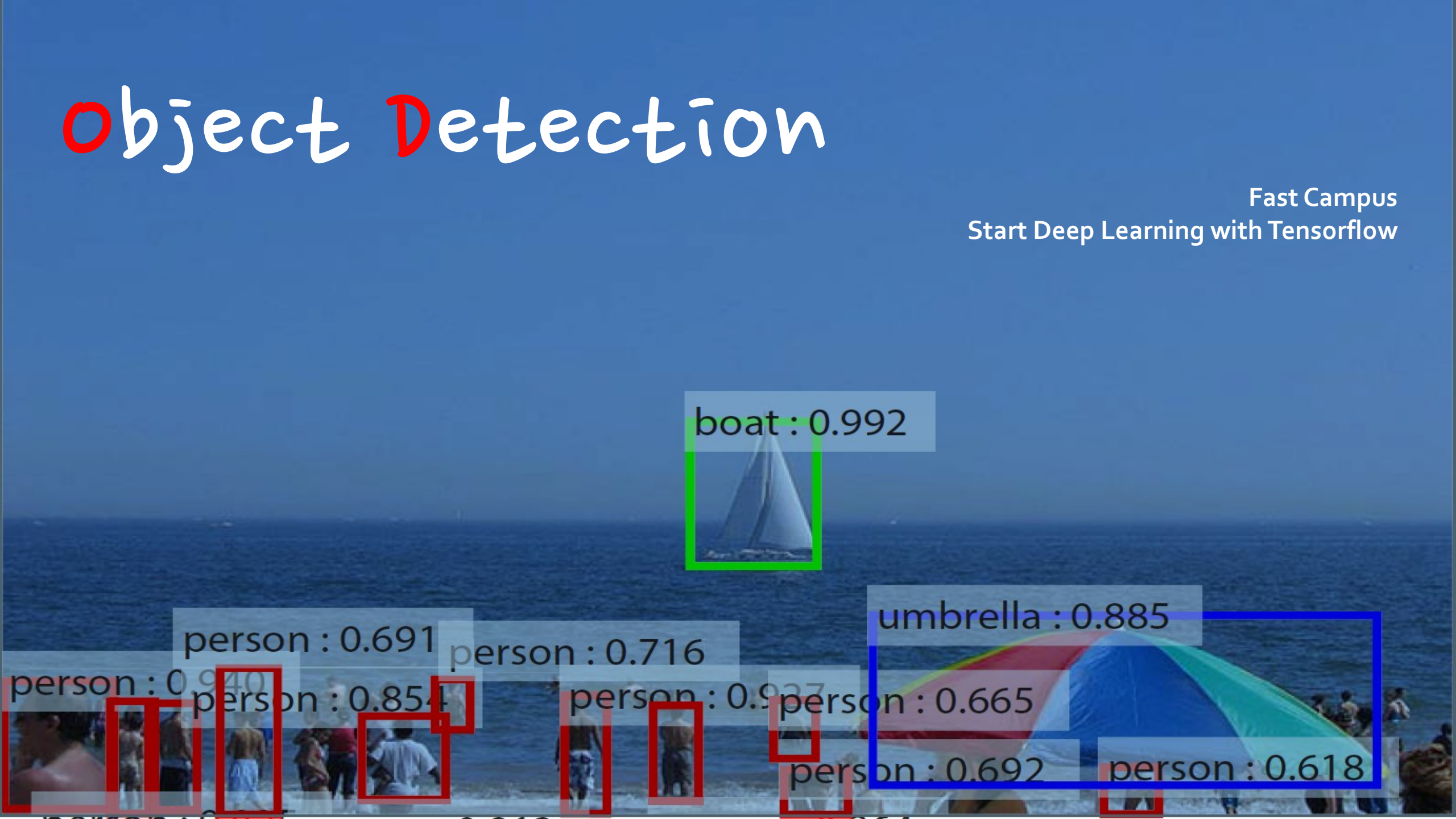
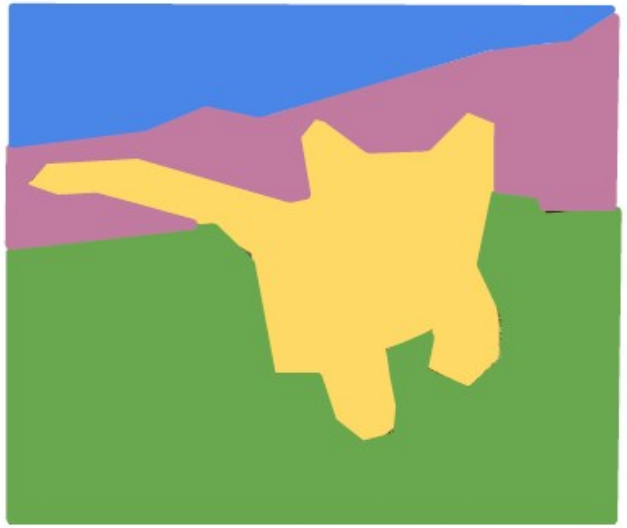
Object Detection

Fast Campus
Start Deep Learning with Tensorflow



Computer Vision Task

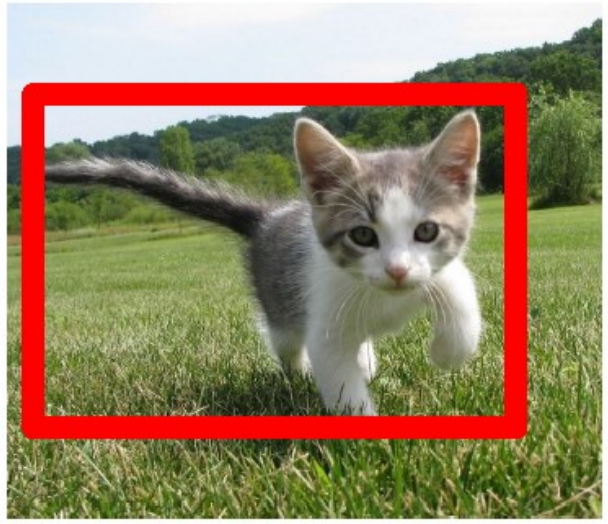
Semantic Segmentation



GRASS, CAT, TREE, SKY

No objects, just pixels

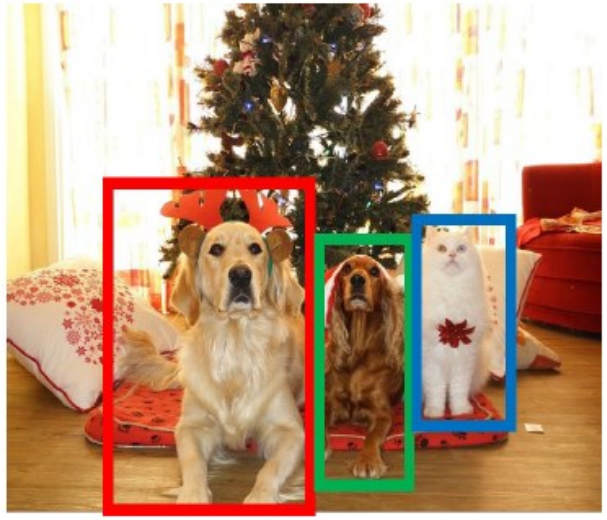
Classification + Localization



CAT

Single Object

Object Detection



DOG, DOG, CAT

Instance Segmentation

DOG, DOG, CAT

Multiple Object

This image is CC0 public domain

Classification+Localization

Classification: C classes
Input: Image
Output: Class label
Evaluation metric: Accuracy

→ CAT

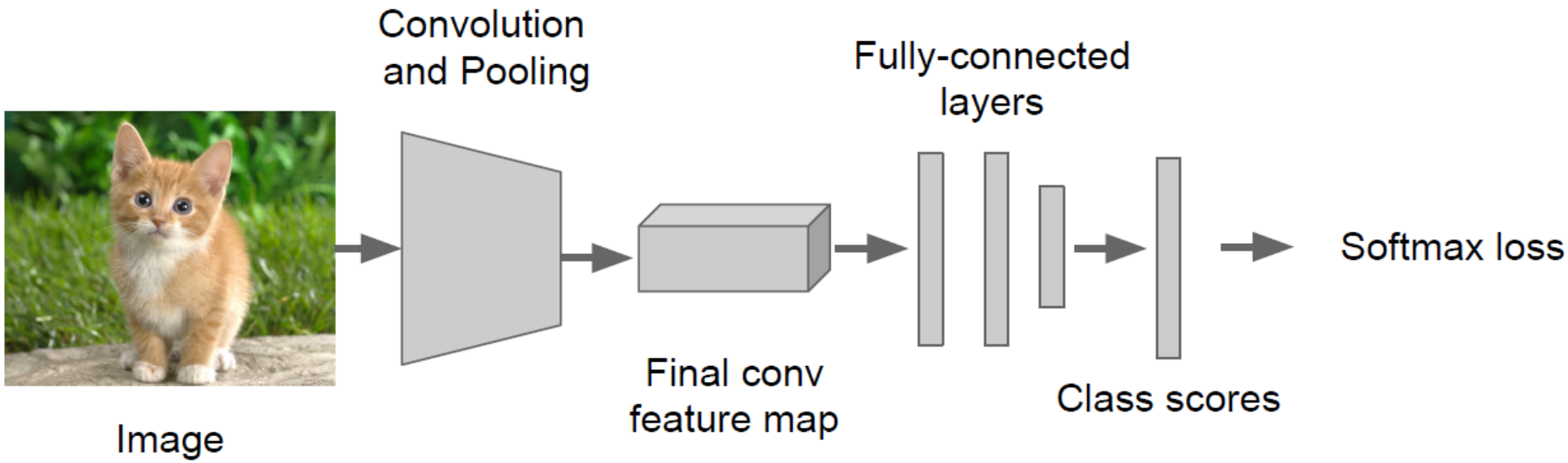
Localization:
Input: Image
Output: Box in the image (x, y, w, h)
Evaluation metric: Intersection over Union

→ (x, y, w, h)

Classification + Localization: Do both

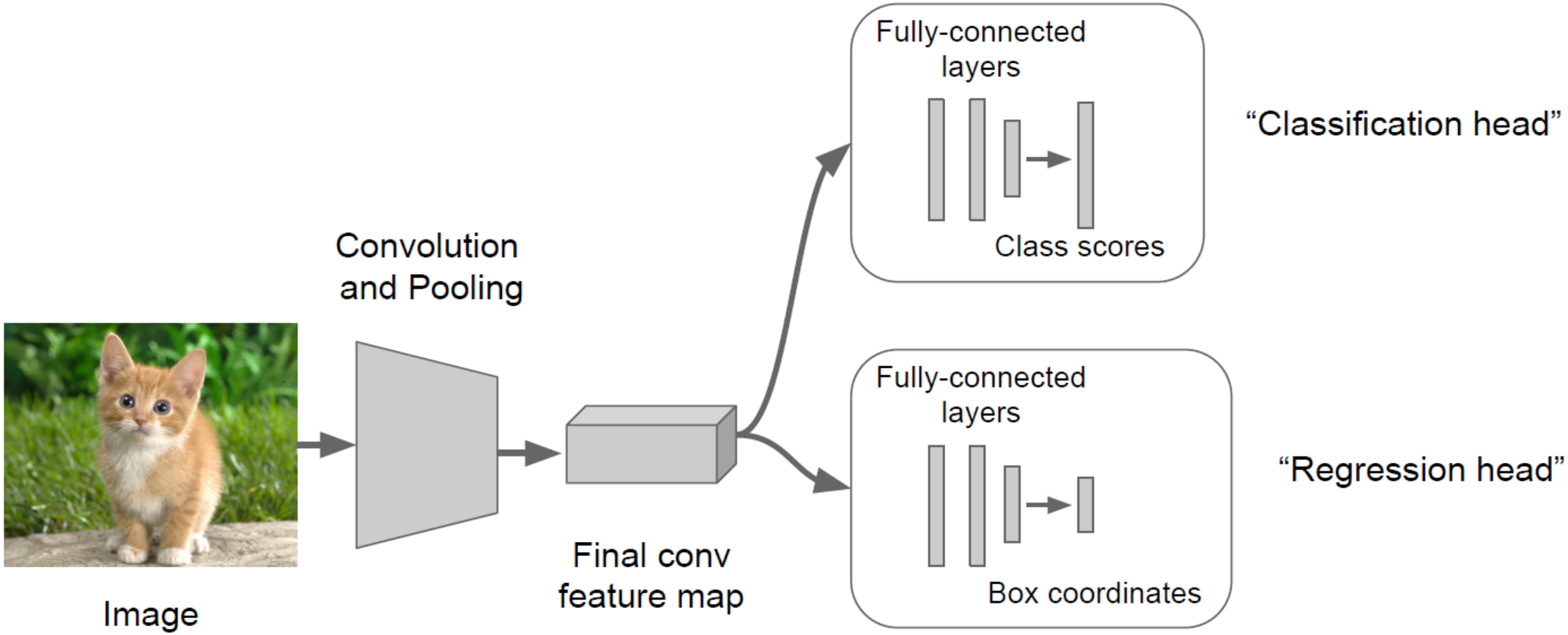
Simple Idea

Step 1: Train (or download) a classification model (AlexNet, VGG, GoogLeNet)



