SSD
: Single Shot MultiBox Detector (2015)

Seho Kim

https://arxiv.org/pdf/1512.02325.pdf

Introduction

- There are variants of the following approach

  : hypothesize bounding boxes, resample pixels or features for each box, and apply a high quality classifier
- While accurate, these approaches have been too computationally intensive and too slow for real-time applications
- SSD (59 FPS with mAP 74.3% on VOC2007 test)
- Faster R-CNN (7 FPS with mAP 73.2%)
- YOLO (45 FPS with mAP 63.4%)

Introduction

- Improvements:
  - Fundamental improvement reason: Eliminating bounding box proposal, Subsequent pixel or feature resampling stage
  - Using a small convolutional filter to predict object categories and offsets in bounding box locations
  - Using separate predictors (filters) for different aspect ratio detections; applying filters to multiple feature maps from the later stage of a network in order to perform detection at multiple scales

    → Achieve high-accuracy using relatively low resolution input, further increasing detection speed

Introduction

- Contributions:
  - Faster than SOTA for single shot detectors (YOLO), and **more accurate** (as accurate as slower techniques(region proposals and pooling; Faster R-CNN))
  - The core of SSD is predicting category scores and box offsets for a fixed set of default bounding boxes using **small convolutional filters** applied to feature maps
  - Produce predictions of **different scales** from feature maps of different scales, and explicitly separate predictions by **aspect ratio**(for high detection accuracy)
  - **End-to-end** training and high accuracy, low resolution input images, improving the speed vs accuracy trade-off
  - Experiments:
    - Timing and accuracy analysis; varying input size evaluated on PASCAL VOC, COCO, and ILSVRC
    - Compared to a range of SOTA approaches

The Single Shot Detector (SSD)

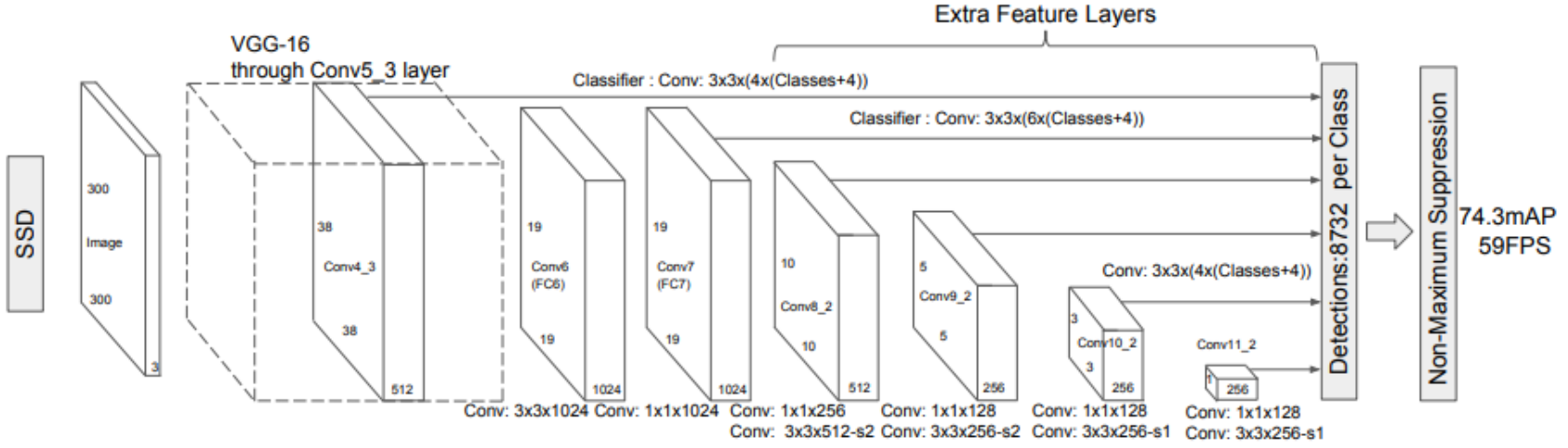
- Model
  - Based on a **feed-forward convolutional network**; produces a fixed-size collection of bounding boxes and scores for the presence of object class instance in those boxes → **NMS**(Non-Maximum Suppression) step to produce the final detections
  - Early network layer are based on a standard architecture; **base network**(ex. VGG-16)

