facebook

Training R-CNNs of various velocities

Slow, fast, and faster

Ross Girshick

Facebook AI Research (FAIR)

Section overview

- Kaiming just covered inference

- This section covers
  - A brief review of the slow R-CNN and SPP-net training pipelines
  - Training Fast R-CNN detectors
  - Training Region Proposal Networks (RPNs) and Faster R-CNN detectors

Review of the slow R-CNN training pipeline

Steps for training a slow R-CNN detector

1. [offline] *M* ← Pre-train a ConvNet for ImageNet classification
2. *M'* ← Fine-tune *M* for object detection (softmax classifier + log loss)
3. *F* ← Cache feature vectors to disk using *M'*
4. Train post hoc linear SVMs on *F* (hinge loss)
5. Train post hoc linear bounding-box regressors on *F* (squared loss)

R Girshick, J Donahue, T Darrell, J Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". CVPR 2014.

Review of the slow R-CNN training pipeline

"Post hoc" means the parameters are learned after the ConvNet is fixed

1. [offline] *M* ← Pre-train a ConvNet for ImageNet classification
2. *M'* ← Fine-tune *M* for object detection (softmax classifier + log loss)
3. *F* ← Cache feature vectors to disk using *M'*
4. Train post hoc linear SVMs on *F* (hinge loss)
5. Train post hoc linear bounding-box regressors on *F* (squared loss)

R Girshick, J Donahue, T Darrell, J Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". CVPR 2014.

Review of the slow R-CNN training pipeline

Ignoring pre-training, there are three separate training stages

1. [offline] *M* ← Pre-train a ConvNet for ImageNet classification
2. *M'* ← Fine-tune *M* for object detection (softmax classifier + log loss)
3. *F* ← Cache feature vectors to disk using *M'*
4. Train post hoc linear SVMs on *F* (hinge loss)
5. Train post hoc linear bounding-box regressors on *F* (squared loss)

R Girshick, J Donahue, T Darrell, J Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". CVPR 2014.

Review of the SPP-net training pipeline

The SPP-net training pipeline is slightly different

1. [offline] *M* ← Pre-train a ConvNet for ImageNet classification
2. *F* ← Cache SPP features to disk using *M*
3. *M'* ← *M.conv* + Fine-tune 3-layer network fc6-fc7-fc8 on *F* (log loss)
4. *F'* ← Cache features on disk using *M'*
5. Train post hoc linear SVMs on *F'* (hinge loss)
6. Train post hoc linear bounding-box regressors on *F'* (squared loss)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". ECCV 2014.

Review of the SPP-net training pipeline

Note that only classifier layers are fine-tuned, the conv layers are fixed

1. [offline] *M* ← Pre-train a ConvNet for ImageNet classification
2. *F* ← Cache SPP features to disk using *M*
3. *M'* ← *M.conv* + Fine-tune 3-layer network fc6-fc7-fc8 on *F* (log loss)
4. *F'* ← Cache features on disk using *M'*
5. Train post hoc linear SVMs on *F'* (hinge loss)
6. Train post hoc linear bounding-box regressors on *F'* (squared loss)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". ECCV 2014.

Why these training pipelines are slow

Example timing for slow R-CNN / SPP-net on VOC07 (only 5k training images!) using VGG16 and a K40 GPU

- Fine-tuning (backprop, SGD): 18 hours / 16 hours
- Feature extraction: 63 hours / 5.5 hours
  - Forward pass time (SPP-net helps here)
  - Disk I/O is costly (it dominates SPP-net extraction time)
- SVM and bounding-box regressor training: 3 hours / 4 hours
- Total: 84 hours / 25.5 hours

Fast R-CNN objectives

Fix most of what's wrong with slow R-CNN and SPP-net

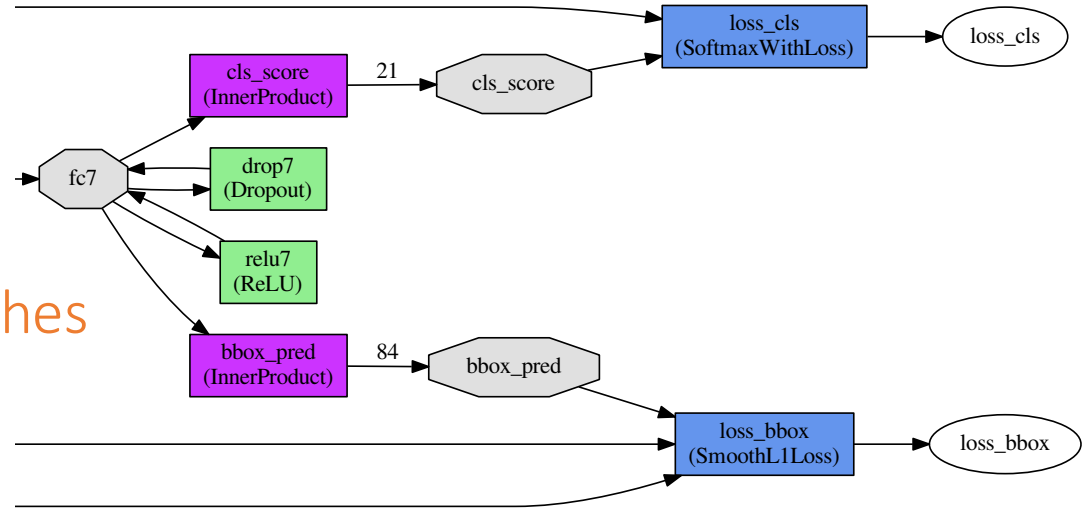
- Train the detector in a single stage, end-to-end
  - No caching features to disk
  - No post hoc training steps
- Train all layers of the network
  - Something that slow R-CNN can do
  - But is lost in SPP-net
  - Conjecture: training the conv layers is important for very deep networks (it was not important for the smaller AlexNet and ZF)

How to train Fast R-CNN end-to-end?

- Define one network with two loss branches
  - Branch 1: softmax classifier

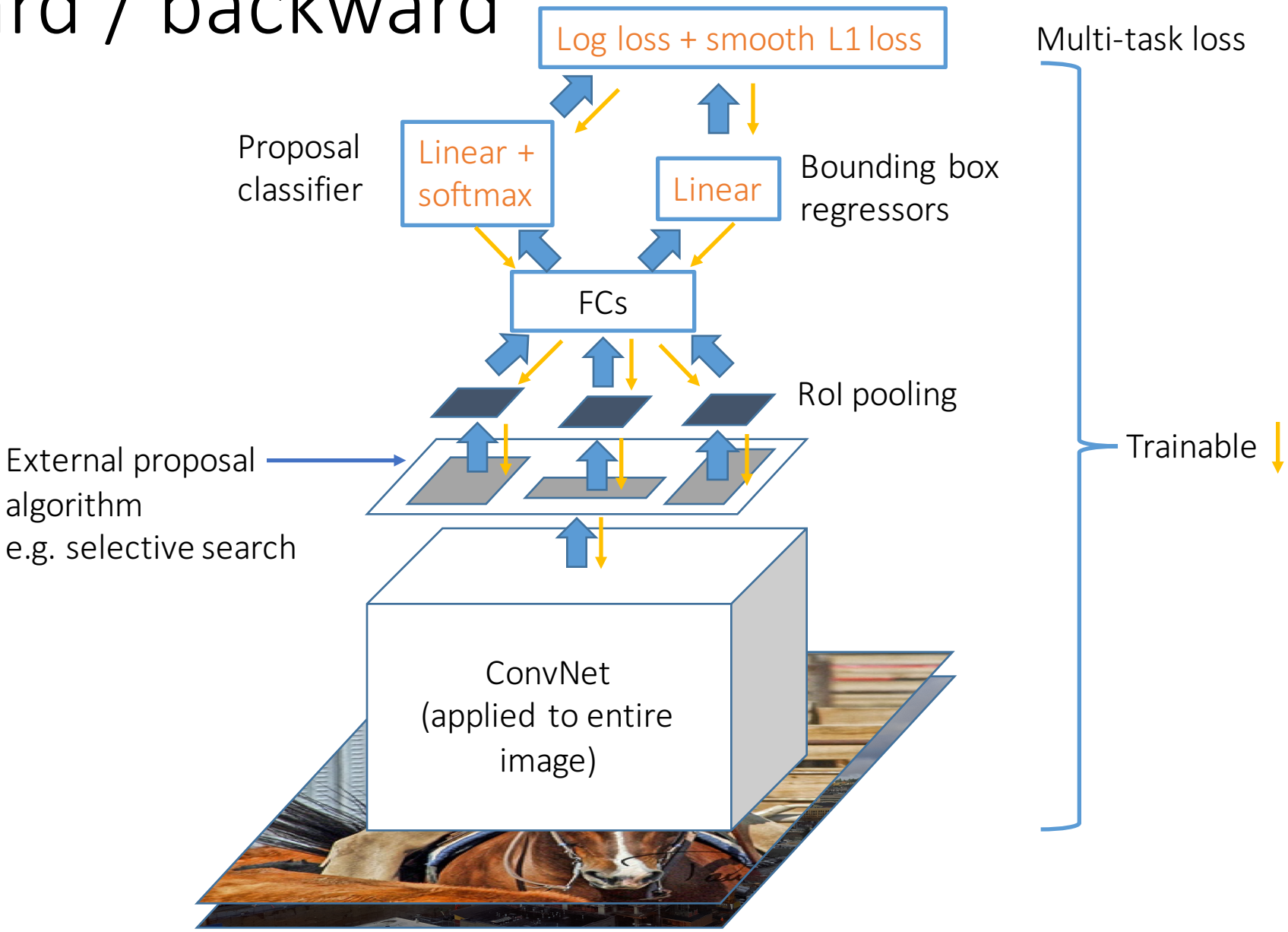
    +

  - Branch 2: linear bounding-box regressors
  - Overall loss is the sum of the two loss branches
- Fine-tune the network jointly with SGD
  - Optimizes features for both tasks
- Back-propagate errors all the way back to the conv layers



Ross Girshick. "Fast R-CNN". ICCV 2015.

Forward / backward



Benefits of end-to-end training

- Simpler implementation
- Faster training
  - No reading/writing features from/to disk
  - No training post hoc SVMs and bounding-box regressors
- Optimizing a single multi-task objective may work better than optimizing objectives independently
  - Verified empirically (see later slides)

End-to-end training requires overcoming two technical obstacles

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

Region of Interest (RoI) pooling must be (sub-)differentiable to train conv layers

Ross Girshick. “Fast R-CNN”. ICCV 2015.

