FSSD: Feature Fusion Single Shot Multibox Detector

Zuo-Xin Li[1]   Lu Yang[2]   Fu-Qiang Zhou[1]

[1]Key Laboratory of Precision Opto-mechatronics Technology, Ministry of Education,
Beihang University, Beijing 100191, China

[2]Beijing University of Posts and Telecommunications

[1]{lizuoxin, zfq}@buaa.edu.cn, [2]soeaver@bupt.edu.cn

Abstract

SSD (Single Shot Multibox Detetor) is one of the best object detection algorithms with both high accuracy and fast speed. However, SSD's feature pyramid detection method makes it hard to fuse the features from different scales. In this paper, we proposed FSSD (Feature Fusion Single Shot Multibox Detector), an enhanced SSD with a novel and lightweight feature fusion module which can improve the performance significantly over SSD with just a little speed drop. In the feature fusion module, features from different layers with different scales are concatenated together, followed by some down-sampling blocks to generate new feature pyramid, which will be fed to multibox detectors to predict the final detection results. On the Pascal VOC 2007 test, our network can achieve 82.7 mAP (mean average precision) at the speed of 65.8 FPS (frame per second) with the input size 300×300 using a single Nvidia 1080Ti GPU. In addition, our result on COCO is also better than the conventional SSD with a large margin. Our FSSD outperforms a lot of state-of-the-art object detection algorithms in both aspects of accuracy and speed. Code is available at https://github.com/lzx1413/CAFFE_SSD/tree/fssd.

1. Introduction

Object detection is one of the core tasks in computer vision. In recent years, a lot of detectors based on ConvNets have been proposed to improve the accuracy and speed in object detection task [8, 27, 3, 20, 26]. But scale variations in object detection is still a critical challenge for all of the detectors. As shown in Fig. 1, there are some approaches which have been proposed to solve the multi-scale objects detection problem. Fig. 1 (a) applies a ConvNet to different scale images to generate different scale feature maps, which is a quite inefficient way. Fig. 1 (b) only selects one scale feature map but creates anchors with different scales to detect multi-scale objects. This method is adopted by Faster RCNN [27], RFCN [3] and so on. But the fixed receptive field size is a limitation to detect too large or too small objects. Top-Down structure like (c) in Fig. 1 is popular recently and has been proved working well in FPN [18], DSSD [7] and SharpMask [24]. But fusing features layer by layer is not efficient enough while there are many layers to be combined together.

The main trade-off of object detectors which are based on ConvNets is that the contradiction between object recognition and location. With deeper ConvNet, the feature maps can represent more semantic information with translation invariance, which is beneficial to object recognition but harmful to object location. To solve this problem, SSD adopts feature pyramid to detect objects with different scales. For VGG16 [31] as the backbone network, Conv4_3 with feature stride 8 is used to detect small objects while Conv8_2 with feature stride 64 to detect large objects. This strategy is rational because small objects will not lose too much location information in the shallow layers and large objects can also be well located and recognized in the deeper layers. But the problem is that the feature of small objects generated by shallow layers lacks enough semantic information which will lead to the poor performance on small object detection. Besides, small objects also rely on the context information heavily [2]. The first row of Fig. 6 shows us some missing detections for small objects by SSD.

In this paper, to tackle these problems mentioned above, we propose Feature Fusion SSD(FSSD) by adding a lightweight and efficient feature fusion module to the conventional SSD. We first define a framework of feature fusion module and abstract the factors which have critical effect on the feature fusion performance on object detection. According to the theoretic analysis and experiment results in Section 3 and Section 4, the architecture of feature fusion module is defined as follows: Features from different layers with different scales are projected and concatenated together followed by a Batch Normalization [14] layer to normalize the feature values. Then we append some down-
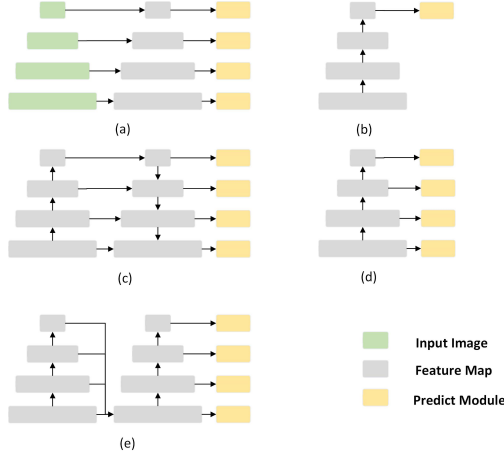
Figure 1. (a) Features are computed from images with different scales independently, which is an inefficient way. (b) Just one scale features are used to detect objects, which is used in some two stage detectors such as Faster R-CNN [27] and R-FCN [3]. (c) Feature fusion method adopted by [18, 24], features are fused from top to bottom layer by layer. (d) Use the feature pyramid generated from a ConvNet. The conventional SSD is one of the examples. (e) Our proposed feature fusion and feature pyramid generation method. Features from different layers with different scales are concatenated together first and used to generate a series of pyramid features later.

sampling blocks to generate new feature pyramid, which are fed to multibox detectors to produce the final detection results.

