Fast R-CNN

Ross Girshick

Facebook AI Research (FAIR)

Work done at Microsoft Research

Reproducible research – get the code!

http://git.io/vBqm5
Fast Region-based ConvNets (R-CNNs) for Object Detection

- Recognition: What?
- Localization: Where?

Figure adapted from Kaiming He
Object detection renaissance (2013-present)
Object detection renaissance (2013-present)

Before deep convnets

Using deep convnets

PASCAL VOC

Year


Mean Average Precision (mAP)

0% 10% 20% 30% 40% 50% 60% 70% 80%
Object detection renaissance (2013-present)

![Graph showing the development of object detection accuracy from 2006 to 2016. The y-axis represents mean average precision (mAP), with values ranging from 0% to 80%. The x-axis represents years from 2006 to 2016. Key points include:

- **R-CNNv1** with + Accurate, + Fast, + Streamlined
- **Fast R-CNN** with + Accurate, - Slow, - Inelegant

The graph illustrates the gradual improvement in detection accuracy over the years, with significant advancements starting from 2010 onwards.]
Region-based convnets (R-CNNs)

- R-CNN (aka “slow R-CNN”) [Girshick et al. CVPR14]
- SPP-net [He et al. ECCV14]
Slow R-CNN

Girshick et al. CVPR14.
Slow R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Input image

Girshick et al. CVPR14.
Slow R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Warped image regions

Input image

Girshick et al. CVPR14.
Slow R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Warped image regions

Forward each region through ConvNet

Input image
Slow R-CNN

Regions of Interest (RoI) from a proposal method (~2k)

Warped image regions

Classify regions with SVMs

Forward each region through ConvNet

SVMs

ConvNet

SVMs

ConvNet

SVMs

Input image

Girshick et al. CVPR14.
Slow R-CNN

- Regions of Interest (RoI) from a proposal method (~2k)
- Warped image regions
- Forward each region through ConvNet
- Classify regions with SVMs
- Apply bounding-box regressors

- Input image

Girshick et al. CVPR14.
What’s wrong with slow R-CNN?
What’s wrong with slow R-CNN?

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressors (squared loss)
What’s wrong with slow R-CNN?

• Ad hoc training objectives
  • Fine-tune network with softmax classifier (log loss)
  • Train post-hoc linear SVMs (hinge loss)
  • Train post-hoc bounding-box regressors (squared loss)

• Training is slow (84h), takes a lot of disk space
What’s wrong with slow R-CNN?

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- **Inference (detection) is slow**
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]

~2000 ConvNet forward passes per image
SPP-net

He et al. ECCV14.
SPP-net

Forward whole image through ConvNet

“conv5” feature map of image

ConvNet

Input image

He et al. ECCV14.
SPP-net

Forward *whole* image through ConvNet

"conv5" feature map of image

Regions of Interest (RoIs) from a proposal method

He et al. ECCV14.
SPP-net

Regions of Interest (RoIs) from a proposal method

Forward whole image through ConvNet

“conv5” feature map of image

Spatial Pyramid Pooling (SPP) layer

He et al. ECCV14.
SPP-net

Regions of Interest (RoIs) from a proposal method

Forward whole image through ConvNet

“conv5” feature map of image

Spatial Pyramid Pooling (SPP) layer

Fully-connected layers

Classify regions with SVMs

Post hoc component

He et al. ECCV14.
SPP-net

- Forward whole image through ConvNet
- "conv5" feature map of image
- Spatial Pyramid Pooling (SPP) layer
- Fully-connected layers
- Classify regions with SVMs
- Apply bounding-box regressors

Regions of Interest (RoIs) from a proposal method

Bbox reg SVMs FCs

Post hoc component

He et al. ECCV14.
What’s good about SPP-net?

- Fixes one issue with R-CNN: makes testing fast
What’s wrong with SPP-net?

- Inherits the rest of R-CNN’s problems
  - Ad hoc training objectives
  - Training is slow (25h), takes a lot of disk space
What’s wrong with SPP-net?

