Object Detection

Cis680
“What & Where” Visual Pathways

- Established with electrophysiology, lesion, neuropsychology and neuroimaging data
Monkey Lesion Data

- **Two types of Delayed Response Task**
- **Monkeys trained to criterion on one of these tasks**
- **Then task was reversed**
- **After learning, either temporal or parietal lobe lesioned**
Effects of Lesion on Landmark Task

• Unoperated monkeys show no impairment
• Temporal-lobe lesion monkeys show minimal impairment
• Parietal-lobe lesion monkeys show much impairment
Effects of Lesion on Object Task

• Temporal-lobe lesion monkeys show much impairment
• Parietal-lobe lesion monkeys show minimal impairment
Monkey Lesion Data

• Subsequent lesion work supports the “what-where” distinction
• Object discrimination: Ventral lesion deficits restricted to visual modality
• Posterior/Anterior Ventral Lobe distinction:
  • Posterior: Visual discrimination
  • Anterior: Visual memory
The What-Where Distinction: Human Neuroimaging

Object task:
Same objects?

Spatial Task”
Same locations?

• Data indicate evidence for what-where distinction
Visual crowding—the deleterious effect of clutter on peripheral object recognition—is ubiquitous in natural scenes. A. It seriously impacts virtually all everyday tasks including reading, driving, and interacting with the environment. For example, fixating the bull’s-eye, near the construction zone, note that it is difficult or impossible to recognize the child on the left side of the road, simply because of the presence of the nearby signs. The child on the right, on the other hand, is relatively easier to recognize. B. While fixating the crosses, identifying the middle shape, letter, or line orientation—or even the number of tilted lines—is difficult or impossible on the bottom half of the panel. Crowding impairs the ability to recognize and scrutinize objects, but it does not make them disappear; one can see that some thing is present in panel (A), but it is difficult to identify the thing as a child as opposed to another sign. Crowding defines the spatial resolution of conscious object recognition throughout most of the visual field.
Region based classification: Proposal (where) and Checking (what)

R-CNN: Regions with CNN Features (Girshick et al., 2014)
Object Detection

R-CNN: Regions with CNN Features (Girshick et al., 2014)
Drawback of R-CNN and the modification:
1. Training is a multi-stage pipeline. -> End-to-end joint training.
2. Training is expensive in space and time. -> Convolutional layer sharing. Classification in memory.
For SVM and regressor training, features are extracted from each warped object proposal in each image and written to disk. (VGG16, 5k VOC07 trainval images: 2.5 GPU days). Hundreds of gigabytes of storage.
3. Test-time detection is slow. -> Single scale testing, SVD fc layer.
At test-time, features are extracted from each warped proposal in each img. (VGG16: 47s / image).

Contributions:
1. Higher detection quality (mAP) than R-CNN
2. Training is single-stage, using a multi-task loss
3. All network layers can be updated during training
4. No disk storage is required for feature caching
Object Detection

Log loss + smooth L1 loss

Linear + softmax

Linear

FCs

ConvNet

Trainable

Multi-task loss

Fast R-CNN (Girshick, 2015)
Mini-batch Sampling
128: 2 randomly sampled images with 64 PoI sampled from each image
25% positive: IoU > 0.5
75% background: IoU in [0.1, 0.5)
horizontally flipped with prob = 0.5

Truncated SVD for faster detection
mAP ~ 0.3% down; speed ~ 30% up
number of RoI for detection is large -> time spent on fc
$W \sim U \Sigma_t V^T$ (U: u*t, Sigma_t: t*t, V: v*t)
Compression : $(Wx + b)$ fc -> $(\Sigma_t V^T x)$ fc + $(Ux + b)$ fc

Forward pass timing
mAP 66.9% @ 320ms / image
fc6: 38.7% (122ms)
fc7: 46.3% (146ms)
other: 5.4% (17ms)
roi_pool5: 5.5% (11ms)

Forward pass timing (SVD)
mAP 66.6% @ 223ms / image
fc6: 17.5% (37ms)
fc7: 1.7% (4ms)
other: 7.9% (17ms)
roi_pool5: 5.1% (11ms)

Faster **RCNN**

Don’t need to have external regional proposals

RPN - Regional Proposal Network

Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks  
Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun  
Jun 2015

What is Faster R-CNN?

