arXiv:1512.02325v2 [cs.CV] 30 Mar 2016

SSD: Single Shot MultiBox Detector

Wei Liu[1], Dragomir Anguelov[2], Dumitru Erhan[3], Christian Szegedy[3], Scott Reed[4], Cheng-Yang Fu[1], Alexander C. Berg[1]

[1]UNC Chapel Hill [2]Zoox Inc. [3]Google Inc. [4]University of Michigan, Ann-Arbor
[1]wliu@cs.unc.edu, [2]drago@zoox.com, [3]{dumitru,szegedy}@google.com,
[4]reedscot@umich.edu, [1]{cyfu,aberg}@cs.unc.edu

Abstract. We present a method for detecting objects in images using a single deep neural network. Our approach, named SSD, discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. Our SSD model is simple relative to methods that require object proposals because it completely eliminates proposal generation and subsequent pixel or feature resampling stage and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems that require a detection component. Experimental results on the PASCAL VOC, MS COCO, and ILSVRC datasets confirm that SSD has comparable accuracy to methods that utilize an additional object proposal step and is much faster, while providing a unified framework for both training and inference. Compared to other single stage methods, SSD has much better accuracy, even with a smaller input image size. For 300×300 input, SSD achieves 72.1% mAP on VOC2007 `test` at 58 FPS on a Nvidia Titan X and for 500×500 input, SSD achieves 75.1% mAP, outperforming a comparable state of the art Faster R-CNN model. Code is available at `https://github.com/weiliu89/caffe/tree/ssd`.

Keywords: Real-time Object Detection; Convolutional Neural Network

1 Introduction

Current state-of-the-art object detection systems are variants of the following approach: hypothesize bounding boxes, resample pixels or features for each box, and apply a high-quality classifier. This pipeline has prevailed on detection benchmarks since the Selective Search work [1] through the current leading results on PASCAL VOC, MS COCO, and ILSVRC detection all based on Faster R-CNN[2] albeit with deeper features such as [3]. Although accurate, these approaches have been too computationally intensive for embedded systems and, even with high-end hardware, too slow for real-time or near real-time applications. Often detection speed for these approaches is measured in seconds per frame, and even the fastest high-accuracy detector, the basic Faster R-CNN, operates at only 7 frames per second (FPS). There have been a wide range of attempts to build faster detectors by attacking each stage of the detection pipeline (see related work

in Sec. 4), but so far, significantly increased speed comes only at the cost of significantly decreased detection accuracy.

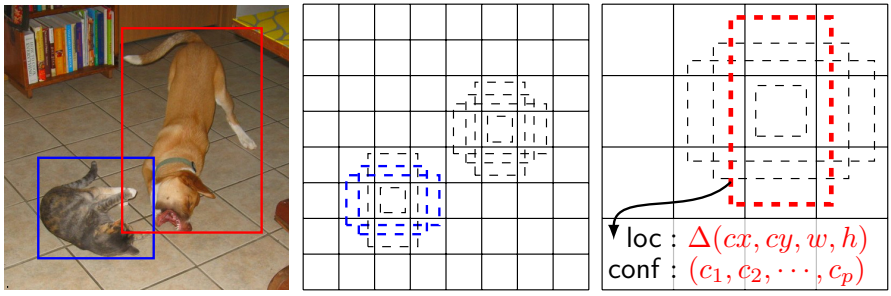
This paper presents the first deep network based object detector that does not resample pixels or features for bounding box hypotheses *and* is as accurate as approaches that do. This results in a significant improvement in speed for high-accuracy detection (58 FPS with mAP 72.1% on VOC2007 `test`, vs Faster R-CNN 7 FPS with mAP 73.2% or YOLO 45 FPS with mAP 63.4%). The fundamental improvement in speed comes from eliminating bounding box proposals and the subsequent pixel or feature resampling stage. This is not the first paper to do this (cf [4,5]) but by adding a series of improvements, we manage to increase the accuracy significantly over previous attempts. Our improvements include using a small convolutional filter to predict object categories and offsets in bounding box locations, using separate predictors (filters) for different aspect ratio detections, and applying these filters to multiple feature maps from the later stages of a network in order to perform detection at multiple scales. With these modifications we can achieve high-accuracy detection using relatively low resolution input, further increasing processing speed. While these contributions may seem small independently, we note that the resulting system improves accuracy on high-speed detection for PASCAL VOC from 63.4% mAP for YOLO to 72.1% mAP for our proposed network. This is a larger relative improvement in detection accuracy than that from the recent, very high-profile work on residual networks [3]. Furthermore, significantly improving the speed of high-quality detection can broaden the range of settings where computer vision is useful.