• Inherits the rest of R-CNN’s problems
  • Ad hoc training objectives
  • Training is slow (though faster), takes a lot of disk space

• Introduces a new problem: cannot update parameters below SPP layer during training
SPP-net: the main limitation

Trainable (3 layers)

Freeze (13 layers)

Post hoc component

He et al. ECCV14.
Fast R-CNN

- Fast test-time, like SPP-net
Fast R-CNN

• Fast test-time, like SPP-net
• One network, trained in one stage
Fast R-CNN

- Fast test-time, like SPP-net
- One network, trained in one stage
- Higher mean average precision than slow R-CNN and SPP-net
Fast R-CNN (test time)

Regions of Interest (RoIs) from a proposal method

“conv5” feature map of image

Forward whole image through ConvNet

ConvNet

Input image
Fast R-CNN (test time)

- Forward whole image through ConvNet
- “conv5” feature map of image
- “RoI Pooling” (single-level SPP) layer
- Regions of Interest (RoIs) from a proposal method

ConvNet

Input image
Fast R-CNN (test time)

- Forward whole image through ConvNet
- "conv5" feature map of image
- "RoI Pooling" (single-level SPP) layer
- Fully-connected layers
- Linear + softmax
- Softmax classifier
- Regions of Interest (RoIs) from a proposal method
Fast R-CNN (test time)

- Input image
- Forward whole image through ConvNet
- Regions of Interest (RoIs) from a proposal method
- "conv5" feature map of image
- "RoI Pooling" (single-level SPP) layer
- Fully-connected layers
- Bounding-box regressors
- Linear + softmax
- Linear
- Softmax classifier

ConvNet
Fast R-CNN (training)

ConvNet

Linear + softmax

Linear

FCs
Fast R-CNN (training)

- ConvNet
- Linear + softmax
- Linear
- Multi-task loss

Log loss + smooth L1 loss

FCs
Fast R-CNN (training)

Log loss + smooth L1 loss

Linear + softmax

Linear

FCs

ConvNet

Multi-task loss

Trainable
Obstacle #1: Differentiable RoI pooling

Region of Interest (RoI) pooling must be (sub-)differentiable to train conv layers
Obstacle #1: Differentiable RoI pooling

\[ \frac{\partial L}{\partial x_i} = \sum_r \sum_j 1 \text{ if } r, j \text{ "pooled"} \]
\[ \text{input } i; 0 \text{ o/w } \]
\[ [i = i^*(r, j)] \frac{\partial L}{\partial y_{rj}} \]

Partial for \(x_i\) Over regions \(r\), locations \(j\)

Partial from next layer

\(i^*(0,2) = 23\)
\(i^*(1,0) = 23\)
Obstacle #2: efficient SGD steps

Slow R-CNN and SPP-net use region-wise sampling to make mini-batches

• Sample 128 example RoIs uniformly at random
• Examples will come from different images with high probability
Obstacle #2: efficient SGD steps

Note the receptive field for one example RoI is often very large

- **Worst case:** the receptive field is the entire image
Obstacle #2: efficient SGD steps

Worst case cost per mini-batch (crude model of computational complexity)

\[
\frac{128 \times 600 \times 1000}{(128 \times 224 \times 224)} = 12 \times \text{more computation than slow R-CNN}
\]
Obstacle #2: efficient SGD steps

Solution: use hierarchical sampling to build mini-batches
Obstacle #2: efficient SGD steps

Solution: use \textit{hierarchical sampling} to build mini-batches

- Sample a \textit{small} number of images (2)
Obstacle #2: efficient SGD steps

Solution: use hierarchical sampling to build mini-batches

- Sample a small number of images (2)
- Sample many examples from each image (64)
Obstacle #2: efficient SGD steps

Use the test-time trick from SPP-net during training

- Share computation between overlapping examples from the same image
Obstacle #2: efficient SGD steps

