M2Det: A Single-Shot Object Detector based on Multi-Level Feature Pyramid Network

Qijie Zhao[1], Tao Sheng[1], Yongtao Wang[1]*, Zhi Tang[1], Ying Chen[2], Ling Cai[2] and Haibin Ling[3]

[1]Institute of Computer Science and Technology, Peking University, Beijing, P.R. China
[2]AI Labs, DAMO Academy, Alibaba Group
[3]Computer and Information Sciences Department, Temple University
{zhaoqijie, shengtao, wyt, tangzhi}@pku.edu.cn,
{cailing.cl, chenying.ailab}@alibaba-inc.com, {hbling}@temple.edu

Abstract

Feature pyramids are widely exploited by both the state-of-the-art one-stage object detectors (*e.g.*, DSSD, RetinaNet, RefineDet) and the two-stage object detectors (*e.g.*, Mask R-CNN, DetNet) to alleviate the problem arising from scale variation across object instances. Although these object detectors with feature pyramids achieve encouraging results, they have some limitations due to that they only simply construct the feature pyramid according to the inherent multi-scale, pyramidal architecture of the backbones which are originally designed for object classification task. Newly, in this work, we present Multi-Level Feature Pyramid Network (MLFPN) to construct more effective feature pyramids for detecting objects of different scales. First, we fuse multi-level features (*i.e.* multiple layers) extracted by backbone as the base feature. Second, we feed the base feature into a block of alternating joint Thinned U-shape Modules and Feature Fusion Modules and exploit the decoder layers of each U-shape module as the features for detecting objects. Finally, we gather up the decoder layers with equivalent scales (sizes) to develop a feature pyramid for object detection, in which every feature map consists of the layers (features) from multiple levels. To evaluate the effectiveness of the proposed MLFPN, we design and train a powerful end-to-end one-stage object detector we call M2Det by integrating it into the architecture of SSD, and achieve better detection performance than state-of-the-art one-stage detectors. Specifically, on MS-COCO benchmark, M2Det achieves AP of 41.0 at speed of 11.8 FPS with single-scale inference strategy and AP of 44.2 with multi-scale inference strategy, which are the new state-of-the-art results among one-stage detectors. The code will be made available on https://github.com/qijiezhao/M2Det.

Introduction

Scale variation across object instances is one of the major challenges for the object detection task (Lin et al. 2017a; He et al. 2015; Singh and Davis 2018), and usually there are two strategies to solve the problem arising from this challenge. The first one is to detect objects in an image pyramid (*i.e.* a series of resized copies of the input image) (Singh and Davis 2018), which can only be exploited at the testing time.
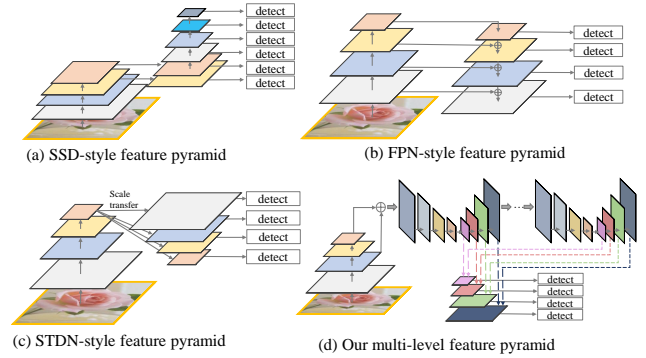
---

*Corresponding author.

Figure 1: Illustrations of four kinds of feature pyramids.

Obviously, this solution will greatly increase memory and computational complexity, thus the efficiency of such object detectors drop dramatically. The second one is to detect objects in a feature pyramid extracted from the input image (Liu et al. 2016; Lin et al. 2017a), which can be exploited at both training and testing phases. Compared with the first solution that uses an image pyramid, it has less memory and computational cost. Moreover, the feature pyramid constructing module can be easily integrated into the state-of-the-art deep neural networks based detectors, yielding an end-to-end solution.

Although the object detectors with feature pyramids (Liu et al. 2016; Lin et al. 2017a; Lin et al. 2017b; He et al. 2017) achieve encouraging results, they still have some limitations due to that they simply construct the feature pyramid according to the inherent multi-scale, pyramidal architecture of the backbones which are actually designed for object classification task. For example, as illustrated in Fig. 1, SSD (Liu et al. 2016) directly and independently uses two layers of the backbone (*i.e.* VGG16) and four extra layers obtained by stride 2 convolution to construct the feature pyramid; STDN (Zhou et al. 2018) only uses the last dense block of DenseNet (Huang et al. 2017) to construct feature pyramid by pooling and scale-transfer operations; FPN (Lin et al. 2017a) constructs the feature pyramid by fusing the deep and shallow layers in a top-down manner. Generally speaking, the above-mentioned methods have the two following limitations. First, feature maps in the pyramid are

not representative enough for the object detection task, instead they are simply constructed from the layers (features) of the backbone designed for object classification task. Second, each feature map in the pyramid (used for detecting objects in a specific range of size) is mainly or even solely constructed from single-level layers of the backbone, that is, it mainly or only contains single-level information. In general, high-level features in the deeper layers are more discriminative for classification subtask while low-level features in the shallower layers can be helpful for object location regression sub-task. Moreover, low-level features are more suitable to characterize objects with simple appearances while high-level features are appropriate for objects with complex appearances. In practice, the appearances of the object instances with similar size can be quite different. For example, a traffic light and a faraway person may have comparable size, and the appearance of the person is much more complex. Hence, each feature map (used for detecting objects in a specific range of size) in the pyramid mainly or only consists of single-level features will result in suboptimal detection performance.