We summarize our contributions as follows:

- We introduce SSD, a single-shot detector for multiple categories that is faster than the previous state of the art for single shot detectors (YOLO), and significantly more accurate, in fact as accurate as slower techniques that perform explicit region proposals and pooling (including Faster R-CNN).
- The core of the SSD approach is predicting category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps.
- In order to achieve high detection accuracy we produce predictions of different scales from feature maps of different scales, and explicitly separate predictions by aspect ratio.
- Together, these design features lead to simple end-to-end training and high accuracy, even with relatively low resolution input images, further improving the speed vs accuracy trade-off.
- Experiments include timing and accuracy analysis on models with varying input size evaluated on PASCAL VOC, MS COCO, and ILSVRC and are compared to a range of recent state-of-the-art approaches.

2 The Single Shot Detector (SSD)

This section describes our proposed SSD framework for detection (Sec. 2.1) and the associated training methodology (Sec. 2.2). Afterwards, Sec. 3 presents dataset-specific model details and experimental results.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Fig. 1: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ($(c_1, c_2, \cdots, c_p)$). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax).

2.1 Model

The SSD approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression step to produce the final detections. The early network layers are based on a standard architecture used for high quality image classification (truncated before any classification layers), which we will call the base network[1]. We then add auxiliary structure to the network to produce detections with the following key features:

Multi-scale feature maps for detection We add convolutional feature layers to the end of the truncated base network. These layers decrease in size progressively and allow predictions of detections at multiple scales. The convolutional model for predicting detections is different for each feature layer (*cf* Overfeat[4] and YOLO[5] that operate on a single scale feature map).

Convolutional predictors for detection Each added feature layer (or optionally an existing feature layer from the base network) can produce a fixed set of detection predictions using a set of convolutional filters. These are indicated on top of the SSD network architecture in Fig. 2. For a feature layer of size $m \times n$ with p channels, the basic element for predicting parameters of a potential detection is a $3 \times 3 \times p$ *small kernel* that produces either a score for a category, or a shape offset relative to the default box

[1] In our reported experiments we use the VGG-16 network as a base, but other networks should also produce good results.

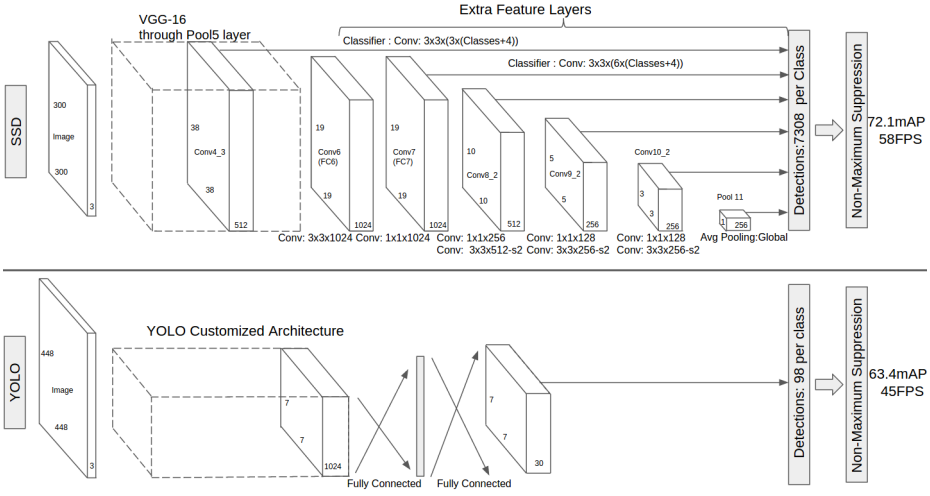


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300×300 input size significantly outperforms its 448×448 YOLO counterpart in accuracy on VOC2007 `test` while also improving the run-time speed, albeit YOLO customized network is faster than VGG16.

coordinates. At each of the $m \times n$ locations where the kernel is applied, it produces an output value. The bounding box offset output values are measured relative to a default box position relative to each feature map location (*cf* the architecture of YOLO[5] that uses an intermediate fully connected layer instead of a convolutional filter for this step).

Default boxes and aspect ratios We associate a set of default bounding boxes with each feature map cell, for multiple feature maps at the top of the network. The default boxes tile the feature map in a convolutional manner, so that the position of each box instance relative to its corresponding cell is fixed. At each feature map cell, we predict the offsets relative to the default box shapes in the cell, as well as the per-class scores that indicate the presence of a class instance in each of those boxes. Specifically, for each box out of k at a given location, we compute c class scores and the 4 offsets relative to the original default box shape. This results in a total of $(c + 4)k$ filters that are applied around each location in the feature map, yielding $(c + 4)kmn$ outputs for a $m \times n$ feature map. For an illustration of default boxes, please refer to Fig. 1. Our default boxes are similar to the *anchor boxes* used in Faster R-CNN [2], however we apply them to several feature maps of different resolutions. Allowing different default box shapes in several feature maps lets us efficiently discretize the space of possible output box shapes.

