Convolutional Networks for Computer Vision Applications

Andrea Vedaldi

Latest version of the slides
http://www.robots.ox.ac.uk/~vedaldi/assets/teach/vedaldi16deepcv.pdf

Lab experience
https://www.robots.ox.ac.uk/~vgg/practicals/cnn-reg/

UNIVERSITY OF OXFORD

Computer Vision & CNNs

Image classification

- Coarse (high-level objects)
- Fine grained (dog, bird species)

Object detection

- R-CNN
- Bounding box regression, YOLO

Image segmentation

- Fully-connected networks
- U architectures
- CRF backprop

Sentence generation

- Recurrent CNNs
- LSTMs

Matching, optical flow, stereo

- Siamese architectures

Synthesis and visualization

- Pre-images and matching statistics
- Stochastic networks
- Adversarial networks

Pose, parts, key points

Action recognition

Attribute prediction

Depth-map estimation

Face recognition and verification

Text recognition and spotting

Computer Vision & CNNs

Image classification

- Coarse (high-level objects)
- Fine grained (dog, bird species)

Object detection

- R-CNN
- Bounding box regression, YOLO

Image segmentation

- Fully-connected networks
- U architectures
- CRF backprop

Sentence generation

- Recurrent CNNs
- LSTMs

Matching, optical flow, stereo

- Siamese architectures

Synthesis and visualization

- Pre-images and matching statistics
- Stochastic networks
- Adversarial networks

Pose, parts, key points

Action recognition

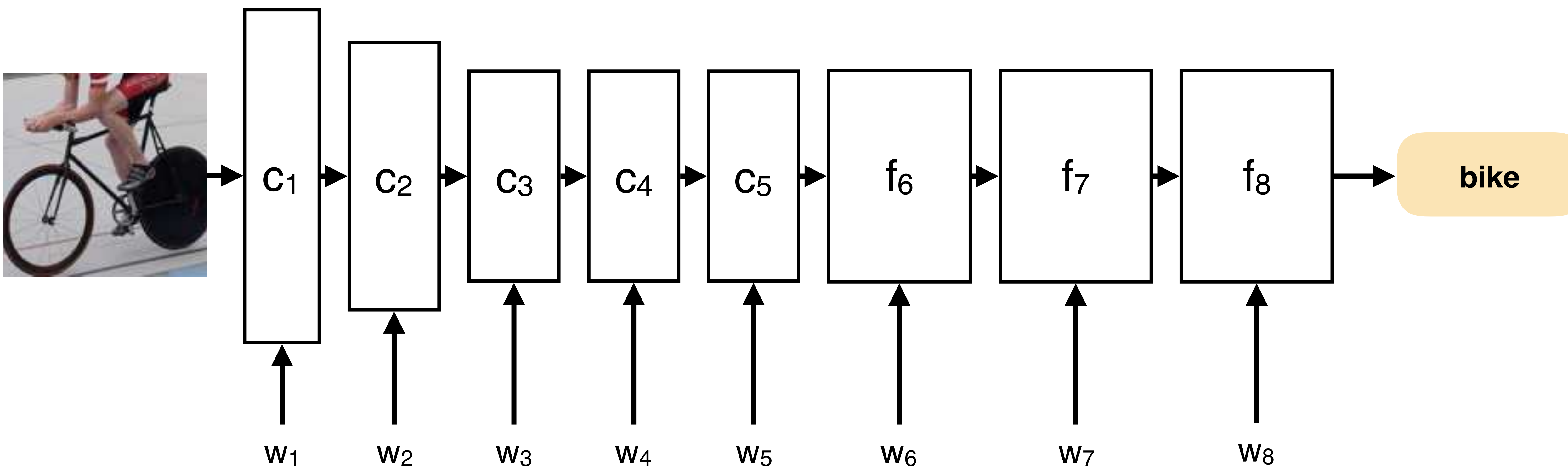
Attribute prediction

Depth-map estimation

Face recognition and verification

Text recognition and spotting

Review



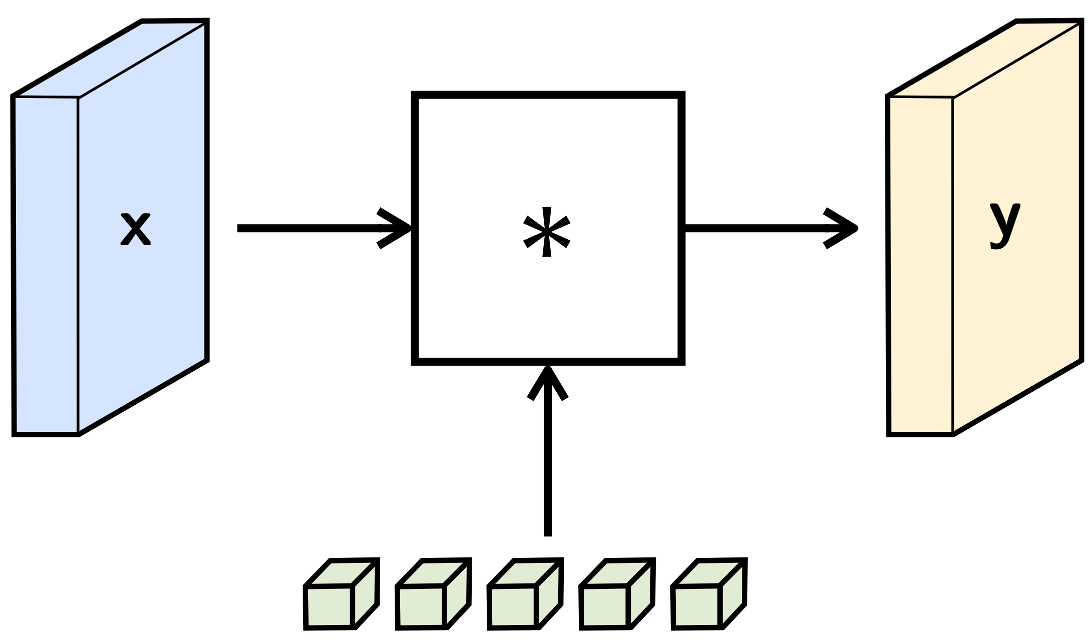
A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

Convolutional Neural Network (CNN)

A sequence of local & shift invariant layers

Example: convolution layer

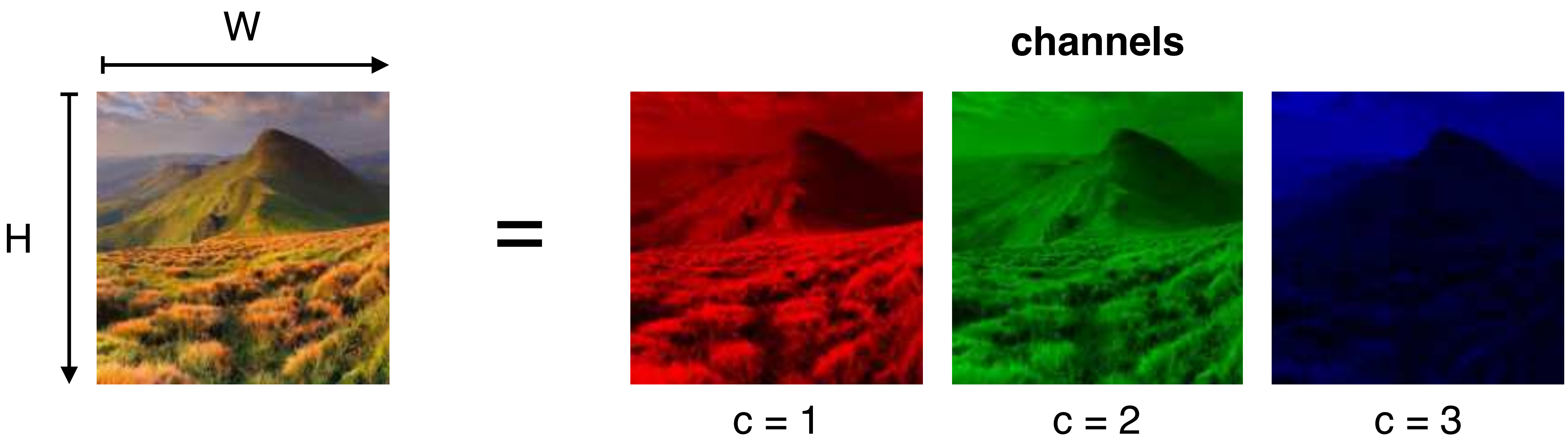
$$\boxed{\mathbf{y} = F * \mathbf{x} + b}$$

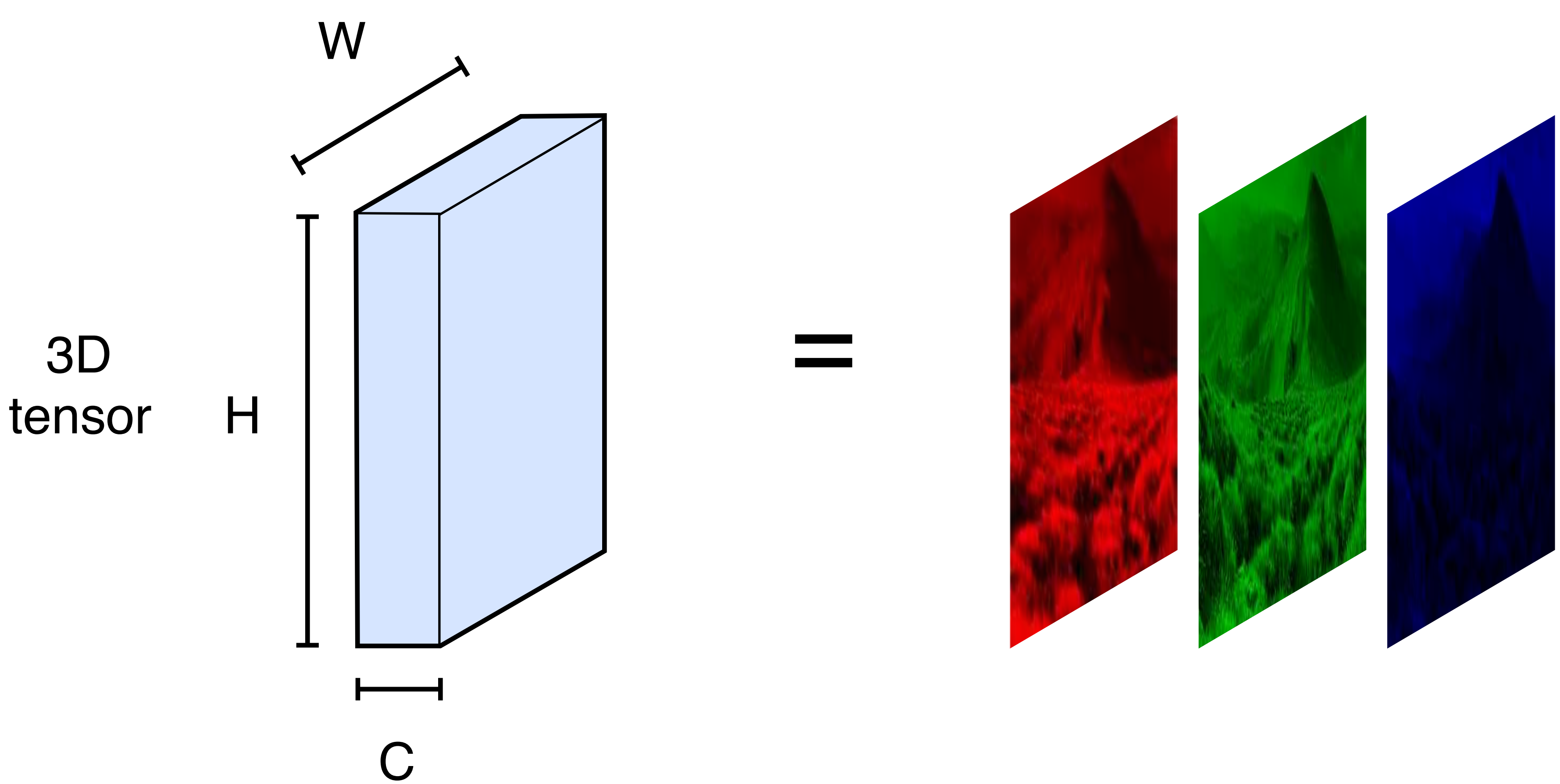


input data $\mathbf{x}$ filter bank F output data $\mathbf{y}$

Data = 3D tensors

There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).



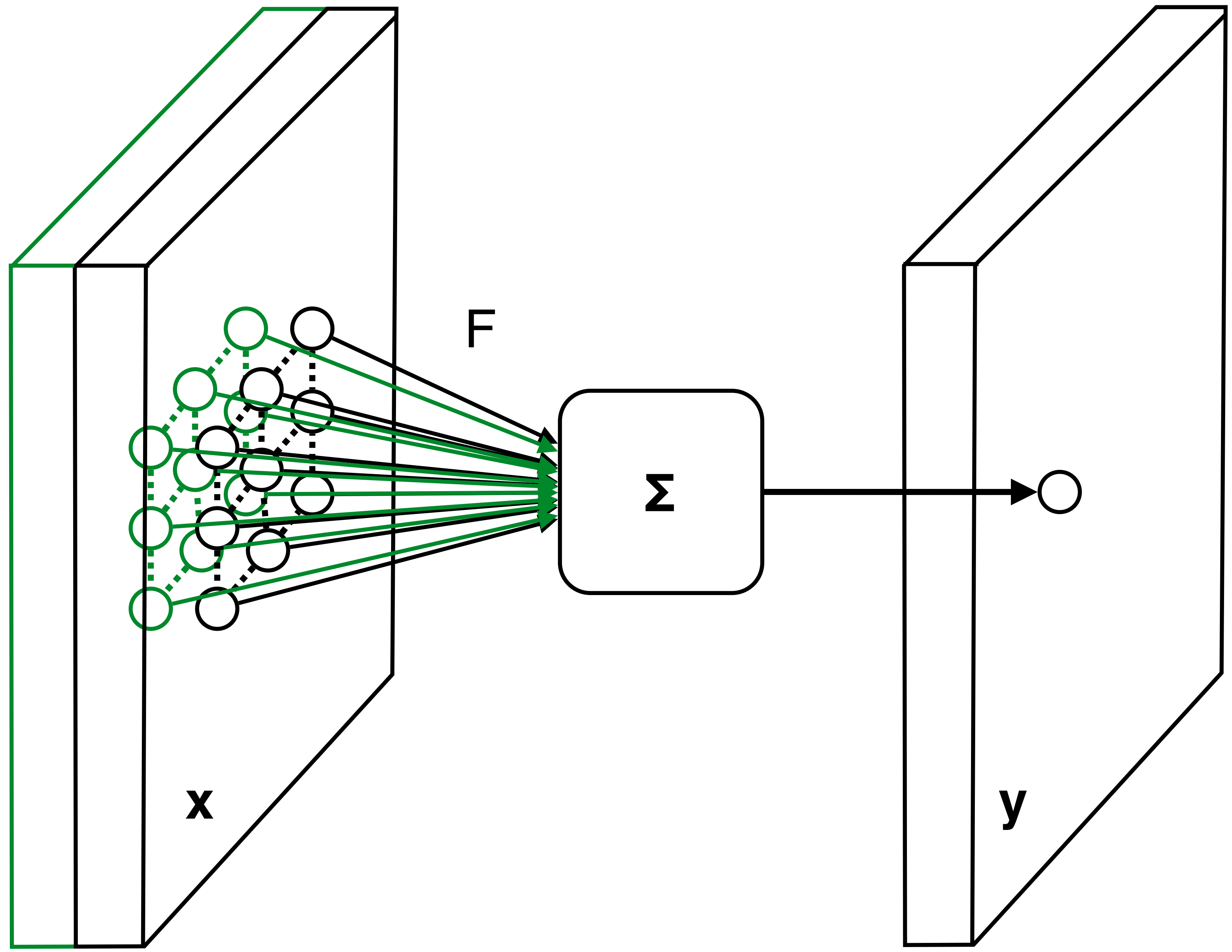


Convolution with 3D filters

Each filter acts on multiple input channels

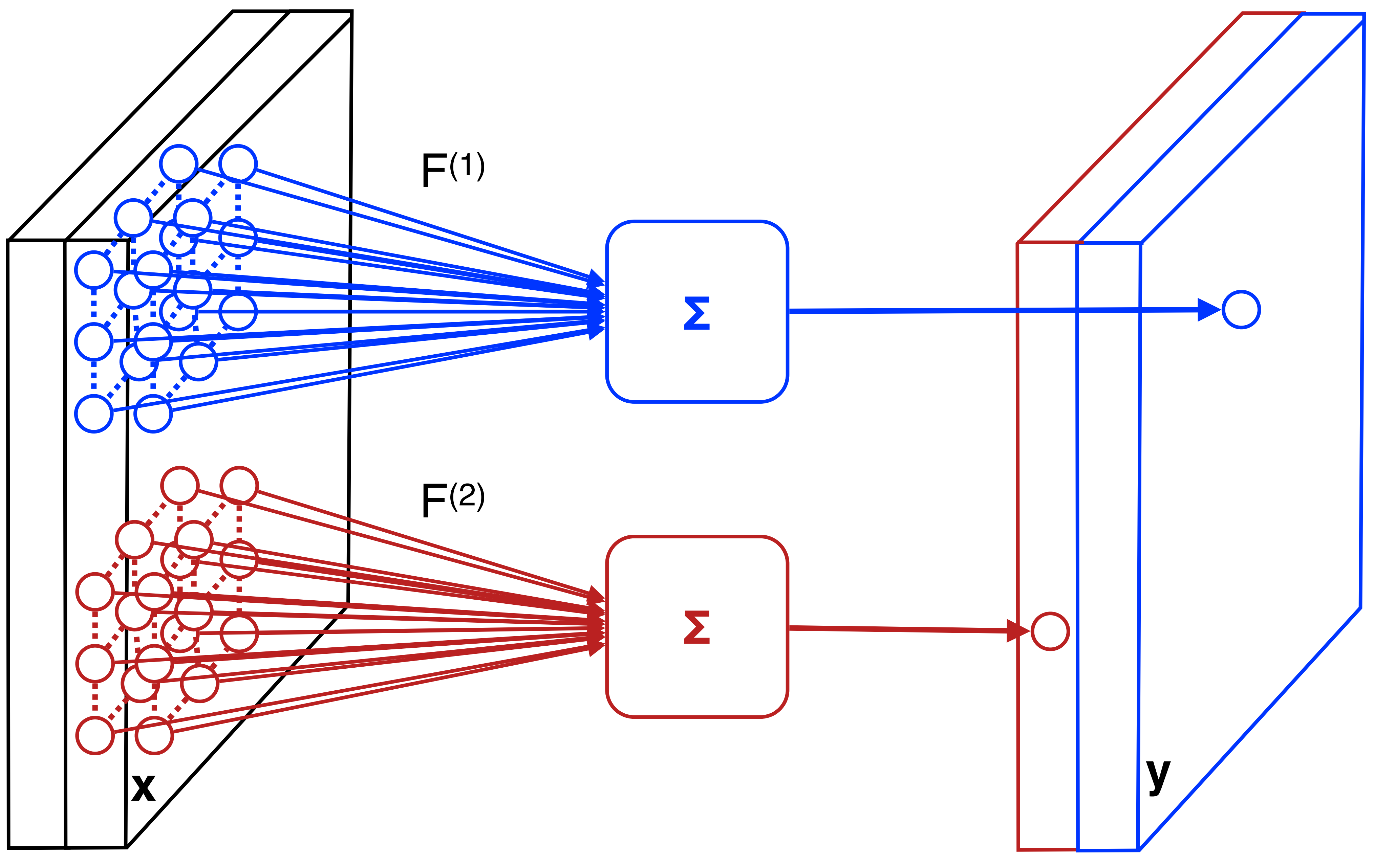
Local
Filters look locally

Translation invariant
Filters act the same everywhere



Filter banks

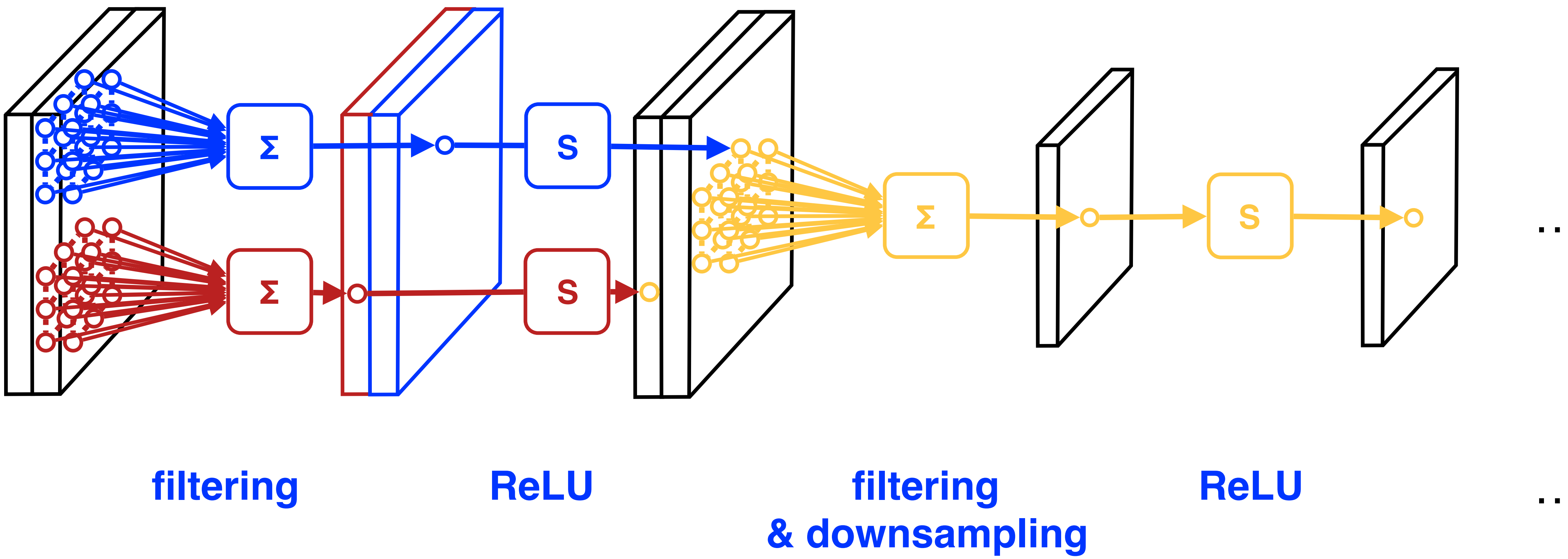
Multiple filters produce multiple output channels



One filter = one output channel

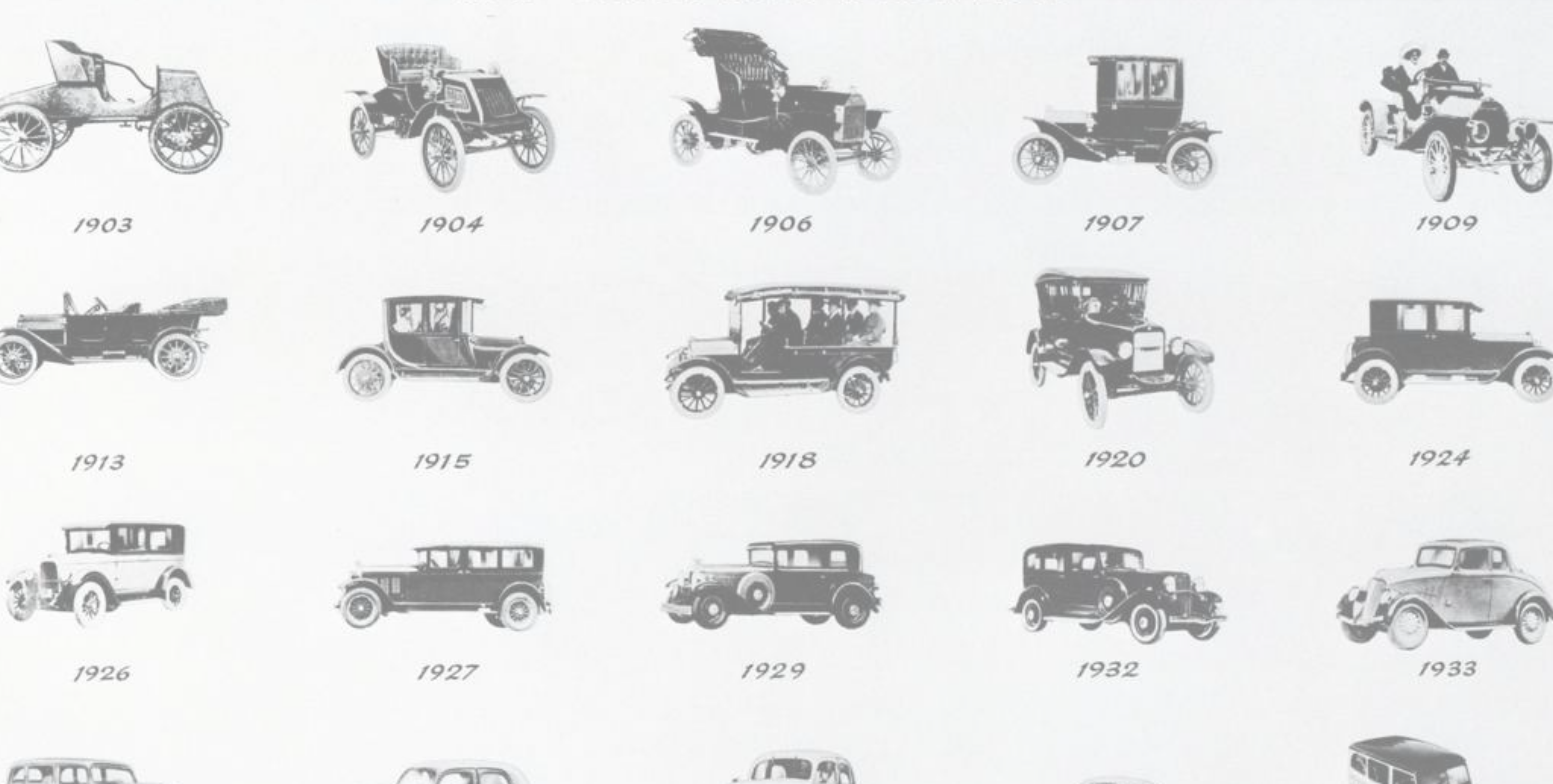
Linear / non-linear chains

The basic blueprint of most architectures



Three years of progress

From AlexNet (2012) to ResNet (2015)

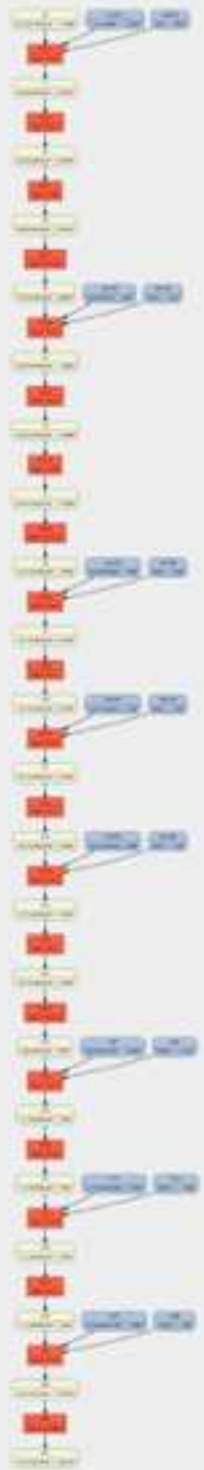


How deep is enough?

AlexNet (2012)

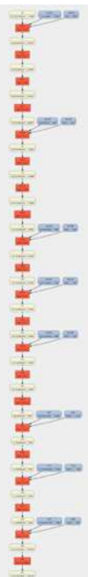
5 convolutional layers

3 fully-connected layers

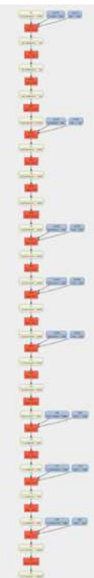


How deep is enough?

AlexNet (2012)



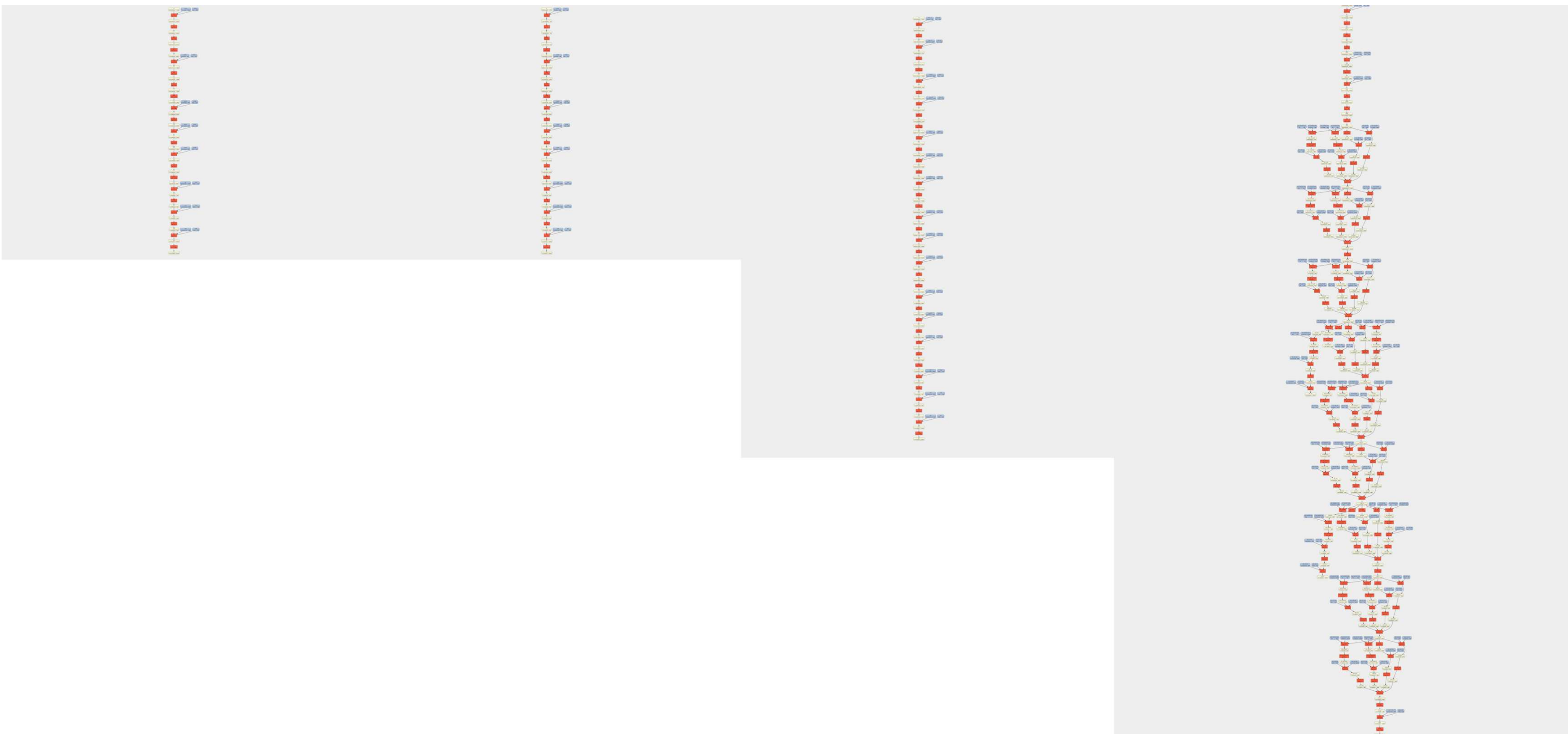
VGG-M (2013)



VGG-VD-16 (2014)

How deep is enough?

