream CNN architectures, such as ResNet [8], ResNeXt [37], DenseNet [11], their output is usually a linear or non-linear combination of the outputs of intermediate layers. Therefore, the output of a k-layer CNN can be expressed as follows:

$$\begin{aligned} y &= F(x_0) = x_k \\ &= H_k(x_{k-1}, H_{k-1}(x_{k-2}), H_{k-2}(x_{k-3}), \ldots, , H_1(x_0), x_0) \end{aligned} \tag{1}$$

where F is the mapping function from input x_0 to target y, which is also the model of the entire CNN. As for H_k, it is the operation function of the k^{th} layer of the CNN. Usually, H_k is composed of a set of convolutional layers and a non-linear activation function. If we use ResNet and DenseNet as examples, they can be represented by Equation 2 and Equation 3 respectively as follows:

$$\begin{aligned} x_k &= R_k(x_{k-1}) + x_{k-1} \\ &= R_k(x_{k-1}) + R_{k-1}(x_{k-2}) + \ldots + R_1(x_0) + x_0 \end{aligned} \tag{2}$$

$$\begin{aligned} x_k &= [D_k(x_{k-1}), x_{k-1}] \\ &= [D_k(x_{k-1}), D_{k-1}(x_{k-2}), \ldots, D_1(x_0), x_0] \end{aligned} \tag{3}$$

In the above two equations, R and D respectively represent the computation operators of the residual layer and dense layer, and these operators often composed of 2$\sim$3 convolutional layers.

From the above two equations, whether it is a residual layer or a dense layer, the input of each convolutional layer that composes them receives the outputs from all the previous layers. Under these circumstances, the length of gradient path can be minimized and makes gradient flow propagation more efficient in the back propagation process. However, we also know that this architecture design will make the k^{th} layer pass the gradient to all $k-1, k-2, ..., 1$ layers and use it to update the weights, which will cause repeated learning redundant information.

Recently, some studies have tried to use the input of screened $H_k(.)$ to improve learning ability and parameter utilization. For example, SparseNet [44] uses exponentially spaced connection to make H_k directly related to $H_{k-1}, H_{k-2}, H_{k-4}, ..., H_{k-2^i}, ...$ only. ShuffleNetV2 [23] use split channels to make H_k directly related to only half of H_{k-1} channels, and its equation can be expressed as $S([H_k(x_{k-1}[1:c/2]), x_{k-1}[(c/2+1):c]])$, where S represents the shuffle operation, and $x_{k-1}[1:c/2]$ represents the first to the $c/2$ channels of x_{k-1}. As for the PyramidNet [7] and PRN [33], they all use feature maps with unequal number of channels to build ResNet to acheive the effect of gradient shunting.

The state-of-the-art methods put their emphasis on optimizing the H_i function at each layer, and we propose that CSPNet directly optimizes the F function as follows:

$$y = M\left(\left[x_{0'}, T\left(F(x_{0''})\right)\right]\right) \tag{4}$$

where x_0 is split into two parts along channel and it can be represented as $x_0 = [x_{0'}, x_{0''}]$. T is the transition function used to truncate the gradient flows of $H_1, H_2, ..., H_k$, and M is the transition function used to mix the two segmented parts. Next, we will show examples of how to integrate CSPNet into DenseNet and explain how to solve the problem of learning duplicate information in CNN.

DenseNet. Figure 2 (a) shows the detailed structure of one-stage of the DenseNet proposed by Huang *et al.* [11]. Each stage of a DenseNet contains a dense block and a transition layer, and each dense block is composed of k dense layers. The output of the i^{th} dense layer will be concatenated with the input of the i^{th} dense layer, and the concatenated outcome will become the input of the $(i+1)^{th}$ dense layer. The equations showing the above-mentioned mechanism can be expressed as:

$$\begin{aligned} x_1 &= w_1 * x_0 \\ x_2 &= w_2 * [x_0, x_1] \\ &\vdots \\ x_k &= w_k * [x_0, x_1, \ldots, x_{k-1}] \end{aligned} \tag{5}$$

where $*$ represents the convolution operator, and $[x_0, x_1, ...]$ means to concatenate $x_0, x_1, ...$, and w_i and x_i are the weights and output of the i^{th} dense layer, respectively.

If one makes use of a backpropagation to update weights, the equations of weight updating can be written as:

$$\begin{aligned} w'_1 &= f_1(w_1, \{g_0\}) \\ w'_2 &= f_2(w_2, \{g_0, g_1\}) \\ &\vdots \\ w'_k &= f_k(w_k, \{g_0, g_1, \ldots, g_{k-1}\}) \end{aligned} \tag{6}$$

where f_i is the function of weight updating of i^{th} dense layer, and g_i represents the gradient propagated to the i^{th} dense layer. We can find that large amount of gradient information are reused for updating weights of different dense layers. This will result in different dense layers repeatedly learn copied gradient information.

