Fast R-CNN

Ross Girshick
Microsoft Research

Abstract

Proposes a Fast Region-based Convolutional Network method (Fast R-CNN) for object detection

Builds on previous work to efficiently classify object proposals using deep convolutional networks

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

Trains the very deep VGG16 network 9 × faster than R-CNN, is 213× faster at test-time

Compared to SPPnet, Fast R-CNN trains VGG16 3 × faster, tests 10 × faster, and is more accurate

content

1. Introduction
    - 1.1. RCNN and SPPnet
    - 1.2. Contributions
2. Fast R-CNN architecture and training
    - 2.1. The RoI pooling layer
    - 2.2. Initializing from pretrained networks
    - 2.3. Finetuning for detection
    - 2.4. Scale invariance
3. Fast R-CNN detection
    - 3.1. Truncated SVD for faster detection
4. Main results
    - 4.1. Experimental setup
    - 4.2. VOC 2010 and 2012 results
    - 4.3. VOC 2007 results
    - 4.4. Training and testing time
    - 4.5. Which layers to finetune?
5. Design evaluation
6. Conclusion

1. Introduction

1. Introduction

Recently, deep ConvNets have significantly improved image classification and object detection accuracy.

Complexity arises because detection requires the accurate localization of objects, creating two primary challenges.

- numerous candidate object locations (often called “proposals”) must be processed
- these candidates provide only rough localization that must be refined to achieve precise localization
- Need: speed, accuracy, or simplicity

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

A very deep detection network (VGG16):**9 faster than R-CNN and 3 faster than SPPnet**

At runtime, the detection 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. Introduction

1.1. RCNN and SPPnet

The Region-based Convolutional Network method (RCNN) 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.

2. Training is expensive in space and time.

3. Object detection is slow.

R-CNN is slow because it performs a ConvNet forward pass for each object proposal, without sharing computation

Spatial pyramid pooling networks (SPPnets) 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.

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

1. Introduction

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 RCNN method has several advantages:

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

2. Training is single-stage, using a multi-task loss

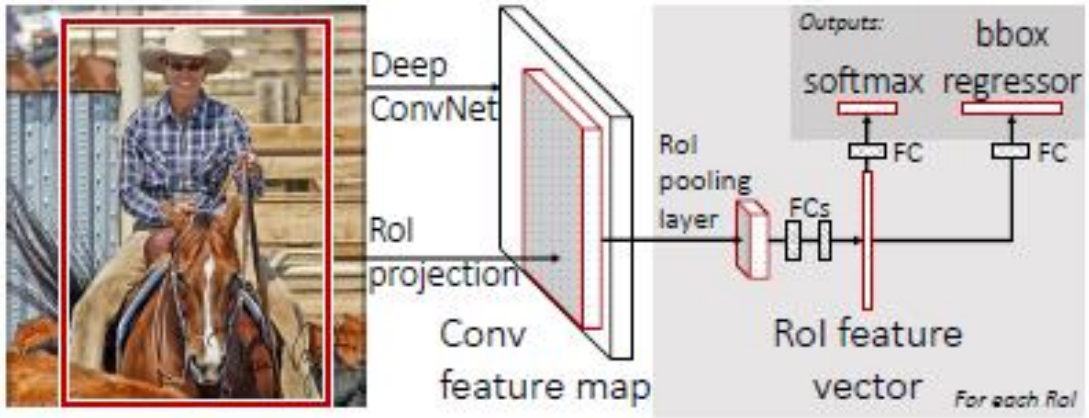
3. Training can update all network layers

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

2. Fast R-CNN architecture and training



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

- 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

2. Fast R-CNN architecture and training

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

- RoI : a rectangular window into a conv feature map.
- Each RoI : 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 w RoI window into an H W grid of sub-windows of approximate size h=H w=W and then 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 [11] in which there is only one pyramid level. We use the pooling sub-window calculation

2. Fast R-CNN architecture and training

2.2. Initializing from pretrained networks

We experiment with three pre-trained ImageNet networks

We experiment with three pre-trained ImageNet networks

Each with: five max pooling layers and between five and thirteen conv layers

- 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. Fast R-CNN architecture and training

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.

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 RCNN training, stochastic gradient descent (SGD) minibatches 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

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

One concern over this strategy :
may cause slow training convergence(RoIs from the same image are correlated)

This concern does not appear to be a practical issue :
we achieve good results with N = 2 and R = 128 using fewer SGD iterations than R-CNN.

Fast R-CNN uses a streamlined training process with one fine-tuning stage that jointly optimizes a softmax classifier and bounding-box regressors

The components of this procedure (the loss, mini-batch sampling strategy, back-propagation through RoI pooling layers, and SGD hyper-parameters) are described below.