Review: Spatial Pyramid Pooling (SPP) layer

From Kaiming's slides

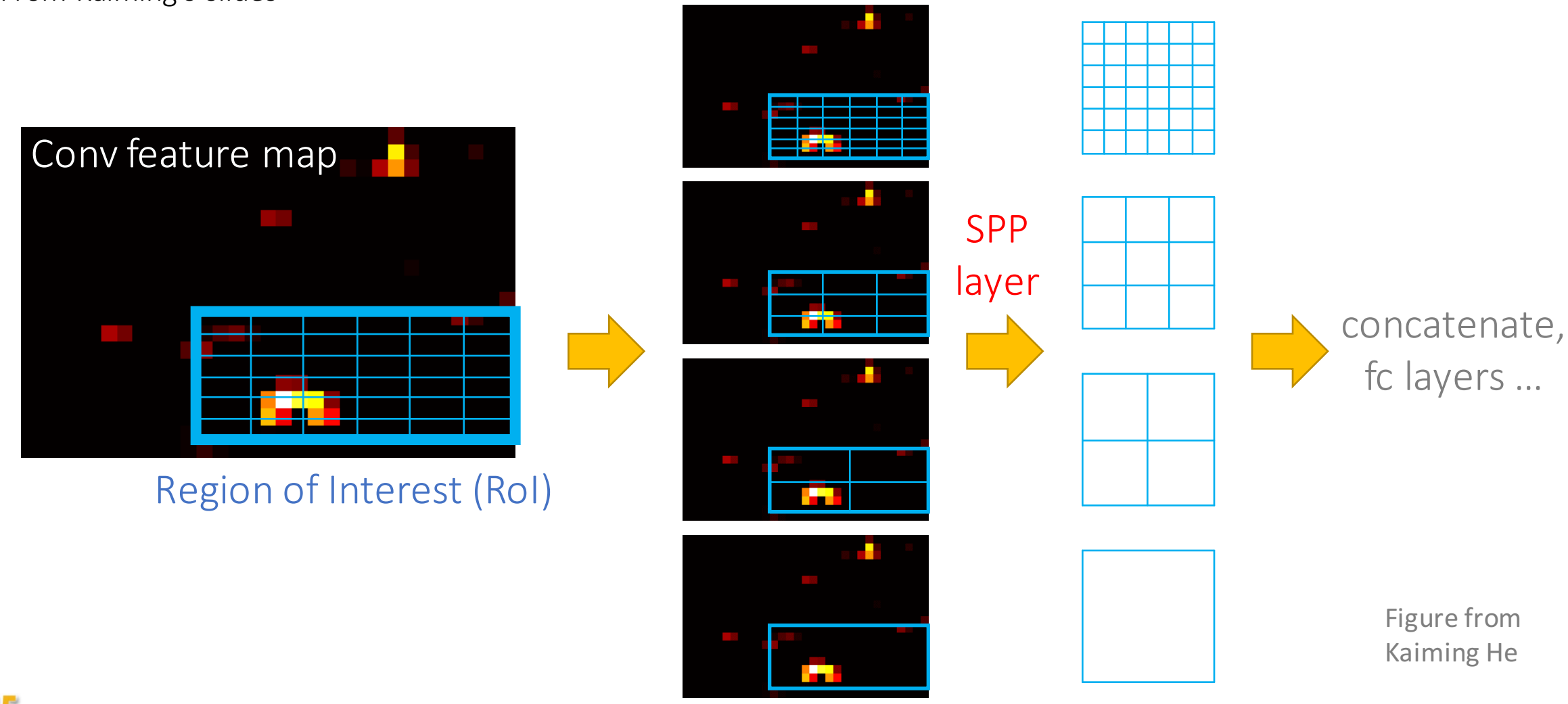


Figure from Kaiming He

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". ECCV 2014.

Review: Region of Interest (RoI) pooling layer

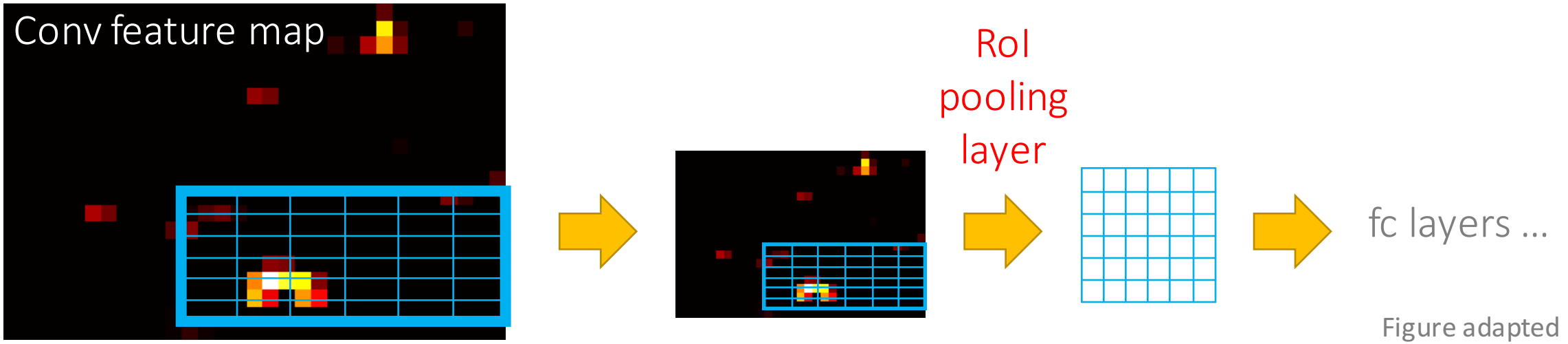


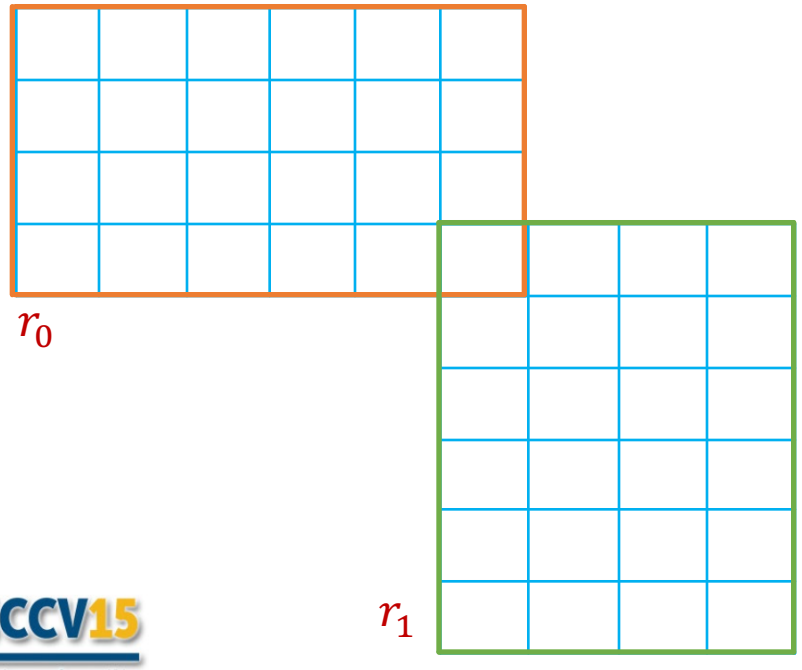
Figure adapted from Kaiming He

Just a special case of the SPP layer with one pyramid level

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

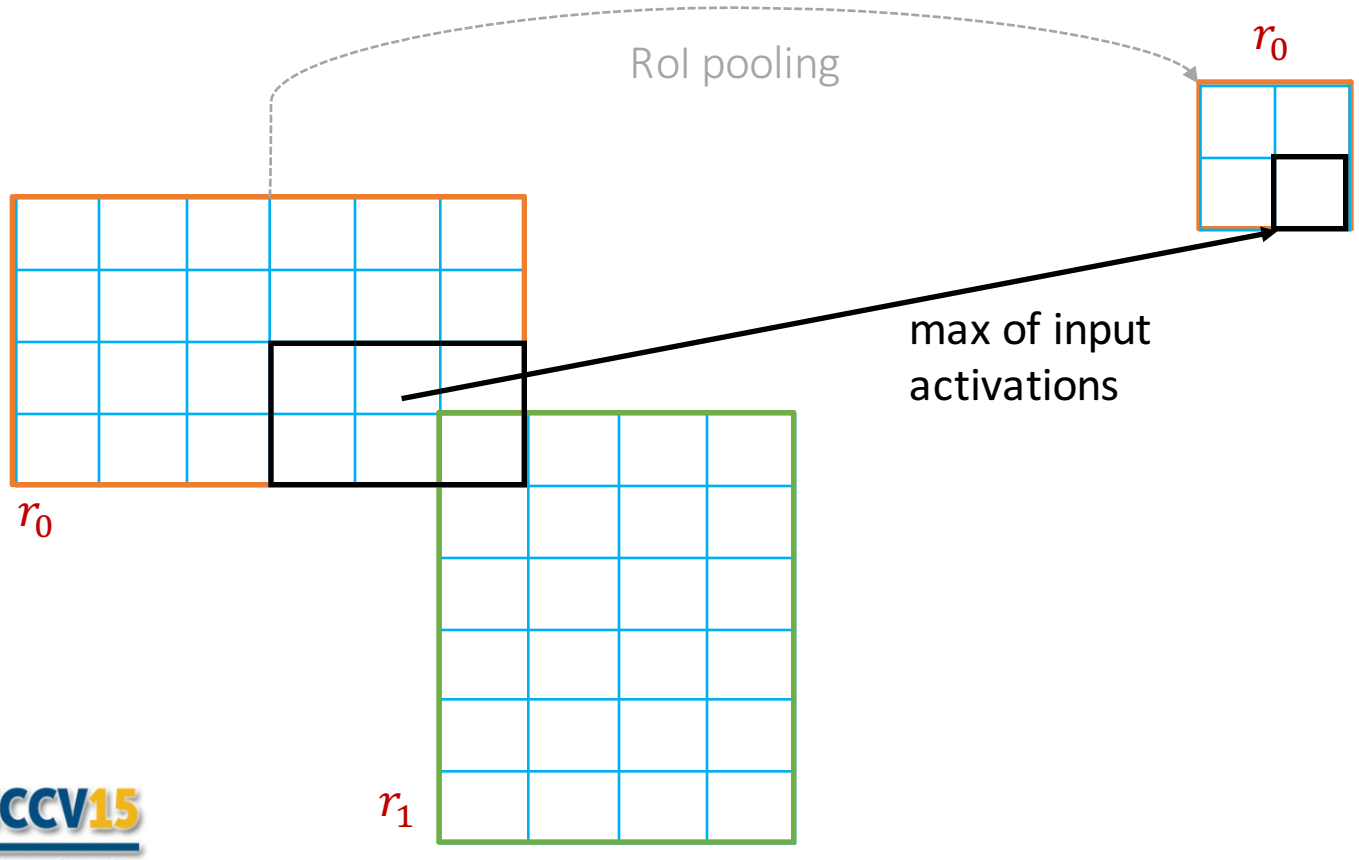
RoI pooling / SPP is just like max pooling, except that pooling regions overlap



Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

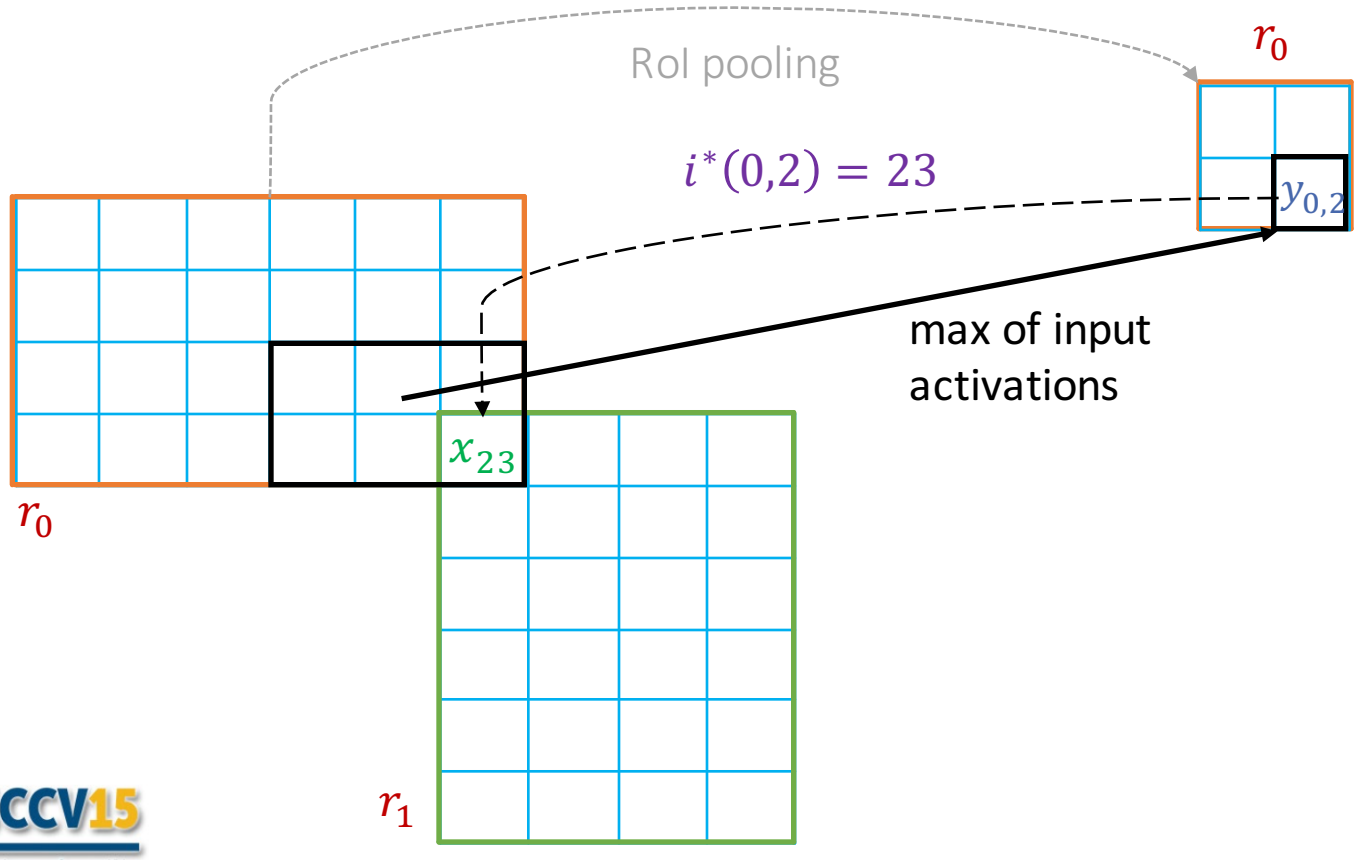
RoI pooling / SPP is just like max pooling, except that pooling regions overlap



Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap

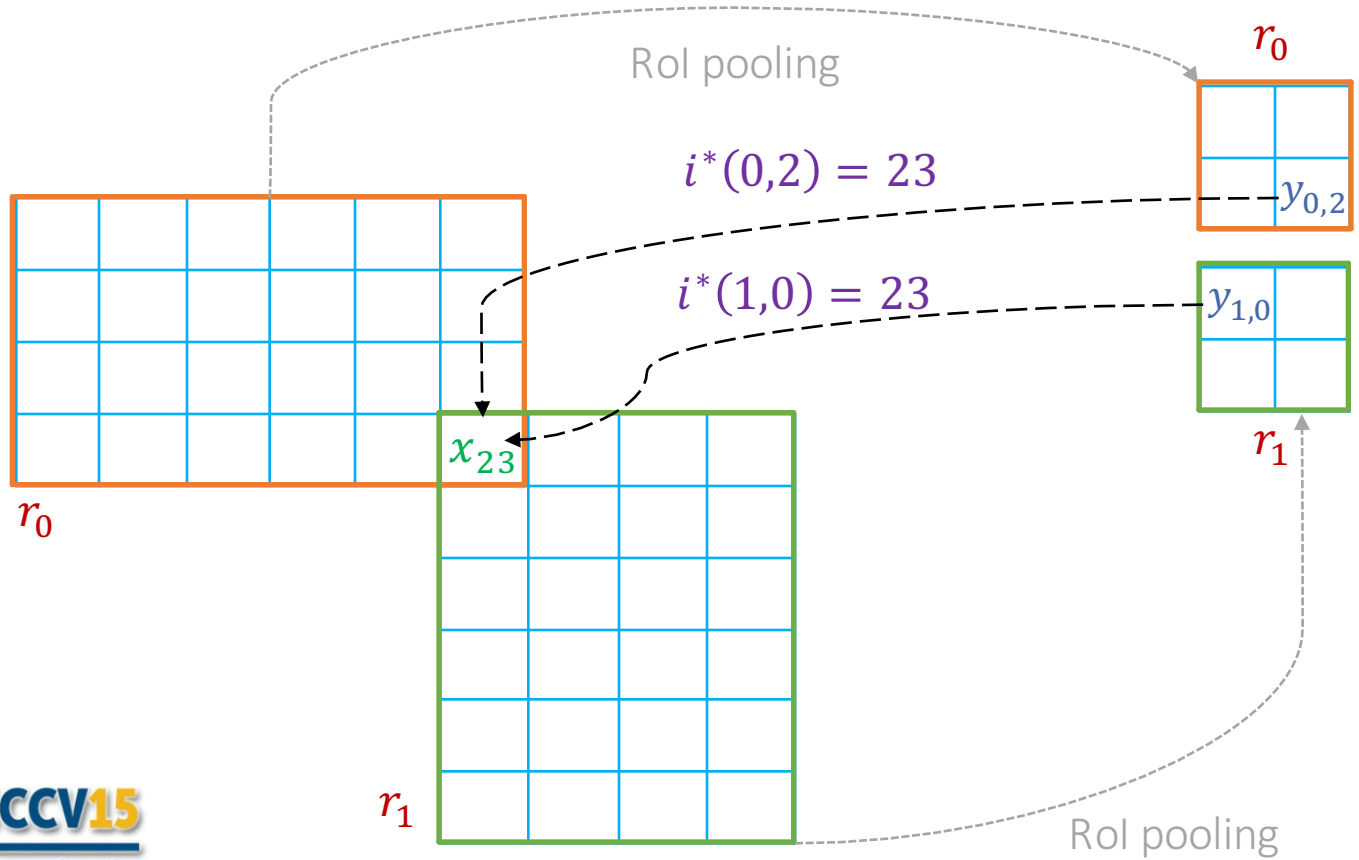


←– – – max pooling "switch" (*i.e.* argmax back-pointer)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap

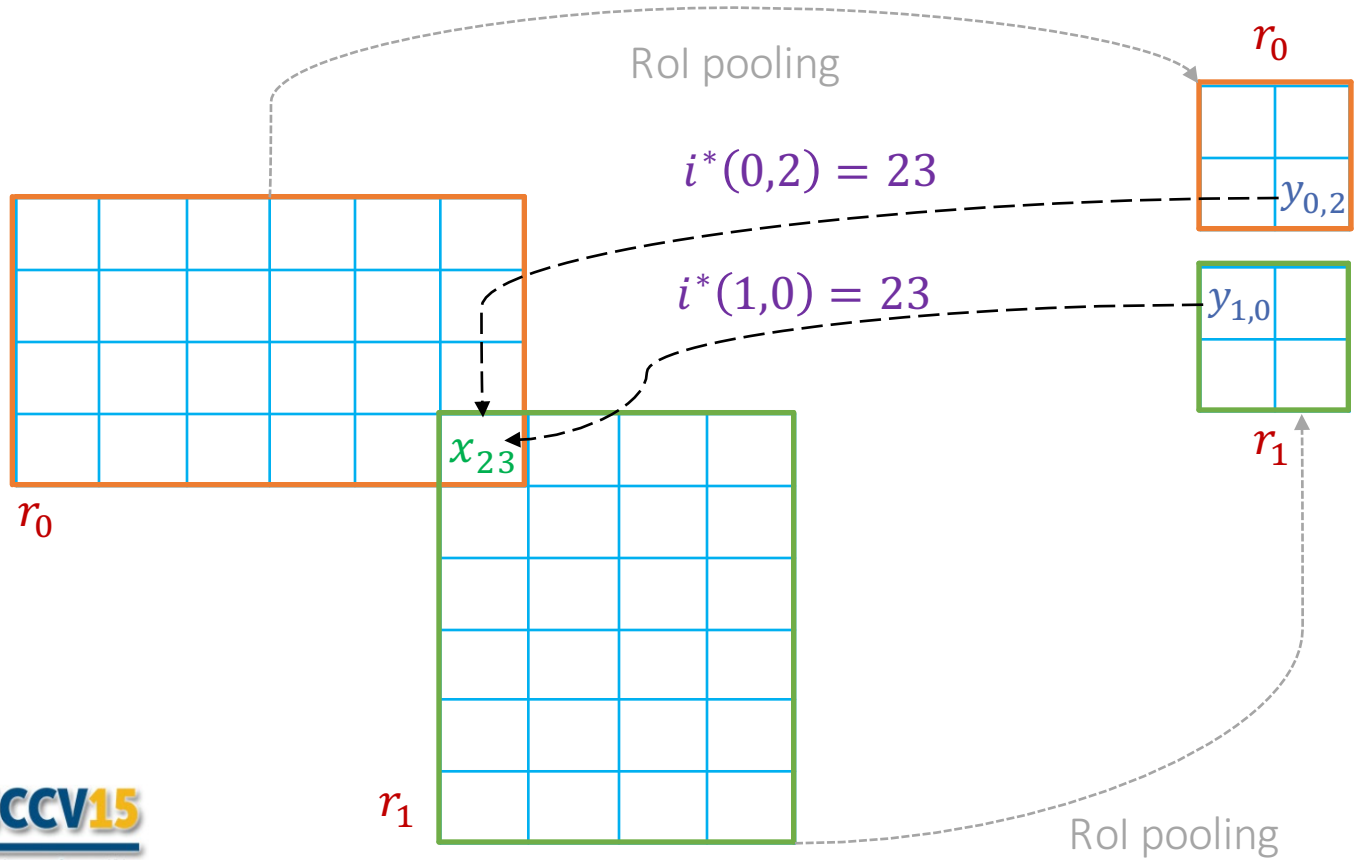


←– – – max pooling "switch" (*i.e.* argmax back-pointer)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #1: Differentiable RoI pooling

RoI pooling / SPP is just like max pooling, except that pooling regions overlap



$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{rj}}$$

1 if r, j "pooled" input i; 0 o/w

Partial for x_i

Over regions r, locations j

Partial from next layer

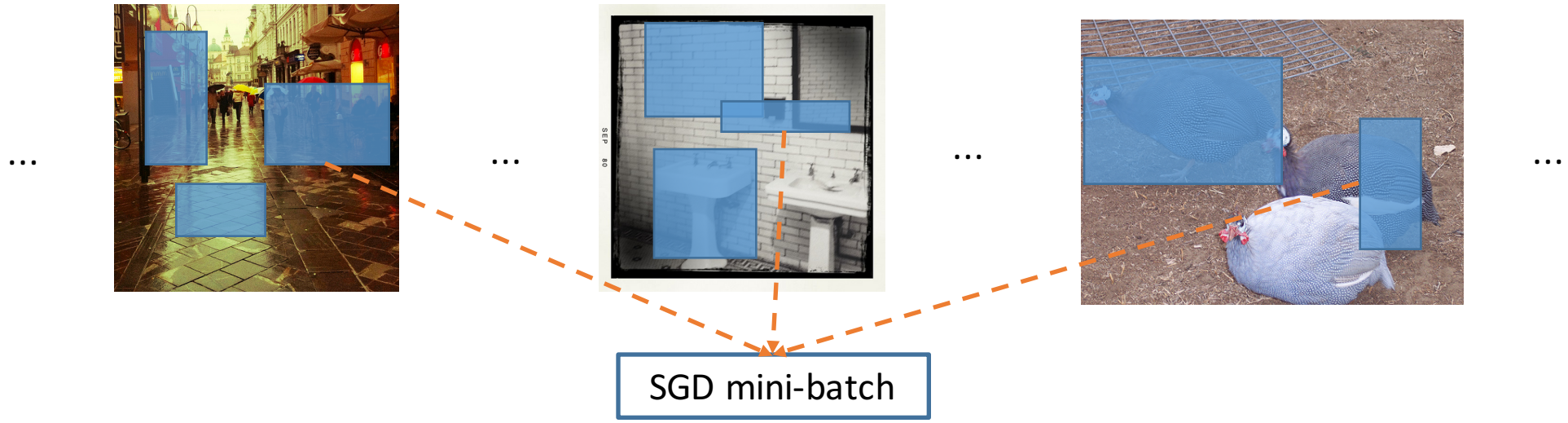
←– – – max pooling "switch" (*i.e.* argmax back-pointer)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #2: Making SGD steps efficient

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

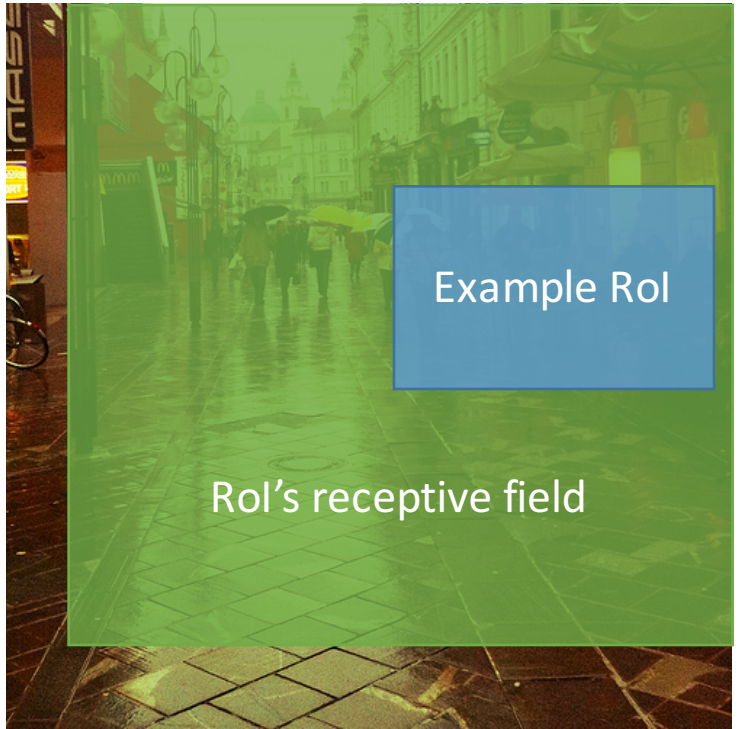
- Sample 128 example RoIs uniformly at random
- Examples will come from different images with high probability



Obstacle #2: Making SGD steps efficient

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

- Worst case: the receptive field is the entire image

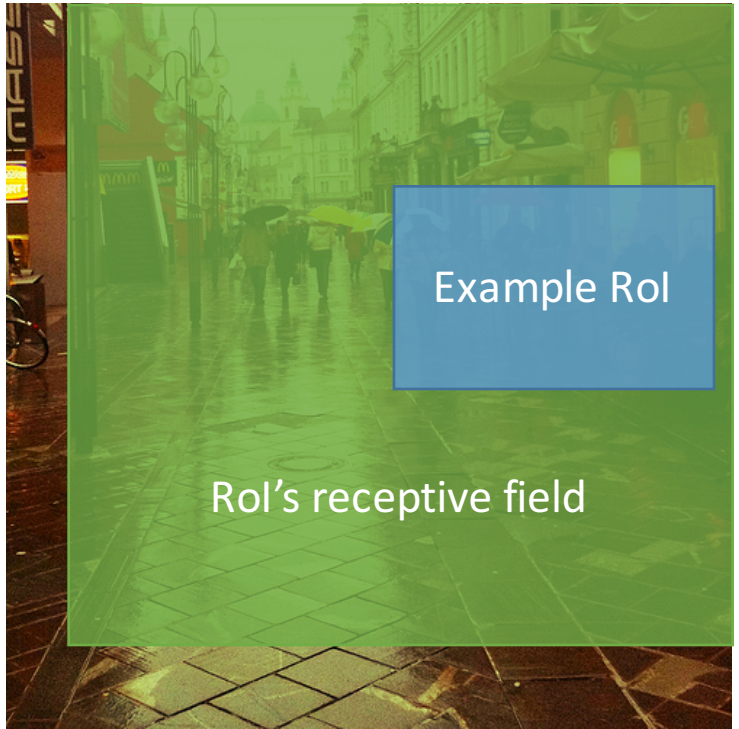


Obstacle #2: Making SGD steps efficient

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

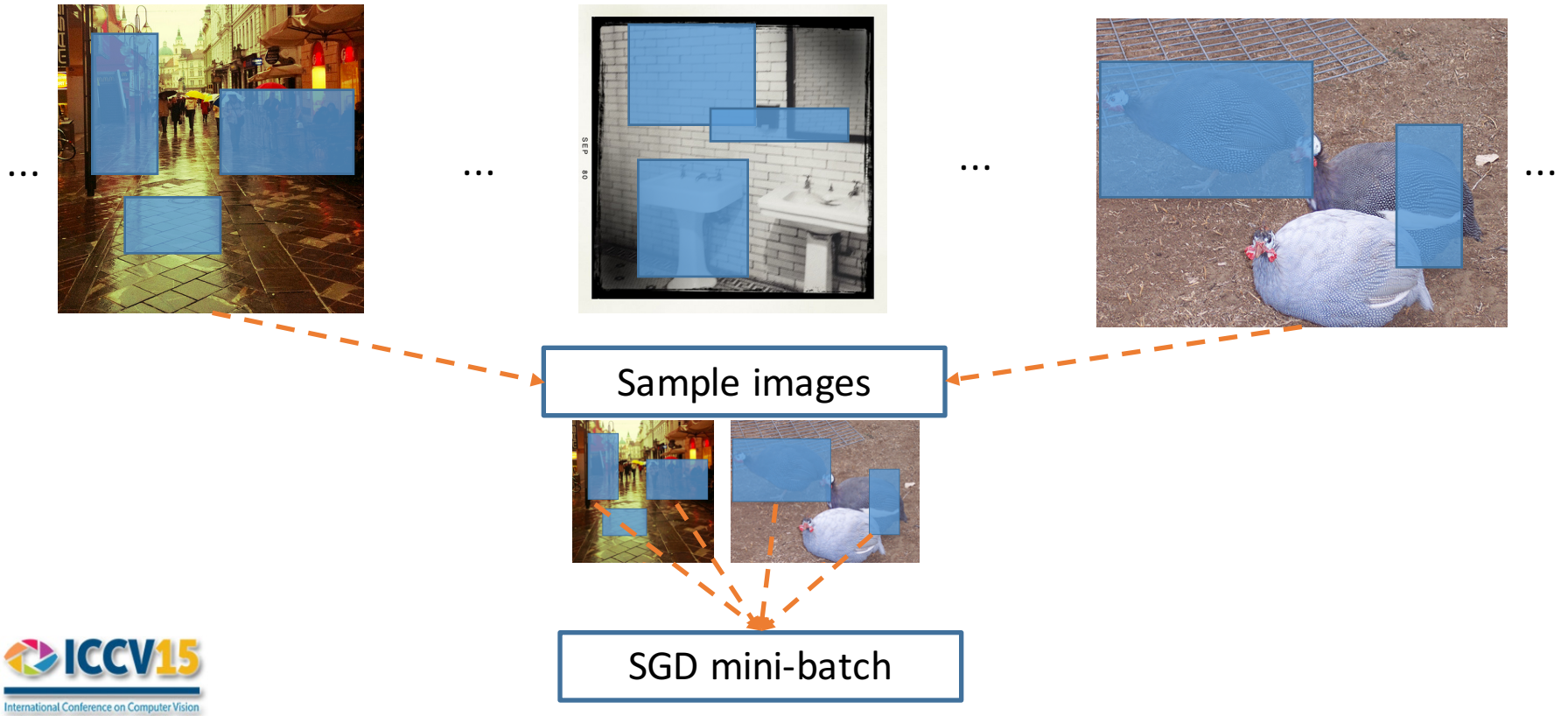
input size for Fast R-CNN input size for slow R-CNN