Cross Stage Partial DenseNet. The architecture of one-stage of the proposed CSPDenseNet is shown in Figure 2 (b). A stage of CSPDenseNet is composed of a partial dense block and a partial transition layer. In a partial dense block, the feature maps of the base layer in a stage are split into two parts through channel $x_0 = [x_{0'}, x_{0''}]$. Between $x_{0'}$ and $x_{0''}$, the former is directly linked to the end of the stage, and the latter will go through a dense block. All steps involved in a partial transition layer are as follows: First, the output of dense layers, $[x_{0''}, x_1, ..., x_k]$, will undergo a transition layer. Second, the output of this transition layer, x_T, will be concatenated with $x_{0'}$ and undergo another transition layer, and then generate output x_U. The equations of feed-forward pass and weight updating of CSPDenseNet are shown in Equations 7 and 8, respectively.

$$\begin{aligned} x_k &= w_k * [x_{0''}, x_1, ..., x_{k-1}] \\ x_T &= w_T * [x_{0''}, x_1, ..., x_k] \\ x_U &= w_U * [x_{0'}, x_T] \end{aligned} \tag{7}$$

$$\begin{aligned} w'_k &= f_k(w_k, \{g_{0''}, g_1, ..., g_{k-1}\}) \\ w'_T &= f_T(w_T, \{g_{0''}, g_1, ..., g_k\}) \\ w'_U &= f_U(w_U, \{g_{0'}, g_T\}) \end{aligned} \tag{8}$$

We can see that the gradients coming from the dense layers are separately integrated. On the other hand, the feature map x_0' that did not go through the dense layers is also separately integrated. As to the gradient information for updating weights, both sides do not contain duplicate gradient information that belongs to other sides.

Overall speaking, the proposed CSPDenseNet preserves the advantages of DenseNet's feature reuse characteristics, but at the same time prevents an excessively amount of duplicate gradient information by truncating the gradient flow. This idea is realized by designing a hierarchical feature fusion strategy and used in a partial transition layer.

Partial Dense Block. The advantages of designing partial dense blocks are: *1.) increase gradient path*: Through the split and merge strategy, the number of gradient paths can be doubled. Because of the cross-stage strategy, one can alleviate the disadvantages caused by using explicit feature map copy for concatenation; *2.) balance computation of each layer*: usually, the channel number in the base layer of a DenseNet is much larger than the growth rate. Since the base layer channels involved in the dense layer operation in a partial dense block account for only half of the original number, it can effectively solve nearly half of the computational bottleneck; and *3.) reduce memory traffic*: Assume the base feature map size of a dense block in a DenseNet is $w \times h \times c$, the growth rate is d, and there are in total m dense layers. Then, the CIO of that dense block is $(c \times m) + ((m^2 + m) \times d)/2$, and the CIO of partial dense block is $((c \times m) + (m^2 + m) \times d)/2$. While m and d are usually far smaller than c, a partial dense block is able to save at most half of the memory traffic of a network.

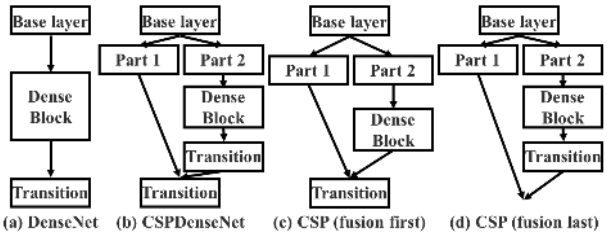

Figure 3: Different kind of feature fusion strategies. (a) single path DenseNet, (b) proposed CSPDenseNet: transition → concatenation → transition, (c) concatenation → transition, and (d) transition → concatenation.

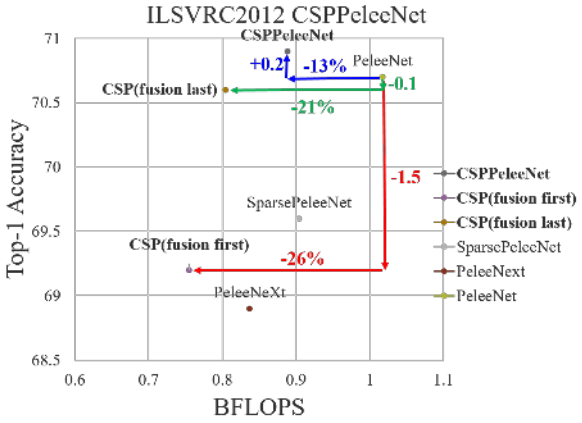

Figure 4: Effect of truncating gradient flow for maximizing difference of gradient combination.

Partial Transition Layer. The purpose of designing partial transition layers is to maximize the difference of gradient combination. The partial transition layer is a hierarchical feature fusion mechanism, which uses the strategy of truncating the gradient flow to prevent distinct layers from learning duplicate gradient information. Here we design two variations of CSPDenseNet to show how this sort of gradient flow truncating affects the learning ability of a network. Figure 3 (c) and Figure 3 (d) show two different fusion strategies. CSP (fusion first) means to concatenate the feature maps generated by two parts, and then do transition operation. If this strategy is adopted, a large amount of gradient information will be reused. As to the CSP (fusion last) strategy, the output from the dense block will go through the transition layer and then do concatenation with the feature map coming from part 1. If one goes with the CSP (fusion last) strategy, the gradient information will not be reused since the gradient flow is truncated. If we use the four architectures shown in 3 to perform image classification, the corresponding results are shown in Figure 4. It can be seen that if one adopts the CSP (fusion last) strategy to perform image classification, the computation cost is significantly dropped, but the top-1 accuracy only drop 0.1%. On the other hand, the CSP (fusion first) strategy does help the significant drop in computation cost, but the top-1 accu-

racy significantly drops 1.5%. By using the split and merge strategy across stages, we are able to effectively reduce the possibility of duplication during the information integration process. From the results shown in Figure 4, it is obvious that if one can effectively reduce the repeated gradient information, the learning ability of a network will be greatly improved.