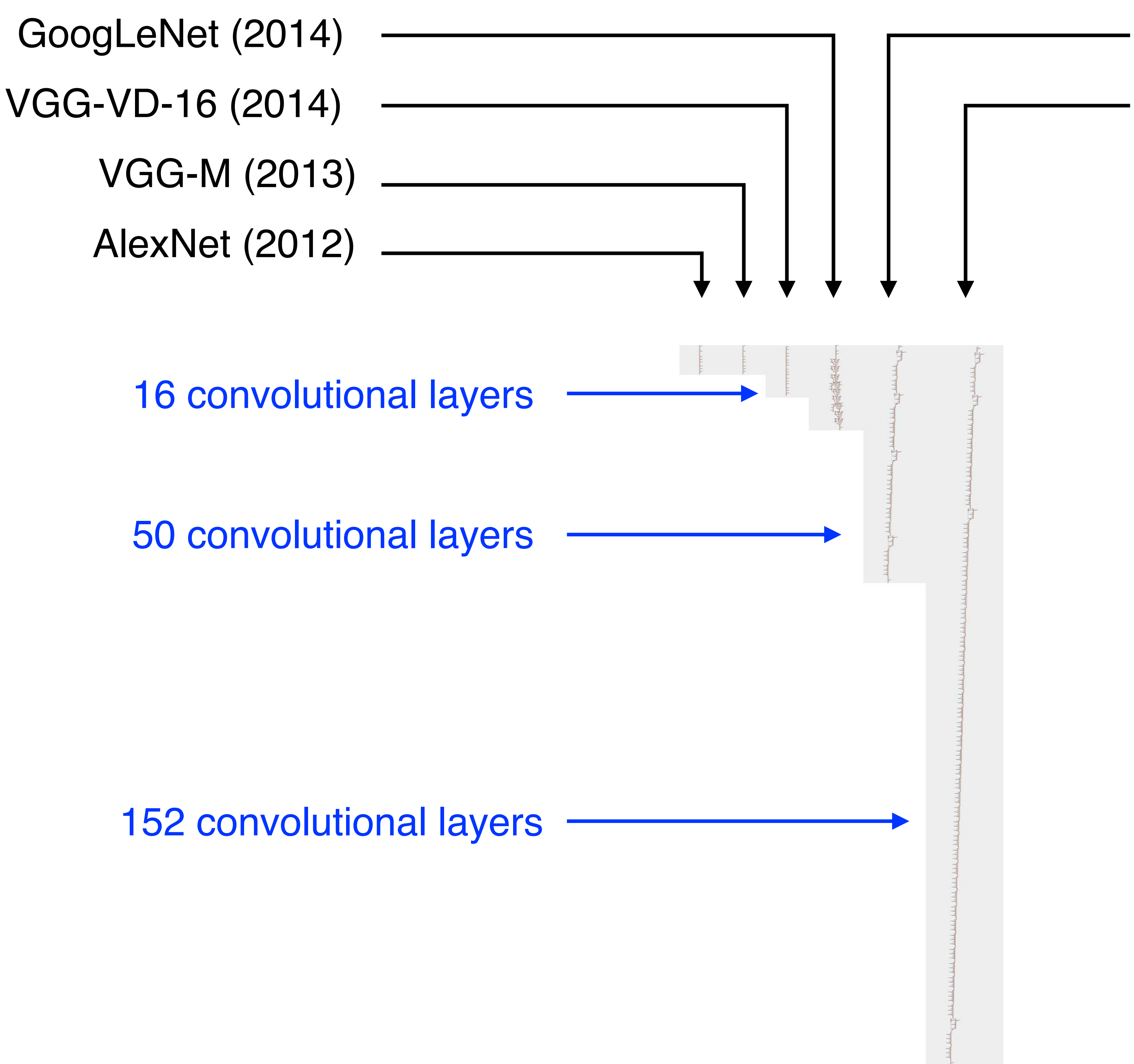
AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)



How deep is enough?

AlexNet (2012) VGG-M (2013) VGG-VD-16 (2014) GoogLeNet (2014)

How deep is enough?



Krizhevsky, I. Sutskever, and G. E. Hinton. *ImageNet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

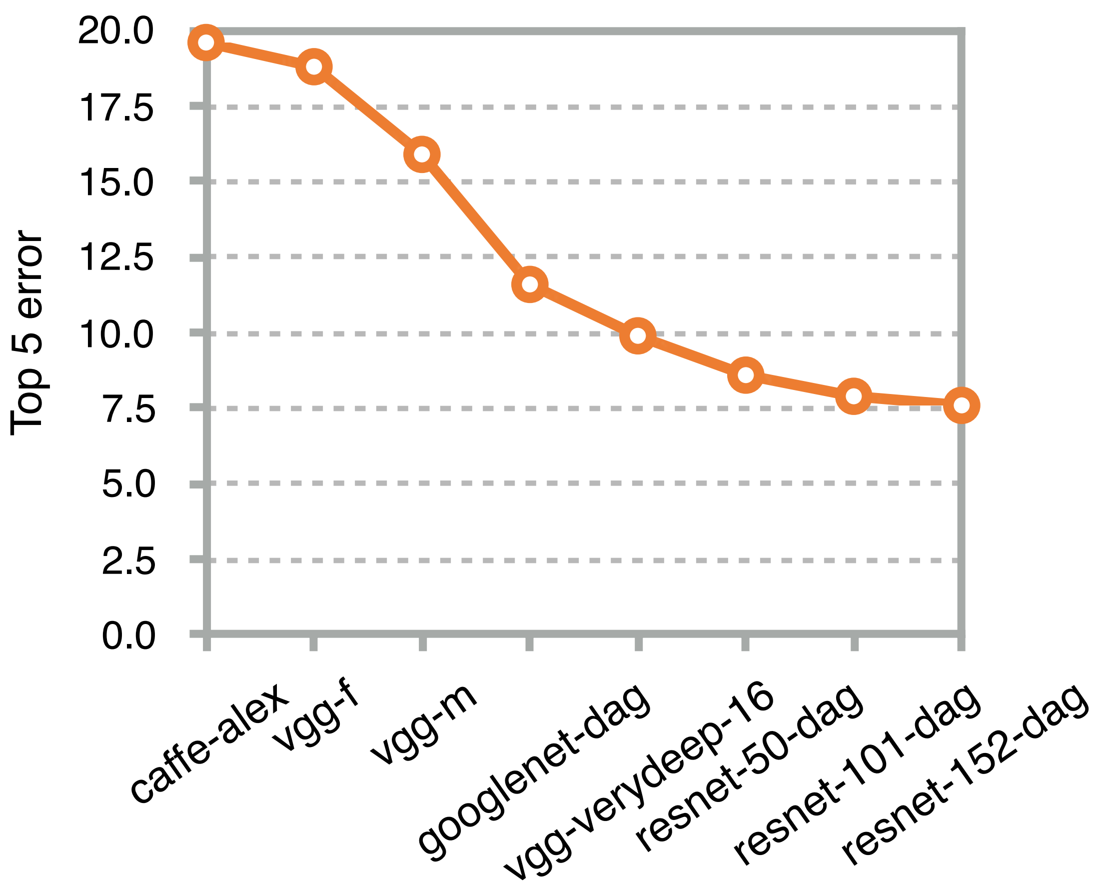
C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going deeper with convolutions*. In Proc. CVPR, 2015.

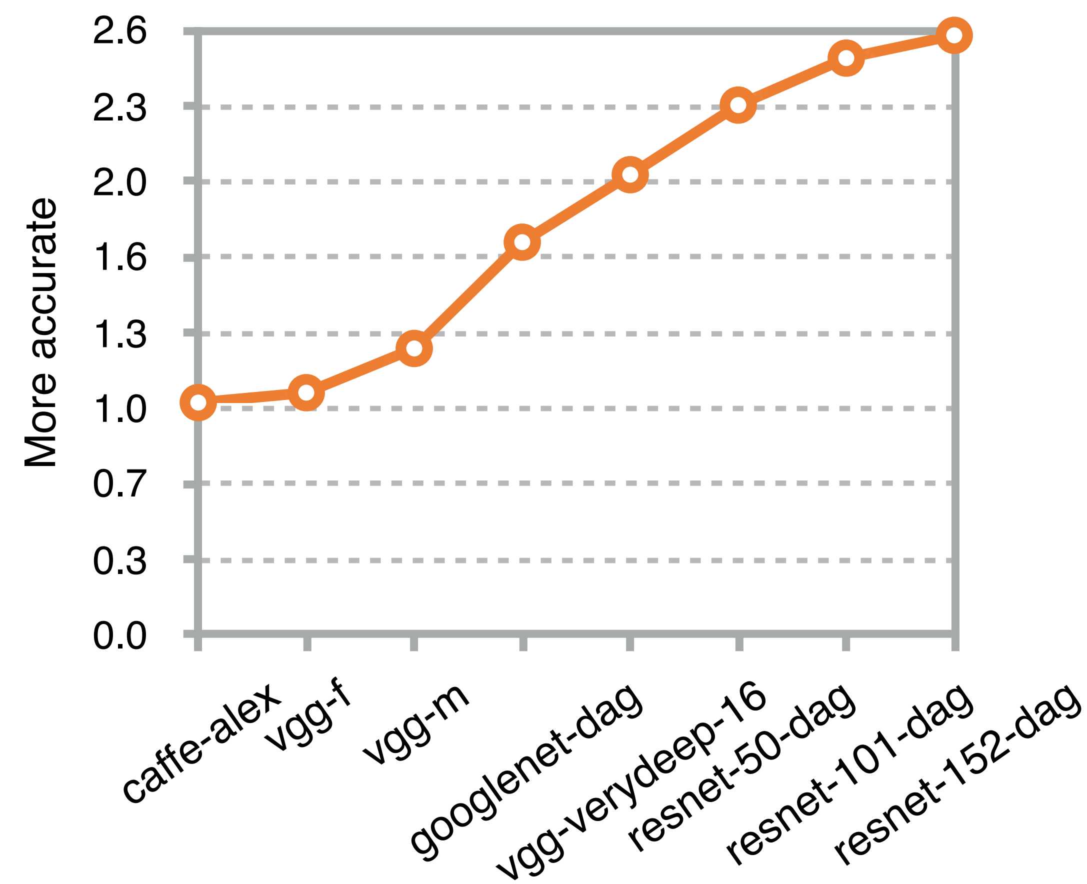
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.

Accuracy

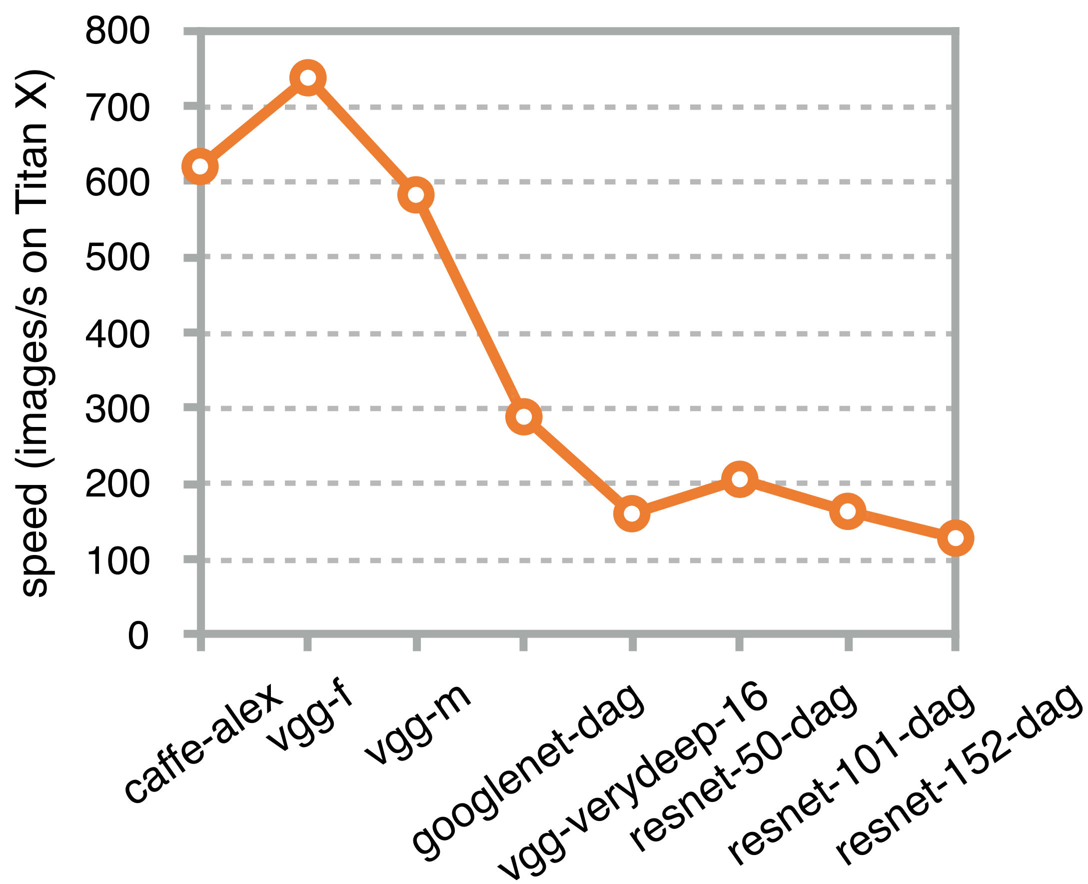
3 × more accurate in 3 years

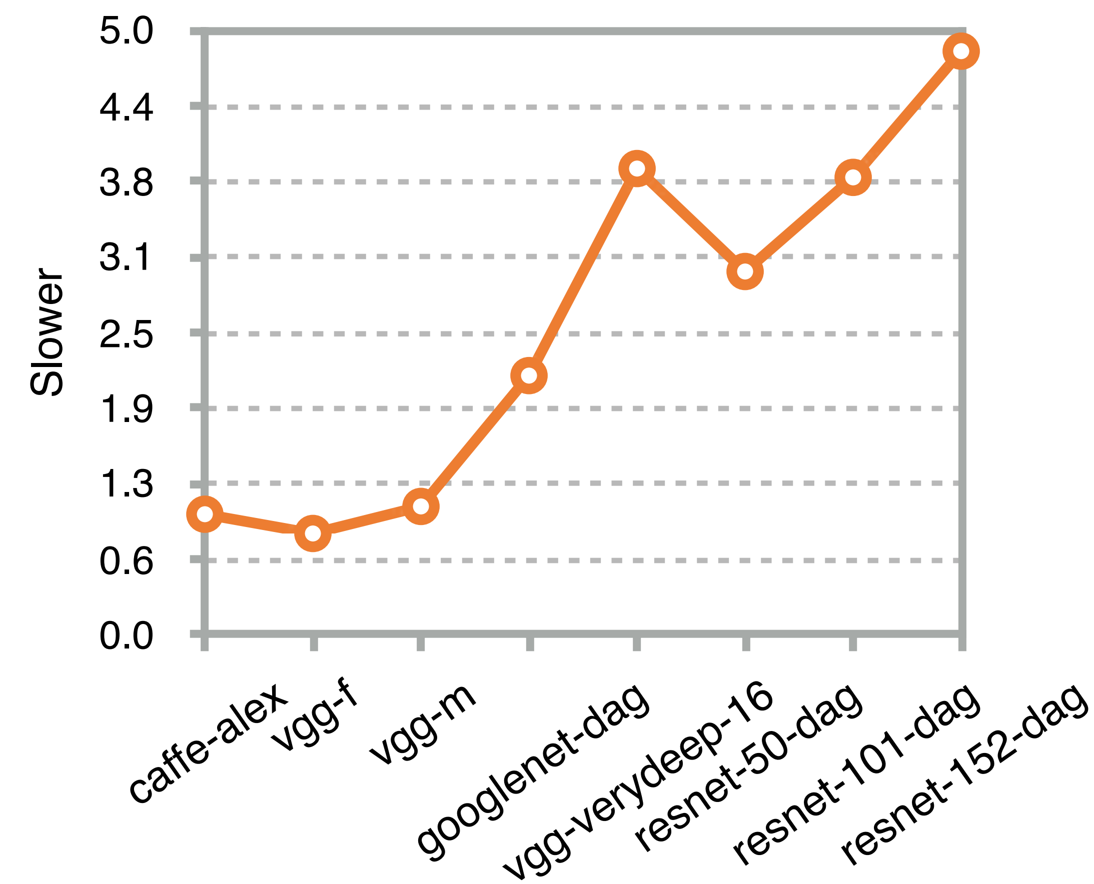




Speed

5 × slower





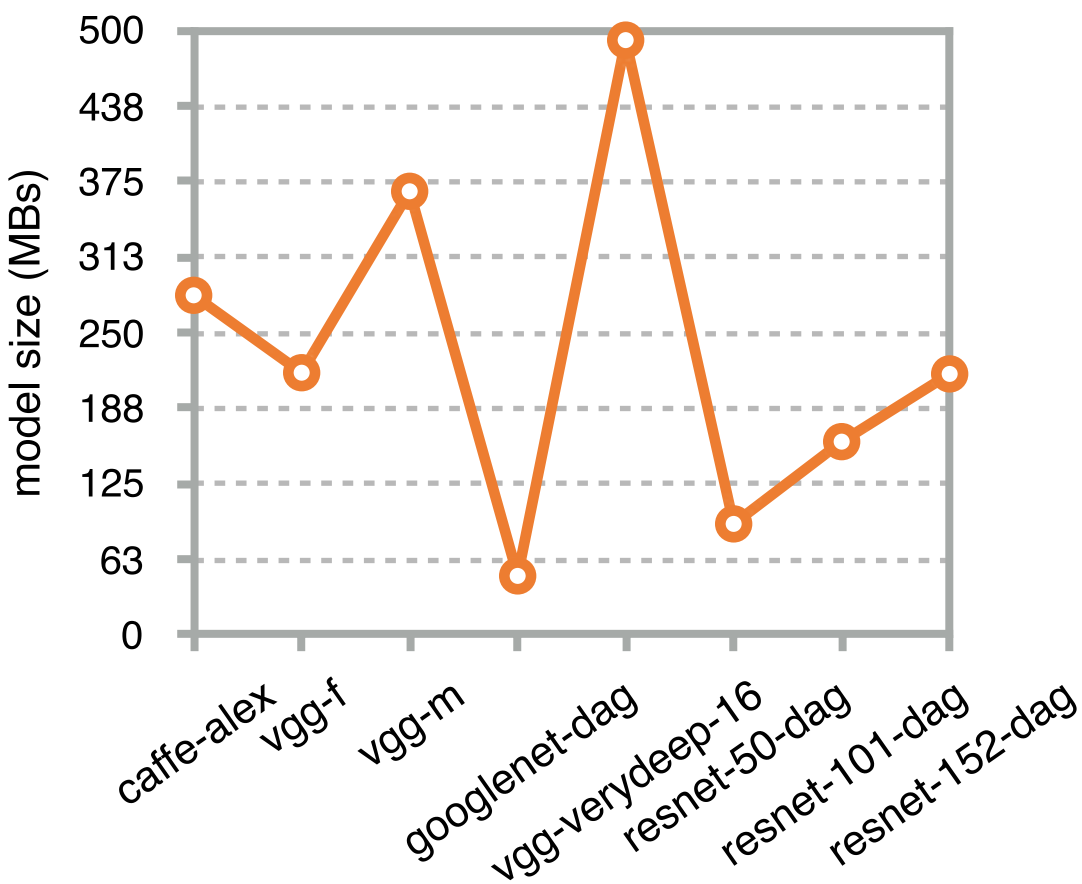
Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

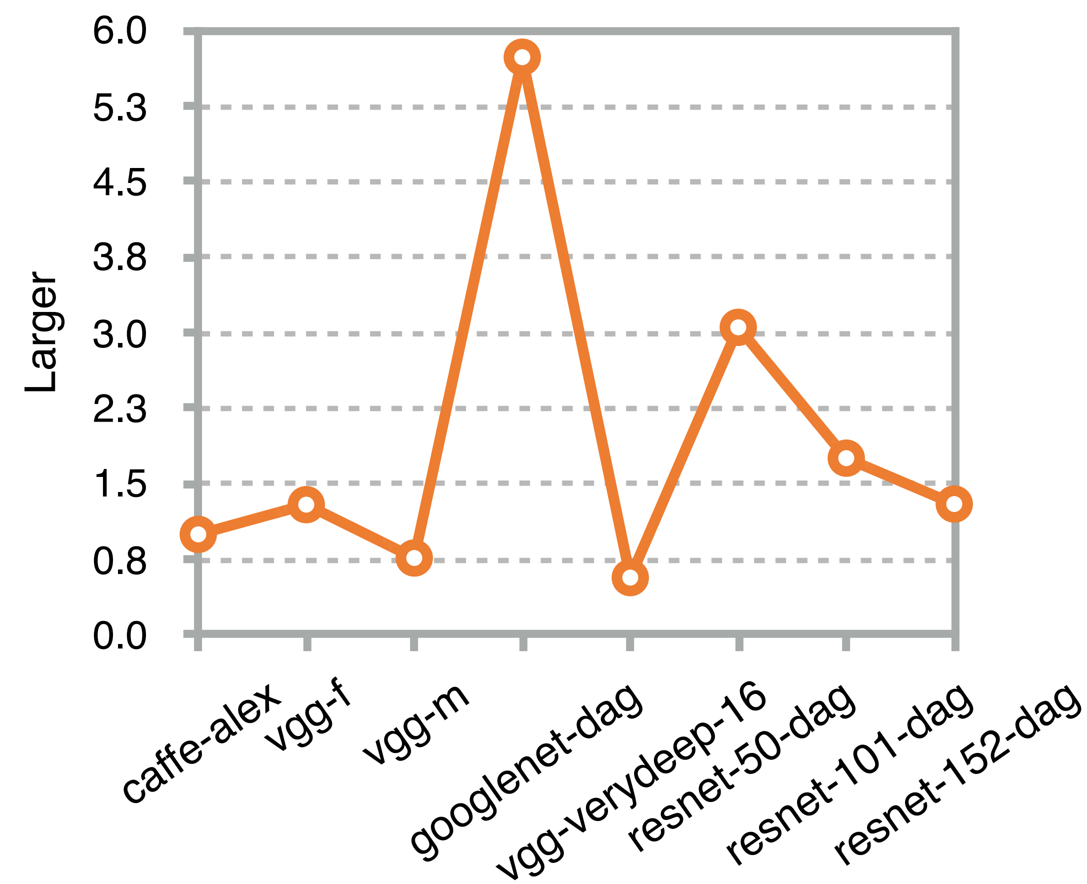
Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture

Model size

Num. of parameters is about the same





Remark: *101 ResNet* layers same size/speed as *16 VGG-VD* layers

Reason: far fewer feature channels (quadratic speed/space gain)

Moral: optimize your architecture

Recent advances

Design guidelines

Batch normalization

Residual learning

Recent advances

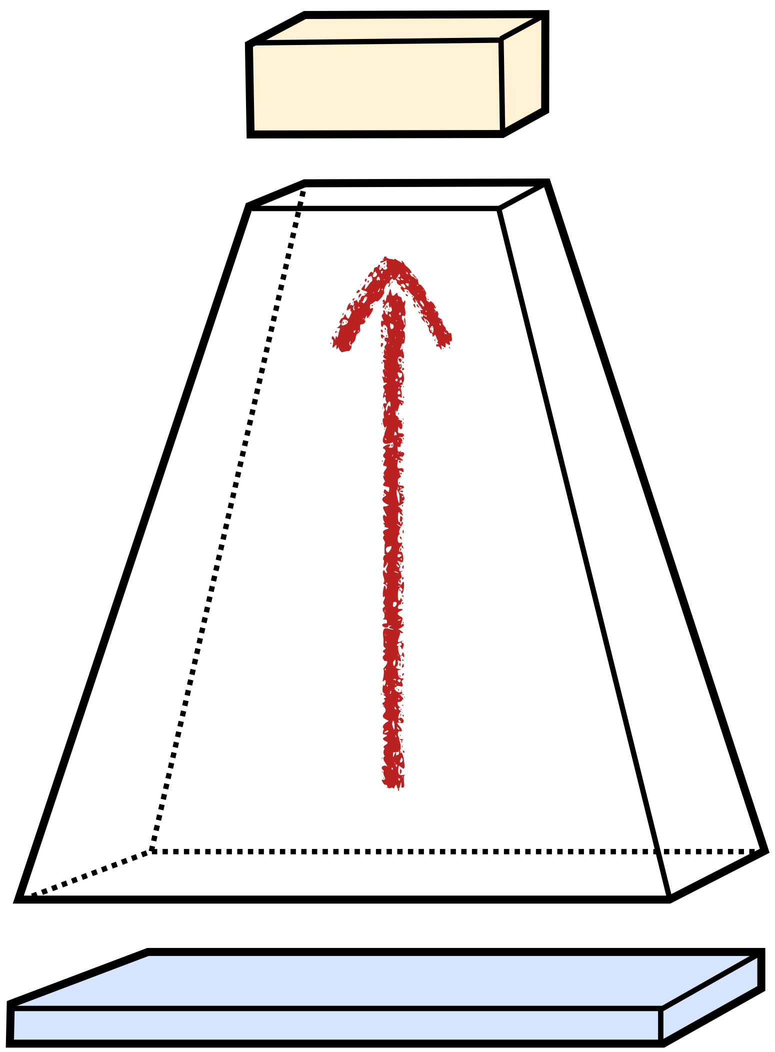
Design guidelines

Batch normalization

Residual learning

Design guidelines

Guideline 1: *Avoid tight bottlenecks*



From bottom to top

- The *spatial resolution* $H \times W$ decreases
- The *number of channels* C increases

Guideline

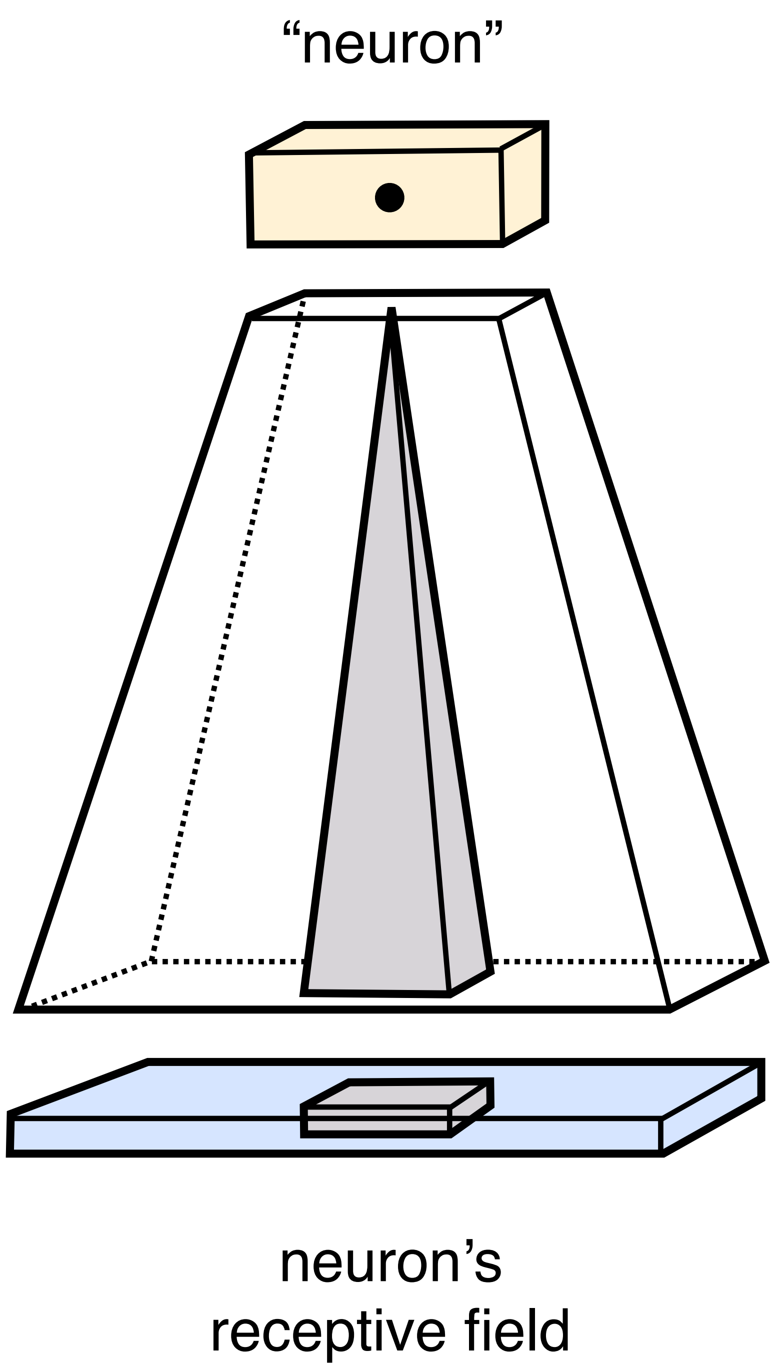
- Avoid tight information bottleneck
- Decrease the *data volume* $H \times W \times C$ slowly

K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

C. Szegedy, V. Vanhoucke, S. Ioffe, and J. Shlens. *Rethinking the inception architecture for computer vision*. In Proc. CVPR, 2016.

Receptive field

Must be large enough



Receptive field of a neuron

- The image region influencing a neuron
- Anything happening outside is invisible to the neuron

Importance

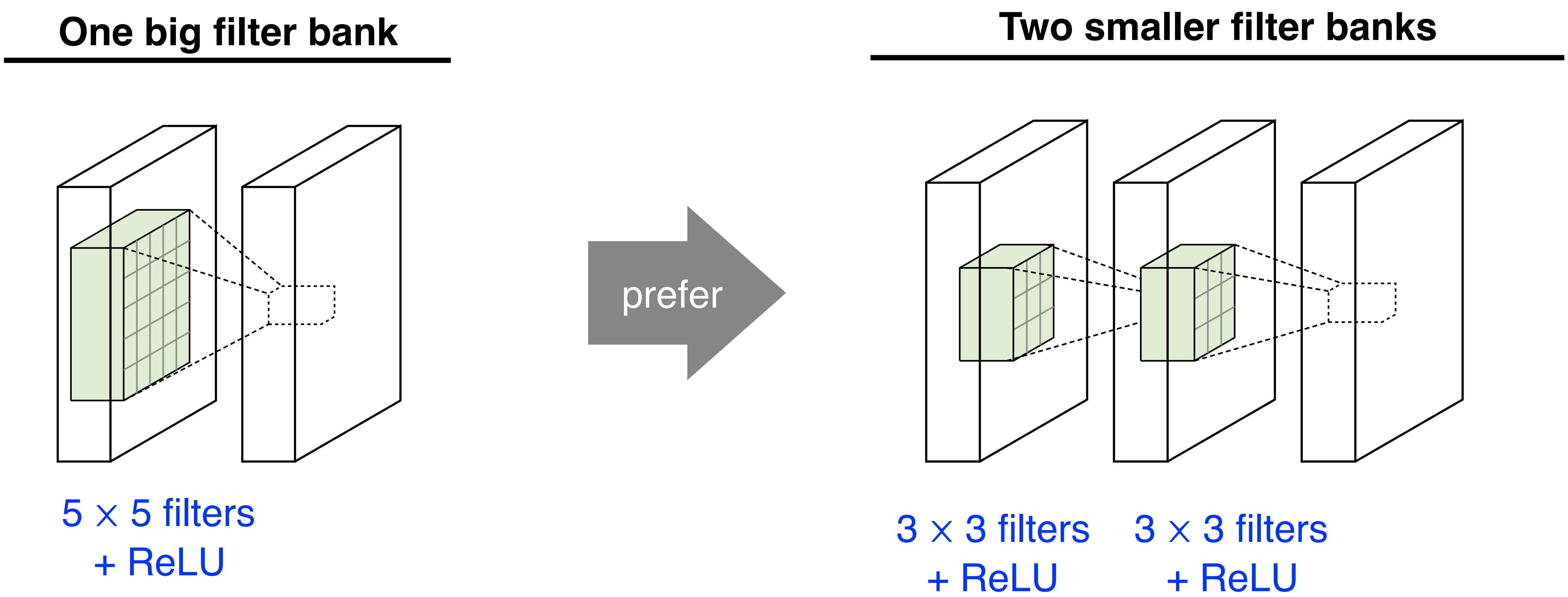
- Large image structures cannot be detected by neurons with small receptive fields

Enlarging the receptive field

- Large filters
- Chains of small filters

Design guidelines

Guideline 2: *Prefer small filter chains*



Benefit 1: less parameters, possibly faster

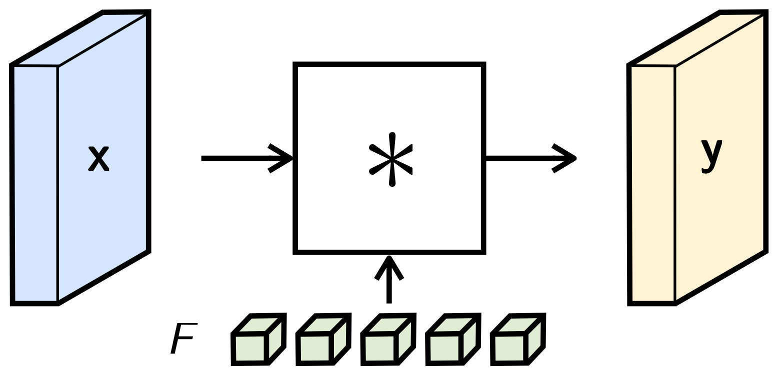
Benefit 2: same receptive field of big filter

Benefit 3: packs two non-linearities (ReLUs)

Design guidelines

Guideline 3: *Keep the number of channels at bay*

$H \times W \times C$



$H_f \times W_f \times C \times K$

C = num. input channels

K = num. output channels

Num. of operations

$$\frac{H \times H_f}{\text{stride}} \times \frac{W \times W_f}{\text{stride}} \times C \times K$$