- 128*600*1000 / (128*224 *224) = 12x > computation than slow R-CNN



Obstacle #2: Making SGD steps efficient

Solution: use hierarchical sampling to build mini-batches



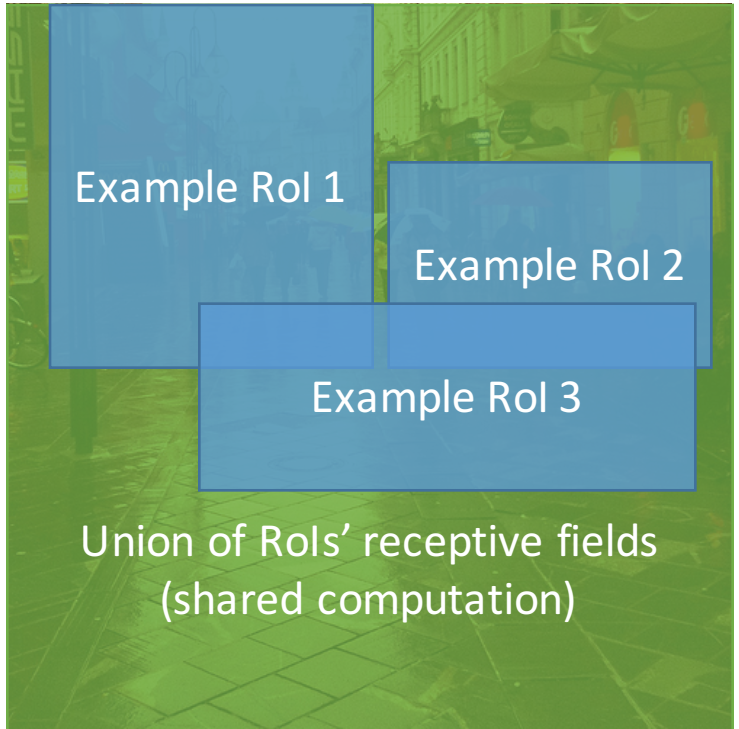
- Sample a small number of images (2)
- Sample many examples from each image (64)

Ross Girshick. "Fast R-CNN". ICCV 2015.

Obstacle #2: Making SGD steps efficient

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

- Share computation between overlapping examples from the same image

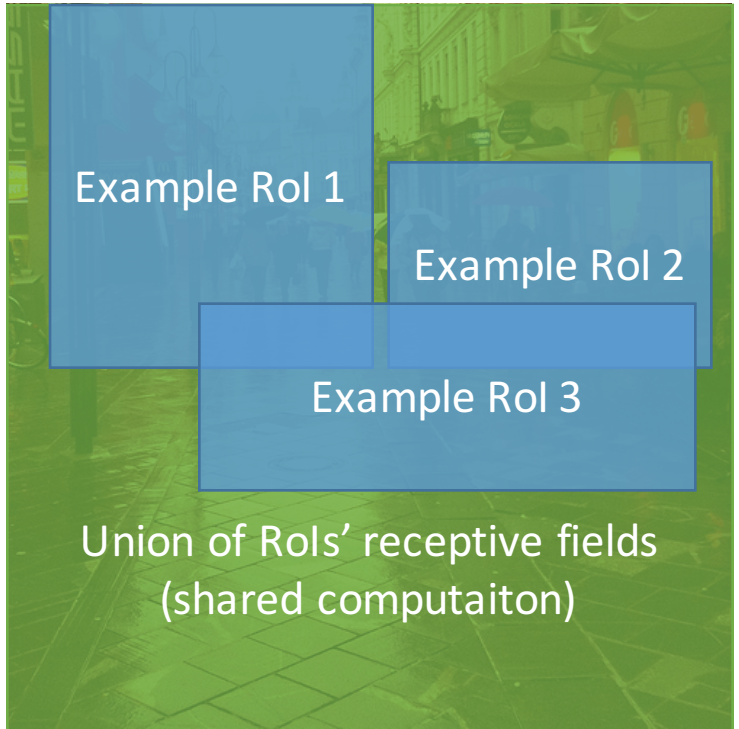


Obstacle #2: Making SGD steps efficient

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

input size for Fast R-CNN input size for slow R-CNN

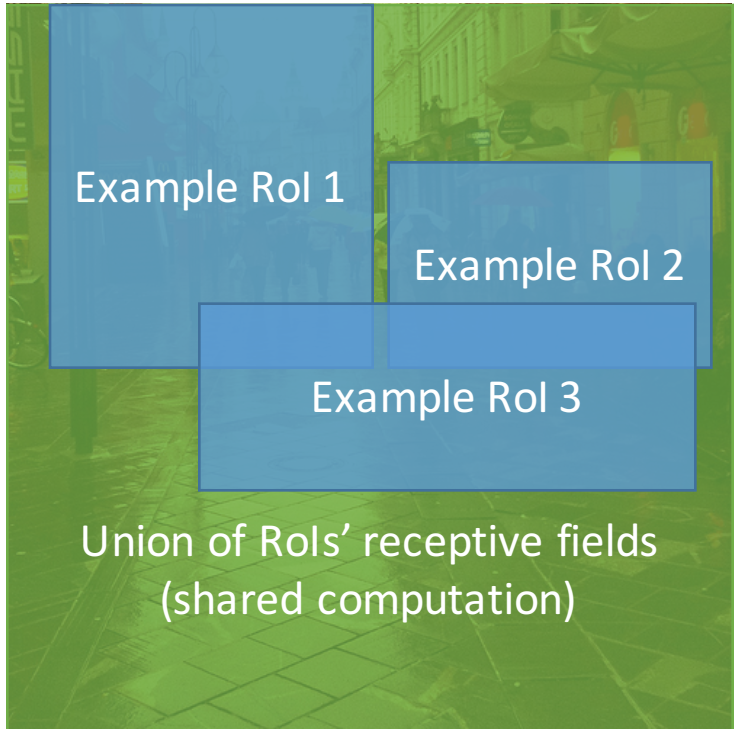
- 2*600*1000 / (128*224*224) = 0.19x < computation than slow R-CNN



Obstacle #2: Making SGD steps efficient

Are the examples from just 2 images diverse enough?

- Concern: examples from the sample image may be too correlated



Fast R-CNN outcome

Better training time and testing time with better accuracy than slow R-CNN or SPP-net

- Training time: 84 hours / 25.5 hours / 8.75 hours (Fast R-CNN)
- VOC07 test mAP: 66.0% / 63.1% / 68.1%
- Testing time per image: 47s / 2.3s / 0.32s
  - Plus 0.2 to > 2s per image depending on proposal method
  - With selective search: 49s / 4.3s / 2.32s

Updated numbers from the ICCV paper based on implementation improvements

Experimental findings

- End-to-end training is important for very deep networks
- Softmax is a fine replacement for SVMs
- Multi-task training is beneficial
- Single-scale testing is a good tradeoff (noted by Kaiming)
- Fast training and testing enables new experiments
  - Comparing proposals

The benefits of end-to-end training

	Using 1 scale			Using 5 scales
	layers that are fine-tuned in model **L**			SPPnet **L**
	≥ fc6	≥ conv3_1	≥ conv2_1	≥ fc6
VOC07 mAP	61.4	66.9	**67.2**	63.1
test rate (s/im)	**0.32**	**0.32**	**0.32**	2.3

- Model **L** = VGG16
- Training layers >= conv3_1 yields 1.4x faster SGD steps, small mAP loss

Ross Girshick. "Fast R-CNN". ICCV 2015.