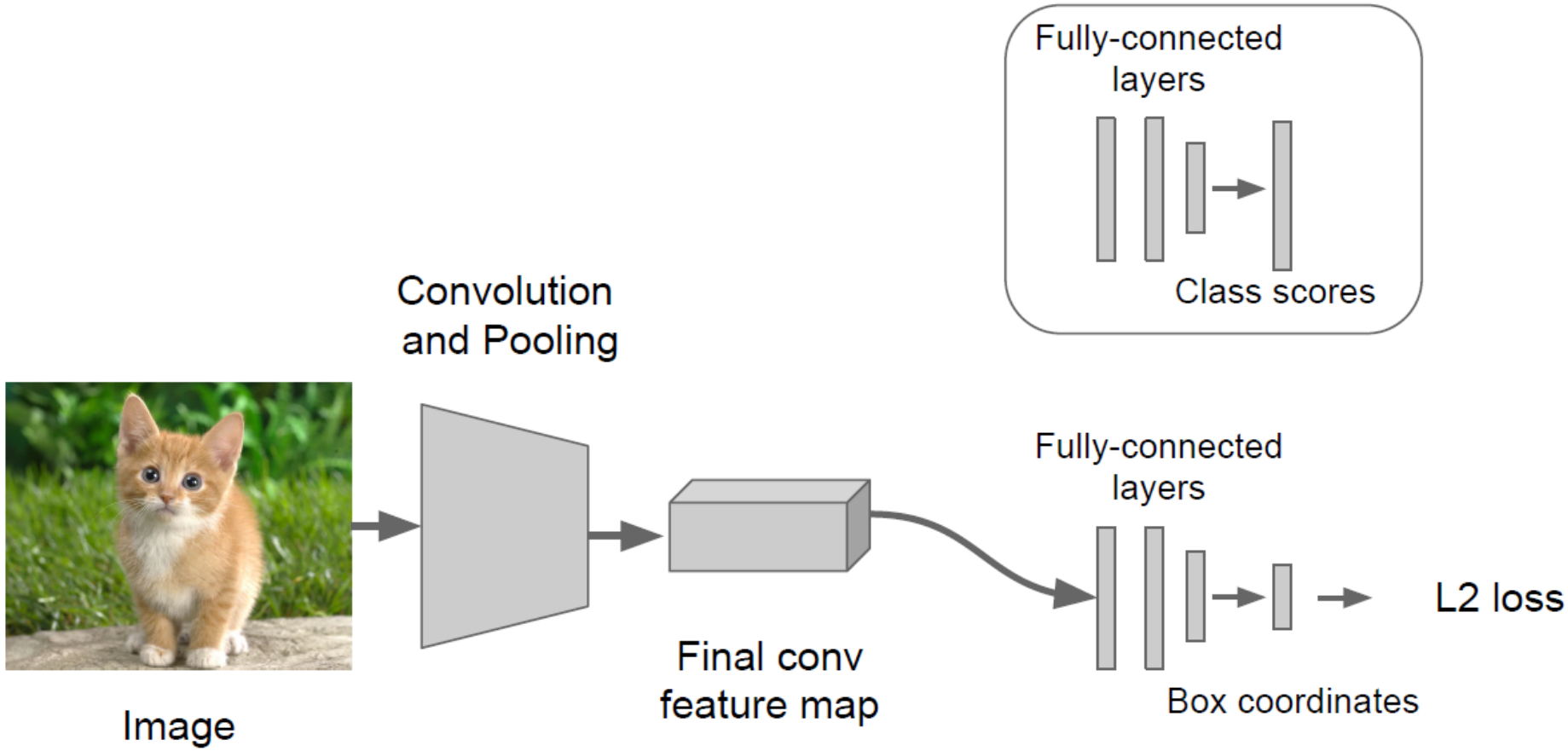
Simple Idea

Step 2: Attach new fully-connected "regression head" to the network



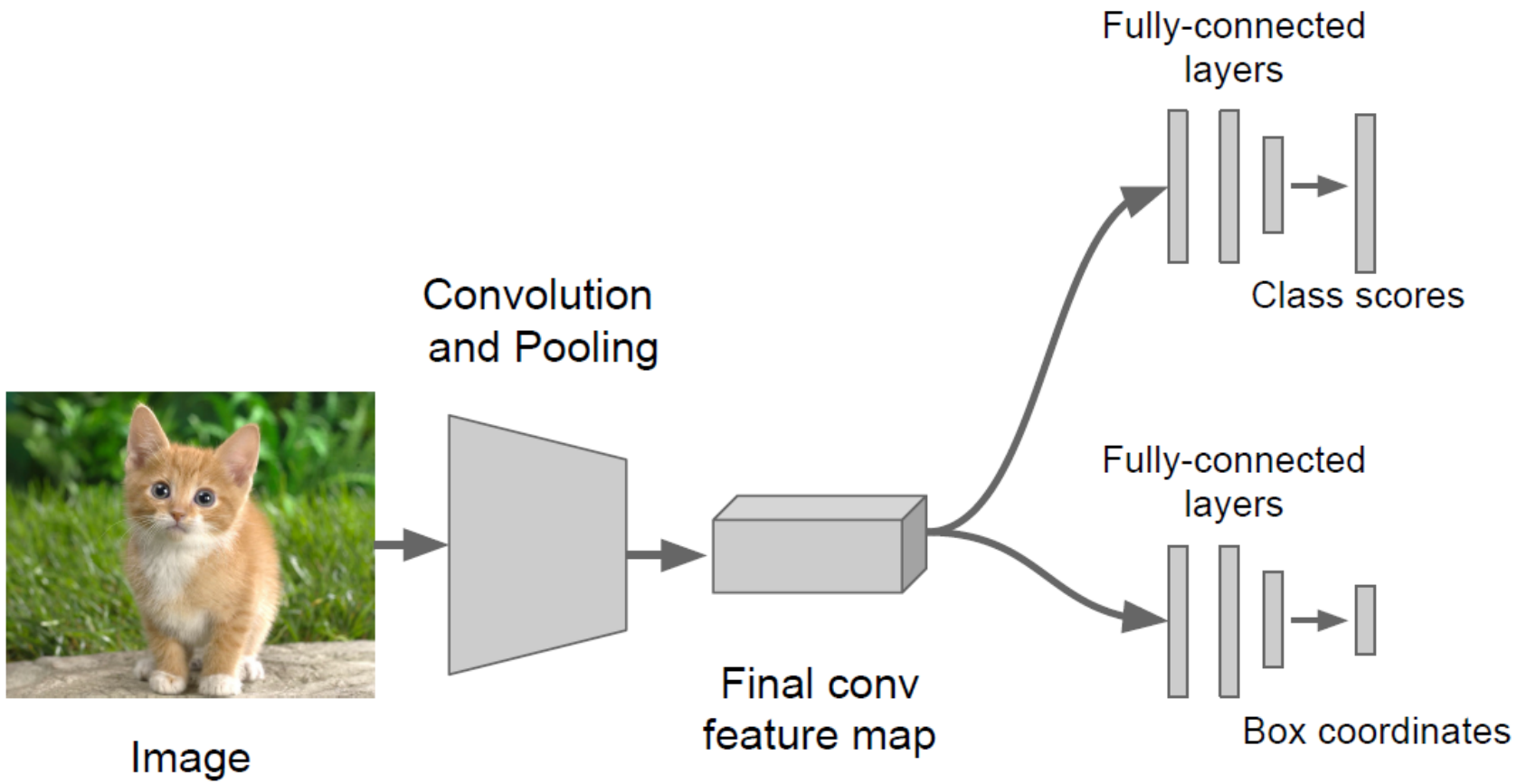
Simple Idea

Step 3: Train the regression head only with SGD and L2 loss



Simple Idea

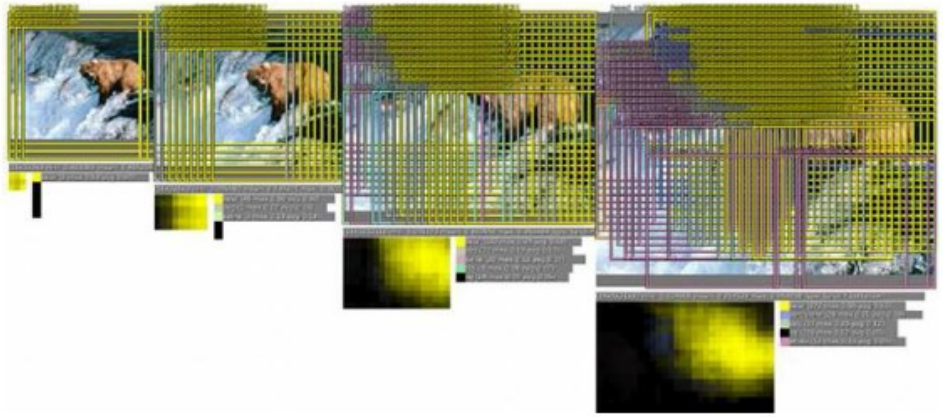
Step 4: At test time use both heads



How to Find BBox

- Intuitive Approach – Sliding window
  - Simple but takes a lot of time
  - Multi-scale testing is necessary

  - Procedure
    - ➢ Perform classification at each location
    - ➢ Perform bounding-box regression on all the classified regions
    - ➢ Merge bounding boxes

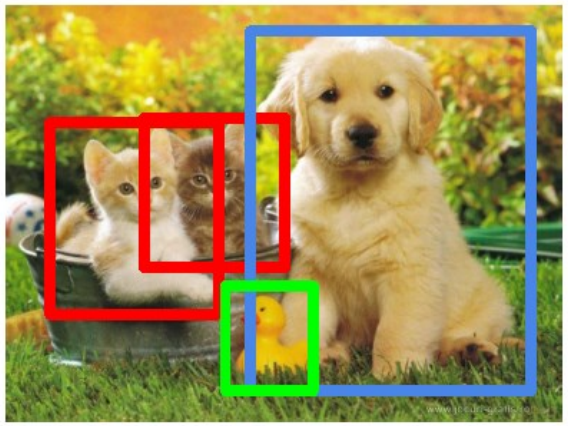


Object Detection

Classification

Classification + Localization

Object Detection



Instance Segmentation

Detection as Regression?

CAT, (x, y, w, h)
CAT, (x, y, w, h)
....
CAT (x, y, w, h)

= many numbers

Need variable sized outputs

History(?) of R-CNN Series

- Rich feature hierarchies for accurate object detection and semantic segmentation(2013)
- Fast R-CNN(2015)
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks(2015)
- Mask R-CNN(2017)

R-CNN

1. Generating category independent region proposals
2. Extracting a fixed length feature vector from CNN
3. Class specific linear SVM

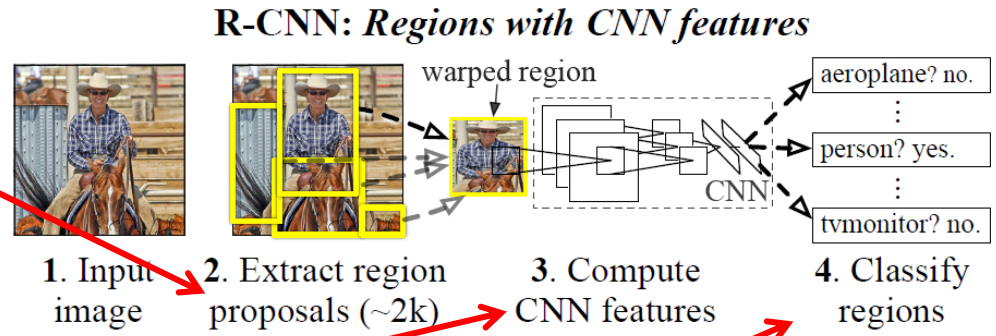
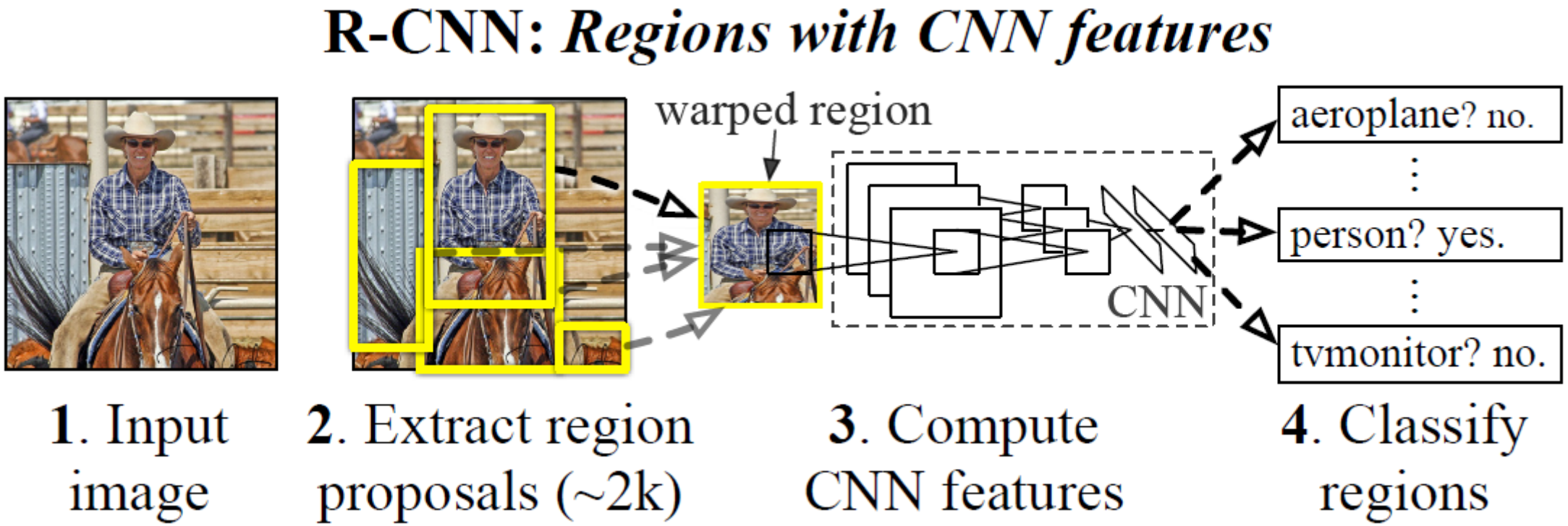
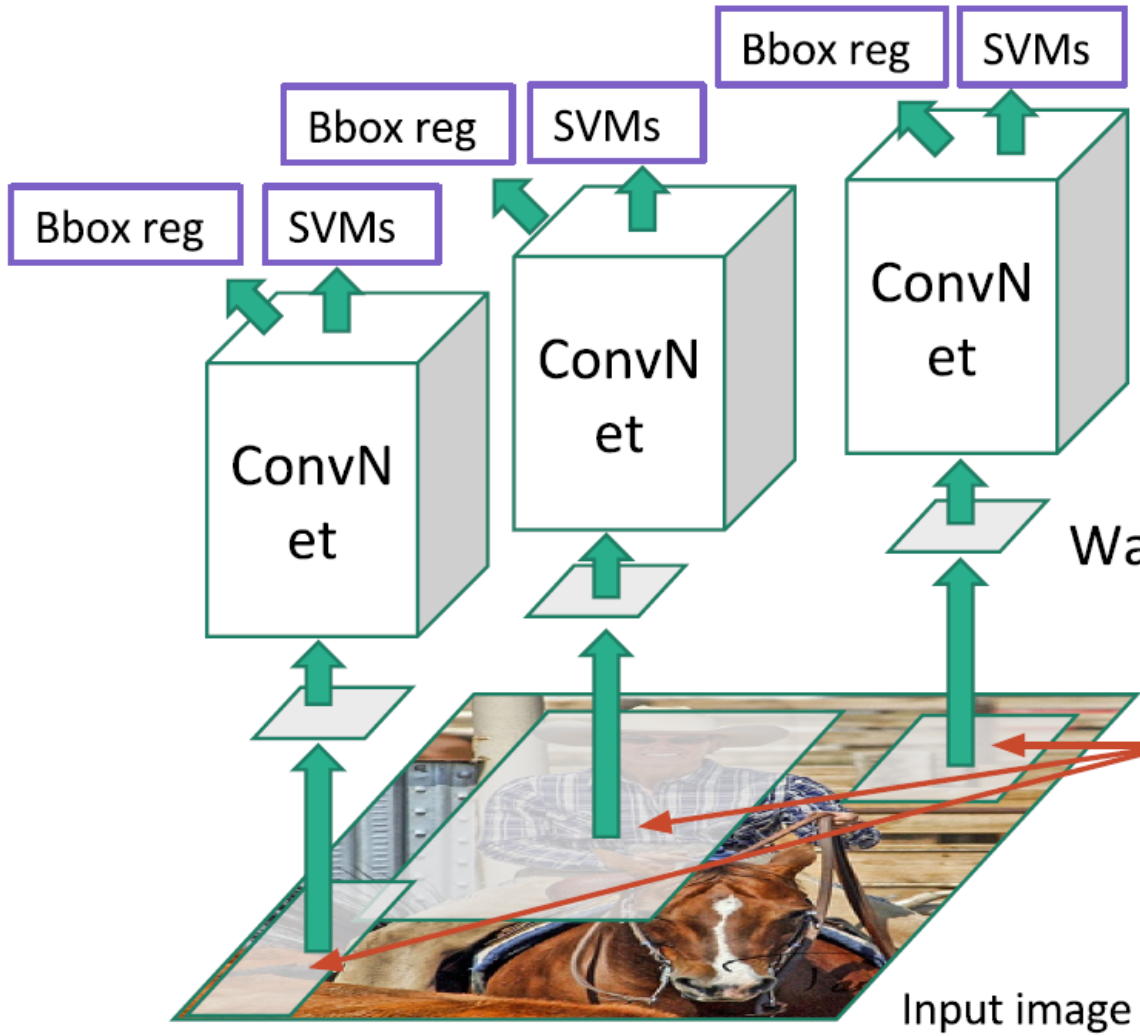


Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [39] reports 35.1% mAP using the same region proposals, but with a spatial pyramid and bag-of-visual-words approach. The popular deformable part models perform at 33.4%. On the 200-class **ILSVRC2013 detection dataset, R-CNN's mAP is 31.4%**, a large improvement over OverFeat [34], which had the previous best result at 24.3%.

R-CNN Architecture



R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions

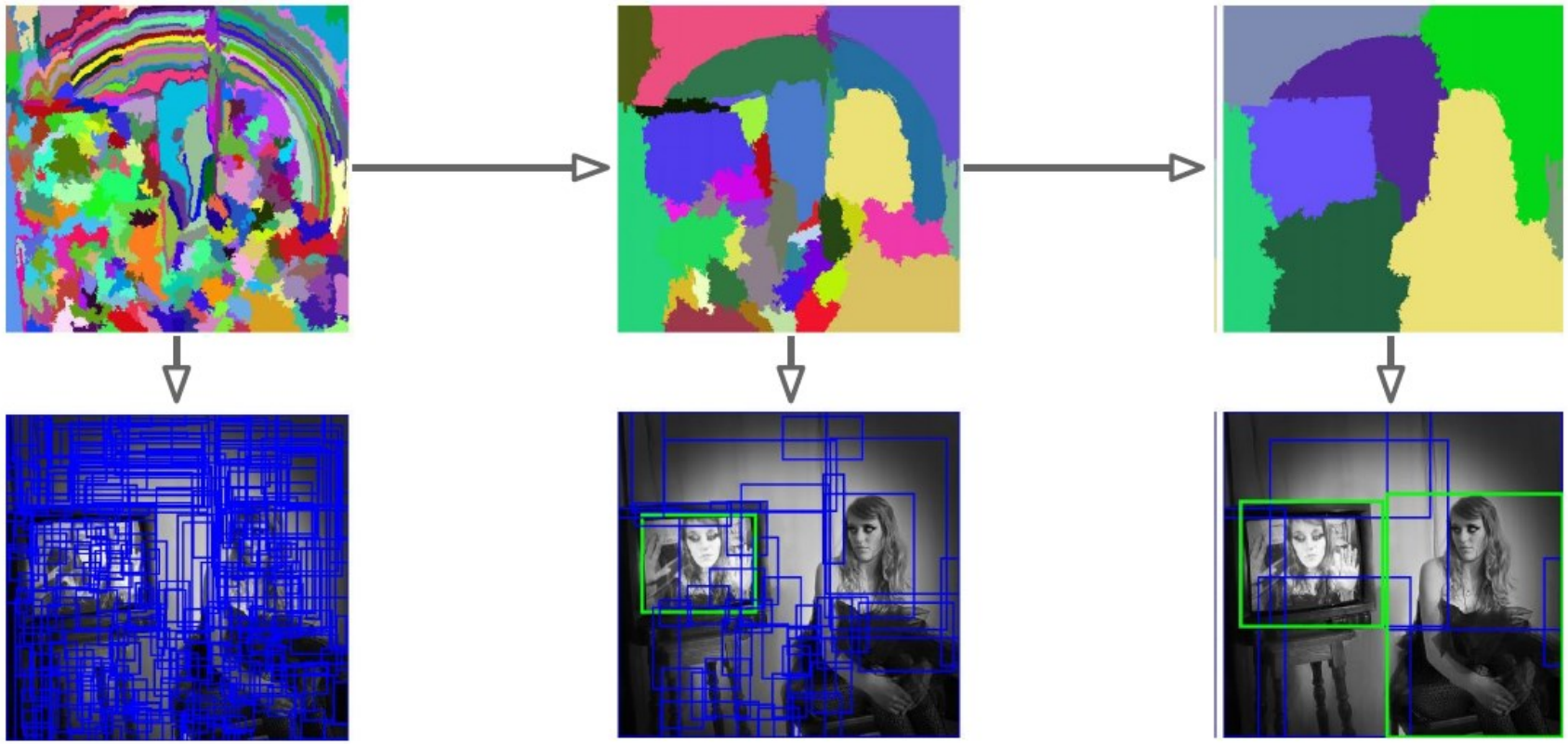
Regions of Interest (RoI) from a proposal method (~2k)

Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

Region Proposals – Selective Search

- Bottom-up segmentation, merging regions at multiple scales

Convert regions to boxes



Feature Extraction

- AlexNet is used

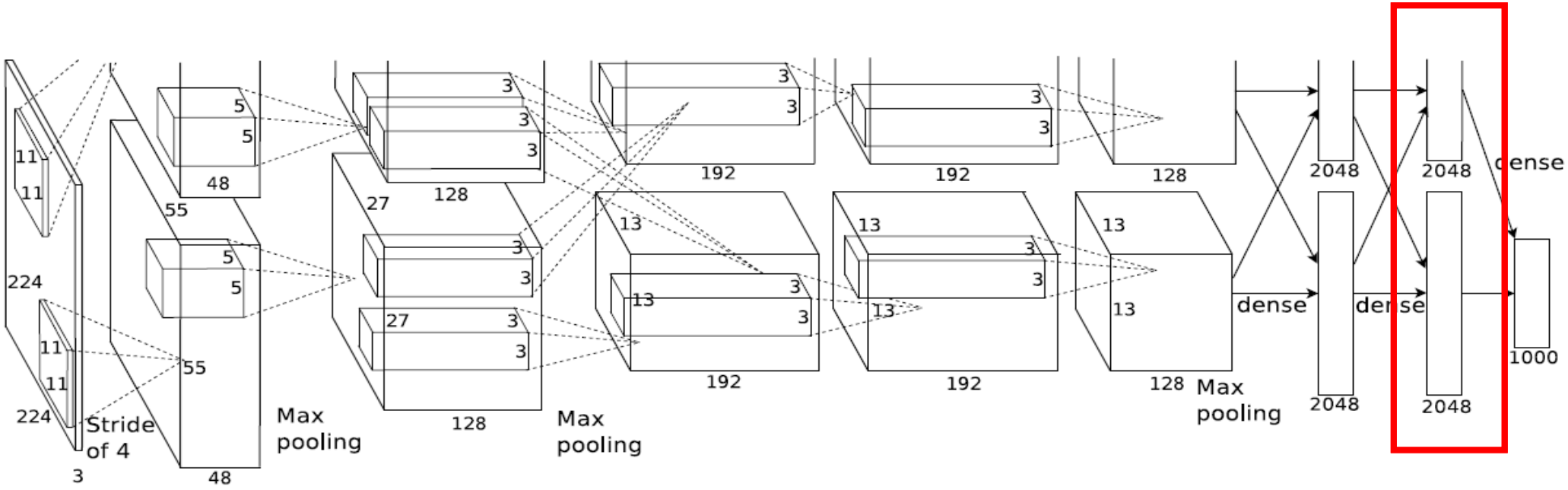
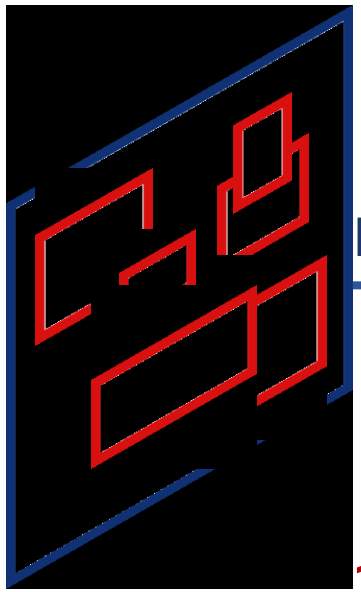


