Fast R-CNN

Ross Girshick, 2015

곽대훈

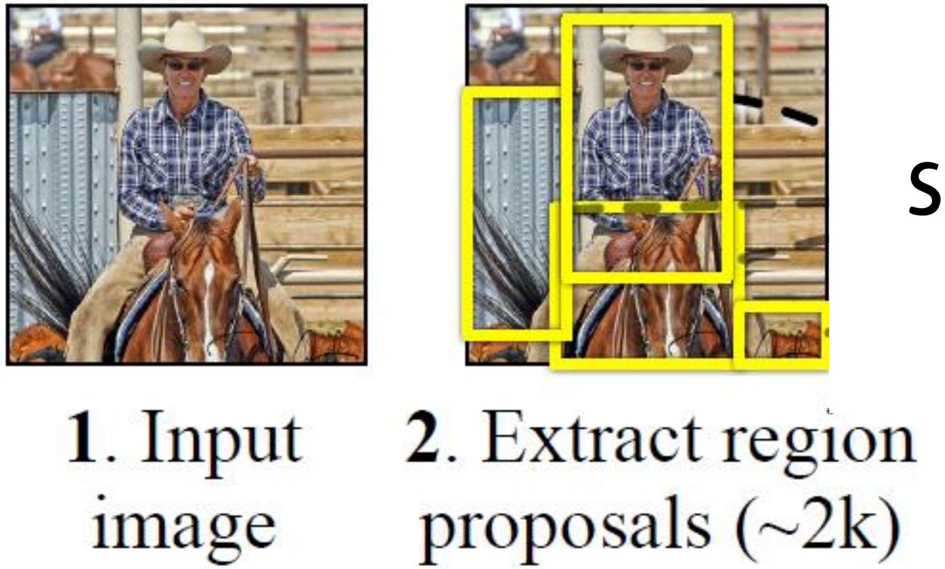
Overview

R-CNN & Motivation

R-CNN

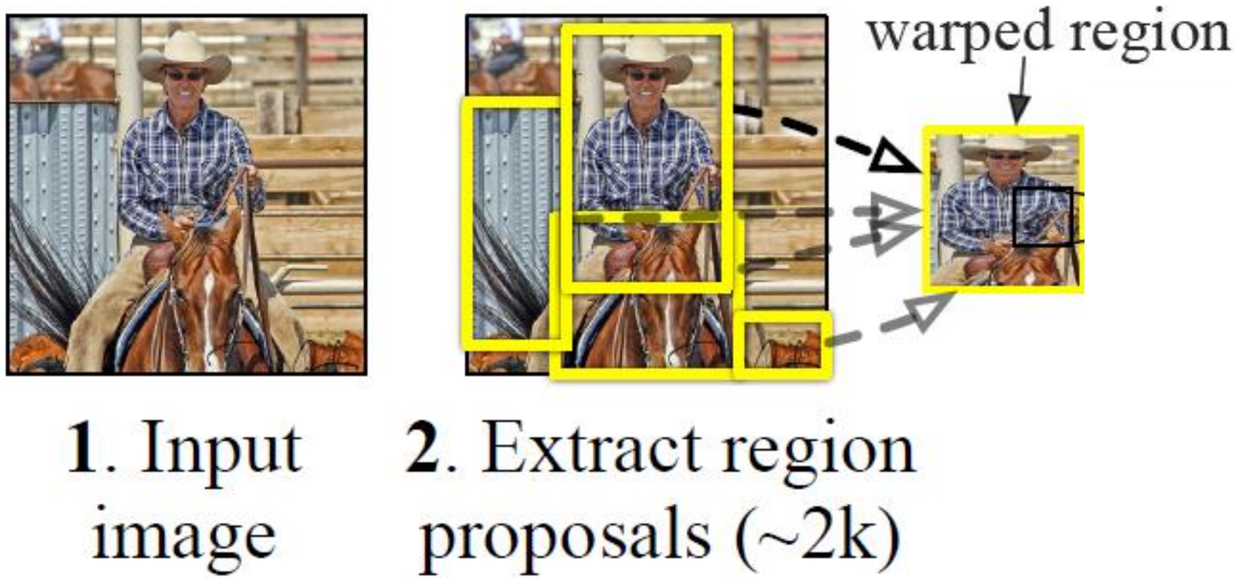
1. Input image

R-CNN



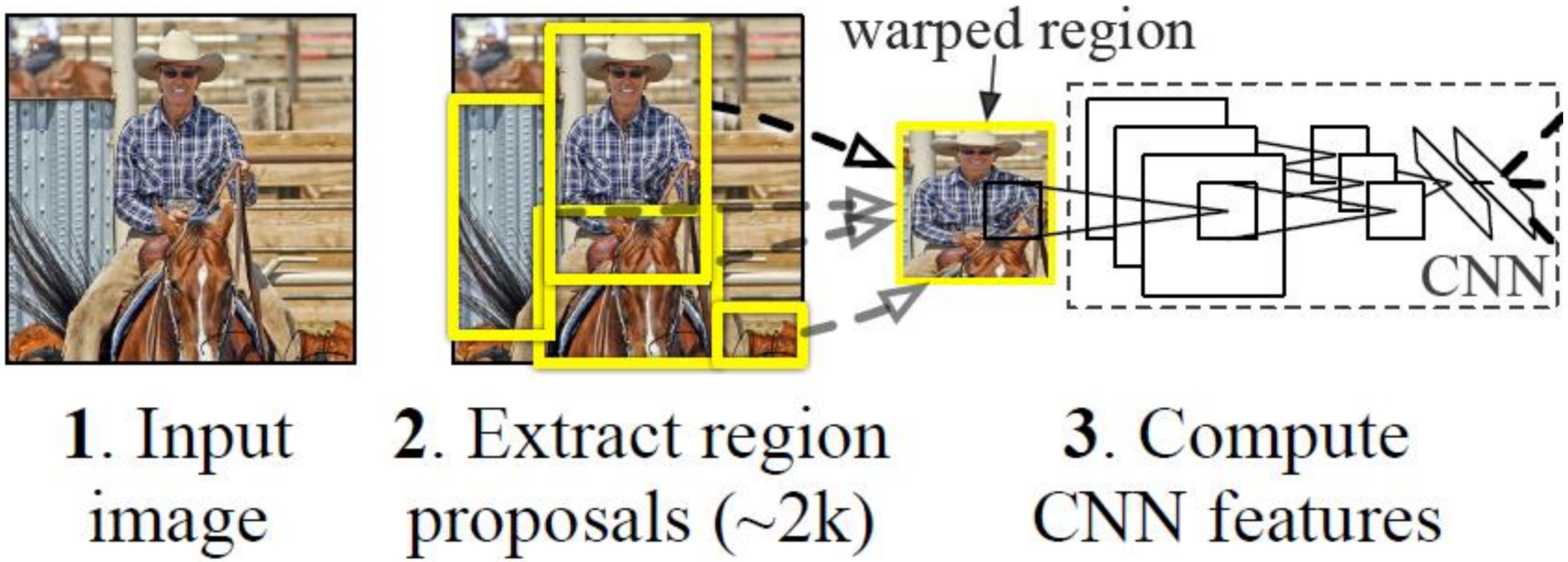
Selective Search

R-CNN



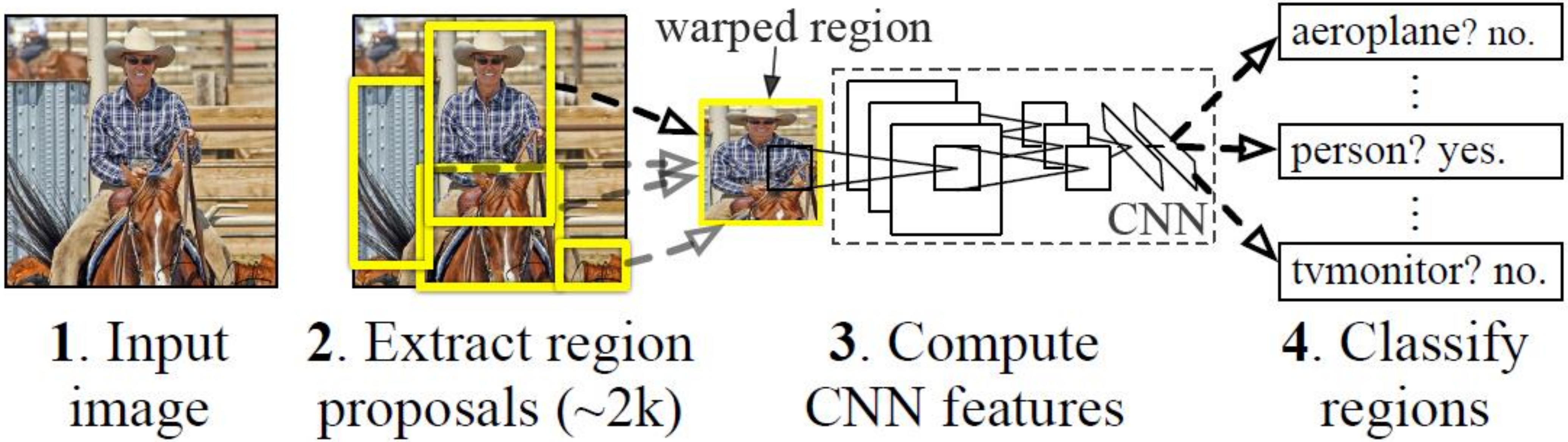
Regardless of the size or aspect ratio of the candidate region,
we warp all pixels in a tight bounding box around it to the required size.

R-CNN



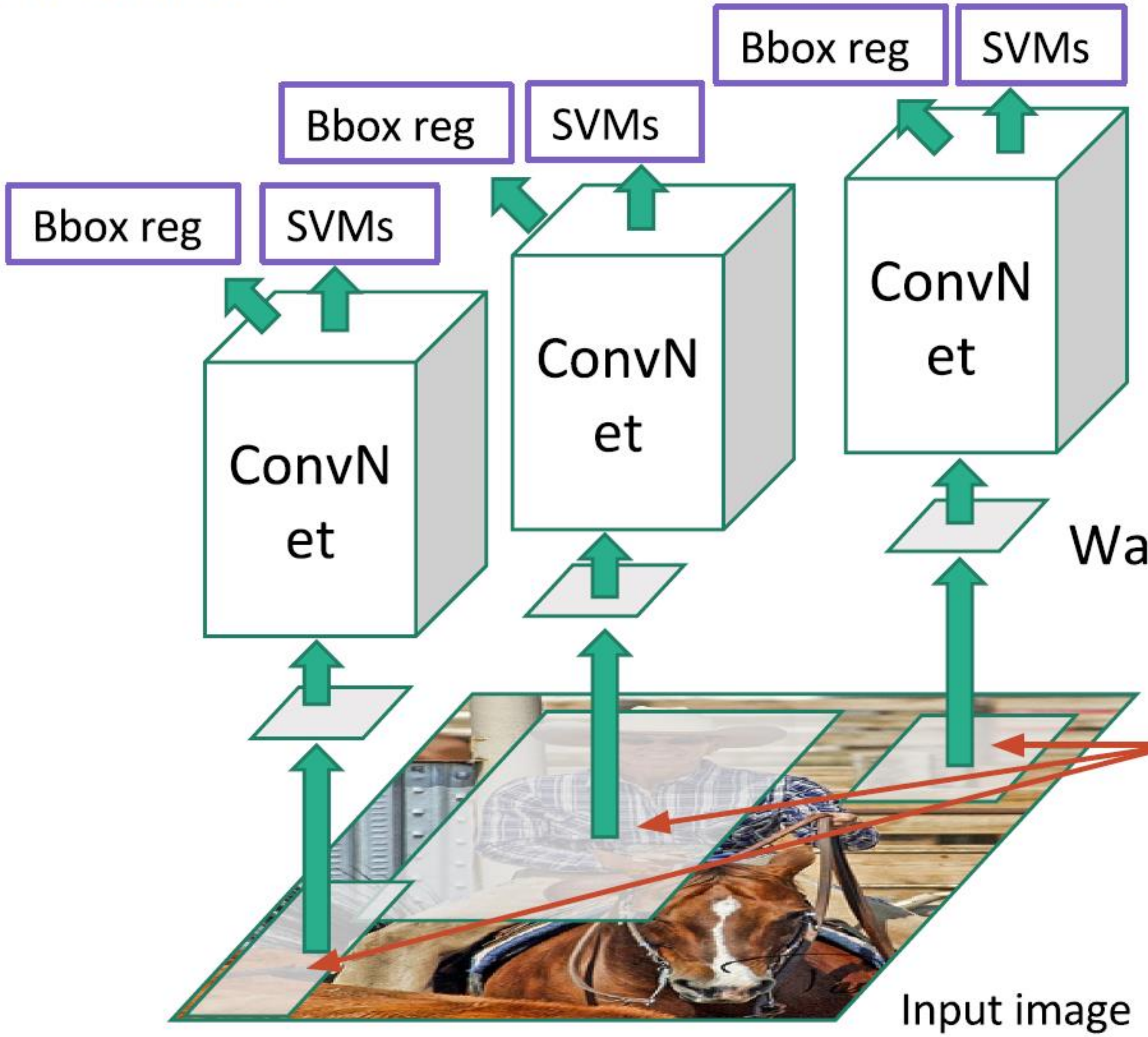
It requires a forward pass of the CNN (AlexNet) for every single region proposal for every single image (that's around 2000 forward passes per image!).