- Presented in Kaiming’s section

Review:
Faster R-CNN = Fast R-CNN + Region Proposal Networks

- Does not depend on an external region proposal algorithm
- Does object detection in a single forward pass
Training goal: Share features

Region Proposal Network

RPN proposals

proposals from any algorithm

Goal: Share so

CNN A == CNN B
One'net,'four'losses

Model:
- CNN
  - Feature map
  - Region Proposal Network
  - Proposals
- RoI Pooling

Losses:
- Classification loss
- Bounding box regression loss

Regions:
- Bounding box
- Classification

Diagram:
- Image
- CNN
- Feature map
- Region Proposal Network
- Proposals
- RoI Pooling
- Losses

---

facebook

ICCV'15
Translation-Invariant Anchors
At each sliding window loc, predict k proposals: 4k outputs for reg layer, 2k outputs for cls layer (binary softmax).
Anchor: centered at sliding window with scale and aspect ratio: \((128^2, 256^2, 512^2; 1:2, 2:1, 1:1)\)
For a conv feature map: \(W \times H \times k\) (k=9 anchors) \((2+4) \times 9\) output layer

Optimization
fcn trained by end-to-end by bp and sgd
image-centric sampling strategy, sample 256 anchors in an image (Pos:neg = 1:1)
nueva layer initialization \(\sim N(0, 0.01)\)
tune ZFnet and conv3_1 and up for VGGnet, lr=0.001 for 60k batches, 0.0001 for 20k on PASCAL
• Loss function for Learning Region Proposal

  positive label: the anchor has highest IoU with a gt-box or has an IoU > 0.7 with any gt-box
  negative label: IoU < 0.3 for all gt-box

Objective function with multi-task loss: Similar to Fast R-CNN.

\[
L(p_i, t_i) = L_{cls}(p_i, p_i^*) + \lambda p_i^* L_{reg}(t_i, t_i^*)
\]

where \(p_i^*\) is 1 if the anchor is labeled positive, and is 0 if the anchor is negative.
\(\lambda = 10\) bias towards better box location
- **Share Convolutional Features for Region Proposal and Objection Detection**

  Four-step training algorithm:
  1. Train RPN, initialized with ImageNet pre-trained model
  2. Train a separate detection network by Fast R-CNN using proposals generated by step-1 RPN, initialized by ImageNet pre-trained model
  3. Fix conv layer, fine-tune unique layers to RPN, initialized by detector network in Step2
  4. Fix conv layer, fine-tune fc-layers of Fast R-CNN
## Result compare

<table>
<thead>
<tr>
<th></th>
<th>Faster R-CNN</th>
<th>Fast R-CNN</th>
<th>R-CNN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test time/image</td>
<td>0.2S</td>
<td>2s</td>
<td>50s</td>
</tr>
<tr>
<td>With proposal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-test speedup</td>
<td>250x</td>
<td>25x</td>
<td>1x</td>
</tr>
<tr>
<td>mAP</td>
<td>66.9</td>
<td>66.9</td>
<td>66</td>
</tr>
</tbody>
</table>

Trained using Pascal VOC 2007 dataset

Source: cs231 stanford
Table 5: **Timing** (ms) on a K40 GPU, except SS proposal is evaluated in a CPU. “Region-wise” includes NMS, pooling, fully-connected, and softmax layers. See our released code for the profiling of running time.

