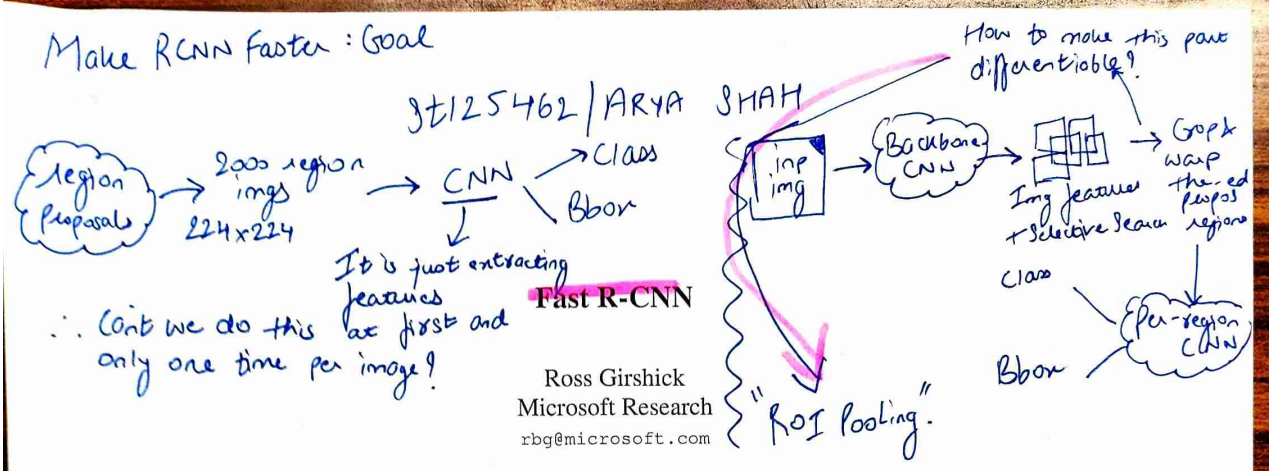

Abstract

This paper proposes a Fast Region-based Convolutional Network method (Fast R-CNN) *for object detection. Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9× faster than R-CNN, is 213× faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3× faster, tests 10× faster, and is more accurate. Fast R-CNN is implemented in Python and C++ (using Caffe) and is available under the open-source MIT License at* `https://github.com/rbgirshick/fast-rcnn`.

1. Introduction

Recently, deep ConvNets [14, 16] have significantly improved image classification [14] and object detection [9, 19] accuracy. Compared to image classification, object detection is a more challenging task that requires more complex methods to solve. Due to this complexity, current approaches (*e.g.*, [9, 11, 19, 25]) train models in multi-stage pipelines that are slow and inelegant.

Complexity arises because detection requires the accurate localization of objects, creating two primary challenges. First, numerous candidate object locations (often called "proposals") must be processed. Second, these candidates provide only rough localization that must be refined to achieve precise localization. Solutions to these problems often compromise speed, accuracy, or simplicity.

In this paper, we streamline the training process for state-of-the-art ConvNet-based object detectors [9, 11]. 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 [20]) 9× faster than R-CNN [9] and 3× faster than SPPnet [11]. At runtime, the detection network processes images in 0.3s (excluding object proposal time) while achieving top accuracy on PASCAL VOC 2012 [7] with a mAP of 66% (vs. 62% for R-CNN).[1]

1.1. R-CNN and SPPnet

The Region-based Convolutional Network method (R-CNN) [9] achieves excellent object detection accuracy by using a deep ConvNet to classify object proposals. R-CNN, however, has notable drawbacks:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and bounding-box regressor training, features are extracted from each object proposal in each image and written to disk. With very deep networks, such as VGG16, this process takes 2.5 GPU-days for the 5k images of the VOC07 trainval set. These features require hundreds of gigabytes of storage.
3. **Object detection is slow.** At test-time, features are extracted from each object proposal in each test image. Detection with VGG16 takes 47s / image (on a GPU).

R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation. Spatial pyramid pooling networks (SPPnets) [11] were proposed to speed up R-CNN by sharing computation. The SPPnet method computes a convolutional feature map for the entire input image and then classifies each object proposal using a feature vector extracted from the shared feature map. Features are extracted for a proposal by max-pooling the portion of the feature map inside the proposal into a fixed-size output (*e.g.*, 6 × 6). Multiple output sizes are pooled and then concatenated as in spatial pyramid pooling [15]. SPPnet accelerates R-CNN by 10 to 100× at test time. Training time is also reduced by 3× due to faster proposal feature extraction.