R-CNN



SVM / Bbox reg

R-CNN



Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

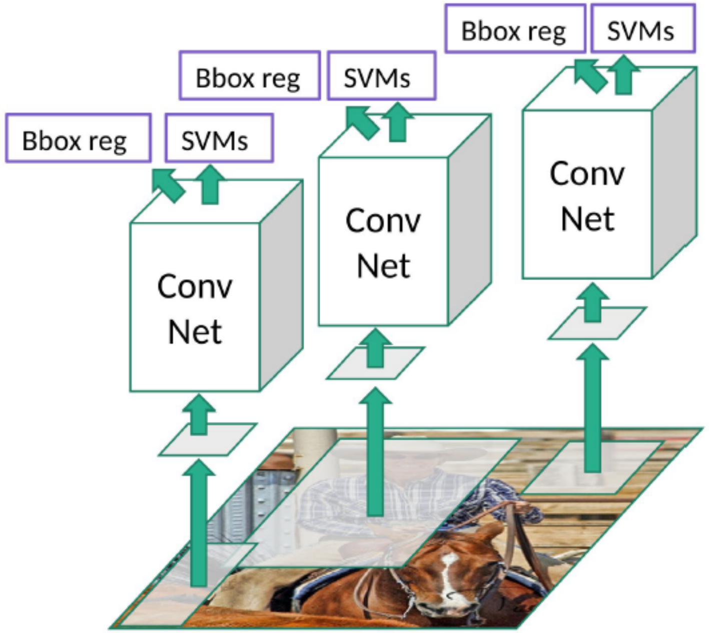
Warped image regions

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

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

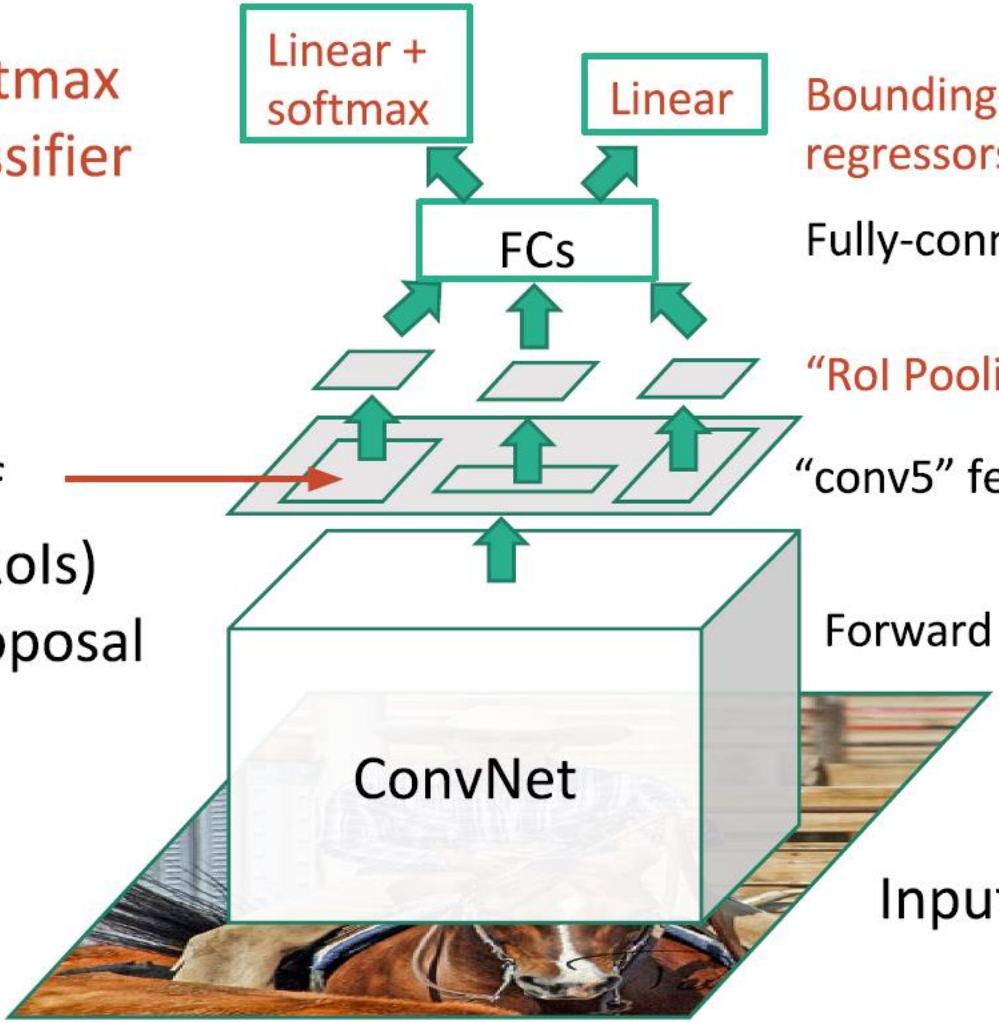
R-CNN: Problems

- 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]



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Slide copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN



Softmax classifier

Bounding-box regressors

Fully-connected layers

"RoI Pooling" layer

"conv5" feature map of image

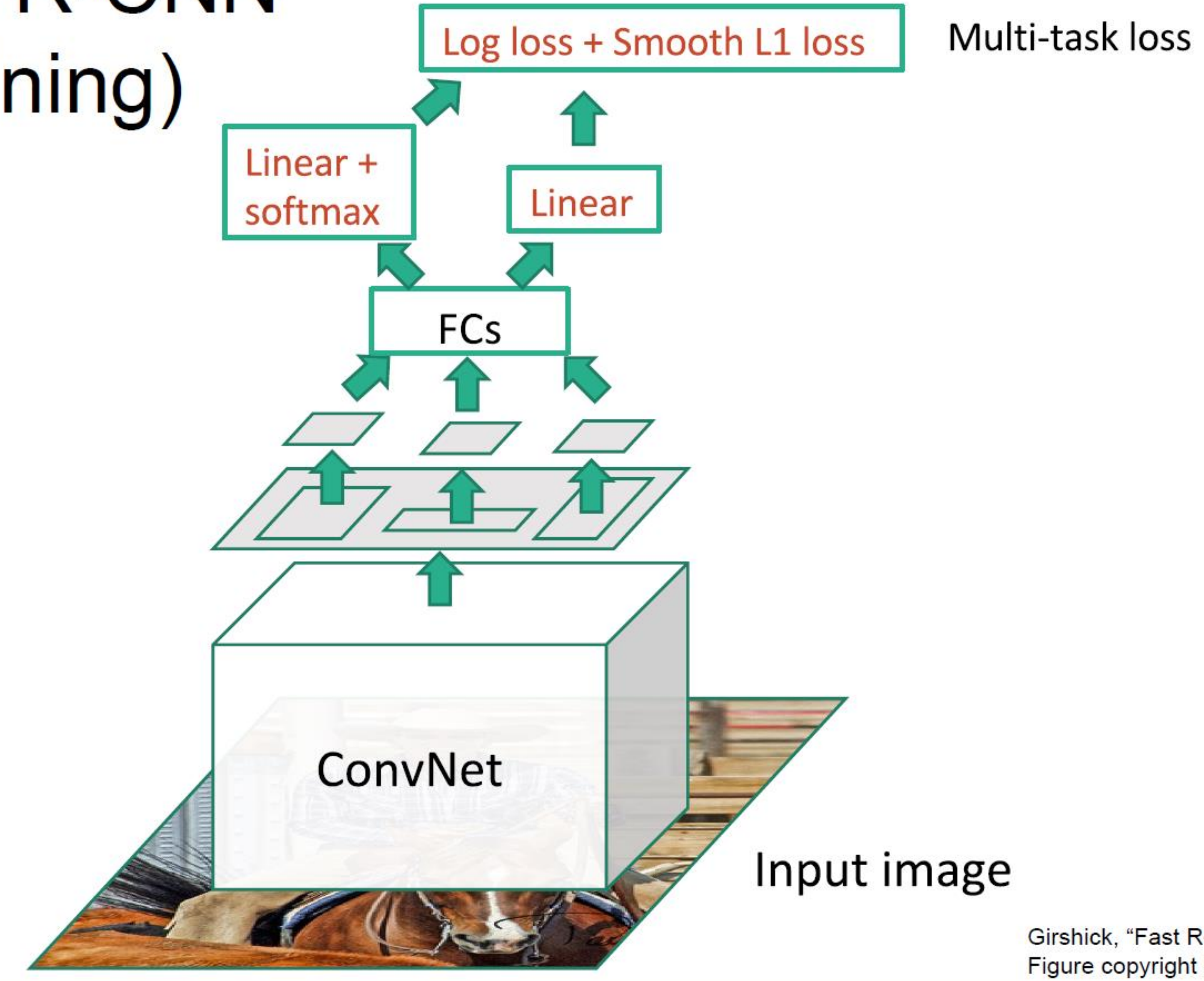
Regions of Interest (RoIs) from a proposal method

Forward whole image through ConvNet

Input image

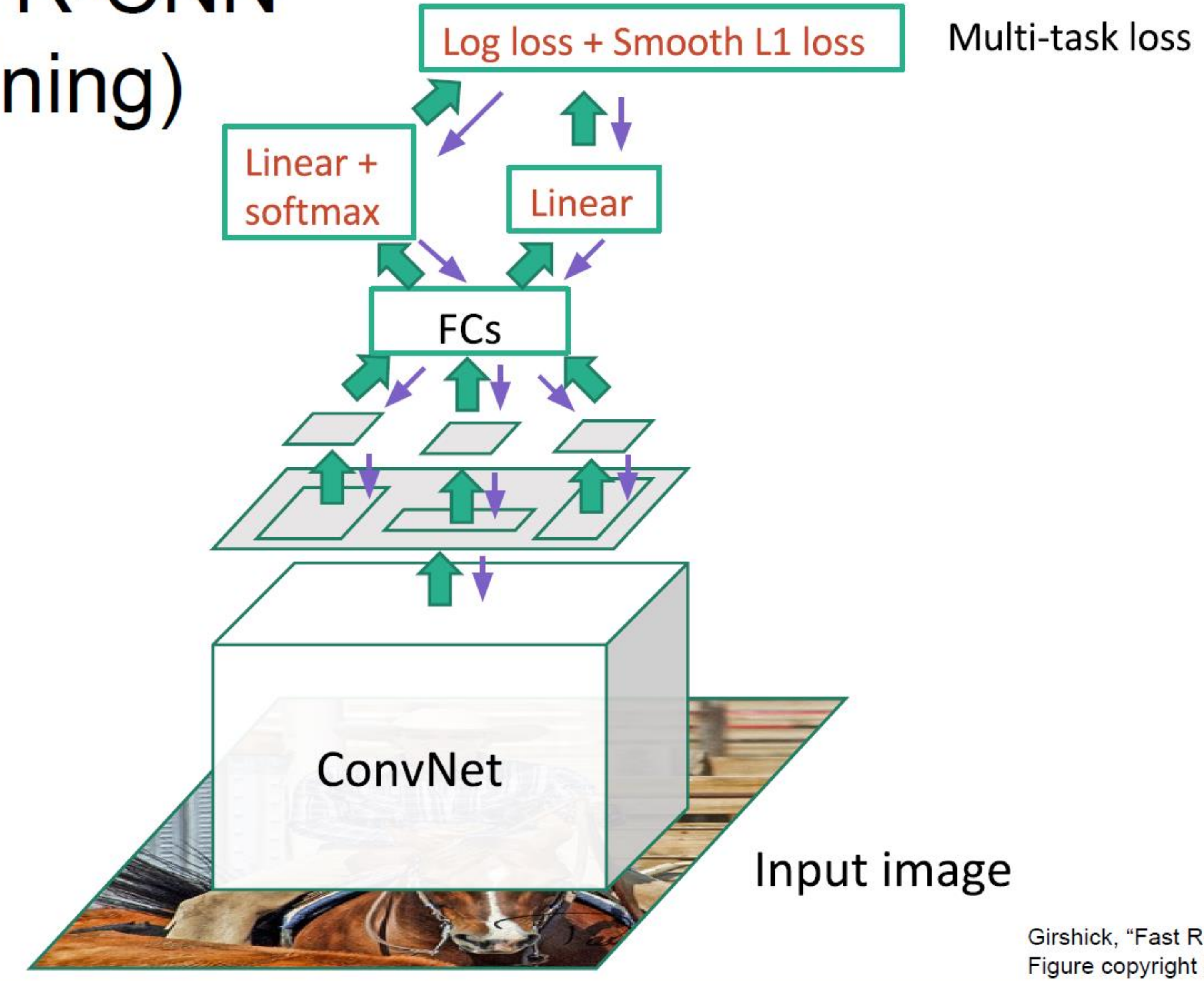
Girshick, "Fast R-CNN", ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN (Training)



Girshick, “Fast R-CNN”, ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN (Training)



Girshick, “Fast R-CNN”, ICCV 2015.
Figure copyright Ross Girshick, 2015; source. Reproduced with permission.

Fast R-CNN

" Fast R-CNN employs several innovations to improve training and testing speed "
while also increasing detection accuracy.