2.2 Training

The key difference between training SSD and training a typical detector that uses region proposals and pooling before a final classifier, is that ground truth information needs

to be assigned to specific outputs in the fixed set of detector outputs. Some version of this is also required for training in YOLO[5] and for the region proposal stages of Faster R-CNN[2] and MultiBox[7]. Once this assignment is determined, the loss function and back propagation are applied end-to-end. Training also involves choosing the set of default boxes and scales for detection as well as hard negative mining and data augmentation strategies.

Matching strategy At training time we need to establish the correspondence between the ground truth and the default boxes. Note that for each ground truth box we are selecting from default boxes that vary over location, aspect ratio, and scale. We begin by matching each ground truth box to the default box with the best jaccard overlap. This is the matching approach used by the original MultiBox [7] and it ensures that each ground truth box has exactly one matched default box. Unlike MultiBox, we then match default boxes to any ground truth with jaccard overlap higher than a threshold (0.5). Adding these matches simplifies the learning problem: it allows the network to predict high confidences for multiple overlapping default boxes rather than requiring it to pick only the one with maximum overlap.

Training objective The SSD training objective is derived from the MultiBox objective [7,8] but is extended to handle multiple object categories. Let's denote $x_{ij}^p = 1$ to indicate that the i-th default box is matched to the j-th ground truth box of category p, and $x_{ij}^p = 0$ otherwise. According to the matching strategy described above, we have $\sum_i x_{ij}^p \geq 1$, meaning there can be more than one default box matched to the j-th ground truth box. The overall objective loss function is a weighted sum of the localization loss (loc) and the confidence loss (conf):

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{1}$$

where N is the number of matched default boxes, and the localization loss is the Smooth L1 loss [6] between the predicted box (l) and the ground truth box (g) parameters. Similar to Faster R-CNN [2], we regress to offsets for the center of the bounding box and for its width and height. Our confidence loss is the softmax loss over multiple classes confidences (c) and the weight term α is set to 1 by cross validation.

Choosing scales and aspect ratios for default boxes Most convolutional networks reduce the size of feature maps at the deeper layers. Not only does this reduce computation and memory cost but it also provides some degree of translation and scale invariance. To handle different object scales, some methods [4,9] suggest converting the image to different sizes, then processing each size individually and combining the results afterwards. However, by utilizing feature maps from several different layers in a single network for prediction we can mimic the same effect, while also sharing parameters across all object scales. Previous works [10,11] have shown that using feature maps from the lower layers can improve semantic segmentation quality because the lower layers capture more fine details of the input objects. Similarly, [12] showed that adding

global context pooled from the topmost feature map can help smooth the segmentation results. Motivated by these methods, we use both the lower and upper feature maps for making detection predictions. Figure 1 shows two exemplar feature maps (8×8 and 4×4) which are used in the framework, of course in practice we can use many more with relatively small computational overhead.

Feature maps from different levels within a network are known to have different (empirical) receptive field sizes [13]. Fortunately, within the SSD framework, the default boxes do not necessary need to correspond to the actual receptive fields of each layer. We can design the tiling so that specific feature map locations learn to be responsive to specific areas of the image and particular scales of the objects. Suppose we want to use m feature maps to do the predictions. The scale of the default boxes for each feature map is computed as:

$$s_k = s_{\min} + \frac{s_{\max} - s_{\min}}{m - 1}(k - 1), \quad k \in [1, m] \tag{2}$$

where $s_{\min}$ is 0.2 and $s_{\max}$ is 0.95, meaning the lowest layer has a scale of 0.2 and the highest layer has a scale of 0.95, and all layers in between are regularly spaced. We impose different aspect ratios for the default boxes, and denote them as $a_r \in \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\}$. We can compute the width ($w_k^a = s_k\sqrt{a_r}$) and height ($h_k^a = s_k/\sqrt{a_r}$) for each default box. For the aspect ratio of 1, we also add a default box whose scale is $s'_k = \sqrt{s_k s_{k+1}}$, resulting in 6 default boxes per feature map location. We set the center of each default box to $(\frac{i+0.5}{|f_k|}, \frac{j+0.5}{|f_k|})$, where $|f_k|$ is the size of the k-th square feature map, $i, j \in [0, |f_k|)$, and we truncate the coordinates of the default boxes such that they are always within [0, 1]. In practice, one can also design a distribution of default boxes to best fit a specific dataset.

By combining predictions for all default boxes with different scales and aspect ratios from all locations of many feature maps, we have a diverse set of predictions, covering various input object sizes and shapes. For example, in Fig. 1, the dog is matched to a default box in the 4×4 feature map, but not to any default boxes in the 8×8 feature map. This is because those boxes have different scales and do not match the dog box, and therefore are considered as negatives during training.