Using the proposed architecture, our FSSD improves a lot in performance at a slight expense of speed compared with conventional SSD. We evaluate the FSSD in VOC PASCAL [6] dataset and MSCOCO [19] dataset. The results indicate that our FSSD can improve the conventional SSD with a large margin especially for small objects without any bells and whistles. Besides, our FSSD also outperforms a lot of state-of-the-art object detectors based on VGGNet including ION [1] and Faster RCNN [27]. Our feature fusion module can also work better than FPN [18] in the object detection task. Our main contributions are summarized as follows:

(1) We define the feature fusion framework and investigate the factors to confirm the structure of the feature fusion module.

(2) We introduce a novel and lightweight way of combining feature maps from different levels and generating feature pyramid to fully utilize the features.

(3) With quantitative and qualitative experiments, we prove that our FSSD has a significant improvement over the conventional SSD with slight speed drop. FSSD can achieve state-of-the-art performance on both PASCAL VOC dataset and MS COCO dataset.

2. Related Work

Object detector with deep ConvNet   Benefited from the power of Deep ConvNes, object detector such as Over-Feat [29] and R-CNN [9] have began to show the dramatic improvements in accuracy. OverFeat applies a ConvNet as a feature extractor in the sliding window on an image pyramid. R-CNN [9] uses the region proposals generated from selective search [32] or Edge boxes [33] to generate the region-based feature from a pre-trained ConvNet and SVMs are adopted to do classification. SPPNet [11] adopts a spatial pyramid pooling layer which allows the classification module to reuse the ConvNet feature regardless of the input image resolutions. Fast R-CNN [8] introduces to train the ConvNet with both the classification and location regression loss end to end. Faster R-CNN [27] suggests to replace selective search with a region proposal network (RPN). RPN is used to generate the candidate bounding boxes (anchor boxes) and filter out the background regions at the same time. Then another small network is used to do classification and bounding box location regression based on these proposals. R-FCN [3] replaces ROI pooling in the Faster RCNN with position sensitive ROI pooling (PSROI) to improve the detector's quality with both aspects of accuracy and speed. Recently, Deformable Convolutional Network [4] proposes deformable convolution and deformable PSROI to enhance the RFCN further with better accuracy.

Except the region based detectors, there are also some efficient one stage object detectors. YOLO (you only look once) [25] divides the input image into several grids and performs localization and classification on each part of image. Benefited from this method, YOLO can run object detection at a very high speed but the accuracy is not satisfactory enough. YOLOv2 [26] is an enhanced version of YOLO and it improves the YOLO by removing the fully connected layers and adopts anchor boxes like the RPN.

SSD [20] is another efficient one stage object detector. As illustrated in Fig. 2 (a), SSD predicts the class scores and location offsets for the default bounding boxes by two 3×3 convolutional layers. In order to detect objects with different scales, SSD adds a series of progressively smaller convolutional layers to generate pyramid feature maps and sets corresponding anchor size according to the receptive filed size of the layers. Then NMS (non-maximum suppression) is used to post-process the final detection results. Because SSD detects objects directly from the plane ConvNet feature maps, it can achieve real-time object detection and process faster than most of other state-of-the-art object detectors.

In order to improve the accuracy, DSSD [7] suggests to augment SSD+ResNet-101 with deconvolution layers to introduce additional larges-scale context. However, the speed is slow because of the model complexity. RSSD [15] uses rainbow concatenation through both pooling and concate-

nation to fully utilize the relationship between the layers in the feature pyramid to enhance the accuracy with a little speed lost. DSOD [30] investigates how to train a object detector from scratch and designs a DenseNet architecture to improve the parameter efficiency.

Algorithms using feature fusion in ConvNet There are a lot of approaches which attempt to use multiple layers' features to improve the performance in computer vision tasks. HyperNet [17], Parsenet [21] and ION [1] concatenate features from multiple layers before predicting the result. FCN [22], U-Net [28], Stacked Hourglass networks [23] also use skip connections to associate low-level and high-level feature maps to fully utilize the synthetic information. SharpeMask [24] and FPN [18] introduce top-down structure to combine the different level features together to enhance the performance.

3. Method

ConvNets have an excellent ability to extract pyramidal feature hierarchy, which has more semantic information from low to high levels. Conventional SSD regards these features in different levels as the same level and generates the object detection results directly from them. This strategy makes SSD lack the capability to capture both the local detailed features and global semantic features. However, the detector should merge the context information and their detailed features to confirm the small objects. So synthesizing the features with a slight structure is an important solution for a ConvNet object detector to improve the accuracy.

3.1. Feature Fusion Module

As mentioned in Section 2, there have been a lot of algorithms trying to observe and fully utilize the pyramidal features. The most common method is like Fig. 1 (c). This type of feature fusion is used in FPN [18] and DSSD [7] and is verified to improve the conventional detector's performance a lot. But this design needs multiple feature merging processes. As shown in Fig. 1 (c), the new features on the right side can only fuse the features from the corresponding level features on the left and higher levels. Besides, latent feature and multiple features element-wise process also consume a lot of time. We propose a lightweight and efficient feature fusion module to handle this task. Our motivation is to fuse the different level features at once with a appropriate way and generate feature pyramid from the fused features. There are several factors we should consider when design the feature fusion module. We will investigate them in the following part. Assuming $X_i, i \in \mathcal{C}$ are the source feature maps we want to fuse, the feature fusion module can be described as follows:

$$X_f = \phi_f\{\mathcal{T}_i(X_i)\} \qquad i \in \mathcal{C} \qquad (1)$$

$$X'_p = \phi_p(X_f) \qquad p \in \mathcal{P} \qquad (2)$$

$$loc, class = \phi_{c,l}(\cup\{X'_p\}) \qquad p \in \mathcal{P} \qquad (3)$$

where $\mathcal{T}_i$ means the transformation function of each source feature map before being concatenated together. ϕ_f is the feature fusion function. ϕ_p is the function to generate pyramid features. $\phi_{c,l}$ is the method to predict object detections from the provided feature maps.