Fast R-cnn

1.Introduction

- We propose a single-stage training algorithm that jointly learns to classify object proposals and refine their spatial locations.

Fast R-cnn

1.Introduction

- We propose a single-stage training algorithm that jointly learns to classify object proposals and refine their spatial locations.
- The resulting method can train a very deep detection network (VGG16) 9x faster than R-CNN and 3x faster than SPPnet.

Fast R-cnn

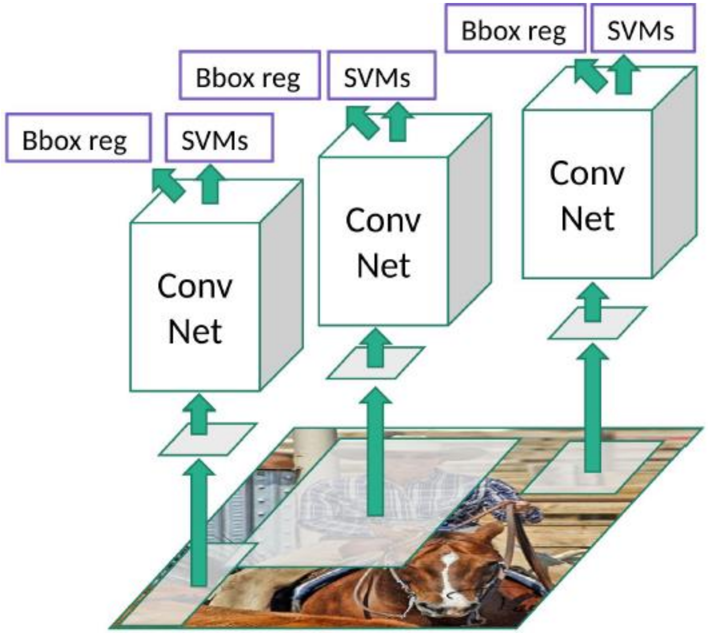
1.Introduction

- We propose a single-stage training algorithm that jointly learns to classify object proposals and refine their spatial locations.
- The resulting method can train a very deep detection network (VGG16) 9x faster than R-CNN and 3x faster than SPPnet.
- achieving top accuracy on PASCAL VOC 2012 with a mAP of 66% (vs. 62% for R-CNN)

Fast R-cnn

1.1. R-CNN and SPPnet

R-CNN has notable drawbacks:

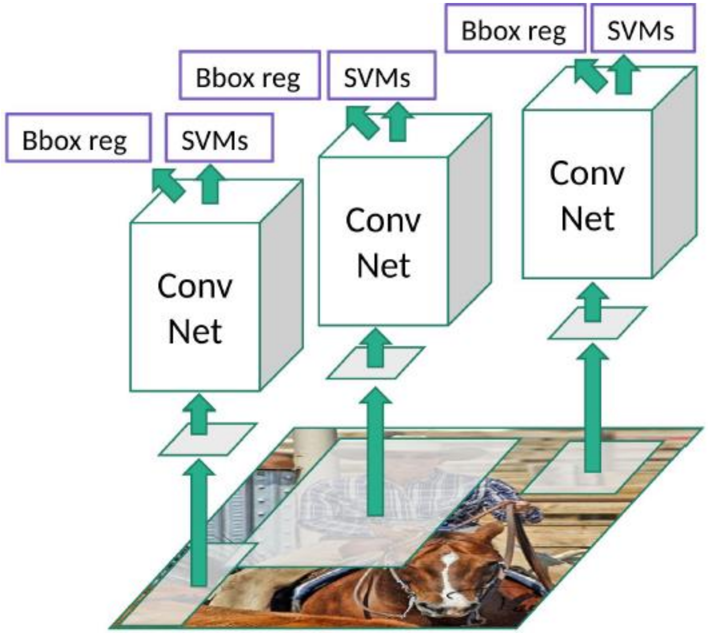


Fast R-cnn

1.1. R-CNN and SPPnet

R-CNN has notable drawbacks:

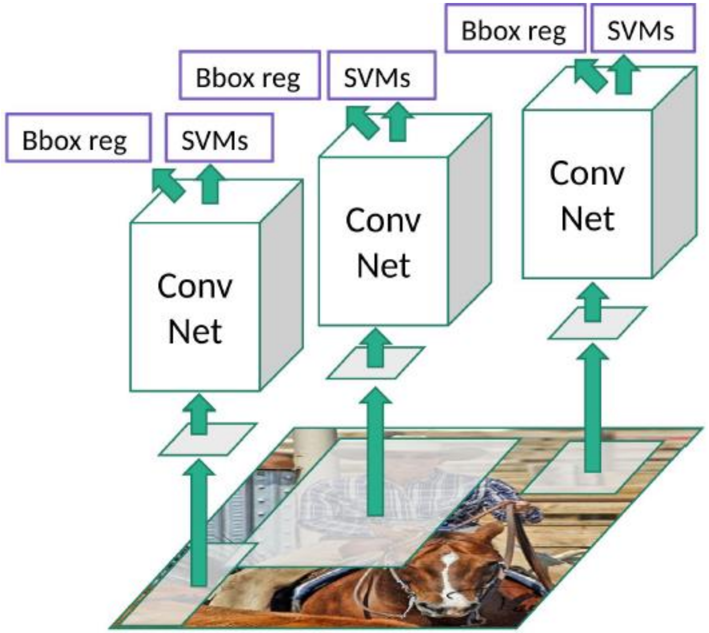
- Training is a multi-stage pipeline.



1.1. R-CNN and SPPnet

R-CNN has notable drawbacks:

- Training is a multi-stage pipeline.
- Training is expensive in space and time.

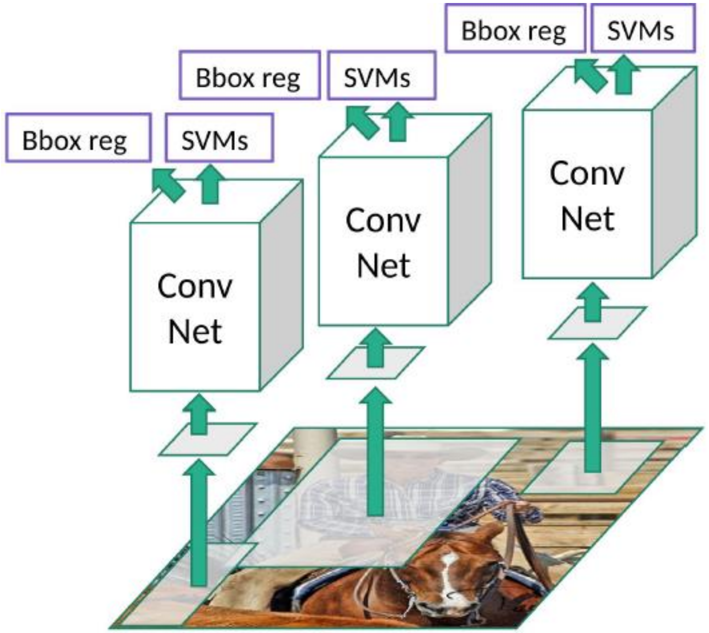


Fast R-cnn

1.1. R-CNN and SPPnet

R-CNN has notable drawbacks:

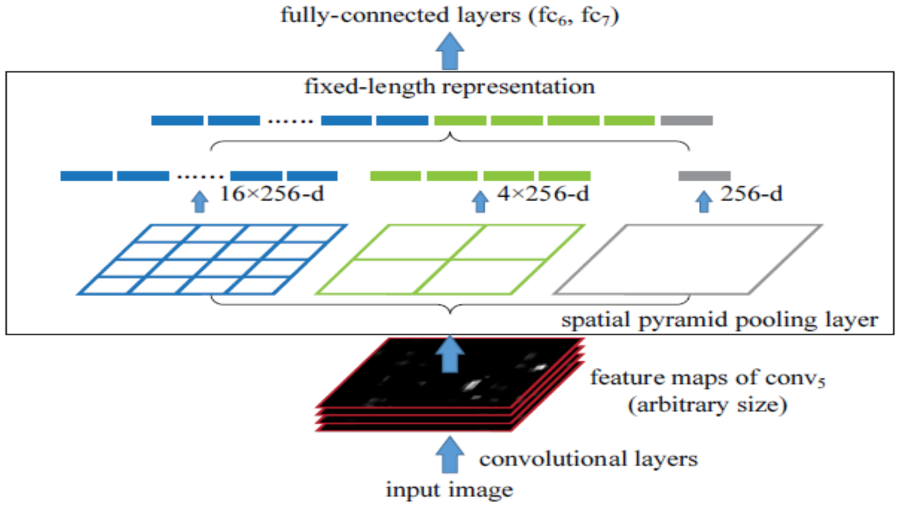
- Training is a multi-stage pipeline.
- Training is expensive in space and time.
- Object detection is slow.



Fast R-cnn

1.1. R-CNN and SPPnet

SPPnet also has notable drawbacks:

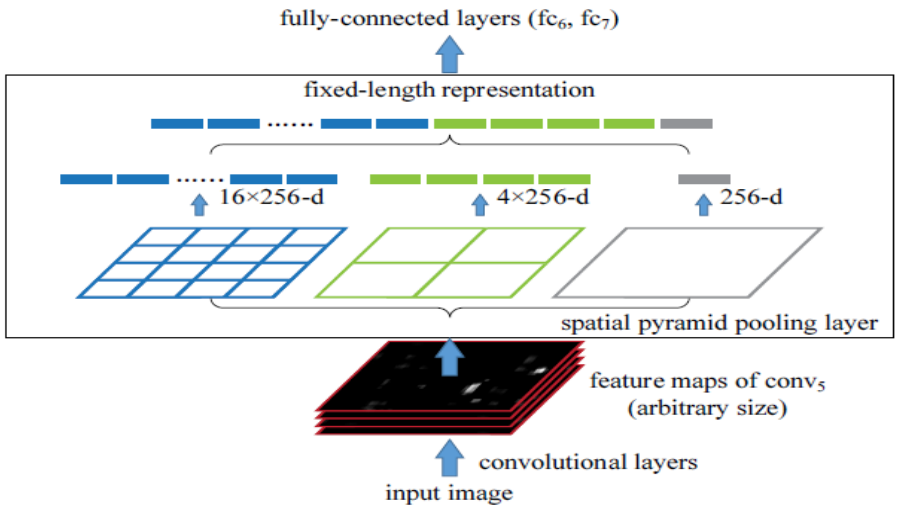

fully-connected layers (fc_6, fc_7)
fixed-length representation
16×256-d
4×256-d
256-d
spatial pyramid pooling layer
feature maps of $conv_5$ (arbitrary size)
convolutional layers
input image

Fast R-cnn

1.1. R-CNN and SPPnet

SPPnet also has notable drawbacks:

- Training is a multi-stage pipeline.

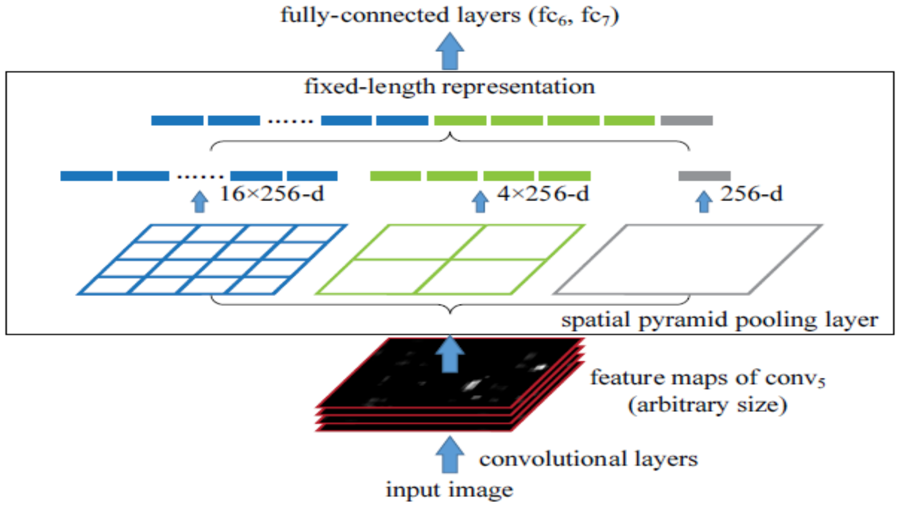


Fast R-cnn

1.1. R-CNN and SPPnet

SPPnet also has notable drawbacks:

- Training is a multi-stage pipeline.
- Features are also written to disk.

