arXiv:1504.08083v1 [cs.CV] 30 Apr 2015

Fast R-CNN

Ross Girshick
Microsoft Research
`rbg@microsoft.com`

Abstract

This paper proposes Fast R-CNN, *a clean and fast framework for object detection. Compared to traditional R-CNN, and its accelerated version SPPnet, Fast R-CNN trains networks using a multi-task loss in a single training stage. The multi-task loss simplifies learning and improves detection accuracy. Unlike SPPnet, all network layers can be updated during fine-tuning. We show that this difference has practical ramifications for very deep networks, such as VGG16, where mAP suffers when only the fully-connected layers are updated. Compared to "slow" R-CNN, Fast R-CNN is 9× faster at training VGG16 for detection, 213× faster at test-time, and achieves a significantly 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++ and is available under the open-source MIT License at* `https://github.com/rbgirshick/fast-rcnn`.

1. Introduction

The use of deep ConvNets [15, 17] for image classification [13] and object detection [9, 18] has led to significant gains in accuracy. However, compared to image classifiation, the greater difficulty of object detection has resulted in approaches of greater complexity. These methods (*e.g.*, [9, 10, 18, 24]) train networks in multi-stage pipelines, which makes training difficult and time consuming.

Detection requires the accurate localization of objects, creating two primary challenges. First, numerous candidate object locations need to be evaluated. In order to make detection efficient, the convolutional features need to be shared between locations [10, 18]. Unfortunately, the approach of [10] restricts the back-propagation of errors through the network during training, potentially limiting accuracy. Second, the candidate object locations need to be refined to achieve high accuracy. Previous refinement processes [9, 18] have been trained in a separate learning stage.

In this paper, we significantly simplify the learning process for state-of-the-art object detectors. A single-stage training algorithm is proposed that simultaneously learns to classify object proposals and refine their spatial locations. In addition, we propose a new method for sharing convolutional features during training that allows for the full back-propagation of errors through the network, leading to increased accuracy. Several improvements are also made to increase runtime efficiency, such as the use of truncated SVD, which is particularly effective for object detection.

The result is a method that can train very deep detection networks in a single stage 9× faster than R-CNN [9] and 3× faster than SPPnet [10]. During runtime, the network processes images in 0.3s (excluding object proposal time) while achieving top accuracy on PASCAL VOC 2012 with a mAP of 66% (vs. 62% for R-CNN).

1.1. R-CNN and SPPnet

R-CNN [9] has excellent object detection accuracy. However, it also has notable drawbacks:

1. **Training is a multi-stage pipeline.** One first fine-tunes a ConvNet for detection using cross-entropy loss. Then, linear SVMs are fit to ConvNet features computed on warped object proposals. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM and regressor training, features are extracted from each warped object proposal in each image and written to disk. With very deep networks, such as VGG16 [19], 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. **Test-time detection is slow.** At test-time, features are extracted from each warped object proposal in each test image. Detection with VGG16 takes 47s / image.

R-CNN is slow because it warps and then processes each object proposal independently. SPPnet [10] was proposed to speed up R-CNN. In SPPnet, proposal warping is shifted to just after the ConvNet's last convolutional layer. This reorganization allows the whole input image to be efficiently processed into a deep feature map (or pyramid), independent of the object proposals. To classify a proposal, the features under the proposal's projection onto the feature map are pooled into a fixed-length feature vector by adaptively-sized pooling bins. He *et al*. [10] advocate concatenating outputs from multiple pooling grids (*e.g*., 6×6, 3×3, 2×2, 1×1), as in spatial pyramid pooling [14]. SPPnet accelerates R-CNN at test-time by 10 to 100×. Training time is also reduced, because feature extraction is fast.