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

2.3.1 Multi-task loss.

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

a. First layer: outputs a discrete probability distribution (per RoI), p = (p_0,… … , p_k), over K + 1 categories.
b. Second sibling layer outputs bounding-box regression offsets, t^k =(t_x^k; t_y^k; t_w^k; t_h^k), foreach of the K object classes, indexed by k.

multi-task loss L on each labeled RoI:

L(p,u, t^u,v) = L_{cls}(p,u) + λ [u≥1] L_{loc}(t^u,v)

L_{cls}(p,u) = -logP_u is log loss for true class u.

L_{loc}: defined over a tuple of true bounding-box regression targets

For background RoIs there is no notion of a ground-truth bounding box and hence Lloc is ignored. For bounding-box regression, we use the loss :

$$L_{loc}(t^u,v) = \sum_{i\in\{x,y,w,h\}} Smooth_{L1}(t_i^u, v_i)$$

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

2.3.1 Multi-task loss.

$$smooth_{L1}(x) = \begin{cases} 0.5x^2 & if\ |x| < 1 \\ |x| - 0.5 & otherwise \end{cases}$$

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.

We note that uses a related loss to train a class agnostic object proposal network

Advocates for a two-network system that separates localization and classification.

use stage-wise training, which we show is suboptimal for Fast R-CNN

- OverFeat
- R−CNN
- SPPnet(train classifiers and bounding−box localizers)

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

2.3.2 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 $\begin{cases} mini-batches\ of\ size\ R\ =\ 128, \\ \textit{sampling 64 RoIs from each image} \\ \textit{take 25\% of the RoIs from object proposals} \end{cases}$

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

During training, images are horizontally flipped with probability 0.5. No other data augmentation is used.

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

2.3.3 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.

$x_i \in$ **R** : i-th activation input into the RoI pooling layer

y_{rj}: layer's j-th output from the r- th RoI.

y_{rj} = x_i * (r,j)

$i * (r, j) = argmax_{i' \in R(r,j)} x_{i'}$

R(r, j) is the index set of inputs in the sub-window over which the output unit y_{rj} max pools.

The RoI pooling layer's backwards:

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

2. Fast R-CNN architecture and training

2.3. Fine-tuning for detection

2.3.4 SGD hyper-parameters.

The fully connected layers : Initialized from zero-mean Gaussian distributions
Standard deviations 0:01 and 0:001, respectively

Biases : initialized to 0.

All layers use $\begin{cases} \text{per−layer learning rate of 1 for weights} \\ \text{learning rate of2 for biases} \\ \text{learning rate of} \end{cases}$

Dataset $\begin{cases} VOC07 \\ VOC12 \end{cases}$ run SGD for 30k mini-batch iterations, and then lower the learning rate to 0.0001 and train for another 10k iterations

On larger datasets : run SGD for more iterations

2. Fast R-CNN architecture and training

2.4. Scale invariance

two ways of achieving scale invariant OD
- via “brute force” learning
- using image pyramids

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, as a form of data augmentation.

3. Fast R-CNN detection

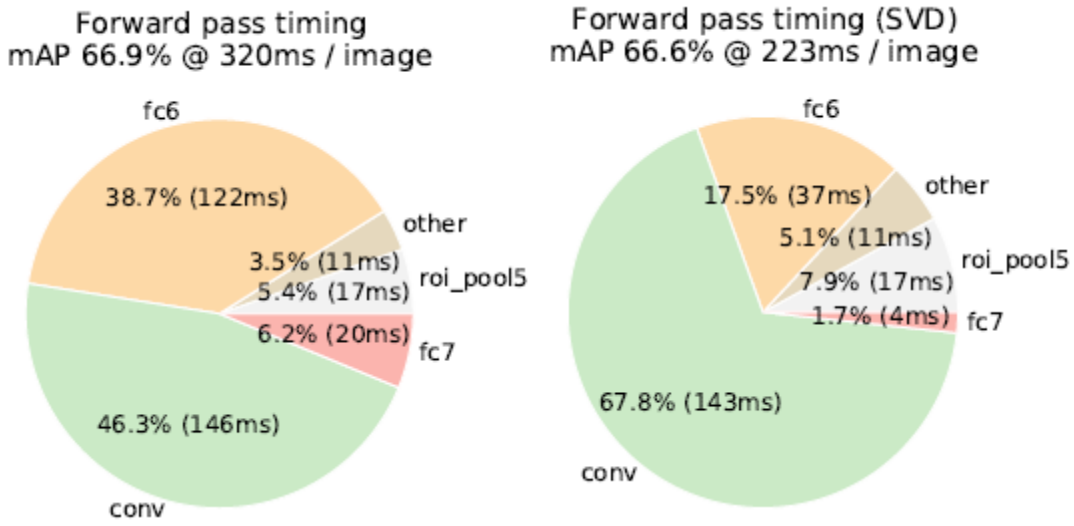
3. Fast R-CNN detection

3.1. Truncated SVD for faster detection

Whole-image classification : the time spent computing the fully connected layers is small compared to the conv layers.