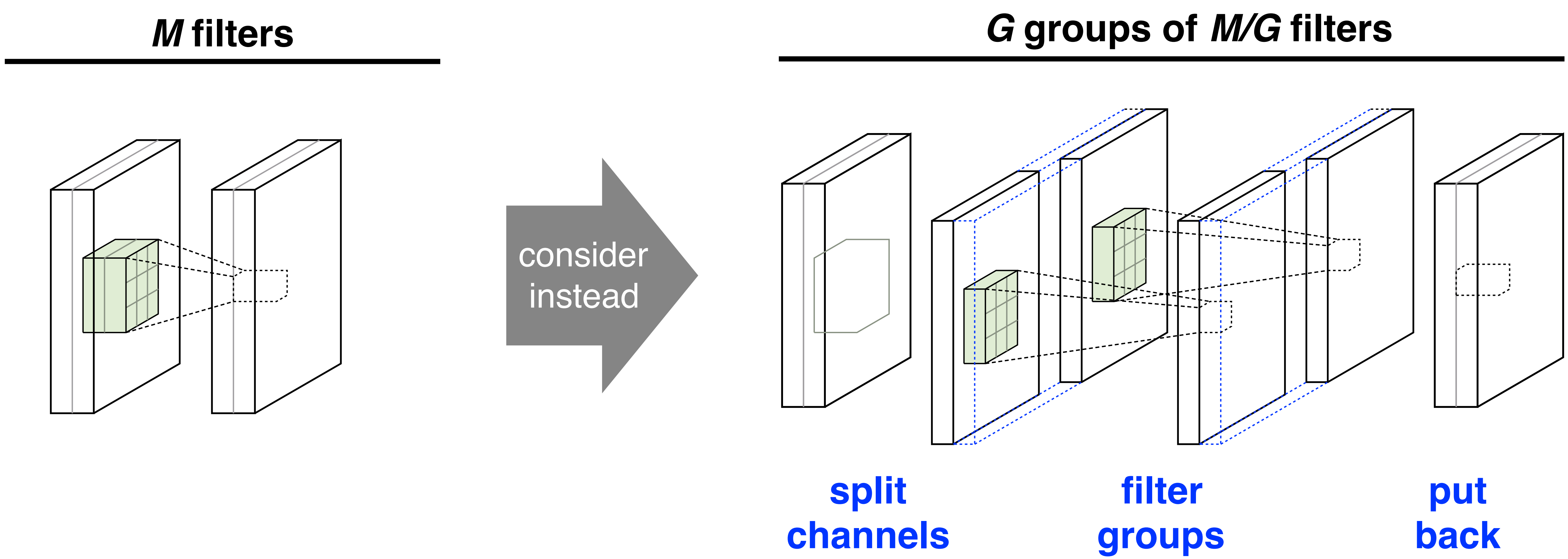
Num. of parameters

$$H_f \times W_f \times C \times K$$

$$\boxed{\text{complexity} \propto C \times K}$$

Design guidelines

Guideline 4: *Less computations with filter groups*

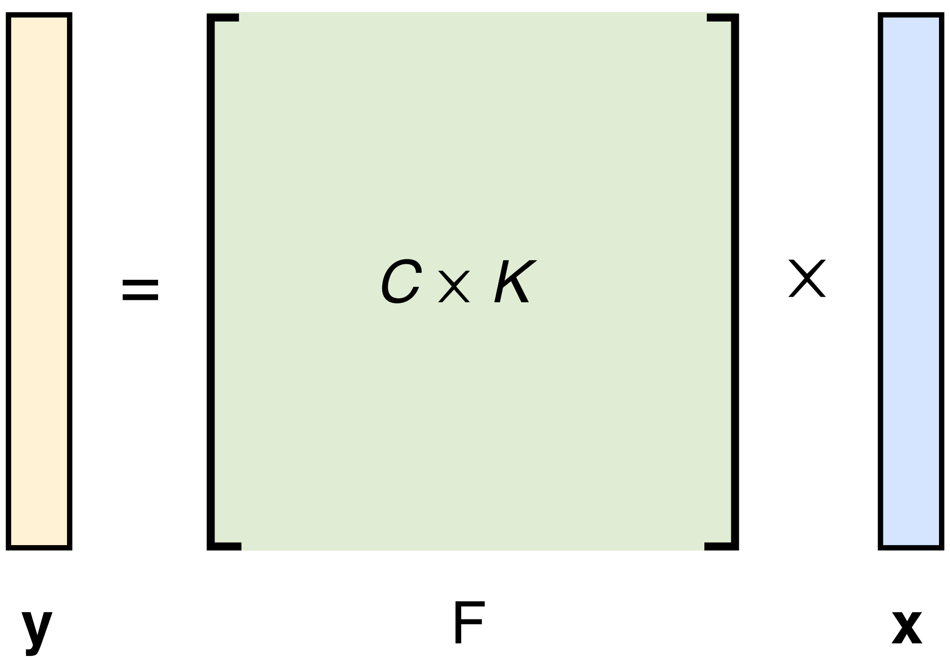


complexity $\propto (C \times K) / G$

Design guidelines

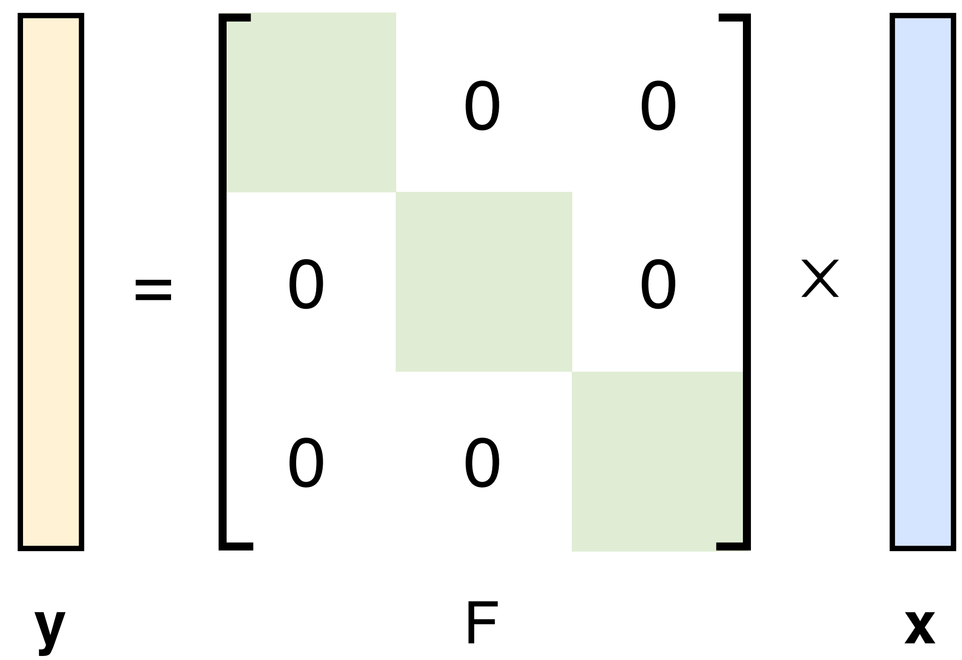
Guideline 4: *Less computations with filter groups*

Full filters



complexity: $C \times K$

Group-sparse filters

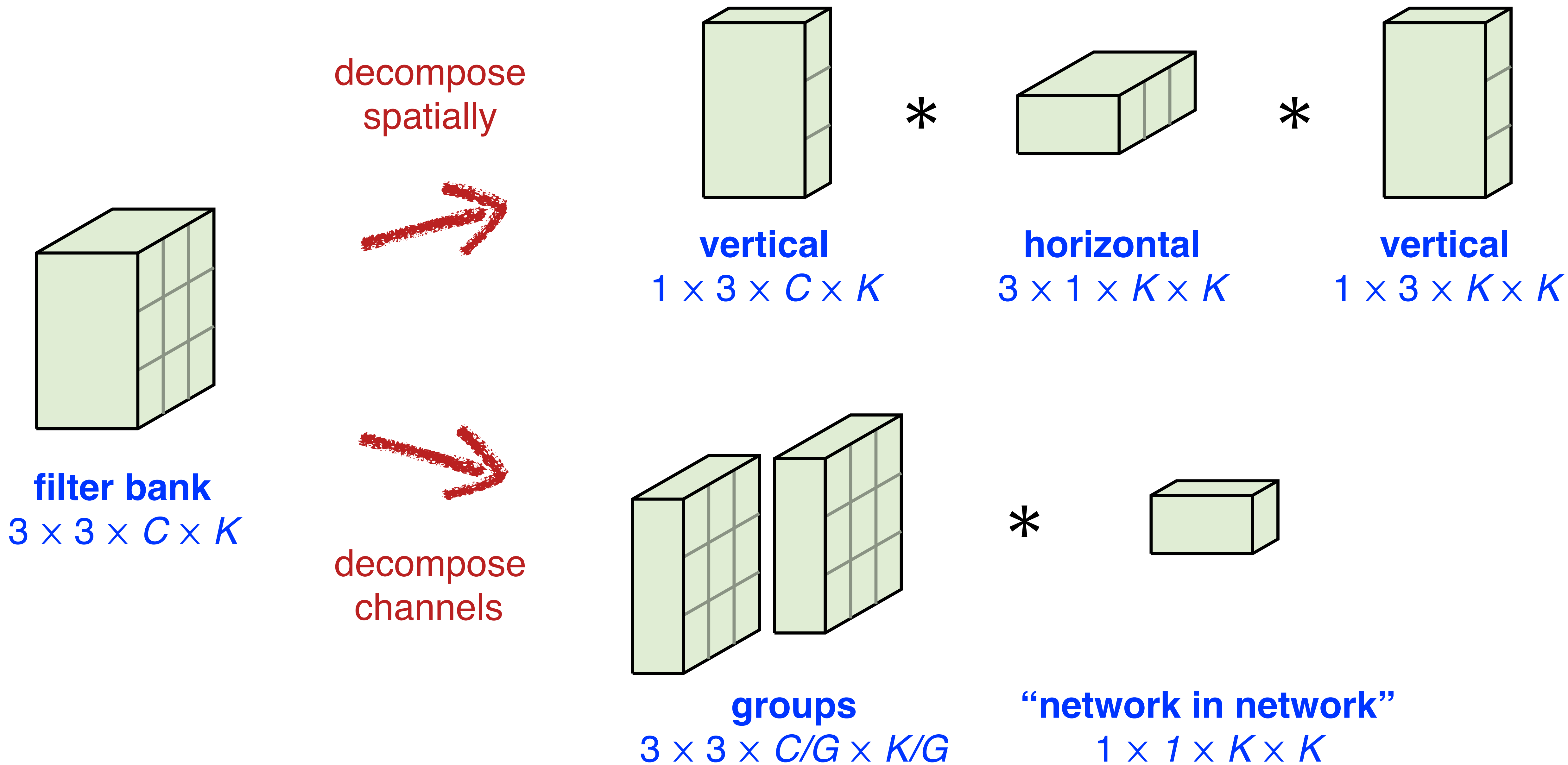


complexity: $C \times K / G$

Groups = filters, seen as a matrix, have a "block" structure

Design guidelines

Guideline 5: *Low-rank decompositions*



Make sure to *mix the information*

Recent advances

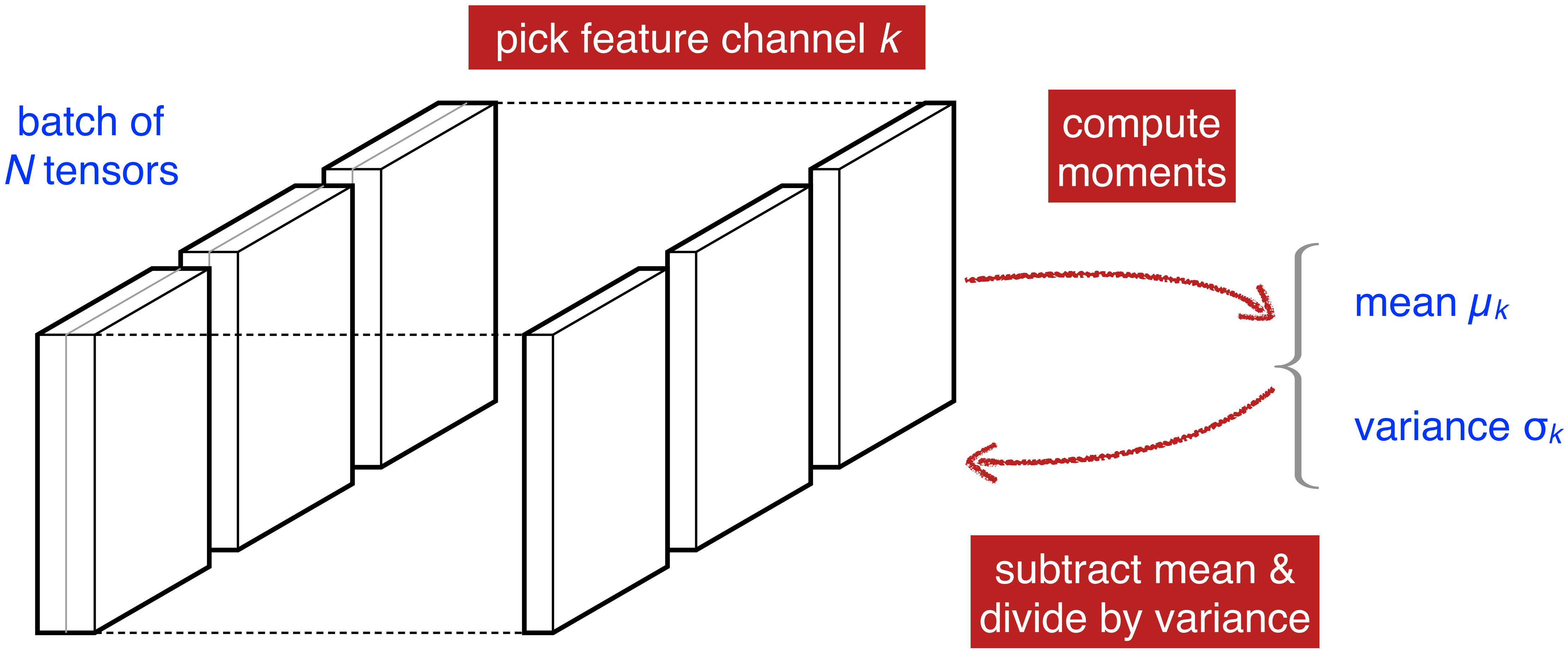
Design guidelines

Batch normalization

Residual learning

Batch normalization

Condition features



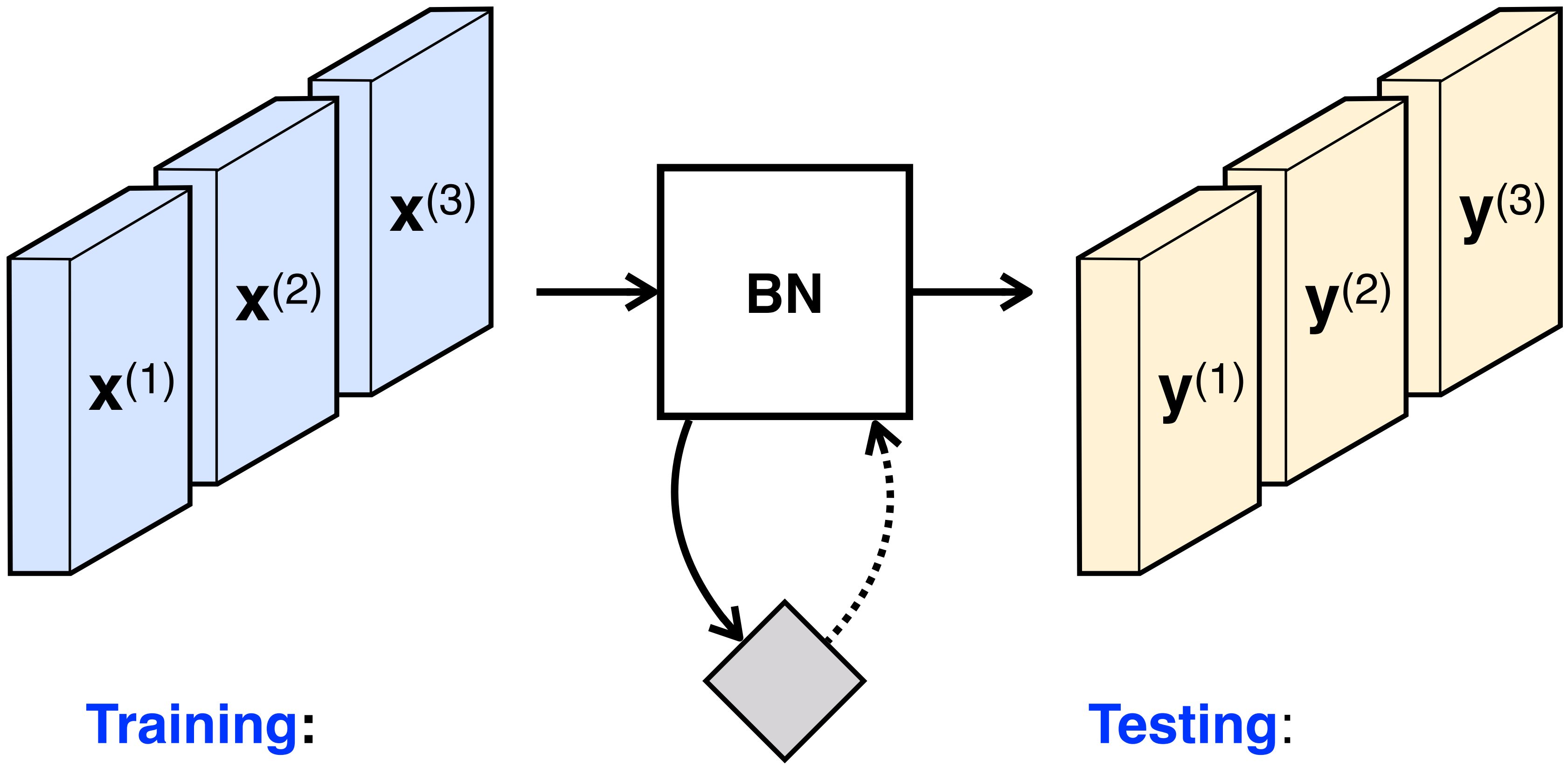
Standardize the response of each feature channel *within the batch*

- Average over spatial locations
- Also, average over multiple images in the batch (e.g. 16-256)

S. Ioffe and C. Szegedy. *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. CoRR, 2015

Batch normalization

Training vs testing modes



Training:
batch-specific
moment averages
are collected

moments
μ, σ

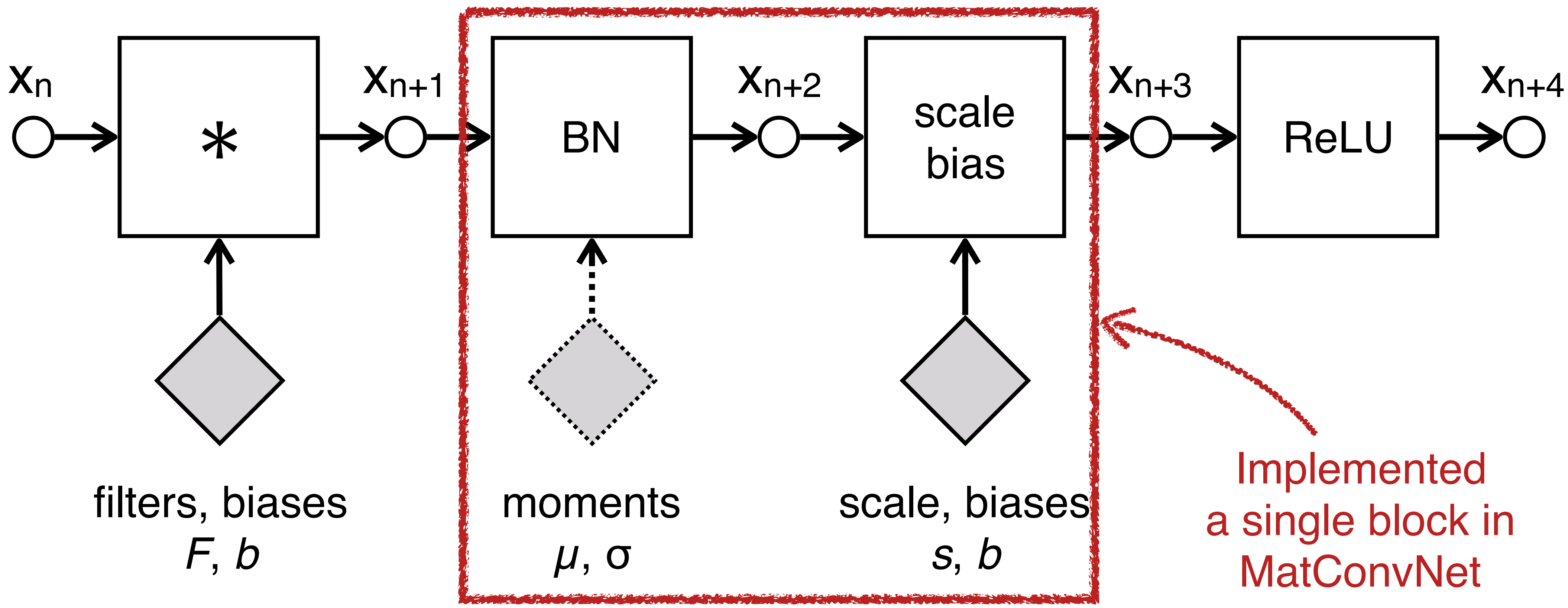
Testing:
moment averages are
used instead
of batch-specific
moments

Moments (mean & variance)

- **Training**: compute anew for each batch
- **Testing**: fixed to their average values

Batch normalization

Utilization



Batch normalization is used after filtering, before ReLU

It is always followed by channel-specific scaling factor s and bias b

Noisy bias/variance estimation **replaces dropout regularization**

Recent advances

Design guidelines

Batch normalization

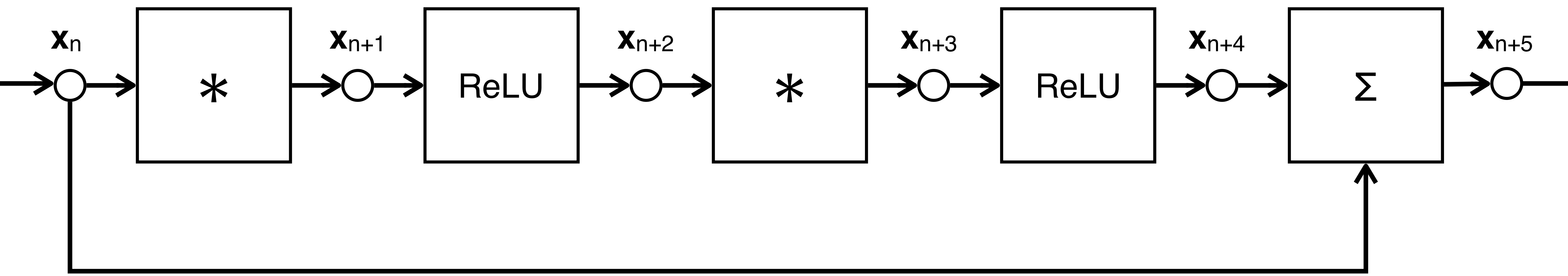
Residual learning

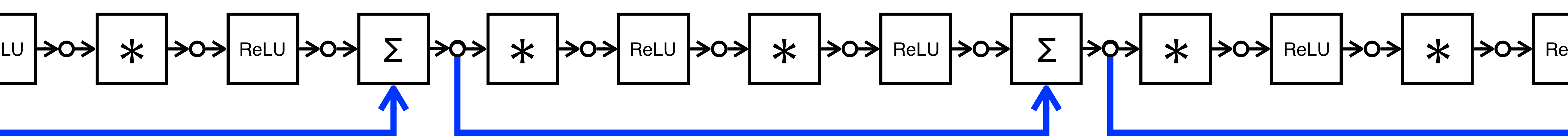
Residual learning

Fixed identity // learned residual

$$\underset{\substack{\uparrow \\ \text{identity}}}{\mathbf{x}_{n+5} = \mathbf{x}_n} + \underbrace{(\phi_{\mathrm{ReLU}} \circ \phi_* \circ \phi_{\mathrm{ReLU}} \circ \phi_*)}_{\text{residual}}(\mathbf{x}_n)$$

K. He, X. Zhang, S. Ren, and J. Sun. *Deep residual learning for image recognition*. In Proc. CVPR, 2016.





Summary

Impact of deep learning in vision

- **2012**: amazing results by AlexNet in the ImageNet challenge
- **2013-15**: massive 3 improvement
- **2016-19**: further massive improvements not unlikely

What have we learned

- several **incremental refinements**
- AlexNet was just a first proof of concept after all

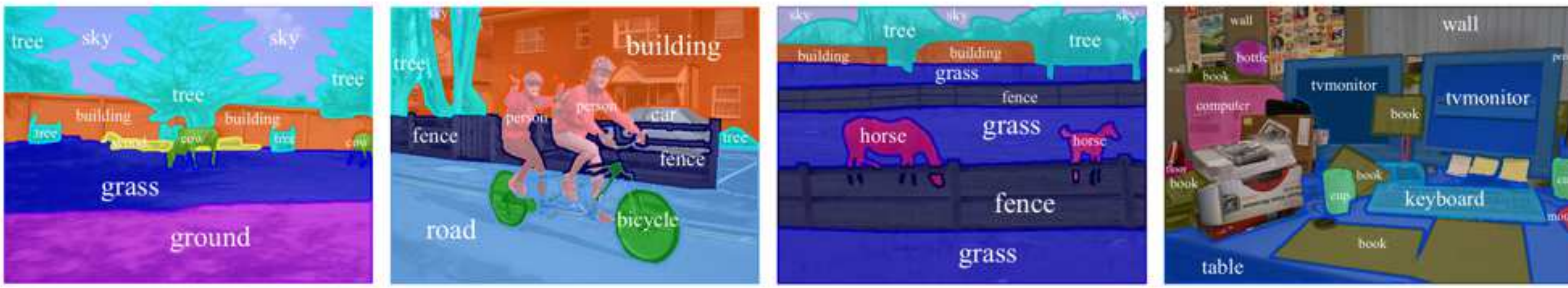
Things that work

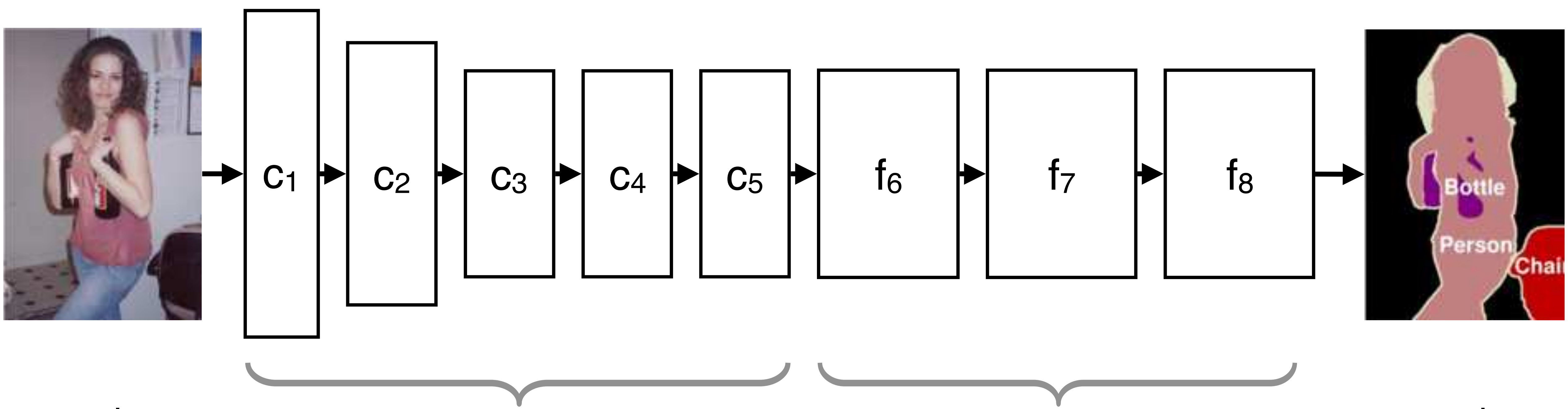
- Deeper architectures
- Smarter architectures (groups, low rank decompositions, …)
- Batch normalization
- Residual connections