The goal of this paper is to construct a more effective feature pyramid for detecting objects of different scales, while avoid the limitations of the existing methods as above mentioned. As shown in Fig. 2, to achieve this goal, we first fuse multi-level features (*i.e.* multiple layers) extracted by backbone as base feature, and then feed it into a block of alternating joint Thinned U-shape Modules(TUM) and Feature Fusion Modules(FFM) to extract more representative, multi-level multi-scale features. It is worth noting that, decoder layers in each U-shape Module share a similar depth. Finally, we gather up the feature maps with equivalent scales to construct the final feature pyramid for object detection. Obviously, decoder layers that form the final feature pyramid are much deeper than the layers in the backbone, namely, they are more representative. Moreover, each feature map in the final feature pyramid consists of the decoder layers from multiple levels. Hence, we call our feature pyramid block Multi-Level Feature Pyramid Network (MLFPN).

To evaluate the effectiveness of the proposed MLFPN, we design and train a powerful end-to-end one-stage object detector we call M2Det (according to that it is built upon multi-level and multi-scale features) by integrating MLFPN into the architecture of SSD (Liu et al. 2016). M2Det achieves the new state-of-the-art result (*i.e.* AP of 41.0 at speed of 11.8 FPS with single-scale inference strategy and AP of 44.2 with multi-scale inference strategy), outperforming the one-stage detectors on MS-COCO (Lin et al. 2014) benchmark.

Related Work

Researchers have put plenty of efforts into improving the detection accuracy of objects with various scales – no matter what kind of detector it is, either an one-stage detector or a two-stage one. To the best of our knowledge, there are mainly two strategies to tackle this scale-variation problem.

The first one is **featurizing image pyramids** (*i.e.* a series of resized copies of the input image) to produce semantically representative multi-scale features. Features from images of different scales yield predictions separately and these predictions work together to give the final prediction. In terms of recognition accuracy and localization precision, features from various-sized images do surpass features that are based merely on single-scale images. Methods such as (Shrivastava et al. 2016) and SNIP (Singh and Davis 2018) employed this tactic. Despite the performance gain, such a strategy could be costly time-wise and memory-wise, which forbid its application in real-time tasks. Considering this major drawback, methods such as SNIP (Singh and Davis 2018) can choose to only employ featurized image pyramids during the test phase as a fallback, whereas other methods including Fast R-CNN (Girshick 2015) and Faster R-CNN (Ren et al. 2015) chose not to use this strategy by default.

The second one is detecting objects in the **feature pyramid** extracted from inherent layers within the network while merely taking a single-scale image. This strategy demands significantly less additional memory and computational cost than the first one, enabling deployment during both the training and test phases in real-time networks. Moreover, the feature pyramid constructing module can be easily revised and fit into state-of-the-art deep neural networks based detectors. MS-CNN (Cai et al. 2016), SSD (Liu et al. 2016), DSSD (Fu et al. 2017), FPN (Lin et al. 2017a), YOLOv3 (Redmon and Farhadi 2018), RetinaNet (Lin et al. 2017b), and RefineDet (Zhang et al. 2018) adopted this tactic in different ways.

To the best of our knowledge, MS-CNN (Cai et al. 2016) proposed two sub-networks and first incorporated multi-scale features into deep convolutional neural networks for object detection. The proposal sub-net exploited feature maps of several resolutions to detect multi-scale objects in an image. SSD (Liu et al. 2016) exploited feature maps from the later layers of VGG16 base-net and extra feature layers for predictions at multiple scales. FPN (Lin et al. 2017a) utilized lateral connections and a top-down pathway to produce a feature pyramid and achieved more powerful representations. DSSD (Fu et al. 2017) implemented deconvolution layers for aggregating context and enhancing the high-level semantics for shallow features. RefineDet (Zhang et al. 2018) adopted two-step cascade regression, which achieves a remarkable progress on accuracy while keeping the efficiency of SSD.

Proposed Method

The overall architecture of M2Det is shown in Fig. 2. M2Det uses the backbone and the Multi-Level Feature Pyramid Network (MLFPN) to extract features from the input image, and then similar to SSD, produces dense bounding boxes and category scores based on the learned features, followed by the non-maximum suppression (NMS) operation to produce the final results. MLFPN consists of three modules, *i.e.* Feature Fusion Module (FFM), Thinned U-shape Module (TUM) and Scale-wise Feature Aggregation Module (SFAM). FFMv1 enriches semantic information into base features by fusing feature maps of the backbone. Each TUM generates a group of multi-scale features, and then the alternating joint TUMs and FFMv2s extract multi-level multi-scale features. In addition, SFAM aggregates the features into the multi-level feature pyramid through a scale-wise feature concatenation operation and an adaptive attention
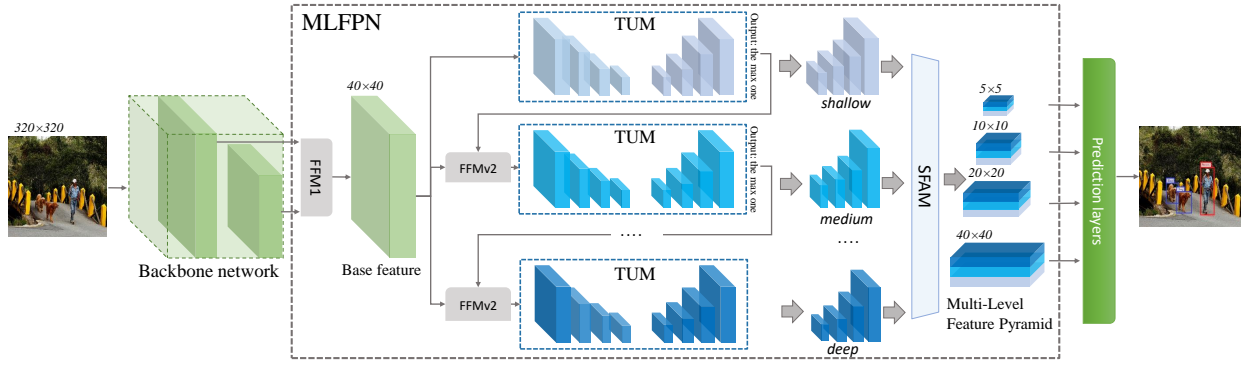
Figure 2: An overview of the proposed M2Det(320×320). M2Det utilizes the backbone and the Multi-level Feature Pyramid Network (MLFPN) to extract features from the input image, and then produces dense bounding boxes and category scores. In MLFPN, FFMv1 fuses feature maps of the backbone to generate the base feature. Each TUM generates a group of multi-scale features, and then the alternating joint TUMs and FFMv2s extract multi-level multi-scale features. Finally, SFAM aggregates the features into a multi-level feature pyramid. In practice, we use 6 scales and 8 levels mostly.
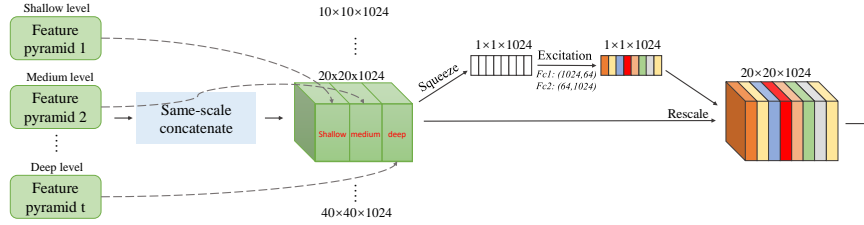

