Fast R-CNN
Object detection with Caffe

Ross Girshick
Microsoft Research

arXiv    code

Latest roasts
Goals for this section

• Super quick intro to object detection
• Show one way to tackle obj. det. with ConvNets
• Highlight some more sophisticated uses of Caffe
  • Python layers
  • Multi-task training with multiple losses
  • Batch sizes that change dynamically during Net::Forward()
• Pointers to open source code so you can explore, try, and understand!
Image classification (mostly what you’ve seen)

- $K$ classes
- Task: Assign the correct class label to the whole image

Digit classification (MNIST)  
Object recognition (Caltech-101, ImageNet, etc.)
Classification vs. Detection

- Dog
- Bridge

Easyish, these days
Still quite a lot harder
The Visual World \( \approx K \) object classes
\{airplane, bird, motorbike, person, sofa, bg\}

Input

\[\text{YODA: Yet another Object Detection Algorithm}\]

Desired output

*Actual results may vary*
PASCAL VOC object detection

Before the successful application of ConvNets

Precision: higher is better

mean Average Precision (mAP)

year


< 2 years
1.8x mAP

~5 years

After
Fast R-CNN (Region-based Convolutional Networks)

A fast object detector implemented with Caffe
- Caffe fork on GitHub that adds two new layers (ROIPoolingLayer and SmoothL1LossLayer)
- Python (using pycaffe) / more advanced Caffe usage
- A type of Region-based Convolutional Network (R-CNN)

Let’s see how it works!
Quick background

Region-based Convolution Networks (R-CNNs)

Input image

Extract region proposals (~2k / image)
e.g., selective search
[van de Sande, Uijlings et al.]

Compute CNN features on regions

Classify and refine regions

[Girshick et al. CVPR’14]
Fast R-CNN (test-time detection)

Given an image and object proposals, detection happens with a single call to the Net::Forward() method. The Net::Forward() takes 60 to 330ms.

Two output types:
1. $NK$ regressed object boxes
2. $P(\text{cls} = k \mid \text{box} = n, \text{image})$ for each $NK$ boxes

A Fast R-CNN network (VGG_CNN_M_1024)

Object box proposals ($N$) e.g., selective search
Fast R-CNN (test-time detection)

Minimal post-processing:
- Non-maximum suppression (NMS)

Object proposals comes from:
- Selective Search (2s / image) [van de Sande/Uijlings et al.]
- EdgeBoxes (0.2s / image) [Zitnick & Dollar]
- MCG (30s / image) [Arbelaez et al.]
- Etc.

Two output types:
1. $NK$ regressed object boxes
2. $P(\text{cls} = k \mid \text{box} = n, \text{image})$ for each $NK$ boxes
Zooming into the net

Image comes in here, blob size = $S \times 3 \times H \times W$ (e.g., $S = 1$ or 5, $H = 600$, $W = 1000$)

2000 image regions come in here, blob size = $2000 \times 5$

Pool5 blob size = $2000 \times 512 \times 6 \times 6$

Conv5 feature map blob size = $S \times 512 \times H/16 \times W/16$

(a bunch of conv layers and whatnot)

2000 x (4 * 21)
Zooming into the net

Image comes in here, blob size = $S \times 3 \times H \times W$ (e.g., $S = 1$ or 5, $H = 600$, $W = 1000$)

RoI Pooling Layer:
- adaptive max pooling layer
- dynamically expands batch from $S$ to $R$ (e.g., 2000)

2000 image regions come in here, blob size = $2000 \times 5$

Pool5 blob size = $2000 \times 512 \times 6 \times 6$

Conv5 feature map blob size = $S \times 512 \times H/16 \times W/16$
Another view of the same thing

These (top and bottom images) are the same
RoI Pooling Layer

- Special case of SPPnet’s SPP layer [He et al. ECCV’14]
- Two inputs ("bottoms")
  - Conv feature map: $S \times 512 \times H \times W$
  - Regions of Interest: $R \times 5$
    - 5 comes from $[r, x1, y1, x2, y2]$, where $r$ in $[0, R – 1]$ specifies an image batch index

```cpp
layer {
  name: "roi_pool5"
  type: "ROI Pooling"
  bottom: "conv5"
  bottom: "rois"
  top: "pool5"
  roi_pooling_param {
    pooled_w: 6
    pooled_h: 6
    spatial_scale: 0.0625 # 1/16
  }
}
```
The train-time net
Single fine-tuning operation all in Caffe