We focus on the range of layers which should be fused or not ($\mathcal{C}$), how to fuse the selected feature maps ($\mathcal{T}$ and ϕ_f) and how to generate pyramid features (ϕ_p).

$\mathcal{C}$: In the conventional SSD300 based on VGG16, the author chooses conv4_3, fc_7 of the VGG16 and new added layer conv6_2, conv7_2, conv8_2, conv9_2 to generate features to process object detections. The corresponding feature size is 38×38, 19×19, 10×10, 5×5, 3×3 and 1×1. We think that feature maps whose spatial size is smaller than 10×10 have little information to merge, so we set the range of layers as conv3_3, conv4_3, fc_7 and conv7_2 (we set the stride of conv6_2 to 1 so the feature map size of conv7_2 is 10×10). According to the analysis in Section 4.1.1, conv3_3 brings no profit to the accuracy, so we do not fuse this layer.

ϕ_f: There are mainly two ways to merge different feature maps together: concatenation and element-wise summation. Element-wise summation requires that feature maps should have the same size which means we have to convert the feature maps to the same channels. Because this requirement limits the flexibility of fusing feature maps, we prefer to use concatenation. Besides, according to the result in Section 4.1.2, concatenation can get better result than element-wise summation. So we use concatenation to combine the features.

$\mathcal{T}$: In order to concatenate the features with different scales in a simple and efficient way, we adopt the following strategy. Conv 1×1 is applied to each of the source layers to reduce the feature dimension firstly. Then we set the size of conv4_3's feature map as the basic feature map size, which means our smallest feature step is 8. Feature maps generated from conv3_3 are down-sampled to 38×38 with 2×2 max-pooling whose stride is 2. As for the feature maps whose size is smaller than 38×38, we use bilinear interpolation to resize the feature maps to the same size with conv4_3. In this way, all the features have the same size on spatial dimension.
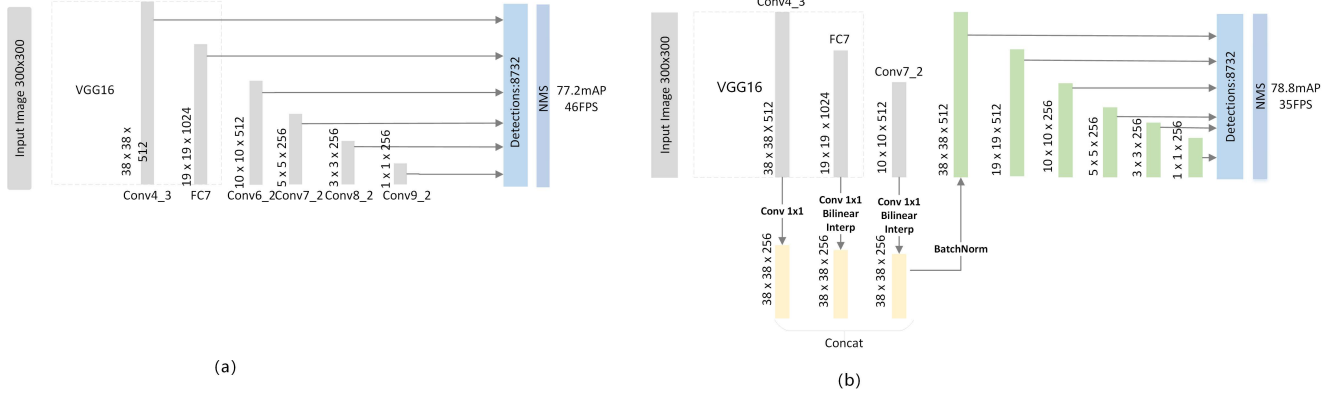
Figure 2. (a) is the SSD framework proposed in [20], (b) is our F-SSD framework.
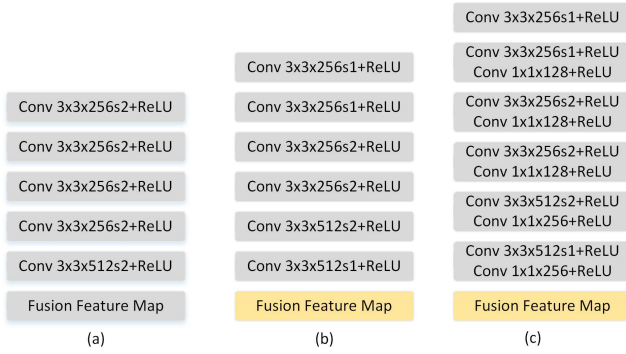

Figure 3. Pyramid feature generators for FSSD300. We use the feature maps from gray blobs to detect objects. In (a), the fusion feature map takes part in the object detection. In (b), we only detect objects on the feature maps after the fusion feature map. (c) We replace the simple group of Conv+ReLU with a bottleneck block which consists of two Conv+ReLU layers.

structure	VOC2007 test mAP
simple block including fusion feature	78.2
simple block after fusion feature	**78.8**
bottleneck block after fusion feature	78.4

Table 1. mAP on VOC2007 test while using different pyramid feature generation structure.