<table>
<thead>
<tr>
<th>model</th>
<th>system</th>
<th>conv</th>
<th>proposal</th>
<th>region-wise</th>
<th>total</th>
<th>rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG</td>
<td>SS + Fast R-CNN</td>
<td>146</td>
<td>1510</td>
<td>174</td>
<td>1830</td>
<td>0.5 fps</td>
</tr>
<tr>
<td>VGG</td>
<td>RPN + Fast R-CNN</td>
<td>141</td>
<td>10</td>
<td>47</td>
<td>198</td>
<td>5 fps</td>
</tr>
<tr>
<td>ZF</td>
<td>RPN + Fast R-CNN</td>
<td>31</td>
<td>3</td>
<td>25</td>
<td>59</td>
<td>17 fps</td>
</tr>
</tbody>
</table>

Table 10: **One-Stage Detection vs. Two-Stage Proposal + Detection**. Detection results are on the PASCAL VOC 2007 test set using the ZF model and Fast R-CNN. RPN uses unshared features.

<table>
<thead>
<tr>
<th></th>
<th>proposals</th>
<th>detector</th>
<th>mAP (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-Stage</td>
<td>RPN + ZF, unshared</td>
<td>Fast R-CNN + ZF, 1 scale</td>
<td>58.7</td>
</tr>
<tr>
<td>One-Stage</td>
<td>dense, 3 scales, 3 aspect ratios</td>
<td>Fast R-CNN + ZF, 1 scale</td>
<td>53.8</td>
</tr>
<tr>
<td>One-Stage</td>
<td>dense, 3 scales, 3 aspect ratios</td>
<td>Fast R-CNN + ZF, 5 scales</td>
<td>53.9</td>
</tr>
</tbody>
</table>
• Recognition = Description + Discrimination. We are doing well on Discrimination.

• Challenges of Deformation and Occlusion. This is also known as hard negative mining and non-maximum suppression. The past solutions assume the features are hard coded. CNN solutions extract ‘deformable’ features. Occlusion is dealt with per pixel labels over many object classes.

• R-CNN, Fast R-CNN, Faster R-CNN: sparse sampling of ‘where’ followed by decision of ‘what’. The two systems (2 steps) share most feature extract pipeline. Using one shared feature extraction front-end, and multiple regression objectives at the FC layers. Enough information from the front-end CNN features allows both detection (of location) and classification (of classes).

• Memory allocation and flow is important for CNN learning and speed up.
Some members of the postdeeluvian* object detection family tree

NOW WITH MORE LAYERS

*Serge Bolongieism
Position-Sensitive Score Maps

Channels take responsibility for relative spatial locations
Efficient Sharing of Diagrams
Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the person category.
Visualisation: Miss

Figure 4: Visualization when an RoI does not correctly overlap the object.
• Bbox regression under standard parameterisation
• Standard loss function
• Online Hard Example Mining during training
• Faster R-CNN-style alternating optimisation
• Dilation used at conv5 (RPN works from conv4) - gives a 2.6 mAP boost
The Effect of Position Sensitivity on fully convolutional strategies

Table 2: Comparisons among fully convolutional (or “almost” fully convolutional) strategies using ResNet-101. All competitors in this table use the à trous trick. Hard example mining is not conducted.

<table>
<thead>
<tr>
<th>method</th>
<th>RoI output size $(k \times k)$</th>
<th>mAP on VOC 07 (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>naïve Faster R-CNN</td>
<td>$1 \times 1$</td>
<td>61.7</td>
</tr>
<tr>
<td></td>
<td>$7 \times 7$</td>
<td>68.9</td>
</tr>
<tr>
<td>class-specific RPN</td>
<td>-</td>
<td>67.6</td>
</tr>
<tr>
<td>R-FCN (w/o position-sensitivity)</td>
<td>$1 \times 1$</td>
<td>fail</td>
</tr>
<tr>
<td>R-FCN</td>
<td>$3 \times 3$</td>
<td>75.5</td>
</tr>
<tr>
<td></td>
<td>$7 \times 7$</td>
<td>76.6</td>
</tr>
</tbody>
</table>

(“naïve” Faster R-CNN still has FC layer after RoI pooling)

Without position sensitivity, Faster R-CNN takes a major performance hit when the RoI pooling is late in the network
Standard Benchmarks: VOC 2007