Figure 3: Illustration of Scale-wise Feature Aggregation Module. The first stage of SFAM is to concatenate features with equivalent scales along channel dimension. Then the second stage uses SE attention to aggregate features in an adaptive way.
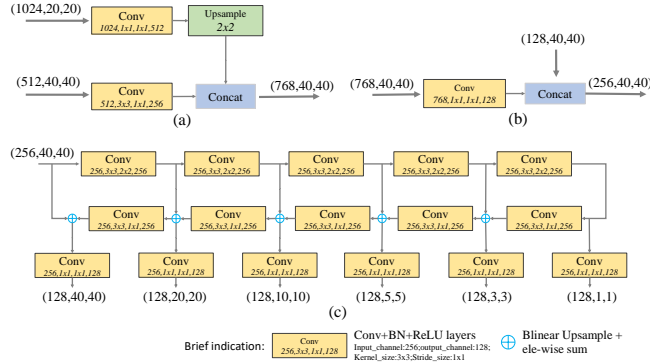

Figure 4: Structural details of some modules. (a) FFMv1, (b) FFMv2, (c) TUM. The inside numbers of each block denote: input channels, Conv kernel size, stride size, output channels.

mechanism. More details about the three core modules and network configurations in M2Det are introduced in the following.

Multi-level Feature Pyramid Network

As shown in Fig. 2, MLFPN contains three parts. Firstly, FFMv1 fuses shallow and deep features to produce the base feature, *e.g.*, conv4_3 and conv5_3 of VGG (Simonyan and Zisserman 2015), which provide multi-level semantic infor-

mation for MLFPN. Secondly, several TUMs and FFMv2 are stacked alternately. Specially, each TUM generates several feature maps with different scales. The FFMv2 fuses the base feature and the largest output feature map of the previous TUM. And the fused feature maps are fed to the next TUM. Note that the first TUM has no prior knowledge of any other TUMs, so it only learns from $\mathbf{X}_{base}$. The output multi-level multi-scale features are calculated as:

$$[\mathbf{x}_1^l, \mathbf{x}_2^l, ..., \mathbf{x}_i^l] = \begin{cases} \mathbf{T}_l(\mathbf{X}_{base}), & l = 1 \\ \mathbf{T}_l(\mathbf{F}(\mathbf{X}_{base}, \mathbf{x}_i^{l-1})), & l = 2...L \end{cases}, \quad (1)$$

where $\mathbf{X}_{base}$ denotes the base feature, x_i^l denotes the feature with the i-th scale in the l-th TUM, L denotes the number of TUMs, $\mathbf{T}_l$ denotes the l-th TUM processing, and $\mathbf{F}$ denotes FFMv1 processing. Thirdly, SFAM aggregates the multi-level multi-scale features by a scale-wise feature concatenation operation and a channel-wise attention mechanism.

FFMs FFMs fuse features from different levels in M2Det, which are crucial to constructing the final multi-level feature pyramid. They use 1x1 convolution layers to compress the channels of the input features and use concatenation operation to aggregate these feature maps. Especially, since FFMv1 takes two feature maps with different scales in backbone as input, it adopts one upsample operation to rescale the deep features to the same scale before the concatenation operation. Meanwhile, FFMv2 takes the base feature and the largest output feature map of the previous TUM –

these two are of the same scale – as input, and produces the fused feature for the next TUM. Structural details of FFMv1 and FFMv2 are shown in Fig. 4 (a) and (b), respectively.

TUMs Different from FPN (Lin et al. 2017a) and RetinaNet (Lin et al. 2017b), TUM adopts a thinner U-shape structure as illustrated in Fig. 4 (c). The encoder is a series of 3x3 convolution layers with stride 2. And the decoder takes the outputs of these layers as its reference set of feature maps, while the original FPN chooses the output of the last layer of each stage in ResNet backbone. In addition, we add 1x1 convolution layers after upsample and element-wise sum operation at the decoder branch to enhance learning ability and keep smoothness for the features (Lin, Chen, and Yan 2014). All of the outputs in the decoder of each TUM form the multi-scale features of the current level. As a whole, the outputs of stacked TUMs form the multi-level multi-scale features, while the front TUM mainly provides shallow-level features, the middle TUM provides medium-level features, and the back TUM provides deep-level features.