Hard negative mining After the matching step, most of the default boxes are negatives, especially when the number of possible default boxes is large. This introduces a significant imbalance between the positive and negative training examples. Instead of using all the negative examples, we sort them using the highest confidence for each default box and pick the top ones so that the ratio between the negatives and positives is at most 3:1. We found that this leads to faster optimization and a more stable training.

Data augmentation To make the model more robust to various input object sizes and shapes, each training image is randomly sampled by one of the following options:

- Use the entire original input image.
- Sample a patch so that the *minimum* jaccard overlap with the objects is 0.1, 0.3, 0.5, 0.7, or 0.9.

– Randomly sample a patch.

The size of each sampled patch is [0.1, 1] of the original image size, and the aspect ratio is between $\frac{1}{2}$ and 2. We keep the overlapped part of the ground truth box if the center of it is in the sampled patch. After the aforementioned sampling step, each sampled patch is resized to fixed size and is horizontally flipped with probability of 0.5.

3 Experimental Results

Base network Our experiments are all based on VGG16 [14], which is pre-trained on the ILSVRC CLS-LOC dataset [15]. Similar to DeepLab-LargeFOV [16], we convert fc6 and fc7 to convolutional layers, subsample parameters from fc6 and fc7, change pool5 from $2 \times 2 - s2$ to $3 \times 3 - s1$, and use the atrous algorithm to fill the "holes". We remove all the dropout layers and the fc8 layer. We fine-tune the resulting model using SGD with initial learning rate 10^{-3}, 0.9 momentum, 0.0005 weight decay, and batch size 32. The learning rate decay policy is slightly different for each dataset, and we will describe details later. The full training and testing code is built on Caffe [17] and is open source at `https://github.com/weiliu89/caffe/tree/ssd`.

3.1 PASCAL VOC2007

On this dataset, we compare against Fast R-CNN [6] and Faster R-CNN [2]. All methods use the same training data and pre-trained VGG16 network. Specifically, we train on VOC2007 `trainval` and VOC2012 `trainval` (16551 images) and test on VOC2007 `test` (4952 images).

Figure 2 shows the architecture details of the SSD300 model. We use conv4_3, conv7 (fc7), conv8_2, conv9_2, conv10_2, and pool11 to predict both location and confidences[2]. We initialize the parameters for all the newly added convolutional layers with the "xavier" method [18]. Since the size of conv4_3 is big (38×38), we only place 3 default boxes on it – a box with scale 0.1 and two other boxes with aspect ratios of $\frac{1}{2}$ and 2. For all other layers, we put 6 default boxes on them as described in Sec. 2.2. Since, as pointed out in [12], conv4_3 has a different feature scale compared to the other layers, we use the L2 normalization technique introduced in [12] to scale the feature norm at each location in the feature map to 20 and learn the scale during back propagation. We use the 10^{-3} learning rate for 40k iterations, then we decay it to 10^{-4} and continue training for another 20k iterations. Table 1 shows that our SSD300 model is already more accurate than Fast R-CNN. When we train SSD on a larger 500×500 input image it is even more accurate, surpassing Faster R-CNN by 1.9% mAP.

To understand the performance of our two SSD models in more details, we used the detection analysis tool from [19]. Figure 3 shows that SSD can detect various object categories with high quality (large white area). The majority of its confident detections are correct. The recall is around 85-90%, and is much higher with "weak" (0.1 jaccard overlap) criteria. Compared to R-CNN [20], SSD has less localization error, indicating that SSD can localize objects better because it directly learns to regress the object shape

[2] For SSD500 model, we add extra conv11_2 for prediction

Method	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	70.0	77.0	78.1	69.3	59.4	38.3	81.6	78.6	86.7	42.8	78.8	68.9	84.7	82.0	76.6	69.9	31.8	70.1	**74.8**	80.4	70.4
Faster [2]	73.2	76.5	79.0	70.9	**65.5**	**52.1**	83.1	84.7	86.4	52.0	**81.9**	65.7	84.8	84.6	77.5	76.7	38.8	73.6	73.9	83.0	72.6
SSD300	72.1	75.2	**79.8**	70.5	62.5	41.3	81.1	80.8	86.4	51.5	74.3	**72.3**	83.5	84.6	80.6	74.5	46.0	71.4	73.8	83.0	69.1
SSD500	**75.1**	**79.8**	79.5	**74.5**	63.4	51.9	**84.9**	**85.6**	**87.2**	**56.6**	80.1	70.0	**85.4**	**84.9**	**80.9**	**78.2**	**49.0**	**78.4**	72.4	**84.6**	**75.5**

Table 1: **PASCAL VOC2007 `test` detection results.** Both Fast and Faster R-CNN use input images whose minimum dimension is 600. The two SSD models have exactly the same settings except that they have different input sizes (300×300 vs. 500×500). It is obvious that larger input size leads to better results.

and classify object categories instead of using two decoupled steps. However, SSD has more confusions with similar object categories (especially for animals), partly because we share locations for multiple categories.