[1] All timings use one Nvidia K40 GPU overclocked to 875 MHz.

How to train such model? → Initializing from pre-trained Networks → fine tuning for detection

SPPnet also has notable drawbacks. Like R-CNN, training is a multi-stage pipeline that involves extracting features, fine-tuning a network with log loss, training SVMs, and finally fitting bounding-box regressors. Features are also written to disk. But unlike R-CNN, the fine-tuning algorithm proposed in [11] cannot update the convolutional layers that precede the spatial pyramid pooling. Unsurprisingly, this limitation (fixed convolutional layers) limits the accuracy of very deep networks.

1.2. Contributions

We propose a new training algorithm that fixes the disadvantages of R-CNN and SPPnet, while improving on their speed and accuracy. We call this method *Fast R-CNN* because it's comparatively fast to train and test. The Fast R-CNN method has several advantages:

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

Fast R-CNN is written in Python and C++ (Caffe [13]) and is available under the open-source MIT License at `https://github.com/rbgirshick/fast-rcnn`.

2. Fast R-CNN architecture and training

Fig. 1 illustrates the Fast R-CNN architecture. 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. Each set of 4 values encodes refined bounding-box positions for one of the K classes.

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 $H \times W$ (*e.g.*, 7×7), where H and W are layer hyper-parameters that are independent of any particular RoI. 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).

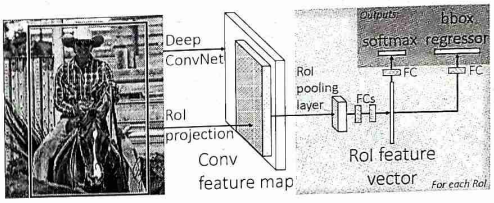

Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

RoI max pooling works by dividing the $h \times w$ RoI window into an $H \times W$ grid of sub-windows of approximate size $h/H \times w/W$ and then max-pooling the values in each sub-window into the corresponding output grid cell. Pooling is applied independently to each feature map channel, as in standard max pooling. The RoI layer is simply the special-case of the spatial pyramid pooling layer used in SPPnets [11] in which there is only one pyramid level. We use the pooling sub-window calculation given in [11].

2.2. Initializing from pre-trained networks

We experiment with three pre-trained ImageNet [4] networks, each with five max pooling layers and between five and thirteen conv layers (see Section 4.1 for network details). 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 described earlier (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.

2.3. Fine-tuning for detection

Training all network weights with back-propagation is an important capability of Fast R-CNN. First, let's elucidate why 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. The inefficiency

two sibling o/p layers
1. $p = (p_0, \dots p_u)$ over k+1 categories
2. $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$

Ground Truths: u, v

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

- $L_{cls}(p, u) = -\log p_u$ ($L_{CE} = -\sum_i p_i^* \log p_i$; p_i^* true class, p_i predicted class)
- $[u \geq 1]$ evaluates to 1 when $u \geq 1$, 0 otherwise.
- $L_{loc}(t^u, v) = \sum_{i \in \{x,y,w,h\}} \text{smooth}_{L_1}(t^u_i - v_i)$

stems from the fact that each RoI may have a very large receptive field, often spanning the entire input image. Since the forward pass must process the entire receptive field, the training inputs are large (often the entire image).

We propose a more efficient training method that takes advantage of feature sharing during training. 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. Making N small decreases mini-batch computation. For example, when using $N = 2$ and $R = 128$, the proposed training scheme is roughly 64× faster than sampling one RoI from 128 different images (*i.e.*, the R-CNN and SPPnet strategy).

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.

In addition to hierarchical sampling, Fast R-CNN uses a streamlined 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 [9, 11]. The components of this procedure (the loss, mini-batch sampling strategy, back-propagation through RoI pooling layers, and SGD hyper-parameters) are described below.

classification problem

Multi-task loss. A Fast R-CNN network has two sibling output layers. The first outputs a discrete probability distribution (per RoI), $p = (p_0, \dots, p_K)$, over $K + 1$ categories. 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, $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$, for each of the K object classes, indexed by k. We use the parameterization for t^k given in [9], in which t^k specifies a scale-invariant translation and log-space height/width shift relative to an object proposal.

