Fast R-CNN

Ross Girshick Microsoft Research

[rbg@microsoft.com](mailto:rbg@microsoft.com)

# 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 ob- ject proposals using deep convolutional networks. Com- pared to previous work, Fast R-CNN employs several in- novations 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:*](https://github.com/rbgirshick/fast-rcnn)

*x*

*x*

*x x*

[*//github.com/rbgirshick/fast-rcnn*](https://github.com/rbgirshick/fast-rcnn)*.*

# Introduction

Recently, deep ConvNets [[14](#_bookmark40), [16](#_bookmark42)] have signiﬁcantly im- proved image classiﬁcation [[14](#_bookmark40)] and object detection [[9](#_bookmark35), [19](#_bookmark45)] accuracy. Compared to image classiﬁcation, object detec- tion is a more challenging task that requires more com- plex methods to solve. Due to this complexity, current ap- proaches (*e.g*., [[9](#_bookmark35), [11](#_bookmark37), [19](#_bookmark45), [25](#_bookmark51)]) train models in multi-stage pipelines that are slow and inelegant.

Complexity arises because detection requires the ac- curate localization of objects, creating two primary chal- lenges. First, numerous candidate object locations (often called “proposals”) must be processed. Second, these can- didates 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](#_bookmark35), [11](#_bookmark37)]. We pro- pose 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](#_bookmark46)]) 9*x* faster than R-CNN [[9](#_bookmark35)] and 3*x*

while achieving top accuracy on PASCAL VOC 2012 [[7](#_bookmark33)] with a mAP of 66% (vs. 62% for R-CNN).[1](#_bookmark1)

## R-CNN and SPPnet

The Region-based Convolutional Network method (R- CNN) [[9](#_bookmark35)] 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 classi- ﬁer 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 ex- tracted 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 re- quire 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](#_bookmark37)] were pro- posed 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 pro- posal using a feature vector extracted from the shared fea- ture 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 pool- ing [[15](#_bookmark41)]. SPPnet accelerates R-CNN by 10 to 100 at test time. Training time is also reduced by 3 due to faster pro- posal feature extraction.

*x*

*x*

*x*

faster than SPPnet [[11](#_bookmark37)]. At runtime, the detection network

processes images in 0.3s (excluding object proposal time)

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

SPPnet also has notable drawbacks. Like R-CNN, train- ing is a multi-stage pipeline that involves extracting fea- tures, 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 al- gorithm proposed in [[11](#_bookmark37)] cannot update the convolutional layers that precede the spatial pyramid pooling. Unsurpris- ingly, this limitation (ﬁxed convolutional layers) limits the accuracy of very deep networks.

## Contributions

We propose a new training algorithm that ﬁxes the disad- vantages of R-CNN and SPPnet, while improving on their speed and accuracy. We call this method *Fast R-CNN* be- cause 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](#_bookmark39)]) and is available under the open-source MIT Li- cense at [https://github.com/rbgirshick/](https://github.com/rbgirshick/fast-rcnn) [fast-rcnn](https://github.com/rbgirshick/fast-rcnn).

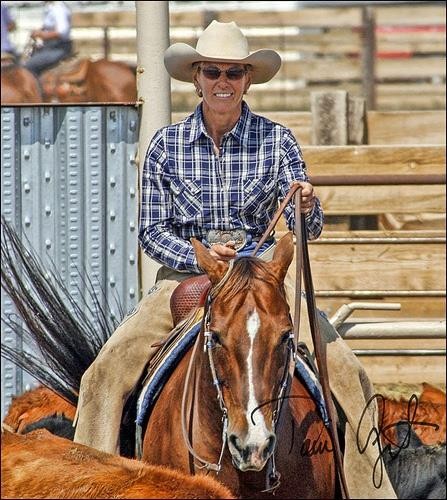
# Fast R-CNN architecture and training

Fig. [1](#_bookmark3) 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 ob- ject proposal a region of interest (*RoI*) pooling layer ex- tracts 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 lay- ers: 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.

## The RoI pooling layer

The RoI pooling layer uses max pooling to convert the features inside any valid region of interest into a small fea- ture map with a ﬁxed spatial extent of *H W* (*e.g*., 7 7), where *H* and *W* are layer hyper-parameters that are inde- pendent 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*).

*x x*



*Outputs:*

bbox

Deep

ConvNet

softmax regressor

RoI

pooling layer

FC

FC

RoI

projection

Conv feature map

FCs

RoI feature

vector *For each RoI*

Figure 1. Fast R-CNN architecture. An input image and multi- ple 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 w* RoI win- dow 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. Pool- ing 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](#_bookmark37)] in which there is only one pyramid level. We use the pooling sub-window calculation given in [[11](#_bookmark37)].