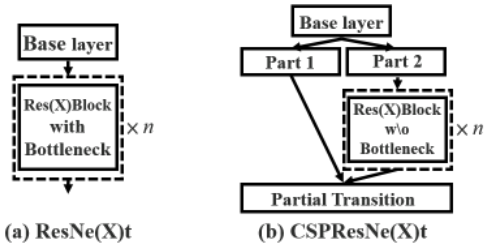

Figure 5: Applying CSPNet to ResNe(X)t.

Apply CSPNet to Other Architectures. CSPNet can also be applied to ResNet and ResNeXt, the architectures are shown in Figure 5. Since only half of the feature channels are going through Res(X)Blocks, there is no need to introduce the bottleneck layer anymore. This makes the theoretical lower bound of the Memory Access Cost (MAC) when the FLoating-point OPerations (FLOPs) is fixed.

3.2. Exact Fusion Model

Looking Exactly to predict perfectly. We propose EFM that captures an appropriate receptive field for each anchor, which enhances the accuracy of the one-stage object detector. For segmentation tasks, since pixel-level labels usually do not contain global information, it is usually more preferable to consider larger patches for better information retrieval [21]. However, for tasks like image classification and object detection, some critical information can be obscure when observed from image-level and bounding box-level labels. Li *et al*. [15] found that CNN can be often distracted when it learns from image-level labels and concluded that it is one of the main reasons that two-stage object detectors outperform one-stage object detectors.

Aggregate Feature Pyramid. The proposed EFM is able to better aggregate the initial feature pyramid. The EFM is based on YOLOv3 [28], which assigns exactly one bounding-box prior to each ground truth object. Each ground truth bounding box corresponds to one anchor box that surpasses the threshold IoU. If the size of an anchor box is equivalent to the receptive field of the grid cell, then for the grid cells of the s^{th} scale, the corresponding bounding box will be lower bounded by the $(s-1)^{th}$ scale and upper bounded by the $(s+1)^{th}$ scale. Therefore, the EFM assembles features from the three scales.

Balance Computation. Since the concatenated feature maps from the feature pyramid are enormous, it introduces a great amount of memory and computation cost. To alleviate the problem, we incorporate the Maxout technique to compress the feature maps.

4. Experiments

We use ImageNet image classification dataset [3] to verify proposed CSPNet. And use the MS COCO object detection dataset [17] to verify the proposed CSPNet and EFM.

4.1. Implementation Details

ImageNet. In ImageNet image classification experiments, all hyper-parameters such as training steps, learning rate schedule, optimizer, data augmentation, etc., we all follow the settings defined in Redmon *et al*. [28]. For ResNet-based models and ResNeXt-based models, we set 8,000,000 training steps. As to DenseNet-based models, we set 1,600,000 training steps. We set the initial learning rate 0.1 and adopt the polynomial decay learning rate scheduling strategy. The momentum and weight decay are respectively set as 0.9 and 0.005. All architectures use a single GPU to train universally in the batch size of 128. Finally, we use the validation set of ILSVRC 2012 to validate our method.

MS COCO. In MS COCO object detection experiments, all hyper-parameters also follow the settings defined in Redmon *et al*. [28]. Altogether we did 500,000 training steps. We adopt the step decay learning rate scheduling strategy and multiply with a factor 0.1 at the 400,000 steps and the 450,000 steps, respectively. The momentum and weight decay are respectively set as 0.9 and 0.0005. All architectures use a single GPU to execute multi-scale training in the batch size of 64. Finally, the COCO test-dev set is adopted to verify our method.

4.2. Ablation Experiments

Ablation study of CSPNet on ImageNet. In the ablation experiments conducted on the CSPNet, we adopt PeleeNet [35] as the baseline, and the ImageNet is used to verify the performance of the CSPNet. We use different partial ratios γ and feature fusion strategies for ablation study. Table 1 shows the results of ablation study on CSPNet. As to CSP (fusion first) and CSP (fusion last), they are proposed to validate the benefits of a partial transition.

From the experimental results of CSP (fusion last), the partial transition layer designed to reduce the learning of redundant information can achieve very good performance. For example, when the computation is cut down by 21%, the accuracy only degrades by 0.1%. One thing to be noted is that when $\gamma = 0.25$, the computation is cut down by 11%, but the accuracy is increased by 0.1%. Compared to the baseline PeleeNet, the proposed CSPPeleeNet achieves the best performance, it can cut down 13% computation, but at the same time upgrade the accuracy by 0.2%. If we adjust the partial ratio to $\gamma = 0.25$, we are able to upgrade the accuracy by 0.8% and cut down 3% computation.