SFAM SFAM aims to aggregate the multi-level multi-scale features generated by TUMs into a multi-level feature pyramid as shown in Fig. 3. The first stage of SFAM is to concatenate features of the equivalent scale together along the channel dimension. The aggregated feature pyramid can be presented as $\mathbf{X} = [\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_i]$, where $\mathbf{X}_i = Concat(\mathbf{x}_i^1, \mathbf{x}_i^2, ..., \mathbf{x}_i^L) \in \mathbb{R}^{W_i \times H_i \times C}$ refers to the features of the i-th largest scale. Here, each scale in the aggregated pyramid contains features from multi-level depths. However, simple concatenation operations are not adaptive enough. In the second stage, we introduce a channel-wise attention module to encourage features to focus on channels that they benefit most. Following SE block (Hu, Shen, and Sun 2017), we use global average pooling to generate channel-wise statistics $\mathbf{z} \in \mathbb{R}^C$ at the squeeze step. And to fully capture channel-wise dependencies, the following excitation step learns the attention mechanism via two fully connected layers:

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(\mathbf{W}_2 \delta(\mathbf{W}_1 \mathbf{z})), \qquad (2)$$

where σ refers to the ReLU function, δ refers to the sigmoid function, $\mathbf{W}_1 \in \mathbb{R}^{\frac{C}{r} \times C}$, $\mathbf{W}_2 \in \mathbb{R}^{C \times \frac{C}{r}}$, r is the reduction ratio ($r = 16$ in our experiments). The final output is obtained by reweighting the input $\mathbf{X}$ with activation $\mathbf{s}$:

$$\tilde{\mathbf{X}}_i^c = \mathbf{F}_{scale}(\mathbf{X}_i^c, s_c) = s_c \cdot \mathbf{X}_i^c, \qquad (3)$$

where $\tilde{\mathbf{X}}_i = [\tilde{\mathbf{X}}_i^1, \tilde{\mathbf{X}}_i^2, ..., \tilde{\mathbf{X}}_i^C]$, each of the features is enhanced or weakened by the rescaling operation.

Network Configurations

We assemble M2Det with two kinds of backbones (Simonyan and Zisserman 2015; He et al. 2016). Before training the whole network, the backbones need to be pre-trained on the ImageNet 2012 dataset (Russakovsky et al. 2015). All of the default configurations of MLFPN contain 8 TUMs, each TUM has 5 striding-Convs and 5 Upsample operations, so it will output features with 6 scales. To reduce the number of parameters, we only allocate 256 channels to each scale

of their TUM features, so that the network could be easy to train on GPUs. As for input size, we follow the original SSD, RefineDet and RetinaNet, *i.e.,* 320, 512 and 800.

At the detection stage, we add two convolution layers to each of the 6 pyramidal features to achieve location regression and classification respectively. The detection scale ranges of the default boxes of the six feature maps follow the setting of the original SSD. And when input size is 800×800, the scale ranges increase proportionally except keeping the minimum size of the largest feature map. At each pixel of the pyramidal features, we set six anchors with three ratios entirely. Afterward, we use a probability score of 0.05 as threshold to filter out most anchors with low scores. Then we use soft-NMS (Bodla et al. 2017) with a linear kernel for post-processing, leaving more accurate boxes. Decreasing the threshold to 0.01 can generate better detection results, but it will slow down the inference time a lot, we do not consider it for pursuing better practical values.

Experiments

In this section, we present experimental results on the bounding box detection task of the challenging MS-COCO benchmark. Following the protocol in MS-COCO, we use the `trainval35k` set for training, which is a union of 80k images from `train` split and a random 35 subset of images from the 40k image `val` split. To compare with state-of-the-art methods, we report COCO AP on the `test-dev` split, which has no public labels and requires the use of the evaluation server. And then, we report the results of ablation studies evaluated on the `minival` split for convenience.

Our experiment section includes 4 parts: (1) introducing implement details about the experiments; (2) demonstrating the comparisons with state-of-the-art approaches; (3) ablation studies about M2Det; (4) comparing different settings about the internal structure of MLFPN and introducing several version of M2Det.

Implementation details

For all experiments based on M2Det, we start training with warm-up strategy for 5 epochs, initialize the learning rate as 2×10^{-3}, and then decrease it to 2×10^{-4} and 2×10^{-5} at 90 epochs and 120 epochs, and stop at 150 epochs. M2Det is developed with PyTorch v0.4.0 [1]. When input size is 320 and 512, we conduct experiments on a machine with 4 NVIDIA Titan X GPUs, CUDA 9.2 and cuDNN 7.1.4, while for input size of 800, we train the network on NVIDIA Tesla V100 to get results faster. The batch size is set to 32 (16 each for 2 GPUs, or 8 each for 4 GPUs). On NVIDIA Titan Xp that has 12 GB memory, the training performance is limited if batch size on a single GPU is less than 5. Notably, for Resnet101, M2Det with the input size of 512 is not only limited in the batch size (only 4 is available), but also takes a long time to train, so we train it on V100.

For training M2Det with the VGG-16 backbone when input size is 320×320 and 512×512 on 4 Titan X devices, the total training time costs 3 and 6 days respectively, and with the ResNet-101 backbone when 320×320 costs 5 days.