Figure 4 shows that SSD is very sensitive to the bounding box size. In other words, it has much worse performance on smaller objects than bigger objects. This is not surprising because those small objects may not even have any information at the very top layers. Increasing the input size (e.g. from 300×300 to 500×500) can help improve detecting small objects, but there is still a lot of room to improve. On the positive side, we can clearly see that SSD performs really well on large objects. And it is very robust to different object aspect ratios because we use default boxes of various aspect ratios per feature map location.

3.2 Model analysis

To understand SSD better, we have also carried out several controlled experiments to examine how each component affects the final performance. For all of the following experiments, we use exactly the same settings and input size (300×300), except for the variable component.

	SSD300					
more data augmentation?		✔	✔	✔	✔	✔
use conv4_3?	✔		✔	✔	✔	✔
include $\{\frac{1}{2}, 2\}$ box?	✔	✔		✔	✔	✔
include $\{\frac{1}{3}, 3\}$ box?	✔	✔			✔	✔
use atrous?	✔	✔	✔	✔		✔
VOC2007 `test` mAP	65.4	68.1	69.2	71.2	71.4	72.1

Table 2: Effects of various design choices and components on SSD performance.

Data augmentation is crucial Fast and Faster R-CNN use the original image and the horizontal flip (with probability 0.5) to train. We use a more extensive sampling strategy, similar to YOLO [5], which also uses photometric distortions which we did not use. Table 2 shows that we can improve 6.7% mAP with this sampling strategy. We do not know how much our sampling strategy will benefit Fast and Faster R-CNN, but they are likely to benefit less because they use a feature pooling step during classification that is relatively robust to object translation by design.

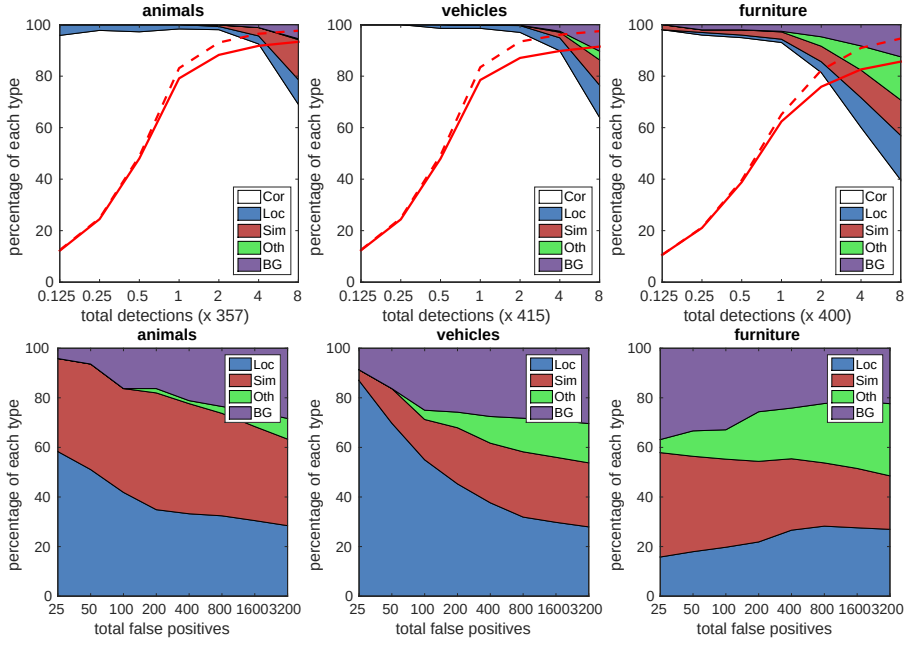


Fig. 3: **Visualization of performance for SSD 500 on animals, vehicles, and furniture from VOC2007 `test`.** The top row shows the cumulative fraction of detections that are correct (Cor) or false positive due to poor localization (Loc), confusion with similar categories (Sim), with others (Oth), or with background (BG). The solid red line reflects the change of recall with "strong" criteria (0.5 jaccard overlap) as the number of detections increases. The dashed red line is using the "weak" criteria (0.1 jaccard overlap). The bottom row shows the distribution of top-ranked false positive types.