<table>
<thead>
<tr>
<th></th>
<th>training data</th>
<th>mAP (%)</th>
<th>test time (sec/img)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN [9]</td>
<td>07+12</td>
<td>76.4</td>
<td>0.42</td>
</tr>
<tr>
<td>Faster R-CNN +++</td>
<td>07+12+COCO</td>
<td>85.6</td>
<td>3.36</td>
</tr>
<tr>
<td>R-FCN</td>
<td>07+12</td>
<td>79.5</td>
<td>0.17</td>
</tr>
<tr>
<td>R-FCN multi-sc train</td>
<td>07+12</td>
<td>80.5</td>
<td>0.17</td>
</tr>
<tr>
<td>R-FCN multi-sc train</td>
<td>07+12+COCO</td>
<td>83.6</td>
<td>0.17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Model</th>
<th>Training Data</th>
<th>mAP (%)</th>
<th>Test Time (sec/img)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN [9]</td>
<td>07++12</td>
<td>73.8</td>
<td>0.42</td>
</tr>
<tr>
<td>Faster R-CNN +++ [9]</td>
<td>07++12+COCO</td>
<td>83.8</td>
<td>3.36</td>
</tr>
<tr>
<td><strong>R-FCN</strong> multi-sc train</td>
<td>07++12</td>
<td>77.6†</td>
<td>0.17</td>
</tr>
<tr>
<td><strong>R-FCN</strong> multi-sc train</td>
<td>07++12+COCO</td>
<td>82.0‡</td>
<td>0.17</td>
</tr>
</tbody>
</table>
The Effect of Proposal Type

<table>
<thead>
<tr>
<th></th>
<th>training data</th>
<th>test data</th>
<th>RPN [18]</th>
<th>SS [27]</th>
<th>EB [28]</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-FCN</td>
<td>07+12</td>
<td>07</td>
<td>79.5</td>
<td>77.2</td>
<td>77.8</td>
</tr>
</tbody>
</table>

Works pretty well with any proposal method
### Standard Benchmarks: MS COCO

<table>
<thead>
<tr>
<th></th>
<th>training data</th>
<th>test data</th>
<th>AP@0.5</th>
<th>AP</th>
<th>AP small</th>
<th>AP medium</th>
<th>AP large</th>
<th>test time (sec/img)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN [9]</td>
<td>train</td>
<td>val</td>
<td>48.4</td>
<td>27.2</td>
<td>6.6</td>
<td>28.6</td>
<td>45.0</td>
<td>0.42</td>
</tr>
<tr>
<td>R-FCN</td>
<td>train</td>
<td>val</td>
<td>48.9</td>
<td>27.6</td>
<td>8.9</td>
<td>30.5</td>
<td>42.0</td>
<td>0.17</td>
</tr>
<tr>
<td>R-FCN multi-sc train</td>
<td>train</td>
<td>val</td>
<td>49.1</td>
<td>27.8</td>
<td>8.8</td>
<td>30.8</td>
<td>42.2</td>
<td>0.17</td>
</tr>
<tr>
<td>Faster R-CNN +++ [9]</td>
<td>trainval</td>
<td>test-dev</td>
<td><strong>55.7</strong></td>
<td><strong>34.9</strong></td>
<td>15.6</td>
<td>38.7</td>
<td>50.9</td>
<td>3.36</td>
</tr>
<tr>
<td>R-FCN</td>
<td>trainval</td>
<td>test-dev</td>
<td>51.5</td>
<td>29.2</td>
<td>10.3</td>
<td>32.4</td>
<td>43.3</td>
<td>0.17</td>
</tr>
<tr>
<td>R-FCN multi-sc train</td>
<td>trainval</td>
<td>test-dev</td>
<td>51.9</td>
<td>29.9</td>
<td>10.8</td>
<td>32.8</td>
<td>45.0</td>
<td>0.17</td>
</tr>
<tr>
<td>R-FCN multi-sc train, test</td>
<td>trainval</td>
<td>test-dev</td>
<td><strong>53.2</strong></td>
<td><strong>31.5</strong></td>
<td>14.3</td>
<td>35.5</td>
<td>44.2</td>
<td>1.00</td>
</tr>
</tbody>
</table>
Single Shot or Class-wise RPN models

Feature Extractor

Detection Generator
- Multiway Classification
- Box Regression

Use **TF-slim** model zoo to swap in multiple feature extractor architectures


(a) SSD.