SPPnet also has notable drawbacks. Like R-CNN, training is a multi-stage pipeline involving extracting features, fine-tuning a network with cross-entropy loss, training SVMs, and finally fitting bounding-box regressors. Like R-CNN, features are written to disk. But unlike R-CNN, the fine-tuning algorithm proposed in [10] *can only update the fully-connected layers that follow spatial pyramid pooling*. We hypothesize that this limitation will prevent very deep networks, like VGG16, from reaching their full potential.

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* (*FRCN*, for short) because it's fast at training and testing. Fast R-CNN has the following advantages:

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

Fast R-CNN is written in Python and C++ (Caffe [12]) and is available under the open-source MIT License at `https://github.com/rbgirshick/fast-rcnn`. Given R-CNN's growing use, *Fast R-CNN* will accelerate numerous research efforts.

2. Fast R-CNN training

Following traditional R-CNN, we start from a ConvNet initialized with discriminative pre-training for ImageNet [4] classification. We consider architectures that have several convolutional (conv) and max pooling layers, followed by a region of interest (RoI) pooling layer, and then several fully-connected (fc) layers. We modify these pre-trained networks so they terminate at two sibling layers, one that outputs softmax probability estimates over the K object classes plus a special "background" class and another layer that outputs four real-valued numbers for each of the K object classes. These $4K$ values encode refined bounding boxes for each class. Fast R-CNN is illustrated in Figure 1.

2.1. The RoI pooling layer

The region of interest (RoI) pooling layer is a simplified version of the spatial pyramid pooling used in SPPnet [10], in which our "pyramid" has only a single level. A RoI pooling layer takes as input N feature maps and a list of R regions of interest; typically $R \gg N$. The N feature maps are supplied by the last conv layer of the network and each is a multi-dimensional array of size $H \times W \times C$, with H rows, W columns, and C channels. Each RoI is a tuple (n, r, c, h, w) that specifies a feature map index $n \in \{0, \ldots, N-1\}$, the RoI's top-left location (r, c), and its height and width (h, w). RoI pooling layers output max-pooled feature maps with spatial extent $H' \times W'$ and the original C channels ($H' \leq H$ and $W' \leq W$).

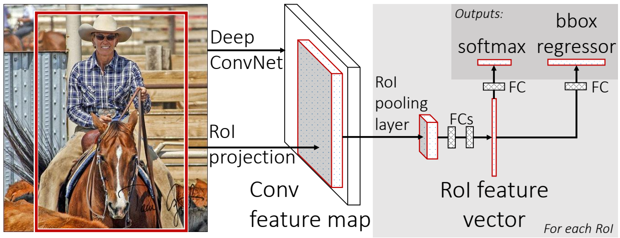

Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully-convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully-connected layers (FCs). The network has two output vectors per RoI: softmax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

For each of the R RoIs, the RoI pooling layer performs max pooling over $H'W'$ output bins. The bin sizes ($\approx h/H' \times w/W'$) are adaptively set such that they tile the $h \times w$ rectangle in the indexed feature map. We use the adaptive bin-size calculation given in [10].

2.2. Using pre-trained networks

We use three pre-trained ImageNet networks, each with five max pooling layers and between five and thirteen conv layers (see Section 4.1). When a pre-trained network initializes Fast R-CNN, 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 final 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 bounding-box regressors).

Third, the network is modified to take two data inputs: a batch of N images and a list of R RoIs. The batch size and number of RoIs can change dynamically from batch to batch, as can the resolution of the input images: all conv layers reshape on-the-fly in proportion to the input resolution [21].

2.3. Fine-tuning for detection

In [10], a 3-layer softmax classifier is fine-tuned for detection. The classifier operates on fixed-length feature vectors that are formed by spatial pyramid pooling (SPP) applied to pre-computed convolutional feature maps of whole images. Since the convolutional features are computed offline, the fine-tuning procedure cannot back-propagate errors to parameters below the SPP layer. For the very deep VGG16 model (which is not used in [10]), this means *the first 13 layers will remain fixed at their initialization; only the 3-layer classifier will be updated.*

This limitation exists so that SPPnets can be trained with stochastic gradient descent (SGD) using the *RoI-centric* sampling procedure developed for R-CNN. In RoI-centric sampling, training RoIs are sampled uniformly from the set of all RoIs in all images, and thus each SGD mini-batch comprises samples from a large number of different images.