Each training RoI is labeled with a ground-truth class u and a ground-truth bounding-box regression target v. We use a multi-task loss L on each labeled RoI to jointly train for classification and bounding-box regression:

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

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

The second task loss, L_{loc}, is defined over a tuple of true bounding-box regression targets for class u, $v = (v_x, v_y, v_w, v_h)$, and a predicted tuple $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$, 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 $u = 0$. For background RoIs there is no notion of a ground-truth bounding box and hence L_{loc} is ignored. For bounding-box regression, we use the loss

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

in which

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

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.

The hyper-parameter λ in Eq. 1 controls the balance between the two task losses. We normalize the ground-truth regression targets v_i to have zero mean and unit variance. All experiments use $\lambda = 1$.

We note that [6] uses a related loss to train a class-agnostic object proposal network. Different from our approach, [6] advocates for a two-network system that separates localization and classification. OverFeat [19], R-CNN [9], and SPPnet [11] also train classifiers and bounding-box localizers, however these methods use stage-wise training, which we show is suboptimal for Fast R-CNN (Section 5.1).

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. As in [9], 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.* $u \geq 1$. The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval $[0.1, 0.5)$, following [11]. 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 [8]. During training, images are horizontally flipped with probability 0.5. No other data augmentation is used.

Back-propagation through RoI pooling layers. Back-propagation routes derivatives through the RoI pooling layer. For clarity, we assume only one image per mini-batch ($N = 1$), though the extension to $N > 1$ is straightforward because the forward pass treats all images independently.

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 $i^*(r, j) = \text{argmax}_{i' \in \mathcal{R}(r,j)} x_{i'}$. $\mathcal{R}(r, j)$ is the index

Scale invariance - Model has to detect object regardless of its scale

Brute Force: img → Fast RCNN → o/p

Img Pyramid: Training time 4× ← sorted pyramid { img versions → Fast RCNN → o/p

SVD → We can write any arbitrary matrix → mult of 3 matrices

$W_{m\times n} = [\vec{w}_1\ \vec{w}_2 \dots \vec{w}_n] = U_{m\times m}\,\Sigma_{m\times n}\,V^T_{n\times n}$

$= [\vec{u}_1\ \vec{u}_2 \dots \vec{u}_m] \begin{bmatrix} \sigma_1 & 0 & 0 & \dots \\ 0 & \sigma_2 & \dots & \\ 0 & \dots & \sigma_3 & \dots \end{bmatrix} [\vec{v}_1\ \vec{v}_2 \dots \vec{v}_n]^T$; $\sigma_1 > \sigma_2 > \sigma_n$

Orthogonal mat. ; $|\vec{u}_i| = 1$; $\vec{u}_i \cdot \vec{u}_j = \vec{0}\ (i \neq j)$ → Basis for mat W

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:

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

In words, for each mini-batch RoI r and for each pooling output unit y_{rj}, the partial derivative $\partial L/\partial y_{rj}$ is accumulated if i is the argmax selected for y_{rj} by max pooling. In back-propagation, the partial derivatives $\partial L/\partial y_{rj}$ are already computed by the `backwards` function of the layer on top of the RoI pooling layer.

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.

2.4. Scale invariance

We explore two ways of achieving scale invariant object detection: (1) via "brute force" learning and (2) by using image pyramids. These strategies follow the two approaches in [11]. In the brute-force approach, each image is processed at a pre-defined pixel size during both training and testing. The network must directly learn scale-invariant object detection from the training data.

The multi-scale approach, in contrast, provides approximate scale-invariance to the network through an image pyramid. At test-time, the image pyramid is used to approximately scale-normalize each object proposal. During multi-scale training, we randomly sample a pyramid scale each time an image is sampled, following [11], as a form of data augmentation. We experiment with multi-scale training for smaller networks only, due to GPU memory limits.

3. Fast R-CNN detection

Once a Fast R-CNN network is fine-tuned, detection amounts to little more than running a forward pass (assuming object proposals are pre-computed). The network takes as input an image (or an image pyramid, encoded as a list of images) and a list of R object proposals to score. At test-time, R is typically around 2000, although we will consider cases in which it is larger ($\approx$ 45k). When using an image pyramid, each RoI is assigned to the scale such that the scaled RoI is closest to 224^2 pixels in area [11].