    → **Add auxiliary structure** to the network to produce detections with the following key features

The Single Shot Detector (SSD)

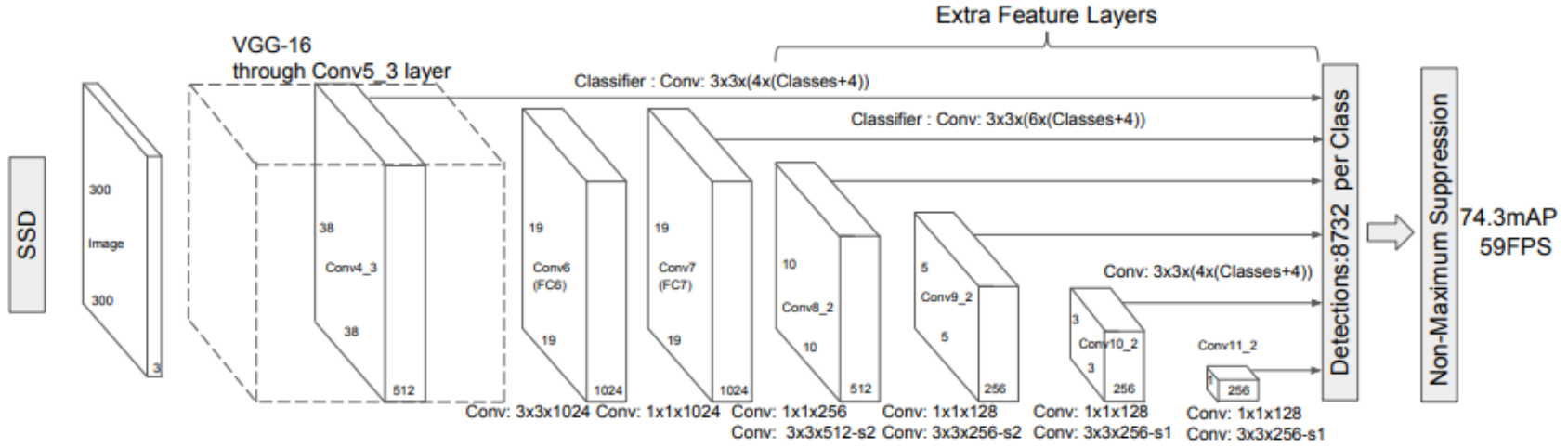
- Model
  - **Multi-scale** feature maps for detection
    - Add convolutional feature layers to the end of the truncated base network; allow predictions of detections at multiple scales

      cf. Overfeat and YOLO that operate on a single scale feature map
  - Convolutional predictors for detection
    - Each added feature layer can produce a fixed set of detection predictions using a set of convolutional filters



The Single Shot Detector (SSD)

- Model
  - Convolutional predictors for detection
    - Bounding box offset output values are measured relative to a default box position relative to each feature map location (cf. YOLO uses an intermediate FC layer)



The Single Shot Detector (SSD)

- Model
  - **Default boxes** and **aspect ratios**
    - Default boxes are similar to the anchor boxes (used in Faster R-CNN)
    - 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
    - Outputs for a mxn feature map: (c + 4)kmn

      ; c class score, k the number of default boxes

The Single Shot Detector (SSD)

- Training
  - Matching strategy
    - Determine which default boxes correspond to a ground truth detection; use best jaccard overlap
  - Training objective (weight term α; 1 by cross validation)

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

    - Localization loss(loc)

      ; N is the number of matched default boxes, prdicted box (l), ground truth box (g), center (cx, cy), default bounding box (d), width (w), height (h)

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

The Single Shot Detector (SSD)

- Training
  - Training objective
    - Confidence loss(conf)

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

  - Choosing scales and aspect ratios for default boxes
    - The scale of the default boxes for each feature map