ϕ_p: Adopted from the conventional SSD, we also use the pyramid feature map to generate object detection results. We test three different structures and compare the results to select the best one. According to the result in Section 4.1.4, we select the structure which is composed of several simple blocks to extract the feature pyramid.

3.2. Training

There are two main training methods we can choose. Firstly, because our FSSD is based on the SSD model, we can adopt the well trained SSD model as our pre-trained model. As for the learning rate, we set the new feature fu-
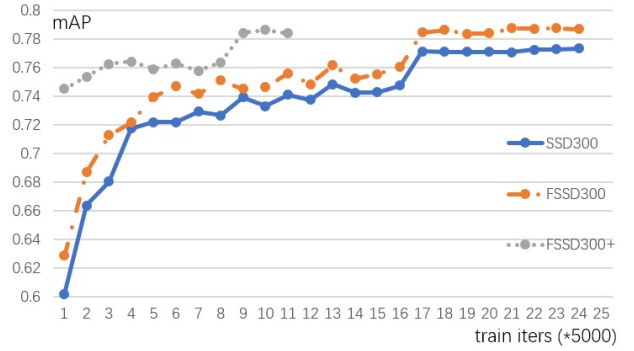

Figure 4. Training process comparison. The vertical axis denotes the mAP calculated on VOC2007 test set and the horizontal axis represents the training iterations. SSD means that training the conventional SSD model with the default settings from a pre-trained VGG16 model. FSSD means that we train the FSSD model with a pre-trained VGG model. FSSD's training parameters are the same with SSD. FSSD+ means that we train the FSSD from a pre-trained SSD model. The FSSD+ is only optimized for 60k iterations. All of the models are trained on VOC07+12 dataset.

sion module's learning rate two times larger than other parameters. Another way to train the FSSD is the same as the conventional SSD algorithm. According to the experiments in Table 2(rows 2 and 5), these two ways have little difference on the final result but training FSSD from VGG16 has a slightly better results than the one which is trained from SSD model. But as illustrated in Fig. 4, training FSSD from SSD models can converge faster than from the pre-trained VGGNet model. Besides, our FSSD also coverages faster than the conventional SSD while having the same training hyper-parameters . In order to test our algorithms in the limited time, we use the first way to train the FSSD by default.

The training objective is the same as SSD. We use the center code type to encode the bounding boxes and have the same matching strategy, hard negative mining strategy and data augmentation with SSD. More details can be seen in

[20].

4. Experiments

In order to compare our FSSD with the conventional SSD fairly, our experiments are all based on VGG16[31] which is preprocessed like SSD [20]. We conduct experiments on PASCAL VOC 2007, 2012 [6] and MS COCO dataset [19]. In VOC 2007 and VOC 2012, a predicted bounding box is correct if its intersection over union (IOU) with the ground truth is higher than 0.5. We adopt the mean average precision (mAP) as the metric for evaluating detection performance. As for MSCOCO, we upload the result to the evaluation server to get the performance analysis. All of our experiments are based on the Caffe [16] version of SSD implementation.

4.1. Ablation Study on PASCAL VOC2007

In this Section, we investigate some important design factors about the feature fusion module. We compare the results on PASCAL VOC 2007 with input size 300×300. In these experiments, the models are trained with the combined dataset from 2007 `trainval` and 2012 `trainval` (VOC07+12) and tested on VOC 2007 `test` set. The results are summarized in Table 2 mostly.

4.1.1 The range of layers to be fused

In Table 2, we compare the FSSD with different fusion layers. While we fuse all of the feature maps (conv3_3, conv4_3, fc_7 and conv7_2), the mAP on VOC2007 `test` set(row 2) is 78.6%. It is interesting that if we remove the conv3_3, the mAP is increased to 78.8%(row 5), which means that the down-sampled feature map from conv3_3 whose feature stride is 4 has no benefit to the final performance. But according to the result in Table 2 4th row, it is better to keep the conv7_2.

4.1.2 Feature fusion: concatenation or element-wise summation

In Table 2, using concatenation to fuse feature can get 78.6% mAP(row 2) while element-wise summation can only achieve 76.3%(row 7). The results show that concatenation is better than element-wise summation with a large margin of 2.3 points.

4.1.3 Normalize feature value or not

Feature maps from different layers always have different ranges of value. In conventional SSD, the author used L2Normalization to scale the feature map from conv4_3. As we want to use a simple and efficient way to scale the feature maps, we add a Batch Normalization layer after the concatenation process. Results in Table 2(rows 2 and 6) show that using the Batch Normalization layer to re-scale the feature maps can bring us about 0.7% mAP improvement.

4.1.4 Pyramid feature extractor design

We compare three different structures of pyramid feature extractor as shown in Fig. 3. There are two kinds of blocks to generate lower resolution feature maps, simple block (one Conv3 × 3 followed by a ReLU) and bottleneck block, which uses Conv 1 × 1 to reduce the feature dimension first and is adopted by the conventional SSD. Structure in Fig. 3 (a) is composed of several simple blocks and uses the fused feature to predict objects. Fig. 3 (b) is the same with (a) except not using fused feature for prediction directly. Fig. 3 (c) replaces the simple block with bottleneck block in (b). The results which are shown in Table 1 indicate that structure (b) is better than other two designs with a little improvement.