Removing this limitation requires back-propagating gradients through the SPP layer. Back-propagation, in turn, requires computing the conv layers over each RoIs' receptive field, which is often the entire input image. This computation is too slow and requires too much memory to be feasible. We propose to a more efficient strategy using *image-centric* sampling. In Fast R-CNN, mini-batches are sampled hierarchically, first sampling images and then RoIs within those images. RoI's from the same image share computation and memory, making training efficient.

In addition to image-centric sampling, Fast R-CNN trains networks in 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, 10]. The components of this procedure—*the loss, mini-batch sampling strategy, back-propagation through RoI pooling layers, and SGD hyper-parameters*—are described below.

Multi-task loss. A Fast R-CNN network has two sibling output layers. The first layer outputs a discrete probability distribution (per RoI), $p = (p_0, \ldots, p_K)$, 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, $t^k = \left(t^k_{\mathrm{x}}, t^k_{\mathrm{y}}, t^k_{\mathrm{w}}, t^k_{\mathrm{h}}\right)$, 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.

We use a multi-task loss L to train networks jointly for classification and bounding-box regression:

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

in which k^* is a true class label and $L_{\mathrm{cls}}(p, k^*) = -\log p_{k^*}$ is the standard cross-entropy/log loss.

The second task loss, L_{loc}, is defined over a tuple of true bounding-box regression targets for class k^*, $t^* = (t^*_{\mathrm{x}}, t^*_{\mathrm{y}}, t^*_{\mathrm{w}}, t^*_{\mathrm{h}})$, and a predicted tuple $t = (t_{\mathrm{x}}, t_{\mathrm{y}}, t_{\mathrm{w}}, t_{\mathrm{h}})$, again for class k^* (we omit the superscript k^* for notational brevity). The Iverson bracket indicator function $[k^* \geq 1]$ evaluates to 1 when $k^* \geq 1$ and 0 otherwise. By convention the special background class is labeled $k^* = 0$. For background RoIs there is no notion of a ground-truth bounding box and hence L_{loc} is ignored in that case. For bounding-box regression, we use the loss

$$L_{\mathrm{loc}}(t, t^*) = \sum_{i \in \{\mathrm{x},\mathrm{y},\mathrm{w},\mathrm{h}\}} \mathrm{smooth}_{L_1}(t_i, t^*_i), \quad (2)$$

in which

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

is a smoothed L_1 loss that is less sensitive to outliers than the L_2 loss used in R-CNN. When the regression targets are unbounded, training with L_2 loss requires significant 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 t^* to have zero mean and unit variance. Under this normalization, $\lambda = 1$ works well and is used in all experiments.

We note that [6] uses a related loss to train a class agnostic object proposal system. Different from our approach, [6] advocates for a two-stage / two-network system that first localizes and then recognizes objects. OverFeat [18], R-CNN [9], and SPPnet [10] also train classifiers and bounding-box localizers, however these methods use *piecewise* 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 ground-truth object class, *i.e.* $k^* \geq 1$. The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval $[0.1, 0.5)$, following [10]. These are the background examples and are labeled with $k^* = 0$. During training each sampled image is horizontally flipped with probability 0.5. No other data augmentation is used.

Back-propagation through RoI pooling layers. Fast R-CNN mini-batches start from whole images and hence contain all information needed to back-propagate derivatives from the loss function to the image. Back-propagation routes derivatives through the RoI pooling layer, as described below.