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

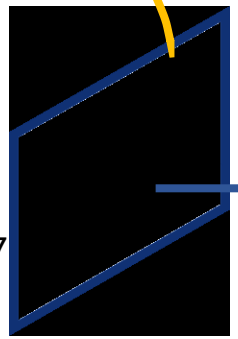
Test Time

2,000 region proposals are extracted



Reshape(warping)

227



CNN

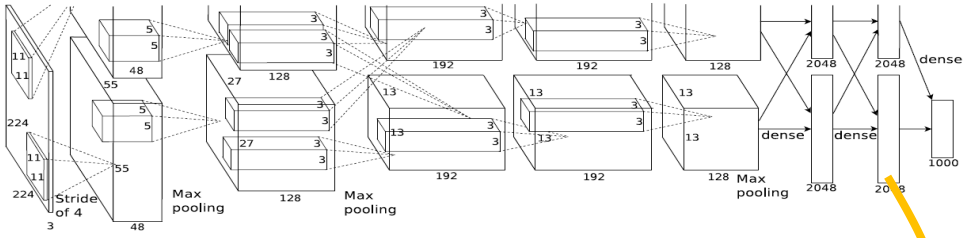


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

4096

SVM

20+1

13s/image on a GPU

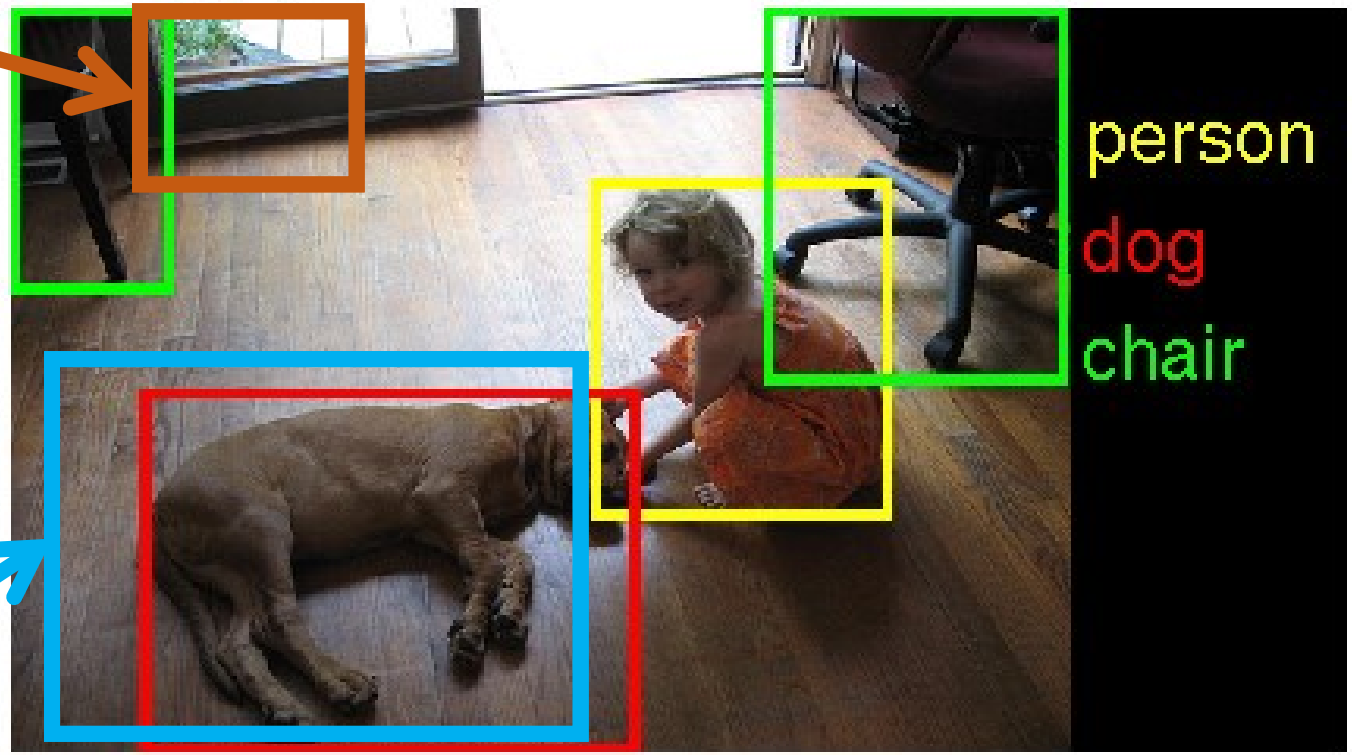
53s/image on a CPU

Training(Region Proposal)

Positive : groundtruth

Negative (BG) : if IoU ≤ 0.3

Training SVM



Fine-tuning

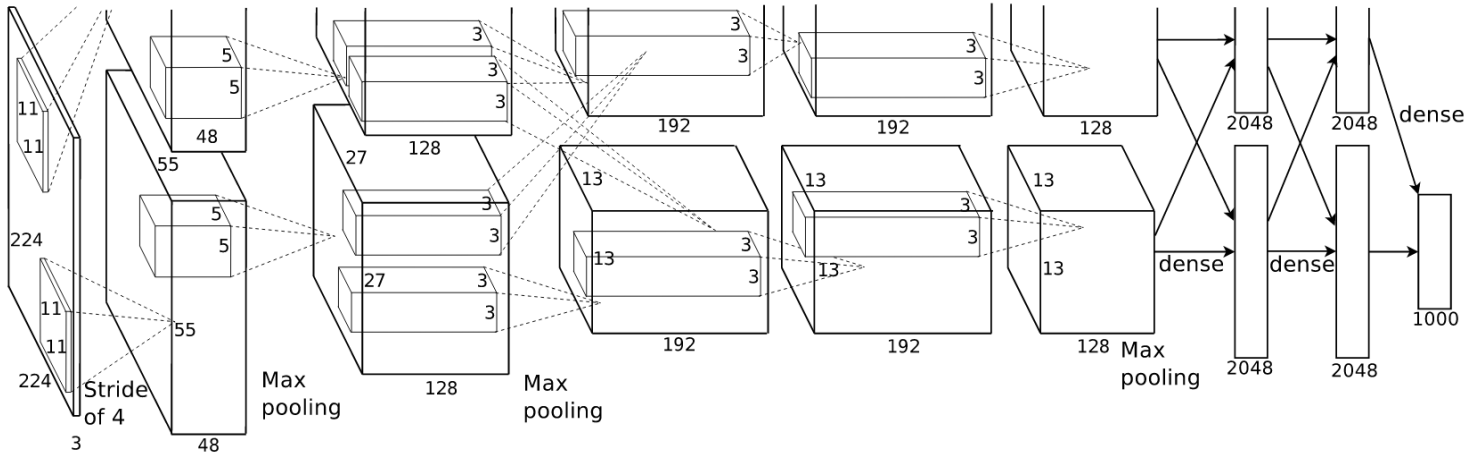
Positive (FG) : if IoU ≥ 0.5

Negative (BG) : else

R-CNN Training

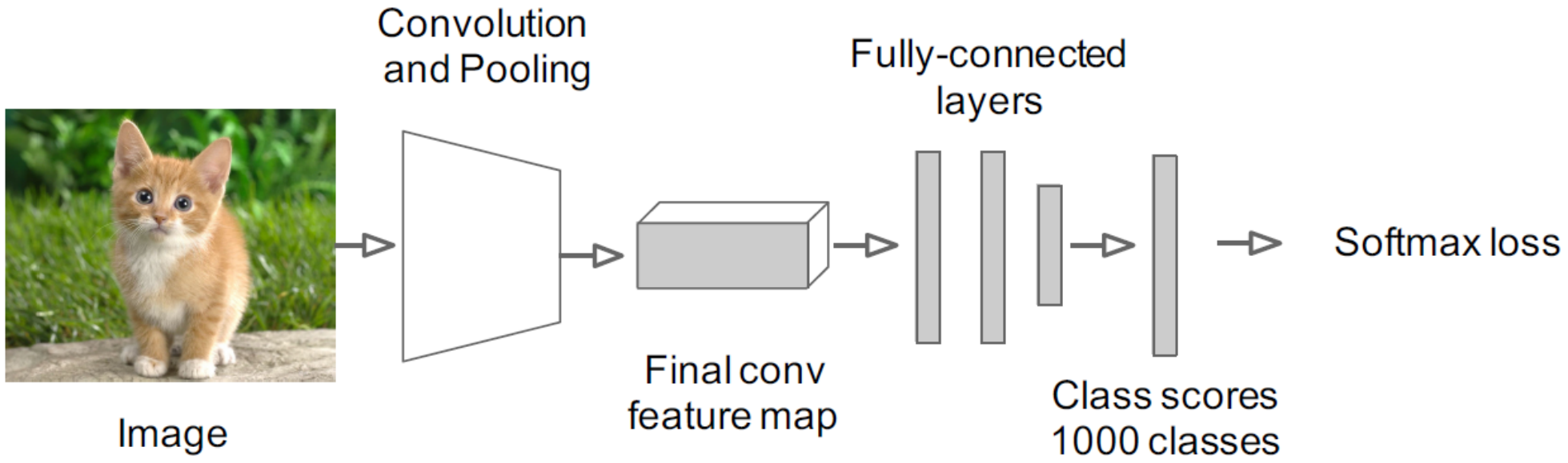
"Post hoc" means the parameters are learned after the ConvNet is fixed

- Pre-train a ConvNet(AlexNet) for ImageNet classification dataset
- Fine-tune for object detection(softmax + log loss)
- Cache feature vectors to disk
- Train post hoc linear SVMs(hinge loss)
- Train post hoc linear bounding-box regressors(squared loss)



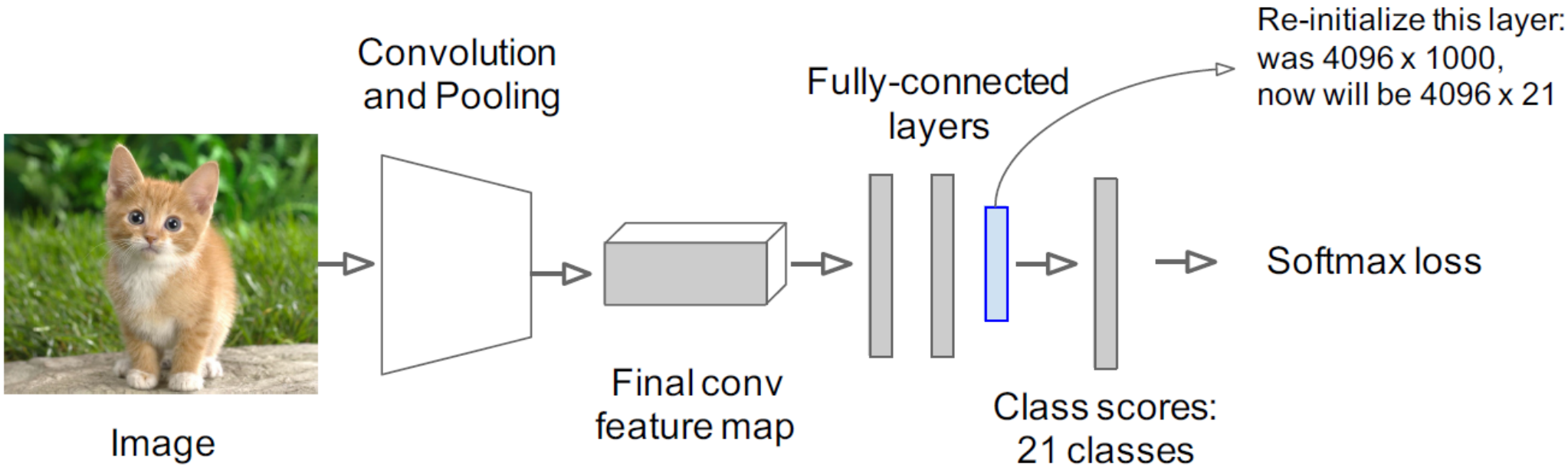
Training

- Step 1: Train (or download) a classification model for ImageNet (AlexNet)



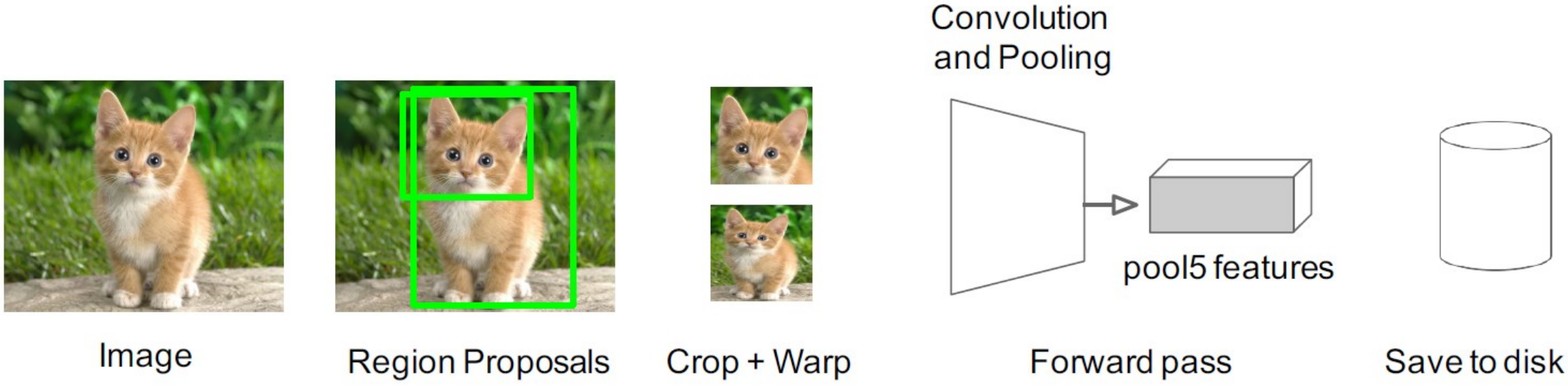
Training

- Step 2: Fine-tune model for detection
  - Instead of 1000 ImageNet classes, want 20 object classes + background
  - Throw away final fully-connected layer, reinitialize from scratch
  - Keep training model using positive / negative regions from detection images



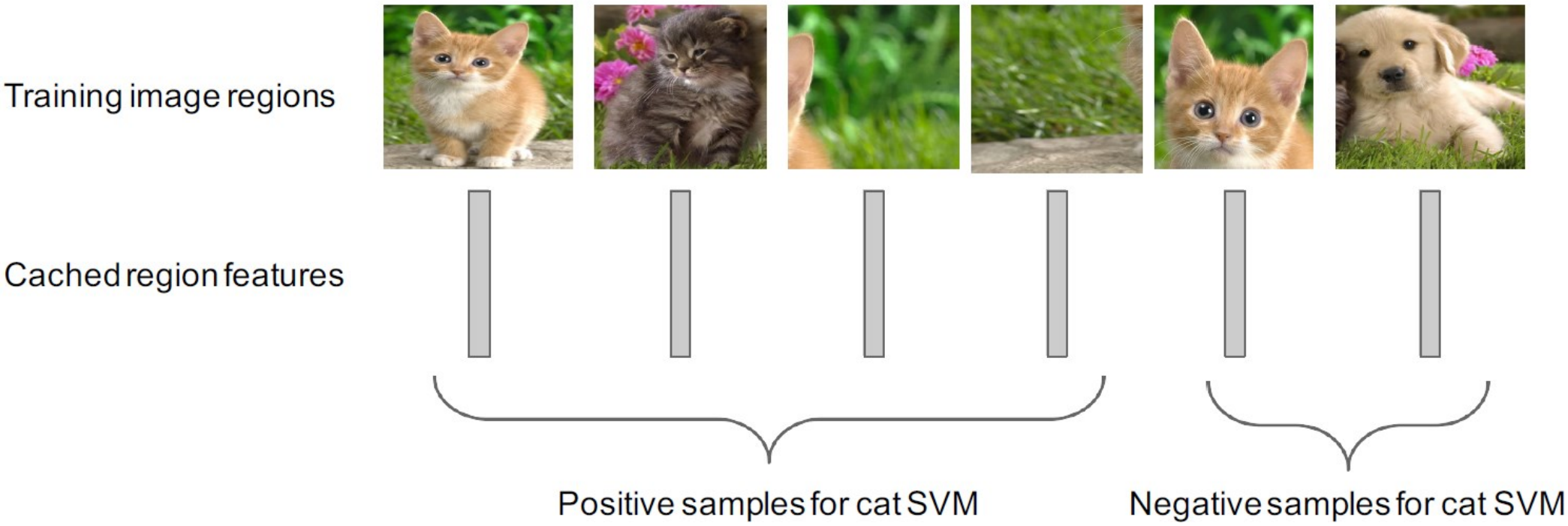
Training

- Step 3: Extract features
  - Extract region proposals for all images
  - For each region: warp to CNN input size, run forward through CNN, save pool5 features to disk
  - Have a big hard drive: features are ~200GB for PASCAL dataset!