Softmax is a good SVM replacement

method	classifier	S	M	L
R-CNN [9, 10]	SVM	**58.5**	**60.2**	66.0
FRCN [ours]	SVM	56.3	58.7	66.8
FRCN [ours]	softmax	57.1	59.2	**66.9**

- VOC07 test mAP
- **L** = VGG16, **M** = VGG_CNN_M_1024, **S** = Caffe/AlexNet

Multi-task training is beneficial

	L			
multi-task training?		✓		✓
stage-wise training?			✓	
test-time bbox reg?			✓	✓
VOC07 mAP	62.6	63.4	64.0	**66.9**

- **L** = VGG16

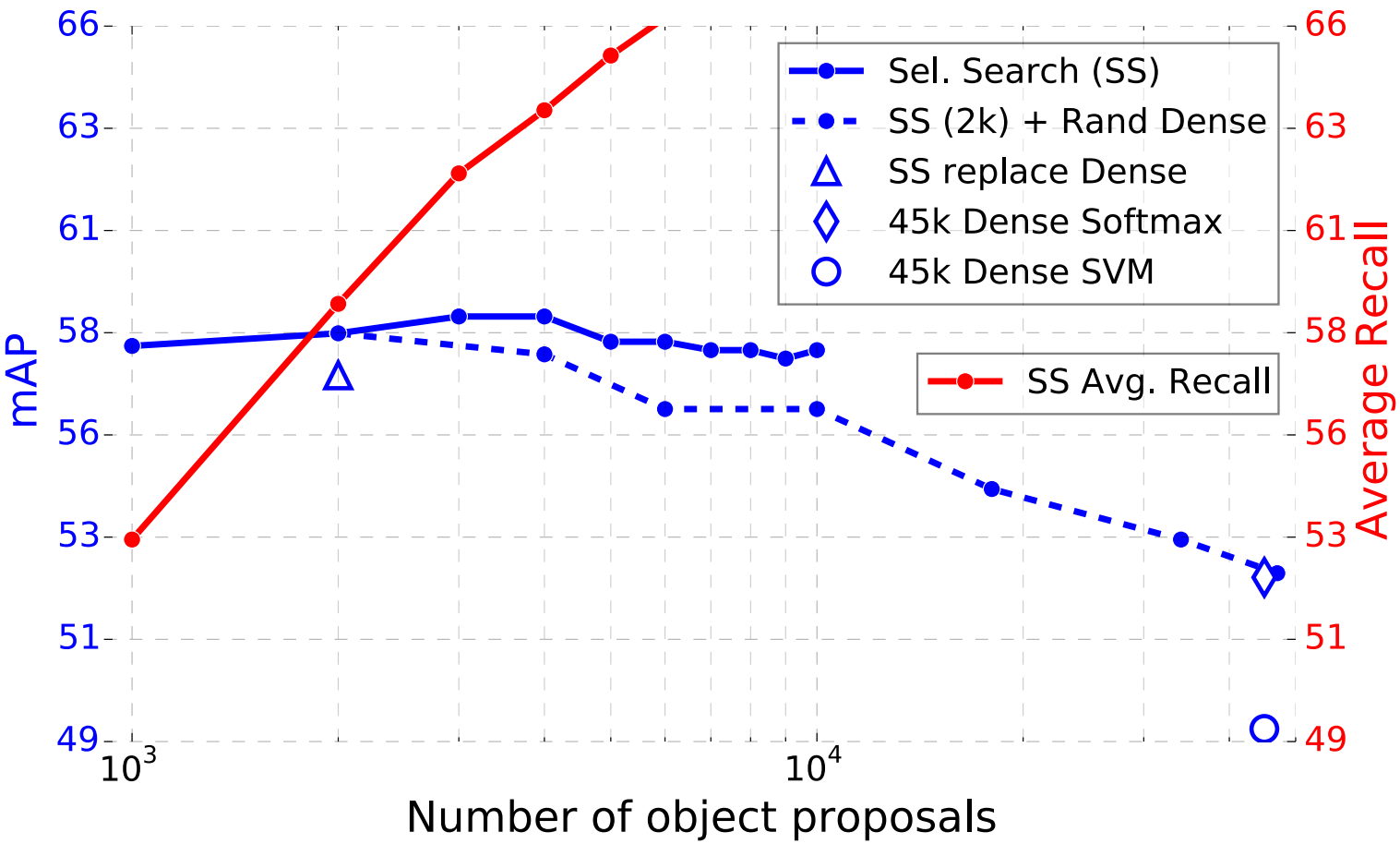
Single-scale testing a good tradeoff

	SPPnet **ZF**		**S**		**M**		**L**
scales	1	5	1	5	1	5	1
test rate (s/im)	0.14	0.38	**0.10**	0.39	0.15	0.64	0.32
VOC07 mAP	58.0	59.2	57.1	58.4	59.2	60.7	**66.9**

- **L** = VGG16, **M** = VGG_CNN_M_1024, **S** = Caffe/AlexNet

Direct region proposal evaluation

- VGG_CNN_M_1024
- Training takes < 2 hours
- Fast training makes these experiments possible



Ross Girshick. "Fast R-CNN". ICCV 2015.

Part 2: Faster R-CNN training

Two algorithms for training Faster R-CNN

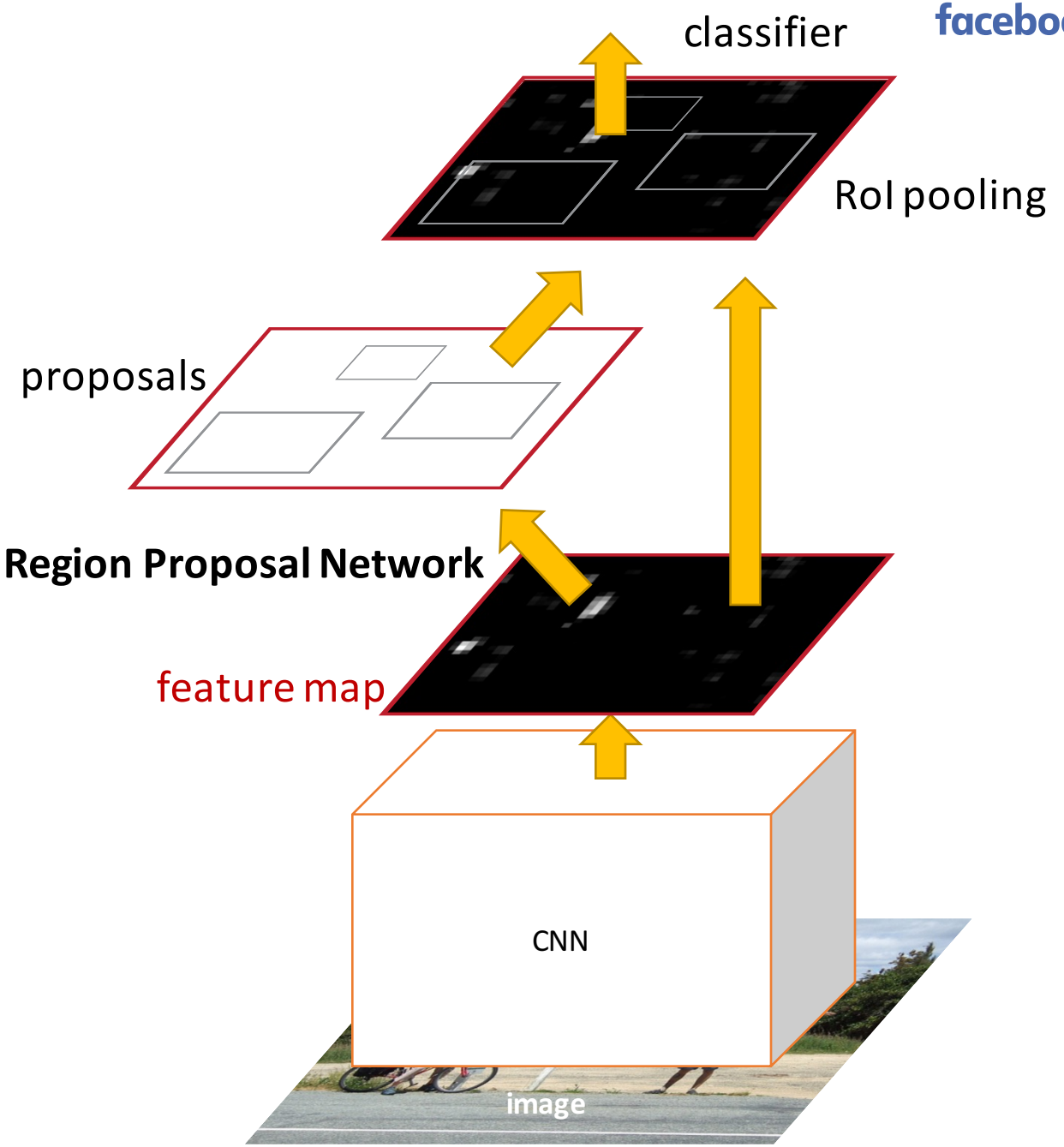
- Alternating optimization
  - Presented in our NIPS 2015 paper
- Approximate joint training
  - Unpublished work, available in the `py-faster-rcnn` Python implementation https://github.com/rbgirshick/py-faster-rcnn
  - Discussion of exact joint training

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

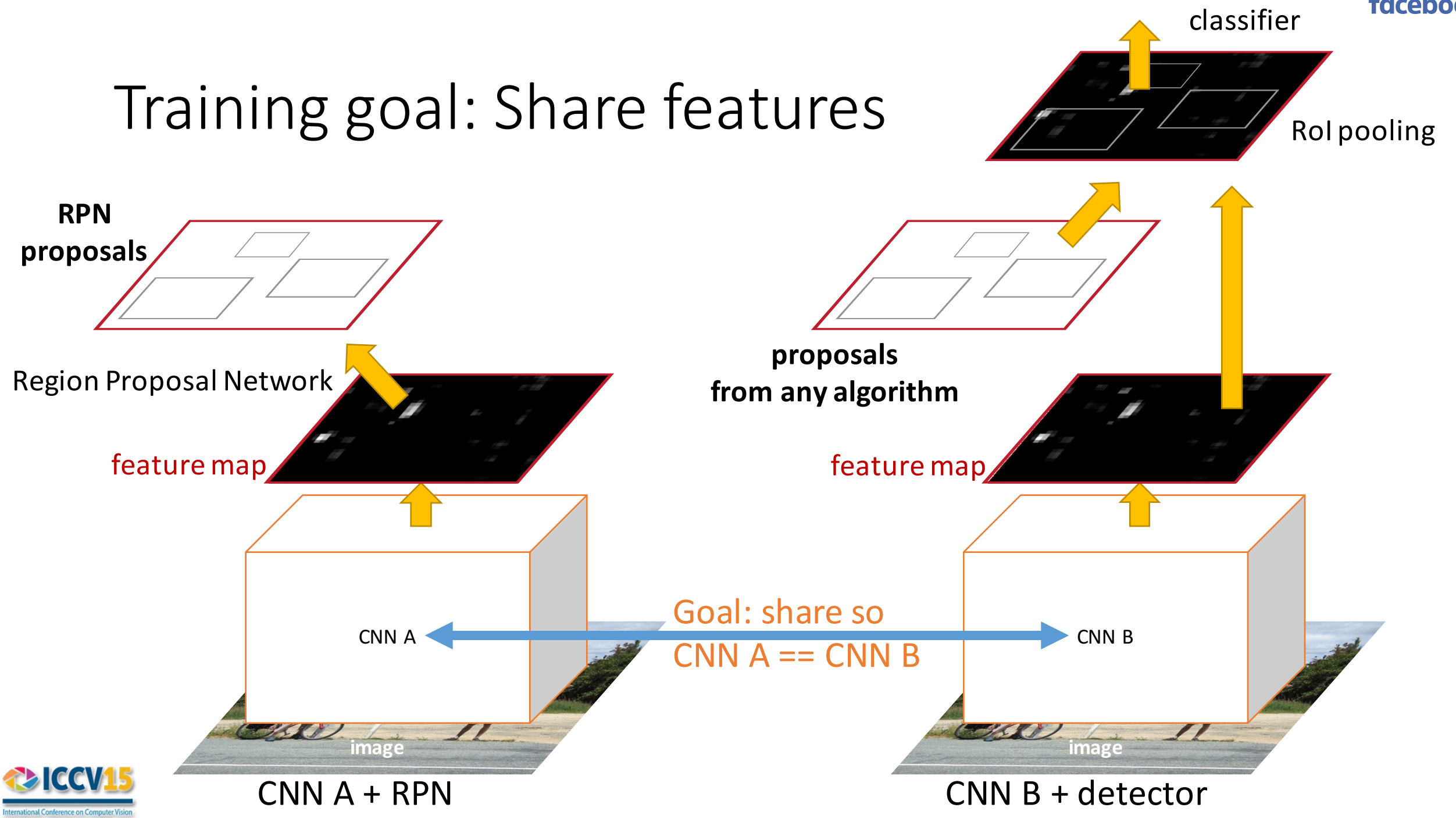
What is Faster R-CNN?