For each test RoI r, the forward pass outputs a class posterior probability distribution p and a set of predicted bounding-box offsets relative to r (each of the K classes gets its own refined bounding-box prediction). 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].

3.1. Truncated SVD for faster detection

For whole-image classification, the time spent computing the fully connected layers is small compared to the conv layers. On the contrary, 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 (see Fig. 2). Large fully connected layers are easily accelerated by compressing them with truncated SVD [5, 23].

In this technique, a layer parameterized by the $u \times v$ weight matrix W is approximately factorized as

$$W \approx U \Sigma_t V^T \quad (5)$$

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.

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

4.1. Experimental setup

Our experiments use three pre-trained ImageNet models that are available online.[2] The first is the CaffeNet (essentially AlexNet [14]) from R-CNN [9]. We alternatively refer

[2] `https://github.com/BVLC/caffe/wiki/Model-Zoo`

3.1 → If σ_i is less than a certain value, then it means the corresponding col. on matrix U is not very important → We can round it to 0!

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [11]†	07 \ diff	73.9	72.3	62.5	51.5	44.4	74.4	73.0	74.4	42.3	73.6	57.7	70.3	74.6	74.3	54.2	34.0	56.4	56.4	67.9	73.5	63.1
R-CNN BB [10]	07	73.4	77.0	63.4	45.4	**44.6**	75.1	78.1	79.8	40.5	73.7	62.2	79.4	78.1	73.1	64.2	**35.6**	66.8	67.2	70.4	**71.1**	66.0
FRCN [ours]	07	74.5	78.3	69.2	53.2	36.6	77.3	78.2	82.0	40.7	72.7	67.9	79.6	79.2	73.0	69.0	30.1	65.4	70.2	75.8	65.8	66.9
FRCN [ours]	07 \ diff	74.6	**79.0**	68.6	57.0	39.3	79.5	**78.6**	81.9	**48.0**	74.0	67.4	80.5	80.7	74.1	69.6	31.8	67.1	68.4	75.3	65.5	68.1
FRCN [ours]	07+12	**77.0**	78.1	**69.3**	**59.4**	38.3	**81.6**	**78.6**	**86.7**	42.8	**78.8**	**68.9**	**84.7**	**82.0**	**76.6**	**69.9**	31.8	**70.1**	**74.8**	**80.4**	70.4	**70.0**

Table 1. **VOC 2007 test** detection average precision (%). All methods use VGG16. Training set key: **07**: VOC07 trainval, **07 \ diff**: **07** without "difficult" examples, **07+12**: union of **07** and VOC12 trainval. †SPPnet results were prepared by the authors of [11].

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	77.7	73.8	62.3	48.8	45.4	67.3	67.0	80.3	41.3	70.8	49.7	79.5	74.7	78.6	64.5	36.0	69.9	55.7	70.4	61.7	63.8
R-CNN BB [10]	12	79.3	72.4	63.1	44.0	44.4	64.6	66.3	84.9	38.8	67.3	48.4	82.3	75.0	76.7	65.7	35.8	66.2	54.8	69.1	58.8	62.9
SegDeepM	12+seg	**82.3**	75.2	67.1	50.7	**49.8**	71.1	69.6	88.2	42.5	71.2	50.0	85.7	76.6	81.8	69.3	**41.5**	**71.9**	62.2	73.2	**64.6**	67.2
FRCN [ours]	12	80.1	74.4	67.7	49.4	41.4	74.2	68.8	87.8	41.9	70.1	50.2	86.1	77.3	81.1	70.4	33.3	67.0	63.3	77.2	60.0	66.1
FRCN [ours]	07++12	82.0	**77.8**	**71.6**	**55.3**	42.4	**77.3**	**71.7**	**89.3**	**44.5**	**72.1**	**53.7**	**87.7**	**80.0**	**82.5**	**72.7**	36.6	68.7	**65.4**	**81.1**	62.7	**68.8**

Table 2. **VOC 2010 test** detection average precision (%). BabyLearning uses a network based on [17]. All other methods use VGG16. Training set key: **12**: VOC12 trainval, **Prop.**: proprietary dataset, **12+seg**: **12** with segmentation annotations, **07++12**: union of VOC07 trainval, VOC07 test, and VOC12 trainval.