During the forward pass, an *input batch* of $N = 2$ images is expanded by the RoI pooling layer into an *output batch* of size $R = 128$. The multi-task loss L is averaged over the R outputs. During back-propagation, derivatives flow through the RoI pooling layer. The RoI pooling layer's `backwards` function computes the partial derivative of the loss function with respect to each input variable x by summing over all RoIs that max-pooled x in the forward pass:

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

In words, for each mini-batch RoI r and for each of the $H'W'$ pooled output units y in r, the partial derivative $\partial L / \partial y$ is accumulated if x was the argmax assigned to y during pooling (*i.e.*, if y "pooled" x, as given by the indicator function). In back-propagation, the partial derivatives $\partial L / \partial y$ 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 randomly from a zero-mean Gaussian distribution with standard deviations 0.01 and 0.001, respectively. All layers use a per-layer learning rate of 1 for weights and 2 for biases (following standard practice) 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 term with weight 0.9 and weight decay factor of 0.0005 are used in all experiments.

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 [10]. In the brute-force approach, each image is processed at a canonical scale 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. The detector is trained to fire only when it "sees" its target at its optimal scale. During multi-scale training, we randomly sample a scale each time an image is sampled (following [10]). 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 a single-scale image (or an image pyramid) 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). For single-scale detection, all RoIs index the single image in the input batch (*i.e.*, $n = 0$). In the multi-scale case, the input batch is an image pyramid. Here, each RoI specifies a pyramid level l (by setting $n = l$) and a rectangle within that level.

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 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 [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 Figure 2, left). Recognizing this fact, we look to truncated SVD, which offers a simple way to compress fully-connected layers [5, 22]. In this technique, a fully-connected layer parameterized by the $u \times v$ weight matrix W is approximately factorized as

$$W \approx U \Sigma_t V^T \tag{5}$$

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 for detection without the need for additional fine-tuning.

method	data	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	persn	plant	sheep	sofa	train	tv	mAP
SPPnet BB [10]†	07	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 [8]	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 [our]	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+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, *including SPPnet*. Training data key: "07": VOC07 trainval, "07+12": VOC07 trainval union with VOC12 trainval. †SPPnet VGG16 results were prepared by the authors of [10].

method	data	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 [8]	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 an unspecified network. All other methods use VGG16. Training data key: "12": VOC12 trainval, "Prop.": proprietary dataset, "12+seg": VOC12 trainval plus segmentation annotations, "07++12": VOC07 trainval and test union with VOC12 trainval. Results: †`http://goo.gl/f1vtls`, ‡`http://goo.gl/kyZcnW`

method	data	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	?	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 [8]	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 unspecified networks. All other methods use VGG16. Training data key: see Table 2, "?": unknown. Results: †`http://goo.gl/weNq2Z`, ‡`http://goo.gl/o1tZ10`

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 times compared to R-CNN and SPPnet
3. Fine-tuning conv layers in VGG16 is important

4.1. Experimental setup

Our experiments use three pre-trained ImageNet models that are available online.[1] The first is the CaffeNet (essentially AlexNet [13]) from R-CNN [9]. We alternatively refer 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 [19]. Since this model is the largest, we call it model **L**. In this section, all experiments use *single-scale* training and testing (see Section 5.2 for details).

4.2. VOC 2010 and 2012 results

On these datasets, we compare against the top methods on the `comp4` (outside data) track from the public leaderboard.[2] For the NUS_NIN_c2000 and BabyLearning methods, there are no associated publications at this time and we could not find information on the ConvNet architectures used; they are unspecified variants of the Network-in-Network design [16]. 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 traditional R-CNN pipelines. On VOC10, SegDeepM [24] 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 an MRF 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 traditional R-CNN, which may lead to better results. When using the enlarged "07++12" training data (discussed in Section 5.3), Fast R-CNN's mAP increases to 68.8%, surpassing SegDeepM.

---

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

[2] `http://host.robots.ox.ac.uk:8080/leaderboard`. Accessed April 18, 2015.