Training

- Step 4: Train one binary SVM per class to classify region features



Training

- Step 5 (bbox regression): For each class, train a linear regression model to map from cached features to offsets to GT boxes to make up for "slightly wrong" proposals

Bounding-Box Regression

$P^i = (P^i_x, P^i_y, P^i_w, P^i_h)$ specifies the pixel coordinates of the center of proposal P^i's bounding box together with P^i's width and height in pixels

$G = (G_x, G_y, G_w, G_h)$ means the ground-truth bounding box

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

Bounding-Box Regression

Each function $d_\star(P)$ (where $\star$ is one of x, y, h, w) is modeled as a linear function of the pool_5 features of proposal P, denoted by $\phi_5(P)$. (The dependence of $\phi_5(P)$ on the image data is implicitly assumed.) Thus we have $d_\star(P) = \mathbf{w}_\star^{\mathrm{T}} \phi_5(P)$, where $\mathbf{w}_\star$ is a vector of learnable model parameters. We learn $\mathbf{w}_\star$ by optimizing the regularized least squares objective (ridge regression):

$$\mathbf{w}_\star = \underset{\hat{\mathbf{w}}_\star}{\operatorname{argmin}} \sum_i^N (t_\star^i - \hat{\mathbf{w}}_\star^{\mathrm{T}} \phi_5(P^i))^2 + \lambda \left\| \hat{\mathbf{w}}_\star \right\|^2 . \quad (5)$$

$$\hat{G}_x = P_w d_x(P) + P_x \quad (1)$$

$$\hat{G}_y = P_h d_y(P) + P_y \quad (2)$$

$$\hat{G}_w = P_w \exp(d_w(P)) \quad (3)$$

$$\hat{G}_h = P_h \exp(d_h(P)). \quad (4)$$

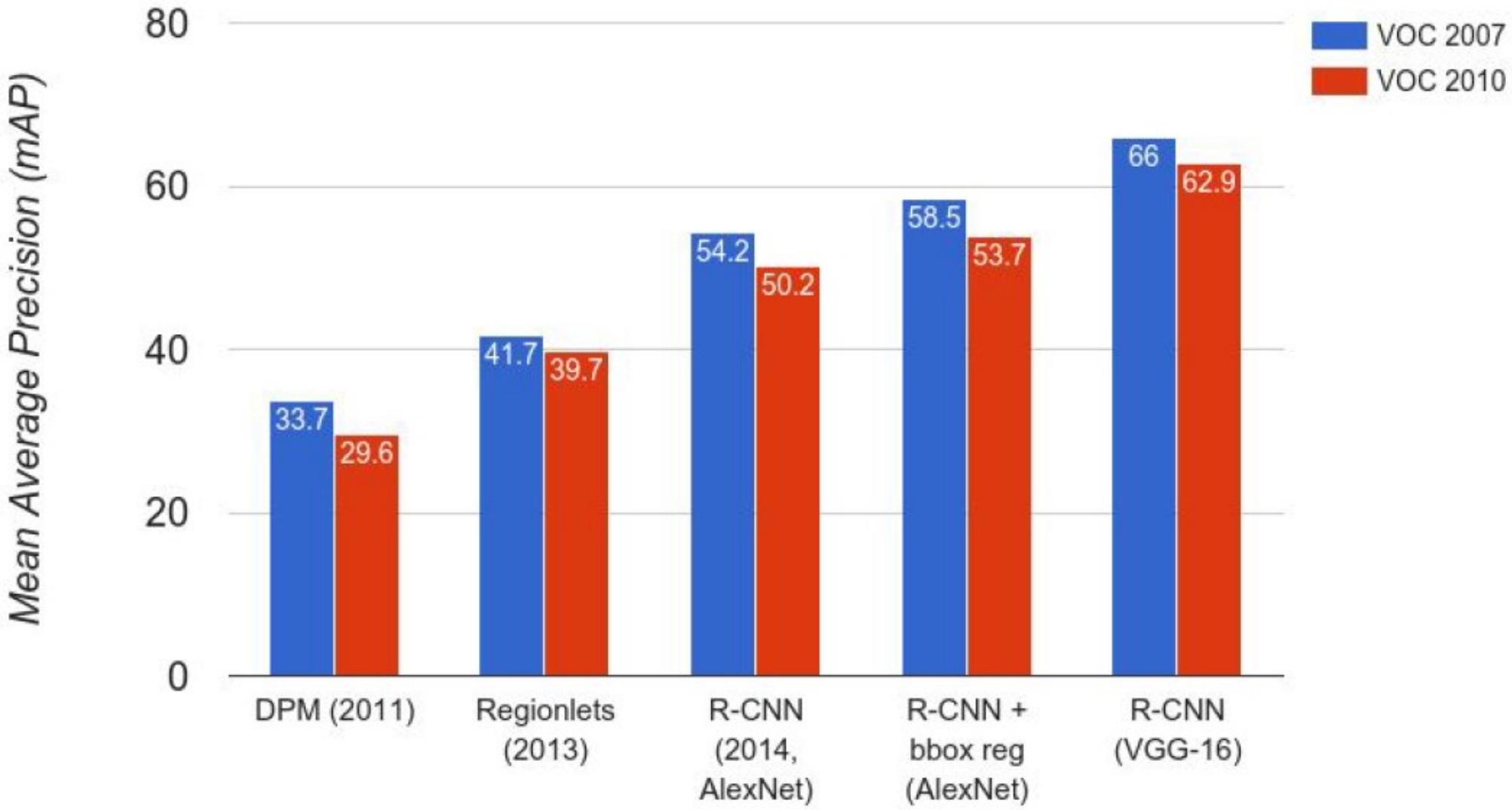
$$t_x = (G_x - P_x)/P_w \quad (6)$$

$$t_y = (G_y - P_y)/P_h \quad (7)$$

$$t_w = \log(G_w/P_w) \quad (8)$$

$$t_h = \log(G_h/P_h). \quad (9)$$

Results


Mean Average Precision (mAP)
80
60
40
20
0
VOC 2007
VOC 2010
33.7
29.6
41.7
39.7
54.2
50.2
58.5
53.7
66
62.9
DPM (2011)
Regionlets (2013)
R-CNN (2014, AlexNet)
R-CNN + bbox reg (AlexNet)
R-CNN (VGG-16)

Additional Experiment

- Performance layer by layer

VOC 2007 test	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv	mAP
R-CNN $pool_5$	51.8	60.2	36.4	27.8	23.2	52.8	60.6	49.2	18.3	47.8	44.3	40.8	56.6	58.7	42.4	23.4	46.1	36.7	51.3	55.7	44.2
R-CNN fc_6	59.3	61.8	43.1	34.0	25.1	53.1	60.6	52.8	21.7	47.8	42.7	47.8	52.5	58.5	44.6	25.6	48.3	34.0	53.1	58.0	46.2
R-CNN fc_7	57.6	57.9	38.5	31.8	23.7	51.2	58.9	51.4	20.0	50.5	40.9	46.0	51.6	55.9	43.3	23.3	48.1	35.3	51.0	57.4	44.7
R-CNN FT $pool_5$	58.2	63.3	37.9	27.6	26.1	54.1	66.9	51.4	26.7	55.5	43.4	43.1	57.7	59.0	45.8	28.1	50.8	40.6	53.1	56.4	47.3
R-CNN FT fc_6	63.5	66.0	47.9	37.7	29.9	62.5	70.2	60.2	32.0	57.9	47.0	53.5	60.1	64.2	52.2	31.3	55.0	50.0	57.7	63.0	53.1
R-CNN FT fc_7	64.2	69.7	50.0	41.9	32.0	62.6	71.0	60.7	32.7	58.5	46.5	56.1	60.6	66.8	54.2	31.5	52.8	48.9	57.9	64.7	54.2
R-CNN FT fc_7 BB	**68.1**	**72.8**	**56.8**	**43.0**	**36.8**	**66.3**	**74.2**	**67.6**	**34.4**	**63.5**	**54.5**	**61.2**	**69.1**	**68.6**	**58.7**	**33.4**	**62.9**	**51.1**	**62.5**	**64.8**	**58.5**
DPM v5 [20]	33.2	60.3	10.2	16.1	27.3	54.3	58.2	23.0	20.0	24.1	26.7	12.7	58.1	48.2	43.2	12.0	21.1	36.1	46.0	43.5	33.7
DPM ST [28]	23.8	58.2	10.5	8.5	27.1	50.4	52.0	7.3	19.2	22.8	18.1	8.0	55.9	44.8	32.4	13.3	15.9	22.8	46.2	44.9	29.1
DPM HSC [31]	32.2	58.3	11.5	16.3	30.6	49.9	54.8	23.5	21.5	27.7	34.0	13.7	58.1	51.6	39.9	12.4	23.5	34.4	47.4	45.2	34.3

fc7 layer : 29%, 16.8M parameter
conv1~pool5 layer : 6%

More Details…

- Which one is the best?

Figure 7: Different object proposal transformations. (A) the original object proposal at its actual scale relative to the transformed CNN inputs; (B) tightest square with context; (C) tightest square without context; (D) warp. Within each column and example proposal, the top row corresponds to $p = 0$ pixels of context padding while the bottom row has $p = 16$ pixels of context padding.

Problems of R-CNN

- Slow at test-time: need to run full forward path of CNN for each region proposal
  - 13s/image on a GPU(K40)
  - 53s/image on a CPU
- SVM and regressors are post-hoc: CNN features not updated in response to SVMs and regressors
- Complex multistage training pipeline (84 hours using K40 GPU)
  - Fine-tune network with softmax classifier(log loss)
  - Train post-hoc linear SVMs(hinge loss)
  - Train post-hoc bounding-box regressions(squared loss)

SPP-Net

Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

SPP-Net's Idea

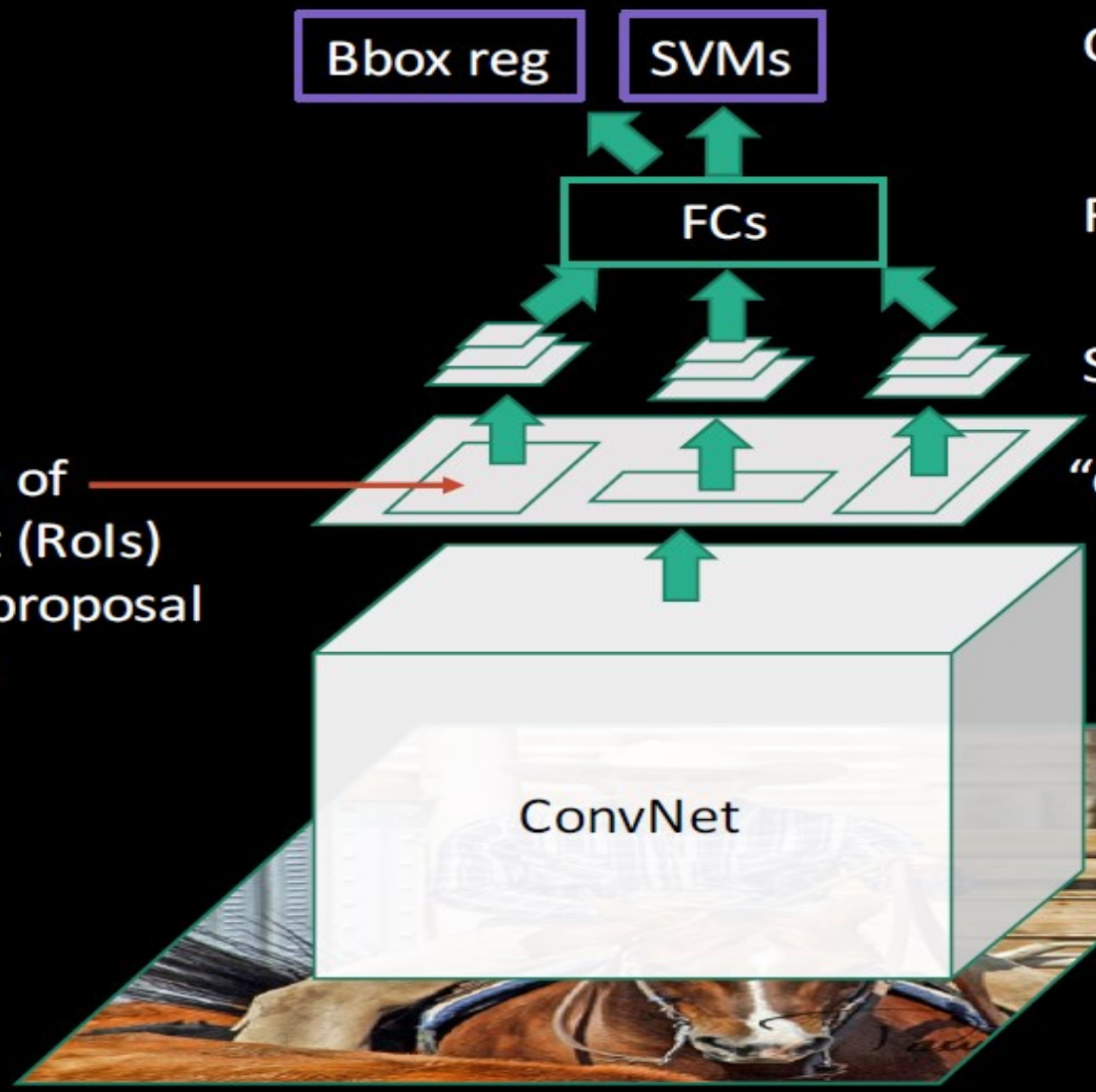
- R-CNN is time-consuming
  - It repeatedly applies the deep CNN to the raw pixels of thousands of warped regions per image

  →Performing CNN ONCE!

- R-CNN requires a fixed input size
  - FC layers need to have fixed size/length input by their definition

  → Using a spatial pyramid pooling layer to remove the fixed-size constraint of the network

SPP-net



Apply bounding-box regressors

Classify regions with SVMs

Fully-connected layers

Spatial Pyramid Pooling (SPP) layer

"conv5" feature map of image

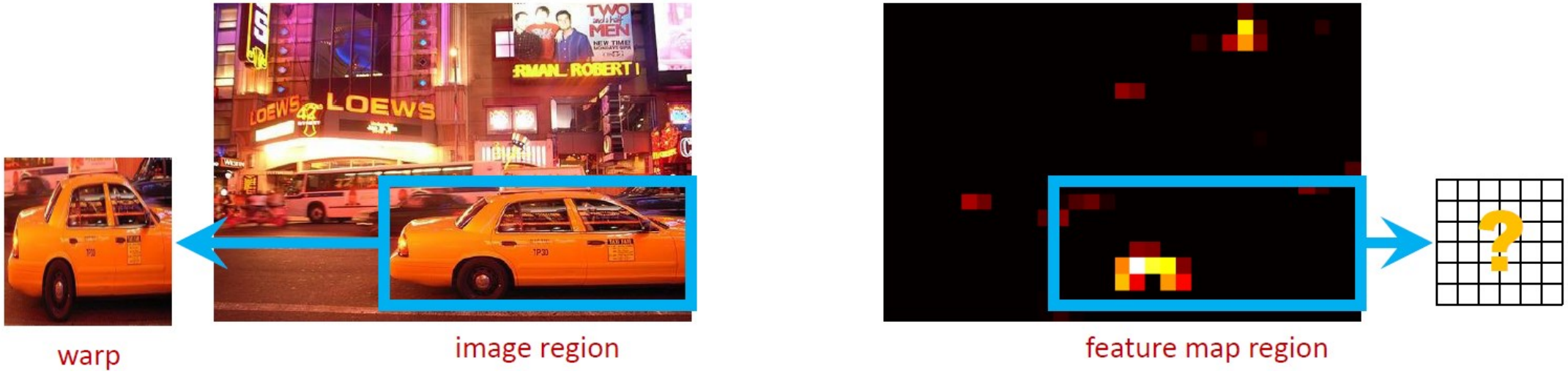
Forward *whole* image through ConvNet

Input image

He et al. ECCV14.

Post hoc component

Regions on Feature Maps



- Compute convolutional feature maps on the entire image only once
- Project an image region to a feature map region
- Extract a region-based feature from the feature map region…
- Fixed-length features are required by fully-connected layers or SVM
- But how to produce a fixed-length feature from a feature map region?