method	train set	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
BabyLearning	Prop.	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6	63.2
NUS_NIN_c2000	Unk.	80.2	73.8	61.9	43.7	**43.0**	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	**38.6**	**68.3**	58.0	68.7	63.3	63.8
R-CNN BB [10]	12	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3	62.4
FRCN [ours]	12	80.3	74.7	66.9	46.9	37.7	73.9	68.6	87.7	41.7	71.1	51.1	86.0	77.8	79.8	69.8	32.1	65.5	63.8	76.4	61.7	65.7
FRCN [ours]	07++12	**82.3**	**78.4**	**70.8**	**52.3**	38.7	**77.8**	**71.6**	**89.3**	**44.2**	**73.0**	**55.0**	**87.5**	**80.5**	**80.8**	**72.0**	35.1	**68.3**	**65.7**	**80.4**	**64.2**	**68.4**

Table 3. **VOC 2012 test** detection average precision (%). BabyLearning and NUS_NIN_c2000 use networks based on [17]. All other methods use VGG16. Training set key: see Table 2, **Unk.**: unknown.

to this CaffeNet as model **S**, for "small." The second network is VGG_CNN_M_1024 from [3], which has the same depth as **S**, but is wider. We call this network model **M**, for "medium." The final network is the very deep VGG16 model from [20]. Since this model is the largest, we call it model **L**. In this section, all experiments use *single-scale* training and testing ($s = 600$; see Section 5.2 for details).

4.2. VOC 2010 and 2012 results

On these datasets, we compare Fast R-CNN (*FRCN*, for short) against the top methods on the `comp4` (outside data) track from the public leaderboard (Table 2, Table 3).[3] For the NUS_NIN_c2000 and BabyLearning methods, there are no associated publications at this time and we could not find exact information on the ConvNet architectures used; they are variants of the Network-in-Network design [17]. All other methods are initialized from the same pre-trained VGG16 network.

Fast R-CNN achieves the top result on VOC12 with a mAP of 65.7% (and 68.4% with extra data). It is also two orders of magnitude faster than the other methods, which are all based on the "slow" R-CNN pipeline. On VOC10, SegDeepM [25] achieves a higher mAP than Fast R-CNN (67.2% vs. 66.1%). SegDeepM is trained on VOC12 trainval plus segmentation annotations; it is designed to boost R-CNN accuracy by using a Markov random field to reason over R-CNN detections and segmentations from the O_2P [1] semantic-segmentation method. Fast R-CNN can be swapped into SegDeepM in place of R-CNN, which may lead to better results. When using the enlarged 07++12 training set (see Table 2 caption), Fast R-CNN's mAP increases to 68.8%, surpassing SegDeepM.

4.3. VOC 2007 results

On VOC07, we compare Fast R-CNN to R-CNN and SPPnet. All methods start from the same pre-trained VGG16 network and use bounding-box regression. The VGG16 SPPnet results were computed by the authors of [11]. SPPnet uses five scales during both training and testing. The improvement of Fast R-CNN over SPPnet illustrates that even though Fast R-CNN uses single-scale training and testing, fine-tuning the conv layers provides a large improvement in mAP (from 63.1% to 66.9%). R-CNN achieves a mAP of 66.0%. As a minor point, SPPnet was trained *without* examples marked as "difficult" in PASCAL. Removing these examples improves Fast R-CNN mAP to 68.1%. All other experiments use "difficult" examples.

[3] `http://host.robots.ox.ac.uk:8080/leaderboard` (accessed April 18, 2015)

4.4. Training and testing time

Fast training and testing times are our second main result. Table 4 compares training time (hours), testing rate (seconds per image), and mAP on VOC07 between Fast R-CNN, R-CNN, and SPPnet. For VGG16, Fast R-CNN processes images 146× faster than R-CNN without truncated SVD and 213× faster with it. Training time is reduced by 9×, from 84 hours to 9.5. Compared to SPPnet, Fast R-CNN trains VGG16 2.7× faster (in 9.5 vs. 25.5 hours) and tests 7× faster without truncated SVD or 10× faster with it. Fast R-CNN also eliminates hundreds of gigabytes of disk storage, because it does not cache features.