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 [10] and provided in personal communication. 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%). Traditional R-CNN achieves a mAP of 66.0%. These results are pragmatically valuable given how much faster and easier Fast R-CNN is to train and test, which we discuss next.

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 processing 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 reduces disk storage by hundreds of gigabytes, 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 [10]. †Timing provided by the authors of [10]. 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. Figure 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.

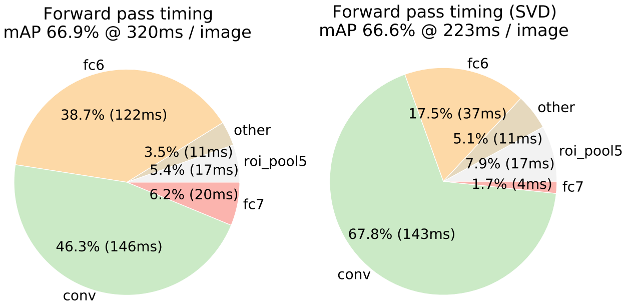

Figure 2. Timing for VGG16 before and after truncated SVD. Before SVD, fully-connected layers fc6 and fc7 take 45% of the time.

4.5. Which layers to fine-tune?

For the less deep networks considered in the SPPnet paper [10], fine-tuning only the fully-connected layers appeared to be sufficient for good accuracy. We hypothesized earlier in this paper 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 the SPPnet training restriction 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**
	$\geq$ fc6	$\geq$ conv3_1	$\geq$ conv2_1	$\geq$ 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 $\geq$ fc6 emulates the SPPnet training algorithm [10], but using a single scale. SPPnet **L** results were obtained using five scales, at a significant 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 [13]). 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.

	S				M				L			
multi-task training?		✓		✓		✓		✓		✓		✓
piecewise 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).

5. Design evaluation

We conducted a set of basic experiments to understand how Fast R-CNN compares to traditional R-CNN and SPPnet, as well as to evaluate design decisions. We follow best practices and perform these control 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 $\lambda = 0$). 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 baselines.

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 *piecewise* training scheme: mAP improves over column one, but piecewise 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. 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 typically 300×500 pixels and thus the single-scale setting upsamples most images by a factor of two. The effective stride at the RoI pooling layer is ≈ 8 pixels for our models.

In the multi-scale setting, we use the same five scales specified in [10] ($s \in \{480, 576, 688, 864, 1200\}$) to facilitate comparison with SPPnet. However, we cap the longest side at 2000 pixels to avoid over-running GPU memory.

	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 [10]. 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.)

Table 7 shows models **S** and **M** when trained and tested with one or five scales. Perhaps the most surprising result in [10] was that single-scale detection performs almost as well as multi-scale detection. Our findings confirm 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 traditional R-CNN [8], even though R-CNN uses "infinite" scales in the sense that each object 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?

An important property of an object detector is its ability to improve as more training data is supplied. In [23], Zhu *et al.* studied this issue in depth for DPMs [7] and found that they saturate after only a few hundred to thousand training examples. Here we perform a simple experiment were we augment the VOC07 trainval set with the VOC12 trainval set, roughly tripling the number of images to 16.5k. The expanded 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 performed similar experiments for VOC10 and 2012, for which we construct an enlarged dataset of 21.5k images from VOC07 trainval and test union with VOC12 trainval. When training on this dataset we use 100k mini-batch 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. Fast R-CNN, and presumably other deep ConvNet-based detectors, clearly have room to grow as more training data become available.

5.4. Does SVM 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 traditional R-CNN.

method	classifier	S	M	L
R-CNN [9, 8]	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 two types of object detectors (roughly speaking): those that use a *sparse set* of "object proposals" (*e.g.*, selective search [20]) and those that use a *dense set*, *i.e.* sliding window. Hosang *et al.*'s recent meta-evaluation [11] of sparse proposal methods states "[sparse proposals] may improve detection quality by reducing spurious false positives." After years of use, this basic issue remains untested, likely due to slow detector training and testing. We approach this issue from two angles using Fast R-CNN.