Spatial Pyramid Pooling(SPP) Layer

From Kaiming's slides

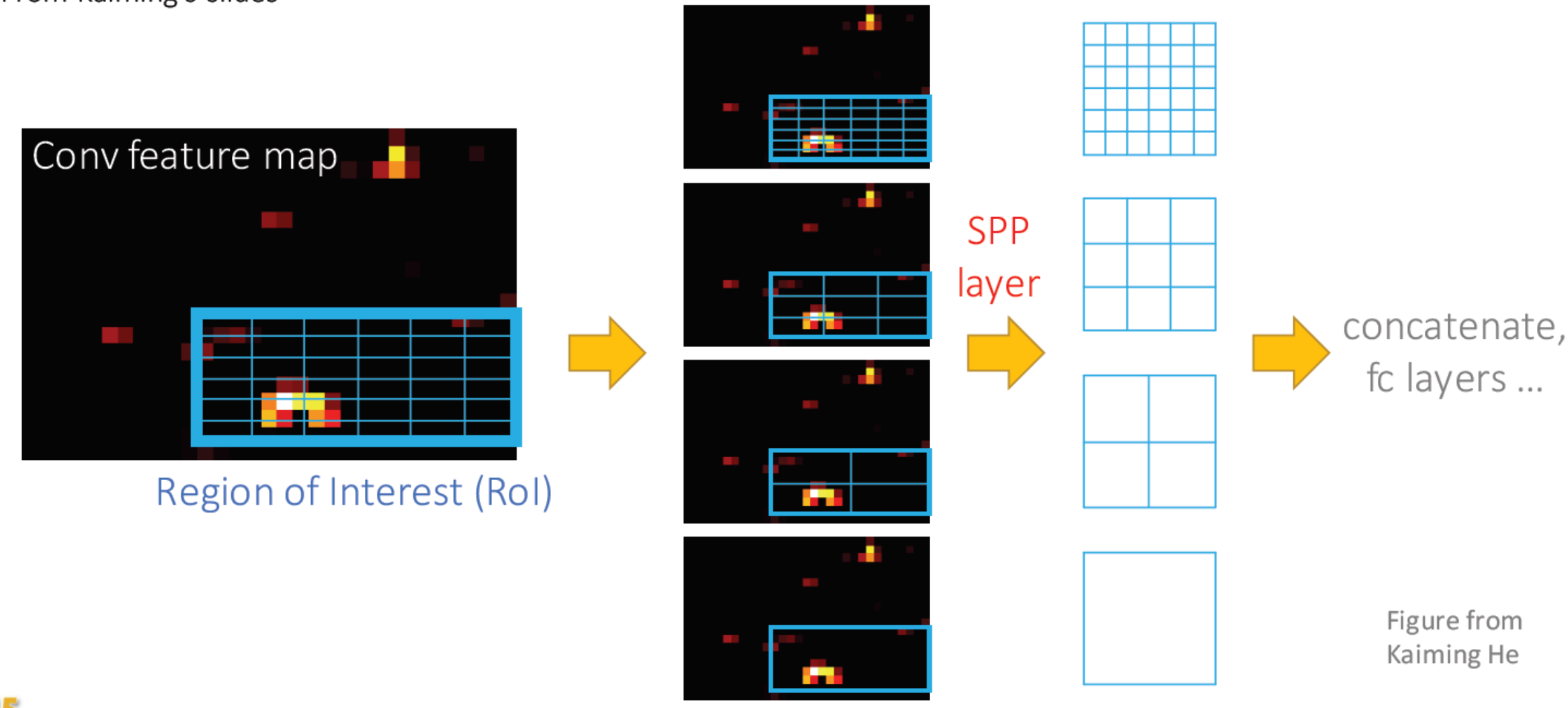
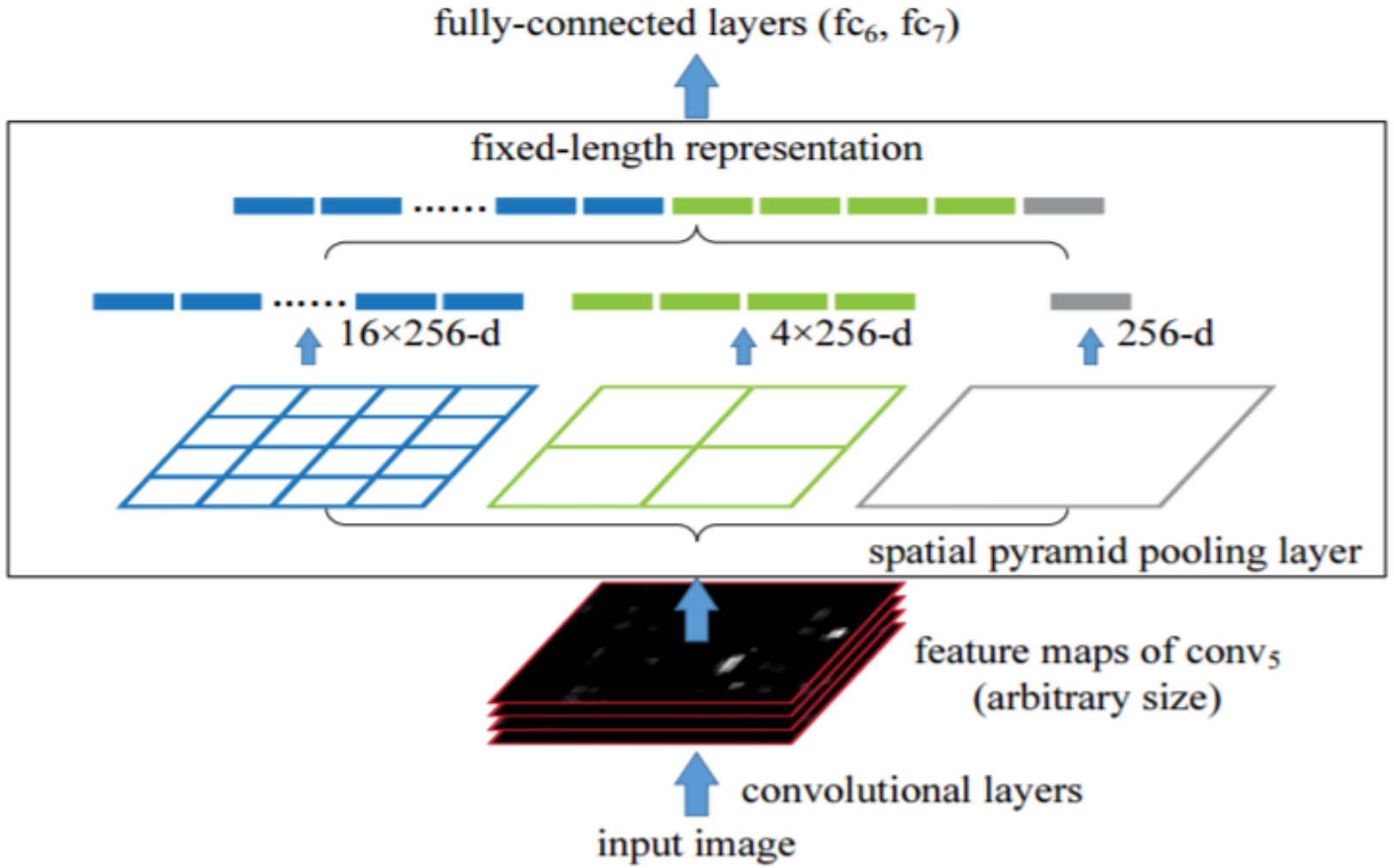


Figure from
Kaiming He

Spatial Pyramid Pooling(SPP) Layer



Review of R-CNN Training

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

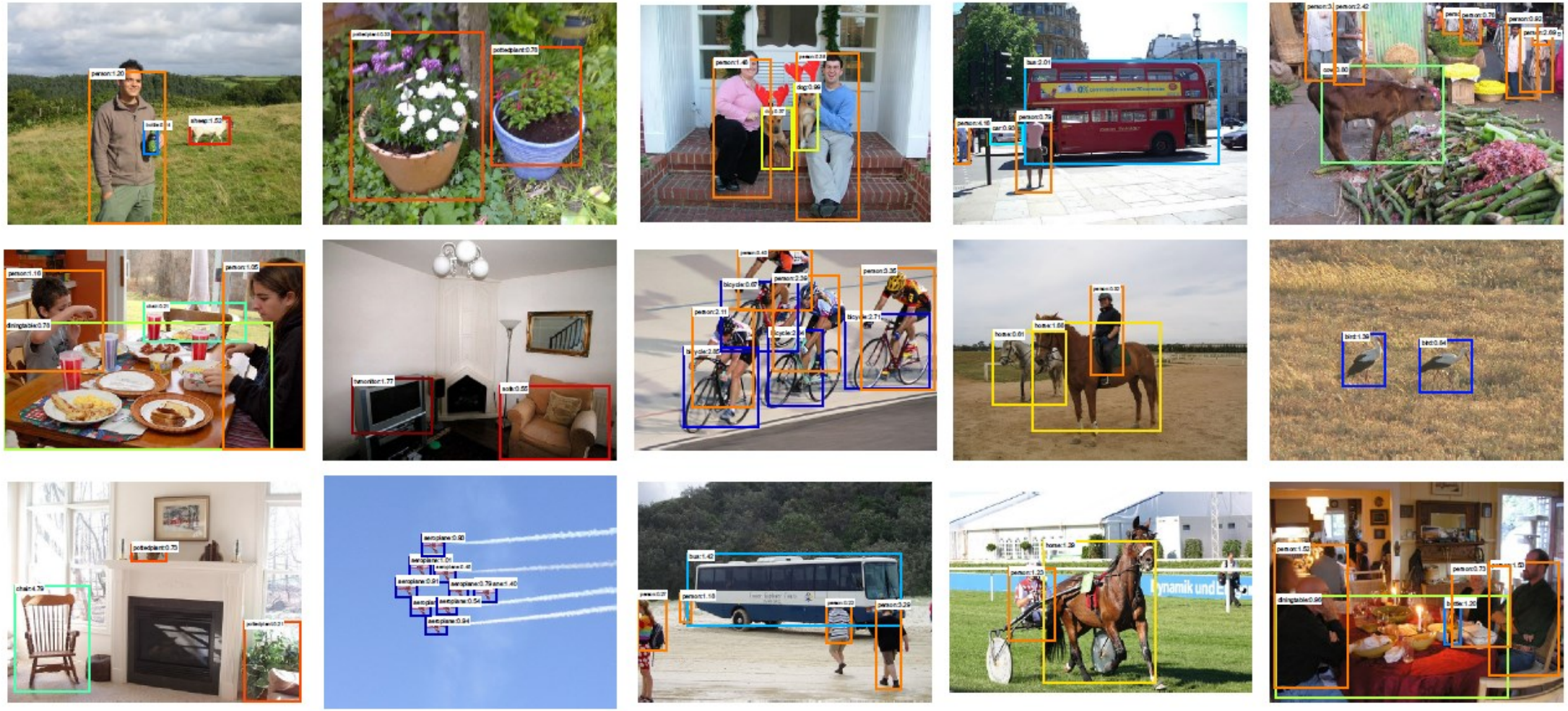
1. [offline] $M \leftarrow$ Pre-train a ConvNet for ImageNet classification
2. $M' \leftarrow$ Fine-tune M for object detection (softmax classifier + log loss)
3. $F \leftarrow$ Cache feature vectors to disk using M'
4. Train post hoc linear SVMs on F (hinge loss)
5. Train post hoc linear bounding-box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

Training of SPP-Net

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)

Results



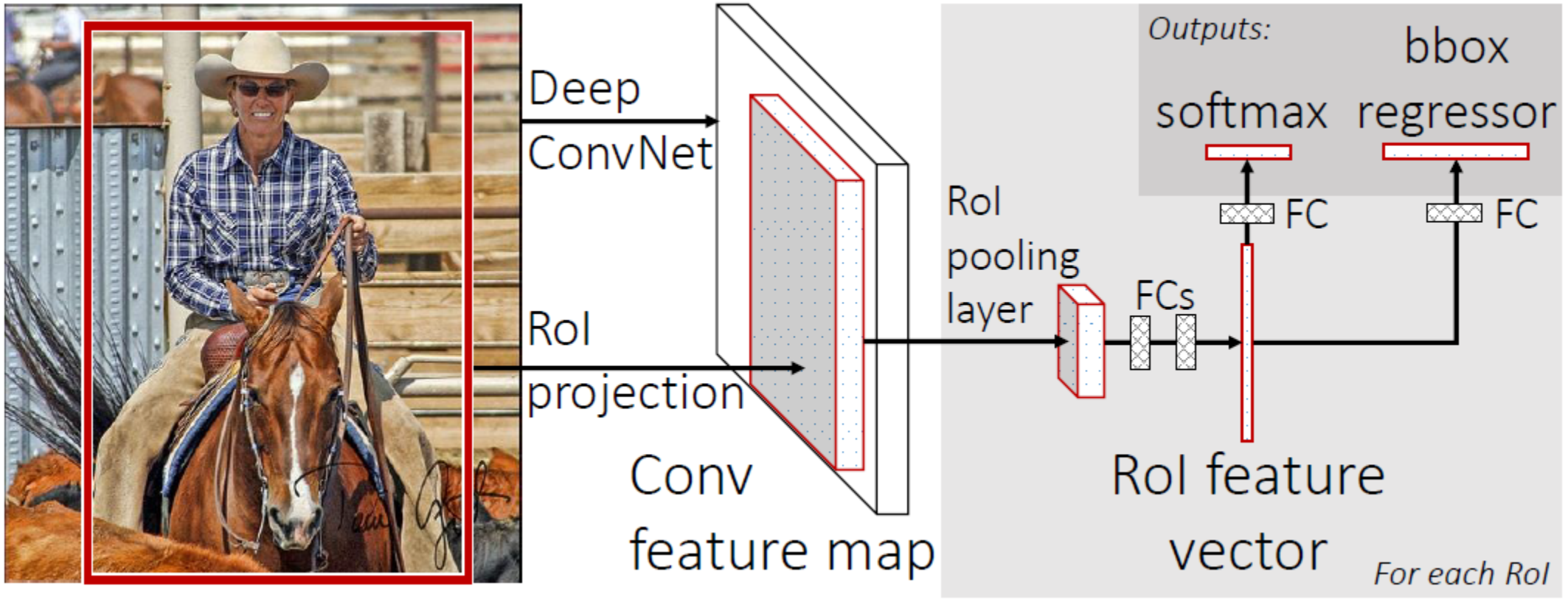
Problems

- CNN features not updated in response to SVMs
- Complex multistage training pipeline (25.5 hours using K40 GPU)

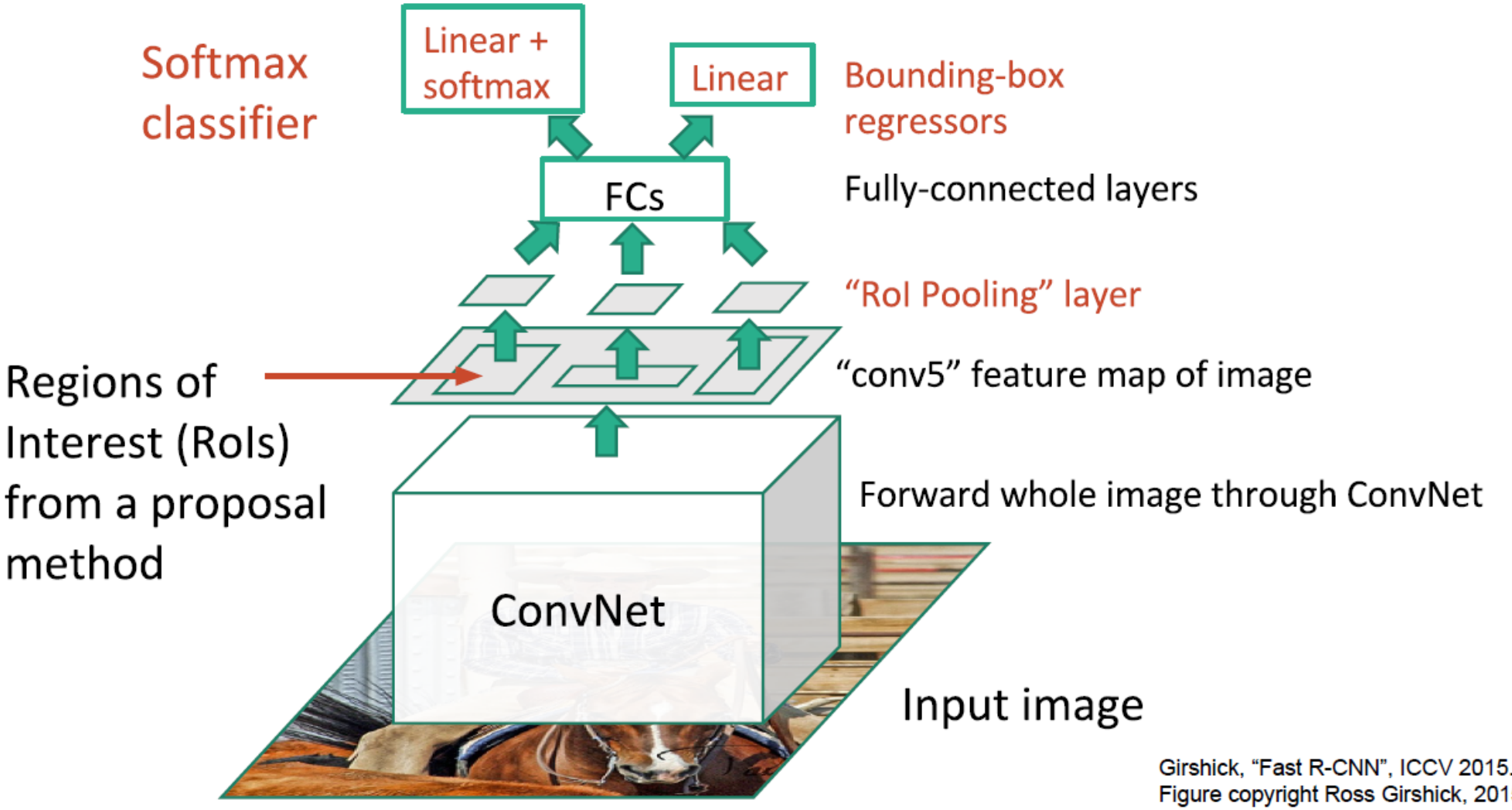
Fast R-CNN

- Fix most of what's wrong with 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

Fast R-CNN Architecture



Fast R-CNN

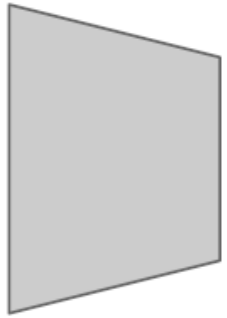


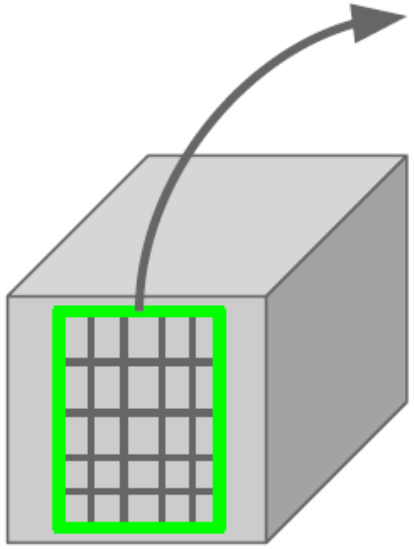
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015;

RoI Pooling

Hi-res input image:
3 x 800 x 600
with region
proposal

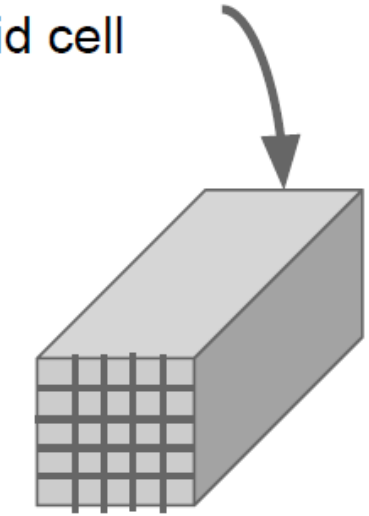
Convolution
and Pooling





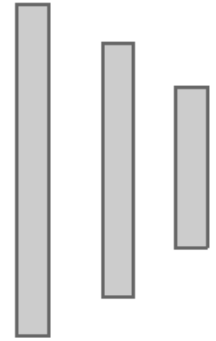
Hi-res conv features:
C x H x W
with region proposal