Semantic segmentation

Semantic image segmentation

Label individual pixels





input = image

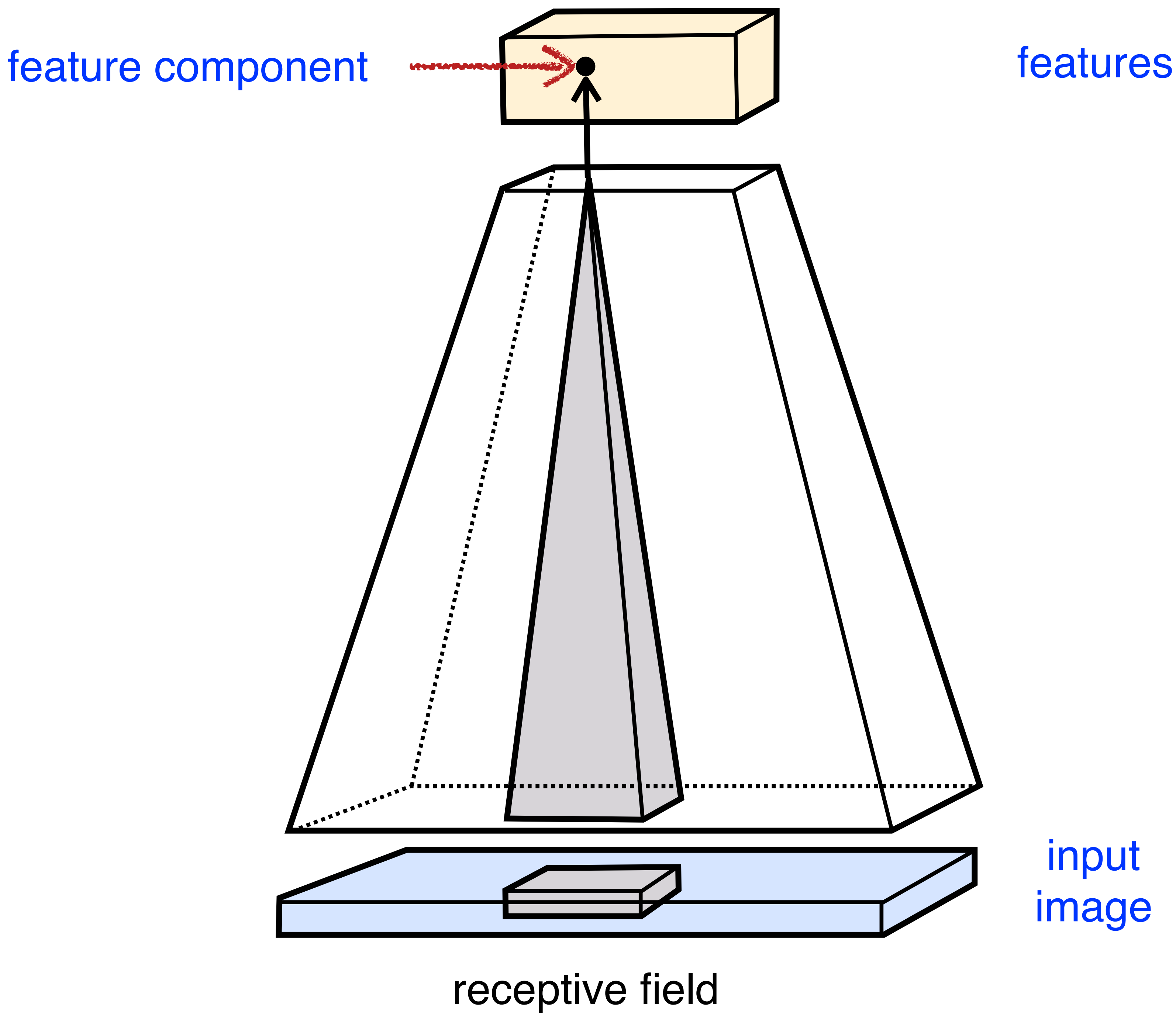
convolutional

fully-connected

output = image

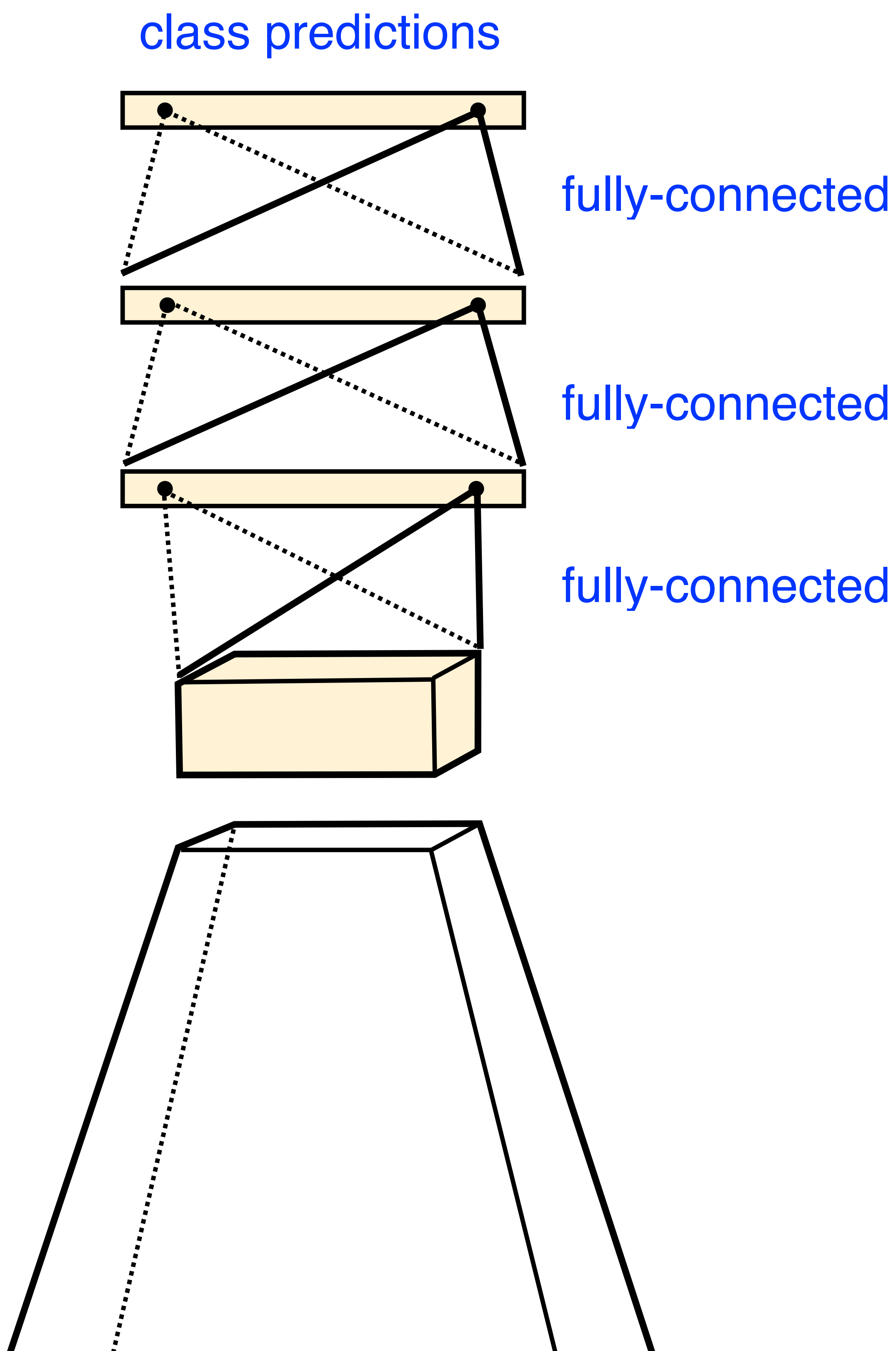
Convolutional layers

Local receptive field



Fully connected layers

Global receptive field



Convolutional vs Fully Connected

Comparing the receptive fields

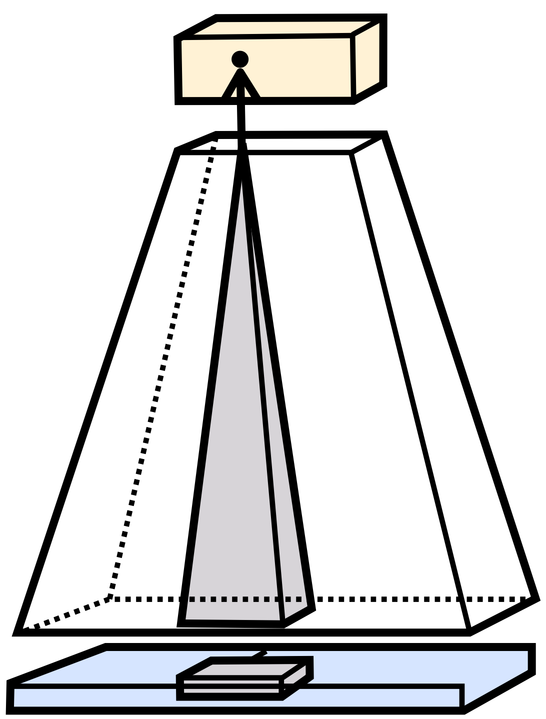
Downsampling filters

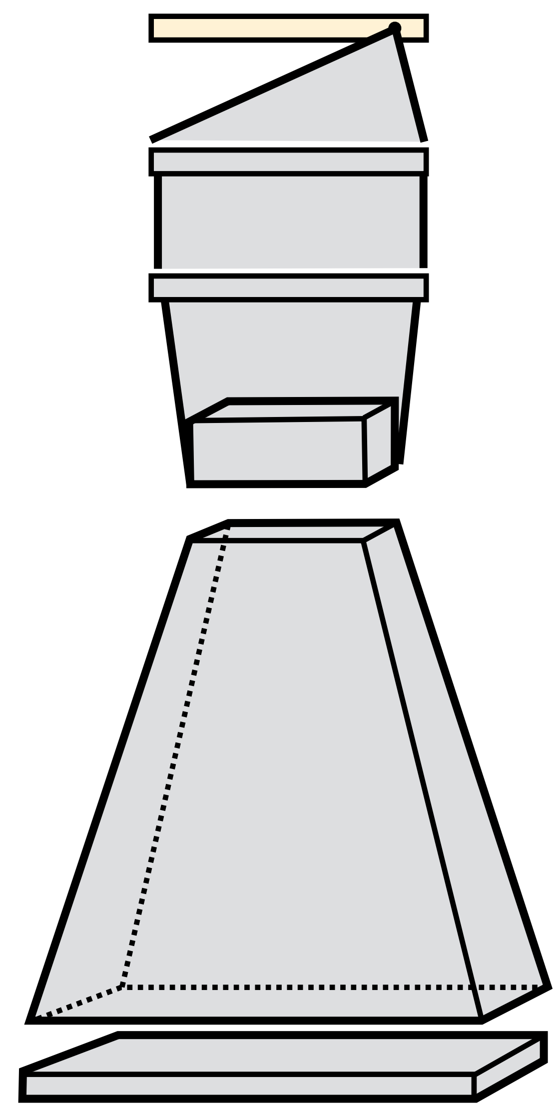
Responses are spatially selective, can be used to localize things.

Upsampling filters

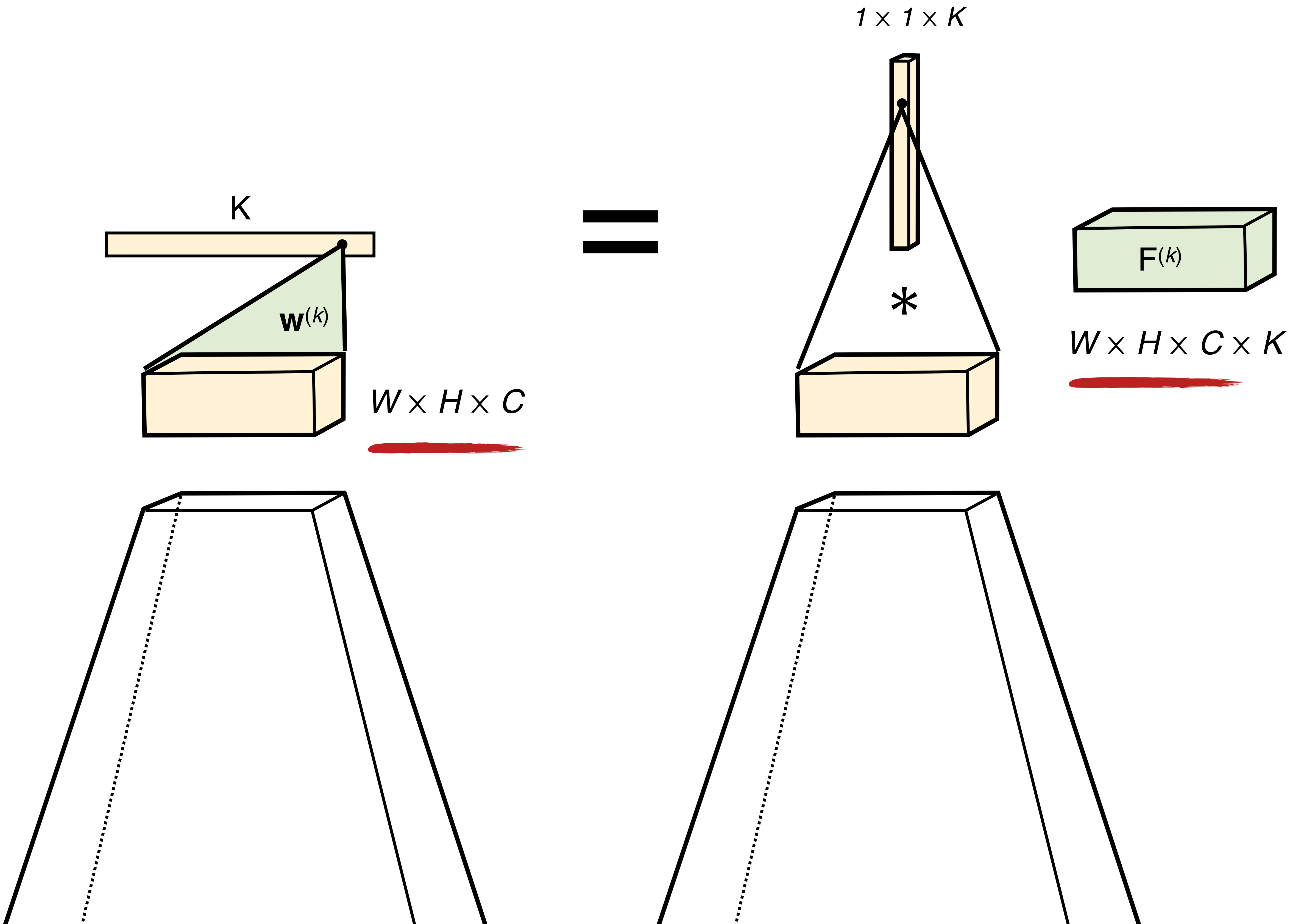
Responses are global, do not characterize well position

Which one is more useful for pixel level labelling?

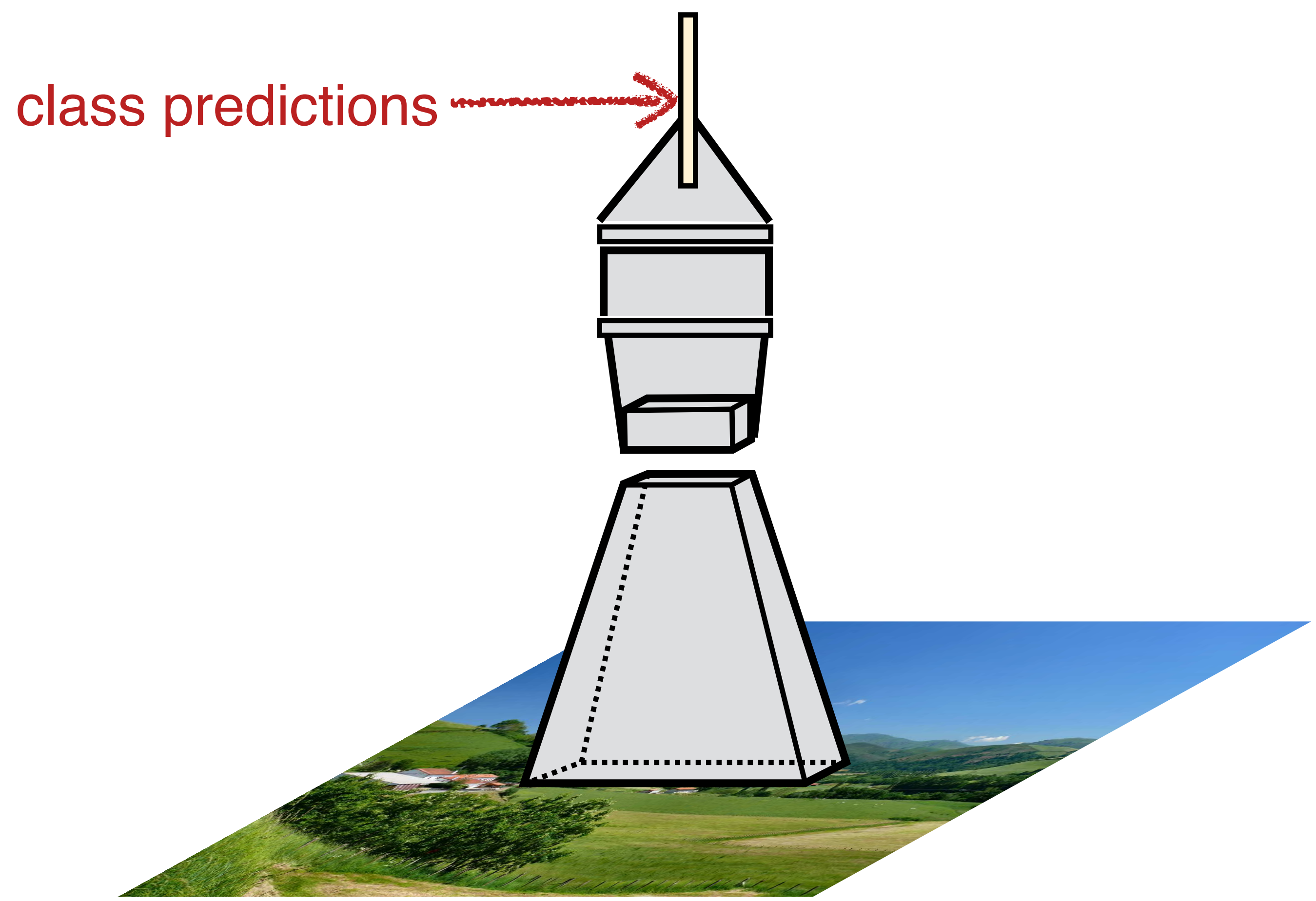




Fully-connected layer = large filter

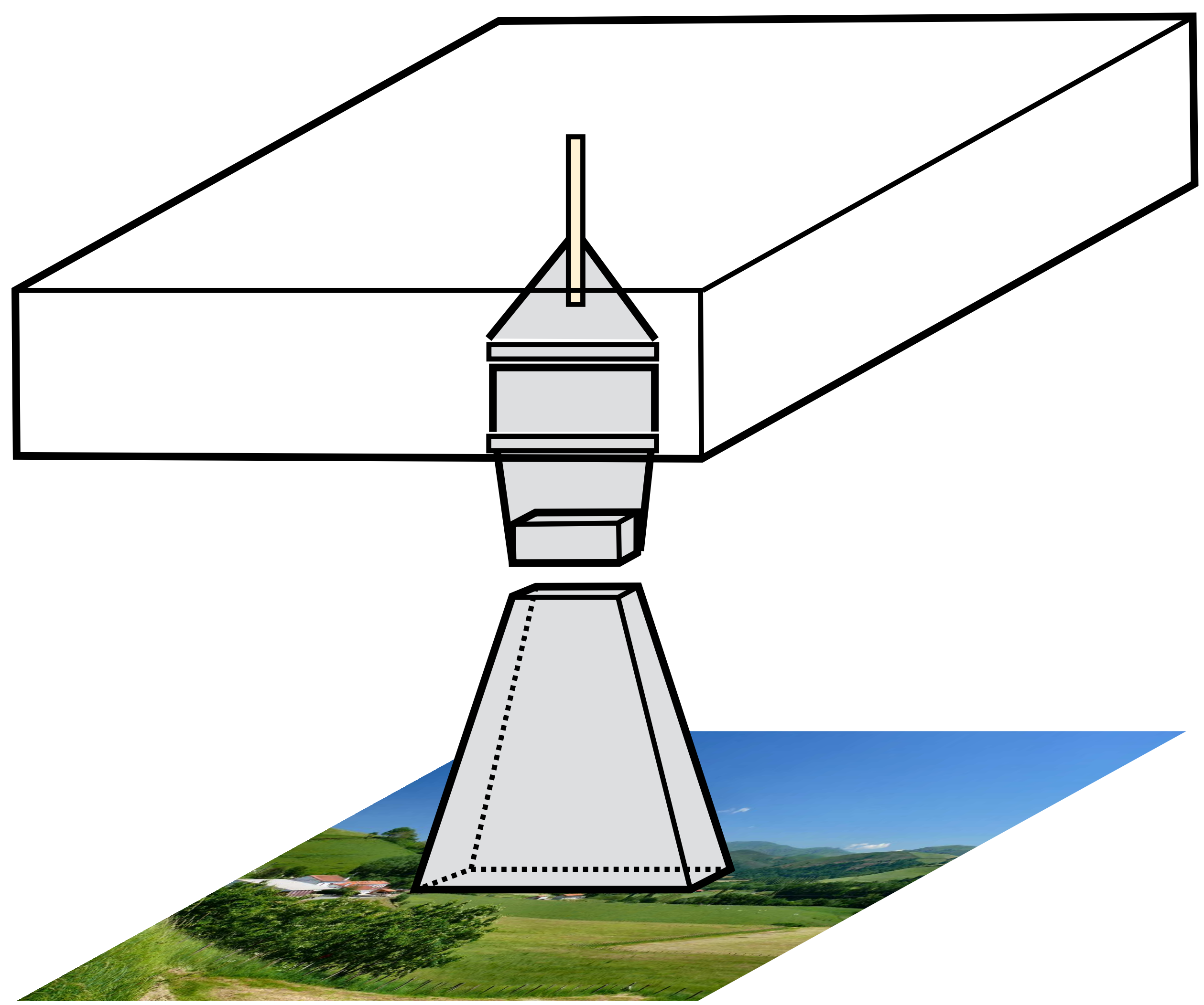


Fully-convolutional neural networks



J. Long, E. Shelhamer, and T. Darrell. *Fully convolutional models for semantic segmentation.* In Proc. CVPR, 2015

Fully-convolutional neural networks



Dense evaluation

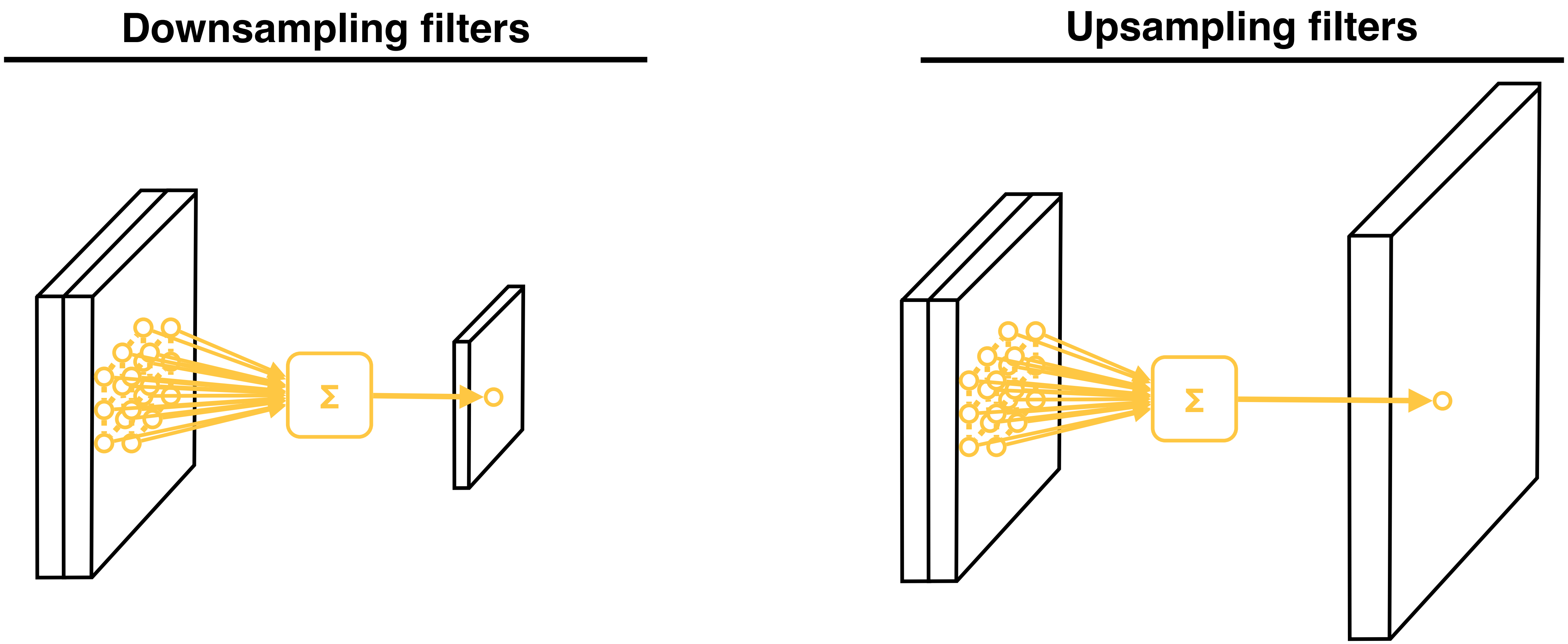
- Apply the whole network convolutional
- Estimates a vector of class probabilities at each pixel

Downsampling

- In practice most network downsample the data fast
- The output is very low resolution (e.g. 1/32 of original)

Upsampling the resolution

Interpolating filter



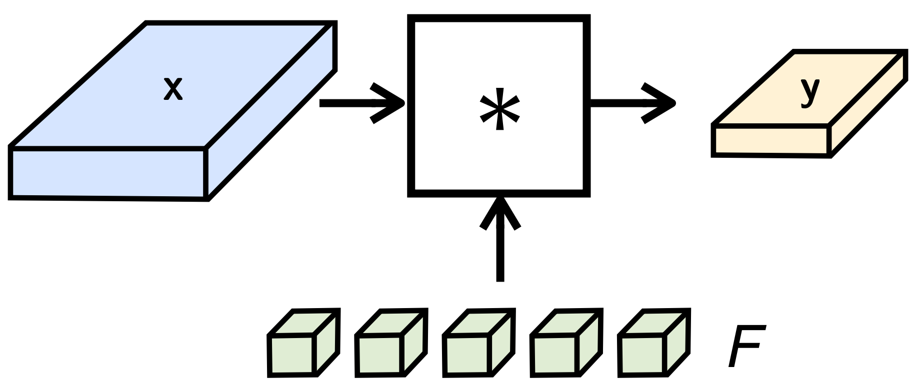
Upsampling filters allow to increase the resolution of the output