Cost per mini-batch compared to slow R-CNN (same crude cost model)

• \( \frac{2 \times 600 \times 1000}{128 \times 224 \times 224} = 0.19 \times \text{less computation than slow R-CNN} \)
## Main results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Train time (h)</td>
<td>9.5</td>
<td>84</td>
<td>25</td>
</tr>
<tr>
<td>- Speedup</td>
<td>8.8x</td>
<td>1x</td>
<td>3.4x</td>
</tr>
<tr>
<td>Test time / image</td>
<td>0.32s</td>
<td>47.0s</td>
<td>2.3s</td>
</tr>
<tr>
<td>Test speedup</td>
<td>146x</td>
<td>1x</td>
<td>20x</td>
</tr>
<tr>
<td>mAP</td>
<td>66.9%</td>
<td>66.0%</td>
<td>63.1%</td>
</tr>
</tbody>
</table>

Timings exclude object proposal time, which is equal for all methods. All methods use VGG16 from Simonyan and Zisserman.

Main results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Train time (h)</td>
<td>9.5</td>
<td>84</td>
<td>25</td>
</tr>
<tr>
<td>- Speedup</td>
<td>8.8x</td>
<td>1x</td>
<td>3.4x</td>
</tr>
<tr>
<td>Test time / image</td>
<td><strong>0.32s</strong></td>
<td>47.0s</td>
<td>2.3s</td>
</tr>
<tr>
<td>Test speedup</td>
<td><strong>146x</strong></td>
<td>1x</td>
<td>20x</td>
</tr>
<tr>
<td>mAP</td>
<td>66.9%</td>
<td>66.0%</td>
<td>63.1%</td>
</tr>
</tbody>
</table>

Timings exclude object proposal time, which is equal for all methods. All methods use VGG16 from Simonyan and Zisserman.

## Main results

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Train time (h)</td>
<td>9.5</td>
<td>84</td>
<td>25</td>
</tr>
<tr>
<td>- Speedup</td>
<td>8.8x</td>
<td>1x</td>
<td>3.4x</td>
</tr>
<tr>
<td>Test time / image</td>
<td>0.32s</td>
<td>47.0s</td>
<td>2.3s</td>
</tr>
<tr>
<td>Test speedup</td>
<td>146x</td>
<td>1x</td>
<td>20x</td>
</tr>
<tr>
<td>mAP</td>
<td>66.9%</td>
<td>66.0%</td>
<td>63.1%</td>
</tr>
</tbody>
</table>

Timings exclude object proposal time, which is equal for all methods. All methods use VGG16 from Simonyan and Zisserman.

Further test-time speedups

Forward pass timing
mAP 66.9% @ 320ms / image

- fc6: 38.7% (122ms)
- roi_pool5: 6.2% (20ms)
- fc7: 46.3% (146ms)
- other: 5.4% (17ms)

Fully connected layers take 45% of the forward pass time.
Further test-time speedups

Compress these layers with truncated SVD

J. Xue, J. Li, and Y. Gong.
Further test-time speedups

Without SVD

Forward pass timing
mAP 66.9% @ 320ms / image

- conv 46.3% (146ms)
- fc7 6.2% (20ms)
- roi_pool5 5.4% (17ms)
- other 3.5% (11ms)
- fc6 38.7% (122ms)

With SVD

Forward pass timing (SVD)
mAP 66.6% @ 223ms / image

- conv 67.8% (143ms)
- fc6 17.5% (37ms)
- roi_pool5 7.9% (17ms)
- other 5.1% (11ms)
- fc7 1.7% (4ms)
Other findings
End-to-end training matters

<table>
<thead>
<tr>
<th>Fine-tune layers</th>
<th>≥ fc6</th>
<th>≥ conv3_1</th>
<th>≥ conv2_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>VOC07 mAP</td>
<td>61.4%</td>
<td>66.9%</td>
<td>67.2%</td>
</tr>
<tr>
<td>Test time per image</td>
<td>0.32s</td>
<td>0.32s</td>
<td>0.32s</td>
</tr>
</tbody>
</table>