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

$$a_r \in \{1, 2, 3, \tfrac{1}{2}, \tfrac{1}{3}\}$$

$$(w_k^a = s_k\sqrt{a_r})$$

$$(h_k^a = s_k/\sqrt{a_r})$$

$$(\tfrac{i+0.5}{|f_k|}, \tfrac{j+0.5}{|f_k|})$$ $|f_k|$ is the size of the k-th square feature map, $i, j \in [0, |f_k|)$

The Single Shot Detector (SSD)

- Training
  - Hard negative mining
    - For imbalance between the positive and negative training examples
    - Sort negative training examples using the highest confidence loss for each default box and pick the top ones so that the ratio between the negatives and positives is at most 3:1
    - Lead to faster optimization and a more stable training

The Single Shot Detector (SSD)

- Training
  - Data augmentation
    - 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 path is [0.1, 1] of the original image size, and the aspect ratio is between ½ and 2

      → resize, horizontal flip with probability of 0.5, apply some photo-metric distortions

Experimental Results

- PASCAL VOC2007

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast [6]	07	66.9	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8
Fast [6]	07+12	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]	07	69.9	70.0	80.6	70.1	57.3	49.9	78.2	80.4	82.0	52.2	75.3	67.2	80.3	79.8	75.0	76.3	39.1	68.3	67.3	81.1	67.6
Faster [2]	07+12	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
Faster [2]	07+12+COCO	78.8	84.3	82.0	77.7	68.9	65.7	88.1	88.4	88.9	63.6	86.3	70.8	85.9	87.6	80.1	82.3	53.6	80.4	75.8	86.6	78.9
SSD300	07	68.0	73.4	77.5	64.1	59.0	38.9	75.2	80.8	78.5	46.0	67.8	69.2	76.6	82.1	77.0	72.5	41.2	64.2	69.1	78.0	68.5
SSD300	07+12	74.3	75.5	80.2	72.3	66.3	47.6	83.0	84.2	86.1	54.7	78.3	73.9	84.5	85.3	82.6	76.2	48.6	73.9	76.0	83.4	74.0
SSD300	07+12+COCO	79.6	80.9	86.3	79.0	**76.2**	57.6	87.3	88.2	88.6	60.5	85.4	**76.7**	**87.5**	**89.2**	84.5	81.4	55.0	81.9	**81.5**	85.9	78.9
SSD512	07	71.6	75.1	81.4	69.8	60.8	46.3	82.6	84.7	84.1	48.5	75.0	67.4	82.3	83.9	79.4	76.6	44.9	69.9	69.1	78.1	71.8
SSD512	07+12	76.8	82.4	84.7	78.4	73.8	53.2	86.2	87.5	86.0	57.8	83.1	70.2	84.9	85.2	83.9	79.7	50.3	77.9	73.9	82.5	75.3
SSD512	07+12+COCO	**81.6**	**86.6**	**88.3**	**82.4**	76.0	**66.3**	**88.6**	**88.9**	**89.1**	**65.1**	**88.4**	73.6	86.5	88.9	**85.3**	**84.6**	**59.1**	**85.0**	80.4	**87.4**	**81.2**

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. 512×512). It is obvious that larger input size leads to better results, and more data always helps. Data: "07": VOC2007 `trainval`, "07+12": union of VOC2007 and VOC2012 `trainval`. "07+12+COCO": first train on COCO `trainval35k` then fine-tune on 07+12.