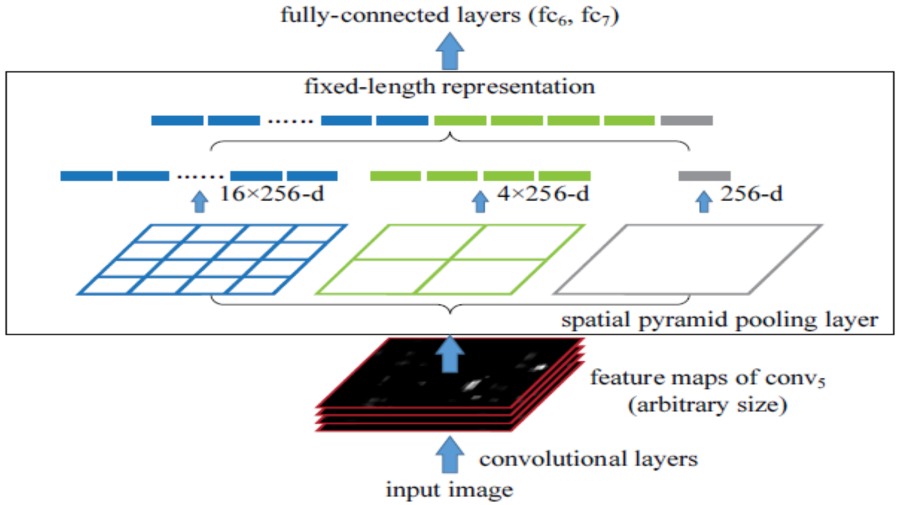


Fast R-cnn

1.1. R-CNN and SPPnet

SPPnet also has notable drawbacks:

- Training is a multi-stage pipeline.
- Features are also written to disk.
- the fine-tuning algorithm in SPPnet cannot update the convolutional layers that precede the spatial pyramid pooling.



Fast R-cnn

1.2. Contributions

Fixing the disadvantages of R-CNN and SPPnet,
while improving on their speed and accuracy.

Fast R-cnn

1.2. Contributions

Fixing the disadvantages of R-CNN and SPPnet,
while improving on their speed and accuracy.

- Higher detection quality (mAP) than R-CNN, SPPnet

Fast R-cnn

1.2. Contributions

Fixing the disadvantages of R-CNN and SPPnet,
while improving on their speed and accuracy.

- Higher detection quality (mAP) than R-CNN, SPPnet
- Training is single-stage, using a multi-task loss

Fast R-cnn

1.2. Contributions

Fixing the disadvantages of R-CNN and SPPnet,
while improving on their speed and accuracy.

- Higher detection quality (mAP) than R-CNN, SPPnet
- Training is single-stage, using a multi-task loss
- Training can update all network layers

Fast R-cnn

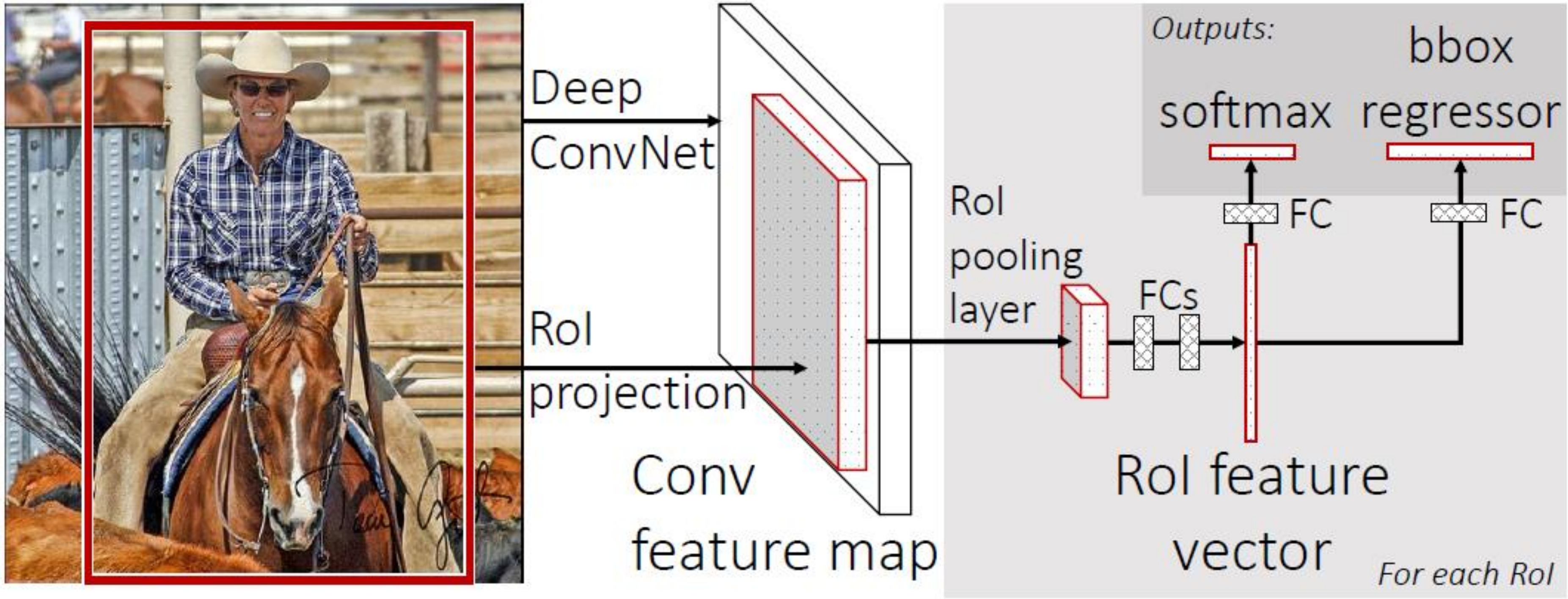
1.2. Contributions

Fixing the disadvantages of R-CNN and SPPnet,
while improving on their speed and accuracy.

- Higher detection quality (mAP) than R-CNN, SPPnet
- Training is single-stage, using a multi-task loss
- Training can update all network layers
- No disk storage is required for feature caching

Fast R-cnn

2. Fast R-CNN architecture and training



Fast R-cnn

2. Fast R-CNN architecture and training

- A Fast R-CNN network takes as input an entire image and a set of object proposals.

Fast R-cnn

2. Fast R-CNN architecture and training

- A Fast R-CNN network takes as input an entire image and a set of object proposals.
- The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.

Fast R-cnn

2. Fast R-CNN architecture and training

- A Fast R-CNN network takes as input an entire image and a set of object proposals.
- The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.
- Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map.

Fast R-cnn

2. Fast R-CNN architecture and training

- A Fast R-CNN network takes as input an entire image and a set of object proposals.
- The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.
- Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map.
- Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers:

Fast R-cnn

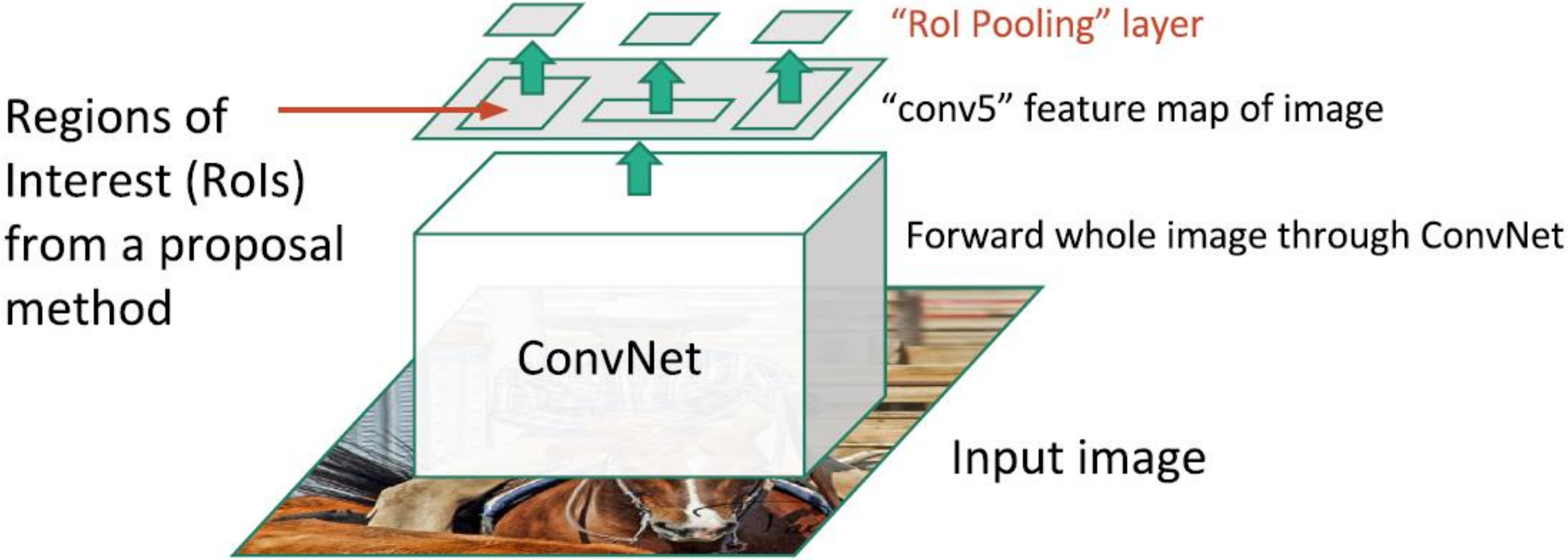
2. Fast R-CNN architecture and training

- A Fast R-CNN network takes as input an entire image and a set of object proposals.
- The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map.
- Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map.
- Each feature vector is fed into a sequence of fully connected (fc) layers that finally branch into two sibling output layers:

“one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes.”

Fast R-cnn

2.1. RoI pooling layer



"RoI Pooling" layer
"conv5" feature map of image
Regions of Interest (RoIs) from a proposal method
Forward whole image through ConvNet
ConvNet
Input image

Fast R-cnn

2.1. The RoI pooling layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)

Fast R-cnn

2.1. The RoI pooling layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window into a conv feature map.

Fast R-cnn

2.1. The RoI pooling layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window into a conv feature map.
- Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).

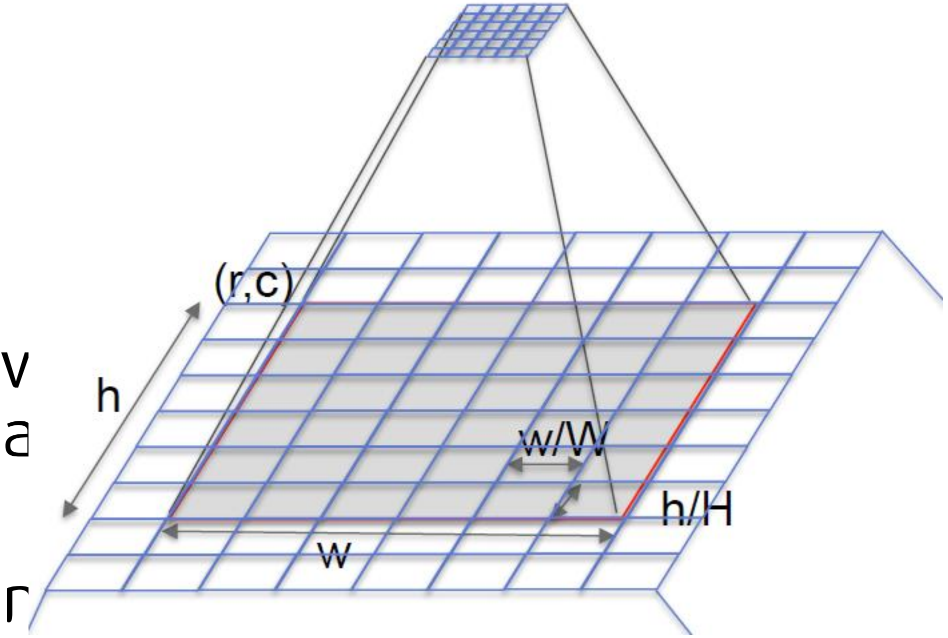
Fast R-cnn

2.1. The RoI pooling layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window into a conv feature map.
- Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).
- RoI max pooling works by dividing the h x w RoI window into an H x W grid of sub-windows of approximate size h/H x w/W

Fast R-cnn

2.1. The RoI pooling layer



- The RoI pooling layer uses max pooling to conv valid region of interest into a small feature ma extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window ir
- Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).
- RoI max pooling works by dividing the h x w RoI window into an H x W grid of sub-windows of approximate size h/H x w/W

Fast R-cnn

2.1. The RoI pooling layer

- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window into a conv feature map.
- Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).
- RoI max pooling works by dividing the h x w RoI window into an H x W grid of sub-windows of approximate size h/H x w/W
- max-pooling the values in each sub-window into the corresponding output grid cell.

Fast R-cnn

2.1. The RoI pooling layer

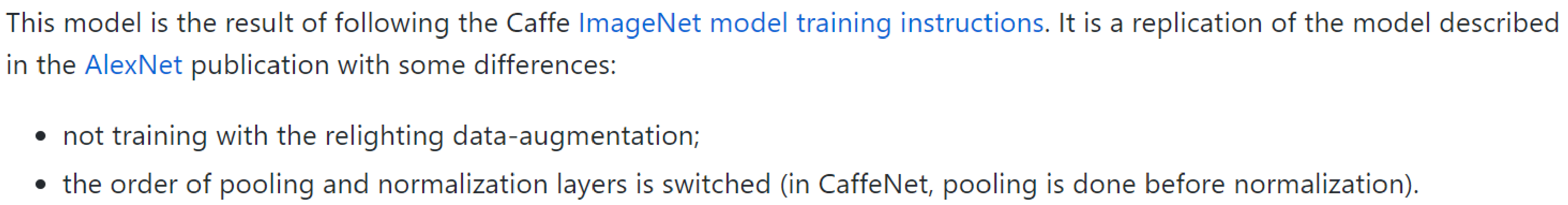
- The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small feature map with a fixed spatial extent of HxW (e.g., 7x7)
- In this paper, an RoI is a rectangular window into a conv feature map.
- Each RoI is defined by a four-tuple (r, c, h, w) that specifies its top-left corner (r, c) and its height and width (h, w).
- RoI max pooling works by dividing the h x w RoI window into an H x W grid of sub-windows of approximate size h/H x w/W
- max-pooling the values in each sub-window into the corresponding output grid cell.
- The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets in which there is only one pyramid level.