	Fast R-CNN			R-CNN			SPPnet
	S	**M**	**L**	**S**	**M**	**L**	†**L**
train time (h)	**1.2**	2.0	9.5	22	28	84	25
train speedup	**18.3**×	14.0×	8.8×	1×	1×	1×	3.4×
test rate (s/im)	0.10	0.15	0.32	9.8	12.1	47.0	2.3
▷ with SVD	**0.06**	0.08	0.22	-	-	-	-
test speedup	98×	80×	146×	1×	1×	1×	20×
▷ with SVD	169×	150×	**213**×	-	-	-	-
VOC07 mAP	57.1	59.2	**66.9**	58.5	60.2	66.0	63.1
▷ with SVD	56.5	58.7	66.6	-	-	-	-

Table 4. Runtime comparison between the same models in Fast R-CNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales specified in [11]. †Timing provided by the authors of [11]. Times were measured on an Nvidia K40 GPU.

Truncated SVD. Truncated SVD can reduce detection time by more than 30% with only a small (0.3 percentage point) drop in mAP and without needing to perform additional fine-tuning after model compression. Fig. 2 illustrates how using the top 1024 singular values from the 25088 × 4096 matrix in VGG16's fc6 layer and the top 256 singular values from the 4096 × 4096 fc7 layer reduces runtime with little loss in mAP. Further speed-ups are possible with smaller drops in mAP if one fine-tunes again after compression.

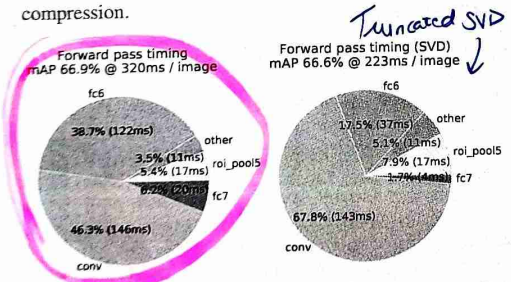

Figure 2. Timing for VGG16 before and after truncated SVD. Before SVD, fully connected layers fc6 and fc7 take 45% of the time.

4.5. Which layers to fine-tune?

For the less deep networks considered in the SPPnet paper [11], fine-tuning only the fully connected layers appeared to be sufficient for good accuracy. We hypothesized that this result would not hold for very deep networks. To validate that fine-tuning the conv layers is important for VGG16, we use Fast R-CNN to fine-tune, but *freeze* the thirteen conv layers so that only the fully connected layers learn. This ablation emulates single-scale SPPnet training and *decreases mAP from 66.9% to 61.4%* (Table 5). This experiment verifies our hypothesis: training through the RoI pooling layer is important for very deep nets.

	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

Table 5. Effect of restricting which layers are fine-tuned for VGG16. Fine-tuning ≥ fc6 emulates the SPPnet training algorithm [11], but using a single scale. SPPnet **L** results were obtained using five scales, at a significant (7×) speed cost.

Does this mean that *all* conv layers should be fine-tuned? In short, *no*. In the smaller networks (**S** and **M**) we find that conv1 is generic and task independent (a well-known fact [14]). Allowing conv1 to learn, or not, has no meaningful effect on mAP. For VGG16, we found it only necessary to update layers from conv3_1 and up (9 of the 13 conv layers). This observation is pragmatic: (1) updating from conv2_1 slows training by 1.3× (12.5 vs. 9.5 hours) compared to learning from conv3_1; and (2) updating from conv1_1 over-runs GPU memory. The difference in mAP when learning from conv2_1 up was only +0.3 points (Table 5, last column). All Fast R-CNN results in this paper using VGG16 fine-tune layers conv3_1 and up; all experiments with models **S** and **M** fine-tune layers conv2 and up.

5. Design evaluation

We conducted experiments to understand how Fast R-CNN compares to R-CNN and SPPnet, as well as to evaluate design decisions. Following best practices, we performed these experiments on the PASCAL VOC07 dataset.

5.1. Does multi-task training help?

Multi-task training is convenient because it avoids managing a pipeline of sequentially-trained tasks. But it also has the potential to improve results because the tasks influence each other through a shared representation (the ConvNet) [2]. Does multi-task training improve object detection accuracy in Fast R-CNN?