→

Detection the number of RoIs to process is large and nearly half of the forward pass time is spent computing the fully connected layers



3. Fast R-CNN detection

3.1. Truncated SVD for faster detection

Large fully connected layers are easily accelerated by compressing them with truncated SVD

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

$$W \approx U \sum_t V^t$$

U is a u $\times$ t matrix comprising the first t left-singular vectors of W, $\sum_t$ 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.

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.

4. Main results

4. Main results

main results support this paper's contributions
- State−of−the−art mAP on VOC07, 2010, and 2012
- Fast training and testing compared to R−CNN, SPPnet
- Fine−tuning conv layers in VGG16 improves mAP

4.1. Experimental setup

pre-trained ImageNet
- *The first is the CaffeNet (essentially AlexNet) from R−CNN S*
- *The second network is VGG CNN M 1024 , the same depth as S, but wider M*
- *The final network is the very deep VGG16 model* L

All experiments use single-scale training and testing (s = 600,see Section 5.2 for details).

4.2. VOC 2010 and 2012 results

Fast R-CNN achieves the top result on VOC12 with a mAP of 65.7% (and 68.4% with extra data). → Two orders of magnitude faster than the other methods, which are all based on the "slow" R-CNN pipeline.

4. Main results

4.2. VOC 2010 and 2012 results

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 O2P semantic-segmentation method.

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**

BabyLearning uses a network baseline
All other methods use VGG16.

4. Main results

4.3. VOC 2007 results

On VOC07, we compare Fast R-CNN to R-CNN and SPPnet.

All methods $\begin{cases} \textit{start from the same pre-trained VGG16 network} \\ \textit{use bounding-box regression} \\ \text{The VGG16 SPPnet results were computed} \end{cases}$

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%).

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%.

4. Main results

4.4. Training and testing time

Fast training and testing times are our second main result.

For VGG16, Fast R-CNN processes images 146 × faster than R-CNN without truncated SVD and 213 × 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	-	-	-	-

Runtime comparison between the same models in Fast RCNN, R-CNN, and SPPnet. Fast R-CNN uses single-scale mode. SPPnet uses the five scales.

4. Main results

4.4. Training and testing time

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.

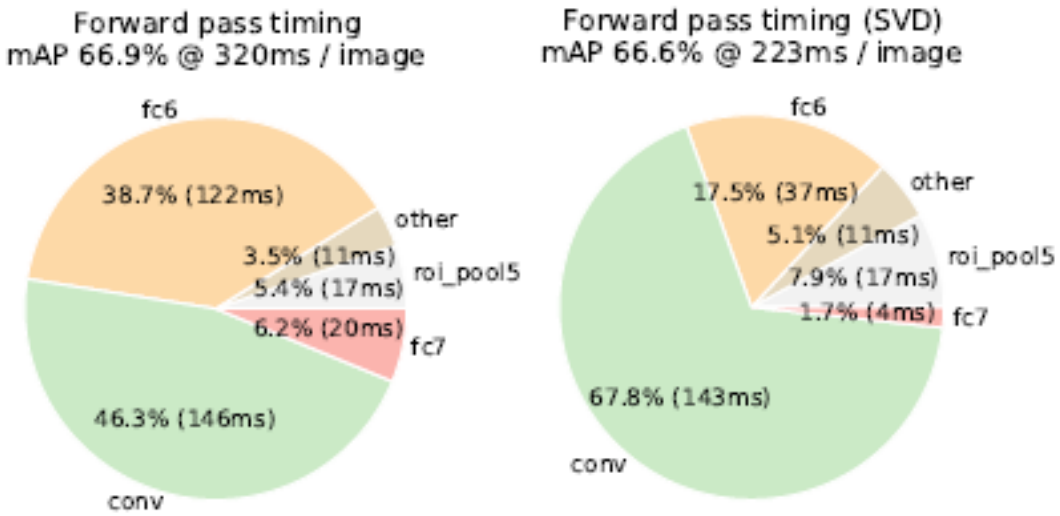


Figure illustrates how using the top 1024 singular values from the 25088 × 4096 matrix in VGG16's fc6 layer and the top 256 singular values from the 4096 × 4096 fc7 layer reduces runtime with little loss in mAP. Further speed-ups are possible with smaller drops in mAP if one fine-tunes again after compression.