Fast R-cnn

2.2. Initializing from pre-trained networks

- We experiment with three pre-trained ImageNet networks.

CaffeNet, VGG_CNN_M_1024, VGG16



This model is the result of following the Caffe ImageNet model training instructions. It is a replication of the model described in the AlexNet publication with some differences:

- not training with the relighting data-augmentation;
- the order of pooling and normalization layers is switched (in CaffeNet, pooling is done before normalization).

2.2. Initializing from pre-trained networks

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations.

Fast R-cnn

2.2. Initializing from pre-trained networks

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations.

- First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net’s first fully connected layer (e.g., H = W = 7 for VGG16).

Fast R-cnn

2.2. Initializing from pre-trained networks

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations.

- First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (e.g., H = W = 7 for VGG16).
- Second, the network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers (a fully connected layer and softmax over K+ 1 categories and category-specific bounding-box regressors).

Fast R-cnn

2.2. Initializing from pre-trained networks

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations.

- First, the last max pooling layer is replaced by a RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (e.g., H = W = 7 for VGG16).
- Second, the network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling layers (a fully connected layer and softmax over K+ 1 categories and category-specific bounding-box regressors).
- Third, the network is modified to take two data inputs: a list of images and a list of RoIs in those images.

Fast R-cnn

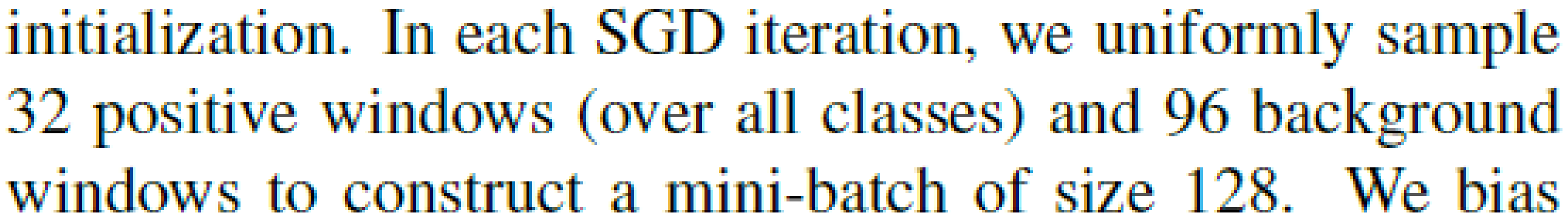
2.3. Fine-tuning for detection

- SPPnet is unable to update weights below the spatial pyramid pooling layer.

Fast R-cnn

2.3. Fine-tuning for detection

- SPPnet is unable to update weights below the spatial pyramid pooling layer.
- The root cause is that back-propagation through the SPP layer is highly inefficient when each training sample (i.e. RoI) comes from a different image, which is exactly how R-CNN and SPPnet networks are trained.



initialization. In each SGD iteration, we uniformly sample 32 positive windows (over all classes) and 96 background windows to construct a mini-batch of size 128. We bias

Fast R-cnn

2.3. Fine-tuning for detection

- In Fast R-CNN training, stochastic gradient descent (SGD) mini-batches are sampled hierarchically.

Fast R-cnn

2.3. Fine-tuning for detection

- In Fast R-CNN training, stochastic gradient descent (SGD) mini-batches are sampled hierarchically.
- first by sampling N images and then by sampling R/N RoIs from each image.

Fast R-cnn

2.3. Fine-tuning for detection

- In Fast R-CNN training, stochastic gradient descent (SGD) mini-batches are sampled hierarchically.
- first by sampling N images and then by sampling R/N RoIs from each image.
- Critically, RoIs from the same image share computation and memory in the forward and backward passes.

Fast R-cnn

2.3. Fine-tuning for detection

- In Fast R-CNN training, stochastic gradient descent (SGD) mini-batches are sampled hierarchically.
- first by sampling N images and then by sampling R/N RoIs from each image.
- Critically, RoIs from the same image share computation and memory in the forward and backward passes.
- For example, when using N = 2 and R = 128, the proposed training scheme is roughly 64x faster than sampling one RoI from 128 different images (i.e., the R-CNN and SPPnet strategy).

Fast R-cnn

2.3. Fine-tuning for detection

- One concern over this strategy is it may cause slow training convergence because RoIs from the same image are correlated.

2.3. Fine-tuning for detection

- One concern over this strategy is it may cause slow training convergence because RoIs from the same image are correlated.
- This concern does not appear to be a practical issue and we achieve good results with N = 2 and R = 128 using fewer SGD iterations than R-CNN.

Fast R-cnn

2.3. Fine-tuning for detection

- One concern over this strategy is it may cause slow training convergence because RoIs from the same image are correlated.
- This concern does not appear to be a practical issue and we achieve good results with N = 2 and R = 128 using fewer SGD iterations than R-CNN.
- Fast R-CNN uses a streamlined(간결한) training process with one fine-tuning stage that jointly optimizes a softmax classifier and bounding-box regressors, rather than training a softmax classifier, SVMs, and regressors in three separate stages

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss

- A Fast R-CNN network has two sibling output layers.

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss

- A Fast R-CNN network has two sibling output layers.
- The first outputs a discrete probability distribution (per RoI), p over K+1 categories. (K : num of class)

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss

- A Fast R-CNN network has two sibling output layers.
- The first outputs a discrete probability distribution (per RoI), p over K+1 categories. (K : num of class)
- As usual, p is computed by a softmax over the K+1 outputs of a fully connected layer.

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss

- A Fast R-CNN network has two sibling output layers.
- The first outputs a discrete probability distribution (per RoI), p over K+1 categories. (K : num of class)
- As usual, p is computed by a softmax over the K+1 outputs of a fully connected layer.
- The second sibling layer outputs bounding-box regression offsets $t^k = (t^k_{\mathrm{x}}, t^k_{\mathrm{y}}, t^k_{\mathrm{w}}, t^k_{\mathrm{h}})$or each of the K object classes, indexed by k.

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss

- A Fast R-CNN network has two sibling output layers.
- The first outputs a discrete probability distribution (per RoI), p over K+1 categories. (K : num of class)
- As usual, p is computed by a softmax over the K+1 outputs of a fully connected layer.
- The second sibling layer outputs bounding-box regression offsets $t^k = (t^k_{\mathrm{x}}, t^k_{\mathrm{y}}, t^k_{\mathrm{w}}, t^k_{\mathrm{h}})$r each of the K object classes, indexed by k.
- Each training RoI is labeled with a ground-truth class u and a ground-truth bounding-box regression target v.

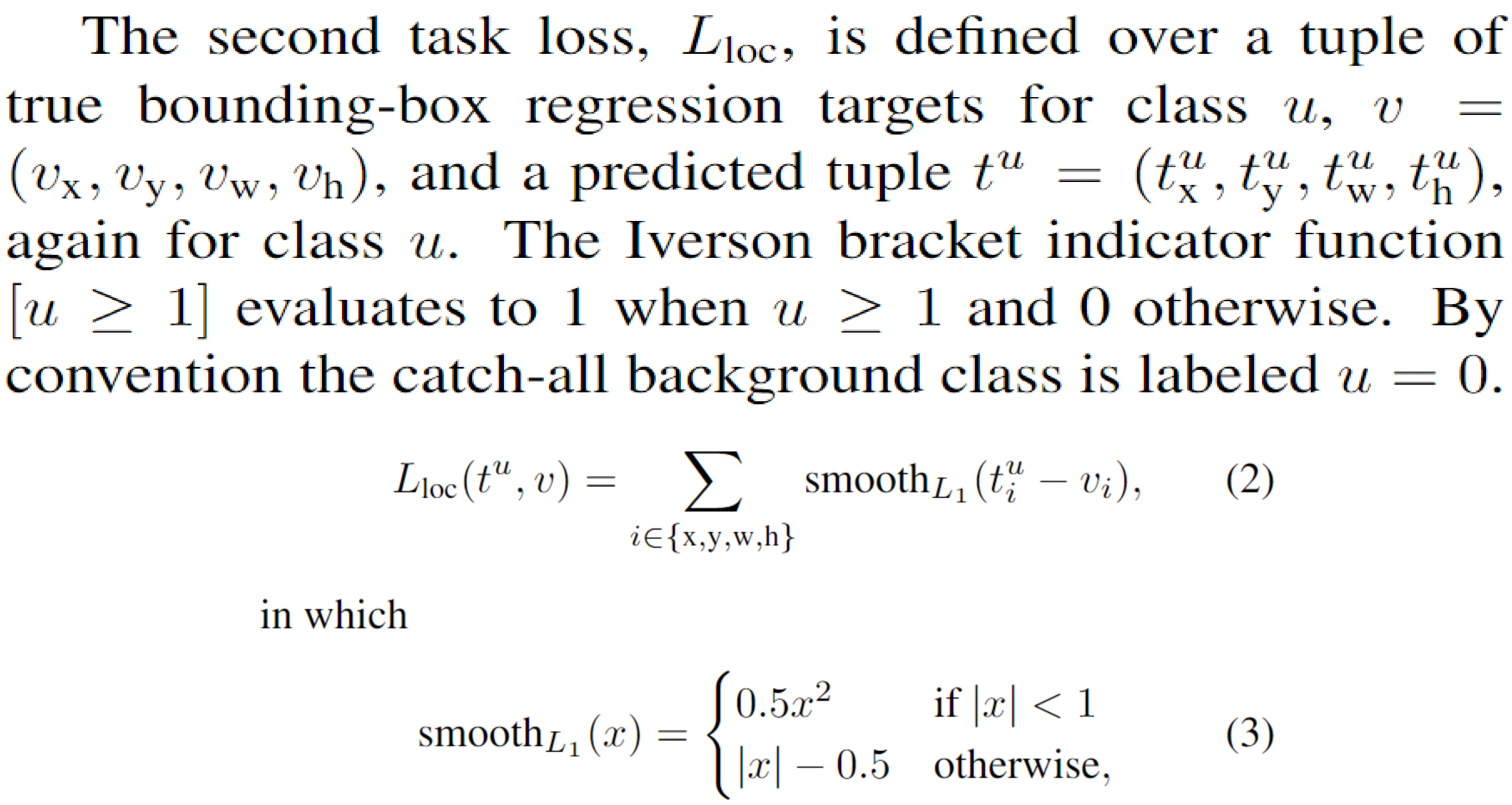
$$L(p, u, t^u, v) = L_{\mathrm{cls}}(p, u) + \lambda[u \geq 1] L_{\mathrm{loc}}(t^u, v), \qquad (1)$$

in which $L_{\mathrm{cls}}(p, u) = -\log p_u$ is log loss for true class u.

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss



The second task loss, L_{loc}, is defined over a tuple of true bounding-box regression targets for class u, $v = (v_{\text{x}}, v_{\text{y}}, v_{\text{w}}, v_{\text{h}})$, and a predicted tuple $t^u = (t^u_{\text{x}}, t^u_{\text{y}}, t^u_{\text{w}}, t^u_{\text{h}})$, again for class u. The Iverson bracket indicator function $[u \geq 1]$ evaluates to 1 when $u \geq 1$ and 0 otherwise. By convention the catch-all background class is labeled $u = 0$.