Max-pool within
each grid cell



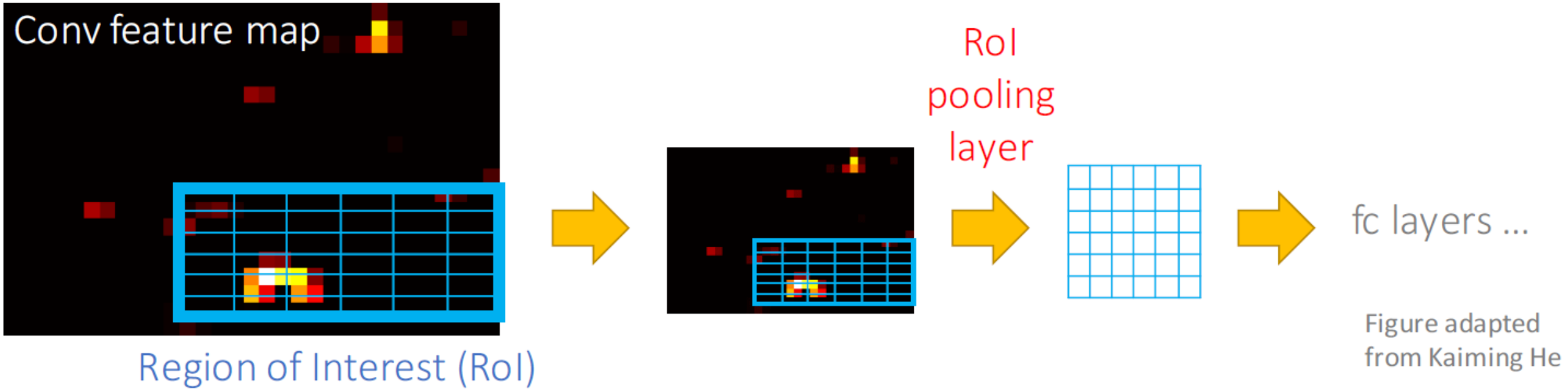
RoI conv features:
C x h x w
for region proposal

Fully-connected
layers



Fully-connected layers expect
low-res conv features:
C x h x w

RoI Pooling



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

RoI in Conv feature map : 21x14 → 3x2 max pooling with stride(3, 2) → output : 7x7
RoI in Conv feature map : 35x42 → 5x6 max pooling with stride(5, 6) → output : 7x7

Training & Testing

1. Takes an input and a set of bounding boxes
2. Generate convolutional feature maps
3. For each bbox, get a fixed-length feature vector from RoI pooling layer
4. Outputs have two information
   - K+1 class labels
   - Bounding box locations

- Loss function

True box coordinates

Predicted box coordinates

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1]L_{\text{loc}}(t^u, v)$$

Predicted class scores

True class scores

Log loss

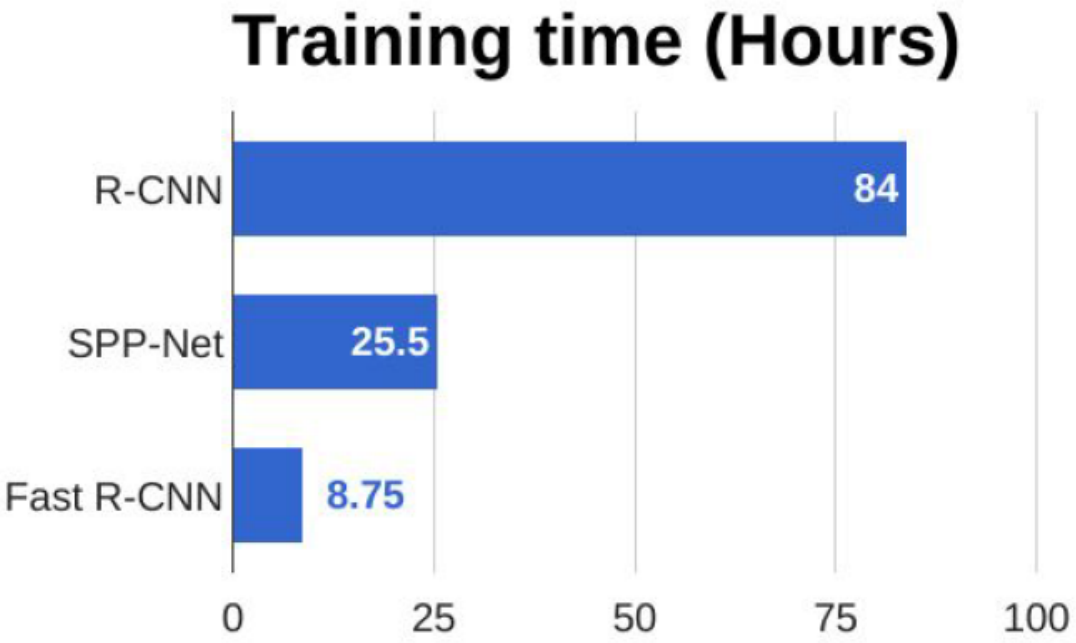
Smooth L1 loss

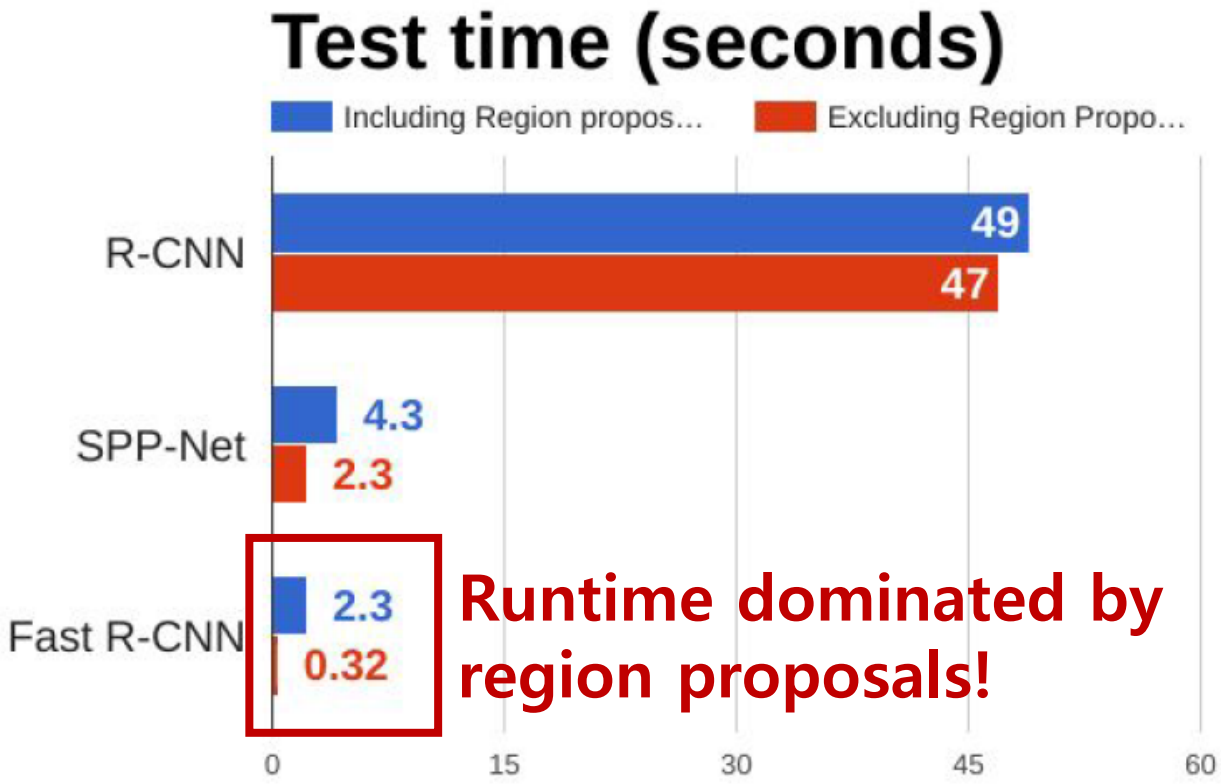
$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

R-CNN vs SPP-net vs Fast R-CNN





Experimental Findings

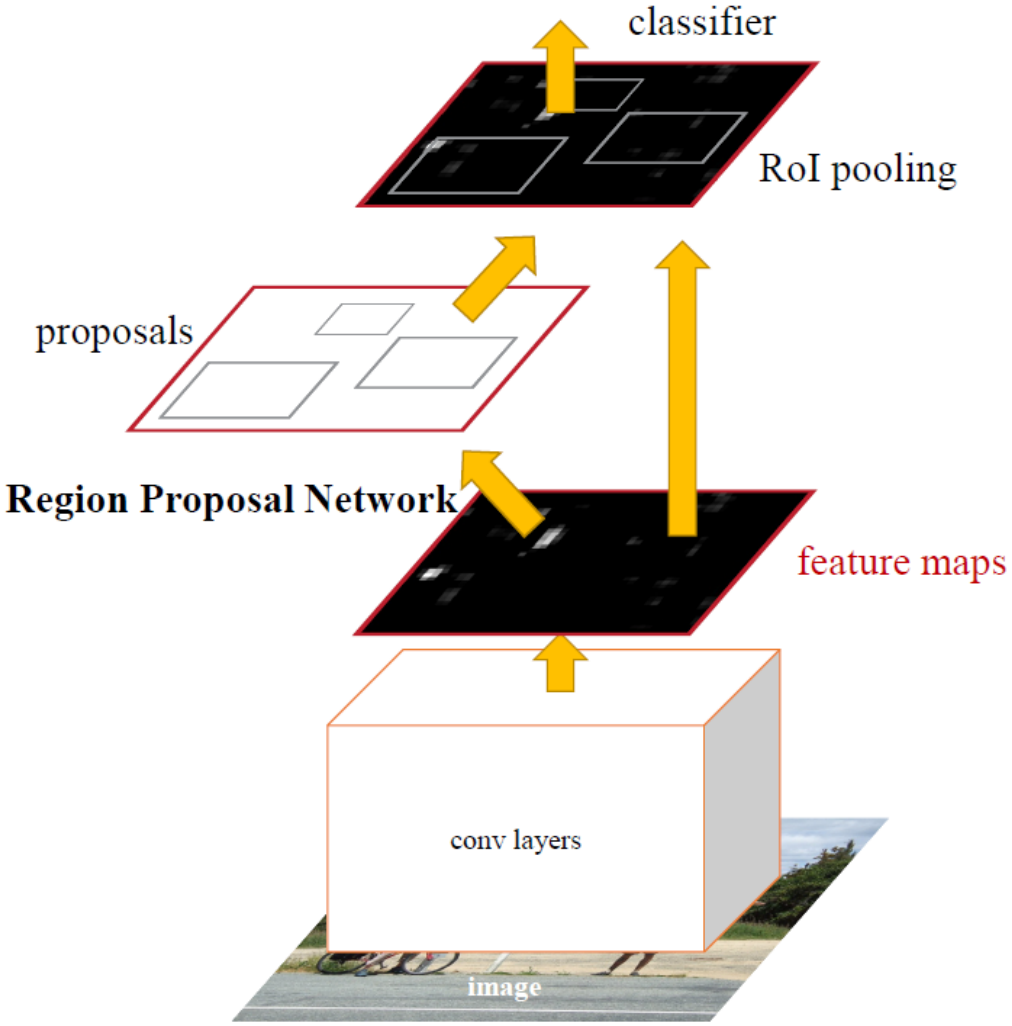
- End-to-end training is important for very deep networks
- Softmax is a fine replacement for SVMs
- Multi-task training is beneficial
- Single-scale testing is a good tradeoff
- Fast training and testing enables new experiments

Problems of Fast R-CNN

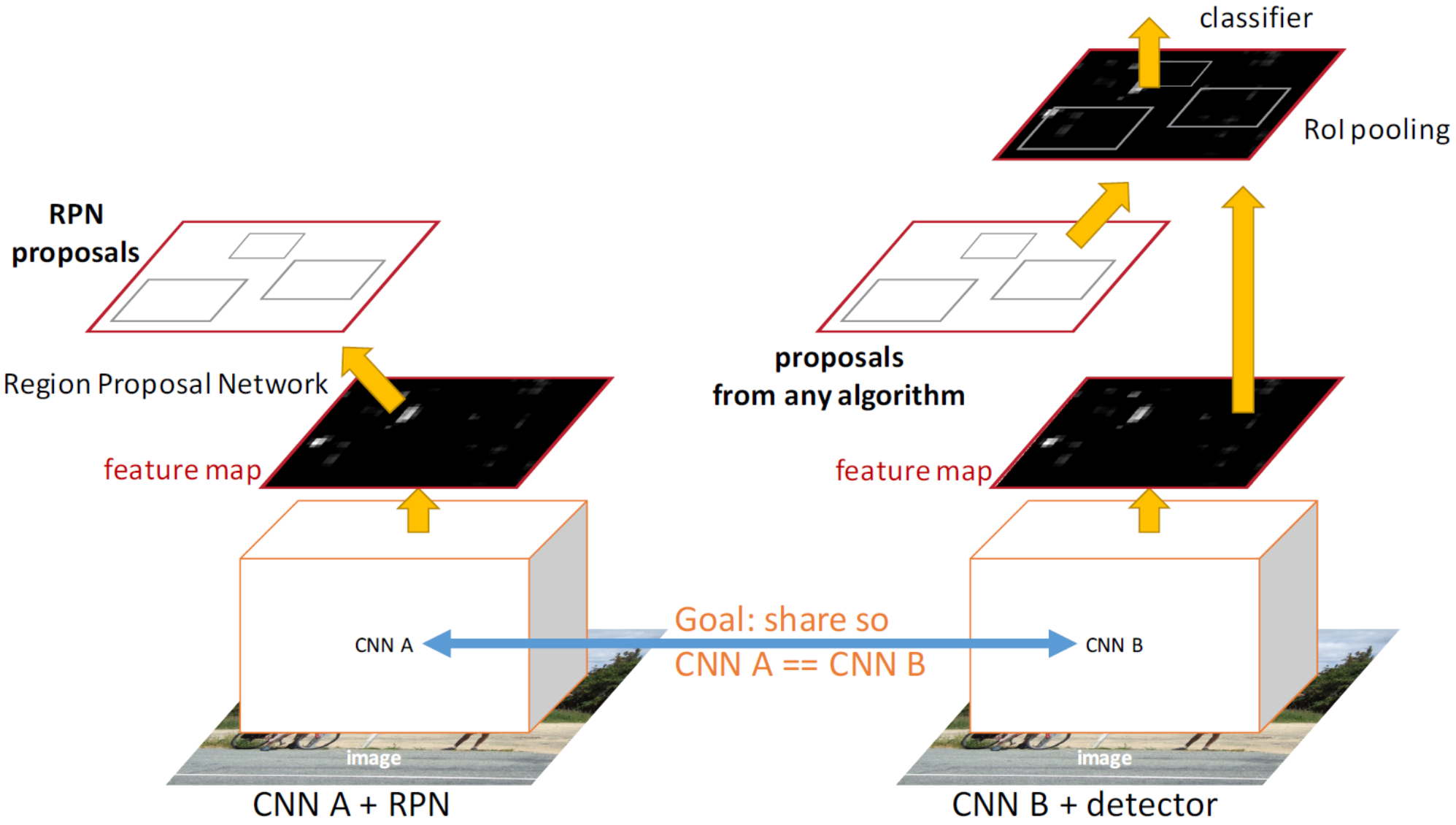
- Out-of-network region proposals are the test-time computational bottleneck
- Is it fast enough??

Faster R-CNN(RPN + Fast R-CNN)

- Insert a Region Proposal Network (RPN) after the last convolutional layer → using GPU!
- RPN trained to produce region proposals directly; no need for external region proposals
- After RPN, use RoI Pooling and an upstream classifier and bbox regressor just like Fast R-CNN