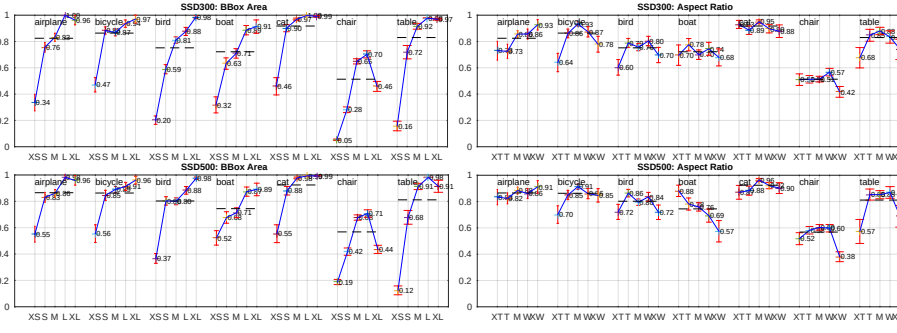


Fig. 4: **Sensitivity and impact of different object characteristics on VOC2007 `test` set.** Each plot shows the normalized AP [19] with standard error bars (red). Black dashed lines indicate overall normalized AP. The plot on the left shows the effects of BBox Area per category, and the right plot shows the effect of Aspect Ratio. Key: BBox Area: XS=extra-small; S=small; M=medium; L=large; XL =extra-large. Aspect Ratio: XT=extra-tall/narrow; T=tall; M=medium; W=wide; XW =extra-wide.

More feature maps is better Inspired by many works in semantic segmentation [10,11,12], we also use lower level feature maps for predicting bounding box outputs. We compare a model utilizing conv4_3 for prediction and a model without it. From Table 2, we can see that by adding conv4_3 for prediction, it has clearly better results (72.1% vs. 68.1%). This also matches our intuition that conv4_3 can capture better the fine-grained details of the objects, especially the small ones.

More default box shapes is better As described in Sec. 2.2, by default we use 6 default boxes per location. If we remove the boxes with $\frac{1}{3}$ and 3 aspect ratios, the performance drops by 0.9%. By further removing the boxes with $\frac{1}{2}$ and 2 aspect ratios, the performance drops another 2%. Using a variety of default box shapes seems to make the task of predicting boxes easier for the network.

Atrous is better and faster As described in Sec. 3, we used the atrous version of VGG16, following DeepLab-LargeFOV [16]. If we use the full VGG16, keeping pool5 with $2 \times 2 - s2$ and not subsampling parameters from fc6 and fc7, and add conv5_3 for prediction, the result is slightly worse (0.7%) while the speed is about 50% slower.

3.3 PASCAL VOC2012

We use the same settings as those used for VOC2007. This time, we use VOC2012 `trainval` and VOC2007 `trainval` and `test` (21503 images) for training, and test on VOC2012 `test` (10991 images). Since there is more training data, we train the model with 10^{-3} learning for 60k iterations and then decay it to 10^{-4} and continue training for another 20k iterations.

Table 3 shows the results of our SSD300 and SSD500[3] model. We see the same performance trend as we observed on VOC2007 test. Our SSD300 is already better than Fast R-CNN and very close to Faster R-CNN (only 0.1% difference). By increasing the training and testing image size to 500×500, we are 2.7% higher than Faster R-CNN. Compared to YOLO, SSD is significantly better, likely due to the use of convolutional default boxes from multiple feature maps and our matching strategy during training.

Method	*mAP*	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	**89.3**	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster [2]	70.4	**84.9**	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	**77.1**	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [5]	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300	70.3	84.2	76.3	69.6	53.2	40.8	78.5	73.6	88.0	50.5	73.5	**61.7**	85.8	80.6	81.2	77.5	44.3	73.2	**66.7**	81.1	65.8
SSD500	**73.1**	**84.9**	**82.6**	**74.4**	**55.8**	**50.0**	**80.3**	**78.9**	88.8	**53.7**	76.8	59.4	**87.6**	**83.7**	**82.6**	**81.4**	**47.2**	**75.5**	65.6	**84.3**	**68.1**

Table 3: **PASCAL VOC2012 `test` detection results.** Fast and Faster R-CNN use images with minimum dimension 600, while the image size for YOLO is 448×448.

[3] `http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?cls=mean&challengeid=11&compid=4`

3.4 MS COCO

To further validate the SSD framework, we trained our SSD300 and SSD500 architectures on the MS COCO dataset. Since objects in COCO tend to be smaller, we use smaller default boxes for all layers. We follow the strategy mentioned in Sec. 2.2, but now our smallest default box has a scale of 0.1 instead of 0.2, and the scale of the default box on conv4_3 is 0.07 (e.g. corresponding to 21 pixels for a 300×300 image).

We use the `trainval35k` [21] to train our model. Since COCO has more object categories, the gradient is not stable in the beginning. We first train the model with 8×10^{-4} learning rate for 4k iterations, followed by 10^{-3} learning rate for 140k iterations, and then continue training for 60k iterations with 10^{-4} and 40k iterations with 10^{-5}. Table 4 shows the results on `test-dev2015`. Similar to what we observed on the PASCAL VOC dataset, SSD300 is better than Fast R-CNN in both mAP@0.5 and mAP@[0.5:0.95]. SSD300 has a similar mAP@[0.5:0.95] to Faster R-CNN. However, the mAP@0.5 is worse and we conjecture that it is because the image size is too small, which prevents the model to localize many small objects accurately. By increasing the image size to 500×500, our SSD500 is better than Faster R-CNN in both criteria. In addition, our SSD500 model is also better than ION [21], a multi-scale version of Fast R-CNN with explicit modeling of context using a recurrent network. In Fig. 5, we show some detection examples on MS COCO `test-dev` with the SSD500 model.