- 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



Training method #1: Alternating optimization

```
# Let M0 be an ImageNet pre-trained network

1.  train_rpn(M0) → M1                  # Train an RPN initialized from M0, get M1
2.  generate_proposals(M1) → P1         # Generate training proposals P1 using RPN M1
3.  train_fast_rcnn(M0, P1) → M2        # Train Fast R-CNN M2 on P1 initialized from M0
4.  train_rpn_frozen_conv(M2) → M3      # Train RPN M3 from M2 without changing conv layers
5.  generate_proposals(M3) → P2
6.  train_fast_rcnn_frozen_conv(M3, P2) → M4  # Conv layers are shared with RPN M3
7.  return add_rpn_layers(M4, M3.RPN)         # Add M3's RPN layers to Fast R-CNN M4
```

Training method #1: Alternating optimization

Motivation behind alternating optimization

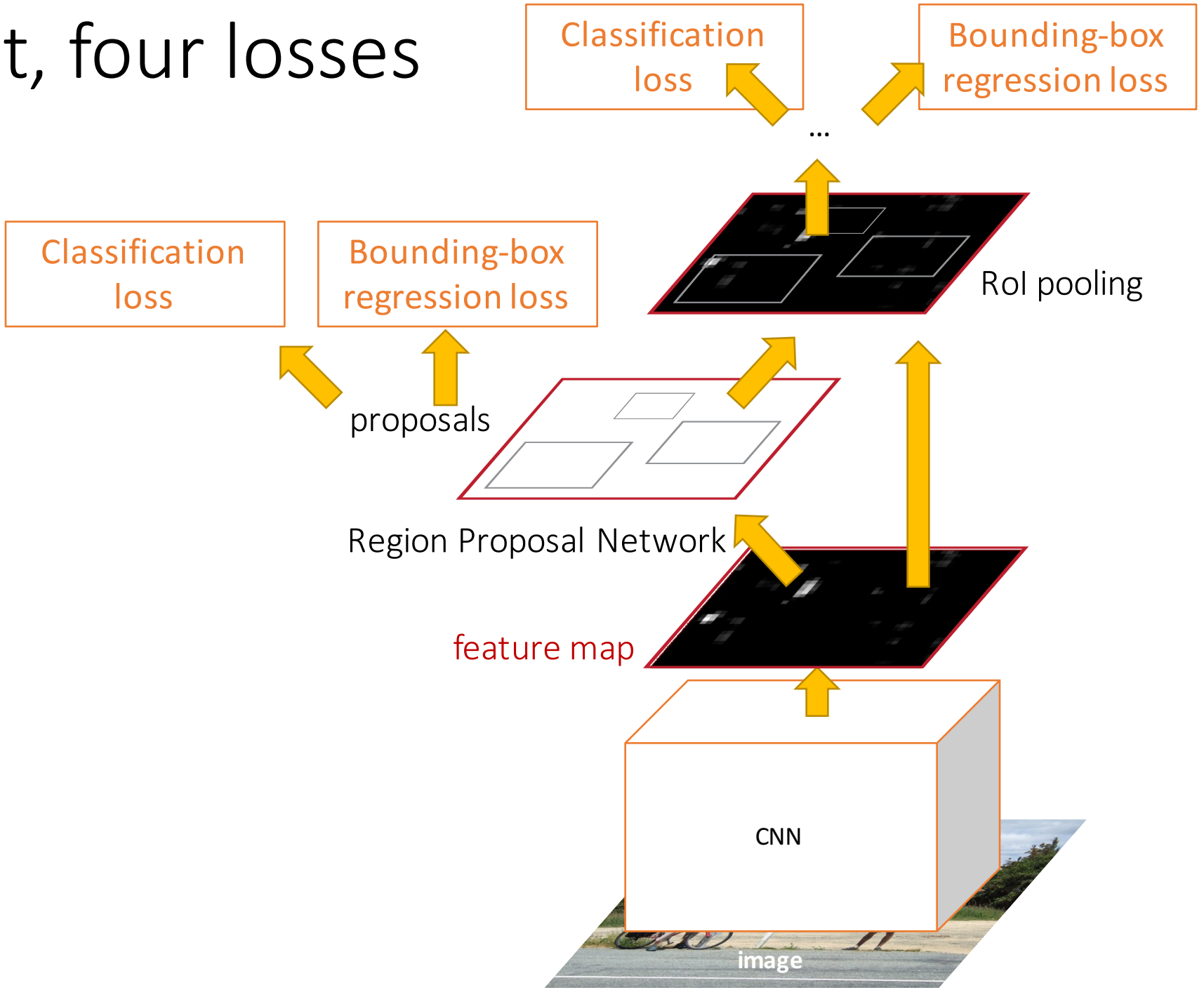
- Not based on any fundamental principles
- Primarily driven by implementation issues and the NIPS deadline ☺
- However, it was unclear if joint training would "just work"
  - Fast R-CNN was always trained on a fixed set of proposals
  - In joint training, the proposal distribution is constantly changing

Training method #2: Approx. joint optimization

Write the network down as a single model and just train it

- Train with SGD as usual
- Even though the proposal distribution changes this just works
  - Implementation challenges eased by writing various modules in Python

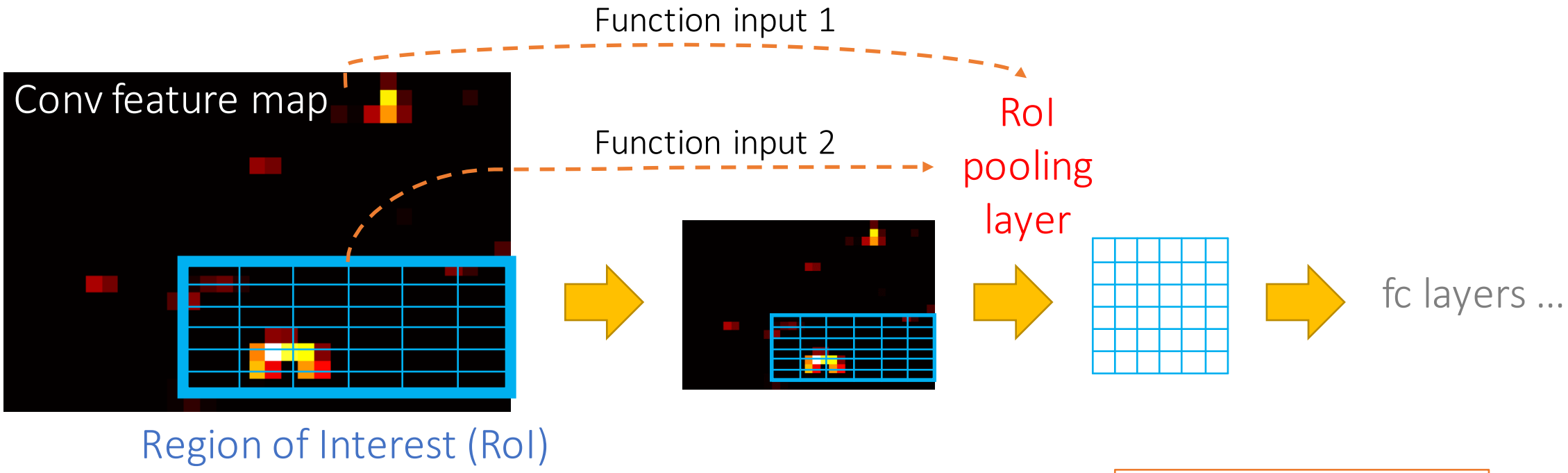
One net, four losses



Training method #2: Approx. joint optimization

Why is this approach approximate?

```
roi_pooling(conv_feat_map, RoI)
```



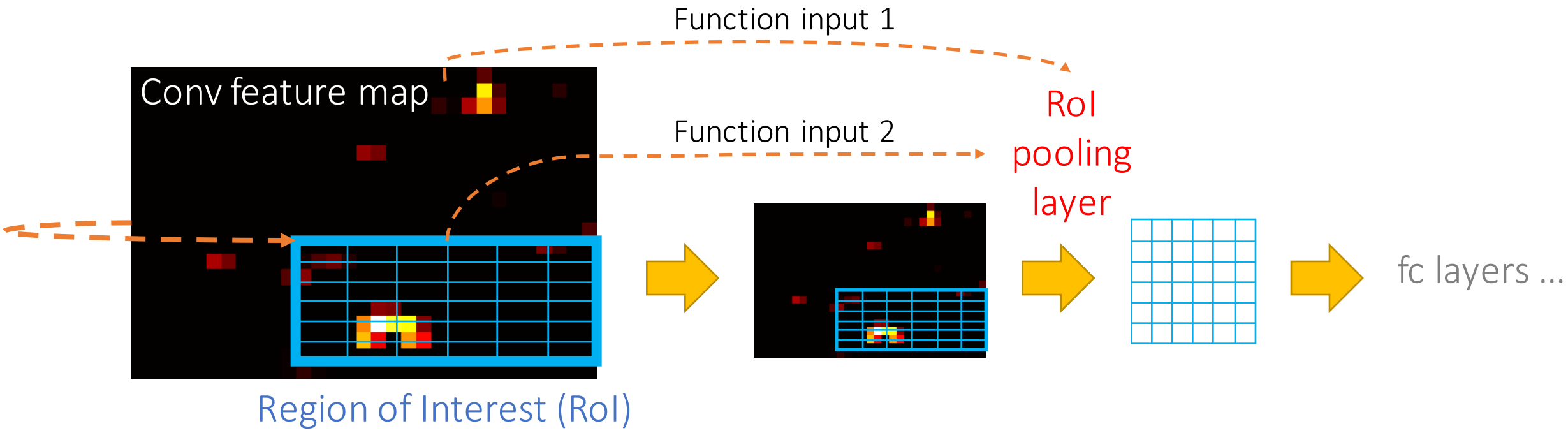
In Fast R-CNN function input 2 (RoI) is a constant, everything is OK

$$\frac{\partial L}{\partial \mathrm{RoI}[i]} = 0 \quad \text{for } i = x_1, y_1, x_2, y_2$$

Training method #2: Approx. joint optimization

Why is this approach approximate?