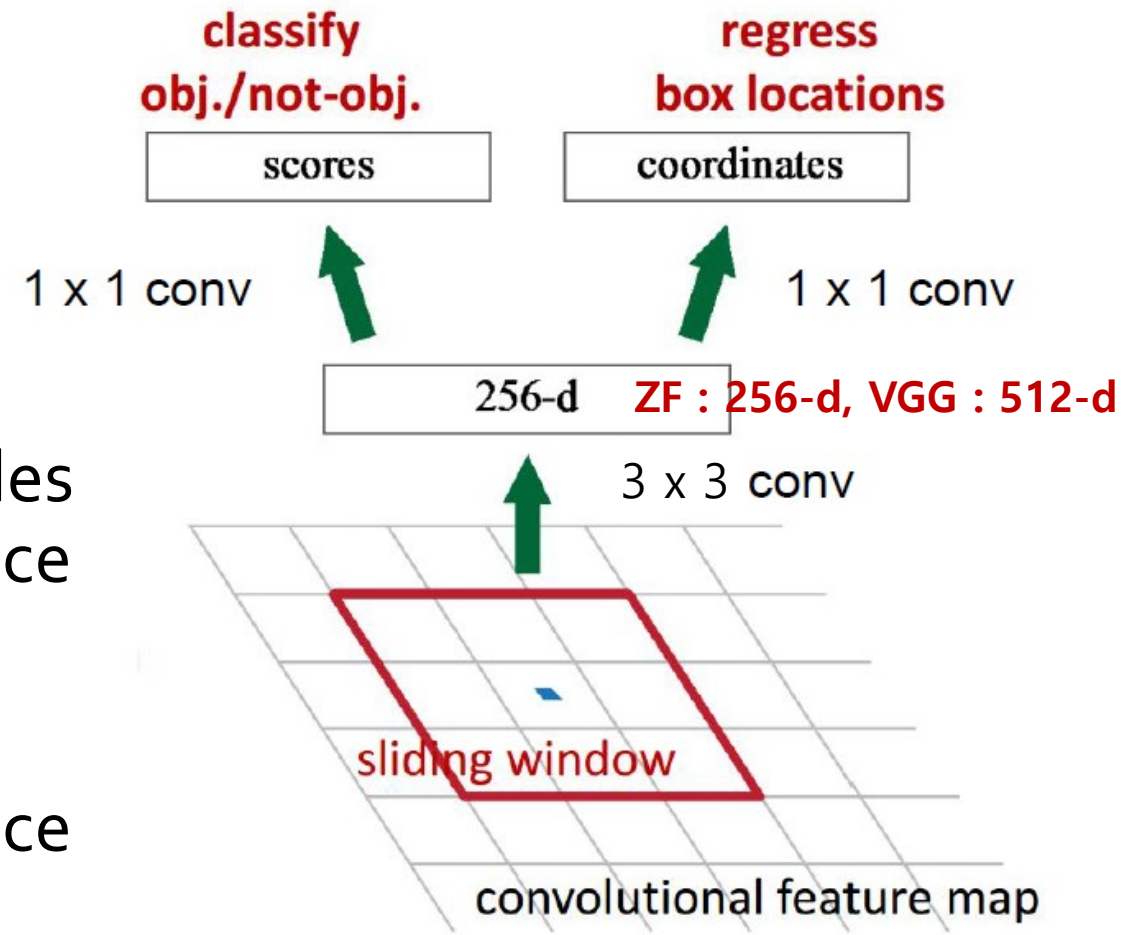


Training Goal : Share Features



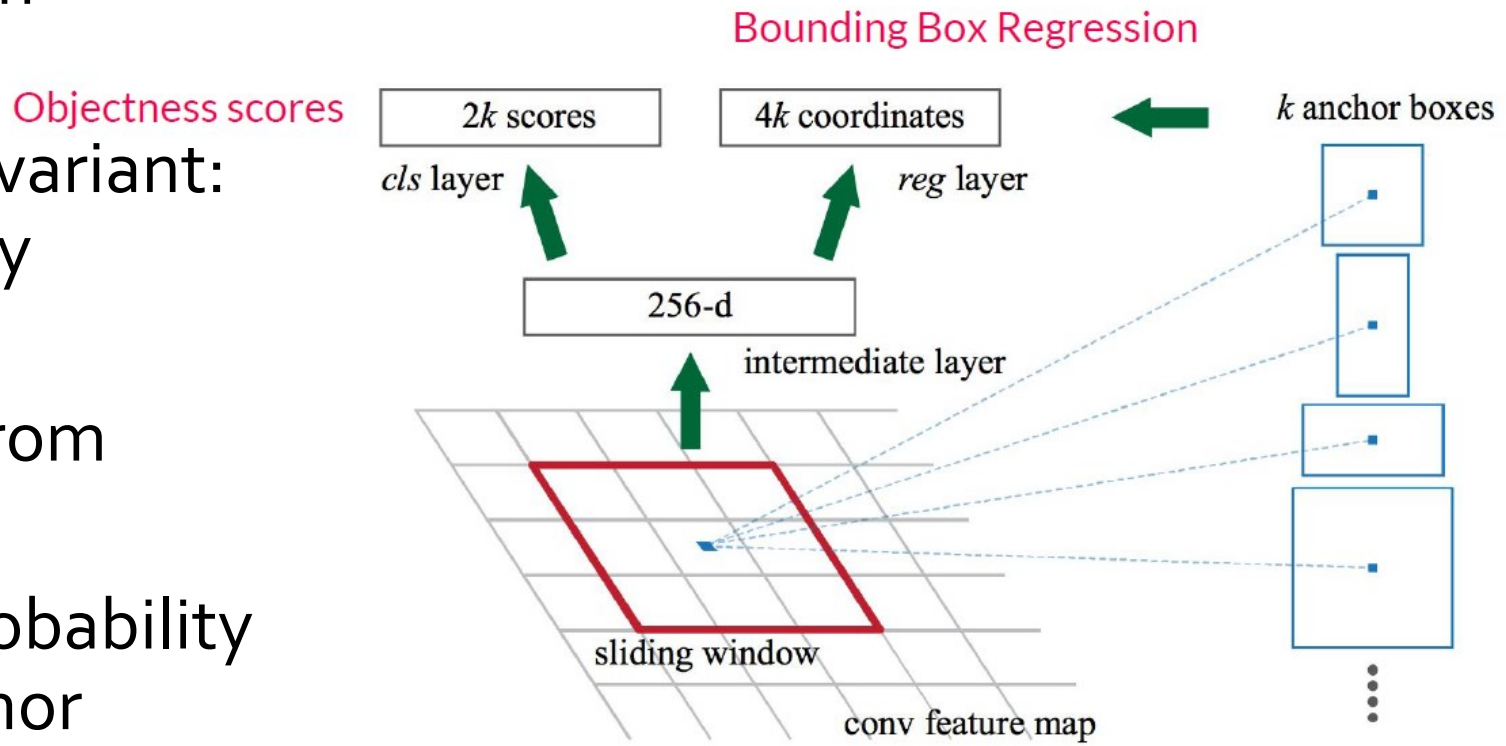
RPN

- Slide a small window on the feature map
- Build a small network for
  - Classifying object or not-object
  - Regressing bbox locations
- Position of the sliding window provides localization information with reference to the image
- Box regression provides finer localization information with reference to this sliding window



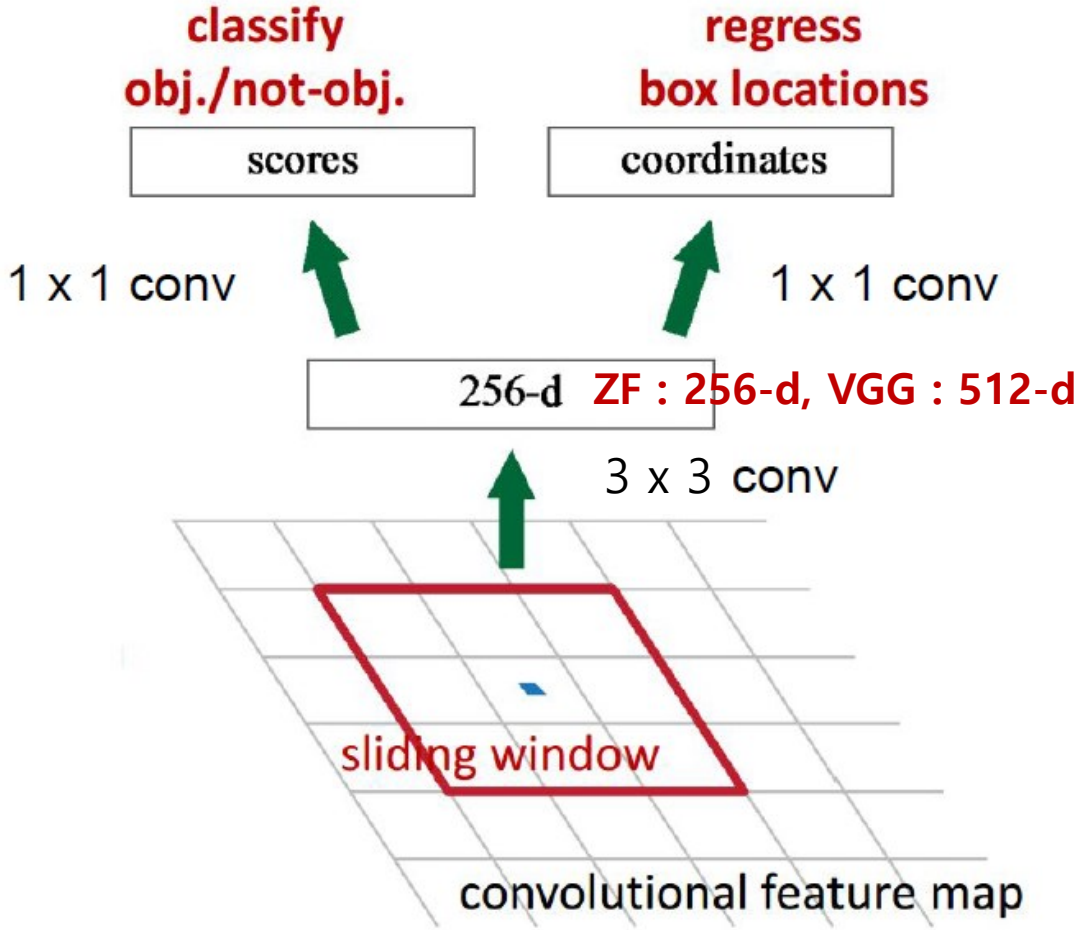
RPN

- Use k anchor boxes at each location
- Anchors are translation invariant: use the same ones at every location
- Regression gives offsets from anchor boxes
- Classification gives the probability that each (regressed) anchor shows an object



RPN(Fully Convolutional Network)

- Intermediate Layer – 256(or 512) 3x3 filter, stride 1, padding 1
- Cls layer – 18(9x2) 1x1 filter, stride 1, padding 0
- Reg layer – 36(9x4) 1x1 filter, stride 1, padding 0



Anchors as references

- Anchors: pre-defined reference boxes
- Multi-scale/size anchors:
  - Multiple anchors are used at each position:
    - 3 scale(128x128, 256x256, 512x512) and 3 aspect rations(2:1, 1:1, 1:2) yield 9 anchors
  - Each anchor has its own prediction function
  - Single-scale features, multi-scale predictions

Positive/Negative Samples

- An anchor is labeled as positive if
  - The anchor is the one with highest IoU overlap with a ground-truth box
  - The anchor has an IoU overlap with a ground-truth box higher than 0.7
- Negative labels are assigned to anchors with IoU lower than 0.3 for all ground-truth boxes
- 50%/50% ratio of positive/negative anchors in a minibatch

RPN Loss Function

i = anchor index in minibatch

Log loss

Ground truth objectness label

Smooth L1 loss

$$L(\{p_i\}, \{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*).$$

True box coordinates

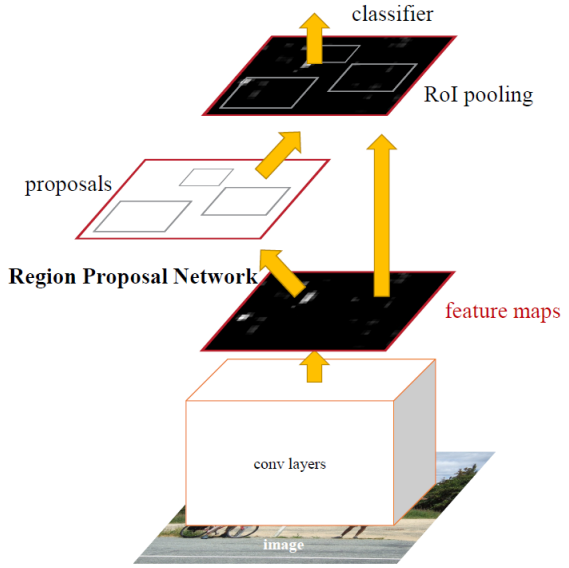
Coordinates of the predicted bounding box for anchor *i*

Predicted probability of being an object for anchor *i*

In practice λ= 10, so that both terms are roughly equally balanced

N_{cls} = Number of anchors in minibatch (~ 256)
N_{reg} = Number of anchor locations (~ 2400)

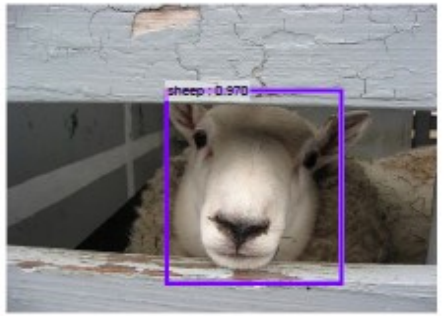
4-Step Alternating Training

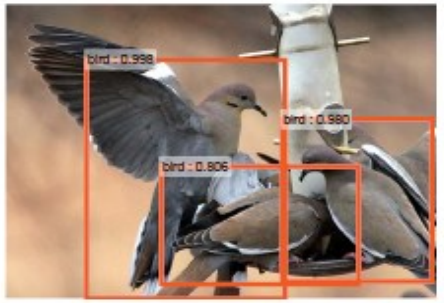


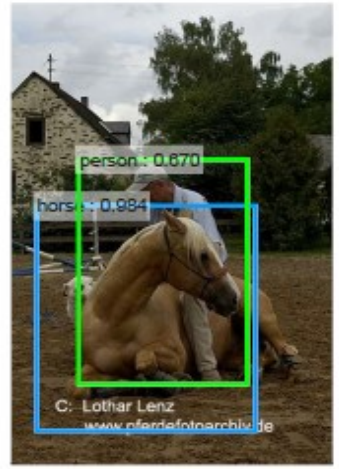

```
# Let M0 be an ImageNet pre-trained network

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
```

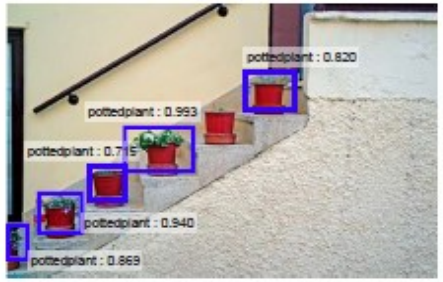
Results


aeroplane : 0.992
aeroplane : 0.986


sheep : 0.970


bird : 0.998
bird : 0.980
bird : 0.806


person : 0.670
C: Lothar Lenz


pottedplant : 0.820
pottedplant : 0.993
pottedplant : 0.940
pottedplant : 0.869

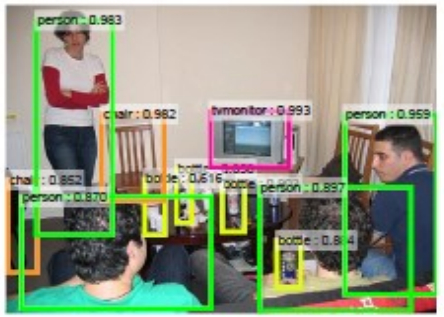

aeroplane : 0.998
person : 0.993
person : 0.987


diningtable : 0.997
chair : 0.962
chair : 0.978
chair : 0.976


bird : 0.997
bird : 0.727


tvmonitor : 0.945
tvmonitor : 0.993
person : 0.968


aeroplane : 0.978
person : 0.988


person : 0.983
chair : 0.982
tvmonitor : 0.993
person : 0.959
person : 0.897
person : 0.870

Results

	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals)	50 seconds	2 seconds	**0.2 seconds**
(Speedup)	1x	25x	**250x**
mAP (VOC 2007)	66.0	**66.9**	**69.9**

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.

model	system	conv	proposal	region-wise	total	rate
VGG	SS + Fast R-CNN	146	1510	174	1830	0.5 fps
VGG	RPN + Fast R-CNN	141	**10**	47	**198**	**5 fps**
ZF	RPN + Fast R-CNN	31	**3**	25	**59**	**17 fps**

Experiments

Table 1: the learned average proposal size for each anchor using the ZF net (numbers for $s = 600$).

anchor	128^2, 2:1	128^2, 1:1	128^2, 1:2	256^2, 2:1	256^2, 1:1	256^2, 1:2	512^2, 2:1	512^2, 1:1	512^2, 1:2
proposal	188×111	113×114	70×92	416×229	261×284	174×332	768×437	499×501	355×715

Table 8: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different settings of anchors**. The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using 3 scales and 3 aspect ratios (69.9%) is the same as that in Table 3.

settings	anchor scales	aspect ratios	mAP (%)
1 scale, 1 ratio	128^2	1:1	65.8
	256^2	1:1	66.7
1 scale, 3 ratios	128^2	{2:1, 1:1, 1:2}	68.8
	256^2	{2:1, 1:1, 1:2}	67.9
3 scales, 1 ratio	$\{128^2, 256^2, 512^2\}$	1:1	**69.8**
3 scales, 3 ratios	$\{128^2, 256^2, 512^2\}$	{2:1, 1:1, 1:2}	**69.9**

Table 9: Detection results of Faster R-CNN on PASCAL VOC 2007 test set using **different values of** λ in Equation (1). The network is VGG-16. The training data is VOC 2007 trainval. The default setting of using $\lambda = 10$ (69.9%) is the same as that in Table 3.

λ	0.1	1	10	100
mAP (%)	67.2	68.9	69.9	69.1

Experiments

Table 2: Detection results on **PASCAL VOC 2007 test set** (trained on VOC 2007 trainval). The detectors are Fast R-CNN with ZF, but using various proposal methods for training and testing.

train-time region proposals		**test-time region proposals**		
method	# boxes	method	# proposals	mAP (%)
SS	2000	SS	2000	58.7
EB	2000	EB	2000	58.6
RPN+ZF, shared	2000	RPN+ZF, shared	300	**59.9**
ablation experiments follow below				
RPN+ZF, unshared	2000	RPN+ZF, unshared	300	58.7
SS	2000	RPN+ZF	100	55.1
SS	2000	RPN+ZF	300	56.8
SS	2000	RPN+ZF	1000	56.3
SS	2000	RPN+ZF (no NMS)	6000	55.2
SS	2000	RPN+ZF (no *cls*)	100	44.6
SS	2000	RPN+ZF (no *cls*)	300	51.4
SS	2000	RPN+ZF (no *cls*)	1000	55.8
SS	2000	RPN+ZF (no *reg*)	300	52.1
SS	2000	RPN+ZF (no *reg*)	1000	51.3
SS	2000	RPN+VGG	300	59.2

Experiments

Table 3: Detection results on **PASCAL VOC 2007 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07+12": union set of VOC 2007 trainval and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: this number was reported in [2]; using the repository provided by this paper, this result is higher (68.1).

method	# proposals	data	mAP (%)
SS	2000	07	66.9†
SS	2000	07+12	70.0
RPN+VGG, unshared	300	07	68.5
RPN+VGG, shared	300	07	69.9
RPN+VGG, shared	300	07+12	**73.2**
RPN+VGG, shared	300	COCO+07+12	**78.8**

Table 4: Detection results on **PASCAL VOC 2012 test set**. The detector is Fast R-CNN and VGG-16. Training data: "07": VOC 2007 trainval, "07++12": union set of VOC 2007 trainval+test and VOC 2012 trainval. For RPN, the train-time proposals for Fast R-CNN are 2000. †: http://host.robots.ox.ac.uk:8080/anonymous/HZJTQA.html. ‡: http://host.robots.ox.ac.uk:8080/anonymous/YNPLXB.html. §: http://host.robots.ox.ac.uk:8080/anonymous/XEDH10.html.

method	# proposals	data	mAP (%)
SS	2000	12	65.7
SS	2000	07++12	68.4
RPN+VGG, shared†	300	12	67.0
RPN+VGG, shared‡	300	07++12	**70.4**
RPN+VGG, shared§	300	COCO+07++12	**75.9**

Is It Enough?