4.2. Results on PASCAL VOC

Experimental setup   According to the ablation study in Section 4.1, the architecture of our FSSD is defined as follows: For FSSD with 300×300 input(FSSD300), we adopt VGG16 as the backbone network. All of the raw features are converted to 256 channels with a 1×1 convolutional layer. Feature maps from fc_7 and conv7_2 are interpolated to 38×38. Then the transformed feature maps are concatenated together followed by a Batch Normalization layer to normalize the feature values. Then several down-sampling blocks(including one 3×3 convolutional layer with stride 2 and one ReLU layer) are appended one by one to generate the pyramid features.

Results on PASCAL VOC2007   We use VOC 2007 `trainval` and VOC2012 `trainval` to train FSSD following SSD [20]. We train the FSSD300 on two Nvidia 1080Ti GPUs with batch size 32 for 120k iterations. The initial learning rate is set to 0.001 and then divided by 10 at step 80k, 100k and 120k. Following the training strategy in SSD [20], the weight decay is set to 0.0005. We adopt a SGD with momentum 0.9 to optimize the FSSD which is initialized by a well pre-trained VGG16 on ImageNet. In order to use COCO models as the pre-trained model, we first train the COCO model with 80 classes which will be described in details in Section 4.3 . Then a subset of the 80 classes model which is corresponding to the classes in PASCAL VOC 20 classes are extracted from the COCO model as the pre-trained VOC model. Our results on VOC2007 `test` set are shown in Table 3. Our FSSD300 can achieve 78.8% mAP, which improves 1.1 points compared with the conventional SSD300. In addition, our result of FSSD300 is also higher than DSSD321, even though DSSD321 uses

Method	data	BN	pre-trained VGG	pre-trained SSD	feature fusion	fusion layers	mAP
FSSD300	07+12	✓	✗	✗	Concat	conv3-conv7	72.7
FSSD300	07+12	✓	✗	✓	Concat	conv3-conv7	78.6
FSSD300	07+12	✓	✗	✓	Concat	conv4-conv7	**78.8**
FSSD300	07+12	✓	✗	✓	Concat	conv4-fc7	78.6
FSSD300	07+12	✓	✓	✗	Concat	conv3-conv7	**78.8**
FSSD300	07+12	✗	✗	✓	Concat	conv3-conv7	78.1
FSSD300	07+12	✓	✗	✓	ele-sum	conv3-conv7	76.3

Table 2. Results of the ablation study on PASCAL VOC2007. **BN** means that a Batch Normalization layer is added after the feature concatenation. **pre-trained VGG** means that a pre-trained VGG16 is adopted to initialize the model. **pre-trained SSD** means that the FSSD is optimized from a well-trained SSD model. The options of layers we can fuse include *conv3_3, conv4_3 ,fc_7, conv7_2*. The **fusion layers** represents the layers we choose to merge. The mAP is measured on VOC2007 `test` set.

Method	data	pre-train	backbone network	speed (*fps*)	GPU	#proposals	input size	mAP
Faster RCNN [27]	07+12	✓	VGGNet	7	Titan X	6000	$\sim 600 \times 1000$	73.2
Faster RCNN [27]	07+12	✓	ResNet-101	2.4	K40	300	$\sim 600 \times 1000$	76.4
R-FCN [3]	07+12	✓	ResNet-50	-	-	300	$\sim 600 \times 1000$	77.0
R-FCN [3]	07+12	✓	ResNet-101	5.8	K40	300	$\sim 600 \times 1000$	79.5
YOLOv2 [26]	07+12	✓	Darknet-19	81	Titan X	-	352×352	73.7
SSD300S[†] [30]	07+12	✗	VGGNet	46	Titan X	8732	300×300	69.6
SSD300* [20]	07+12+COCO	✓	VGGNet	46	Titan X	8732	300×300	81.2
SSD300 [20]	07+12	✓	VGGNet	46	Titan X	8732	300×300	77.2
SSD300 [20]	07+12	✓	VGGNet	85	1080Ti	8732	300×300	77.2
SSD512 [20]	07+12	✓	VGGNet	19	Titan X	24564	512×512	78.5
SSD512* [20]	07+12+COCO	✓	VGGNet	19	Titan X	24564	512×512	83.2
DSOD300 [30]	07+12	✗	DS/64-192-48-1	17.4	Titan X	-	300×300	77.7
DSOD300* [30]	07+12+COCO	✗	DS/64-192-48-1	17.4	Titan X	-	300×300	81.7
DSSD321 [7]	07+12	✓	ResNet-101	9.5	Titan X	17080	321×321	78.6
DSSD513 [7]	07+12	✓	ResNet-101	5.5	Titan X	43688	321×321	81.5
RSSD300 [15]	07+12	✓	VGGNet	35	Titan X	8732	300×300	78.5
RSSD512 [15]	07+12	✓	VGGNet	16.6	Titan X	24564	512×512	80.8
FSSD300S[†]	07+12	✗	VGGNet	65.8	1080Ti	8732	300×300	72.7
FSSD300	07+12	✓	VGGNet	65.8	1080Ti	8732	300×300	78.8
FSSD300*	07+12+COCO	✓	VGGNet	65.8	1080Ti	8732	300×300	**82.7**
FSSD512	07+12	✓	VGGNet	35.7	1080Ti	24564	512×512	80.9
FSSD512*	07+12+COCO	✓	VGGNet	35.7	1080Ti	24564	512×512	**84.5**

Table 3. **PASCAL VOC 2007 `test` detection results.** SSDs' results we cited here are the updated version by the author after the paper publication with more data argumentation. SSD300S[†] indicates training SSD300 from scratch VGGNet, which is tested in DSOD[30]. FSSD300S[†] is also trained from scratch VGGNet. The speed of FSSDs is tested on a single Nvidia 1080Ti GPU. For fair comparison, we also test the SSD's speed on the same single Nvidia 1080Ti GPU.