Very useful to get full-resolution segmentation results

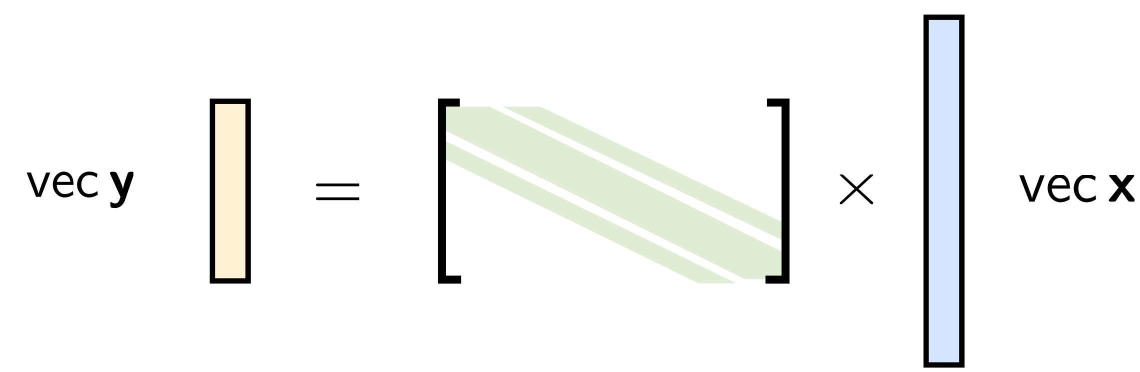
Deconvolution layer

Or convolution transpose

Convolution

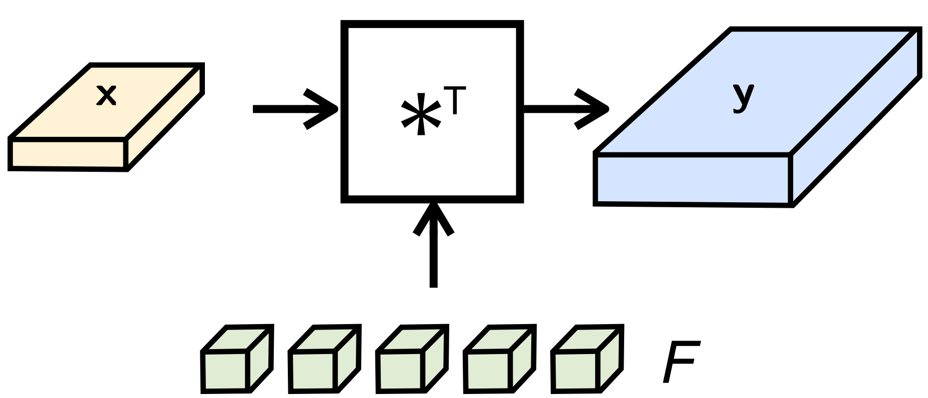


As matrix multiplication

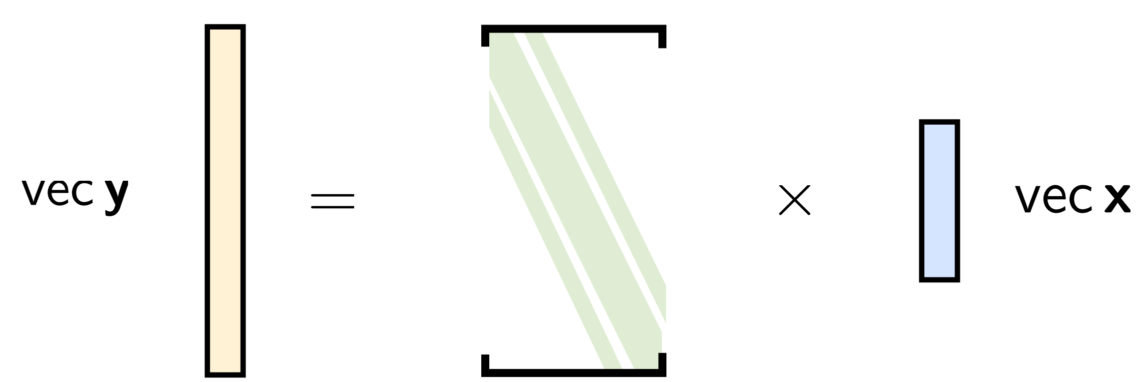


Banded matrix equivalent to F

Convolution transpose



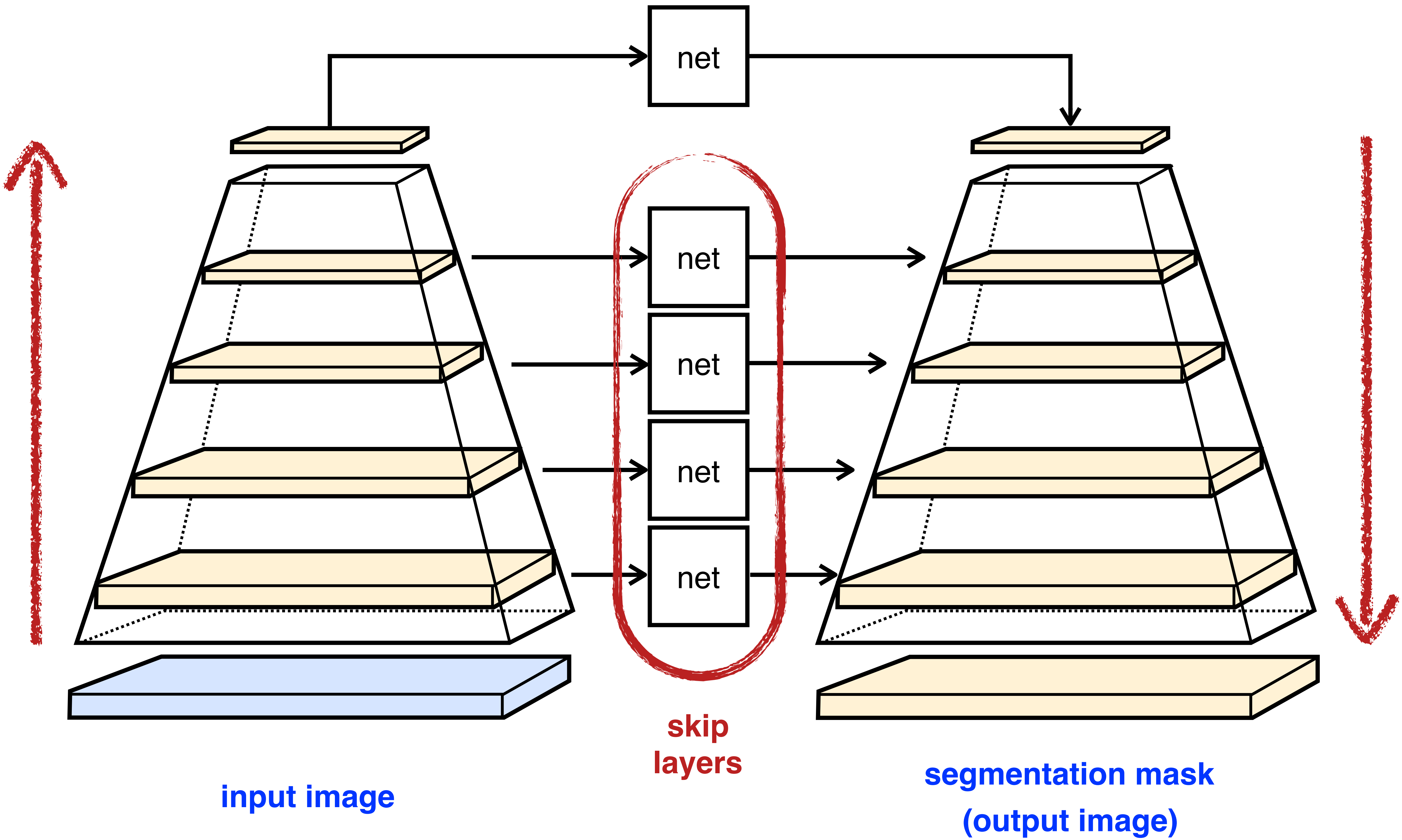
Transposed



Transposed matrix

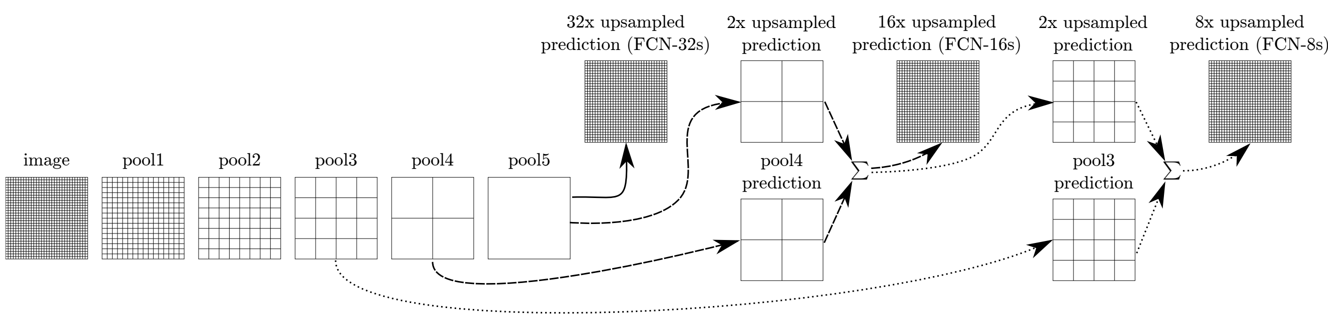
U-architectures

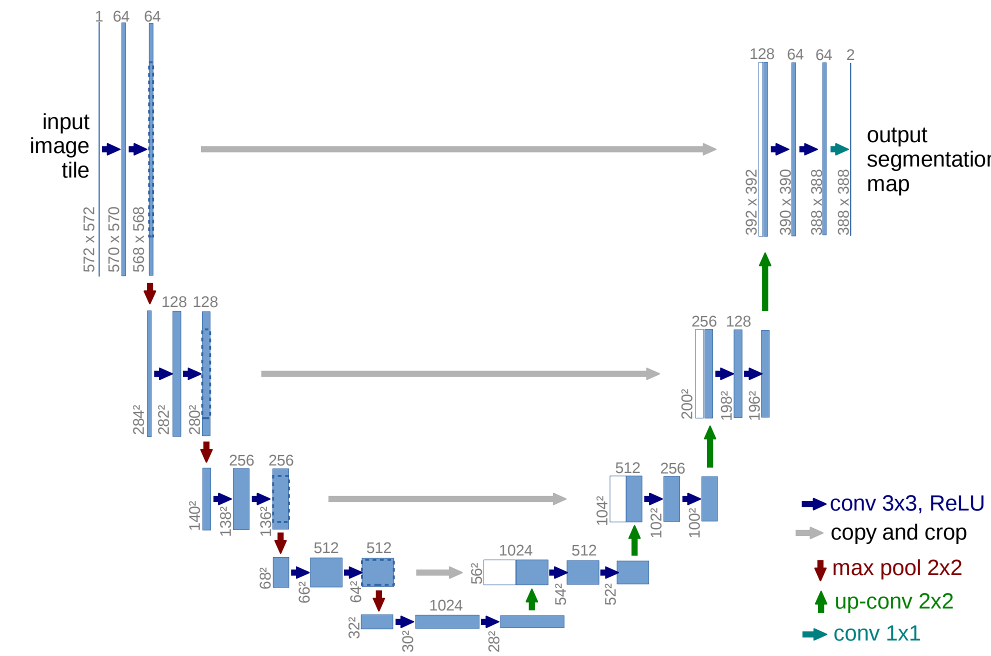
From image to image

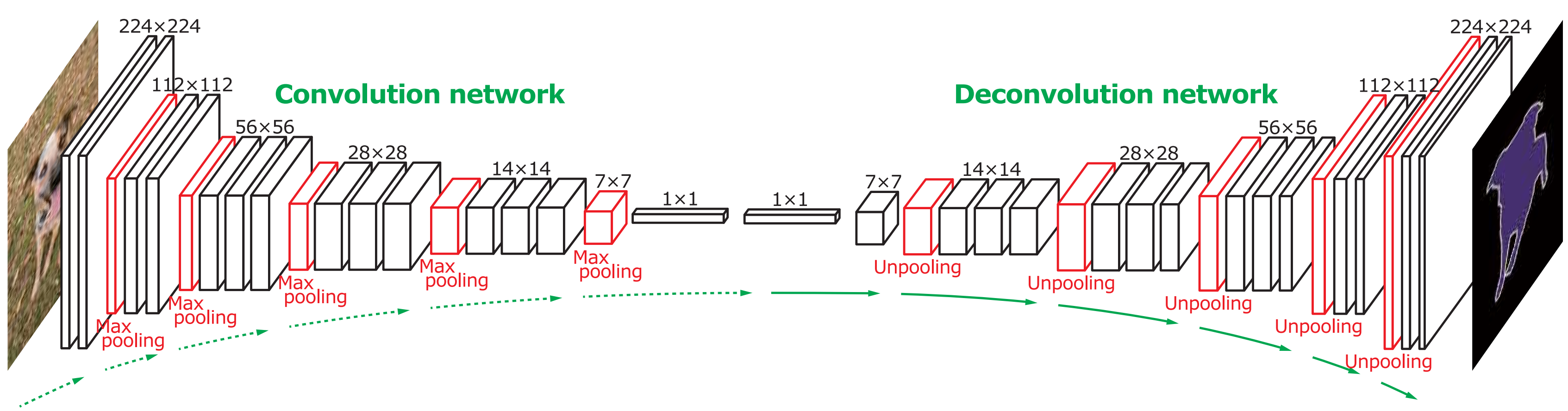


U-architectures

Several variants: FCN, U-arch, deconvolution, …



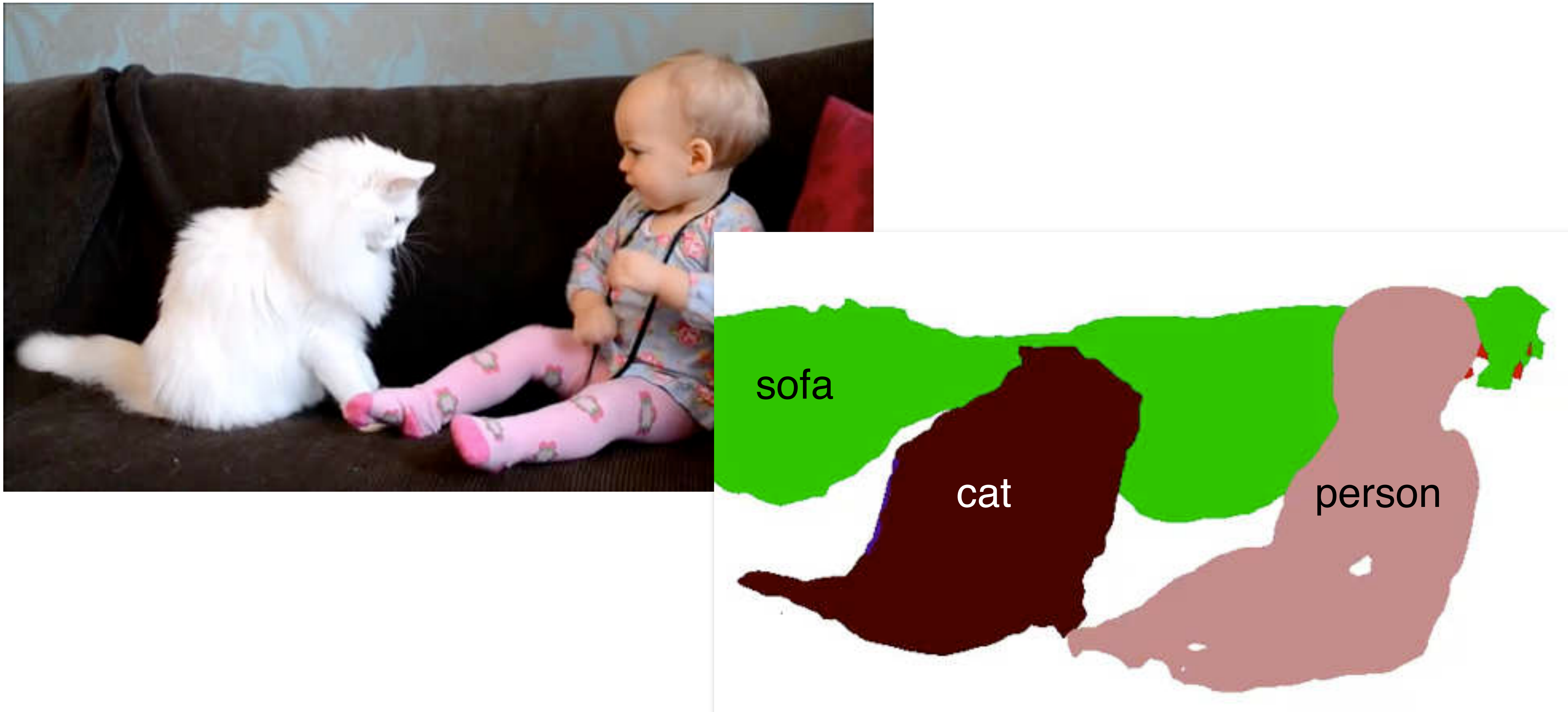




J. Long, E. Shelhamer, and T. Darrell. *Fully convolutional models for semantic segmentation.* In Proc. CVPR, 2015
H. Noh, S. Hong, and B. Han. *Learning deconvolution network for semantic segmentation.* In Proc. ICCV, 2015
O. Ronneberger, P. Fischer, and T. Brox. *U-net: Convolutional networks for biomedical image segmentation.* In Proc. MICCAI, 2015

Try it yourself: MatConvNet-FCN demo

Dense networks for semantic segmentation

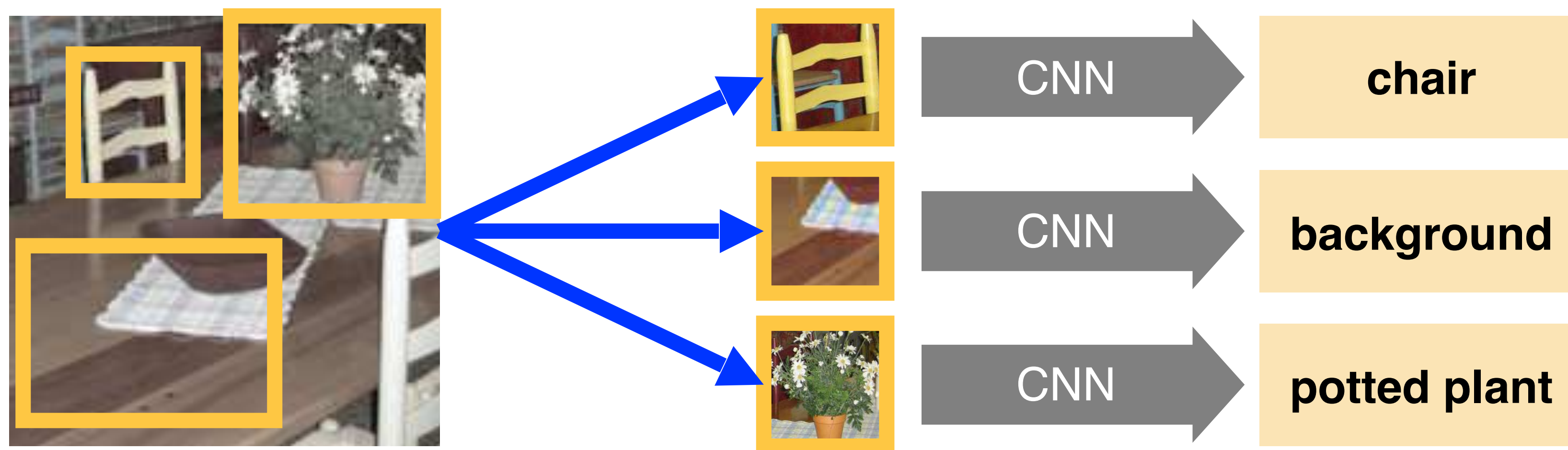


Object detection

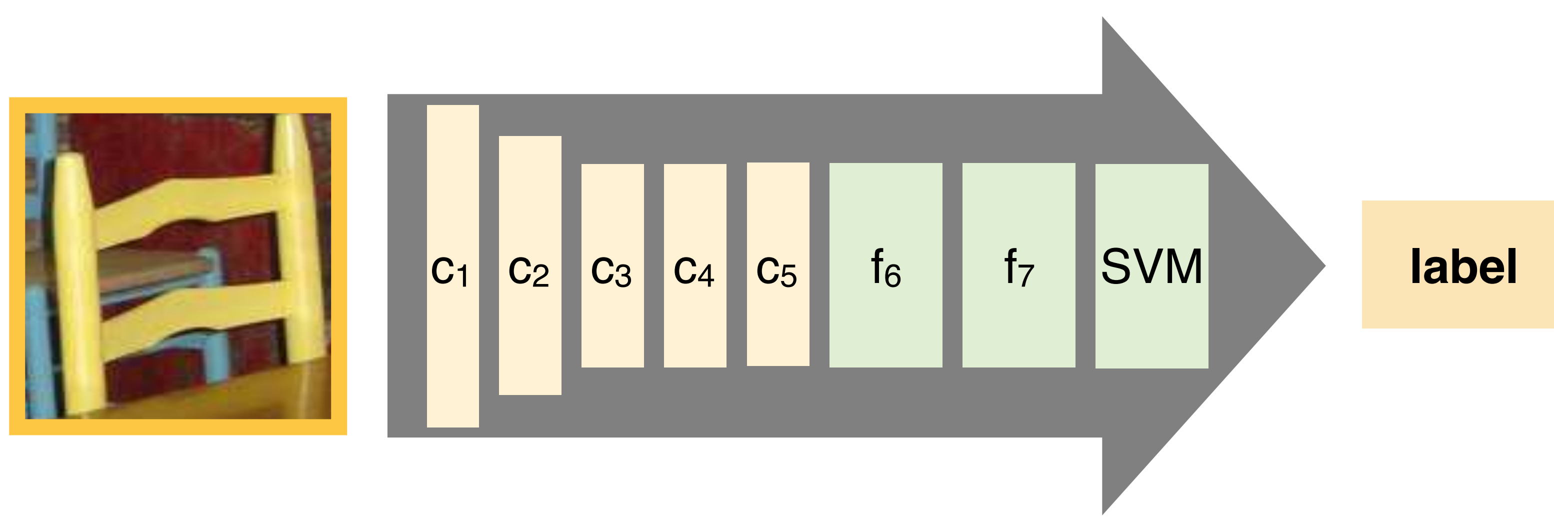
R-CNN

Region-based Convolutional Neural Network

Pros: simple and effective



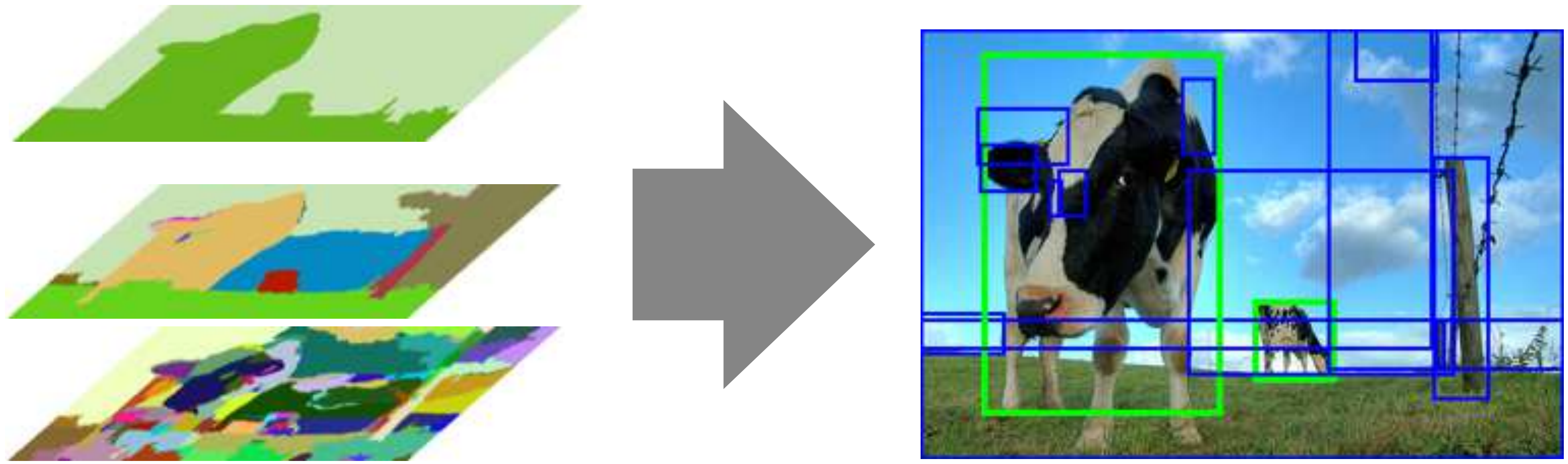
Cons: slow as the CNN is re-evaluated for each tested region



Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation
R. Girshick, J. Donahue, T. Darrell, J. Malik, CVPR 2014

Region proposals

Cut down the number of candidates



Proposal-method: Selective Search [van de Sande, Uijlings et al.]

- hierarchical segmentation
- each region generates a ROI
- ~ 2000 regions / image

From proposals to CNN features

Dilate, crop, reshape

Propose

Dilate

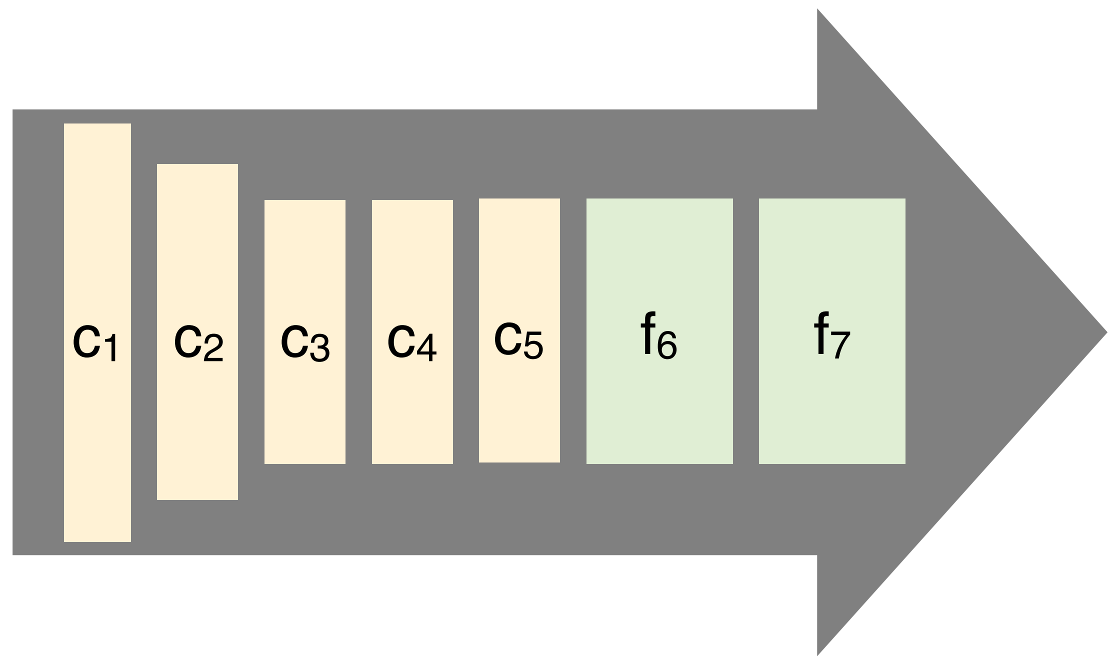
Crop & scale

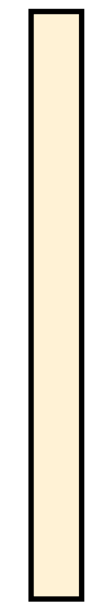
Anisotropic

227 x 227

From proposal to CNN features

Evaluate CNN





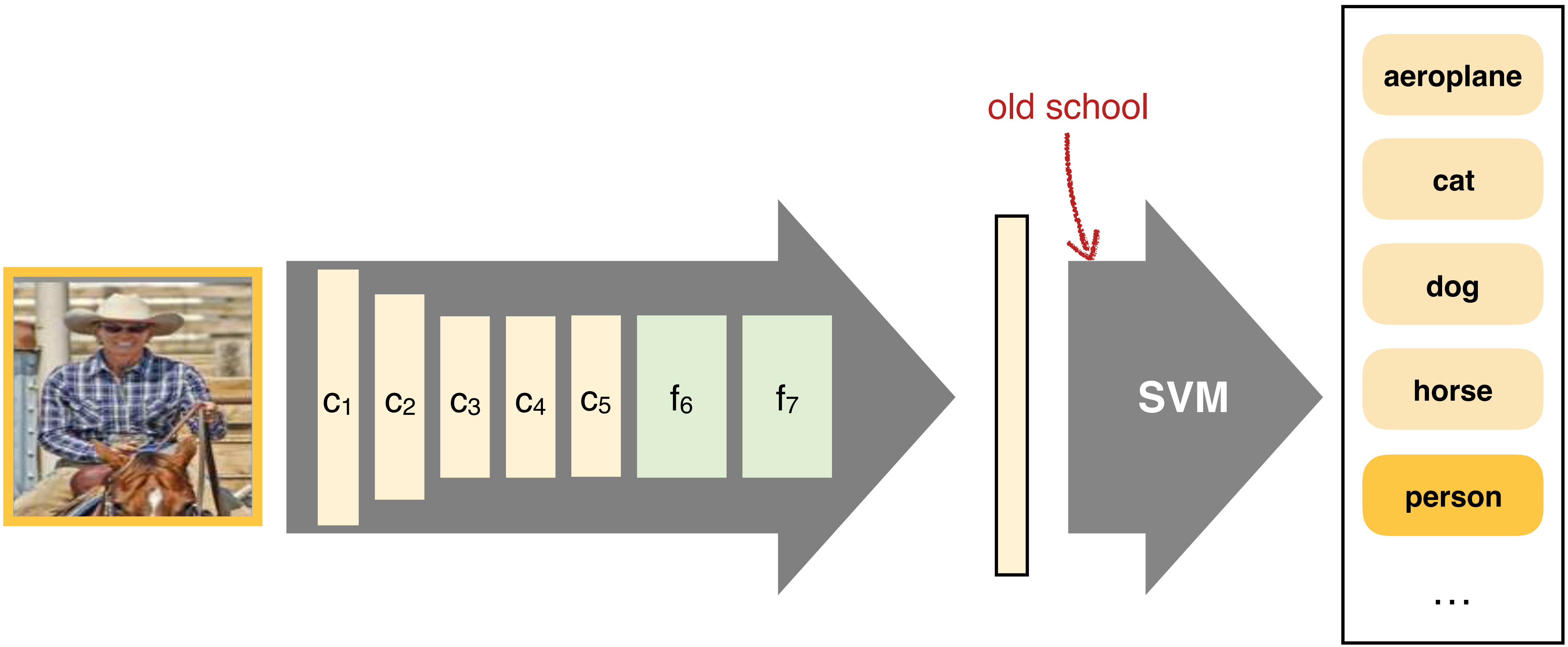
Scale
Anisotropic
227 x 227

CNN features
Up to FC-7
AlexNet

Feature vector
4096 D

Classification of a region

Run an SVM or similar on top



Scale
Anisotropic
227 x 227

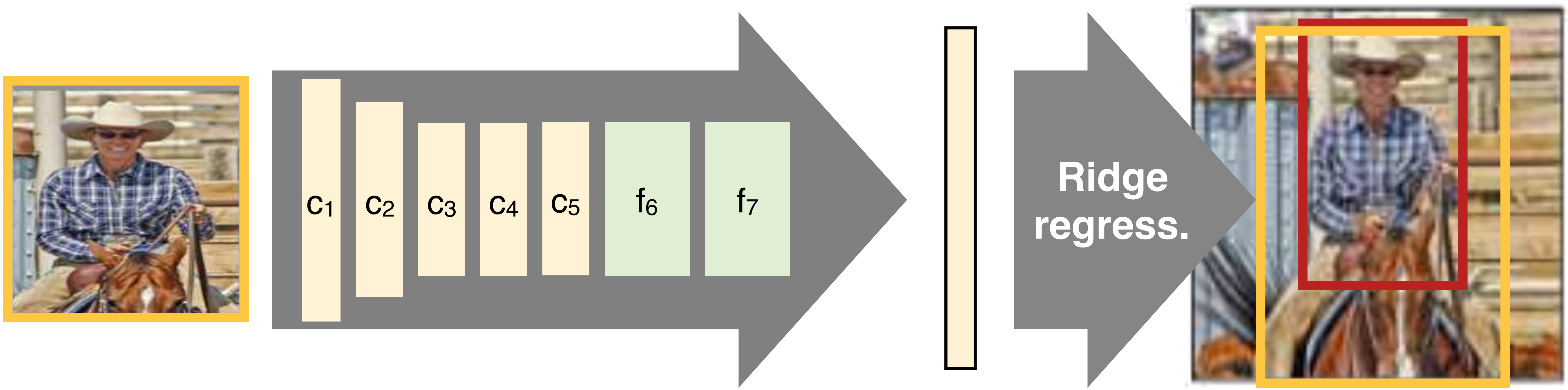
CNN features
Up to FC-7
AlexNet

Feature vector
4096 D

Label
One out of *N*

Region adjustment

Bounding-box regression



Scale
Anisotropic
227 x 227

CNN features
Up to FC-7
AlexNet

Feature vector
4096 D

Box adjustment
dx1, dx2, dy1, dy2

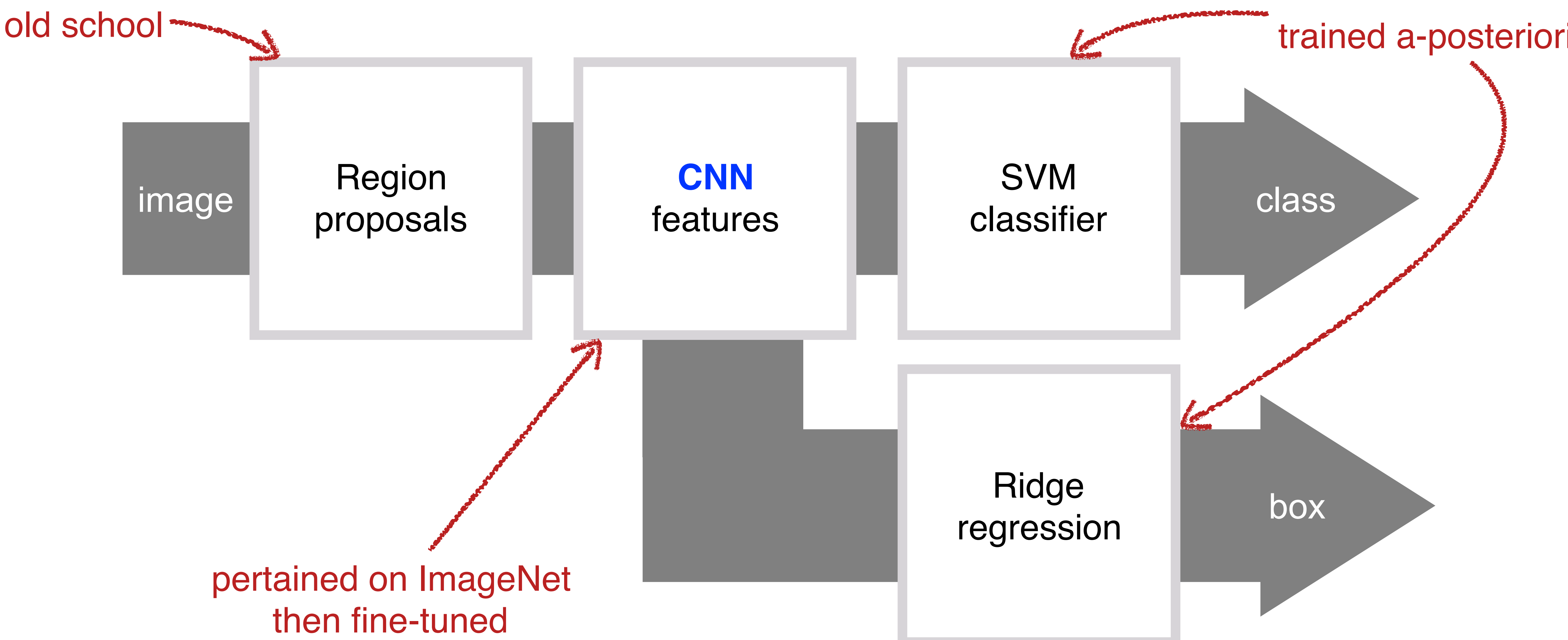
R-CNN results on PASCAL VOC

At the time of introduction (2013)

	VOC 2007	VOC 2010
DPM v5 (Girshick et al. 2011)	33.7%	29.6%
UVA sel. search (Uijlings et al. 2013)		35.1%
Regionlets (Wang et al. 2013)	41.7%	39.7%
SegDPM (Fidler et al. 2013)		40.4%
R-CNN (TorontoNet)	54.2%	50.2%
R-CNN (TorontoNet) + bbox regression	58.5%	53.7%
R-CNN (VGG-VD)	62.1%	
R-CNN (ONet) + bbox regression	66.0%	62.9%

R-CNN summary

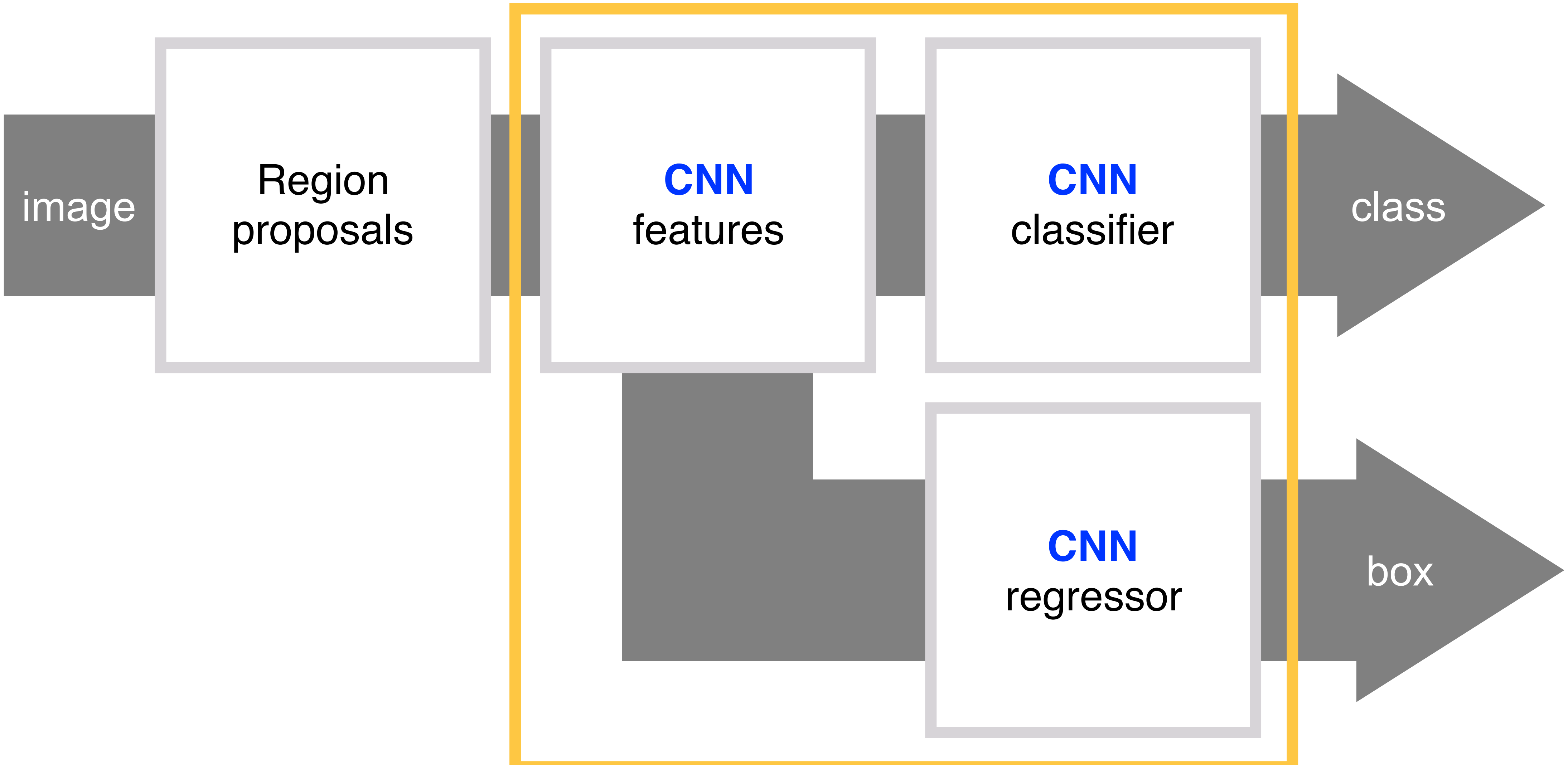
Region-based Convolutional Neural Network



Can we achieve end-to-end training?

Towards better R-CNNs

Region-based Convolutional Neural Network

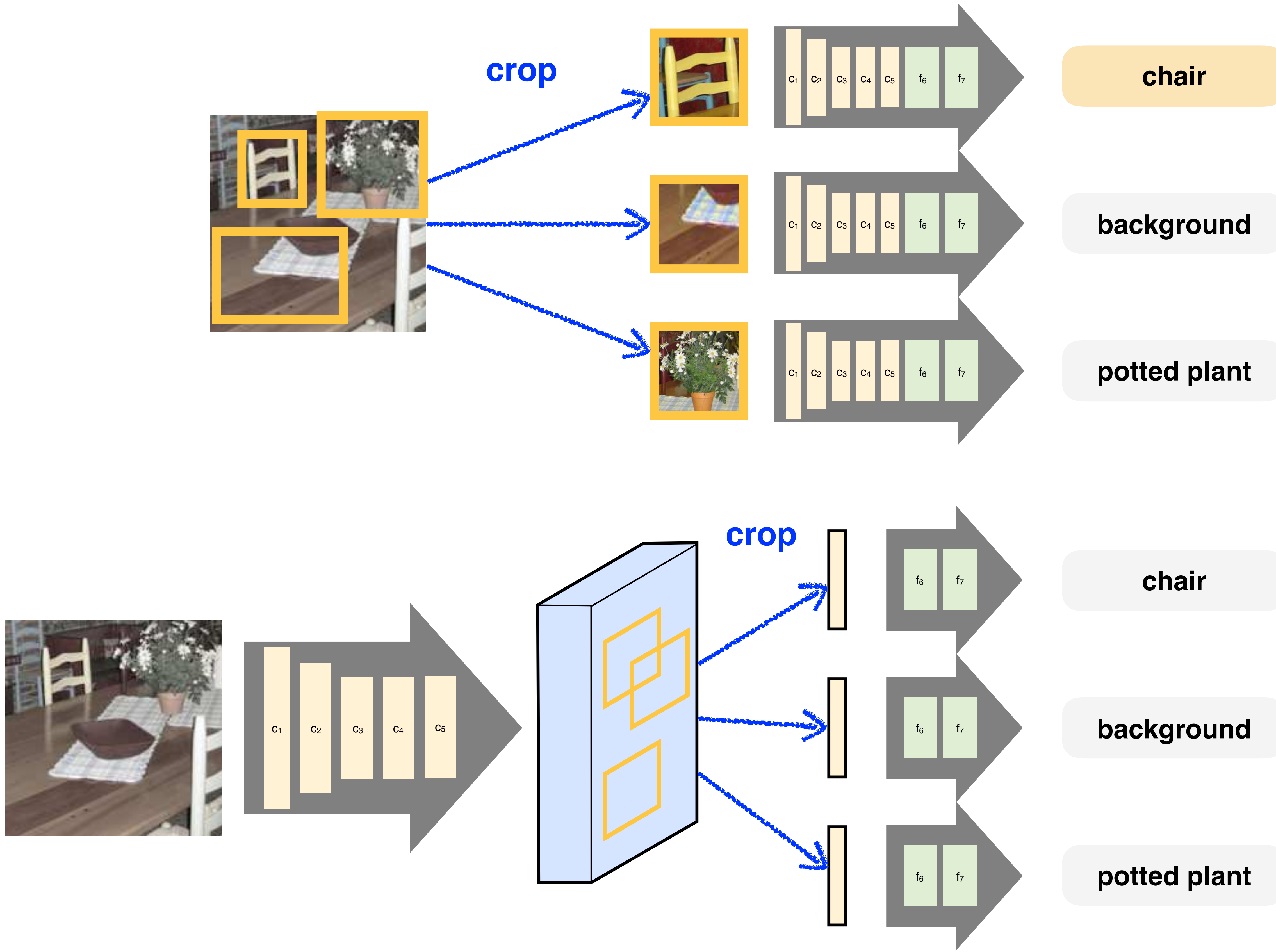


End-to-end training

Except for region proposals

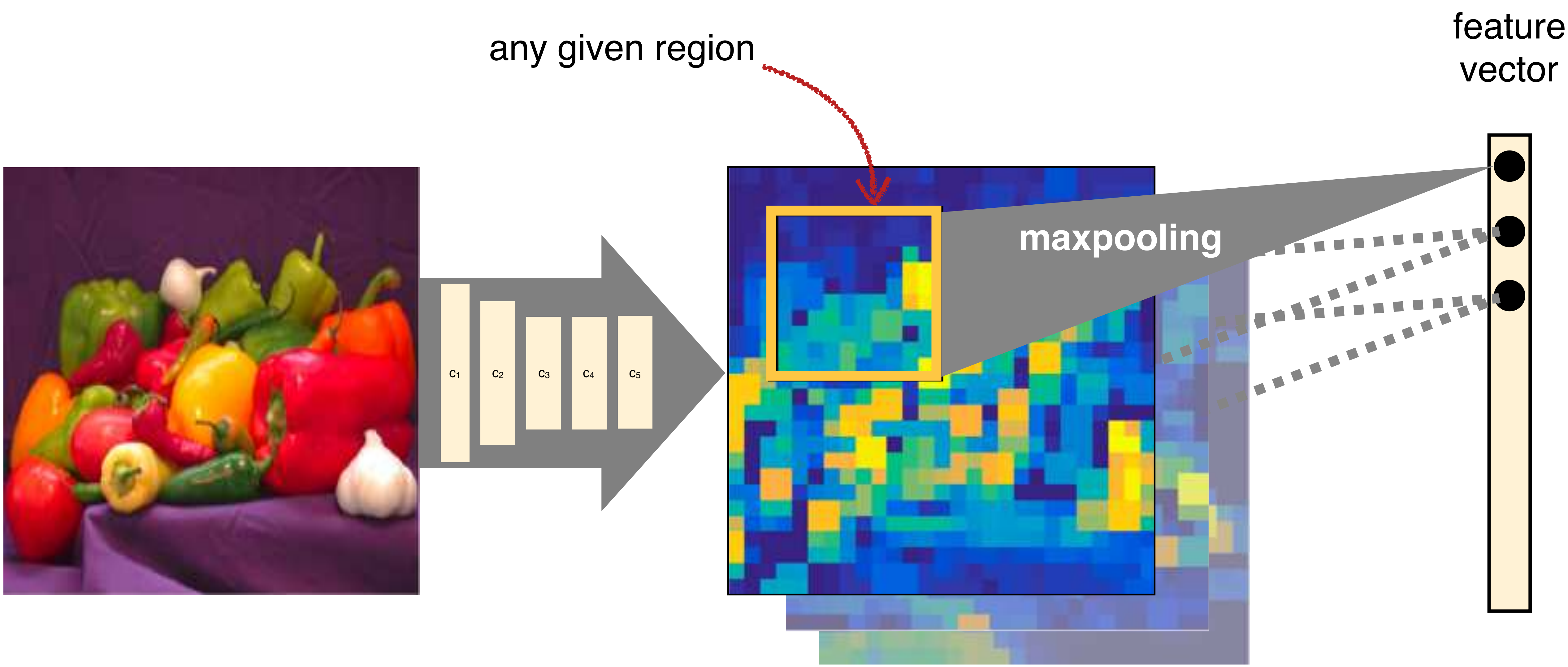
Problem: this is still pretty slow!