*x*

*x*

*x*

## Initializing from pre-trained networks

We experiment with three pre-trained ImageNet [[4](#_bookmark30)] net- works, each with ﬁve max pooling layers and between ﬁve and thirteen conv layers (see Section [4.1](#_bookmark8) for network de- tails). 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 soft-

max (which were trained for 1000-way ImageNet classiﬁ- cation) are replaced with the two sibling layers described earlier (a fully connected layer and softmax over *K* + 1 cat- egories 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.

## 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

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).

bounding box and hence *L*loc is ignored. For bounding-box regression, we use the loss

*L*loc(*tu, v*) = smooth*L* (*tu - vi*)*,* (2)

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-

in which

← *i*

*ie{*x*,*y*,*w*,*h*}*

batches are sampled hierarchically, ﬁrst by sampling *N* im- ages and then by sampling *R/N* RoIs from each image. Critically, RoIs from the same image share computation

smooth*L*← (*x*) =

0*.*5*x*2 if *IxI <* 1

*IxI -* 0*.*5 otherwise*,*

,

(3)

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 train- ing convergence because RoIs from the same image are cor- related. This concern does not appear to be a practical issue and we achieve good results with *N* = 2 and *R* = 128

*x*

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 re- gressors, rather than training a softmax classiﬁer, SVMs, and regressors in three separate stages [[9](#_bookmark35), [11](#_bookmark37)]. The compo- nents of this procedure (the loss, mini-batch sampling strat- egy, 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 outputs a discrete probability distri- bution (per RoI), *p* = (*p*0*, . . . , pK*), 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, *tk* = *tk, tk, tk , tk* , for

╱ 、

x

y

w

h

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](#_bookmark5) eliminates this sensitivity.

The hyper-parameter *λ* in Eq. [1](#_bookmark6) controls the balance be- tween the two task losses. We normalize the ground-truth regression targets *vi* to have zero mean and unit variance. All experiments use *λ* = 1.

We note that [[6](#_bookmark32)] uses a related loss to train a class-

agnostic object proposal network. Different from our ap- proach, [[6](#_bookmark32)] advocates for a two-network system that sepa- rates localization and classiﬁcation. OverFeat [[19](#_bookmark45)], R-CNN [[9](#_bookmark35)], and SPPnet [[11](#_bookmark37)] 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](#_bookmark18)).

**Mini-batch sampling.** During ﬁne-tuning, each SGD mini-batch is constructed from *N* = 2 images, chosen uni- formly at random (as is common practice, we actually iter- ate over permutations of the dataset). We use mini-batches of size *R* = 128, sampling 64 RoIs from each image. As in [[9](#_bookmark35)], 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

each of the *K* object classes, indexed by *k*. We use the pa- rameterization for *tk* given in [[9](#_bookmark35)], in which *tk* 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, tu, v*) = *L*cls(*p, u*) + *λ*[*u >* 1]*L*loc(*tu, v*)*,* (1) in which *L*cls(*p, u*) = log *pu* 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 *tu* = (*tu, tu, tu, tu*),

the examples labeled with a foreground object class, *i.e*. *u* 1. The remaining RoIs are sampled from object pro- posals that have a maximum IoU with ground truth in the in- terval [0*.*1*,* 0*.*5), following [[11](#_bookmark37)]. 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](#_bookmark34)]. 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.

*>*

x y w h

again for class *u*. The Iverson bracket indicator function [*u* 1] evaluates to 1 when *u* 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

*> >*

*e*

Let *xi* 十be the *i*-th activation input into the RoI pool-

ing layer and let *yrj* be the layer’s *j*-th output from the *r*- th RoI. The RoI pooling layer computes *yrj* = *xi皿*(*r,j*), in which *i\**(*r, j*) = argmax*i╱eR*(*r,j*) *xi╱* . *R*(*r, j*) is the index

set of inputs in the sub-window over which the output unit *yrj* max pools. A single *xi* may be assigned to several dif- ferent outputs *yrj*.

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

*∂L* = [*i* = *i\**(*r, j*)] *∂L .* (4)

*∂xi*

*∂yrj*

*r*

*j*

In words, for each mini-batch RoI *r* and for each pooling output unit *yrj*, the partial derivative *∂L/∂yrj* is accumu- lated if *i* is the argmax selected for *yrj* by max pooling. In back-propagation, the partial derivatives *∂L/∂yrj* are al- ready 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 stan- dard deviations 0*.*01 and 0*.*001, respectively. Biases are ini- tialized 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.