ResNet-101 as the backbone network, which has better performance compared with VGG16. With COCO as the training data, FSSD300's performance can be further increased to 82.7%, which exceeds DSOD by 1% and SSD300 by 1.5%. Our FSSD512 also improves the SSD512 from 79.8% to 80.9%, which is also a little higher than RSSD512. DSSD512 is better than our FSSD512 with the aspect of accuracy but we think the ResNet-101 backbone plays a crit-

ical role in this progress. However, our FSSD512 is much faster than DSSD512.

It is interesting that DSOD [30] researches the object detections task with a particular way: training an object detector from scratch. The conventional SSD with VGG16 can only achieve 69.6% mAP without the pre-trained VGGNet model. We also investigate whether our FSSD can improve the conventional SSD. Result in Table 3 (row 19) shows us

Method	data	backbone network	pre-train	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Faster RCNN [27]	07++12+COCO	ResNet-101	✓	83.8	92.1	88.4	84.8	**75.9**	**71.4**	86.3	87.8	94.2	66.8	89.4	69.2	**93.9**	91.9	90.9	89.6	**67.9**	88.2	76.8	90.3	80.0
Faster RCNN [27]	07++12+COCO	VGGNet	✓	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2
YOLOv2 [26]	07++12+COCO	Darknet-19	✓	81.5	90.0	88.6	82.2	71.7	65.5	85.5	84.2	92.9	67.2	87.6	70.0	91.2	90.5	90.0	88.6	62.5	83.8	70.7	88.8	79.4
BlitzNet512 [5]	07++12+S+COCO	ResNet-50	✓	83.8	**93.1**	89.4	84.7	75.5	65.0	86.6	87.4	94.5	**69.9**	88.8	**71.7**	92.5	91.6	91.1	88.9	61.2	**90.4**	**79.2**	**91.8**	**83.0**
SSD300 [20]	07++12+COCO	VGGNet	✓	79.3	91.0	86.0	78.1	65.0	55.4	84.9	84.0	93.4	62.1	83.6	67.3	91.3	88.9	88.6	85.6	54.7	83.8	77.3	88.3	76.5
SSD512 [20]	07++12+COCO	VGGNet	✓	82.2	91.4	88.6	82.6	71.4	63.1	87.4	88.1	93.9	66.9	86.6	66.3	92.0	91.7	90.8	88.5	60.9	87.0	75.4	90.2	80.4
DSOD300 [30]	07++12+COCO	DS/64-192-48-1	✗	79.3	90.5	87.4	77.5	67.4	57.7	84.7	83.6	92.6	64.8	81.3	66.4	90.1	87.8	88.1	87.3	57.9	80.3	75.6	88.1	76.7
FSSD300	07++12+COCO	VGGNet	✓	**82.0**	92.2	89.2	81.8	72.3	59.7	87.4	84.4	93.5	66.8	87.7	70.4	92.1	90.9	89.6	67.7	56.9	86.8	79.0	90.7	81.3
FSSD512	07++12+COCO	VGGNet	✓	**84.2**	92.8	**90.0**	**86.2**	**75.9**	67.7	**88.9**	**89.0**	**95.0**	68.8	**90.9**	68.7	92.8	**92.1**	**91.4**	**90.2**	63.1	90.1	76.9	91.5	82.7

Table 4. **PASCAL VOC 2012 `test` detection results. 07++12+COCO**: 07 `trainval` + 07 `test` + 12 `trainval` + MSCOCO. **07++12+S+COCO**: 07++12 plus segmentation labels and MSCOCO. Result links are FSSD300 (07++12+COCO) : http://host.robots.ox.ac.uk:8080/anonymous/YMA3TZ.html; FSSD512 (07++12+COCO): http://host.robots.ox.ac.uk:8080/anonymous/LQXCQK.html.

that our FSSD trained from scratch also improves the performance with a large margin 3.1 points. Even though this result is not as good as the DSOD, it should be noted that our FSSD still takes VGGNet as the base network while DSOD pays more attention on the design of the backbone network for better performance and DSOD is trained for more iterations (about 4 times than SSD).

Results on PASCAL VOC2012 We use VOC 2012 `trainval`, VOC 2007 `trainval_test` and MSCOCO to train our FSSD and test it on VOC2012 `test` set. Training hyper-parameters are the same with the experiments on VOC2007 except the training dataset. Table 4 summarizes some state-of-the-art detectors from the PASCAL VOC2012 leaderboard. FSSD300 with COCO can achieve 82.0% mAP, which is higher than the conventional SSD(79.3%) by 2.7 points. In addition, our FSSD512 can achieve 84.2% mAP, which exceeds the conventional SSD(82.2%) by 2 points. The FSSD512 takes the first place in VOC2012 leaderboard among all of the one-stage object detectors as of the time of submission.