---

[1] https://pytorch.org/

Method	Backbone	Input size	MultiScale	FPS	Avg. Precision, IoU:			Avg. Precision, Area:		
					0.5:0.95	0.5	0.75	S	M	L
two-stage:										
Faster R-CNN (Ren et al. 2015)	VGG-16	~1000×600	False	7.0	21.9	42.7	-	-	-	-
OHEM++ (Shrivastava et al. 2016)	VGG-16	~1000×600	False	7.0	25.5	45.9	26.1	7.4	27.7	40.3
R-FCN (Dai et al. 2016)	ResNet-101	~1000×600	False	9	29.9	51.9	-	10.8	32.8	45.0
CoupleNet (Zhu et al. 2017)	ResNet-101	~1000×600	False	8.2	34.4	54.8	37.2	13.4	38.1	50.8
Faster R-CNN w FPN (Lin et al. 2017a)	Res101-FPN	~1000×600	False	6	36.2	59.1	39.0	18.2	39.0	48.2
Deformable R-FCN (Dai et al. 2017)	Inc-Res-v2	~1000×600	False	-	37.5	58.0	40.8	19.4	40.1	52.5
Mask R-CNN (He et al. 2017)	ResNeXt-101	~1280×800	False	3.3	39.8	62.3	43.4	22.1	43.2	51.2
Fitness-NMS (Tychsen-Smith and Petersson 2018)	ResNet-101	~1024×1024	True	5.0	41.8	60.9	44.9	21.5	45.0	57.5
Cascade R-CNN (Cai and Vasconcelos 2018)	Res101-FPN	~1280×800	False	7.1	42.8	62.1	46.3	23.7	45.5	55.2
SNIP (Singh and Davis 2018)	DPN-98	-	True	-	45.7	67.3	51.1	29.3	48.8	57.1
one-stage:										
SSD300* (Liu et al. 2016)	VGG-16	300×300	False	43	25.1	43.1	25.8	6.6	25.9	41.4
RON384++ (Kong et al. 2017)	VGG-16	384×384	False	15	27.4	49.5	27.1	-	-	-
DSSD321 (Fu et al. 2017)	ResNet-101	321×321	False	9.5	28.0	46.1	29.2	7.4	28.1	47.6
RetinaNet400 (Lin et al. 2017b)	ResNet-101	~640×400	False	12.3	31.9	49.5	34.1	11.6	35.8	48.5
RefineDet320 (Zhang et al. 2018)	VGG-16	320×320	False	38.7	29.4	49.2	31.3	10.0	32.0	44.4
RefineDet320 (Zhang et al. 2018)	ResNet-101	320×320	True	-	38.6	59.9	41.7	21.1	41.7	52.3
M2Det (Ours)	VGG-16	320×320	False	33.4	33.5	52.4	35.6	14.4	37.6	47.6
M2Det (Ours)	VGG-16	320×320	True	-	38.9	59.1	42.4	24.4	41.5	47.6
M2Det (Ours)	ResNet-101	320×320	False	21.7	34.3	53.5	36.5	14.8	38.8	47.9
M2Det (Ours)	ResNet-101	320×320	True	-	**39.7**	**60.0**	**43.3**	**25.3**	**42.5**	**48.3**
YOLOv3 (Redmon and Farhadi 2018)	DarkNet-53	608×608	False	19.8	33.0	57.9	34.4	18.3	35.4	41.9
SSD512* (Liu et al. 2016)	VGG-16	512×512	False	22	28.8	48.5	30.3	10.9	31.8	43.5
DSSD513 (Fu et al. 2017)	ResNet-101	513×513	False	5.5	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet500 (Lin et al. 2017b)	ResNet-101	~832×500	False	11.1	34.4	53.1	36.8	14.7	38.5	49.1
RefineDet512 (Zhang et al. 2018)	VGG-16	512×512	False	22.3	33.0	54.5	35.5	16.3	36.3	44.3
RefineDet512 (Zhang et al. 2018)	ResNet-101	512×512	True	-	41.8	62.9	45.7	25.6	45.1	54.1
CornerNet (Law and Deng 2018)	Hourglass	512×512	False	4.4	40.5	57.8	45.3	20.8	44.8	56.7
CornerNet (Law and Deng 2018)	Hourglass	512×512	True	-	42.1	57.8	45.3	20.8	44.8	56.7
M2Det (Ours)	VGG-16	512×512	False	18.0	37.6	56.6	40.5	18.4	43.4	51.2
M2Det (Ours)	VGG-16	512×512	True	-	42.9	62.5	47.2	28.0	47.4	52.8
M2Det (Ours)	ResNet-101	512×512	False	15.8	38.8	59.4	41.7	20.5	43.9	53.4
M2Det (Ours)	ResNet-101	512×512	True	-	**43.9**	**64.4**	**48.0**	**29.6**	**49.6**	**54.3**
RetinaNet800 (Lin et al. 2017b)	Res101-FPN	~1280×800	False	5.0	39.1	59.1	42.3	21.8	42.7	50.2
M2Det (Ours)	VGG-16	800×800	False	11.8	41.0	59.7	45.0	22.1	46.5	53.8
M2Det (Ours)	VGG-16	800×800	True	-	**44.2**	**64.6**	**49.3**	**29.2**	**47.9**	**55.1**

Table 1: Detection accuracy comparisons in terms of mAP percentage on *MS COCO* test-dev set.

While for training M2Det with ResNet-101 when input size is 512×512 on 2 V100 devices, it costs 11 days. The most accurate model is M2Det with the VGG backbone and 800×800 input size, it costs 14 days.

Comparison with State-of-the-art

We compare the experimental results of the proposed M2Det with state-of-the-art detectors in Table 1. For these experiments, we use 8 TUMs and set 256 channels for each TUM. The main information involved in the comparison includes the input size of the model, the test method (whether it uses multi-scale strategy), the speed of the model, and the test results. Test results of M2Det with 10 different setting versions are reported in Table 1, which are produced by testing it on MS-COCO `test-dev` split, with a single NVIDIA Titan X PASCAL and the batch size 1. Other statistical results stem from references. It is noteworthy that, M2Det-320 with VGG backbone achieves AP of 38.9, which has surpassed most object detectors with more powerful backbones and larger input size, *e.g.*, AP of Deformable R-FCN (Dai et al. 2017) is 37.5, AP of Faster R-CNN with FPN is 36.2. Assembled with ResNet-101 can further improve M2Det, the single-scale version obtains AP of 38.8, which is competitive with state-of-the-art two-stage detectors Mask R-CNN (He et al. 2017). In addition, based on the optimization of PyTorch, it can run at 15.8 FPS. RefineDet (Zhang et al. 2018) inherits the merits of one-stage detectors and two-stage detectors, gets AP of 41.8, CornerNet (Law and Deng 2018) proposes key point regression for detection and borrows the advantages of Hourglass (Newell, Yang, and Deng 2016) and focal loss (Lin et al. 2017b), thus gets AP of 42.1. In contrast, our proposed M2Det is based on the regression method of original SSD, with the assistance of Multi-scale Multi-level features, obtains 44.2 AP, which exceeds all one-stage detectors. Most approaches do not compare the speed of multi-scale inference strategy due to different methods or tools used, so we also only focus on the speed of single-scale inference methods.