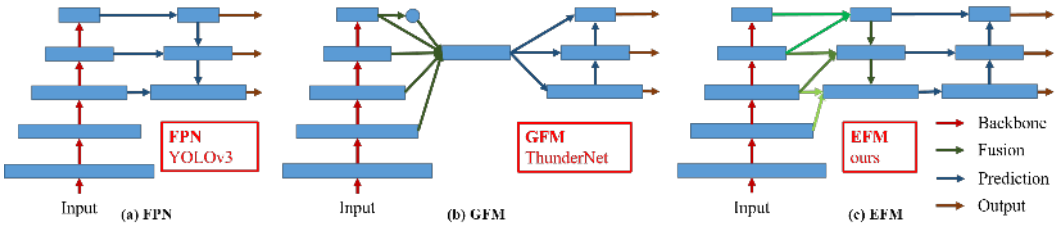

Figure 6: Different feature pyramid fusion strategies. (a) Feature Pyramid Network (FPN): fuse features from current scale and previous scale. (b) Global Fusion Model (GFM): fuse features of all scales. (c) Exact Fusion Model (EFM): fuse features depand on anchor size.

Table 1: Ablation study of CSPNet on ImageNet.

Model	γ	two-way dense	partial dense	trans.	partial trans.	Top-1	BFLOPs
PeleeNet [35]	-	✓		✓		70.7	1.017
CSP (fusion first)	0.75		✓	✓		68.4	0.649
	0.5					69.2	0.755
	0.25					70.0	0.861
CSP (fusion last)	0.75		✓		✓	69.2	0.716
	0.5					70.6	0.804
	0.25					**70.8**	**0.902**
CSPPeleeNet	0.75		✓		✓	70.4	0.800
	0.5					**70.9**	**0.888**
	0.25					**71.5**	**0.986**

Ablation study of EFM on MS COCO. Next, we conduct an ablation study of EFM and compare three different feature fusion strategies shown in Figure 6. We choose PRN [33] and ThunderNet [25] to make comparison. PRN and ThunderNet with Context Enhancement Module (CEM) and Spatial Attention Module (SAM) are FPN and GFM architectures, respectively. We also design a GFM to compare with the proposed EFM. Moreover, GIoU [29], SPP, and SAM are also applied to EFM. All experiment results listed in Table 2 adopt CSPPeleeNet as the backbone.

Table 2: Ablation study of EFM on MS COCO.

Head	global fusion	exact fusion	atten.	BFLOPs	FPS	AP	AP_{50}	AP_{75}
PRN [33]				3.590	169	23.1	44.5	22.0
PRN-3l [33]				4.586	151	23.7	46.0	22.2
CEM [25]	✓			4.049	148	23.8	45.4	22.6
CEM (SAM) [25]	✓		✓	4.165	144	24.1	46.0	23.1
GFM	✓			4.605	134	24.3	46.2	23.3
EFM		✓		4.868	132	26.4	48.6	26.3
EFM (GIoU [29])		✓		4.868	132	**27.1**	45.9	**28.2**
EFM (SAM)		✓	✓	5.068	129	26.8	**49.0**	26.7
EFM (SPP)		✓		4.863	128	26.2	48.5	25.7

As reflected in the experiment results, the proposed EFM is 2 fps slower than GFM, but its AP_{50} is significantly upgraded by 2.4%. The GIoU can upgrade AP by 0.7%, but the AP_{50} is significantly degraded by 2.7%. For edge computing, what really matters is the number and locations of the objects. Therefore, we will not use GIoU training in the subsequent models. SAM can get a better frame rate and AP than SPP, so we use EFM (SAM) as the final architecture.

4.3. ImageNet Image Classification

We apply the CSPNet to ResNet-10 [8], ResNeXt-50 [37], DenseNet-201 [11], PeleeNet [35], and DenseNet-201-Elastic [34] and compare with state-of-the-art methods. The experimental results are shown in Table 3.

Table 3: Compare with state-of-the-art methods on ImageNet.

Model	#Parameter	BFLOPs	Top-1	Top-5
PeleeNet [35]	2.79M	1.017	70.7%	90.0%
CSPPeleeNet	2.83M	0.888 **(-13%)**	70.9%	90.2%
SparsePeleeNet [44]	2.39M	0.904	69.6%	89.3%
Darknet Reference [26]	7.31M	0.96	61.1%	83.0%
CSPDenseNet Reference	3.48M	0.886	65.7%	86.6%
CSPPeleeNet Reference	4.10M	1.103	**68.9%**	**88.7%**
ResNet-10 [8]	5.24M	2.273	63.5%	85.0%
CSPResNet-10	2.73M	1.905 **(-16%)**	**65.3%**	**86.5%**
ResNeXt-50 [37]	22.19M	10.11	77.8%	**94.2%**
CSPResNeXt-50	20.50M	7.93 **(-22%)**	**77.9%**	94.0%
HarDNet-138s [2]	35.5M	13.4	77.8%	-
DenseNet-264 [11]	27.21M	11.03	77.8%	93.9%
ResNet-152 [8]	60.2M	22.6	77.8%	93.6%
DenseNet-201 [11]	20.01M	8.63	77.4%	93.7%
CSPDenseNet-201	23.07M	8.47**(-2%)**	**77.7%**	**93.6%**
DenseNet-201-Elastic [34]	19.48M	8.77	**77.9%**	**94.0%**
CSPDenseNet-201-Elastic	20.17M	7.13 **(-19%)**	**77.9%**	**94.0%**
Res2NeXt-50 (10 crop) [5]	24.27M	8.4×10	**78.2%**	93.9%
CSPResNeXt-50 (10 crop)	20.50M	7.9×10	**78.2%**	**94.3%**