4.3. Results on MS COCO

MSCOCO dataset has 80 object categories. We use the COCO Challenge 2017 data split to prepare our dataset. The train set involves 115K images which is comparable with the original trainval35k. We test the FSSDs on the 20k test-dev set. For training FSSD300, the learning rate is set to 0.001 for the first 280k iterations, then divided by 10 at the step 360k and 400k. But if we train the FSSD300 from a well trained SSD model, it will only need 120k totally to make the FSSD300 converge well. For training FSSD512, the learning rate is set to 0.001 for the first 280k iterations and then divided by 10 at the step 320k and 360k.

The COCO test results are shown in Table 5. FSSD 300 achieves 27.1% on the `test-dev` set, which is higher than the SSD300*(25.1%) with a large margin. Even though our FSSD does not perform as well as DSOD and DSSD, it should be noted that our base model is VGG16 and FSSD has the best accuracy compared with other algorithms such
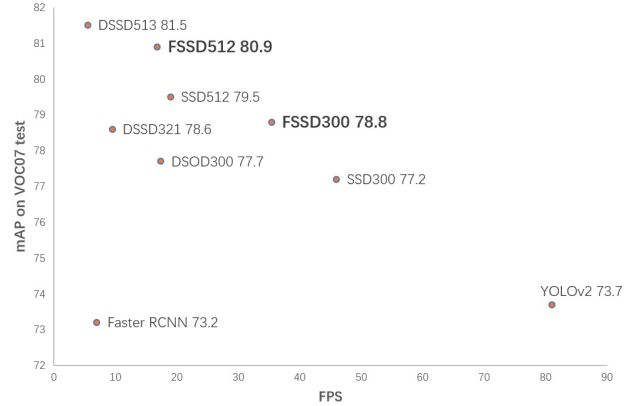

Figure 5. Speed and accuracy distribution with different object detection algorithms. The speeds are all measured on a single Titan X GPU. As we do not have a Titan X GPU, the speed of our FSSDs is calculated by comparing with SSD's speed which we have tested on our own Nvidia 1080Ti.

as Faster RCNN and ION in Table 5(rows 1 and 2) which is also based on VGGNet. Besides, FSSD512(31.8%) outperforms conventional SSD(28.8%) by 3 points. Even though our FSSD512 is slightly lower than DSSD513, it should be noted that FSSD's mAP on small objects is still higher than DSSD513, which proves that our feature fusion module is more powerful than DSSD's FPN module on small objects' detection.

4.4. Speed

The inference speed is shown in the 5th column of Table 3. Our FSSD can run at 65.8 FPS with a 300×300 input image on a single 1080Ti GPU. For fair comparisons, we also test the speed of SSD on the 1080Ti GPU. Because our FSSD adds additional layers on the SSD model, our FSSD consumes about 25% extra time. But compared with DSSD and DSOD, our methods are still much faster than them while the improvement from the SSD is about the same level. In Fig. 5, it is clear that our FSSD is faster than most of the object detection algorithms while having the competitive accuracy.

Method	data	network	pre-train	Avg. Precision, IoU:			Avg. Precision, Area:			Avg. Recall, #Dets:			Avg. Recall, Area:		
				0.5:0.95	0.5	0.75	S	M	L	1	10	100	S	M	L
Faster RCNN [27]	trainval	VGGNet	✓	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [1]	train	VGGNet	✓	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
R-FCN [3]	trainval	ResNet-101	✓	29.2	51.5	-	10.3	32.4	43.3	-	-	-	-	-	-
R-FCN_multi-sc [3]	trainval	ResNet-101	✓	29.9	51.9	-	10.8	32.8	45.0	-	-	-	-	-	-
YOLOv2 [26]	trainval35k	Darknet-19	✓	21.6	44.0	19.2	5.0	22.4	35.5	20.7	31.6	33.3	9.8	36.5	54.4
SSD300* [20]	trainval35k	VGGNet	✓	25.1	43.1	25.8	6.6	25.9	41.4	23.7	35.1	37.2	11.2	40.4	58.4
SSD512* [20]	trainval35k	VGGNet	✓	28.8	48.5	30.3	10.9	31.8	43.5	26.1	39.5	42.0	16.5	46.6	60.8
DSOD300 [30]	trainval	DS/64-192-48-1	✗	29.3	47.3	30.6	9.4	31.5	47.0	27.3	40.7	43.0	16.7	47.1	65.0
DSSD321 [7]	trainval35k	ResNet-101	✓	28.0	46.1	29.2	7.4	28.1	47.6	25.5	37.1	39.4	12.7	42.0	62.6
DSSD513 [7]	trainval35k	ResNet-101	✓	**33.2**	**53.3**	**35.2**	13.0	**35.4**	**51.1**	**28.9**	**43.5**	**46.2**	21.8	49.1	**66.4**
FSSD300	trainval35k	VGGNet	✓	27.1	47.7	27.8	8.7	29.2	42.2	24.6	37.4	40.0	15.9	44.2	58.6
FSSD512	trainval35k	VGGNet	✓	31.8	52.8	33.5	**14.2**	35.1	45.0	27.6	42.4	45.0	**22.3**	**49.9**	62.0