Even more boxes and arrows
Let’s look at them
The train-time net (inputs)

**B full images:** $B \times 3 \times H \times W$ (e.g., $B = 2$, $H = 600$, $W = 1000$)

**Class labels:** $128 \times 21$

**Bounding-box regression targets:** $128 \times 84$

**Bounding-box regression loss weights:** $128 \times 84$

**RoIs:** $128 \times 5$ (75% background)
The train-time net (exotic data layers)

Custom Python data layer

- Samples 2 images
- From each sampled image, takes 64 RoIs
- Input batch is initially 2 elements
- Gets expanded by the RoI Pooling Layer to 128 elements
- Outputs 5 “tops”
  - data [images]
  - rois [regions of interest]
  - labels [class labels for the rois]
  - bbox_targets [box regression targets]
  - bbox_loss_weights […details…]

```python
layer {
  name: 'data'
  type: 'Python'
  top: 'data'
  top: 'rois'
  top: 'labels'
  top: 'bbox_targets'
  top: 'bbox_loss_weights'
  python_param {
    module: 'roi_data_layer.layer'
    layer: 'RoIDataLayer'
    param_str: "'num_classes': 21"
  }
}
```
The train-time net (multi-task losses)
Code is on GitHub (MIT License, Runs on Linux)
A brief tour of some of the code

<table>
<thead>
<tr>
<th>Caffe fork</th>
<th>Train, test</th>
<th>Python modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>caffe-fast-rcnn</code></td>
<td><code>data</code></td>
<td><code>lib</code></td>
</tr>
<tr>
<td>Update caffe submodule (no roi pooling CPU test)</td>
<td>add mat files for boxes, remove pickled version (and update demo.py)</td>
<td>add args to tools/reval.py; fix comp_mode abstraction</td>
</tr>
<tr>
<td><code>experiments</code></td>
<td><code>matlab</code></td>
<td><code>models</code></td>
</tr>
<tr>
<td>Make with python layer</td>
<td>rm accidentally added file</td>
<td>Improve readmes</td>
</tr>
<tr>
<td><code>output</code></td>
<td><code>tools</code></td>
<td><code>tools</code></td>
</tr>
<tr>
<td>Improve readmes</td>
<td>fixed cpu mode</td>
<td>fixed cpu mode</td>
</tr>
<tr>
<td><code>tools</code></td>
<td><code>LICENSE</code></td>
<td><code>.gitignore</code></td>
</tr>
<tr>
<td><code>README.md</code></td>
<td><code>LICENSE</code></td>
<td>ignore ipython notebook checkpoints</td>
</tr>
<tr>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
<td>change to https urls for easier installation</td>
</tr>
<tr>
<td><code>README.md</code></td>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
</tr>
<tr>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
</tr>
<tr>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
<td><code>LICENSE</code></td>
</tr>
</tbody>
</table>
A brief tour of some of the code (**Caffe bits**)

<table>
<thead>
<tr>
<th>Folder</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>caffe-fast-rcnn</td>
<td>Update caffe submodule (no roi pooling CPU test)</td>
<td>a month ago</td>
</tr>
<tr>
<td>data</td>
<td>add mat files for boxes, remove pickled version (and update demo.py)</td>
<td>a month ago</td>
</tr>
<tr>
<td>experiments</td>
<td>Make with python layer</td>
<td>a month ago</td>
</tr>
<tr>
<td>lib</td>
<td>add args to tools/reval.py; fix comp_mode abstraction</td>
<td>a month ago</td>
</tr>
<tr>
<td>matlab</td>
<td>rm accidentally added file</td>
<td>a month ago</td>
</tr>
<tr>
<td>models</td>
<td>Improve readmes</td>
<td>a month ago</td>
</tr>
<tr>
<td>output</td>
<td>Improve readmes</td>
<td>a month ago</td>
</tr>
<tr>
<td>tools</td>
<td>fixed cpu mode</td>
<td>a month ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>ignore ipython notebook checkpoints</td>
<td>3 months ago</td>
</tr>
<tr>
<td>.gitmodules</td>
<td>change to https urls for easier installation</td>
<td>a month ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>update license</td>
<td>2 months ago</td>
</tr>
<tr>
<td>README.md</td>
<td>Add pointers for computing object proposals</td>
<td>a month ago</td>
</tr>
<tr>
<td>todo.txt</td>
<td>update todos</td>
<td>a month ago</td>
</tr>
</tbody>
</table>
Region of Interest (RoI) Pooling Layer