It is confirmed by experimental results that no matter it is which architecture, when the concept of CSPNet is introduced, the computational load is reduced and the accuracy is either remain unchanged or upgraded, especially useful for the improvement of lightweight models. For example, compared to ResNet-10, CSPResNet-10 can improve accuracy by 1.8%. As to PeleeNet and DenseNet-201-Elastic, CSPPeleeNet and CSPDenseNet-201-Elastic can respectively cut down 13% and 19% computation, and either upgrade a little bit or maintain the accuracy. As to the case of ResNeXt-50, CSPResNeXt-50 can cut down 22% computation and upgrade top-1 accuracy to 77.9%.

Proposed CSPResNeXt-50 is also compared with ResNet-152 [8], DenseNet-264 [11], and HarDNet-138s [2], regardless of #parameter, BFLOPs, and top-1 accuracy, CSPResNeXt-50 all achieve the best result. As to the 10-crop test, CSPResNeXt-50 outperforms Res2NeXt-50 [5].

Table 4: Compare with state-of-the-art methods on MSCOCO Object Detection.

Method	Backbone	Size	FPS	BFLOPs	#Parameter	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv3 [28]	DarkNet53 [28]	608×608	30	140.7	62.3M	33.0	57.9	34.4	18.3	35.4	41.9
YOLOv3 (SPP) [28]	DarkNet53 [28]	608×608	30	141.5	62.9M	36.2	**60.6**	38.2	20.6	37.4	46.1
LRF [36]	ResNet101 [8]	512×512	31	-	-	37.3	58.5	39.7	19.7	**42.8**	50.1
SSD [20]	HarDNet85 [2]	512×512	32	-	-	35.1	54.8	37.6	15.0	38.9	**51.5**
M2Det [42]	VGG16 [31]	320×320	33	-	-	33.5	52.4	35.6	14.4	37.6	47.6
PFPNet (R) [12]	VGG16 [31]	320×320	33	-	-	31.8	52.9	33.6	12.0	35.5	46.1
RFBNet [18]	VGG16 [31]	512×512	35	-	-	33.8	54.2	35.9	16.2	37.1	47.4
PANet (SPP) [19]	**CSPResNeXt50**	608×608	35	100.6	56.9M	**38.4**	**60.6**	**41.6**	**22.1**	41.8	47.6
SSD [20]	HarDNet68 [2]	512×512	38	-	-	31.7	51.0	33.8	12.5	35.1	47.9
LRF [36]	VGG16 [31]	512×512	38	-	-	36.2	56.6	38.7	19.0	39.9	48.8
PFPNet (S) [12]	VGG16 [31]	300×300	39	-	-	29.6	49.6	31.1	10.6	32.0	44.9
RefineDet [39]	VGG16 [31]	320×320	40	-	-	29.4	49.2	31.3	10.0	32.0	44.4
HSD [1]	VGG16 [31]	320×320	40	-	-	33.5	53.2	36.1	15.0	35.0	47.1
SSD [20]	VGG16 [31]	300×300	44	70.4	34.3M	25.7	43.9	26.2	6.9	27.7	42.6
PANet (SPP) [19]	**CSPResNeXt50**	512×512	44	71.3	56.9M	38.0	60.0	40.8	19.7	41.4	49.9
CenterNet [43]	ResNet101 [8]	512×512	45	-	-	34.6	53.0	36.9			
YOLOv3 [28]	DarkNet53 [28]	416×416	46	65.9	62.3M	31.0	55.3	32.3	15.2	33.2	42.8
EFGRNet [24]	VGG16 [31]	320×320	48	-	-	33.2	53.4	35.4	13.4	37.1	47.9
PANet (SPP) [19]	**CSPResNeXt50**	416×416	53	47.1	56.9M	36.6	58.1	39.0	16.2	39.5	50.9
TTFNet [22]	DarkNet53 [28]	512×512	54	-	-	35.1	52.5	37.8	17.0	38.5	49.5
YOLOv3 [28]	DarkNet53 [28]	320×320	56	39.0	62.3M	28.2	51.5	29.7	11.9	30.6	43.4
PANet (SPP) [19]	**CSPResNeXt50**	320×320	58	27.9	56.9M	33.4	54.0	35.1	11.8	35.3	50.9
Pelee [35]	PeleeNet [35]	304×304	106	2.58	5.98M	22.4	38.3	22.9			
EFM (SAM)	**CSPPeleeNet**	512×512	109	7.68	4.31M	27.6	**50.4**	27.7	**12.4**	**30.1**	36.2
TTFNet [22]	ResNet18 [8]	512×512	112	-	-	**28.1**	43.8	**30.2**	11.8	29.5	**41.5**
CenterNet [43]	ResNet18 [8]	512×512	129	-	-	**28.1**	44.9	29.6			
EFM (SAM)	**CSPPeleeNet**	416×416	129	5.07	4.31M	26.8	49.0	26.7	9.8	28.2	38.8
PRN [33]	PeleeNet [35]	416×416	145	4.04	3.16M	23.3	45.0	22.0	6.7	24.8	35.1
EFM (SAM) [33]	**CSPPeleeNet Ref.**	320×320	205	3.43	5.67M	23.5	44.6	22.7	7.1	23.6	36.1
ThunderNet [25]	SNet535 [25]	320×320	214	2.60	-	28.0	46.2	29.5			
EFM (SAM) [33]	**CSPDenseNet Ref.**	320×320	235	2.89	5.05M	21.7	42.2	20.6	6.3	21.3	33.3
ThunderNet [25]	SNet146 [25]	320×320	248	0.95	-	23.6	40.2	24.5			
ThunderNet [25]	SNet49 [25]	320×320	267	0.52	-	19.1	33.7	19.6			
PRN (3l) [33]	**CSPPeleeNet Ref.**	320×320	267	3.15	4.79M	19.4	40.0	17.0	5.8	18.8	31.1
PRN [33]	**CSPPeleeNet Ref.**	320×320	306	2.56	4.59M	18.8	38.5	16.6	5.0	18.9	31.4
YOLOv3 (tiny) [28]	DarkNet Ref. [28]	416×416	330	5.57	8.86M		33.1				
PRN [33]	**CSPDenseNet Ref.**	320×320	387	2.01	3.97M	16.8	35.4	14.3	4.4	16.6	28.5
PRN [33]	DarkNet Ref. [28]	416×416	400	3.47	4.96M		33.1				
PRN [33]	**CSPDenseNetb Ref.**	320×320	400	1.59	1.87M	15.3	34.2	12.0	3.6	16.1	23.4