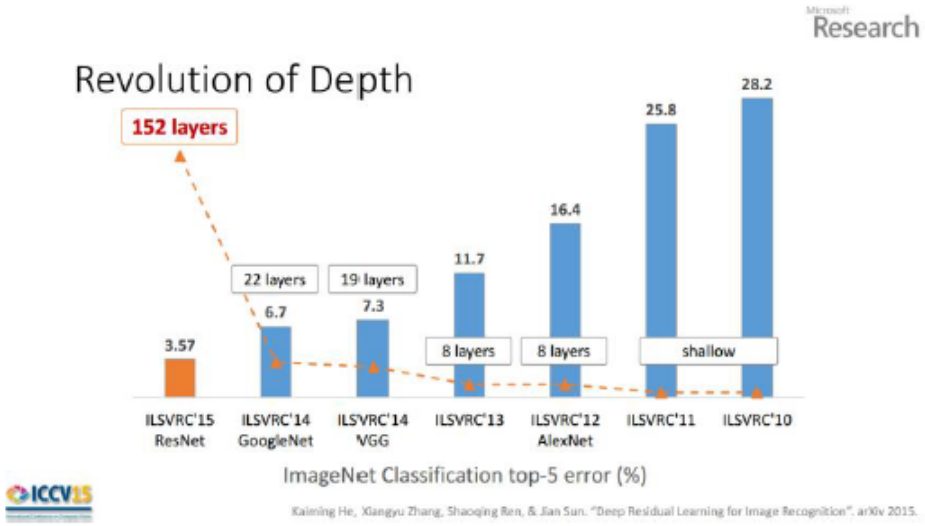
- RoI Pooling has some quantization operations
- These quantizations introduce misalignments between the RoI and the extracted features
- While this may not impact classification, it can make a negative effect on predicting bbox

PVANet – Design Principles

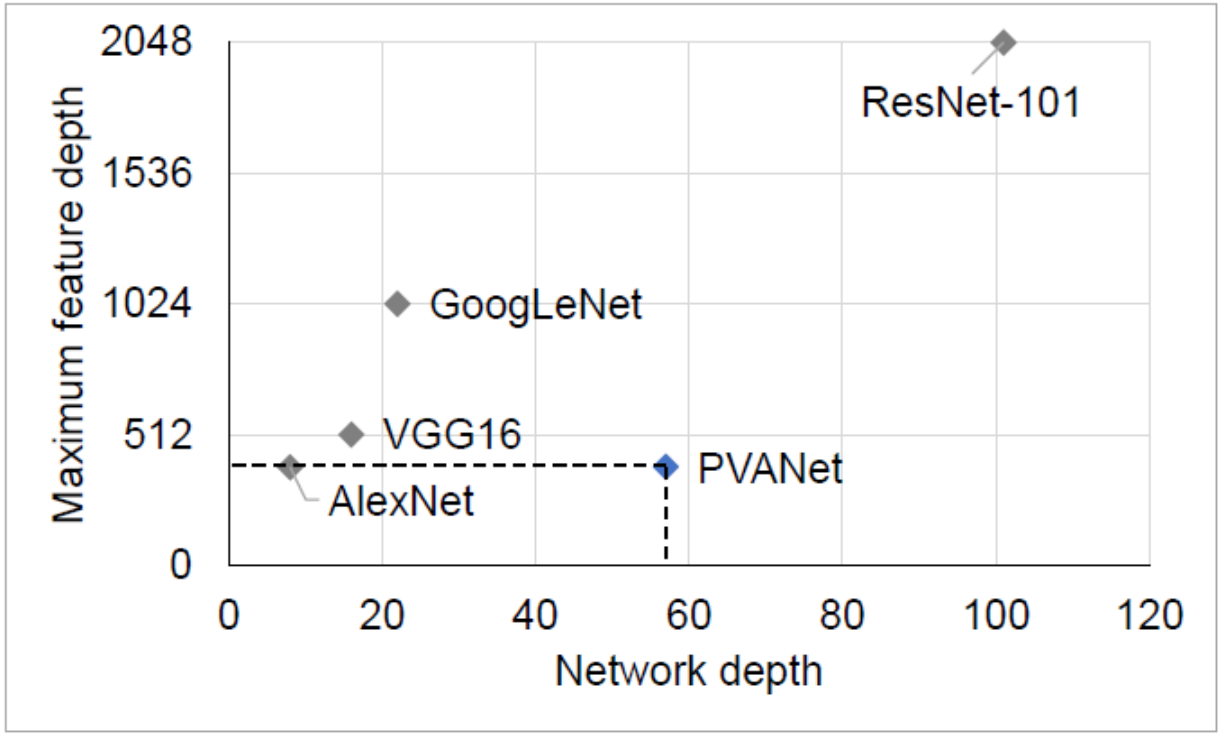
- Deep but Narrow
- Modified concatenated ReLU
- Inception
- Hyper-feature concatenation

Deep but Narrow

- Reduce redundancies from excessive convolutional outputs

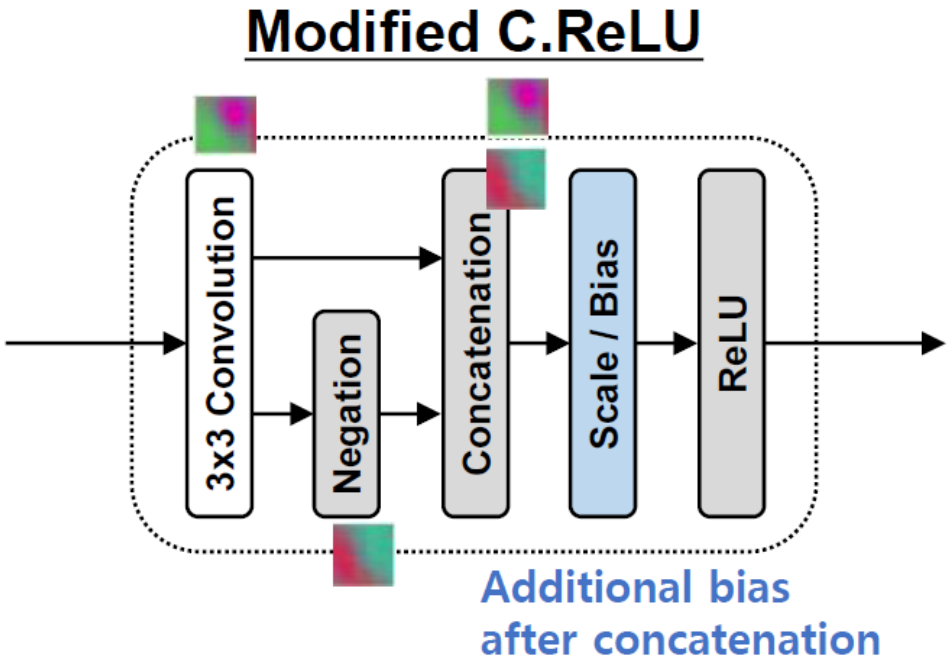


An excerpt from http://image-net.org/challenges/talks/ilsvrc2015_deep_residual_learning_kaiminghe.pdf



Modified Concatenated ReLU

- Reduce redundancies in the early convolutional layers
- Better accuracy and less training loss than the original C.ReLU

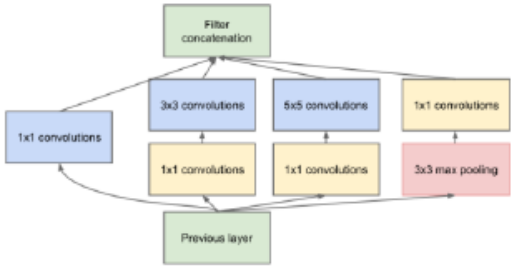


Model	Test error (%)	Accuracy drop (%)	Cost (MMACs)
ALL-CNN-C (our reproduction)	9.83	0.0	270
HALF-CNN (Halved output chns)	10.91	1.08	72
HALF-CNN-CReLU (Shang et al., 2016)	9.99	0.16	140
HALF-CNN-mCReLU (ours)	9.84	**0.01**	140

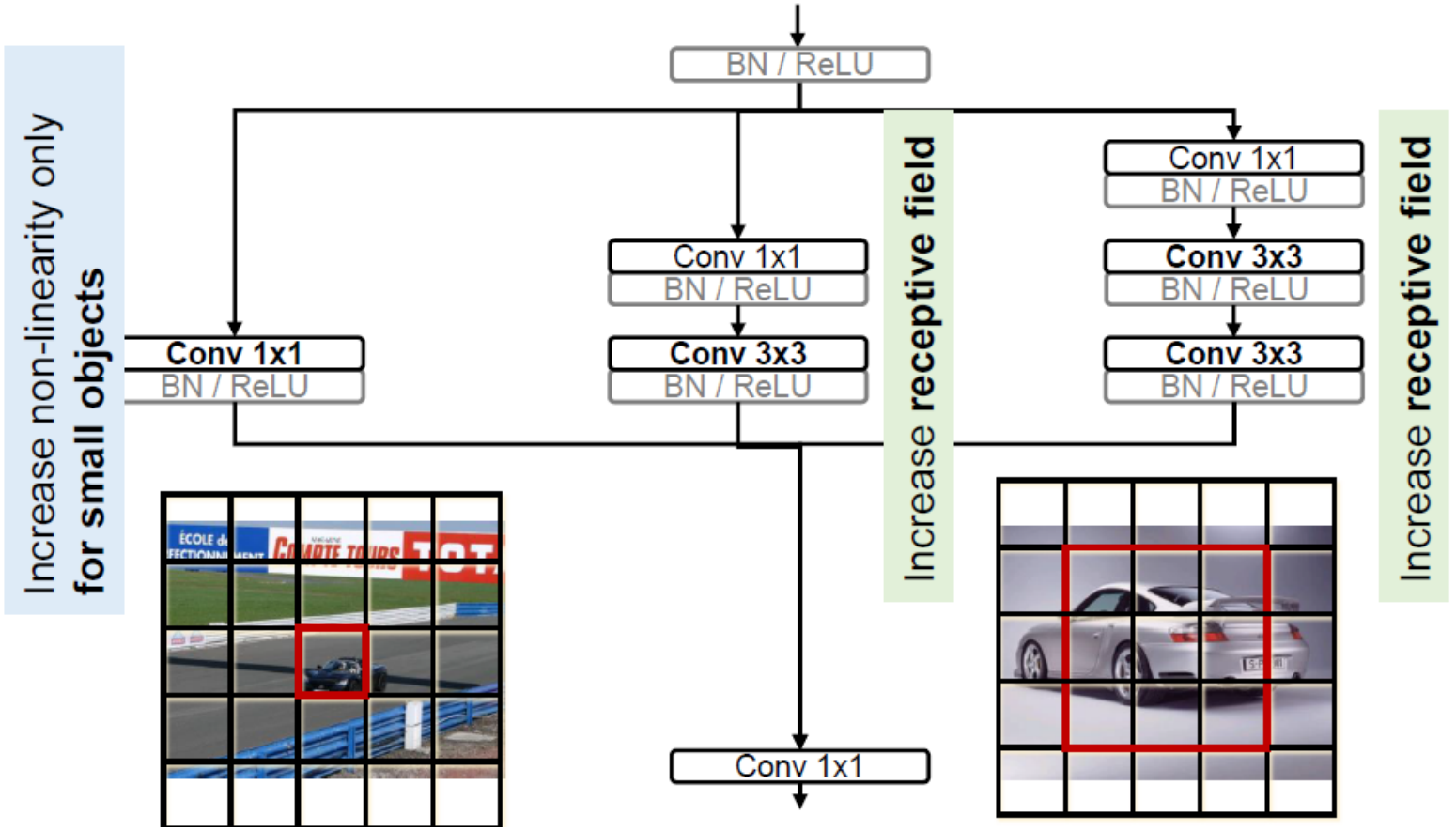
Tested on CIFAR-10

Inception

- Reduce redundancies resulted from various-sized objects

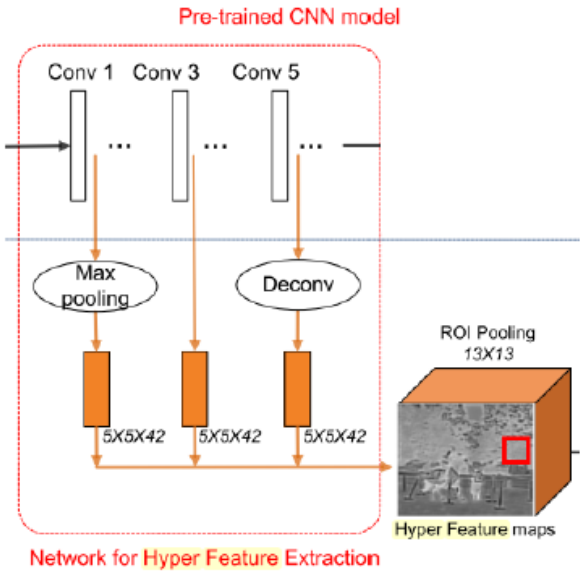


Szegedy et al. (2015)
Going deeper with convolutions

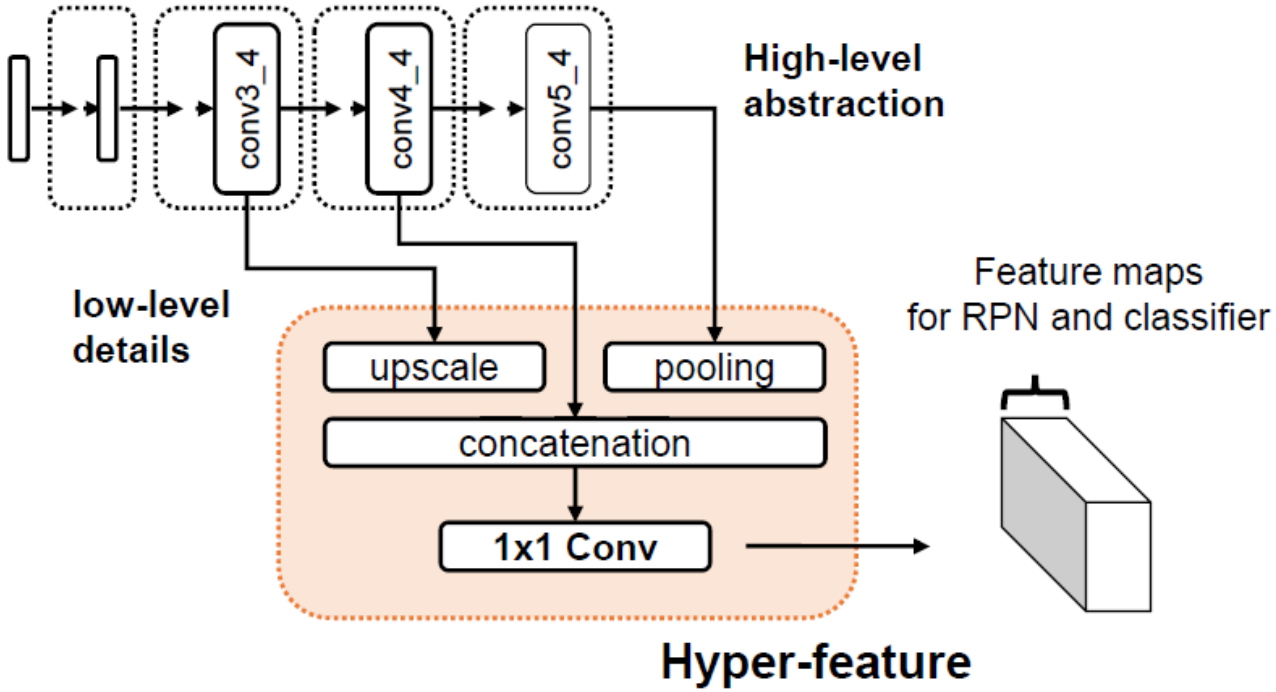


Hyper-feature Concatenation

- Low-level details bypass redundant convolutional layers
- Higher-level convolutions concentrate on contexts/abstractions

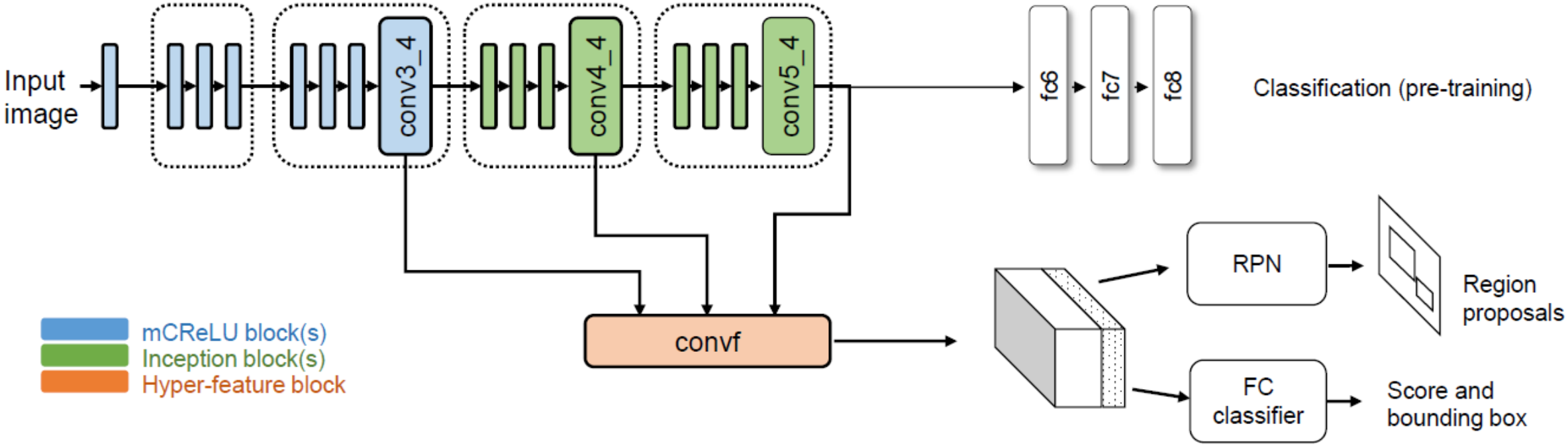


Kong et al. (2016) HyperNet: Towards Accurate Region Proposal Generation and Joint Object Detection



Overall Structure

- 54 convolutional + 3 fully connected
- Residual connections and batch normalization



Results

- VOC2012 Detection
  - Compressed model runs real-time(30 fps) on a GPU

Model	Computation cost (GMAC)				Running time		mAP
	Shared CNN	RPN	Classifier	Total	ms	×(PVANET)	(%)
PVANET+	7.9	1.4	18.5	27.8	46	1.0	84.2
PVANET+ compressed	7.9	1.4	3.2	12.5	32	0.7	83.7
Faster R-CNN + ResNet-101	80.5	N/A	125.9	>206.4	2240	48.6	83.8
Faster R-CNN + VGG-16	183.2	2.7	18.5	204.4	110	2.4	75.9
R-FCN + ResNet-101	122.9	0	0	122.9	133	2.9	82.0
SSD512 (VGG16)	86.7	0	N/A	>86.7	53	1.15	82.2