More sparse proposals. 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, and may even improve it.

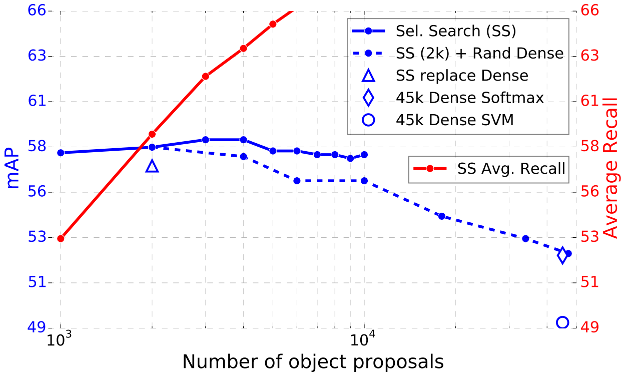

Figure 3. VOC07 test mAP and AR for various proposal schemes.

We find that mAP rises and then falls slightly as the proposal count increases (Figure 3, solid blue line). The state-of-the-art for measuring object proposal quality is Average Recall (AR) [11]. AR correlates well with mAP for several proposal methods using R-CNN, *when using a fixed number of boxes per image*. Figure 3 shows that AR (solid red line) *does not* correlate well with mAP as the number of boxes per image is varied. AR must be used with care; higher AR due to more boxes does not imply that mAP will increase.

Fast R-CNN enables efficient, direct evaluation of object proposal mAP, which is preferable to proxy metrics. Re-training and re-testing with model **M** takes less than 2.5 hours.

Dense proposals. We also investigate Fast R-CNN when using *densely* generated boxes (over scale, position, and aspect ratio), at a rate of about 45k / 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%, Figure 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). We also check if SVMs with hard negative mining are needed to cope with the dense box distribution. SVMs do even worse: 49.3% (blue circle).

Dense vs. sparse. Sparse object proposal methods are currently the speed bottleneck in Fast R-CNN. Replacing them with a dense set of "sliding windows" is attractive, since it is essentially free. Yet, these experiments provide the first evidence that sparse proposals do indeed "improve detection quality by reducing spurious false positives" [11].

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 would like to thank Kaiming He and Larry Zitnick for helpful discussions and encouragement.

References

[1] J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Semantic segmentation with second-order pooling. In *ECCV*, 2012. 5

[2] R. Caruana. Multitask learning. *Machine learning*, 28(1), 1997. 7

[3] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convolutional nets. In *BMVC*, 2014. 5

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 2

[5] E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014. 4

[6] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014. 3

[7] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *TPAMI*, 2010. 7

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013. 5, 7, 8

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. 1, 3, 4, 5, 8

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. 1, 2, 3, 4, 5, 6, 7

[11] J. H. Hosang, R. Benenson, P. Dollár, and B. Schiele. What makes for effective detection proposals? *arXiv preprint arXiv:1502.05082*, 2015. 8, 9

[12] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the ACM International Conf. on Multimedia*, 2014. 2

[13] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In *NIPS*, 2012. 1, 5, 6

[14] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 2

[15] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Comp.*, 1989. 1

[16] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. 5

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. *Parallel Distributed Processing*, 1:318–362, 1986. 1

[18] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks. In *ICLR*, 2014. 1, 3

[19] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1, 5

[20] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IJCV*, 2013. 8

[21] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc. on Vision, Image, and Signal Processing*, 1994. 2

[22] J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *Interspeech*, 2013. 4

[23] X. Zhu, C. Vondrick, D. Ramanan, and C. Fowlkes. Do we need more training data or better models for object detection? In *BMVC*, 2012. 7

[24] Y. Zhu, R. Urtasun, R. Salakhutdinov, and S. Fidler. segDeepM: Exploiting segmentation and context in deep neural networks for object detection. In *CVPR*, 2015. 1, 5