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x},\text{y},\text{w},\text{h}\}} \text{smooth}_{L_1}(t^u_i - v_i), \qquad (2)$$

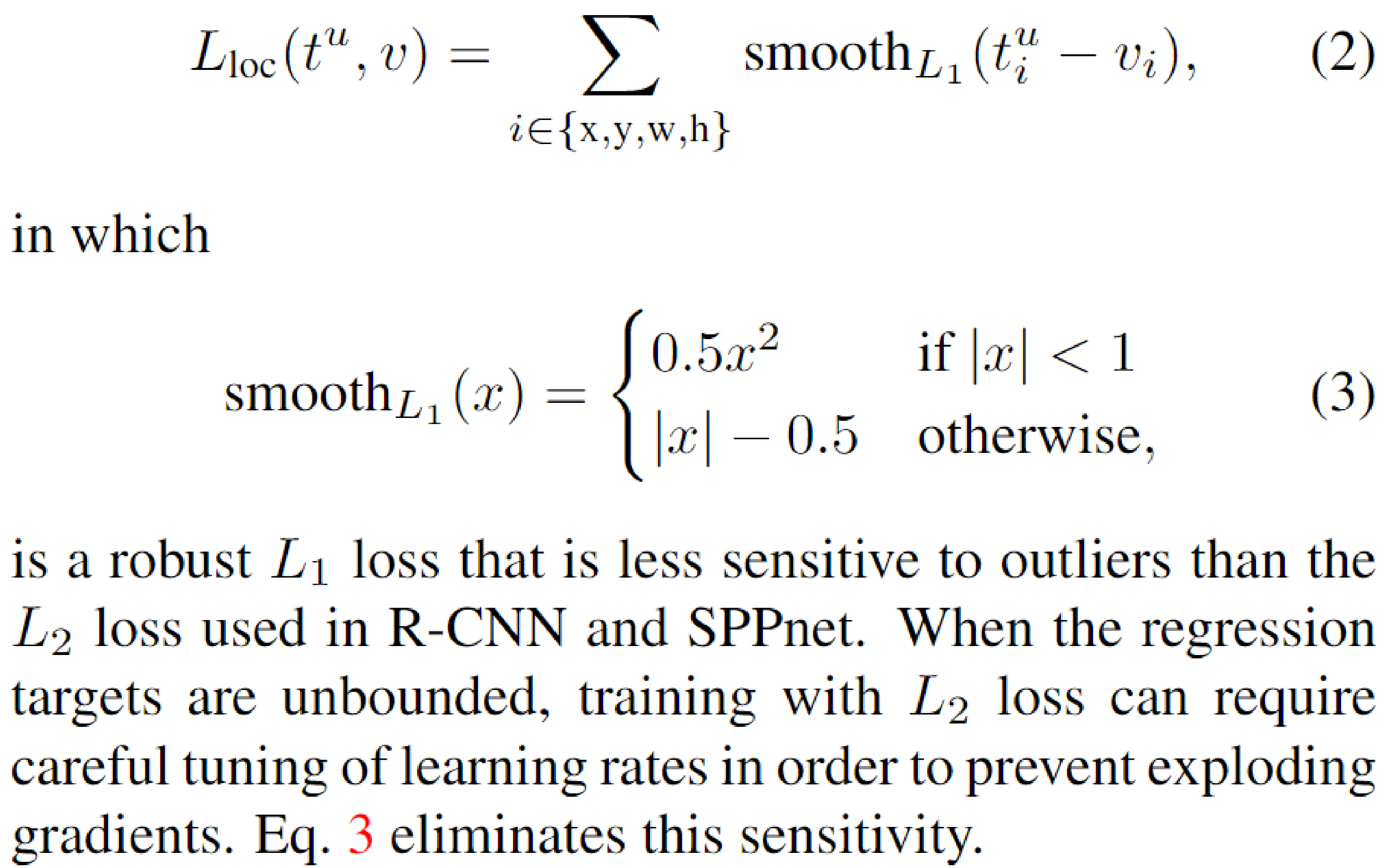
in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \qquad (3)$$

Fast R-cnn

2.3. Fine-tuning for detection

- Multi-task loss



$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i), \qquad (2)$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \qquad (3)$$

is a robust L_1 loss that is less sensitive to outliers than the L_2 loss used in R-CNN and SPPnet. When the regression targets are unbounded, training with L_2 loss can require careful tuning of learning rates in order to prevent exploding gradients. Eq. 3 eliminates this sensitivity.

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random (as is common practice, we actually iterate over permutations of the dataset).

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random (as is common practice, we actually iterate over permutations of the dataset).
- We use mini-batches of size R = 128, sampling 64 RoIs from each image.

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random (as is common practice, we actually iterate over permutations of the dataset).
- We use mini-batches of size R = 128, sampling 64 RoIs from each image.
- We take 25% of the RoIs from object proposals that have intersection over union (IoU) overlap with a ground truth bounding box of at least 0.5.

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random (as is common practice, we actually iterate over permutations of the dataset).
- We use mini-batches of size R = 128, sampling 64 RoIs from each image.
- We take 25% of the RoIs from object proposals that have intersection over union (IoU) overlap with a ground truth bounding box of at least 0.5.
- These RoIs comprise the examples labeled with a foreground object class, i.e. u >= 1.

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5)

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5)
- These are the background examples and are labeled with u = 0.

Fast R-cnn

2.3. Fine-tuning for detection

- Mini-batch sampling

- The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5)
- These are the background examples and are labeled with u = 0.
- The lower threshold of 0.1 appears to act as a heuristic for hard example mining

Fast R-cnn

2.3. Fine-tuning for detection

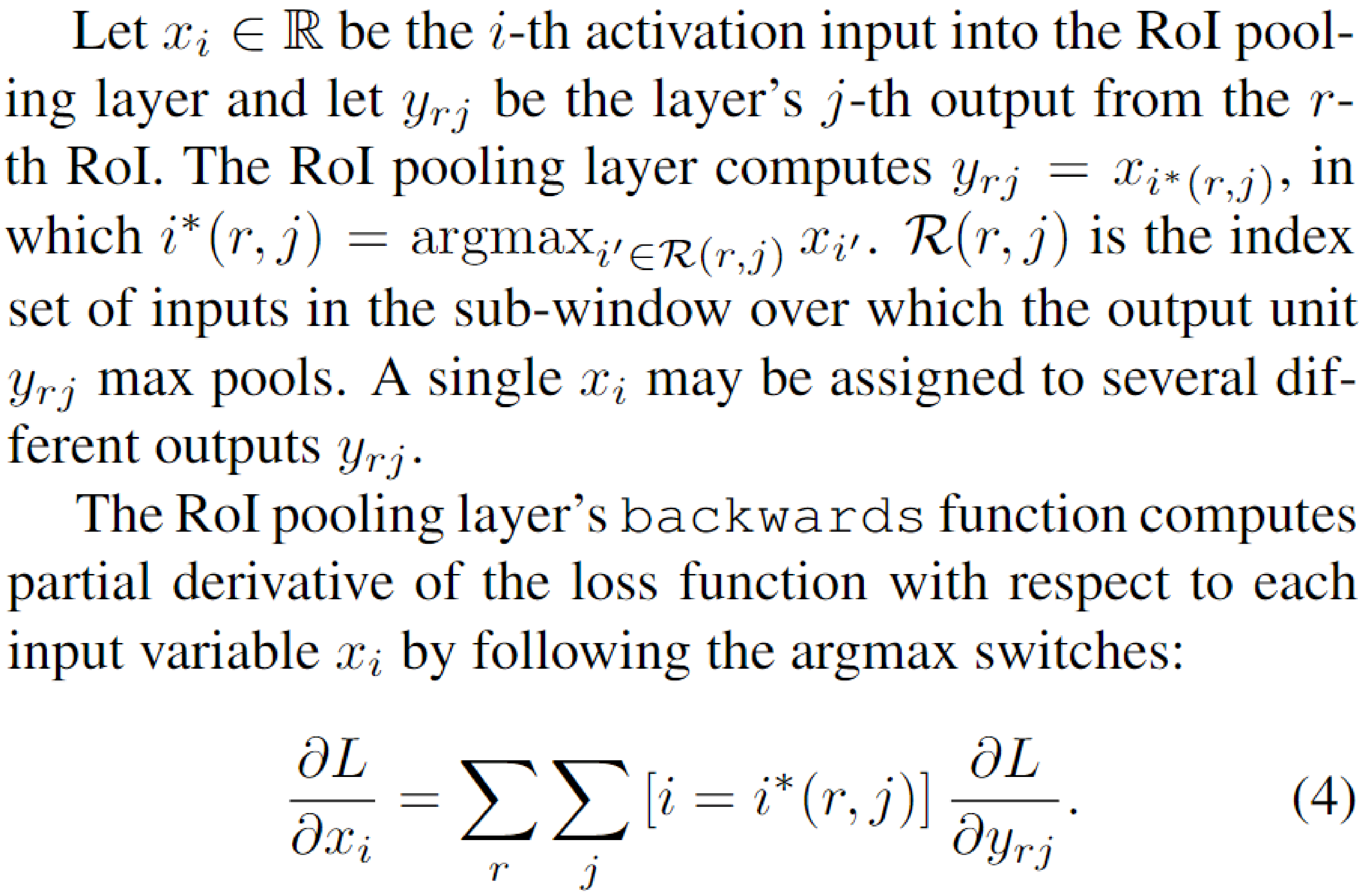
- Mini-batch sampling

- The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5)
- These are the background examples and are labeled with u = 0.
- The lower threshold of 0.1 appears to act as a heuristic for hard example mining
- During training, images are horizontally flipped with probability 0:5. No other data augmentation is used.

Fast R-cnn

2.3. Fine-tuning for detection

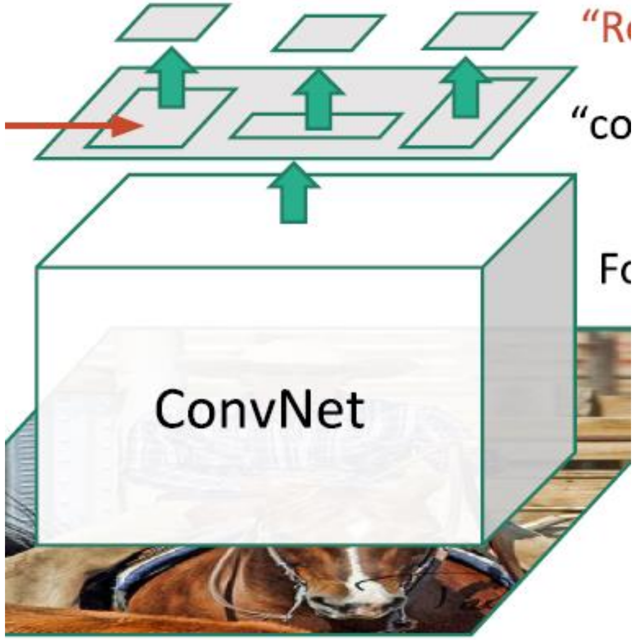
- Back-propagation through RoI pooling layers.



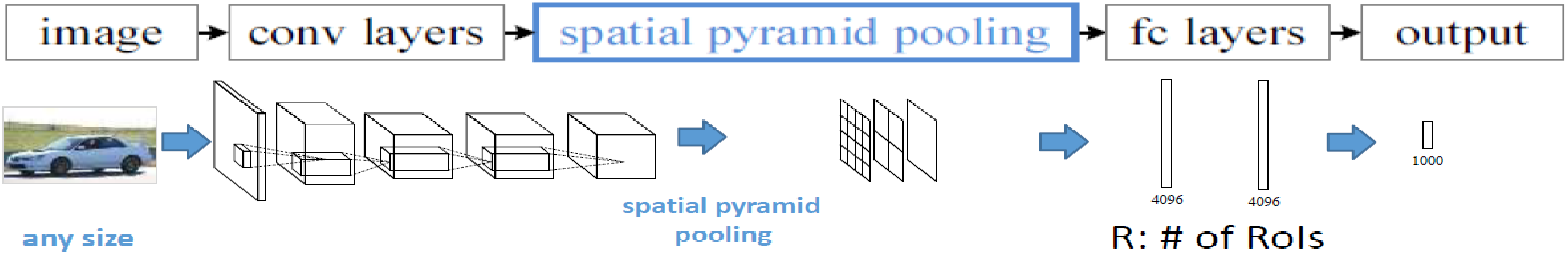
Let $x_i \in \mathbb{R}$ be the i-th activation input into the RoI pooling layer and let y_{rj} be the layer's j-th output from the r-th RoI. The RoI pooling layer computes $y_{rj} = x_{i^*(r,j)}$, in which $i^*(r,j) = \operatorname{argmax}_{i' \in \mathcal{R}(r,j)} x_{i'}$. $\mathcal{R}(r,j)$ is the index set of inputs in the sub-window over which the output unit y_{rj} max pools. A single x_i may be assigned to several different outputs y_{rj}.