1.4x slower training
Multi-task training helps

<table>
<thead>
<tr>
<th></th>
<th>Fast R-CNN (VGG16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-task training?</td>
<td>Y</td>
</tr>
<tr>
<td>Stage-wise training?</td>
<td></td>
</tr>
<tr>
<td>Test-time bbox reg.</td>
<td>Y</td>
</tr>
<tr>
<td>VOC07 mAP</td>
<td>62.6%</td>
</tr>
</tbody>
</table>
Multi-task training helps

<table>
<thead>
<tr>
<th></th>
<th>Fast R-CNN (VGG16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-task training?</td>
<td>Y</td>
</tr>
<tr>
<td>Stage-wise training?</td>
<td>Y</td>
</tr>
<tr>
<td>Test-time bbox reg.</td>
<td>Y</td>
</tr>
<tr>
<td>VOC07 mAP</td>
<td>62.6% 63.4% 64.0% 66.9%</td>
</tr>
</tbody>
</table>

Trained without a bbox regressor
Multi-task training helps

<table>
<thead>
<tr>
<th></th>
<th>Fast R-CNN (VGG16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-task training?</td>
<td>Y</td>
</tr>
<tr>
<td>Stage-wise training?</td>
<td></td>
</tr>
<tr>
<td>Test-time bbox reg.</td>
<td>Y</td>
</tr>
<tr>
<td>VOC07 mAP</td>
<td>62.6%</td>
</tr>
</tbody>
</table>

Trained with a bbox regressor, but it’s disabled at test time.
Multi-task training helps

<table>
<thead>
<tr>
<th></th>
<th>Fast R-CNN (VGG16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-task training?</td>
<td>Y</td>
</tr>
<tr>
<td>Stage-wise training?</td>
<td>Y</td>
</tr>
<tr>
<td>Test-time bbox reg.</td>
<td>Y</td>
</tr>
<tr>
<td>VOC07 mAP</td>
<td>62.6% 63.4% 64.0% 66.9%</td>
</tr>
</tbody>
</table>

Post hoc bbox regressor, used at test time.
Multi-task training helps

<table>
<thead>
<tr>
<th></th>
<th>Fast R-CNN (VGG16)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-task training?</td>
<td>Y</td>
</tr>
<tr>
<td>Stage-wise training?</td>
<td></td>
</tr>
<tr>
<td>Test-time bbox reg.</td>
<td></td>
</tr>
<tr>
<td>VOC07 mAP</td>
<td>62.6%</td>
</tr>
</tbody>
</table>

Multi-task objective, using bbox regressors at test time
What’s still wrong?

- Out-of-network region proposals
  - Selective search: 2s / im; EdgeBoxes: 0.2s / im
- Fortunately, we have a solution
  - Our follow-up work was presented last week at NIPS

Fast R-CNN take-aways

• End-to-end training of deep ConvNets for detection
• Fast training times
• Open source for easy experimentation
  “I think [the Fast R-CNN] code is average-somewhat above average for what it is.” – sporkles on r/MachineLearning
• A large number of ImageNet detection and COCO detection methods are built on Fast R-CNN
  Checkout the ImageNet / COCO Challenge workshop on Thursday!
Thanks!

rbg@fb.com

Reproducible research – get the code!

http://git.io/vBqm5
Softmax works well (vs. post hoc SVMs)

<table>
<thead>
<tr>
<th>Method (VGG16)</th>
<th>classifier</th>
<th>VOC07 mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow R-CNN</td>
<td>Post hoc SVM</td>
<td>66.0%</td>
</tr>
<tr>
<td>Fast R-CNN</td>
<td>Post hoc SVM</td>
<td>66.8%</td>
</tr>
<tr>
<td>Fast R-CNN</td>
<td>Softmax</td>
<td>66.9%</td>
</tr>
</tbody>
</table>
More proposals is harmful