Accelerating R-CNN



The Spatial Pooling layer

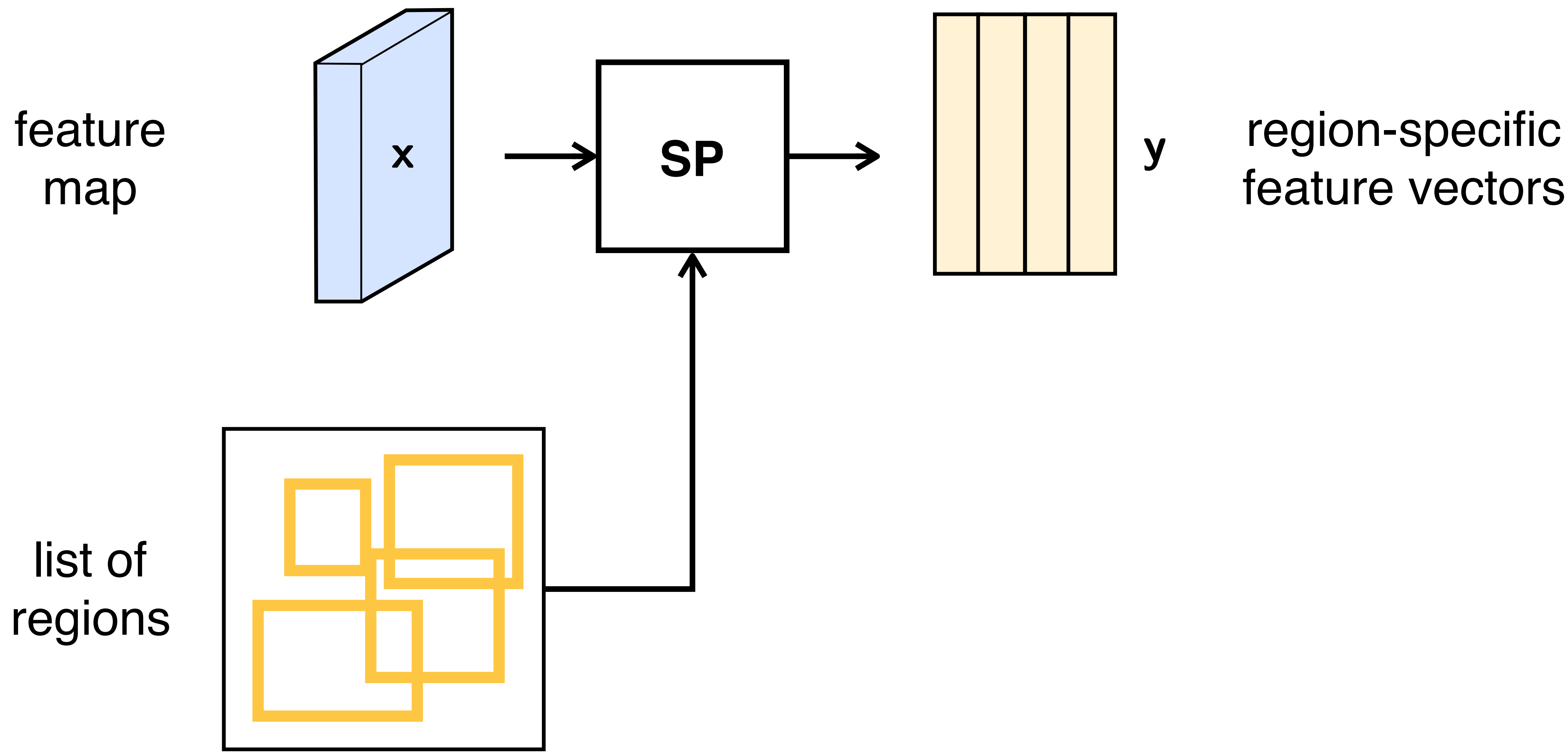
Max pooling in arbitrary regions



He, Zhang,Ren & Sun, “Spatial Pyramid Pooling (SPP) in Deep Convolutional Networks for Visual Recognition”, ECCV 2014

The Spatial Pooling layer

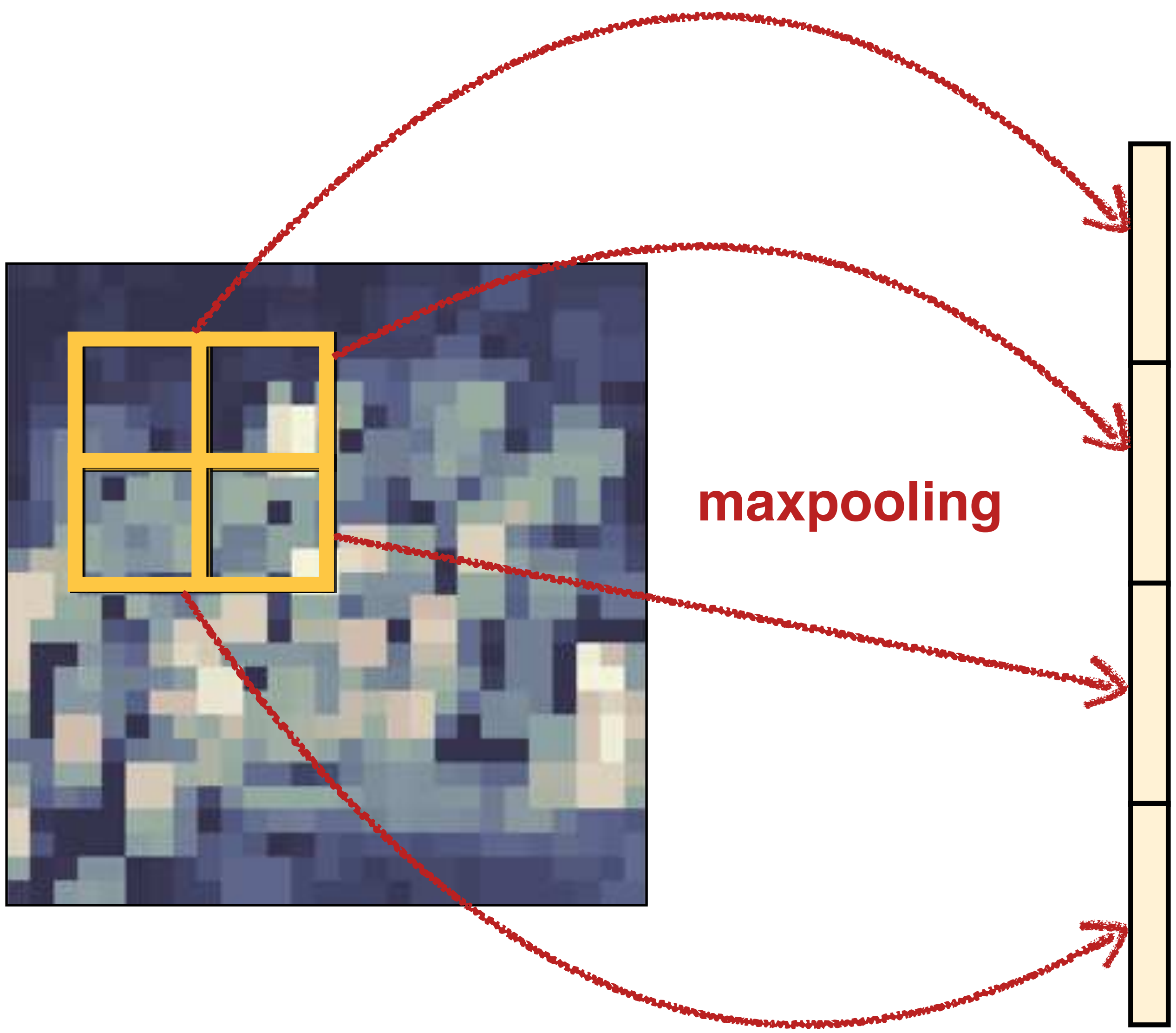
As a building block



He, Zhang,Ren & Sun, “Spatial Pyramid Pooling (SPP) in Deep Convolutional Networks for Visual Recognition”, ECCV 2014

The Spatially ***Pyramid*** Pooling Layer

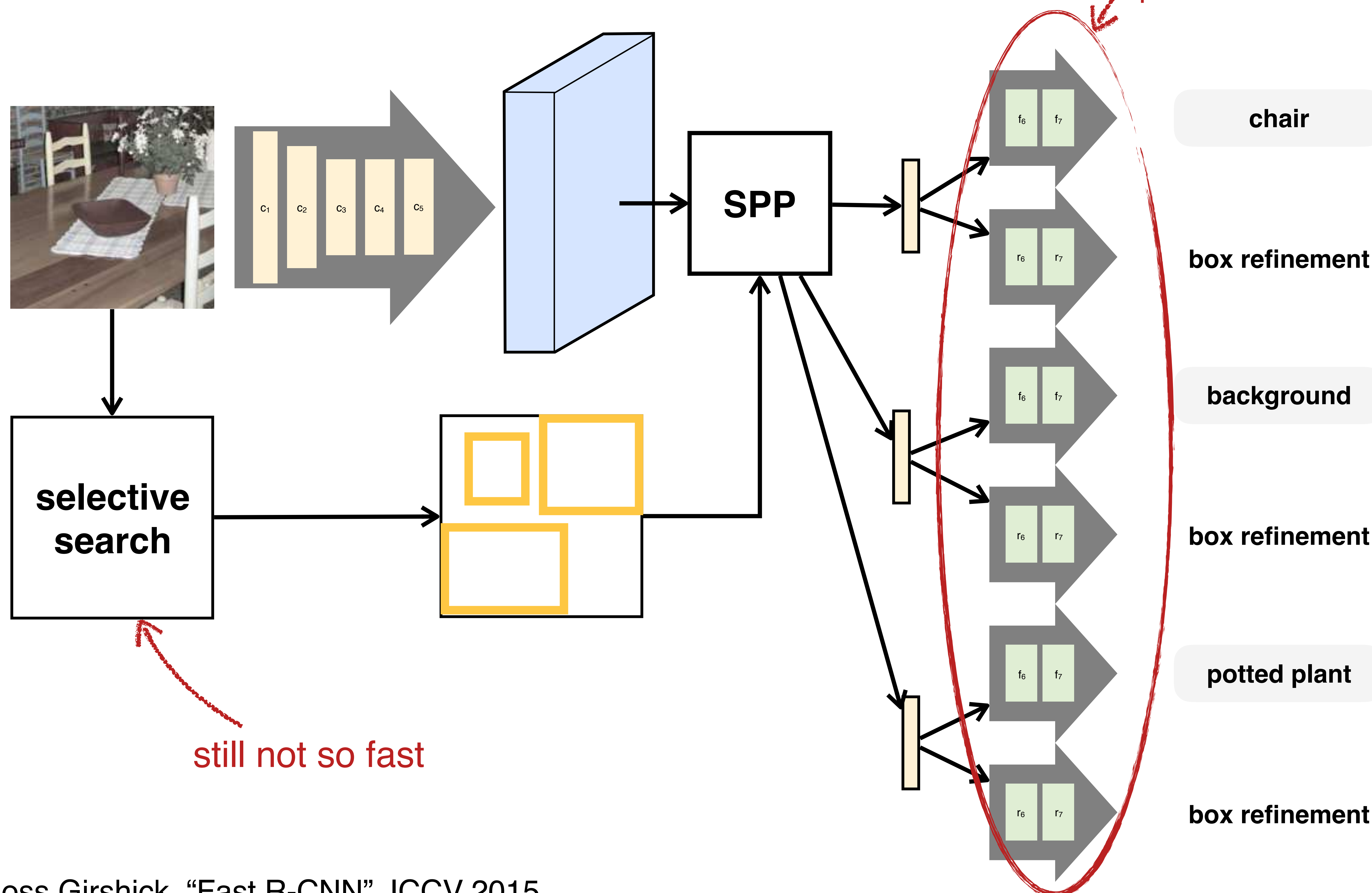
Same as above, but for multiple subdivisions



Fast R-CNN

Summary

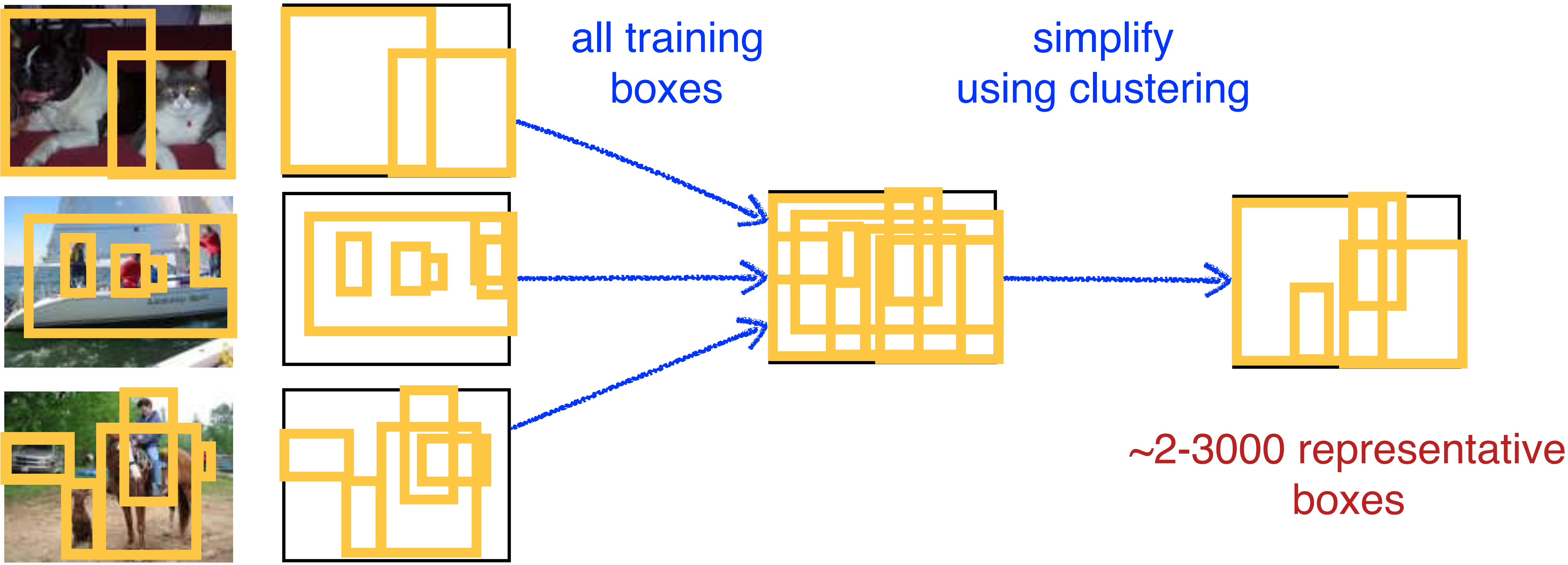
same
parameters



Ross Girshick. “Fast R-CNN”. ICCV 2015

R-CNN minus R

Fixed image-independent proposal set



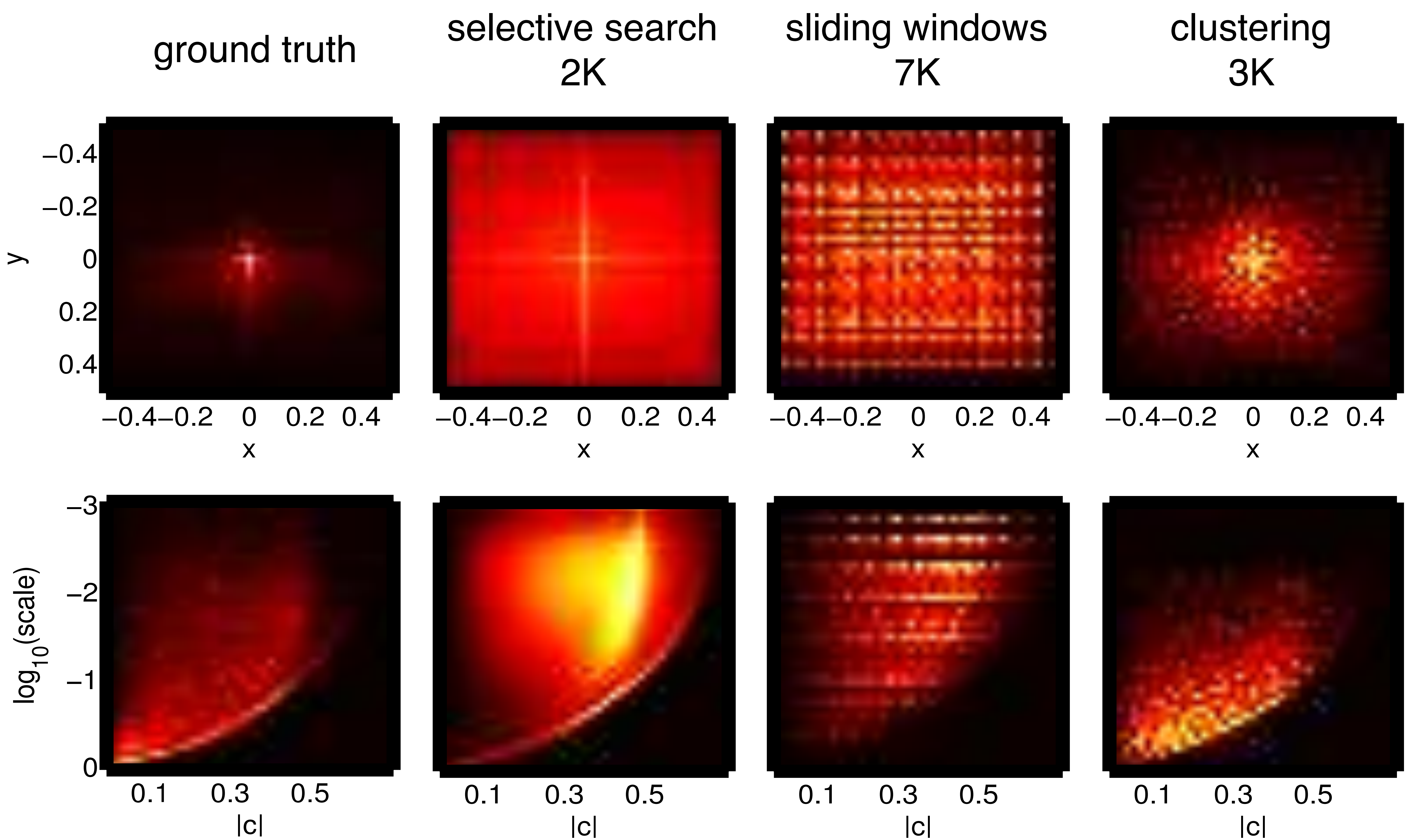
Fixed proposal generation

- Take all bounding box in the training set
- Run K-means clustering to distill a few thousands

[Lenc Vedaldi BMVC 2015]

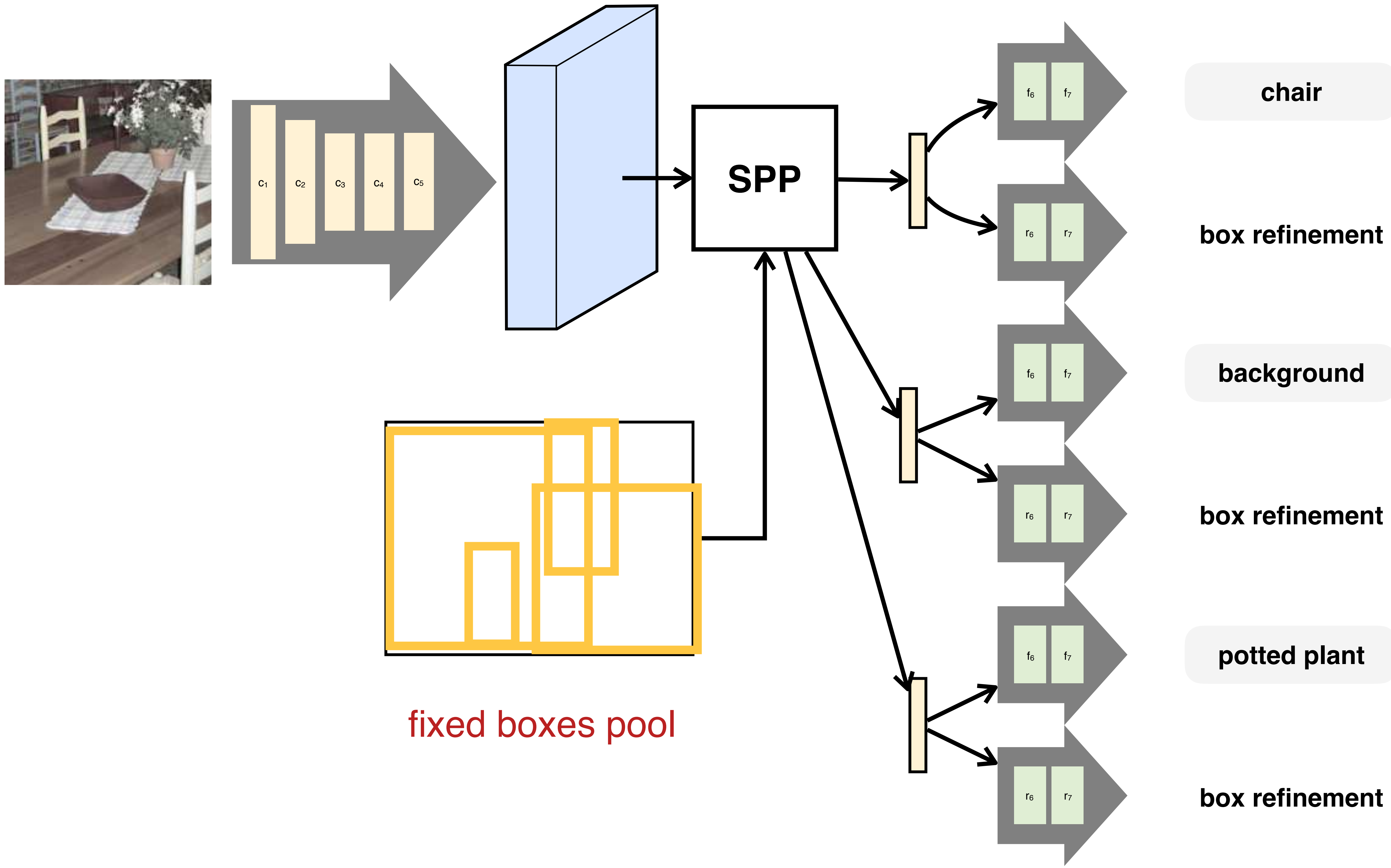
Vs other proposal sets

Matches the training set statistics by construction



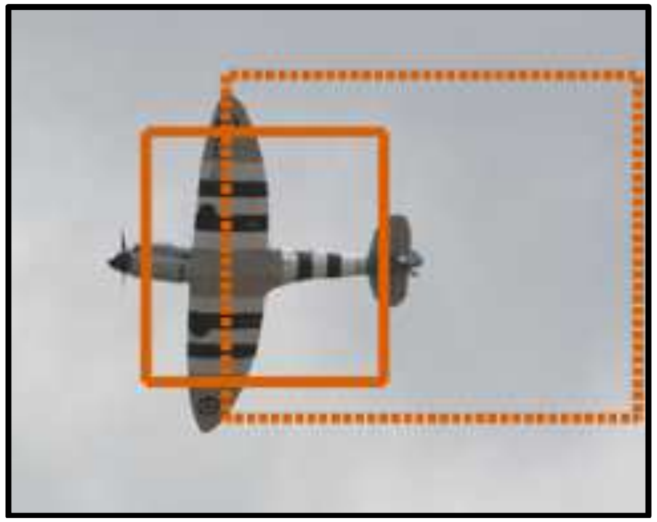
R-CNN minus R

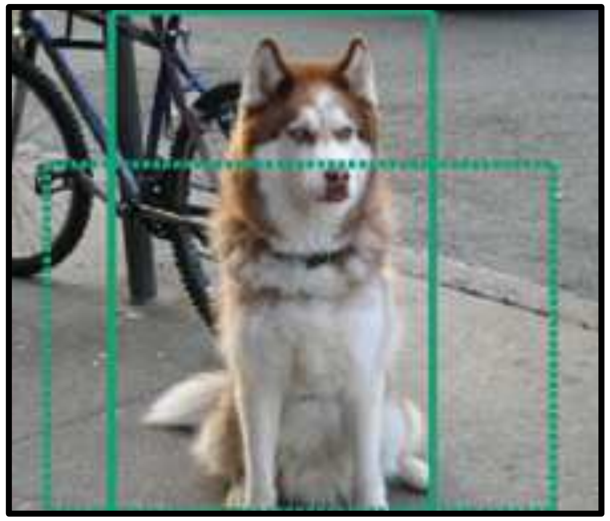
Replace image-specific boxes with a fixed pool



Why does it work?

Answer: regression is quite powerful

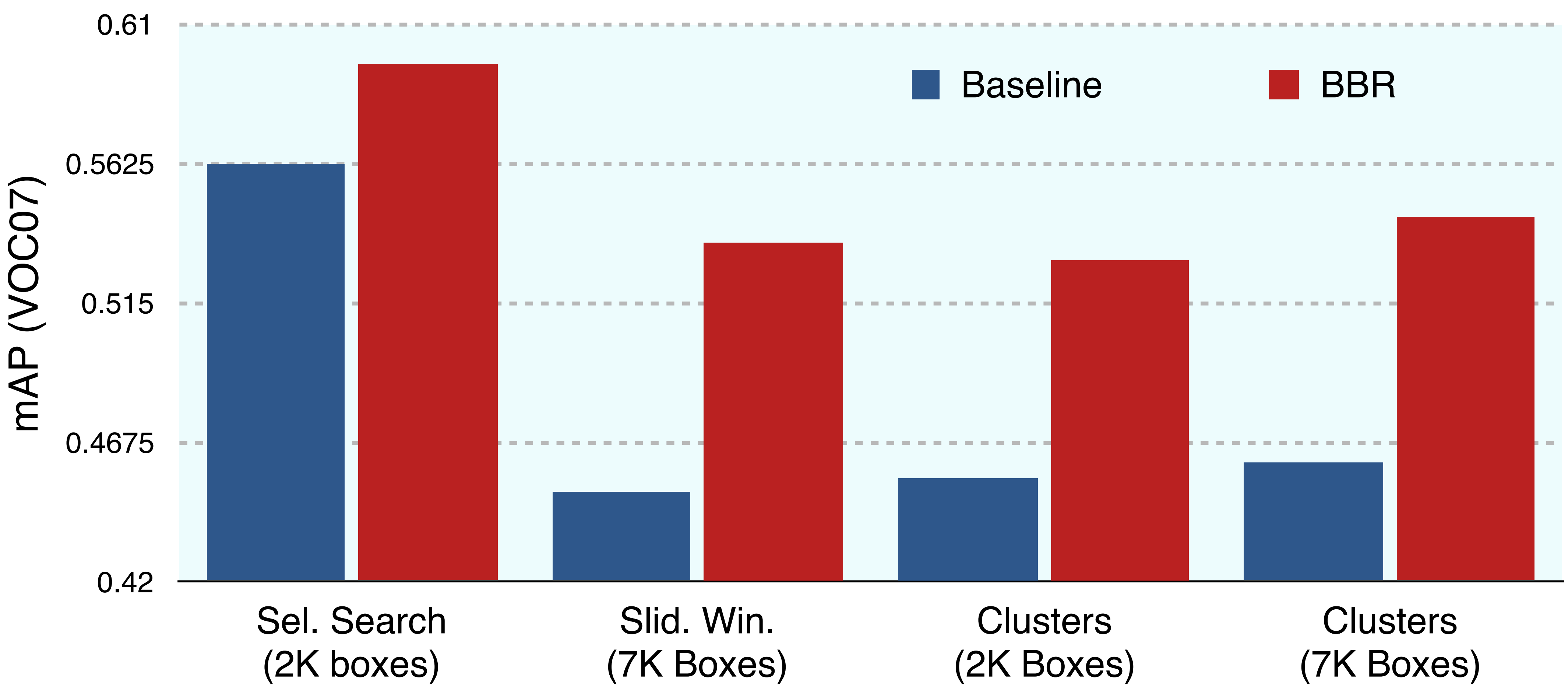



Dashed line: initial **Solid line**: corrected by the CNN

Quantitative comparisons

Image-specific vs fixed



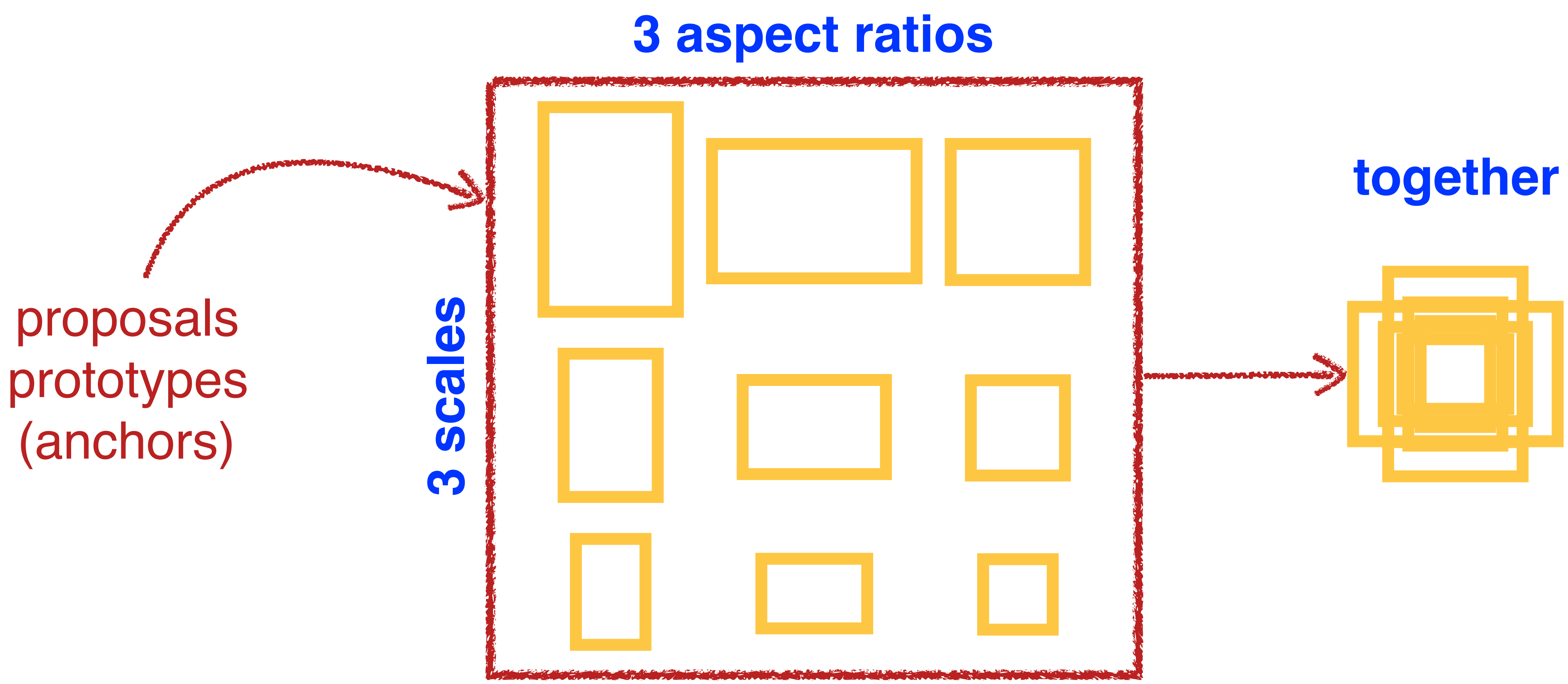
Selective search is much better than fixed generators