## Scale invariance

We explore two ways of achieving scale invariant ob- ject detection: (1) via “brute force” learning and (2) by us- ing image pyramids. These strategies follow the two ap- proaches in [[11](#_bookmark37)]. 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 approx- imate scale-invariance to the network through an image pyramid. At test-time, the image pyramid is used to ap- proximately scale-normalize each object proposal. During multi-scale training, we randomly sample a pyramid scale each time an image is sampled, following [[11](#_bookmark37)], as a form of data augmentation. We experiment with multi-scale train- ing for smaller networks only, due to GPU memory limits.

# Fast R-CNN detection

Once a Fast R-CNN network is ﬁne-tuned, detection amounts to little more than running a forward pass (assum- ing object proposals are pre-computed). The network takes

test-time, *R* is typically around 2000, although we will con- sider cases in which it is larger ( 45k). When using an image pyramid, each RoI is assigned to the scale such that the scaled RoI is closest to 2242 pixels in area [[11](#_bookmark37)].

For each test RoI *r*, the forward pass outputs a class

*s*

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(class = *k r*) = *pk*. We then perform non-maximum suppression independently for each class using the algorithm and settings from R-CNN [[9](#_bookmark35)].

∆

*I*

## Truncated SVD for faster detection

For whole-image classiﬁcation, the time spent comput- ing 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](#_bookmark19)). Large fully connected layers are easily accelerated by com- pressing them with truncated SVD [[5](#_bookmark31), [23](#_bookmark49)].

In this technique, a layer parameterized by the *u v*

*x*

weight matrix *W* is approximately factorized as

*W s U* Σ*tV T* (5)

using SVD. In this factorization, *U* is a *u t* matrix com- prising the ﬁrst *t* left-singular vectors of *W* , Σ*t* is a *t t* diagonal matrix containing the top *t* singular values of *W* , and *V* is *v 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 Σ*tV T* (and no biases) and the second uses *U* (with the original bi- ases associated with *W* ). This simple compression method gives good speedups when the number of RoIs is large.

*x*

*x*

*x*

# 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

## Experimental setup

Our experiments use three pre-trained ImageNet models that are available online.[2](#_bookmark9) The ﬁrst is the CaffeNet (essen- tially AlexNet [[14](#_bookmark40)]) from R-CNN [[9](#_bookmark35)]. We alternatively refer

as input an image (or an image pyramid, encoded as a list

of images) and a list of *R* object proposals to score. At

2<https://github.com/BVLC/caffe/wiki/Model-Zoo>

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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](#_bookmark37)]*|*  R-CNN BB [[10](#_bookmark36)] | 07 *\* diff  07 | 73.9  73.4 | 72.3  77.0 | 62.5  63.4 | 51.5  45.4 | 44.4  **44.6** | 74.4  75.1 | 73.0  78.1 | 74.4  79.8 | 42.3  40.5 | 73.6  73.7 | 57.7  62.2 | 70.3  79.4 | 74.6  78.1 | 74.3  73.1 | 54.2  64.2 | 34.0  **35.6** | 56.4  66.8 | 56.4  67.2 | 67.9  70.4 | 73.5  **71.1** | 63.1  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]  FRCN [ours] | 07 *\* diff  07+12 | 74.6  **77.0** | **79.0**  78.1 | 68.6  **69.3** | 57.0  **59.4** | 39.3  38.3 | 79.5  **81.6** | **78.6**  **78.6** | 81.9  **86.7** | **48.0**  42.8 | 74.0  **78.8** | 67.4  **68.9** | 80.5  **84.7** | 80.7  **82.0** | 74.1  **76.6** | 69.6  **69.9** | 31.8  31.8 | 67.1  **70.1** | 68.4  **74.8** | 75.3  **80.4** | 65.5  70.4 | 68.1  **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. *Ⅰ*SPPnet results were prepared by the authors of [[11](#_bookmark37)].

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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](#_bookmark36)] | 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](#_bookmark43)]. 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](#_bookmark36)] | 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](#_bookmark43)]. All other methods use VGG16. Training set key: see Table [2](#_bookmark12), **Unk.**: unknown.

to this CaffeNet as model **S**, for “small.” The second net- work is VGG CNN M 1024 from [[3](#_bookmark29)], 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](#_bookmark46)]. 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](#_bookmark22) for details).