`roi_pooling(conv_feat_map, RoI)`



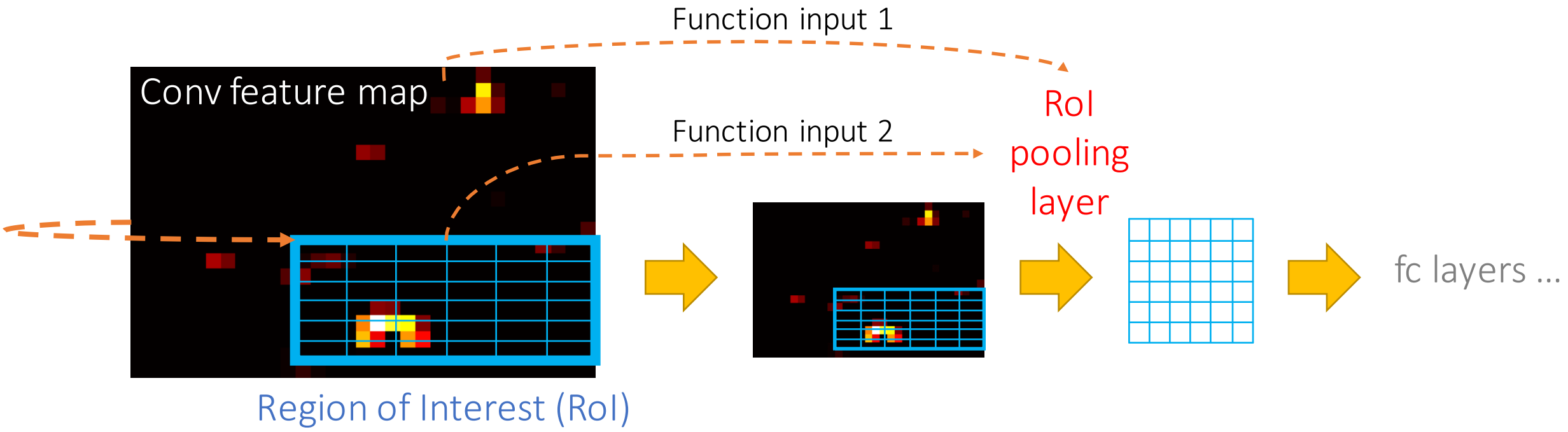
In Faster R-CNN function input 2 (RoI) depends on the input image

$$\frac{\partial L}{\partial \mathrm{RoI}[i]} \neq 0 \text{ in general}$$
$$\text{for } i = x_1, y_1, x_2, y_2$$

Training method #2: Approx. joint optimization

Why is this approach approximate?

`roi_pooling(conv_feat_map, RoI)`



However, $\frac{\partial L}{\partial \mathrm{RoI}[i]}$ is actually undefined,

`roi_pooling()` is not differentiable w.r.t. RoI

$\frac{\partial L}{\partial \mathrm{RoI}[i]}$ does not even exist for $i = x_1, y_1, x_2, y_2$

Training method #2: Approx. joint optimization

What happens in practice?

- We ignore the undefined derivatives of loss w.r.t. RoI coordinates
  - Run SGD with this “surrogate” gradient
- This just works
- Why?
  - RPN network receives direct supervision
  - Error propagation from RoI pooling might help, but is not strictly needed

Faster R-CNN exact joint training

- Modify RoI pooling so that it's a differentiable function of *both* the input conv feature map and input RoI coordinates
- One option (untested, theoretical)
  - Use a differentiable sampler instead of max pooling
  - If RoI pooling uses bilinear interpolation instead of max pooling, then we can compute a derivatives w.r.t. the RoI coordinates
  - See: Jaderberg *et al.* "Spatial Transformer Networks" NIPS 2015.

Experimental findings
(`py-faster-rcnn` implementation)

- Approximate joint training gives similar mAP to alternating optimization
- VOC07 test mAP
  - Alt. opt.: 69.9%
  - Approx. joint: 70.0%
- Approximate joint training is faster and simpler
  - Alt. opt.: 26.2 hours
  - Approx. joint: 17.2 hours

Code pointers

- Fast R-CNN (Python): https://github.com/rbgirshick/fast-rcnn
- Faster R-CNN (matlab): https://github.com/ShaoqingRen/faster_rcnn
- Faster R-CNN (Python): https://github.com/rbgirshick/py-faster-rcnn
  - Now includes code for approximate joint training

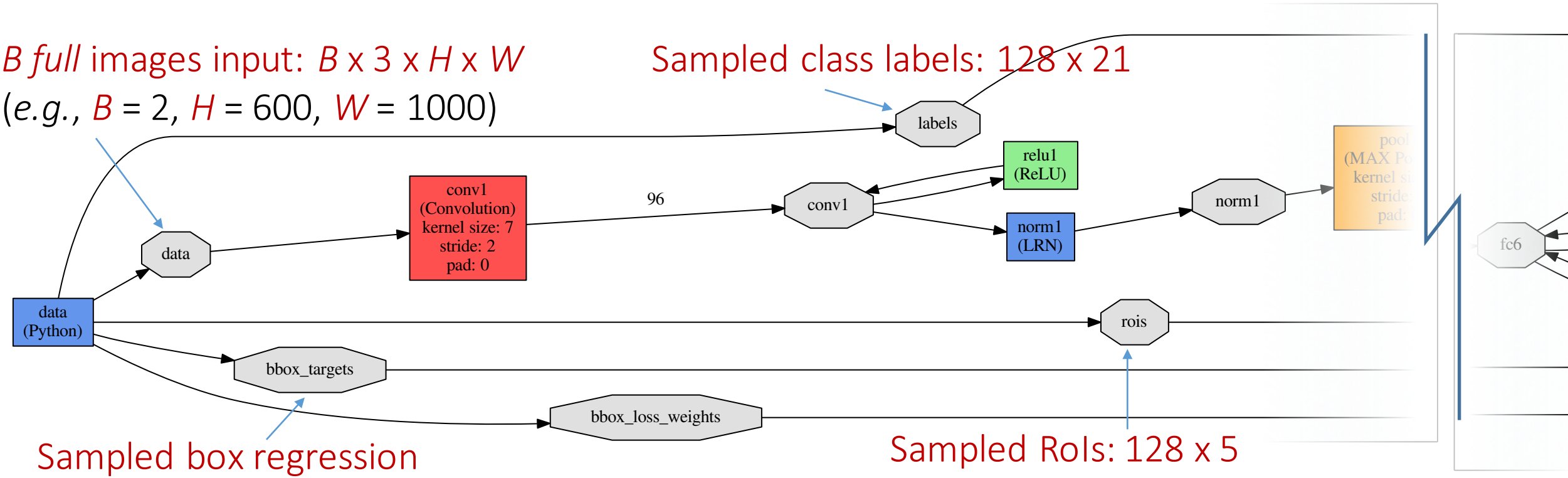
Reproducible research – get the code!

Extra slides follow

Fast R-CNN unified network

B full images input: *B* x 3 x *H* x *W*
(*e.g.*, *B* = 2, *H* = 600, *W* = 1000)

Sampled class labels: 128 x 21

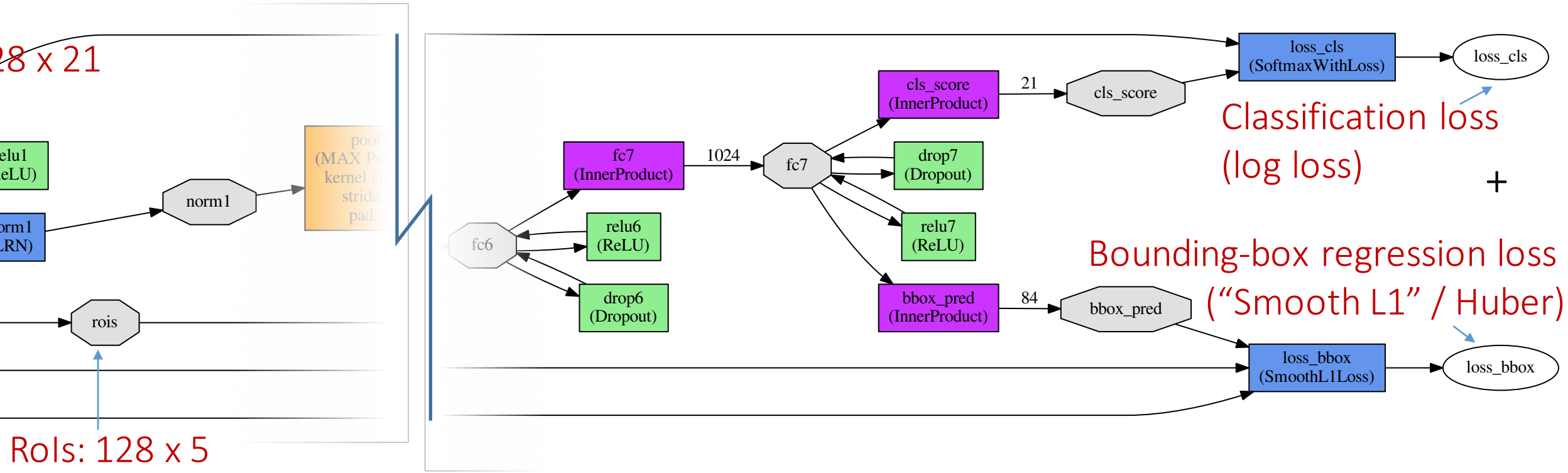


Sampled box regression targets: 128 x 84

Sampled RoIs: 128 x 5

Ross Girshick. "Fast R-CNN". ICCV 2015.

Fast R-CNN unified network



Ross Girshick. "Fast R-CNN". ICCV 2015.