Method	data	Average Precision		
		0.5	0.75	0.5:0.95
Fast R-CNN [6]	train	35.9	-	19.7
Faster R-CNN [2]	train	42.1	-	21.5
Faster R-CNN [2]	trainval	42.7	-	21.9
ION [21]	train	42.0	23.0	23.0
SSD300	trainval35k	38.0	20.5	20.8
SSD500	trainval35k	43.7	24.7	24.4

Table 4: **MS COCO `test-dev2015` detection results.**

3.5 Preliminary ILSVRC results

We applied the same network architecture we used for MS COCO to the ILSVRC DET dataset [15]. We train a SSD300 model using the ILSVRC2014 DET `train` and `val1` as used in [20]. We first train the model with 8×10^{-4} learning rate for 4k iterations, and train it with 10^{-3} learning rate for 320k iterations, and then continue training for 100k iterations with 10^{-4} and 60k iterations with 10^{-5}. We can achieve 41.1 mAP on the `val2` set [20]. Again, it validates that SSD is a general framework for high quality real-time detection.

3.6 Inference time

Considering the large number of boxes generated from our method, it is essential to perform non-maximum suppression (nms) efficiently during inference. By using a confidence threshold of 0.01, we can filter out most boxes. We then use the Thrust CUDA library for sorting, use a GPU implementation to compute overlap between all pairs of

the remaining boxes, and then apply nms with jaccard overlap of 0.45 per class and keep top 200 detections per image. This step costs about 2.2 msec per image for SSD300 for 20 VOC classes, which is close to the total time spent on all newly added layers.

Table 5 shows the comparison between SSD, Faster R-CNN[2], and YOLO[5]. Faster R-CNN uses extra prediction layers for region proposals and requires feature resampling. In contrast, our SSD500 method outperforms Faster R-CNN in both speed and accuracy. It is worth mentioning that our method SSD300 is the only real-time method to achieve above 70% mAP. Although Fast YOLO[5] can run at 155 FPS, this verion has much lower accuracy by almost 20% mAP.

Method	*mAP*	**FPS**	**# Boxes**
Faster R-CNN [2](VGG16)	73.2	7	300
Faster R-CNN [2](ZF)	62.1	17	300
YOLO [5]	63.4	45	98
Fast YOLO [5]	52.7	155	98
SSD300	72.1	58	7308
SSD500	**75.1**	23	20097

Table 5: **Results on Pascal VOC2007 `test`.** SSD300 is the only real-time detection method that can achieve above 70% mAP. By using a larger input image, SSD500 outperforms all methods on accuracy while maintaining a close to real-time speed. The speed of SSD models is measured with batch size of 8.

4 Related Work

There are two established classes of methods for object detection in images, one based on sliding windows and the other based on region proposal classification. Before the advent of convolutional neural networks, the state of the art for those two approaches – Deformable Part Model (DPM) [22] and Selective Search [1] – had comparable performance. However, after the dramatic improvement brought on by R-CNN [20], which combines selective search region proposals and convolutional network based post-classification, region proposal object detection methods became prevalent.

The original R-CNN approach has been improved in a variety of ways. The first set of approaches improve the quality and speed of post-classification, since it requires the classification of thousands of image crops, which is expensive and time-consuming. SPPnet [9] speeds up the original R-CNN approach significantly. It introduces a spatial pyramid pooling layer that is more robust to region size and scale and allows the classification layers to reuse features computed over feature maps generated at several image resolutions. Fast R-CNN [6] extends SPPnet so that it can fine-tune all layers end-to-end by minimizing a loss for both confidences and bounding box regression, which was first introduced in MultiBox [7] for learning objectness.

The second set of approaches improve the quality of proposal generation using deep neural networks. In the most recent works like MultiBox [7,8], the Selective Search region proposals, which are based on low-level image features, are replaced by proposals generated directly from a separate deep neural network. This further improves the detection accuracy but results in a somewhat complex setup, requiring the training of two neural networks with a dependency between them. Faster R-CNN [2] replaces

selective search proposals by ones learned from a region proposal network (RPN), and introduces a method to integrate the RPN with Fast R-CNN by alternating between fine-tuning shared convolutional layers and prediction layers for these two networks. This way region proposals are used to pool mid-level features and the final classification step is less expensive. Our SSD is very similar to the region proposal network (RPN) in Faster R-CNN in that we also use a fixed set of (default) boxes for prediction, similar to the achor boxes in the RPN. But instead of using these to pool features and evaluate another classifier, we simultaneously produce a score for each object category in each box. Thus, our approach avoids the complication of merging RPN with Fast R-CNN and is easier to train, faster, and straightforward to integrate in other tasks.