[1] The table is separated into four parts, <100 fps, 100∼200 fps, 200∼300 fps, and >300 fps.
[2] We mainly focus on **FPS** and **AP_{50}** since almost all applications need fast inference to locate and count objects.
[3] Inference speed are tested on **GTX 1080ti** with batch size equals to 1 if possible, and our models are tested using Darknet [27].
[4] All results are obtained by **COCO test-dev set** except for TTFNet [22] models which are verified on minval5k set.

4.4. MS COCO Object Detection

In the task of object detection, we aim at three targeted scenarios: (1) real-time on GPU: we adopt CSPResNeXt50 with PANet (SPP) [19]; (2) real-time on mobile GPU: we adopt CSPDenseNet-based models with the proposed EFM (SAM); and (3) real-time on CPU: we adop CSPDenseNet-based models with PRN [33]. The comparisons between the above models and the state-of-the-art methods are listed in Table 4. As to the analysis on the inference speed of CPU and mobile GPU will be detailed in the next subsection.

If compared to object detectors running at 30∼100 fps, CSPResNeXt50 with PANet (SPP) achieves the best performance in AP, AP_{50} and AP_{75}. They receive, respectively, 38.4%, 60.6%, and 41.6% detection rates. If compared to LRF [36] under the input image size 512×512, CSPResNeXt50 with PANet (SPP) outperforms ResNet101 with LRF by 0.7% AP, 1.5% AP_{50} and 1.1% AP_{75}. If compared to object detectors running at 100∼200 fps, CSPPeleeNet with EFM (SAM) boosts 12.1% and 4.1% AP_{50} at the same speed as Pelee [35] and CenterNet [43], respectively.

If compared to very fast object detectors such as ThunderNet [25], YOLOv3-tiny [28], and YOLOv3-tiny-PRN [33], the proposed CSPDenseNetb Reference with PRN is the fastest. It can reach 400 fps frame rate, i.e., 133 fps faster than ThunderNet with SNet49. Besides, it gets 0.5% higher on AP_{50}. If compared to ThunderNet146, CSPPeleeNet Reference with PRN (3l) increases the frame rate by 19 fps while maintaining the same level of AP_{50}.

4.5. Analysis

Computational Bottleneck. Figure 7 shows the BLOPS of each layer of PeleeNet-YOLO, PeleeNet-PRN and proposed CSPPeleeNet-EFM. The computational bottleneck of PeleeNet-YOLO occurs when the head integrates the feature pyramid, while the computational bottleneck of PeleeNet-PRN occurs on the transition layers of the PeleeNet backbone. As to the proposed CSPPeleeNet-EFM, it can balance the overall computational bottleneck, which reduces the PeleeNet backbone 44% computational bottleneck and reduces PeleeNet-YOLO 80% computational bottleneck. Therefore, we can say that the proposed CSPNet can provide hardware with a higher utilization rate.