Experimental Results

- Model analysis

	SSD300				
more data augmentation?		✔	✔	✔	✔
include $\{\frac{1}{2}, 2\}$ box?	✔		✔	✔	✔
include $\{\frac{1}{3}, 3\}$ box?	✔			✔	✔
use atrous?	✔	✔	✔		✔
VOC2007 `test` mAP	65.5	71.6	73.7	74.2	**74.3**

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

- Data augmentation is crucial
- More default box shapes is better
- Atrous is faster

Prediction source layers from:						mAP use boundary boxes?		# Boxes
conv4_3	conv7	conv8_2	conv9_2	conv10_2	conv11_2	Yes	No	
✔	✔	✔	✔	✔	✔	74.3	63.4	8732
✔	✔	✔	✔	✔		**74.6**	63.1	8764
✔	✔	✔	✔			73.8	68.4	8942
✔	✔	✔				70.7	69.2	9864
✔	✔					64.2	64.4	9025
	✔					62.4	64.0	8664

Table 3: **Effects of using multiple output layers.**

- Multiple output layers at different resolutions is better

Experimental Results

- PASCAL VOC2012

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast[6]	07++12	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]	07++12	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
Faster[2]	07++12+COCO	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
YOLO[5]	07++12	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	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD300	07++12+COCO	77.5	90.2	83.3	76.3	63.0	53.6	83.8	82.8	92.0	59.7	82.7	63.5	89.3	87.6	85.9	84.3	52.6	82.5	**74.1**	**88.4**	74.2
SSD512	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
SSD512	07++12+COCO	**80.0**	**90.7**	**86.8**	**80.5**	**67.8**	**60.8**	**86.3**	**85.5**	**93.5**	**63.2**	**85.7**	**64.4**	**90.9**	**89.0**	**88.9**	**86.8**	**57.2**	**85.1**	72.8	**88.4**	**75.9**

Table 4: **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. data: "07++12": union of VOC2007 `trainval` and `test` and VOC2012 `trainval`. "07++12+COCO": first train on COCO `trainval35k` then fine-tune on 07++12.

Experimental Results

- COCO

Method	data	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
Fast [6]	train	19.7	35.9	-	-	-	-	-	-	-	-	-	-
Fast [24]	train	20.5	39.9	19.4	4.1	20.0	35.8	21.3	29.5	30.1	7.3	32.1	52.0
Faster [2]	trainval	21.9	42.7	-	-	-	-	-	-	-	-	-	-
ION [24]	train	23.6	43.2	23.6	6.4	24.1	38.3	23.2	32.7	33.5	10.1	37.7	53.6
Faster [25]	trainval	24.2	45.3	23.5	7.7	26.4	37.1	23.8	34.0	34.6	12.0	38.5	54.4
SSD300	trainval35k	23.2	41.2	23.4	5.3	23.2	39.6	22.5	33.2	35.3	9.6	37.6	56.5
SSD512	trainval35k	**26.8**	**46.5**	**27.8**	**9.0**	**28.9**	**41.9**	**24.8**	**37.5**	**39.8**	**14.0**	**43.5**	**59.0**