## 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](#_bookmark12), Table [3](#_bookmark13)).[3](#_bookmark14) 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](#_bookmark43)]. 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,

3http://host.robots.ox.ac.uk:8080/leaderboard

(accessed April 18, 2015)

SegDeepM [[25](#_bookmark51)] achieves a higher mAP than Fast R-CNN (67.2% vs. 66.1%). SegDeepM is trained on VOC12 train- val 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

[[1](#_bookmark27)] 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](#_bookmark12) caption), Fast R-CNN’s mAP in- creases to 68.8%, surpassing SegDeepM.

## 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](#_bookmark37)]. SPPnet uses ﬁve scales during both training and test- ing. The improvement of Fast R-CNN over SPPnet illus- trates that even though Fast R-CNN uses single-scale train- ing 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.

## Training and testing time

Fast training and testing times are our second main re- sult. Table [4](#_bookmark16) 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 pro- cesses images 146 faster than R-CNN without truncated SVD and 213 faster with it. Training time is reduced by

*x*

*x*

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.

*x x*

*x*

*x*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Fast R-CNN  **S M L** | | | R-CNN  **S M L** | | | SPPnet  *Ⅰ* **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 |
| S with SVD | **0.06** | 0.08 | 0.22 | - | - | - | - |
| test speedup  S with SVD | 98*×*  169*×* | 80*×*  150*×* | 146*×*  **213***×* | 1*×*  - | 1*×*  - | 1*×*  - | 20*×*  - |
| VOC07 mAP | 57.1 | 59.2 | **66.9** | 58.5 | 60.2 | 66.0 | 63.1 |
| S 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](#_bookmark37)]. *Ⅰ*Timing provided by

the authors of [[11](#_bookmark37)]. 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 percent- age point) drop in mAP and without needing to perform additional ﬁne-tuning after model compression. Fig. [2](#_bookmark19) il- lustrates 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 run- time with little loss in mAP. Further speed-ups are possi- ble with smaller drops in mAP if one ﬁne-tunes again after compression.

*x*

*x*

## Which layers to ﬁne-tune?

For the less deep networks considered in the SPPnet pa- per [[11](#_bookmark37)], ﬁne-tuning only the fully connected layers ap- peared 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](#_bookmark17)). 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**

VOC07 mAP 61.4

test rate (s/im) **0.32**

*竺* fc6 *竺* conv3 1

66.9

**0.32**

*竺* conv2 1

**67.2**

**0.32**

*竺* fc6

63.1

2.3

Table 5. Effect of restricting which layers are ﬁne-tuned for VGG16. Fine-tuning fc6 emulates the SPPnet training algo- rithm [[11](#_bookmark37)], but using a single scale. SPPnet **L** results were ob-

tained using ﬁve scales, at a signiﬁcant (7*×*) speed cost.

*竺*

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](#_bookmark40)]). Allowing conv1 to learn, or not, has no mean- ingful effect on mAP. For VGG16, we found it only nec- essary 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 (Ta- ble [5](#_bookmark17), last column). All Fast R-CNN results in this paper using VGG16 ﬁne-tune layers conv3 1 and up; all experi- ments with models **S** and **M** ﬁne-tune layers conv2 and up.

*x*

# Design evaluation

We conducted experiments to understand how Fast R-

Forward pass timing mAP 66.9% @ 320ms / image

fc6

38.7% (122ms)

other

3.5% (11ms) roi\_pool5

5.4% (17ms)

6.2% (20ms)

Forward pass timing (SVD) mAP 66.6% @ 223ms / image

fc6

17.5% (37ms)

other

5.1% (11ms)

roi\_

7.9% (17ms)

1.7% (4ms) fc7

67.8% (143ms)

conv

pool5

CNN compares to R-CNN and SPPnet, as well as to eval- uate design decisions. Following best practices, we per- formed these experiments on the PASCAL VOC07 dataset.

## Does multi-task training help?

Multi-task training is convenient because it avoids man-

46.3% (146ms)

conv

fc7

aging 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](#_bookmark28)]. Does multi-task training improve object detection ac-

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

curacy in Fast R-CNN?

To test this question, we train baseline networks that use only the classiﬁcation loss, *L*cls, in Eq. [1](#_bookmark6) (*i.e*., setting

multi-task training? stage-wise training? test-time bbox reg? VOC07 mAP

**S**

√ √

√

√ √

52.2 53.3 54.6 **57.1**

**M**

√ √

√

√ √

54.7 55.5 56.6 **59.2**

**L**

√ √

√

√ √

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).