4. Main results

4.5. Which layers to finetune?

For the less deep networks considered in the SPPnet paper, 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%

	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

4. Main results

4.5. Which layers to finetune?

Does this mean that all conv layers should be fine-tuned? In short, no.

We hypothesized that this result would not hold for very deep networks.

In the smaller networks (S and M) we find that conv1 is generic and task independent

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:

- *updating from conv2 1 slows training by 1.3 × compared to learning from conv3_1*
- *updating from conv1 1 over−runs GPU memory.*
- *The difference in mAP when learning from conv2 1 up was only +0.3 points*

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

5. Design evaluation

We conducted experiments to understand how Fast RCNN 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 multitask 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

	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**

5. Design evaluation

5.1. Does multitask training help?

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, 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.

all three networks : observe that multi-task training improves pure classification accuracy relative to training for classification alone.

Improvement: anges from +0:8 to +1:1 mAP points

Finally, we take the baseline models (trained with only the classification loss), tack on the bounding-box regression layer, and train them with Lloc while keeping all other network parameters frozen.

5. Design evaluation

5.2. Scale invariance: to brute force or finesse?

compare two strategies for achieving scale-invariant object detection
- brute-force learning (single scale)
- 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.

The average effective stride at the RoI pooling layer is thus $\times$ 10 pixels.

In the multi-scale setting, we use the same five scales ($s \in \{480,576, 688, 864,1200\}$) to facilitate comparison with SPPnet.

However, we cap the longest side at 2000 pixels to avoid exceeding GPU memory.

5. Design evaluation

5.2. Scale invariance: to brute force or finesse?

	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**

It shows models S and M when trained and tested with either one or five scales.

Perhaps the most surprising result 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.

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.

single-scale processing offers the best tradeoff between speed and accuracy, especially for very deep models.

5. Design evaluation

5.3. Do we need more training data?

A good object detector should improve when supplied with more training data.

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× 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. Design evaluation

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 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. Design evaluation

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)and those that use a dense set .

Classifying sparse proposals is a type of cascade in which the proposal mechanism first rejects a vast number of candidates leaving the classifier with a small set to evaluate.

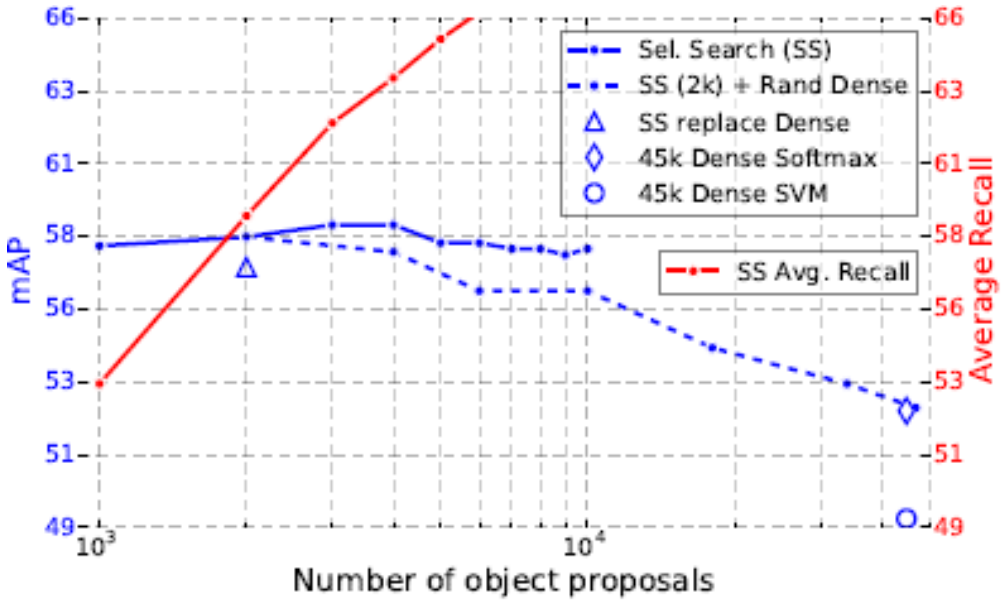
We find evidence that the proposal-classifier cascade also improves Fast R-CNN accuracy.

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

5. Design evaluation

5.5. Are more proposals always better?

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.

5. Design evaluation

5.5. Are more proposals always better?

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).

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.

investigate Fast R-CNN {
when using densely generated boxes
at a rate of about 45k boxes / image.
enough that when each selective search box is replaced by its closest (in IoU) dense box
mAP drops only 1 point (to 57.7%

5. Design evaluation

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

6. Conclusion

This paper proposes Fast R-CNN, a clean and fast update to R-CNN and SPPnet.

In addition to reporting state-of-theart 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.