Table 5: **COCO `test-dev2015` detection results.**

Experimental Results

- Preliminary ILSVRC results
- Data Augmentation for Small Object Accuracy

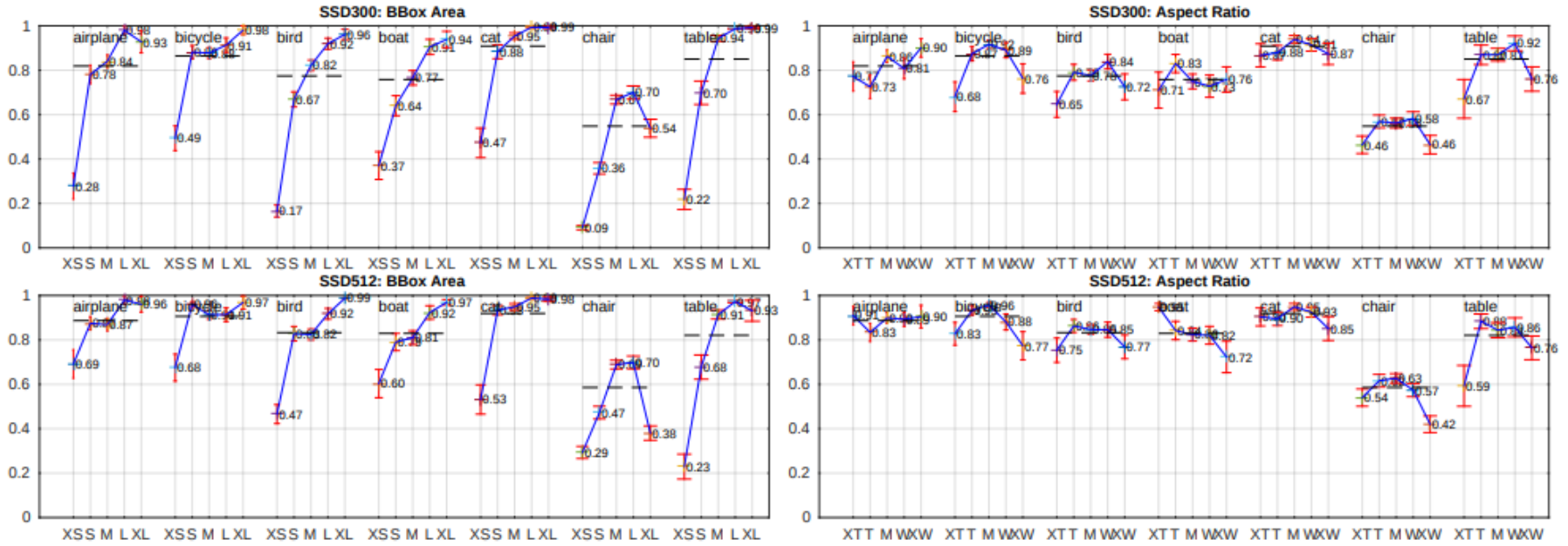


Fig. 4: **Sensitivity and impact of different object characteristics on VOC2007 `test` set using [21].** The plot on the left shows the effects of BBox Area per category, and the right plot shows the effect of Aspect Ratio. Key: BBox Area: XS=extra-small; S=small; M=medium; L=large; XL =extra-large. Aspect Ratio: XT=extra-tall/narrow; T=tall; M=medium; W=wide; XW =extra-wide.