To test this question, we train baseline networks that use only the classification loss, L_{cls}, in Eq. 1 (*i.e.*, setting

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
stage-wise training?			✓				✓				✓	
test-time bbox reg?			✓	✓			✓	✓			✓	✓
VOC07 mAP	52.2	53.3	54.6	**57.1**	54.7	55.5	56.6	**59.2**	62.6	63.4	64.0	**66.9**

Table 6. Multi-task training (forth column per group) improves mAP over piecewise training (third column per group).

$\lambda = 0$). These baselines are printed for models **S**, **M**, and **L** in the first column of each group in Table 6. Note that these models *do not* have bounding-box regressors. Next (second column per group), we take networks that were trained with the multi-task loss (Eq. 1, $\lambda = 1$), but we *disable* bounding-box regression at test time. This isolates the networks' classification accuracy and allows an apples-to-apples comparison with the baseline networks.

Across all three networks we observe that multi-task training improves pure classification accuracy relative to training for classification alone. The improvement ranges from $+0.8$ to $+1.1$ mAP points, showing a consistent positive effect from multi-task learning.

Finally, we take the baseline models (trained with only the classification loss), tack on the bounding-box regression layer, and train them with L_{loc} while keeping all other network parameters frozen. The third column in each group shows the results of this *stage-wise* training scheme: mAP improves over column one, but stage-wise training underperforms multi-task training (forth column per group).

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.

All single-scale experiments use $s = 600$ pixels; s may be less than 600 for some images as we cap the longest image side at 1000 pixels and maintain the image's aspect ratio. These values were selected so that VGG16 fits in GPU memory during fine-tuning. The smaller models are not memory bound and can benefit from larger values of s; however, optimizing s for each model is not our main concern. We note that PASCAL images are 384×473 pixels on average and thus the single-scale setting typically upsamples images by a factor of 1.6. The average effective stride at the RoI pooling layer is thus ≈ 10 pixels.

In the multi-scale setting, we use the same five scales specified in [11] ($s \in \{480, 576, 688, 864, 1200\}$) to facilitate comparison with SPPnet. However, we cap the longest side at 2000 pixels to avoid exceeding GPU memory.

Table 7 shows models **S** and **M** when trained and tested with either one or five scales. Perhaps the most surprising result in [11] was that single-scale detection performs almost as well as multi-scale detection. Our findings con-

	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**

Table 7. Multi-scale vs. single scale. SPPnet **ZF** (similar to model **S**) results are from [11]. Larger networks with a single-scale offer the best speed / accuracy tradeoff. (**L** cannot use multi-scale in our implementation due to GPU memory constraints.)

firm their result: deep ConvNets are adept at directly learning scale invariance. The multi-scale approach offers only a small increase in mAP at a large cost in compute time (Table 7). In the case of VGG16 (model **L**), we are limited to using a single scale by implementation details. Yet it achieves a mAP of 66.9%, which is slightly higher than the 66.0% reported for R-CNN [10], even though R-CNN uses "infinite" scales in the sense that each proposal is warped to a canonical size.

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.

5.3. Do we need more training data?

A good object detector should improve when supplied with more training data. Zhu *et al.* [24] found that DPM [8] mAP saturates after only a few hundred to thousand training examples. Here we augment the VOC07 trainval set with the VOC12 trainval set, roughly tripling the number of images to 16.5k, to evaluate Fast R-CNN. Enlarging the training set improves mAP on VOC07 test from 66.9% to 70.0% (Table 1). When training on this dataset we use 60k mini-batch iterations instead of 40k.

We perform similar experiments for VOC10 and 2012, for which we construct a dataset of 21.5k images from the union of VOC07 trainval, test, and VOC12 trainval. When training on this dataset, we use 100k SGD iterations and lower the learning rate by $0.1\times$ each 40k iterations (instead of each 30k). For VOC10 and 2012, mAP improves from 66.1% to 68.8% and from 65.7% to 68.4%, respectively.

5.4. Do SVMs outperform softmax?

Fast R-CNN uses the softmax classifier learnt during fine-tuning instead of training one-vs-rest linear SVMs

post-hoc, as was done in R-CNN and SPPnet. To understand the impact of this choice, we implemented post-hoc SVM training with hard negative mining in Fast R-CNN. We use the same training algorithm and hyper-parameters as in R-CNN.

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**

Table 8. Fast R-CNN with softmax vs. SVM (VOC07 mAP).