However, bounding box regression almost eliminates the difference

Clustering allows to use significantly less boxes than sliding windows

Faster R-CNN

Even better performance with fixed proposals



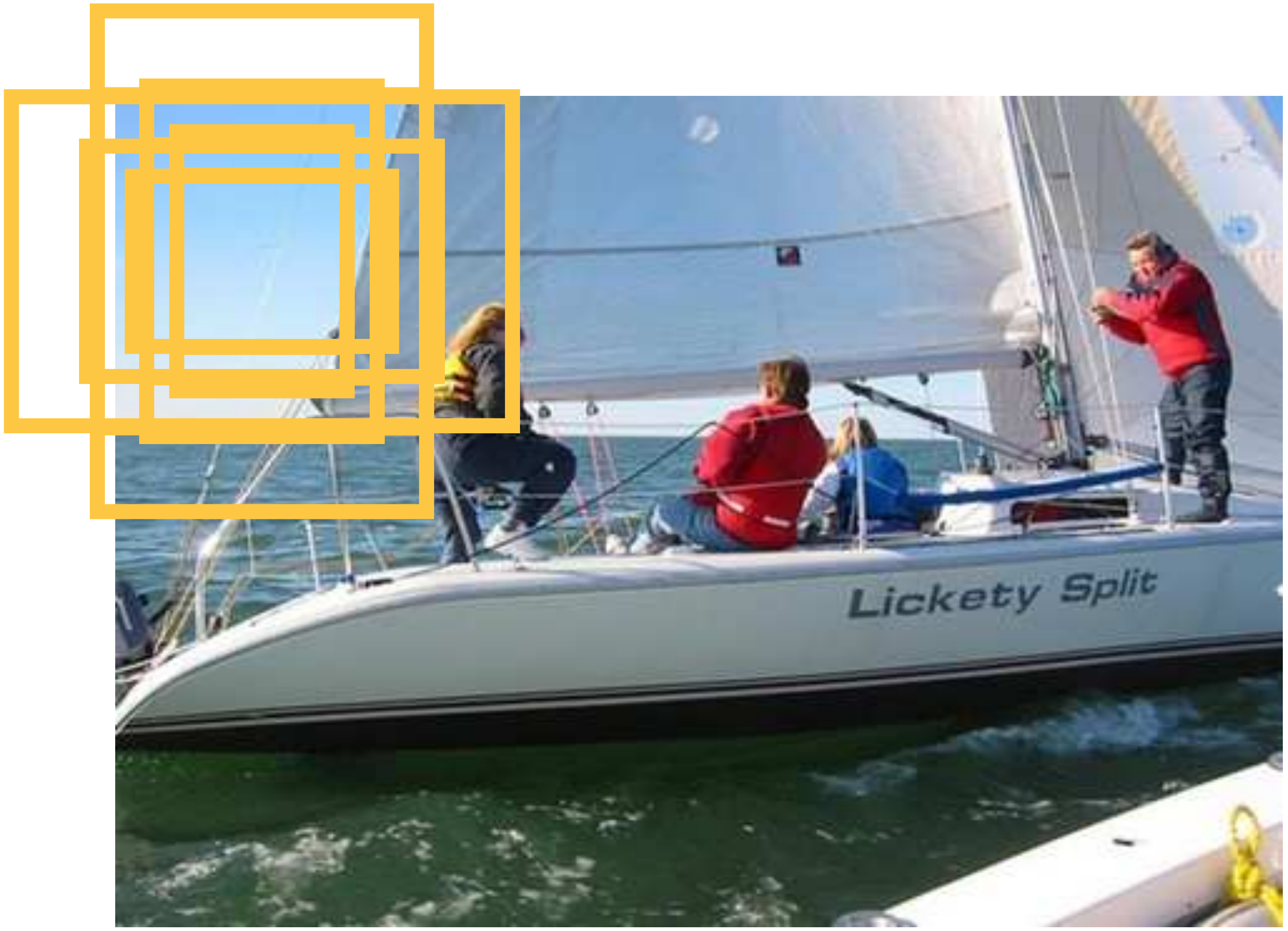
Ideas:

- Better fixed region proposal sampling
- Proposal shape specific classifier / regressors

Ren, He, Girshick, & Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". NIPS 2015.

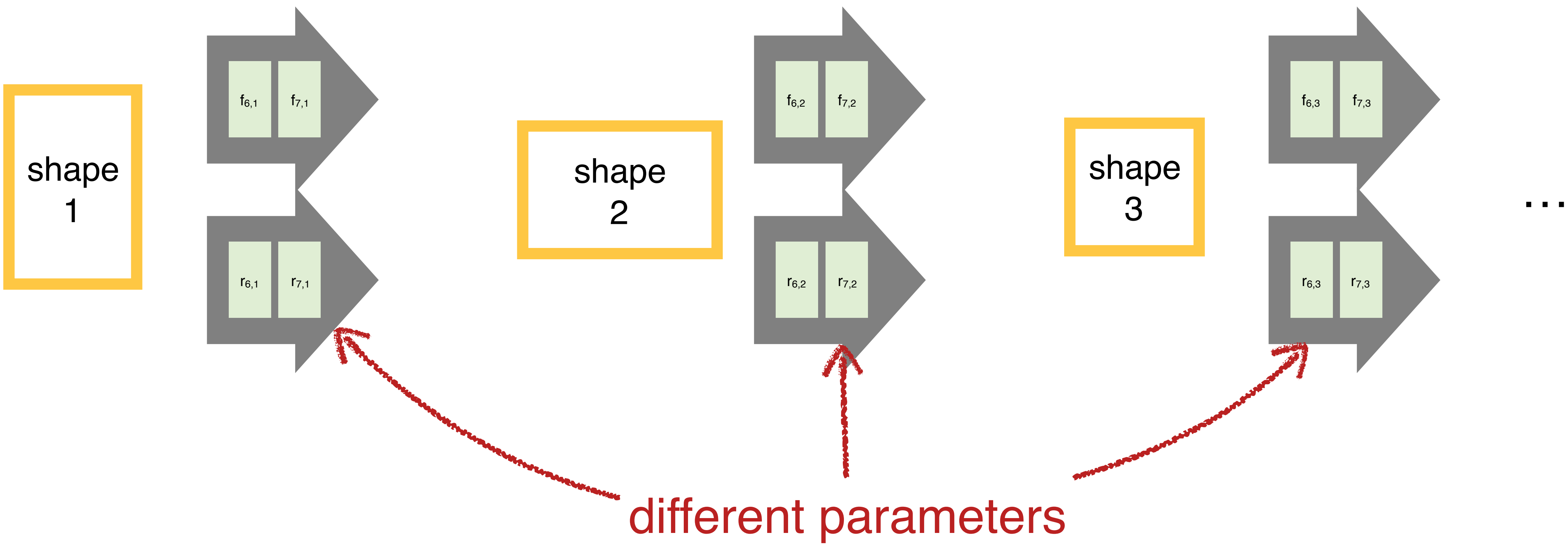
Faster R-CNN

Even better performance with fixed proposals



Ren, He, Girshick, & Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. NIPS 2015.

Shape-specific classifiers / regressors



Model parameters: translation invariant but shape/scale specific

Object aspects are learned by brute force

Ren, He, Girshick, & Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". NIPS 2015.

Training: what is a positive or negative box?

Based on overlap with ground truth

treat as **positive**

overlap > 70%

treat as **negative**

overlap < 30%

Ren, He, Girshick, & Sun. “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. NIPS 2015.

Fast and Faster R-CNN performance

Better, faster!

Method	Time / image	mAP (%)
R-CNN	~50s	66.0
Fast R-CNN	~2s	66.9
Faster R-CNN	198ms	69.9

Detection mAP on PASCAL VOC 2007, with VGG-16 pre-trained on ImageNet.

Multi-scale representations

Three strategies

scale image

scale feature filters

fixed scale features

model parameters shared for all scales

recompute features for each scale

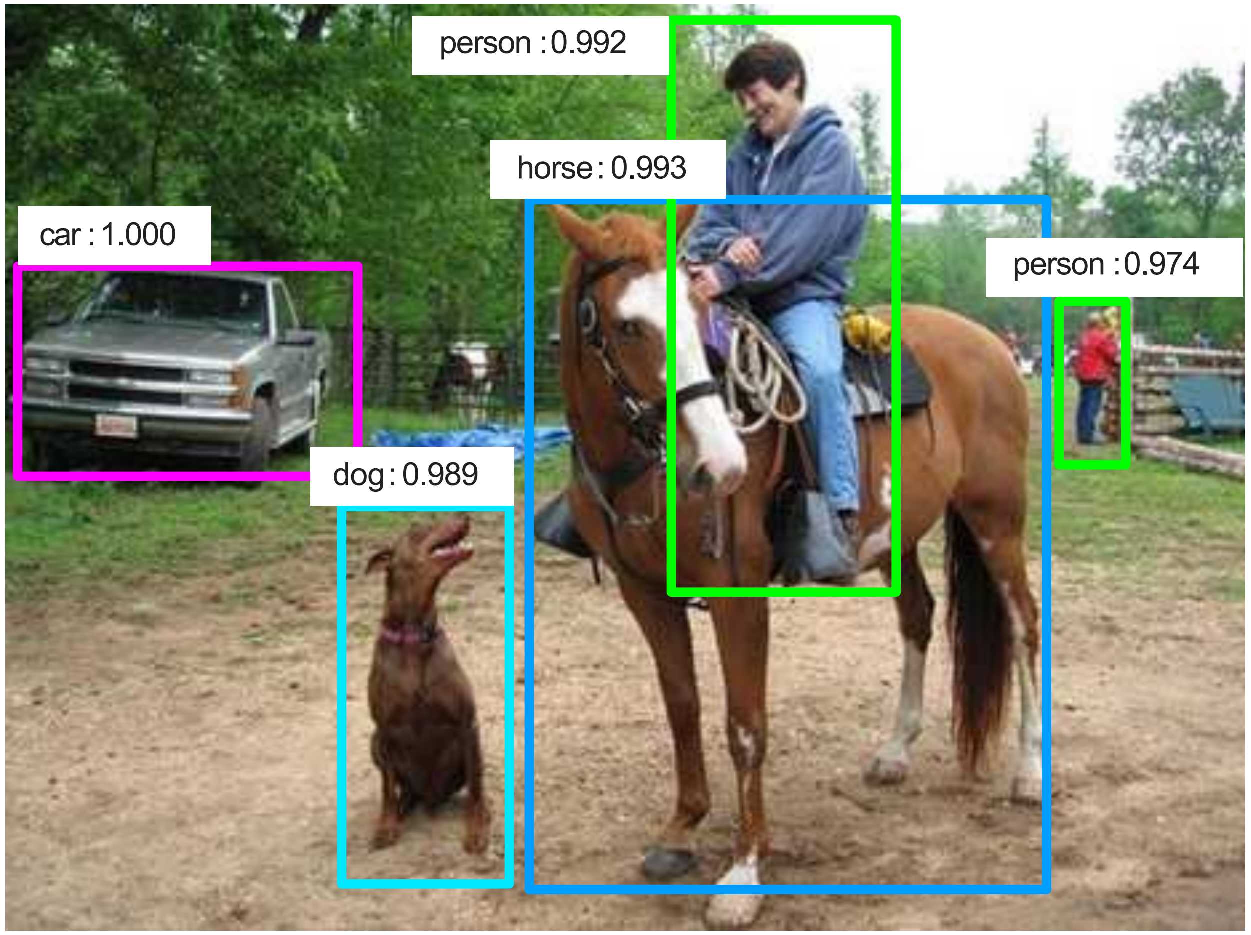
cannot exploit fine details when visible

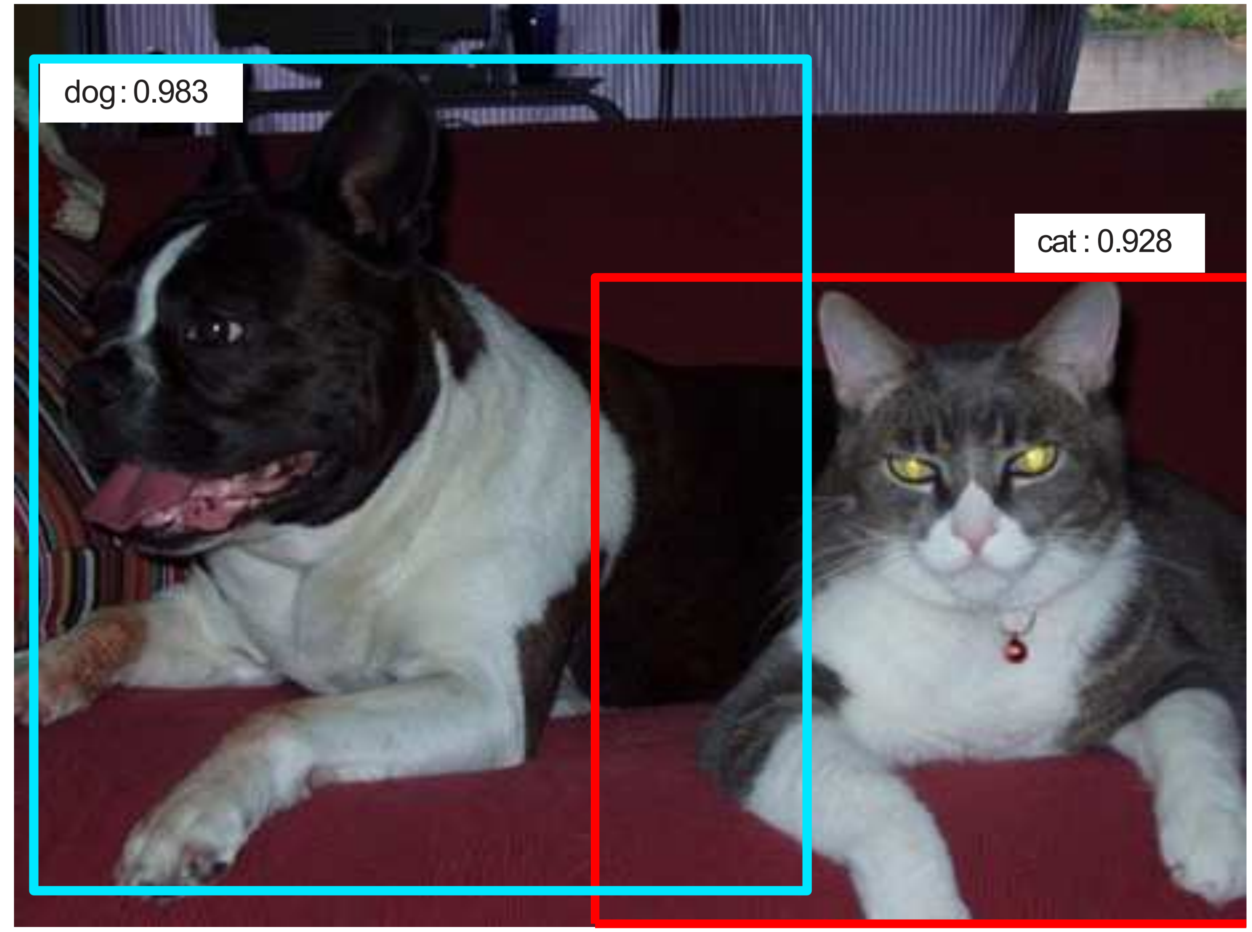
"brute force" modeling of scale

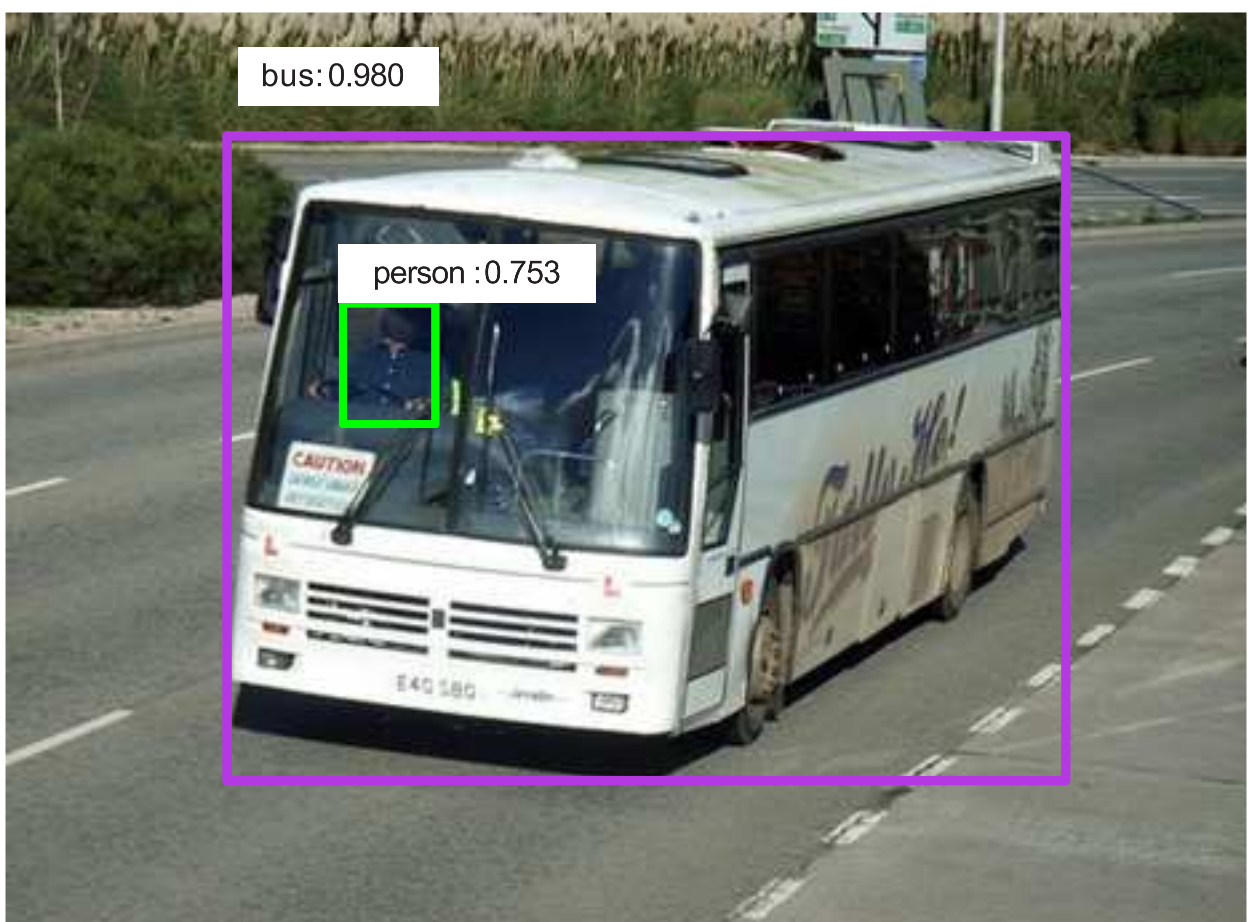
compute features at a single scale

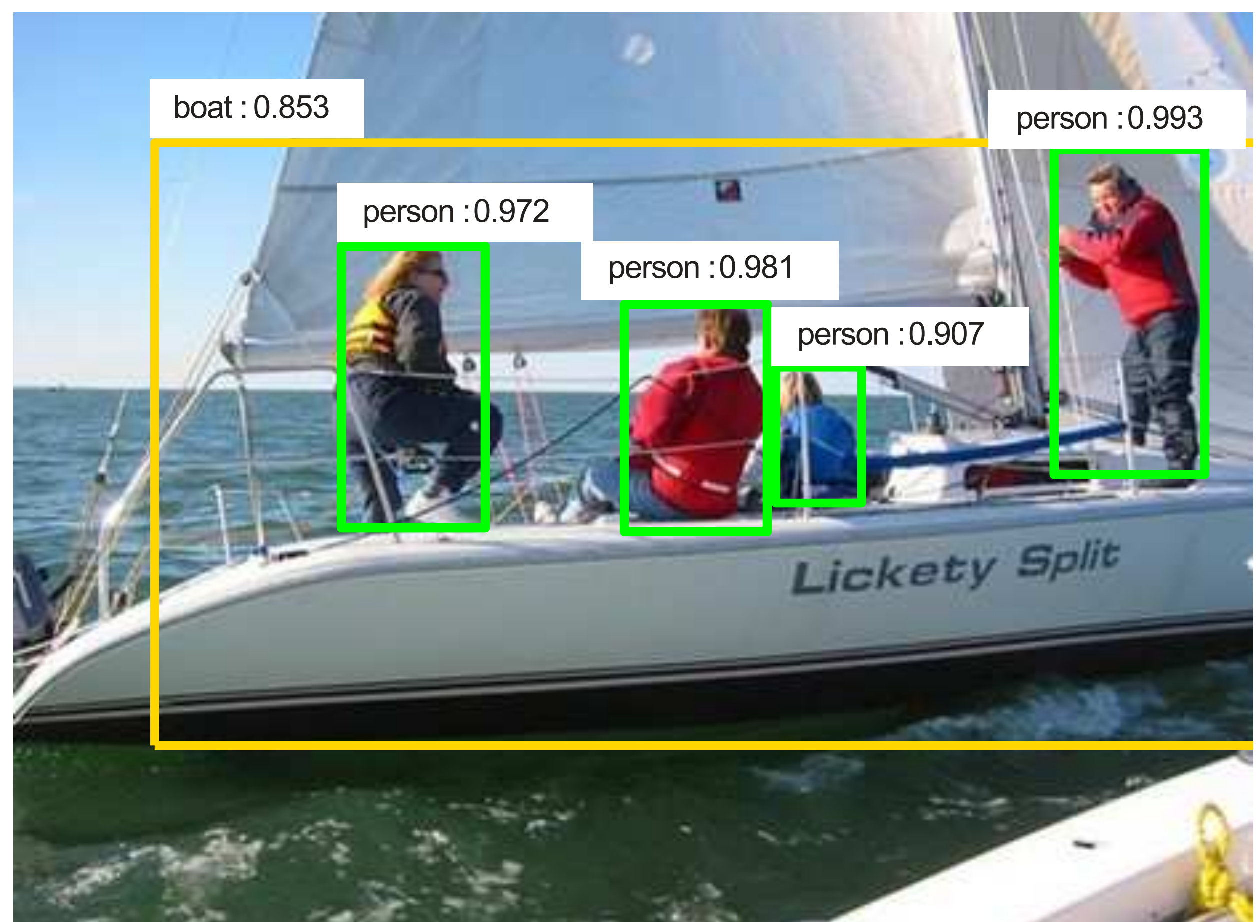
can model fine details just fine

Example detections









PASCAL Leaderboards (Nov 2014)

Detection challenge comp4: train on own data

Average Precision (AP %)

	mean	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv/ monitor	submission date
NUS_NIN_c2000 [?]	**63.8**	**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**	30-Oct-2014
BabyLearning [?]	63.2	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	12-Nov-2014
R-CNN (bbox reg) [?]	62.4	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	26-Oct-2014
NUS_NIN [?]	62.4	77.9	73.1	**62.6**	39.5	**43.3**	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7	30-Oct-2014
R-CNN [?]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7	25-Oct-2014
Feature Edit [?]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8	06-Sep-2014
R-CNN (bbox reg) [?]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1	13-Mar-2014
SDS [?]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7	21-Jul-2014
R-CNN [?]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6	30-Jan-2014
Poselets2 [?]	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	58.7	-	-	-	-	-	06-Jun-2014

PASCAL Leaderboards (Dec 2015)

Detection challenge comp4: train on own data

	mean	aero plane	bicycle	bird	boat	bottle	bus	car	cat	chair	cow	dining table	dog	horse	motor bike	person	potted plant	sheep	sofa	train	tv/ monitor	submission date
Faster RCNN, ResNet (VOC+COCO) [?]	83.8	92.1	88.4	84.8	75.9	71.4	86.3	87.8	94.2	66.8	89.4	69.2	93.9	91.9	90.9	89.6	67.9	88.2	76.8	90.3	80.0	10-Dec-2015
ION [?]	76.4	87.5	84.7	76.8	63.8	58.3	82.6	79.0	90.9	57.8	82.0	64.7	88.9	86.5	84.7	82.3	51.4	78.2	69.2	85.2	73.5	23-Nov-2015
MNC baseline [?]	75.9	86.4	81.1	76.4	64.3	57.8	81.1	80.3	92.0	55.2	82.6	61.0	89.9	86.4	84.6	85.4	53.1	79.8	66.1	84.7	69.9	15-Dec-2015
Faster RCNN baseline (VOC+COCO) [?]	75.9	87.4	83.6	76.8	62.9	59.6	81.9	82.0	91.3	54.9	82.6	59.0	89.0	85.5	84.7	84.1	52.2	78.9	65.5	85.4	70.2	24-Nov-2015
LocNet [?]	74.8	86.3	83.0	76.1	60.8	54.6	79.9	79.0	90.6	54.3	81.6	62.0	89.0	85.7	85.5	82.8	49.7	76.6	67.5	83.2	67.4	06-Nov-2015
** HRCNN ** [?]	74.6	85.9	83.9	75.5	60.9	54.5	81.4	79.1	90.6	53.3	79.7	61.6	89.9	86.2	85.8	78.2	49.1	75.1	68.6	86.1	67.7	13-Nov-2015
MR_CNN_S_CNN_MORE_DATA [?]	73.9	85.5	82.9	76.6	57.8	62.7	79.4	77.2	86.6	55.0	79.1	62.2	87.0	83.4	84.7	78.9	45.3	73.4	65.8	80.3	74.0	06-Jun-2015
HyperNet_VGG [?]	71.4	84.2	78.5	73.6	55.6	53.7	78.7	79.8	87.7	49.6	74.9	52.1	86.0	81.7	83.3	81.8	48.6	73.5	59.4	79.9	65.7	12-Oct-2015
HyperNet_SP [?]	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6	28-Oct-2015
Fast R-CNN + YOLO [?]	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4	49.4	75.5	57.0	87.5	80.9	81.0	74.7	41.8	71.5	68.5	82.1	67.2	06-Nov-2015
MR_CNN_S_CNN [?]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1	09-May-2015
RPN [?]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5	01-Jun-2015
DEEP_ENSEMBLE_COCO [?]	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	68.8	75.9	71.4	03-May-2015
Networks on Convolutional Feature Maps [?]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1	17-Apr-2015
Fast R-CNN VGG16 extra data [?]	68.4	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	17-Apr-2015
UMICH_FGS_STRUCT [?]	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2	20-Jun-2015
NUS_NIN_c2000 [?]	63.8	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	30-Oct-2014
BabyLearning [?]	63.2	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	12-Nov-2014
NUS_NIN [?]	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7	30-Oct-2014
R-CNN (bbox reg) [?]	62.4	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	26-Oct-2014
R-CNN [?]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7	25-Oct-2014

Other applications

Huge variety of applications

Siamese networks for face recognition/verification

E.g. VGG-Face

Huge variety of applications

Text spotting

E.g. SynthText and VGG-Text

http://zeus.robots.ox.ac.uk/textsearch/#/search/

A two-step approach

detection

classification

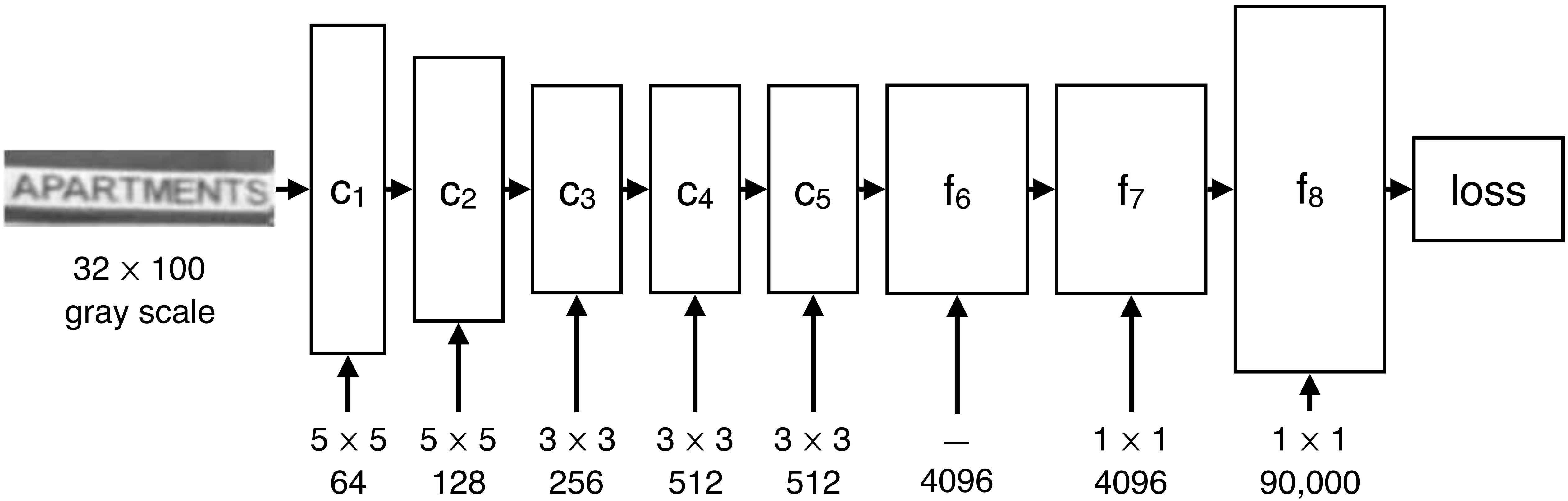
“APARTMENTS”

We will focus on the “classification” step

Most previous approaches start by recognising individual characters
[Yao et al. 2014, Bissacco et al. 2013, Jaderberg et al. 2014, Posner et al. 2010, Quack et al. 2009, Wang et al. 2011, Wang et al. 2012, Weinman et al. 2014, …]

The alternative is to **directly map word images to words**
[Almazan et al. 2014, Goel et al. 2013, Mishra et al. 2012, Novikova et al. 2012, Rodriguez-Serrano et al. 2012, **Godfellow et al. 2013**]

A massive classifier



Goal: map images to one of 90K classes (one per word)

Architecture

- each linear operator is followed by ReLU
- c_1, c_2, c_3, c_5 are followed by 2×2 max pooling
- 500 million parameters
- evaluation requires 2.2ms on a GPU

Learning a massive classifier

Massive training data

9 million images spanning 90K words (100 examples per word)

Learning algorithm

- SGD
- dropout (after fc_6 and fc_7)
- mini batches

Problem

- in practice each batch must contain at least 1/5 of all the classes
- batch size = 18K (!!)