(b) Faster RCNN.

(c) R-FCN.
SSD Output Layer
SSD Training

- Match default boxes to ground truth boxes to determine true/false positives.

- Loss = $\textbf{SmoothL1}\text{ (box param)} + \textbf{Softmax}\text{ (class prob)}$
Multi-Scale Feature Maps

SSD

8 × 8 feature map  4 × 4 feature map
Multi-Scale Feature Maps

SSD

8 × 8 feature map

4 × 4 feature map

Faster R-CNN Objectness Proposal, Ren 2015

vs.

8 × 8 feature map
## Multi-Scale Feature Maps Experiment

<table>
<thead>
<tr>
<th>Prediction source layers from:</th>
<th>mAP use boundary boxes?</th>
<th># Boxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 × 38</td>
<td>Yes</td>
<td>74.3</td>
</tr>
<tr>
<td>19 × 19</td>
<td>Yes</td>
<td>70.7</td>
</tr>
<tr>
<td>10 × 10</td>
<td>Yes</td>
<td>62.4</td>
</tr>
<tr>
<td>5 × 5</td>
<td>No</td>
<td>63.4</td>
</tr>
<tr>
<td>3 × 3</td>
<td>No</td>
<td>69.2</td>
</tr>
<tr>
<td>1 × 1</td>
<td>No</td>
<td>64.0</td>
</tr>
</tbody>
</table>
## Multi-Scale Feature Maps Experiment

<table>
<thead>
<tr>
<th>Prediction source layers from:</th>
<th>mAP use boundary boxes?</th>
<th># Boxes</th>
</tr>
</thead>
<tbody>
<tr>
<td>38 × 38</td>
<td>Yes</td>
<td>8732</td>
</tr>
<tr>
<td>19 × 19</td>
<td>No</td>
<td>9864</td>
</tr>
<tr>
<td>10 × 10</td>
<td>74.3</td>
<td></td>
</tr>
<tr>
<td>5 × 5</td>
<td>70.7</td>
<td></td>
</tr>
<tr>
<td>3 × 3</td>
<td>62.4</td>
<td></td>
</tr>
<tr>
<td>1 × 1</td>
<td>64.0</td>
<td>8664</td>
</tr>
</tbody>
</table>

boundary boxes
Splitting the Region Space

<table>
<thead>
<tr>
<th>include ${\frac{1}{2}, 2}$ box?</th>
<th>SSD300</th>
</tr>
</thead>
<tbody>
<tr>
<td>include ${\frac{1}{3}, 3}$ box?</td>
<td>✔️ ✔️</td>
</tr>
<tr>
<td>number of Boxes</td>
<td></td>
</tr>
<tr>
<td>VOC2007 test mAP</td>
<td></td>
</tr>
</tbody>
</table>

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSD300</td>
<td></td>
</tr>
<tr>
<td>3880</td>
<td>7760</td>
</tr>
<tr>
<td>8732</td>
<td></td>
</tr>
</tbody>
</table>

3880 7760 8732
71.6 73.7 74.3
• Matching ground truth and default boxes
  • Match each GT box to closest default box
Handling Many Default Boxes

- Matching ground truth and default boxes
  - Match each GT box to closest default box
  - Also match each GT box to all unassigned default boxes with IoU > 0.5

- Hard negative mining
  - Unbalanced training: 1-30 TP, 8k-25k FP
  - Keep TP:FP ratio fixed (1:3), use worst-misclassified FPs.
SSD Architecture

Extra Convolutional Feature Maps

Classifier: Conv: 3x3x(3x(Classes+4))

Classifier: Conv: 3x3x(6x(Classes+4))

Conv: 3x3x(4x(Classes+4))