In addition, in order to emphasize that the improvement of M2Det is not entirely caused by the deepened depth of the model or the gained parameters, we compare with state-of-the-art one-stage detectors and two-stage detectors. CornerNet with Hourglass has 201M parameters, Mask R-CNN (He et al. 2017) with ResNeXt-101-32x8d-FPN (Xie et al. 2017) has 205M parameters. By contrast, M2Det800-VGG has only 147M parameters. Besides, consider the comparison of depth, it is also not dominant.

Ablation study

Since M2Det is composed of multiple subcomponents, we need to verify each of its effectiveness to the final performance. The baseline is a simple detector based on the original SSD, with 320×320 input size and VGG-16 reduced backbone.

+ 1 s-TUM		✓					
+ 8 s-TUM			✓				
+ 8 TUM				✓	✓	✓	✓
+ Base feature					✓	✓	✓
+ SFAM						✓	✓
VGG16 ⇒ Res101							✓
AP	25.8	27.5	30.6	30.8	32.7	33.2	**34.1**
AP_{50}	44.7	45.2	50.0	50.3	51.9	52.2	**53.7**
AP_{small}	7.2	7.7	13.8	13.7	13.9	15.0	**15.9**
AP_{medium}	27.4	28.0	35.3	35.3	37.9	38.2	**39.5**
AP_{large}	41.4	47.0	44.5	44.8	48.8	49.1	**49.3**

Table 2: Ablation study of M2Det. The detection results are evaluated on `minival` set

TUM To demonstrate the effectiveness of TUM, we conduct three experiments. First, following DSSD, we extend the baseline detector with a series of Deconv layers, and the AP has improved from 25.8 to 27.5 as illustrated in the third column in Table 2. Then we replace with MLFPN into it. As for the U-shape module, we firstly stack 8 s-TUMs, which is modified to decrease the 1×1 Convolution layers shown in Fig. 4, then the performance has improved 3.1 compared with the last operation, shown in the forth column in Table 2. Finally, replacing TUM by s-TUM in the fifth column has reached the best performance, it comes to AP of 30.8.

Base feature Although stacking TUMs can improve detection, but it is limited by input channels of the first TUM. That is, decreasing the channels will drop the abstraction of MLFPN, while increasing them will highly increase the parameters number. Instead of using base feature only once, We afferent base feature at the input of each TUM to alleviate the problem. For each TUM, the embedded base feature provides necessary localization information since it contains shallow features. The AP percentage increases to 32.7, as shown in the sixth column in Table 2.

SFAM As shown in the seventh column in Table 2, compared with the architecture that without SFAM, all evaluation metrics have been upgraded. Specifically, all boxes including small, medium and large become more accurate.

Backbone feature As in many visual tasks, we observe a noticeable AP gain from 33.2 to 34.1 when we use well-

tested ResNet-101 (He et al. 2016) instead of VGG-16 as the backbone network. As shown in Table 2, such observation remains true and consistent with other AP metrics.

TUMs	Channels	Params(M)	AP	AP_{50}	AP_{75}
2	256	40.1	30.5	50.5	32.0
2	512	106.5	32.1	51.8	34.0
4	128	34.2	29.8	49.7	31.2
4	256	60.2	31.8	51.4	33.0
4	512	192.2	33.4	52.6	34.2
8	128	47.5	31.8	50.6	33.6
8	256	98.9	33.2	52.2	35.2
8	512	368.8	34.0	52.9	36.4
16	128	73.9	32.5	51.7	34.4
16	256	176.8	33.6	52.6	35.7

Table 3: Different configurations of MLFPN in M2Det. The backbone is VGG and input image is 320×320.

Variants of MLFPN

The Multi-scale Multi-level Features have been proved to be effective. But what is the boundary of the improvement brought by MLFPN? Step forward, how to design TUM and how many TUMs should be OK? We implement a group of variants to find the regular patterns. To be more specific, we fix the backbone as VGG-16 and the input image size as 320x320, and then tune the number of TUMs and the number of internal channels of each TUM.

As shown in Table 3, M2Det with different configurations of TUMs is evaluated on COCO `minival` set. Comparing the number of TUMs when fixing the channels, *e.g.*,256, it can be concluded that stacking more TUMs brings more promotion in terms of detection accuracy. Then fixing the number of TUMs, no matter how many TUMs are assembled, more channels consistently benefit the results. Furthermore, assuming that we take a version with 2 TUMS and 128 channels as the baseline, using more TUMs could bring larger improvement compared with increasing the internal channels, while the increase in parameters remains similar.