*λ* = 0). These baselines are printed for models **S**, **M**, and **L** in the ﬁrst column of each group in Table [6](#_bookmark21). 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](#_bookmark6), *λ* = 1), but we *disable* bounding- box regression at test time. This isolates the networks’ clas- siﬁcation accuracy and allows an apples-to-apples compar- ison 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 posi- tive 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 *Lloc* while keeping all other net- work 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 under- performs multi-task training (forth column per group).

## Scale invariance: to brute force or ﬁnesse?

We compare two strategies for achieving scale-invariant object detection: brute-force learning (single scale) and im- age 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 im- age side at 1000 pixels and maintain the image’s aspect ra- tio. 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*; how- ever, optimizing *s* for each model is not our main concern. We note that PASCAL images are 384 473 pixels on av- erage 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.

*x*

*s*

In the multi-scale setting, we use the same ﬁve scales speciﬁed in [[11](#_bookmark37)] (*s* 480*,* 576*,* 688*,* 864*,* 1200 ) to facili- tate comparison with SPPnet. However, we cap the longest side at 2000 pixels to avoid exceeding GPU memory.

*e { }*

Table [7](#_bookmark23) shows models **S** and **M** when trained and tested with either one or ﬁve scales. Perhaps the most surpris- ing result in [[11](#_bookmark37)] was that single-scale detection performs almost as well as multi-scale detection. Our ﬁndings con-

SPPnet **ZF**

**S**

**M**

scales 1 5 1 5 1 5

test rate (s/im) 0.14 0.38 **0.10** 0.39 0.15 0.64

VOC07 mAP 58.0 59.2 57.1 58.4 59.2 60.7

**L**

1

0.32

**66.9**

Table 7. Multi-scale vs. single scale. SPPnet **ZF** (similar to model

**S**) results are from [[11](#_bookmark37)]. 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 learn- ing scale invariance. The multi-scale approach offers only a small increase in mAP at a large cost in compute time (Table [7](#_bookmark23)). In the case of VGG16 (model **L**), we are lim- ited 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](#_bookmark36)], 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 be- tween 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.

## Do we need more training data?

A good object detector should improve when supplied with more training data. Zhu *et al*. [[24](#_bookmark50)] found that DPM [[8](#_bookmark34)] mAP saturates after only a few hundred to thousand train- ing 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](#_bookmark11)). 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.

*x*

## 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 under- stand 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](#_bookmark35), [10](#_bookmark36)] | SVM | **58.5** | **60.2** | 66.0 |
| FRCN [ours]  FRCN [ours] | SVM  softmax | 56.3  57.1 | 58.7  59.2 | 66.8  **66.9** |

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

Table [8](#_bookmark25) shows softmax slightly outperforming SVM for all three networks, by +0*.*1 to +0*.*8 mAP points. This ef- fect is small, but it demonstrates that “one-shot” ﬁne-tuning is sufﬁcient compared to previous multi-stage training ap- proaches. We note that softmax, unlike one-vs-rest SVMs, introduces competition between classes when scoring a RoI.

## 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](#_bookmark47)]) and those that use a *dense* set (*e.g*., DPM [[8](#_bookmark34)]). Classifying sparse proposals is a type of *cascade* [[22](#_bookmark48)] 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](#_bookmark47)]. 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.

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](#_bookmark38)]. AR correlates well with mAP for several proposal methods using R-CNN, *when using a ﬁxed number of proposals per image*. Fig. [3](#_bookmark26) 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](#_bookmark26), 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 2*,* 4*,* 6*,* 8*,* 10*,* 32*,* 45 dense boxes. For each exper- iment we re-train and re-test model **M**. When these dense

*x { }*

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 nega- tive mining are needed to cope with the dense box distribu- tion. SVMs do even worse: 49.3% (blue circle).

## Preliminary MS COCO results

We applied Fast R-CNN (with VGG16) to the MS COCO dataset [[18](#_bookmark44)] to establish a preliminary baseline. We trained on the 80k image training set for 240k iterations and

66

63

61

58

mAP

56

53

51

49 3 4

10 10

Number of object proposals

66 evaluated on the “test-dev” set using the evaluation server.