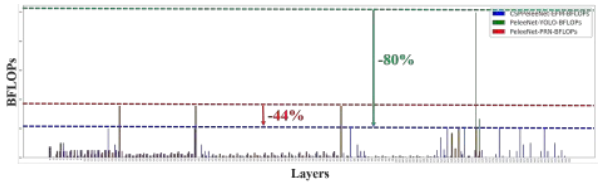

Figure 7: Computational bottleneck of PeleeNet-YOLO, PeleeNet-PRN and CSPPeleeNet-EFM.

Memory Traffic. Figure 8 shows the size of each layer of ResNeXt50 and the proposed CSPResNeXt50. The CIO of the proposed CSPResNeXt (32.6M) is lower than that of the original ResNeXt50 (34.4M). In addition, our CSPResNeXt50 removes the bottleneck layers in the ResXBlock and maintains the same numbers of the input channel and the output channel, which is shown in Ma *et al*. [23] that this will have the lowest MAC and the most efficient computation when FLOPs are fixed. The low CIO and FLOPs enable our CSPResNeXt50 to outperform the vanilla ResNeXt50 by 22% in terms of computations.

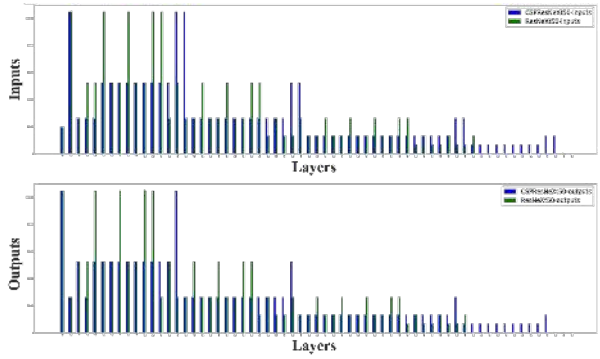

Figure 8: Input size and output size of ResNeXt and CSPResNeXt.

Inference Rate. We further evaluate whether the proposed methods are able to be deployed on real-time detectors with mobile GPU or CPU. Experiments are based on NVIDIA Jetson TX2 and Intel Core i9-9900K. The inference rate on CPU is evaluated with the OpenCV DNN module. We do not adopt model compression or quantization for fair comparisons. The results are shown in Table 5.

Table 5: Inference rate on mobile GPU (mGPU) and CPU real-time object detectors (in fps).

Model	Size	GPU	CPU	mGPU	AP_{50}
SNet146-Thunder [25]	320	248	32	-	40.2
SNet49-Thunder [25]	320	267	47	-	33.7
YOLOv3-tiny [28]	416	330	54	37	33.1
YOLOv3-tiny-PRN [33]	416	**400**	71	49	33.1
CSPPeleeNet Ref.-EFM (SAM)	320	205	-	41	**44.6**
CSPDenseNet Ref.-EFM (SAM)	320	235	-	49	42.2
CSPPeleeNet Ref.-PRN (3l)	320	267	52	38	40.0
CSPPeleeNet Ref.-PRN	320	306	75	52	38.5
CSPDenseNet Ref.-PRN	320	387	95	64	35.4
CSPDenseNetb Ref.-PRN	320	**400**	**102**	**73**	34.2

If we compare the inference speed executed on CPU, CSPDenseNetb Ref.-PRN receives higher AP_{50} than SNet49-TunderNet, and it also outperforms SNet49-TunderNet by 55 fps, in terms of frame rate. On the other hand, CSPPeleeNet Ref.-PRN (3l) reaches the same accuracy level as SNet146-ThunderNet but significantly upgrades the frame rate by 20 fps on CPU.

If we compare the inference speed executed on mobile GPU, our proposed EFM can greatly reduce the memory requirement when generating feature pyramids, it is definitely beneficial to function under the memory bandwidth restricted mobile environment. For example, CSPPeleeNet Ref.-EFM (SAM) can have a higher frame rate than YOLOv3-tiny, and its AP_{50} is 11.5% higher than YOLOv3-tiny. For the same CSPPeleeNet Ref. backbone, although EFM (SAM) is 62 fps slower than PRN (3l) on GTX 1080ti, it reaches 41 fps on Jetson TX2, 3 fps faster than PRN (3l), and with 4.6% AP_{50} increase.

5. Conclusion

We have proposed the CSPNet that enables state-of-the-art methods such as ResNet, ResNeXT, and DenseNet to be light-weighted for mobile GPUs or CPUs. One of the main contributions is that we have recognized the redundant gradient information problem that results in inefficient optimization and costly inference computations. We have proposed to utilize the cross-stage feature fusion strategy and the truncating gradient flow to enhance the variability of the learned features within different layers. In addition, we have proposed the EFM that incorporates the Maxout operation to compress the features maps generated from the feature pyramid, which largely reduces the required memory bandwidth and thus the inference is efficient enough to be compatible with edge computing devices. Experimentally, we have shown that the proposed CSPNet with the EFM significantly outperforms competitors in terms of accuracy and inference rate on mobile GPU and CPU for real-time object detection tasks.