The RoI pooling layer's `backwards` function computes partial derivative of the loss function with respect to each input variable x_i by following the argmax switches:

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j \left[i = i^*(r,j)\right] \frac{\partial L}{\partial y_{rj}}. \tag{4}$$

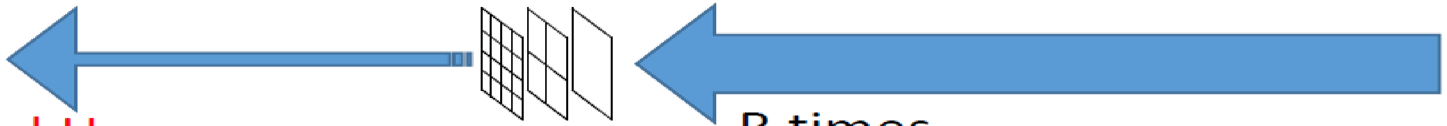


SPP-net Forward



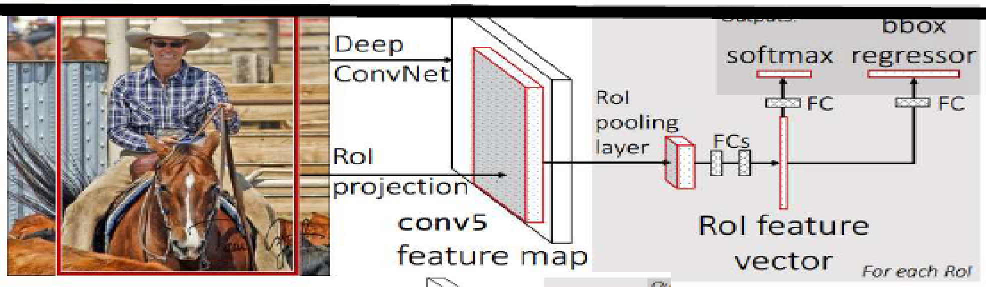
Backward

R times
Slow and Heavy computation

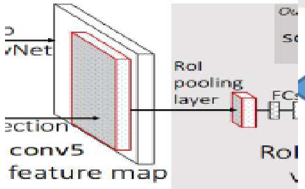


R times back-propagation

FRCN Backward



1 time back-propagation
Fast & Efficient



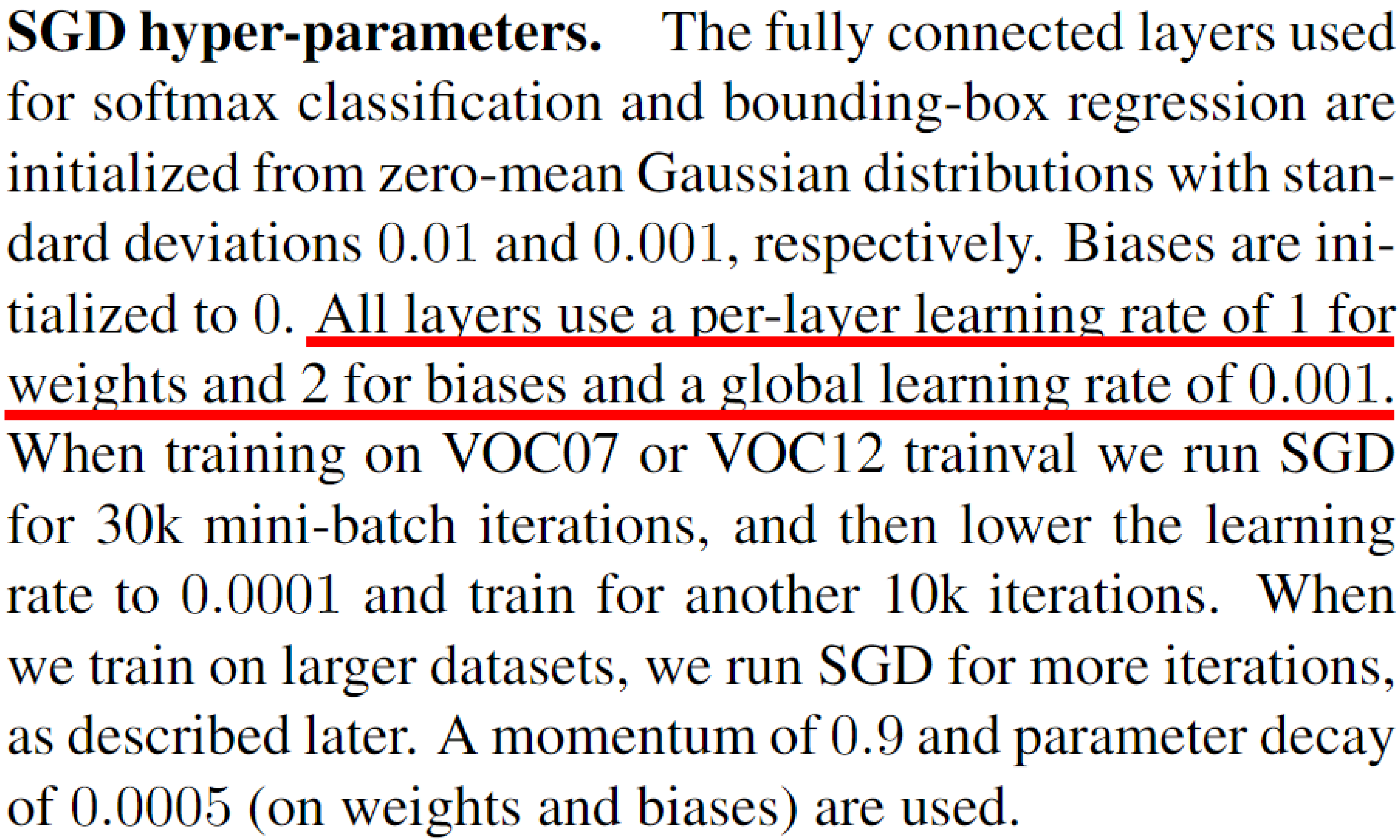
R times back-propagation

$$\frac{\partial L}{\partial x} = \sum_{r \in R} \sum_{y \in r} [y \text{ pooled } x] \frac{\partial L}{\partial y}. \qquad (4)$$

Fast R-cnn

2.3. Fine-tuning for detection

SGD hyper-parameters



SGD hyper-parameters. The fully connected layers used for softmax classification and bounding-box regression are initialized from zero-mean Gaussian distributions with standard deviations 0.01 and 0.001, respectively. Biases are initialized to 0. All layers use a per-layer learning rate of 1 for weights and 2 for biases and a global learning rate of 0.001. When training on VOC07 or VOC12 trainval we run SGD for 30k mini-batch iterations, and then lower the learning rate to 0.0001 and train for another 10k iterations. When we train on larger datasets, we run SGD for more iterations, as described later. A momentum of 0.9 and parameter decay of 0.0005 (on weights and biases) are used.

Fast R-cnn

2.4. Scale invariance

- We explore two ways of achieving scale invariant object detection:

(1) via “brute force” learning

each image is processed at a pre-defined pixel size during both training and testing.

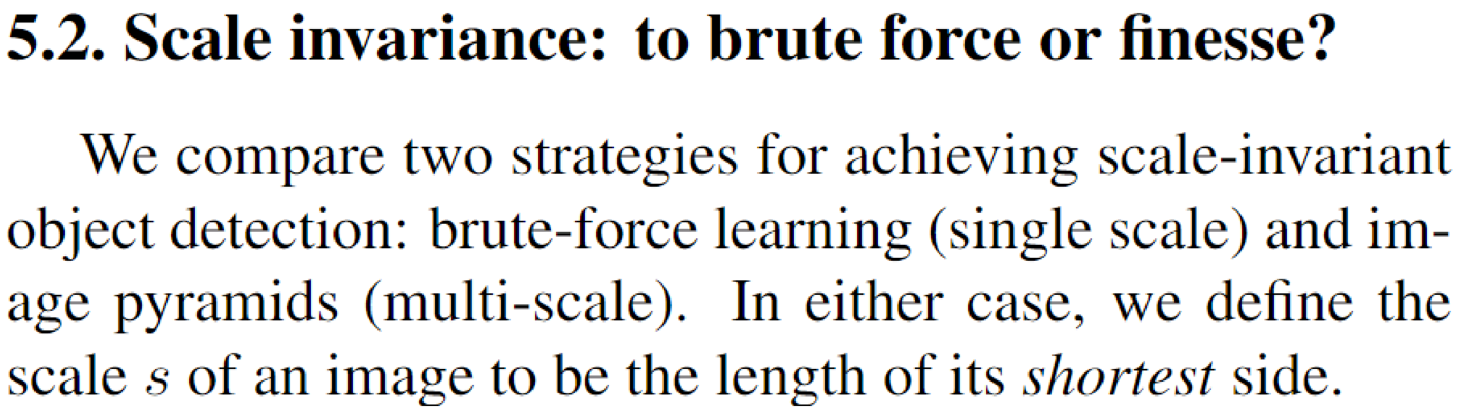
(2) by using image pyramids

The multi-scale approach, in contrast, provides approximate scale-invariance to the network through an image pyramid.

Fast R-cnn

2.4. Scale invariance

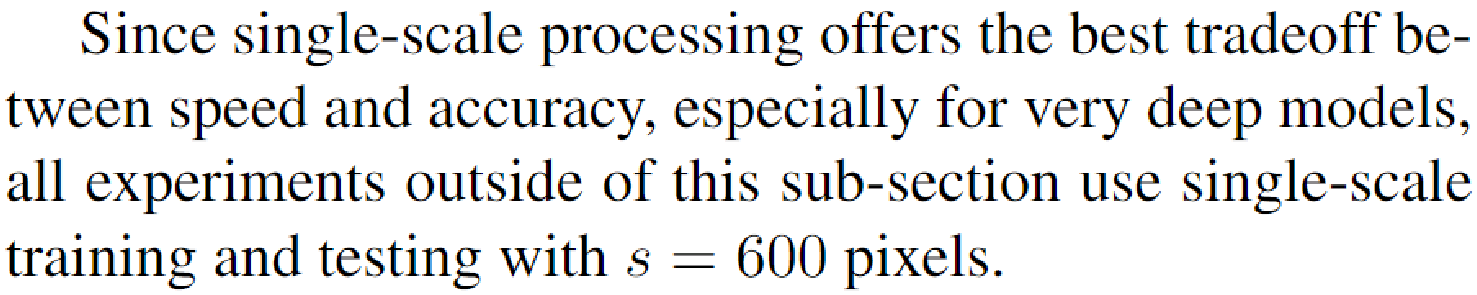
- We experiment with multi-scale training for smaller networks only, due to GPU memory limits.



5.2. Scale invariance: to brute force or finesse?

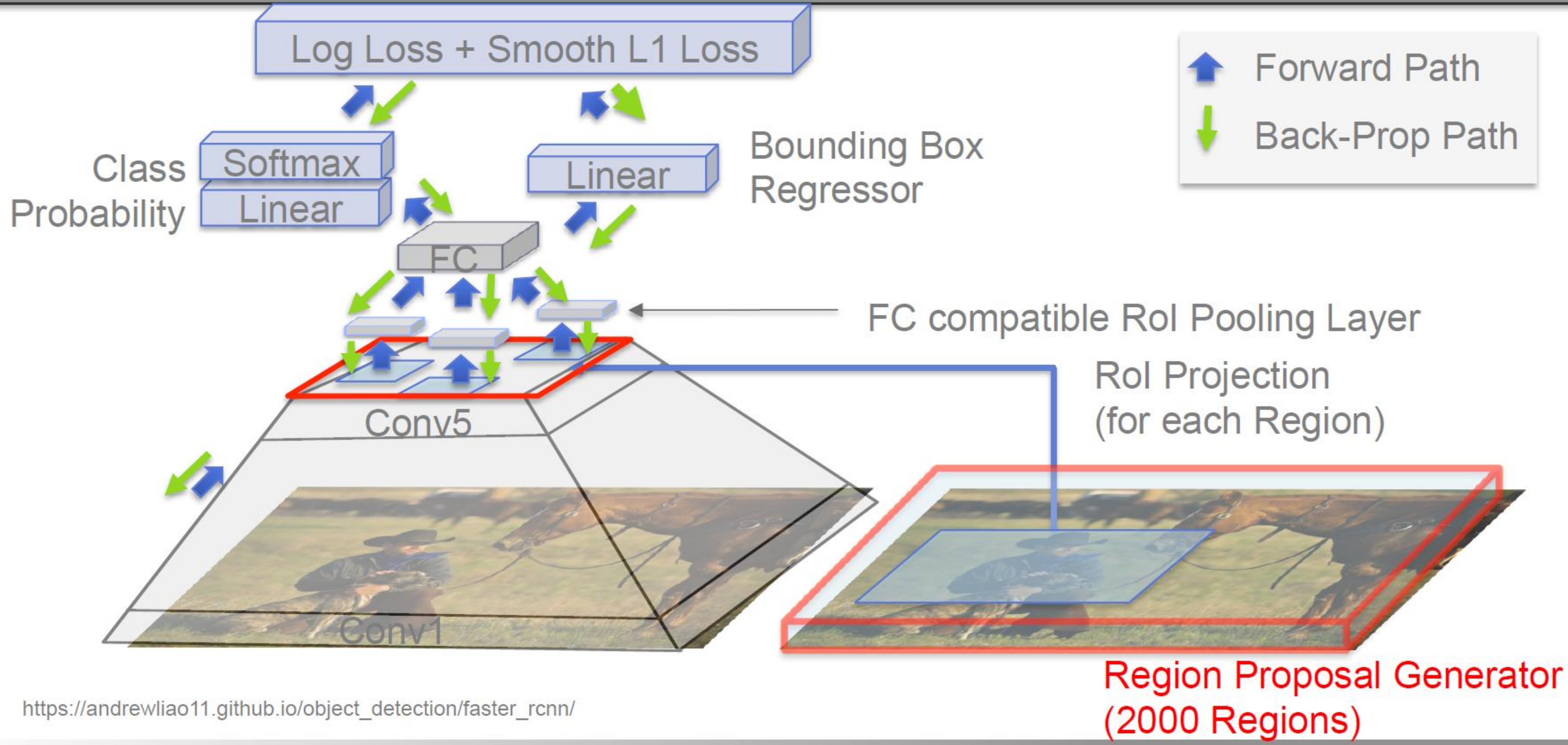
We compare two strategies for achieving scale-invariant object detection: brute-force learning (single scale) and image pyramids (multi-scale). In either case, we define the scale s of an image to be the length of its *shortest* side.