Experimental Results

- Data Augmentation for Small Object Accuracy

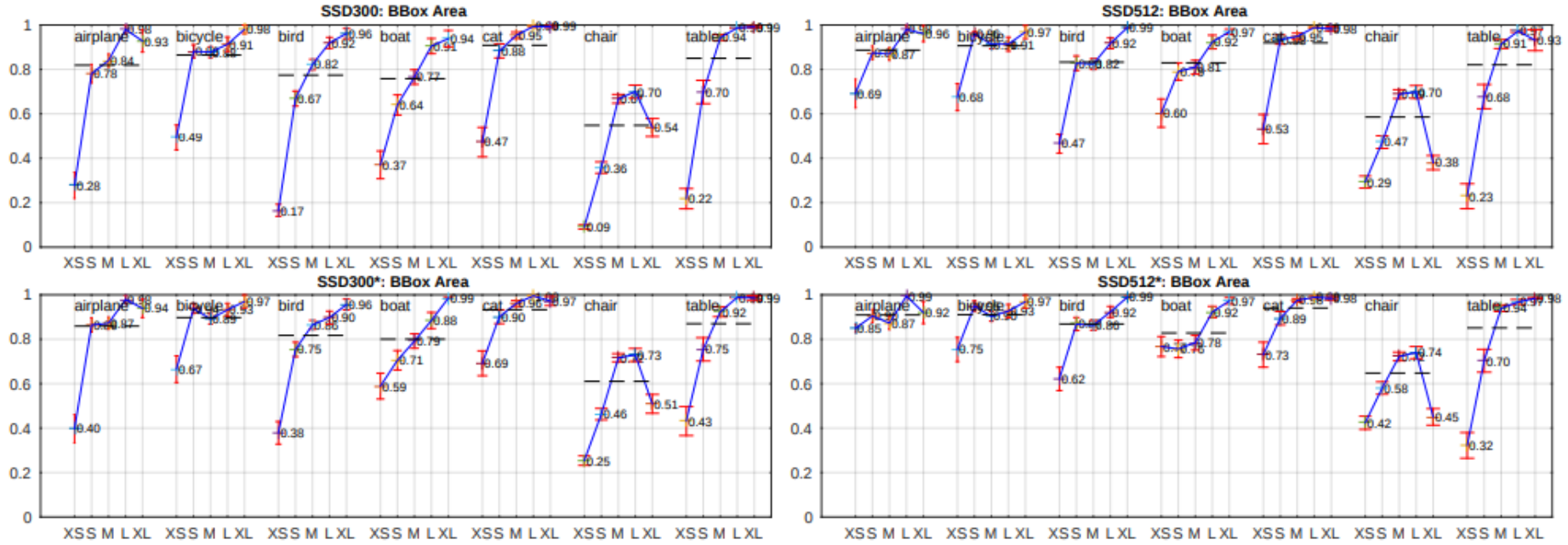


Fig. 6: **Sensitivity and impact of object size with new data augmentation on VOC2007 `test` set using [21].** The top row shows the effects of BBox Area per category for the original SSD300 and SSD512 model, and the bottom row corresponds to the SSD300* and SSD512* model trained with the new data augmentation trick. It is obvious that the new data augmentation trick helps detecting small objects significantly.

Experimental Results

- Data Augmentation for Small Object Accuracy

Method	VOC2007 test		VOC2012 test		COCO test-dev2015 trainval35k		
	07+12	07+12+COCO	07++12	07++12+COCO			
	0.5	0.5	0.5	0.5	0.5:0.95	0.5	0.75
SSD300	74.3	79.6	72.4	77.5	23.2	41.2	23.4
SSD512	76.8	81.6	74.9	80.0	26.8	46.5	27.8
SSD300*	77.2	81.2	75.8	79.3	25.1	43.1	25.8
SSD512*	**79.8**	**83.2**	**78.5**	**82.2**	**28.8**	**48.5**	**30.3**

Table 6: **Results on multiple datasets when we add the image expansion data augmentation trick.** SSD300* and SSD512* are the models that are trained with the new data augmentation.

Experimental Results

- Inference time

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	$\sim 1000 \times 600$
Fast YOLO	52.7	155	1	98	448×448
YOLO (VGG16)	66.4	21	1	98	448×448
SSD300	74.3	46	1	8732	300×300
SSD512	76.8	19	1	24564	512×512
SSD300	74.3	59	8	8732	300×300
SSD512	76.8	22	8	24564	512×512

Table 7: **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, SSD512 outperforms all methods on accuracy while maintaining a close to real-time speed.

Related Work

- Two established classes of methods for object detection in images(Based on sliding windows, region proposal classification)
- Before the advent of CNN, Two approaches(Deformable Part Model(DPM), Selective Search)
- After R-CNN, region proposal object detection method became prevalent
  - SPPnet, Fast R-CNN, MultiBox
  - Using deep neural networks(RPN, Faster R-CNN)
- OverFeat, YOLO

Conclusions

```python
class SSD300(nn.Module):
    steps = (8, 16, 32, 64, 100, 300)
    box_sizes = (30, 60, 111, 162, 213, 264, 315)  # default bounding box sizes for each feature map.
    aspect_ratios = ((2,), (2,3), (2,3), (2,3), (2,), (2,))
    fm_sizes = (38, 19, 10, 5, 3, 1)

    def __init__(self, num_classes):
        super(SSD300, self).__init__()
        self.num_classes = num_classes
        self.num_anchors = (4, 6, 6, 6, 4, 4)
        self.in_channels = (512, 1024, 512, 256, 256, 256)

        self.extractor = VGG16Extractor300()
        self.loc_layers = nn.ModuleList()
        self.cls_layers = nn.ModuleList()
        for i in range(len(self.in_channels)):
                self.loc_layers += [nn.Conv2d(self.in_channels[i], self.num_anchors[i]*4, kernel_size=3, padding=1)]
                self.cls_layers += [nn.Conv2d(self.in_channels[i], self.num_anchors[i]*self.num_classes, kernel_size=3, padding=

    def forward(self, x):
        loc_preds = []
        cls_preds = []
        xs = self.extractor(x)
        for i, x in enumerate(xs):
            loc_pred = self.loc_layers[i](x)
            loc_pred = loc_pred.permute(0,2,3,1).contiguous()
            loc_preds.append(loc_pred.view(loc_pred.size(0),-1,4))

            cls_pred = self.cls_layers[i](x)
            cls_pred = cls_pred.permute(0,2,3,1).contiguous()
            cls_preds.append(cls_pred.view(cls_pred.size(0),-1,self.num_classes))

        loc_preds = torch.cat(loc_preds, 1)
        cls_preds = torch.cat(cls_preds, 1)
        return loc_preds, cls_preds
```