The PASCAL-style mAP is 35.9%; the new COCO-style

Sel. Search (SS)

SS (2k) + Rand Dense SS replace Dense

45k Dense Softmax 45k Dense SVM

SS Avg. Recall

63 AP, which also averages over IoU thresholds, is 19.7%.

61

Average Recall

# Conclusion

58

This paper proposes Fast R-CNN, a clean and fast update to R-CNN and SPPnet. In addition to reporting state-of-the-

56

53 art detection results, we present detailed experiments that

51 we hope provide new insights. Of particular note, sparse object proposals appear to improve detector quality. This

49 issue was too costly (in time) to probe in the past, but be-

comes practical with Fast R-CNN. Of course, there may ex-

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

We ﬁnd that mAP rises and then falls slightly as the pro- posal count increases (Fig. [3](#_bookmark26), solid blue line). This exper- iment shows that swamping the deep classiﬁer with more proposals does not help, and even slightly hurts, accuracy.

ist yet undiscovered techniques that allow dense boxes to perform as well as sparse proposals. Such methods, if de- veloped, may help further accelerate object detection.

**Acknowledgements.** I thank Kaiming He, Larry Zitnick, and Piotr Dolla´r for helpful discussions and encouragement.

# References

* + 1. J. Carreira, R. Caseiro, J. Batista, and C. Sminchisescu. Se- mantic segmentation with second-order pooling. In *ECCV*, 2012. [5](#_bookmark10)
    2. R. Caruana. Multitask learning. *Machine learning*, 28(1), 1997. [6](#_bookmark15)
    3. K. Chatﬁeld, K. Simonyan, A. Vedaldi, and A. Zisserman. Return of the devil in the details: Delving deep into convo- lutional nets. In *BMVC*, 2014. [5](#_bookmark10)
    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](#_bookmark2)

* + 1. E. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efﬁcient evaluation. In *NIPS*, 2014. [4](#_bookmark7)
    2. D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov. Scalable object detection using deep neural networks. In *CVPR*, 2014. [3](#_bookmark4)
    3. M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and

A. Zisserman. The PASCAL Visual Object Classes (VOC)

Challenge. *IJCV*, 2010. [1](#_bookmark0)

* + 1. P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ra- manan. Object detection with discriminatively trained part based models. *TPAMI*, 2010. [3](#_bookmark4), [7](#_bookmark20), [8](#_bookmark24)
    2. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich fea- ture hierarchies for accurate object detection and semantic segmentation. In *CVPR*, 2014. [1](#_bookmark0), [3](#_bookmark4), [4](#_bookmark7), [8](#_bookmark24)
    3. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Region- based convolutional networks for accurate object detection and segmentation. *TPAMI*, 2015. [5](#_bookmark10), [7](#_bookmark20), [8](#_bookmark24)
    4. K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014. [1](#_bookmark0), [2](#_bookmark2), [3](#_bookmark4), [4](#_bookmark7), [5](#_bookmark10), [6](#_bookmark15), [7](#_bookmark20)
    5. J. H. Hosang, R. Benenson, P. Dolla´r, and B. Schiele. What makes for effective detection proposals? *arXiv preprint* *arXiv:1502.05082*, 2015. [8](#_bookmark24)
    6. Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Gir- shick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proc. of the ACM* *International Conf. on Multimedia*, 2014. [2](#_bookmark2)
    7. A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet clas- siﬁcation with deep convolutional neural networks. In *NIPS*, 2012. [1](#_bookmark0), [4](#_bookmark7), [6](#_bookmark15)
    8. S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. [1](#_bookmark0)
    9. 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](#_bookmark0)

* + 1. M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014. [5](#_bookmark10)
    2. T. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick,

J. Hays, P. Perona, D. Ramanan, P. Dolla´r, and C. L. Zit- nick. Microsoft COCO: common objects in context. *arXiv e-prints*, arXiv:1405.0312 [cs.CV], 2014. [8](#_bookmark24)

* + 1. P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated Recognition, Localiza- tion and Detection using Convolutional Networks. In *ICLR*, 2014. [1](#_bookmark0), [3](#_bookmark4)
    2. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. [1](#_bookmark0), [5](#_bookmark10)
    3. J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders.

Selective search for object recognition. *IJCV*, 2013. [8](#_bookmark24)

* + 1. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR*, 2001. [8](#_bookmark24)
    2. J. Xue, J. Li, and Y. Gong. Restructuring of deep neural network acoustic models with singular value decomposition.

In *Interspeech*, 2013. [4](#_bookmark7)

* + 1. X. Zhu, C. Vondrick, D. Ramanan, and C. Fowlkes. Do we need more training data or better models for object detec- tion? In *BMVC*, 2012. [7](#_bookmark20)
    2. 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](#_bookmark0), [5](#_bookmark10)