실험 진행



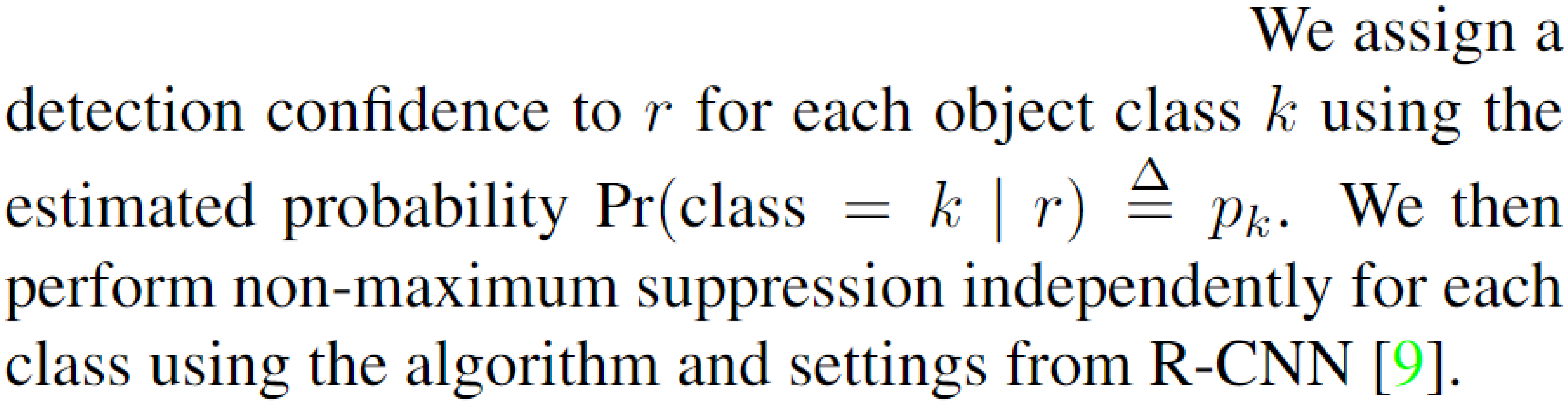
Since single-scale processing offers the best tradeoff between speed and accuracy, especially for very deep models, all experiments outside of this sub-section use single-scale training and testing with $s = 600$ pixels.

Fast R-CNN – Forward and Back-Prop Paths Using Multi-Class Loss


Log Loss + Smooth L1 Loss
Forward Path
Back-Prop Path
Class Probability
Softmax
Linear
Linear
Bounding Box Regressor
FC
FC compatible RoI Pooling Layer
RoI Projection (for each Region)
Conv5
Conv1
Region Proposal Generator (2000 Regions)

https://andrewliao11.github.io/object_detection/faster_rcnn/

3. Fast R-CNN detection



We assign a detection confidence to r for each object class k using the estimated probability $\Pr(\text{class} = k \mid r) \triangleq p_k$. We then perform non-maximum suppression independently for each class using the algorithm and settings from R-CNN [9].

Fast R-cnn

3.1 Truncated SVD for faster detection

- for detection the number of RoIs to process is large and nearly half of the forward pass time is spent computing the fully connected layers.

Fast R-cnn

3.1 Truncated SVD for faster detection

- for detection the number of RoIs to process is large and nearly half of the forward pass time is spent computing the fully connected layers.
- Large fully connected layers are easily accelerated by compressing them with truncated SVD

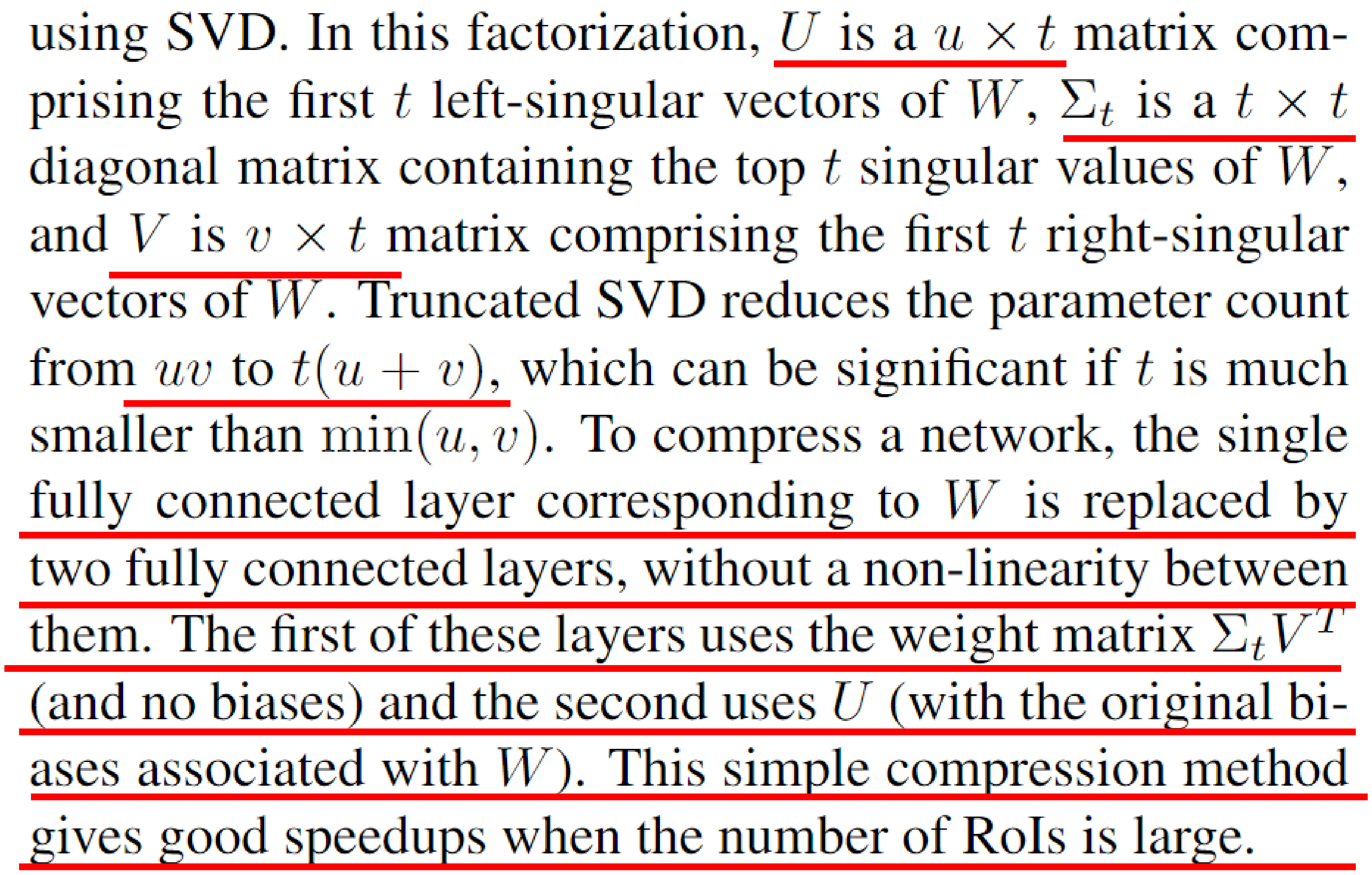
Fast R-cnn

3.1 Truncated SVD for faster detection

- for detection the number of RoIs to process is large and nearly half of the forward pass time is spent computing the fully connected layers.
- Large fully connected layers are easily accelerated by compressing them with truncated SVD
- In this technique, a layer parameterized by the u x v weight matrix W is approximately factorized as $W \approx U\Sigma_t V^T$sing SVD.

Fast R-cnn

3.1 Truncated SVD for faster detection



using SVD. In this factorization, U is a $u \times t$ matrix comprising the first t left-singular vectors of W, Σ_t is a $t \times t$ diagonal matrix containing the top t singular values of W, and V is $v \times t$ matrix comprising the first t right-singular vectors of W. Truncated SVD reduces the parameter count from uv to $t(u + v)$, which can be significant if t is much smaller than $\min(u, v)$. To compress a network, the single fully connected layer corresponding to W is replaced by two fully connected layers, without a non-linearity between them. The first of these layers uses the weight matrix $\Sigma_t V^T$ (and no biases) and the second uses U (with the original biases associated with W). This simple compression method gives good speedups when the number of RoIs is large.

Fast R-cnn

3.1 Truncated SVD for faster detection

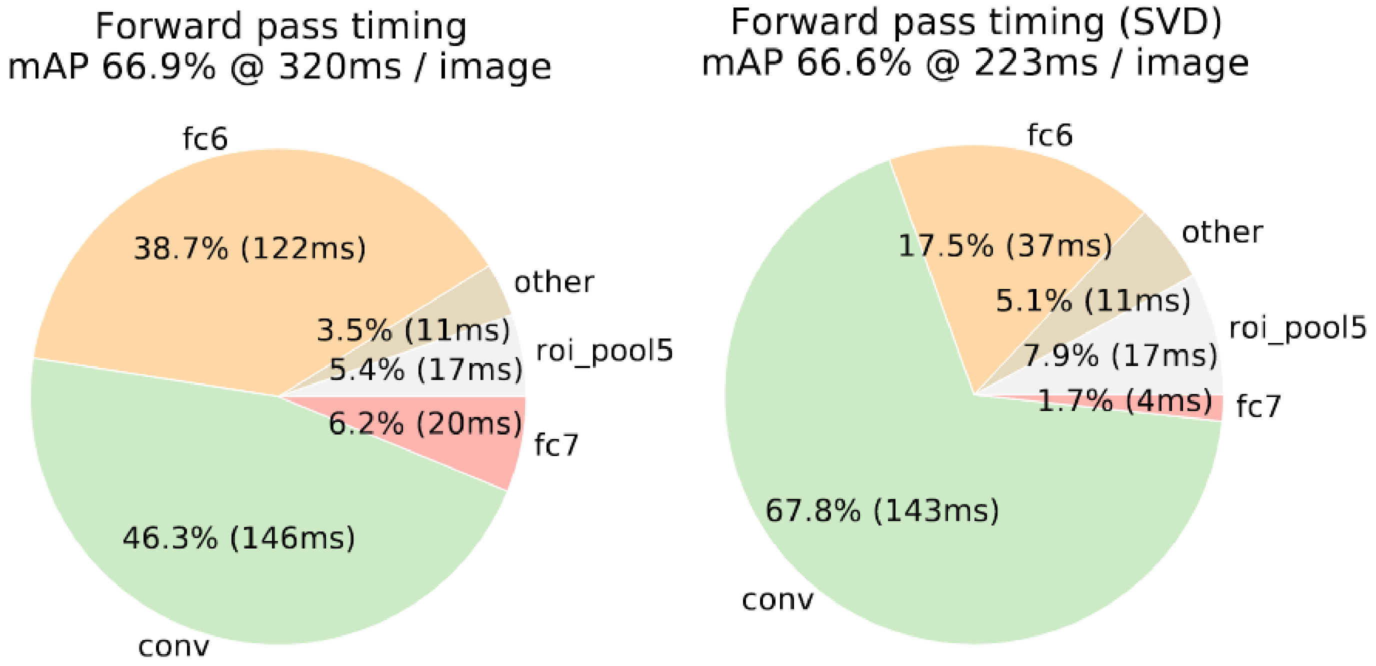
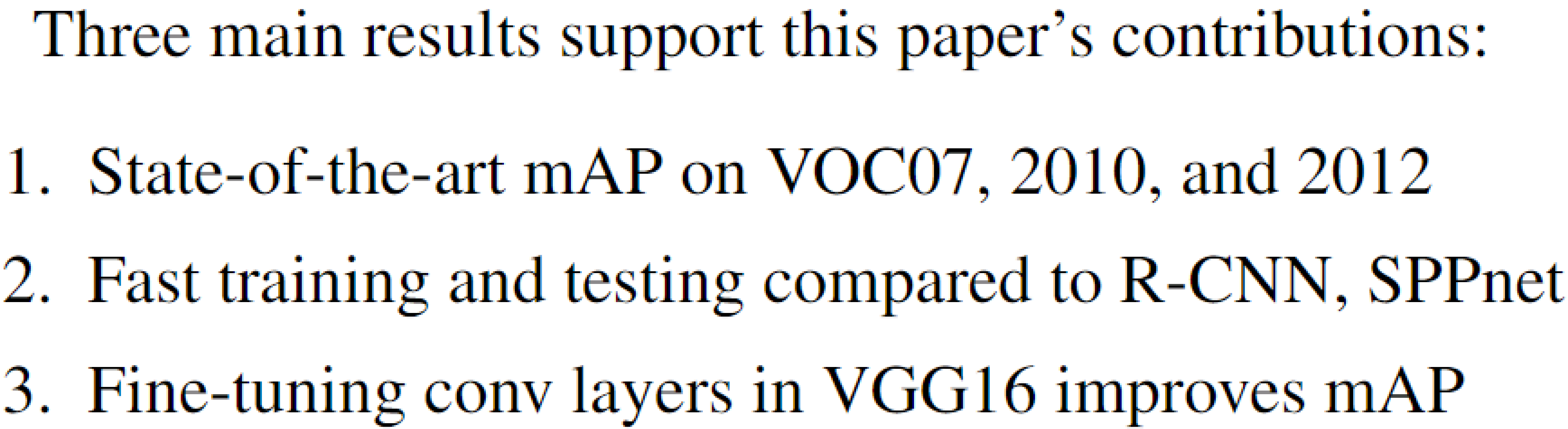


Figure 2. Timing for VGG16 before and after truncated SVD. Before SVD, fully connected layers fc6 and fc7 take 45% of the time.

Fast R-cnn

4. Main results



Three main results support this paper's contributions:

1. State-of-the-art mAP on VOC07, 2010, and 2012
2. Fast training and testing compared to R-CNN, SPPnet
3. Fine-tuning conv layers in VGG16 improves mAP

Thank you!

Fast R-CNN