References

[1] Jiale Cao, Yanwei Pang, Jungong Han, and Xuelong Li. Hierarchical shot detector. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 9705–9714, 2019. 7

[2] Ping Chao, Chao-Yang Kao, Yu-Shan Ruan, Chien-Hsiang Huang, and Youn-Long Lin. HarDNet: A low memory traffic network. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. 2, 6, 7

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 2, 5

[4] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. CenterNet: Keypoint triplets for object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 6569–6578, 2019. 3

[5] Shang-Hua Gao, Ming-Ming Cheng, Kai Zhao, Xin-Yu Zhang, Ming-Hsuan Yang, and Philip Torr. Res2Net: A new multi-scale backbone architecture. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020. 2, 6

[6] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. In *Proceedings of International Conference on Machine Learning (ICML)*, 2013. 2

[7] Dongyoon Han, Jiwhan Kim, and Junmo Kim. Deep pyramidal residual networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5927–5935, 2017. 3

[8] 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 (CVPR)*, pages 770–778, 2016. 1, 2, 3, 6, 7

[9] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. 1

[10] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1

[11] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4700–4708, 2017. 1, 2, 3, 6

[12] Seung-Wook Kim, Hyong-Keun Kook, Jee-Young Sun, Mun-Cheon Kang, and Sung-Jea Ko. Parallel feature pyramid network for object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 234–250, 2018. 7

[13] Hei Law and Jia Deng. CornerNet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018. 3

[14] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. CornerNet-Lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019. 3

[15] Kunpeng Li, Ziyan Wu, Kuan-Chuan Peng, Jan Ernst, and Yun Fu. Tell me where to look: Guided attention inference network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 9215–9223, 2018. 5

[16] Zeming Li, Chao Peng, Gang Yu, Xiangyu Zhang, Yangdong Deng, and Jian Sun. Light-Head R-CNN: In defense of two-stage object detector. *arXiv preprint arXiv:1711.07264*, 2017. 3

[17] 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 *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 740–755, 2014. 2, 5

[18] Songtao Liu, Di Huang, et al. Receptive field block net for accurate and fast object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 385–400, 2018. 3, 7

[19] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8759–8768, 2018. 7

[20] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 21–37, 2016. 3, 7

[21] Wei Liu, Andrew Rabinovich, and Alexander C Berg. ParseNet: Looking wider to see better. *International Conference on Learning Representations (ICLR)*, 2016. 5

[22] Zili Liu, Tu Zheng, Guodong Xu, Zheng Yang, Haifeng Liu, and Deng Cai. Training-time-friendly network for real-time object detection. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2020. 7

[23] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNetV2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018. 1, 2, 3, 8

[24] Jing Nie, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Enriched feature guided refinement network for object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 9537–9546, 2019. 7

[25] Zheng Qin, Zeming Li, Zhaoning Zhang, Yiping Bao, Gang Yu, Yuxing Peng, and Jian Sun. ThunderNet: Towards real-time generic object detection. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019. 3, 6, 7, 8

[26] Joseph Redmon. Darknet: Open source neural networks in C. http://pjreddie.com/darknet/, 2013–2016. 6

[27] Joseph Redmon, Alexey Bochkovskiy, and Stefano Sinigardi. Darknet: YOLOv3 - neural network for object detection. https://github.com/AlexeyAB/darknet, 2019. 7

[28] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018. 3, 5, 7, 8

[29] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 658–666, 2019. 6

[30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4510–4520, 2018. 1

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 7

[32] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MNASnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2820–2828, 2019. 1

[33] Chien-Yao Wang, Hong-Yuan Mark Liao, Ping-Yang Chen, and Jun-Wei Hsieh. Enriching variety of layer-wise learning information by gradient combination. *Proceedings of the IEEE International Conference on Computer Vision Workshop (ICCV Workshop)*, 2019. 2, 3, 6, 7, 8

[34] Huiyu Wang, Aniruddha Kembhavi, Ali Farhadi, Alan L Yuille, and Mohammad Rastegari. Elastic: Improving cnns with dynamic scaling policies. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2258–2267, 2019. 2, 6

[35] Robert J Wang, Xiang Li, and Charles X Ling. Pelee: A real-time object detection system on mobile devices. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 1963–1972, 2018. 2, 3, 5, 6, 7

[36] Tiancai Wang, Rao Muhammad Anwer, Hisham Cholakkal, Fahad Shahbaz Khan, Yanwei Pang, and Ling Shao. Learning rich features at high-speed for single-shot object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1971–1980, 2019. 3, 7

[37] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1492–1500, 2017. 1, 2, 3, 6

[38] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2016. 1

[39] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, and Stan Z Li. Single-shot refinement neural network for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4203–4212, 2018. 7

[40] Xiaosong Zhang, Fang Wan, Chang Liu, Rongrong Ji, and Qixiang Ye. FreeAnchor: Learning to match anchors for visual object detection. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 3

[41] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6848–6856, 2018. 1

[42] Qijie Zhao, Tao Sheng, Yongtao Wang, Zhi Tang, Ying Chen, Ling Cai, and Haibin Ling. M2det: A single-shot object detector based on multi-level feature pyramid network. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, volume 33, pages 9259–9266, 2019. 7

[43] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv preprint arXiv:1904.07850*, 2019. 3, 7

[44] Ligeng Zhu, Ruizhi Deng, Michael Maire, Zhiwei Deng, Greg Mori, and Ping Tan. Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 186–201, 2018. 2, 3, 6