Expands a small batch into a big batch
Smooth L1 Loss
Layer
Robust to outliers
Optimizer friendly
Per-dimension loss weights
A brief tour of some of the code (Python bits)

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib</td>
<td>add args to tools/reval.py; fix comp_mode abstraction</td>
<td>a month ago</td>
</tr>
<tr>
<td>data</td>
<td>add mat files for boxes, remove pickled version (and update demo.py)</td>
<td>a month ago</td>
</tr>
<tr>
<td>experiments</td>
<td>Make with python layer</td>
<td>a month ago</td>
</tr>
<tr>
<td>lib</td>
<td>update license</td>
<td>2 months ago</td>
</tr>
<tr>
<td>tools</td>
<td>fixed cpu mode</td>
<td>a month ago</td>
</tr>
<tr>
<td>.gitignore</td>
<td>ignore ipython notebook checkpoints</td>
<td>3 months ago</td>
</tr>
<tr>
<td>.gitmodules</td>
<td>change to https urls for easier installation</td>
<td>a month ago</td>
</tr>
<tr>
<td>LICENSE</td>
<td>update license</td>
<td>2 months ago</td>
</tr>
<tr>
<td>README.md</td>
<td>Add pointers for computing object proposals</td>
<td>a month ago</td>
</tr>
<tr>
<td>todo.txt</td>
<td>update todos</td>
<td>a month ago</td>
</tr>
</tbody>
</table>

Caffe fork

Python modules

Train, test
Python data layer for Fast R-CNN

Reshapes blobs on-the-fly
Python training code

Custom solver loop with custom snapshot method

def train_model(self, max_iters):
    """Network training loop."""

    last_snapshot_iter = -1
    timer = Timer()

    while self.solver.iter < max_iters:
        # Make one SGD update
        timer.tic()
        self.solver.step(1)
        timer.toc()

        if self.solver.iter % (10 * self.solver_param.display) == 0:
            print 'speed: {:.3f}s / it'.format(timer.average_time)

        if self.solver.iter % cfg.TRAIN.SNAPSHOT_ITERS == 0:
            last_snapshot_iter = self.solver.iter
            self.snapshot()

        if last_snapshot_iter != self.solver.iter:
            self.snapshot()
A brief tour of some of the code (CLI tools)

- Caffe fork
- Python modules
- Train, test
Train a Fast R-CNN network

optional arguments:
-h, --help show this help message and exit
--gpu GPU_ID GPU device id to use [0]
--cpu Use CPU mode (overrides --gpu)
--net {vgg16,caffenet,vgg_cnn_m_1024} Network to use [vgg16]

Usage: demo.py [-h] [--gpu GPU_ID] [--cpu]

./tools/demo.py --gpu 7
Figure 1

Figure 2

Figure 3

Loaded network /mnt/data/rbg/fast-rcnn/fast-rcnn/data/fast_rcnn_models/vgg16_fast_rcnn_iter_40000.caffemodel

Demo for data/demo/008004.jpg
Detection took 0.578s for 2888 object proposals
All car detections with p(car | box) >= 0.8

Demo for data/demo/001551.jpg
Detection took 0.364s for 2057 object proposals
All sofa detections with p(sofa | box) >= 0.8
All tvmonitor detections with p(tvmonitor | box) >= 0.8
Teaser: Faster R-CNN
Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Microsoft Research

- The detection network also proposes objects
- Marginal cost of proposals: 10ms
- VGG16 runtime ~200ms including all steps
- Higher mAP, faster
- Open-source Caffe code coming later this summer

Region Proposal Network shares conv layers with Fast R-CNN object detection network