Speed

We compare the inference speed of M2Det with state-of-the-art approaches. Since VGG-16 (Simonyan and Zisserman 2015) reduced backbone has removed FC layers, it is very fast to use it for extracting base feature. We set the batch size to 1, take the sum of the CNN time and NMS time of 1000 images, and divide by 1000 to get the inference time of a single image. We assemble VGG-16 to M2Det and propose the fast version M2Det with the input size 320×320, the standard version M2Det with 512×512 input size and the most accurate version M2Det with 800×800 input size. Based on the optimization of PyTorch, M2Det can achieve accurate results with high speed. As shown in Fig. 5, M2Det benefits the advantage of one-stage detection and our proposed MLFPN structure, draws a significantly better speed-accuracy curve compared with other methods. For fair comparison, we reproduce and test the speed of SSD321-ResNet101, SSD513-ResNet101 (Fu et al.
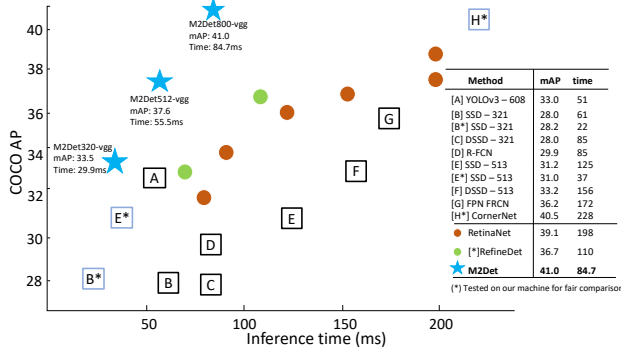
Method	mAP	time
[A] YOLOv3 − 608	33.0	51
[B] SSD − 321	28.0	61
[B*] SSD − 321	28.2	22
[C] DSSD − 321	28.0	85
[D] R-FCN	29.9	85
[E] SSD − 513	31.2	125
[E*] SSD − 513	31.0	37
[F] DSSD − 513	33.2	156
[G] FPN FRCN	36.2	172
[H*] CornerNet	40.5	228
RetinaNet	39.1	198
[*]RefineDet	36.7	110
M2Det	41.0	84.7

(*) Tested on our machine for fair comparison

Figure 5: Speed (ms) vs. accuracy (mAP) on COCO *test-dev*.

2017), RefineDet512-ResNet101, RefineDet320-ResNet101 (Zhang et al. 2018) and CornerNet (Law and Deng 2018) on our device. It is clear that M2Det performs more accurately and efficiently. In addition, replacing Soft-NMS with Hard-NMS, the M2Det-VGG-800 can even achieve a speed of 20 fps, only sacrifice little accuracy.

Discussion

We think the detection accuracy improvement of M2Det is mainly brought by the proposed MLFPN. On one hand, we fuse multi-level features extracted by backbone as the base feature, and then feed it into a block of alternating joint Thinned U-shape Modules and Feature Fusion Modules to extract more representative, multi-level multi-scale features, *i.e.* the decoder layers of each TUM. Obviously, these decoder layers are much deeper than the layers in the backbone, and thus more representative for object detection. Contrasted with our method, the existing detectors (Zhang et al. 2018; Lin et al. 2017a; Fu et al. 2017) just use the layers of the backbone or extra layers with few depth increase. Hence, our method can achieve superior detection performance. On the other hand, each feature map of the multi-level feature pyramid generated by the SFAM consists of the decoder layers from multiple levels. In particular, at each scale, we use multi-level features to detect objects, which would be better for **handling complex appearance variation across object instances**.

To verify that the proposed MLFPN can learn effective feature for detecting objects with different scales and complex appearances, we visualize the activation values of classification Conv layers along scale and level dimensions, such an example shown in Fig. 6. The input image contains two persons, two cars and a traffic light. Moreover, the sizes of the two persons are different, as well as the two cars. And the traffic light, the smaller person and the smaller car have similar sizes. We can find that: 1) compared with the smaller person, the larger person has strongest activation value at the feature map of large scale, so as to the smaller car and larger car; 2) the traffic light, the smaller person and the smaller car have strongest activation value at the feature maps of the same scale; 3) the persons, the cars and the traffic light have strongest activation value at the highest-level, middle-level, lowest-level feature maps respectively. This example
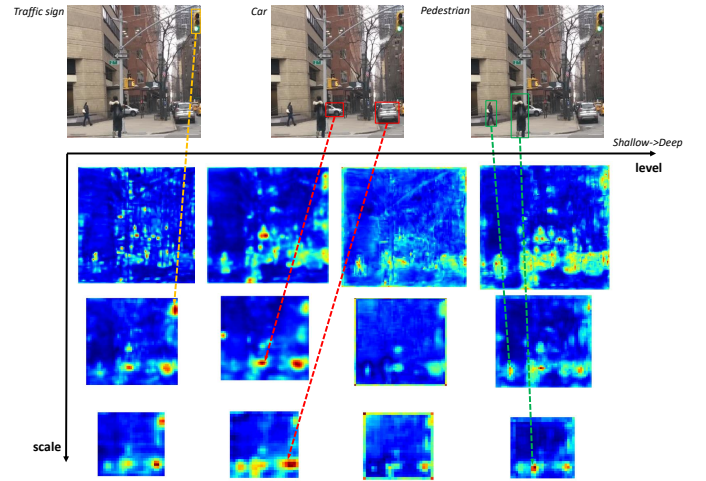

Figure 6: Example activation values of multi-scale multi-level features. Best view in color.

presents that: 1) our method learns very effective features to handle scale variation and complex appearance variation across the object instances; 2) it is necessary to use multi-level features to detect objects with similar size.

Conclusion

In this work, a novel method called Multi-Level Feature Pyramid Network (MLFPN) is proposed to construct effective feature pyramids for detecting objects of different scales. MLFPN consists of several novel modules. First, multi-level features (*i.e.* multiple layers) extracted by backbone are fused by a Feature Fusion Module (FFMv1) as the base feature. Second, the base feature is fed into a block of alternating joint Thinned U-shape Modules (TUMs) and Feature Fusion Modules (FFMv2s) and multi-level multi-scale features (*i.e.* the decoder layers of each TUM) are extracted. Finally, the extracted multi-level multi-scale features with the same scale (size) are aggregated to construct a feature pyramid for object detection by a Scale-wise Feature Aggregation Module (SFAM). A powerful end-to-end one-stage object detector called M2Det is designed based on the proposed MLFPN, which achieves a new state-of-the-art result (*i.e.* AP of 41.0 at speed of 11.8 FPS with single-scale inference strategy and AP of 44.2 with multi-scale inference strategy) among the one-stage detectors on MS-COCO benchmark. Additional ablation studies further demonstrate the effectiveness of the proposed architecture and the novel modules.