Another set of methods, which are directly related to our approach, skip the proposal step altogether and predict bounding boxes and confidences for multiple categories directly. OverFeat [4], a deep version of the sliding window method, predicts a bounding box directly from each location of the topmost feature map after knowing the confidences of the underlying object categories. YOLO [5] uses the whole topmost feature map to predict both confidences for multiple categories and bounding boxes (which are shared for these categories). Our SSD method falls in this category because we do not have the proposal step but use the default boxes. However, our approach is more flexible than the existing methods because we can use default boxes of different aspect ratios on each feature location from multiple feature maps at different scales. If we only use one default box per location from the topmost feature map, our SSD would have similar architecture to OverFeat [4]; if we use the whole topmost feature map and add a fully connected layer for predictions instead of our convolutional predictors, and do not explicitly consider multiple aspect ratios, we can approximately reproduce YOLO [5].

5 Conclusions

This paper introduces SSD, a fast single-shot object detector for multiple categories. A key feature of our model is the use of multi-scale convolutional bounding box outputs attached to multiple feature maps at the top of the network. This representation allows us to efficiently model the space of possible box shapes. We experimentally validate that given appropriate training strategies, a larger number of carefully chosen default bounding boxes results in improved performance. We build SSD models with at least an order of magnitude more box predictions sampling location, scale, and aspect ratio, than the existing methods [2,5,7].

We demonstrate that given the same VGG-16 base architecture, SSD compares favorably to its state-of-the-art object detector counterparts in terms of both accuracy and speed. Our SSD500 model significantly outperforms the state-of-the-art Faster R-CNN [2] in terms of accuracy on PASCAL VOC and MS COCO, while being 3x faster. Our real time SSD300 model runs at 58 FPS, which is faster than the current real time YOLO [5] alternative, while producing markedly superior detection quality.

Apart from its standalone utility, we believe that our monolithic and relatively simple SSD model provides a great building block for larger systems that employ an object detection component. A promising future direction is to explore its use as part of a system using recurrent neural networks to detect and track objects in video.

6 Acknowledgment

This project was started as an intern project at Google and continued at UNC. We would like to thank Alex Toshev for helpful discussions and are indebted to the Image Understanding and DistBelief teams at Google. We also thank Philip Ammirato and Patrick Poirson for helpful comments. We thank NVIDIA for providing K40 GPUs and acknowledge support from NSF 1452851.

Fig. 5: Detection examples on MS COCO `test-dev` with SSD500 model. We show detections with scores higher than 0.6. Each color corresponds to an object category.

References

1. Uijlings, J.R., van de Sande, K.E., Gevers, T., Smeulders, A.W.: Selective search for object recognition. IJCV (2013)
2. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards real-time object detection with region proposal networks. In: NIPS. (2015)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
4. Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. In: ICLR. (2014)
5. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: CVPR. (2016)
6. Girshick, R.: Fast R-CNN. In: ICCV. (2015)
7. Erhan, D., Szegedy, C., Toshev, A., Anguelov, D.: Scalable object detection using deep neural networks. In: CVPR. (2014)
8. Szegedy, C., Reed, S., Erhan, D., Anguelov, D.: Scalable, high-quality object detection. arXiv preprint arXiv:1412.1441 v3 (2015)
9. He, K., Zhang, X., Ren, S., Sun, J.: Spatial pyramid pooling in deep convolutional networks for visual recognition. In: ECCV. (2014)
10. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: CVPR. (2015)
11. Hariharan, B., Arbeláez, P., Girshick, R., Malik, J.: Hypercolumns for object segmentation and fine-grained localization. In: CVPR. (2015)
12. Liu, W., Rabinovich, A., Berg, A.C.: ParseNet: Looking wider to see better. In: ILCR. (2016)
13. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A.: Object detectors emerge in deep scene cnns. In: ICLR. (2015)
14. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: NIPS. (2015)
15. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Li, F.F.: Imagenet large scale visual recognition challenge. IJCV (2015)
16. Chen, L.C., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Semantic image segmentation with deep convolutional nets and fully connected crfs. In: ICLR. (2015)
17. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding. In: MM, ACM (2014)
18. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: AISTATS. (2010)
19. Hoiem, D., Chodpathumwan, Y., Dai, Q.: Diagnosing error in object detectors. In: ECCV 2012. (2012)
20. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: CVPR. (2014)
21. Bell, S., Zitnick, C.L., Bala, K., Girshick, R.: Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks. In: CVPR. (2016)
22. Felzenszwalb, P., McAllester, D., Ramanan, D.: A discriminatively trained, multiscale, deformable part model. In: CVPR. (2008)