Detections: 8732 per Class

Non-Maximum Suppression

74.3mAP
46FPS
Data Augmentation

<table>
<thead>
<tr>
<th>data augmentation</th>
<th>SSD300</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal flip</td>
<td>✓</td>
</tr>
<tr>
<td>random crop &amp; color distortion</td>
<td>✓</td>
</tr>
<tr>
<td>VOC2007 test mAP</td>
<td>65.5</td>
</tr>
<tr>
<td></td>
<td>74.3</td>
</tr>
</tbody>
</table>
Data Augmentation

Random expansion creates more small training examples

<table>
<thead>
<tr>
<th>data augmentation</th>
<th>SSD300</th>
</tr>
</thead>
<tbody>
<tr>
<td>horizontal flip</td>
<td>✔️</td>
</tr>
<tr>
<td>random crop &amp; color distortion</td>
<td>✔️</td>
</tr>
<tr>
<td>random expansion</td>
<td>✔️</td>
</tr>
<tr>
<td>VOC2007 test mAP</td>
<td>65.5</td>
</tr>
<tr>
<td></td>
<td>74.3</td>
</tr>
<tr>
<td></td>
<td><strong>77.2</strong></td>
</tr>
</tbody>
</table>
Figure 1: **Networks of SSD and DSSD on residual network.** The blue modules are the layers added in SSD framework, and we call them SSD Layers. In the bottom figure, the red layers are DSSD layers.
Figure 3: Deconvolution module
Speed/accuracy trade-offs for modern convolutional object detectors

Jonathan Huang
Alireza Fathi
Vivek Rathod
Ian Fischer
Chen Sun
Zbigniew Wojna
Kevin Murphy
Menglong Zhu
Yang Song
Anoop Korattikara
Sergio Guadarrama
Google Research
Residual Blocks vs. Inception Resnet Blocks

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 accuracy</th>
<th>Num. Params.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>71.0</td>
<td>14,714,688</td>
</tr>
<tr>
<td>MobileNet</td>
<td>71.1</td>
<td>3,191,072</td>
</tr>
<tr>
<td>Inception V2</td>
<td>73.9</td>
<td>10,173,112</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>76.4</td>
<td>42,605,504</td>
</tr>
<tr>
<td>Inception V3</td>
<td>78.0</td>
<td>21,802,784</td>
</tr>
<tr>
<td>Inception ResNet V2</td>
<td>80.4</td>
<td>54,336,736</td>
</tr>
</tbody>
</table>

Table 2: Properties of the 6 feature extractors that we use. Top-1 accuracy is the classification accuracy on ImageNet.
Meta Architecture:
- Faster RCNN
- R-FCN
- SSD

Overall mAP vs. Feature Extractor Accuracy:
- VGG-16
- MobileNet
- Inception
- Inception V2
- Inception V3
- ResNet-101
- Inception ResNet V2
Figure 4: Accuracy stratified by object size, meta-architecture and feature extractor. We fix the image resolution to 300.
Figure 5: Effect of image resolution.
Figure 6: Effect of proposing increasing number of regions on mAP accuracy (solid lines) and GPU inference time (dotted). Surprisingly, for Faster R-CNN with Inception Resnet, we obtain 96% of the accuracy of using 300 proposals by using only 50 proposals, which reduces running time by a factor of 3.
Figure 7: GPU time (milliseconds) for each model, for image resolution of 300.
Figure 8: FLOPS vs time.
Figure 9: Memory (Mb) usage for each model. Note that we measure total memory usage rather than peak memory usage. Moreover, we include all data points corresponding to the low-resolution models here. The error bars reflect variance in memory usage by using different numbers of proposals for the Faster R-CNN and R-FCN models (which leads to the seemingly considerable variance in the Faster-RCNN with Inception Resnet bar).
• Good localization at .75 IOU means good localization at all IOU thresholds

• using many fewer proposals than is usual for Faster R-CNN: at 100 proposals, the speed and accuracy for Faster R-CNN models with ResNet becomes roughly comparable to that of equivalent R-FCN models which use 300 proposals in both mAP and GPU speed

• high correlation with running time with larger and more powerful feature extractors requiring much more memory

• SSD models typically have (very) poor performance on small objects, they are competitive with Faster RCNN and R-FCN on large objects, even outperforming these meta architectures for the faster and more lightweight feature extractors