Acknowledgements

References

[Bodla et al. 2017] Bodla, N.; Singh, B.; Chellappa, R.; and Davis, L. S. 2017. Soft-nms - improving object detection with one line of code. In *ICCV 2017*, 5562–5570.

[Cai and Vasconcelos 2018] Cai, Z., and Vasconcelos, N. 2018. Cascade r-cnn: Delving into high quality object detection. In *CVPR 2018*.

[Cai et al. 2016] Cai, Z.; Fan, Q.; Feris, R. S.; and Vasconcelos, N. 2016. A unified multi-scale deep convolutional neural network for fast object detection. In *ECCV 2016*, 354–370.

[Dai et al. 2016] Dai, J.; Li, Y.; He, K.; and Sun, J. 2016. R-FCN: Object detection via region-based fully convolutional networks. In *NIPS 2016*, 379–387.

[Dai et al. 2017] Dai, J.; Qi, H.; Xiong, Y.; Li, Y.; Zhang, G.; Hu, H.; and Wei, Y. 2017. Deformable convolutional networks. In *ICCV 2017*, 764–773.

[Fu et al. 2017] Fu, C.; Liu, W.; Ranga, A.; Tyagi, A.; and Berg, A. C. 2017. DSSD : Deconvolutional single shot detector. *CoRR* abs/1701.06659.

[Girshick 2015] Girshick, R. B. 2015. Fast R-CNN. In *ICCV 2015*, 1440–1448.

[He et al. 2015] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE TPAMI.* 37(9):1904–1916.

[He et al. 2016] He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *CVPR 2016*, 770–778.

[He et al. 2017] He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask R-CNN. In *ICCV 2017*, 2980–2988. IEEE.

[Hu, Shen, and Sun 2017] Hu, J.; Shen, L.; and Sun, G. 2017. Squeeze-and-excitation networks. *CoRR* abs/1709.01507.

[Huang et al. 2017] Huang, G.; Liu, Z.; Van Der Maaten, L.; and Weinberger, K. Q. 2017. Densely connected convolutional networks. In *CVPR 2017*, volume 1, 3.

[Kong et al. 2017] Kong, T.; Sun, F.; Yao, A.; Liu, H.; Lu, M.; and Chen, Y. 2017. RON: reverse connection with objectness prior networks for object detection. In *CVPR 2017*, 5244–5252.

[Law and Deng 2018] Law, H., and Deng, J. 2018. Cornernet: Detecting objects as paired keypoints. In *ECCV 2018*.

[Lin et al. 2014] Lin, T.; Maire, M.; Belongie, S. J.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; and Zitnick, C. L. 2014. Microsoft COCO: common objects in context. In *ECCV 2014*, 740–755.

[Lin et al. 2017a] Lin, T.; Dollár, P.; Girshick, R. B.; He, K.; Hariharan, B.; and Belongie, S. J. 2017a. Feature pyramid networks for object detection. In *CVPR 2017*, 936–944.

[Lin et al. 2017b] Lin, T.; Goyal, P.; Girshick, R. B.; He, K.; and Dollár, P. 2017b. Focal loss for dense object detection. In *ICCV 2017*, 2999–3007.

[Lin, Chen, and Yan 2014] Lin, M.; Chen, Q.; and Yan, S. 2014. Network in network. In *ICLR 2014*.

[Liu et al. 2016] Liu, W.; Anguelov, D.; Erhan, D.; Szegedy, C.; Reed, S. E.; Fu, C.; and Berg, A. C. 2016. SSD: single shot multibox detector. In *ECCV 2016*, 21–37.

[Newell, Yang, and Deng 2016] Newell, A.; Yang, K.; and Deng, J. 2016. Stacked hourglass networks for human pose estimation. In *ECCV 2016*, 483–499.

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

[Ren et al. 2015] Ren, S.; He, K.; Girshick, R. B.; and Sun, J. 2015. Faster R-CNN: towards real-time object detection with region proposal networks. In *NIPS 2015*, 91–99.

[Russakovsky et al. 2015] Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M. S.; Berg, A. C.; and Li, F. 2015. Imagenet large scale visual recognition challenge. *IJCV 2015* 115(3):211–252.

[Shrivastava et al. 2016] Shrivastava, A.; Gupta, A.; Girshick, R. B.; and AA. 2016. Training region-based object detectors with online hard example mining. In *CVPR 2016*, 761–769.

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

[Singh and Davis 2018] Singh, B., and Davis, L. S. 2018. An analysis of scale invariance in object detection–snip. In *CVPR 2018*, 3578–3587.

[Tychsen-Smith and Petersson 2018] Tychsen-Smith, L., and Petersson, L. 2018. Improving object localization with fitness NMS and bounded iou loss. In *CVPR 2018*.

[Xie et al. 2017] Xie, S.; Girshick, R. B.; Dollár, P.; Tu, Z.; and He, K. 2017. Aggregated residual transformations for deep neural networks. In *CVPR 2017*, 5987–5995.

[Zhang et al. 2018] Zhang, S.; Wen, L.; Bian, X.; Lei, Z.; and Li, S. Z. 2018. Single-shot refinement neural network for object detection. In *IEEE CVPR*.

[Zhou et al. 2018] Zhou, P.; Ni, B.; Geng, C.; Hu, J.; and Xu, Y. 2018. Scale-transferrable object detection. In *CVPR 2018*, 528–537.

[Zhu et al. 2017] Zhu, Y.; Zhao, C.; Wang, J.; Zhao, X.; Wu, Y.; and Lu, H. 2017. Couplenet: Coupling global structure with local parts for object detection. In *ICCV 2017*, 4146–4154.