https://github.com/kuangliu/torchcv

```python
class L2Norm(nn.Module):
    '''L2Norm layer across all channels.'''
    def __init__(self, in_features, scale):
        super(L2Norm, self).__init__()
        self.weight = nn.Parameter(torch.Tensor(in_features))
        self.reset_parameters(scale)

    def reset_parameters(self, scale):
        nn.init.constant(self.weight, scale)

    def forward(self, x):
        x = F.normalize(x, dim=1)
        scale = self.weight[None,:,None,None]
        return scale * x


class VGG16Extractor300(nn.Module):
    def __init__(self):
        super(VGG16Extractor300, self).__init__()

        self.features = VGG16()
        self.norm4 = L2Norm(512, 20)

        self.conv5_1 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1
        self.conv5_2 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1
        self.conv5_3 = nn.Conv2d(512, 512, kernel_size=3, padding=1, dilation=1

        self.conv6 = nn.Conv2d(512, 1024, kernel_size=3, padding=6, dilation=6)
        self.conv7 = nn.Conv2d(1024, 1024, kernel_size=1)

        self.conv8_1 = nn.Conv2d(1024, 256, kernel_size=1)
        self.conv8_2 = nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1)

        self.conv9_1 = nn.Conv2d(512, 128, kernel_size=1)
        self.conv9_2 = nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1)

        self.conv10_1 = nn.Conv2d(256, 128, kernel_size=1)
        self.conv10_2 = nn.Conv2d(128, 256, kernel_size=3)

        self.conv11_1 = nn.Conv2d(256, 128, kernel_size=1)
        self.conv11_2 = nn.Conv2d(128, 256, kernel_size=3)

    def forward(self, x):
        hs = []
        h = self.features(x)
        hs.append(self.norm4(h))  # conv4_3

        h = F.max_pool2d(h, kernel_size=2, stride=2, ceil_mode=True)

        h = F.relu(self.conv5_1(h))
        h = F.relu(self.conv5_2(h))
        h = F.relu(self.conv5_3(h))
        h = F.max_pool2d(h, kernel_size=3, stride=1, padding=1, ceil_mode=True)

        h = F.relu(self.conv6(h))
        h = F.relu(self.conv7(h))
        hs.append(h)  # conv7

        h = F.relu(self.conv8_1(h))
        h = F.relu(self.conv8_2(h))
        hs.append(h)  # conv8_2

        h = F.relu(self.conv9_1(h))
        h = F.relu(self.conv9_2(h))
        hs.append(h)  # conv9_2

        h = F.relu(self.conv10_1(h))
        h = F.relu(self.conv10_2(h))
        hs.append(h)  # conv10_2

        h = F.relu(self.conv11_1(h))
        h = F.relu(self.conv11_2(h))
        hs.append(h)  # conv11_2
        return hs
```

https://github.com/kuangliu/torchcv

```python
class VGG16(nn.Module):
    def __init__(self):
        super(VGG16, self).__init__()
        self.layers = self._make_layers()

    def forward(self, x):
        y = self.layers(x)
        return y

    def _make_layers(self):
        '''VGG16 layers.'''
        cfg = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512]
        layers = []
        in_channels = 3
        for x in cfg:
            if x == 'M':
                layers += [nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)]
            else:
                layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                           nn.ReLU(True)]
                in_channels = x
        return nn.Sequential(*layers)
```

https://github.com/kuangliu/torchcv