Table 5. **MS COCO `test-dev` 2015 detection results.**

4.5. Performance improvement from SSD

Our FSSD performs better than conventional SSD mainly in two aspects. Firstly, FSSD reduces the probability of detecting multi-part of one object repeatedly or merging multi-objects to one single object . For example, as illustrated in Fig. 6 column 1, The SSD model regards the head of the dog as a single dog as well, which is clearly wrong while the whole dog is in this image. However, FSSD can detect the whole dog at once. In addition, the column 2 in Fig. 6 shows us that the SSD detects one big dog which includes two dogs in fact. But FSSD does not make the mistake. Secondly, FSSD performs better on small objects. On the one hand, small objects can only activate smaller regions in the network comparing with large objects and the location information is easy to be lost in the detection process. On the other hand, small object's recognition relies more on the context around it. Because SSD only detects small objects from the shallow layers such as conv4_3, whose receptive field is too small to observe the lager context information, which leads to the SSD's bad performance on small objects. FSSD can observe all of the objects synthetically benefited from the feature fusion module. As shown in Fig. 6 column 3 to column 6. FSSD detects more small objects than SSD successfully.

5. Conclusion and future work

In this paper, We proposed FSSD, an enhanced SSD by applying a lightweight and efficient feature fusion module on it. First, we investigate the framework to fuse different features together and generate pyramid feature maps. The experiments' results show that feature maps from different layers can be well fully utilized by being concatenated together. Then several convolutional layers with stride 2 are applied to fused feature map to generate pyramid features. Experiments on VOC PASCAL and MSCOCO prove that our FSSD improves the conventional SSD a lot and outperforms several other state-of-the-art object detectors both in accuracy and efficiency without any bells and whistles.

In the future, it is worth enhancing our FSSD with much stronger backbone networks such as ResNet [12] and DenseNet [13] to get better performance on the MSCOCO dataset and replacing the FPN in Mask RCNN [10] with our feature fusion module is also an interesting research field.

References

[1] S. Bell, C. Lawrence Zitnick, K. Bala, and R. Girshick. Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2883, 2016. 00128. 2, 3, 8

[2] C. Chen, M.-Y. Liu, O. Tuzel, and J. Xiao. *R-CNN for Small Object Detection*, pages 214–230. Springer International Publishing, Cham, 2017. 1

[3] J. Dai, Y. Li, K. He, and J. Sun. R-fcn: Object detection via region-based fully convolutional networks. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 379–387. Curran Associates, Inc., 2016. 1, 2, 6, 8

[4] J. Dai, H. Qi, Y. Xiong, Y. Li, G. Zhang, H. Hu, and Y. Wei. Deformable convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 2

[5] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid. BlitzNet: A real-time deep network for scene understanding. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. 7

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

[7] C. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg. DSSD : Deconvolutional single shot detector. *CoRR*, abs/1701.06659, 2017. 1, 2, 3, 6, 8

[8] R. Girshick. Fast r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015. 1, 2

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic

Figure 6. SSD300 vs FSSD300. Both models are trained with VOC07+12 dataset. The top row contains the results from the conventional SSD300 and the bottom row is from our FSSD300 model. Bounding boxes with score of 0.5 or higher is drawn. Better viewed on screen.

segmentation. In *Computer Vision and Pattern Recognition*, 2014. 2

[10] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask r-cnn. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 8

[11] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence*, 37(9):1904–1916, 2015. 2

[12] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 8

[13] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 8

[14] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach and D. M. Blei, editors, *ICML*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org, 2015. 1

[15] J. Jeong, H. Park, and N. Kwak. Enhancement of SSD by concatenating feature maps for object detection. *CoRR*, abs/1705.09587, 2017. 2, 6

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 5

[17] T. Kong, A. Yao, Y. Chen, and F. Sun. Hypernet: Towards accurate region proposal generation and joint object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 845–853, 2016. 00026. 3

[18] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie. Feature pyramid networks for object detection. *CoRR*, abs/1612.03144, 2016. 1, 2, 3

[19] 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. 01470. 2, 5

[20] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A. Berg. *SSD: Single shot multibox detector*, volume

9905 LNCS of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 21–37. Springer Verlag, Germany, 2016. 1, 2, 4, 5, 6, 7, 8

[21] W. Liu, A. Rabinovich, and A. C. Berg. Parsenet: Looking wider to see better. *arXiv preprint arXiv:1506.04579*, 2015. 3

[22] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. 3

[23] A. Newell, K. Yang, and J. Deng. *Stacked Hourglass Networks for Human Pose Estimation*, pages 483–499. Springer International Publishing, Cham, 2016. 3

[24] P. O. Pinheiro, T.-Y. Lin, R. Collobert, and P. Dollár. Learning to refine object segments. In *ECCV*, 2016. 1, 2, 3

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016. 2

[26] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. 1, 2, 6, 7, 8

[27] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 91–99. Curran Associates, Inc., 2015. 1, 2, 6, 7, 8

[28] O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*, pages 234–241. Springer International Publishing, Cham, 2015. 3

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

[30] Z. Shen, Z. Liu, J. Li, Y.-G. Jiang, Y. Chen, and X. Xue. Dsod: Learning deeply supervised object detectors from scratch. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 3, 6, 7, 8

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 5

[32] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013. 2

[33] L. Zitnick and P. Dollar. Edge boxes: Locating object proposals from edges. In *ECCV*. European Conference on Computer Vision, September 2014. 2