Table 8 shows softmax slightly outperforming SVM for all three networks, by +0.1 to +0.8 mAP points. This effect is small, but it demonstrates that "one-shot" fine-tuning is sufficient compared to previous multi-stage training approaches. We note that softmax, unlike one-vs-rest SVMs, introduces competition between classes when scoring a RoI.

5.5. Are more proposals always better?

There are (broadly) two types of object detectors: those that use a *sparse* set of object proposals (*e.g.*, selective search [21]) and those that use a *dense* set (*e.g.*, DPM [8]). Classifying sparse proposals is a type of *cascade* [22] in which the proposal mechanism first rejects a vast number of candidates leaving the classifier with a small set to evaluate. This cascade improves detection accuracy when applied to DPM detections [21]. We find evidence that the proposal-classifier cascade also improves Fast R-CNN accuracy.

Using selective search's *quality mode*, we sweep from 1k to 10k proposals per image, each time *re-training* and *re-testing* model **M**. If proposals serve a purely computational role, increasing the number of proposals per image should not harm mAP.

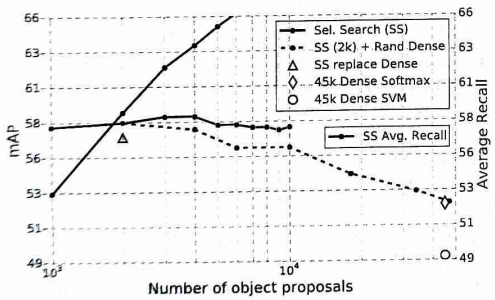

Figure 3. VOC07 test mAP and AR for various proposal schemes.

We find that mAP rises and then falls slightly as the proposal count increases (Fig. 3, solid blue line). This experiment shows that swamping the deep classifier with more proposals does not help, and even slightly hurts, accuracy.

This result is difficult to predict without actually running the experiment. The state-of-the-art for measuring object proposal quality is Average Recall (AR) [12]. AR correlates well with mAP for several proposal methods using R-CNN, *when using a fixed number of proposals per image*. Fig. 3 shows that AR (solid red line) does not correlate well with mAP as the number of proposals per image is varied. AR must be used with care; higher AR due to more proposals does not imply that mAP will increase. Fortunately, training and testing with model **M** takes less than 2.5 hours. Fast R-CNN thus enables efficient, direct evaluation of object proposal mAP, which is preferable to proxy metrics.

We also investigate Fast R-CNN when using *densely* generated boxes (over scale, position, and aspect ratio), at a rate of about 45k boxes / image. This dense set is rich enough that when each selective search box is replaced by its closest (in IoU) dense box, mAP drops only 1 point (to 57.7%, Fig. 3, blue triangle).

The statistics of the dense boxes differ from those of selective search boxes. Starting with 2k selective search boxes, we test mAP when *adding* a random sample of $1000 \times \{2, 4, 6, 8, 10, 32, 45\}$ dense boxes. For each experiment we re-train and re-test model **M**. When these dense boxes are added, mAP falls more strongly than when adding more selective search boxes, eventually reaching 53.0%.

We also train and test Fast R-CNN using *only* dense boxes (45k / image). This setting yields a mAP of 52.9% (blue diamond). Finally, we check if SVMs with hard negative mining are needed to cope with the dense box distribution. SVMs do even worse: 49.3% (blue circle).

5.6. Preliminary MS COCO results

We applied Fast R-CNN (with VGG16) to the MS COCO dataset [18] to establish a preliminary baseline. We trained on the 80k image training set for 240k iterations and evaluated on the "test-dev" set using the evaluation server. The PASCAL-style mAP is 35.9%; the new COCO-style AP, which also averages over IoU thresholds, is 19.7%.

6. Conclusion

This paper proposes Fast R-CNN, a clean and fast update to R-CNN and SPPnet. In addition to reporting state-of-the-art detection results, we present detailed experiments that we hope provide new insights. Of particular note, sparse object proposals appear to improve detector quality. This issue was too costly (in time) to probe in the past, but becomes practical with Fast R-CNN. Of course, there may exist yet undiscovered techniques that allow dense boxes to perform as well as sparse proposals. Such methods, if developed, may help further accelerate object detection.

Acknowledgements. I thank Kaiming He, Larry Zitnick, and Piotr Dollár for helpful discussions and encouragement.