Solution: incremental training

- learn first using 5K classes only (1K minibatches)
- then incrementally add 5K more classes

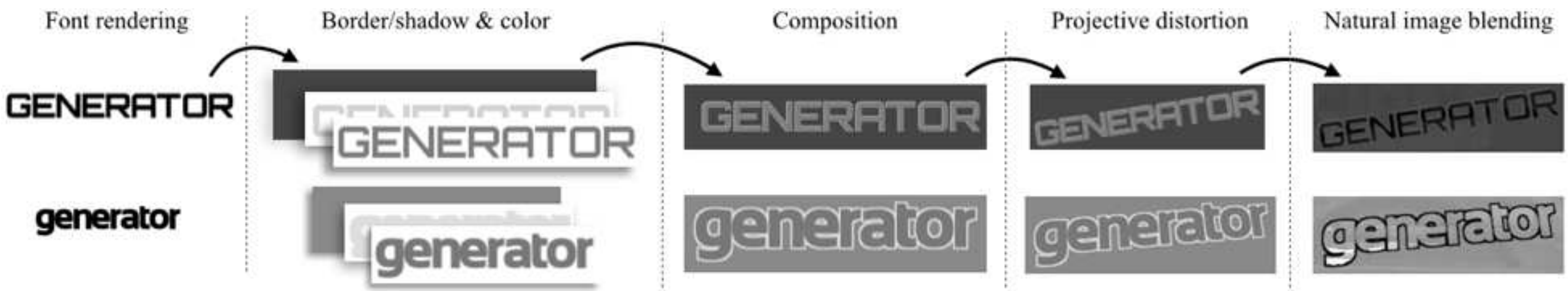
Synth Text dataset

Existing text recognition benchmark datasets are too small to train the model

Synth Text

- http://www.robots.ox.ac.uk/~vgg/data/text/
- a new synthetic dataset for text spotting
- include realistic visual effects
- infinity large (9M images available for download)

Synth Text generation



Font rendering

- sample at random one of 1400 Google Fonts

Border/shadow

- randomly add inset/outset border and shadow

Projective distortion

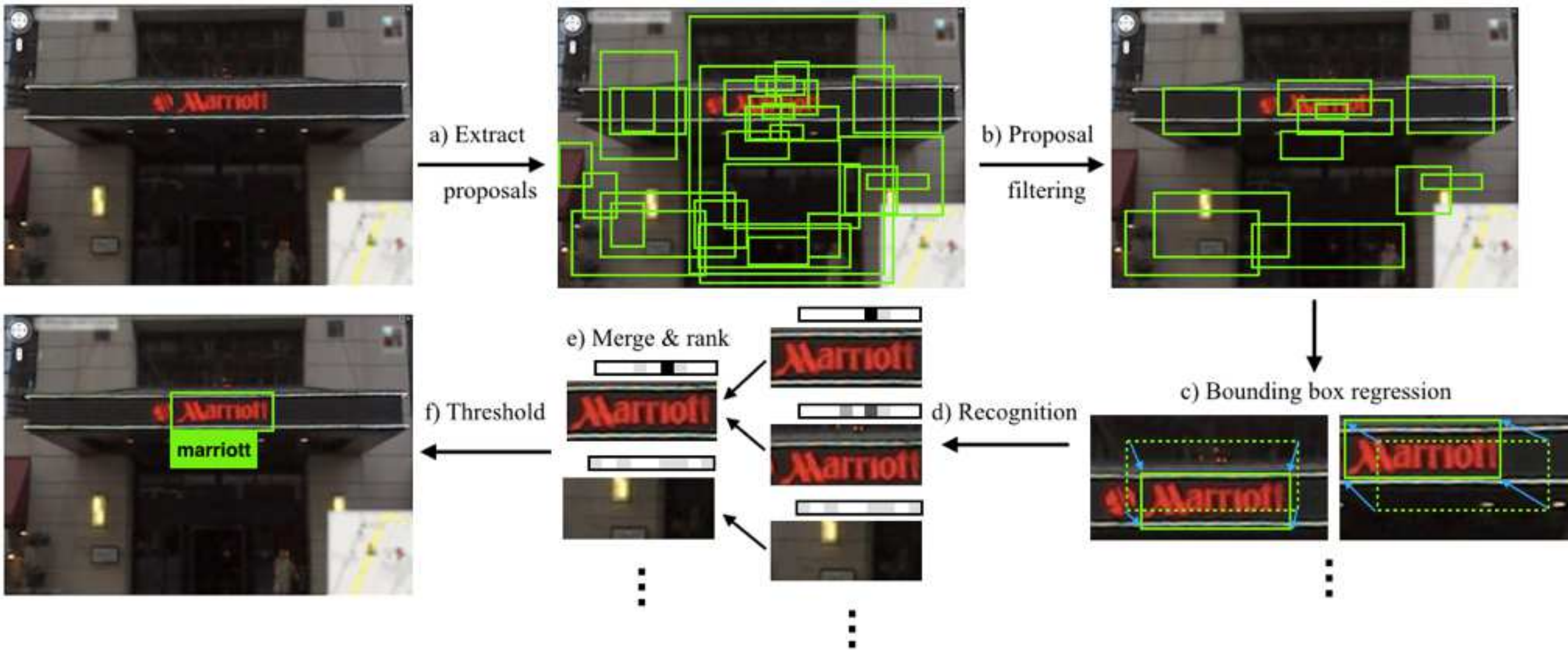
Blending

- use a random crop from SVT as background
- randomly sample alpha channel, mixing operator (normal, burn, …)

Noise

- elastic distortion, white noise, blur, JPEG compression, …

Overall system



Proposal generation: edge boxes, AST

Proposal filtering: HOG, RF

Bounding box-regression: CNN

Text recognition: CNN

Post-processing: merging, non-max. suppression

Qualitative results: text spotting

1.00/1.00/1.00

1.00/1.00/1.00

1.00/1.00/1.00

1.00/1.00/1.00

Qualitative results: text spotting

1.00/1.00/1.00

1.00/1.00/1.00

1.00/0.88/0.93

1.00/1.00/1.00

Qualitative results: text retrieval

“APARTMENTS”

“BORIS JOHNSON”

“HOLLYWOOD”

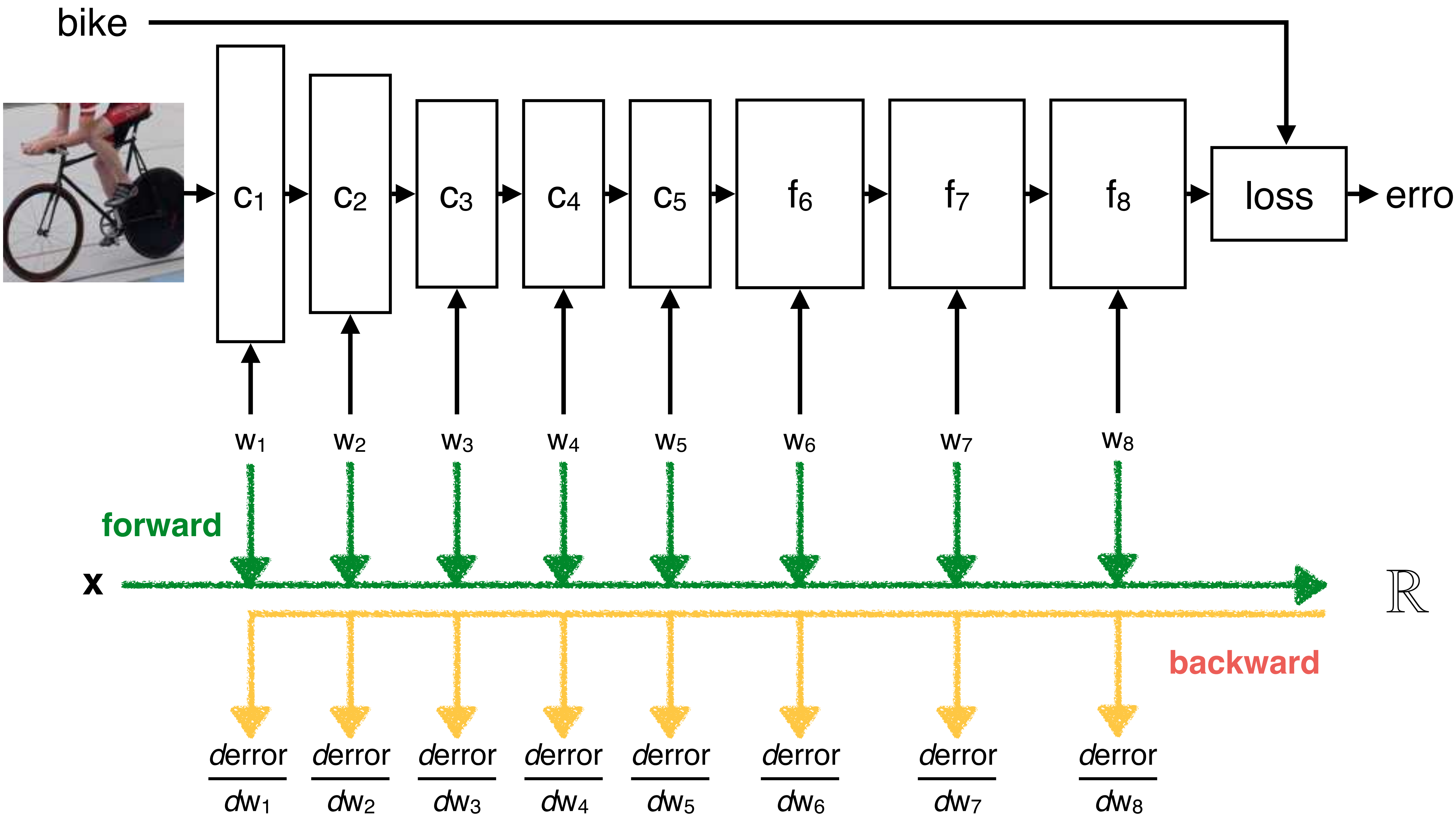
Qualitative results: text retrieval

“POLICE” “CASTROL” “VISION”

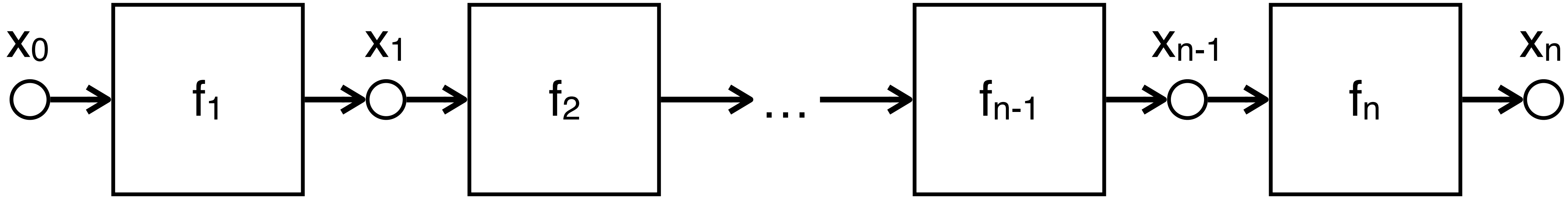
Backpropagation revisited

Backpropagation

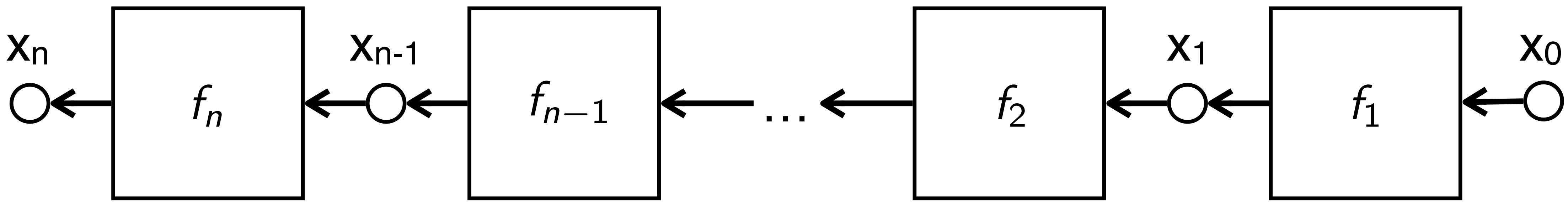
Compute derivatives using the chain rule



Chain rule: scalar version



Chain rule: scalar version



A composition of *n* functions

$$\mathbf{x}_n = (\; f_n \; \circ \; f_{n-1} \; \circ \; \dots \; \circ \; f_2 \; \circ \; f_1 \;) \; (\mathbf{x}_0)$$

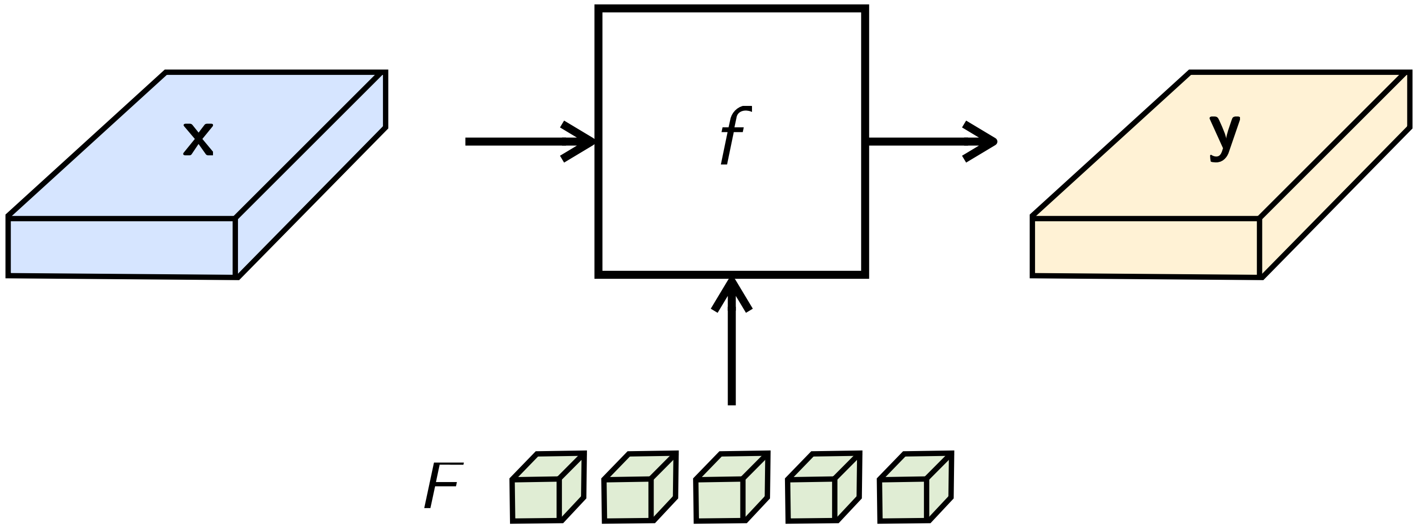
$$\frac{dx_n}{dx_0} = \frac{df_n}{dx_{n-1}} \times \frac{df_{n-1}}{dx_{n-2}} \times \dots \times \frac{df_2}{dx_1} \times \frac{df_1}{dx_0}$$

Derivative ← chain rule

Tensor-valued functions

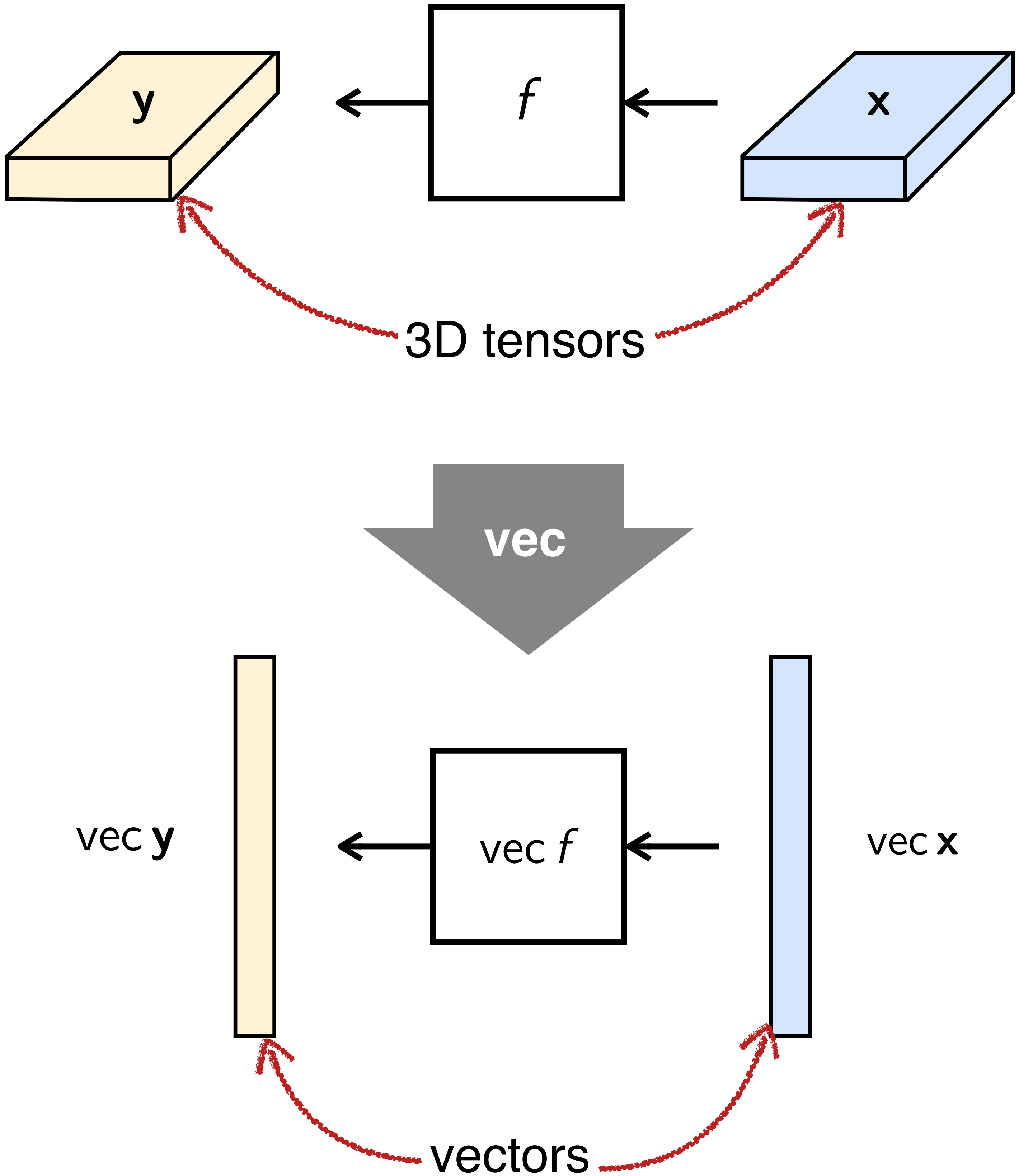
E.g. linear convolution = bank of 3D filters

$$\boxed{\mathbf{y} = F * \mathbf{x} + b}$$

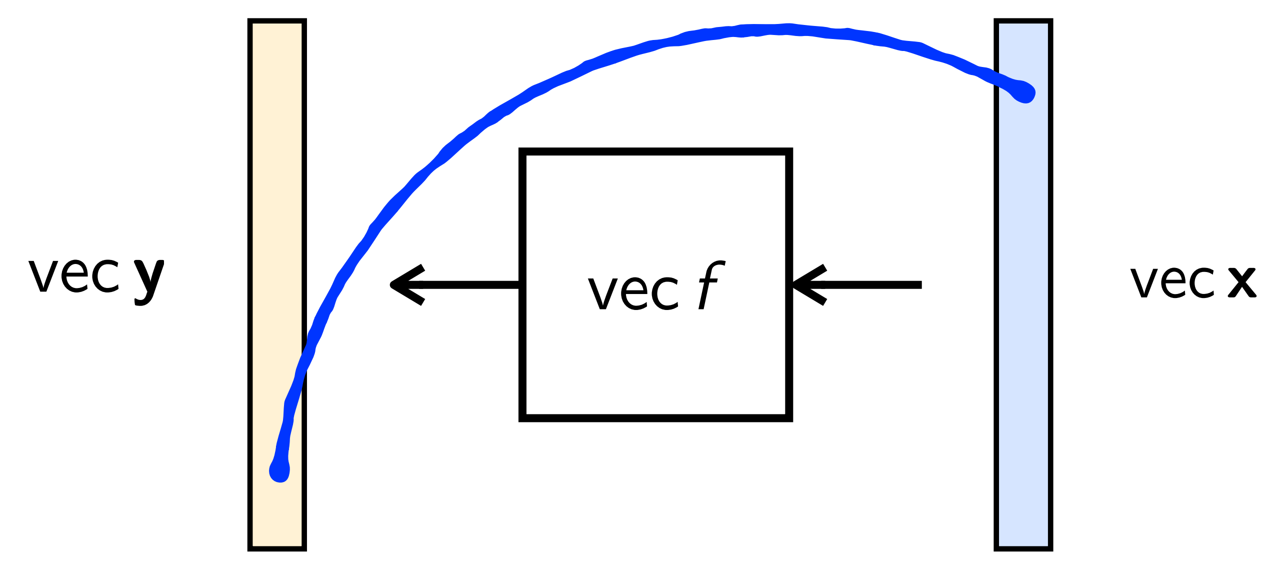


	height	width	channels	instances
input **x**	*H*	*W*	*C*	*1* or *N*
filters F	H_f	W_f	*C*	*K*
output **y**	$H - H_f + 1$	$W - W_f + 1$	*K*	*1* or *N*

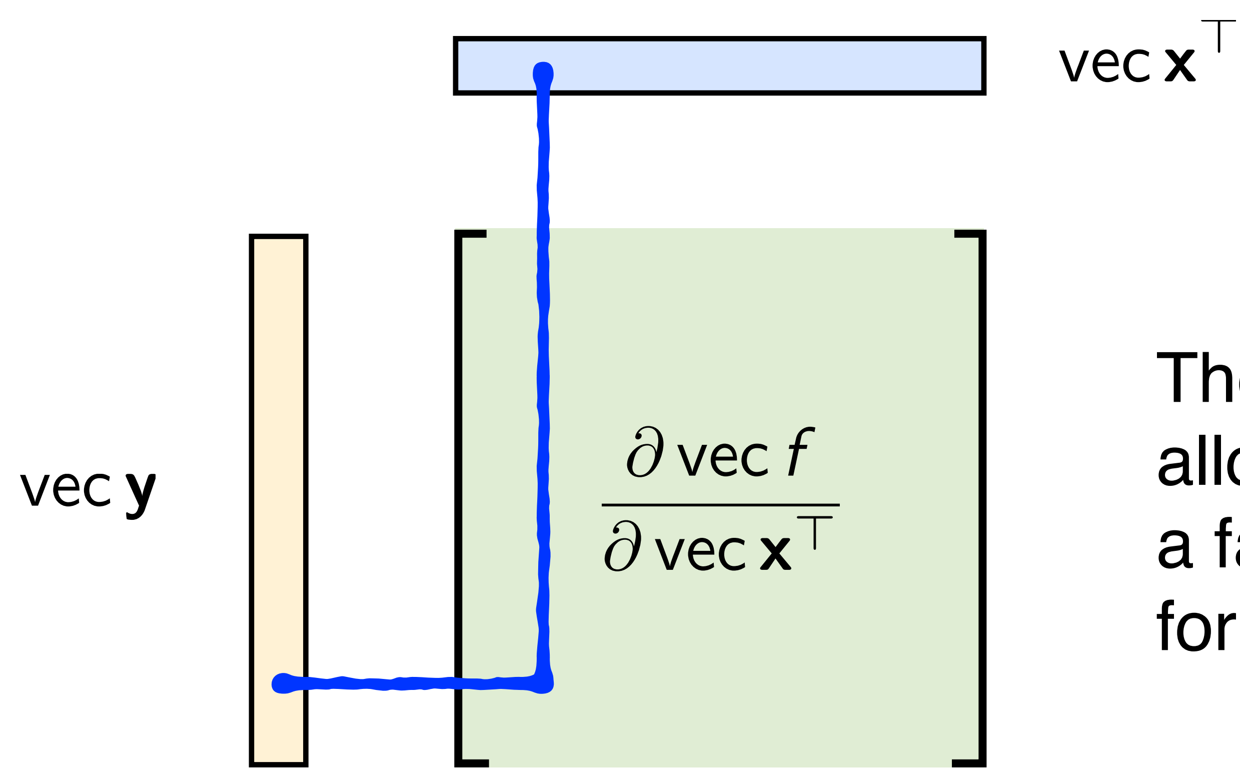
Vector representation



Derivative of tensor-valued functions



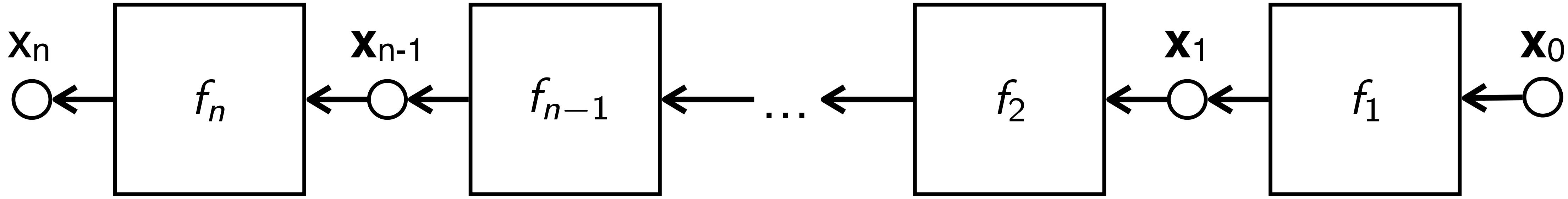
Derivative (Jacobian): every output element w.r.t. every input element!



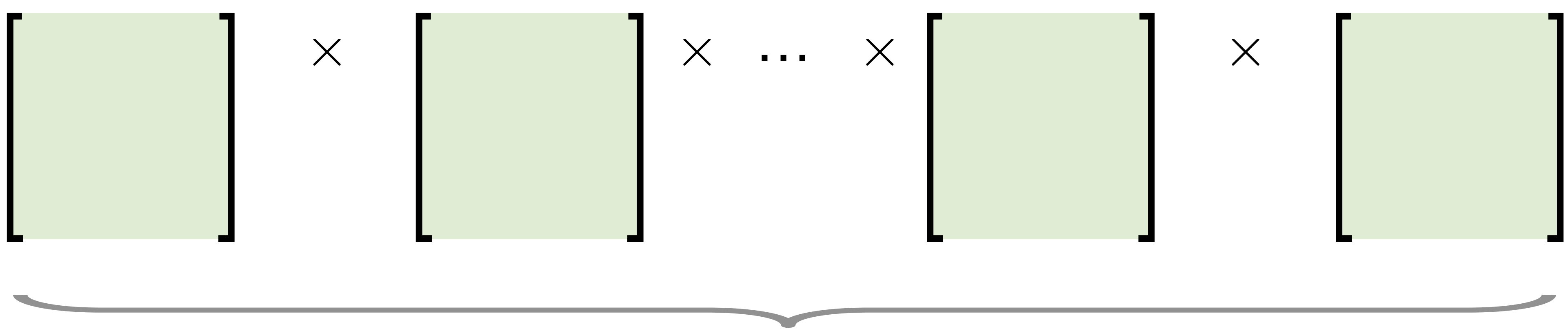
The **vec** operator allows us to use a familiar matrix notation for the derivatives

Chain rule: tensor version

Using vec() and matrix notation

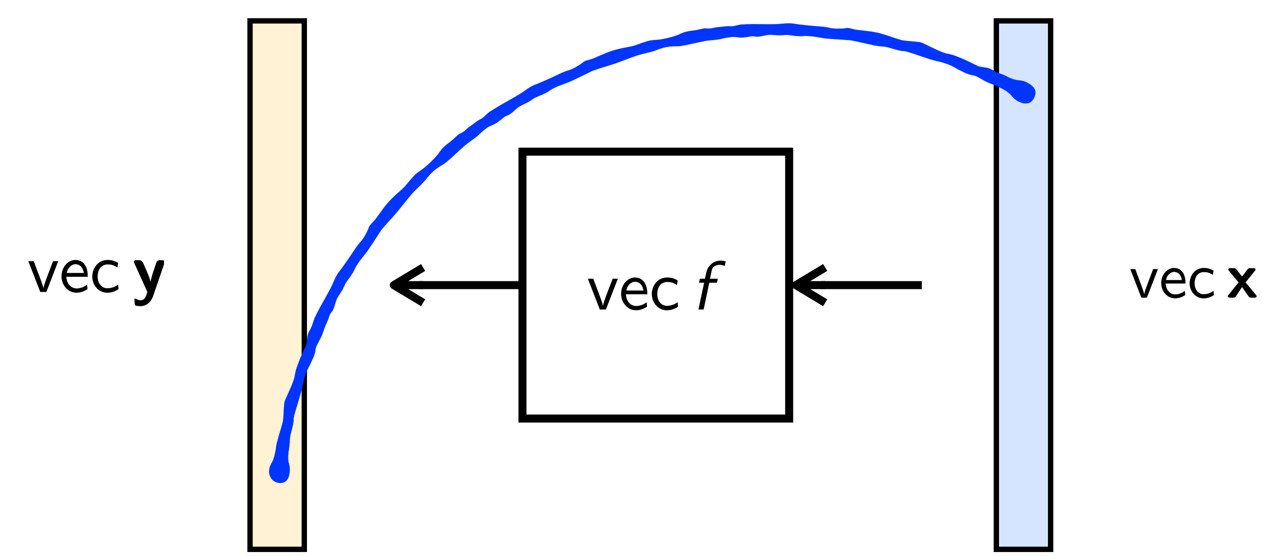


$$\frac{d\,\mathrm{vec}\, f_n}{d\,\mathrm{vec}\,\mathbf{x}_{n-1}^\top} \times \frac{d\,\mathrm{vec}\, f_{n-1}}{d\,\mathrm{vec}\,\mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d\,\mathrm{vec}\, f_2}{d\,\mathrm{vec}\,\mathbf{x}_1^\top} \times \frac{d\,\mathrm{vec}\, f_1}{d\,\mathrm{vec}\,\mathbf{x}_0^\top}$$

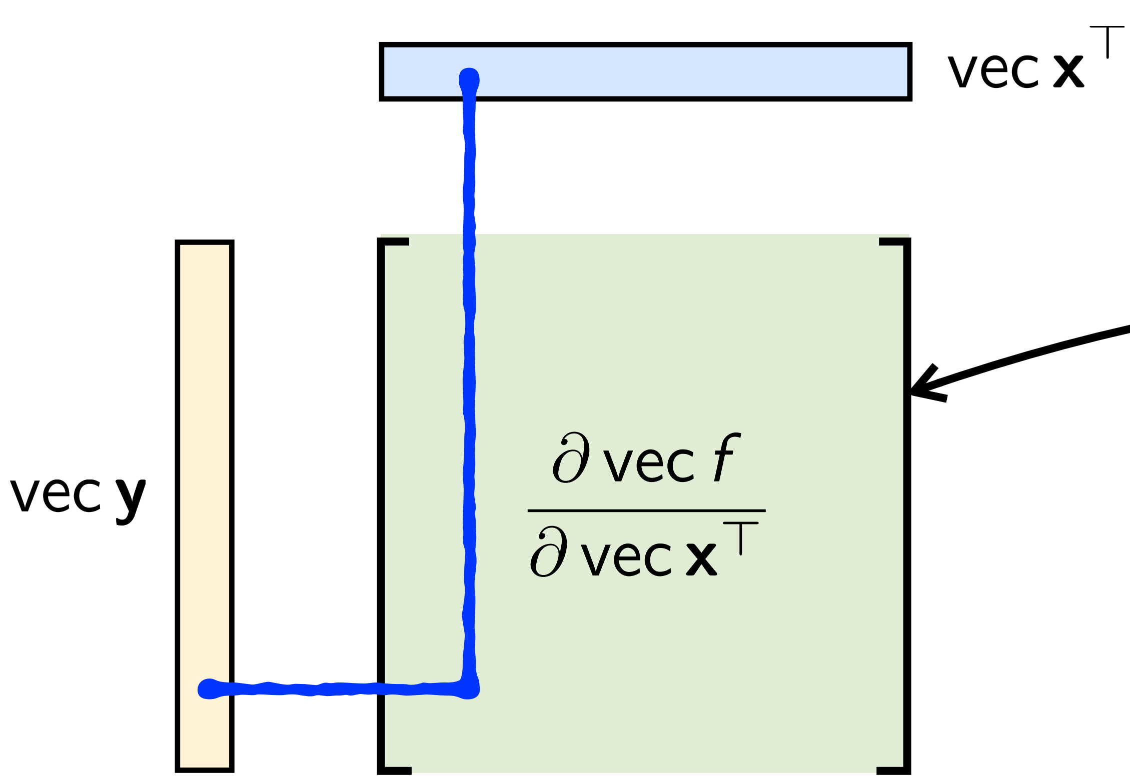


$$\frac{d\,\mathrm{vec}\, f}{d\,\mathrm{vec}\,\mathbf{x}_0^\top}$$

The (unbearable) size of tensor derivatives

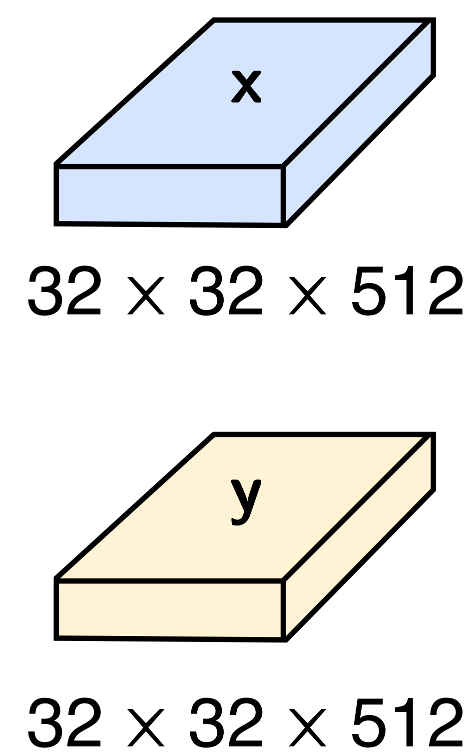


The size of these Jacobian matrices is **huge**. Example:



275 B elements

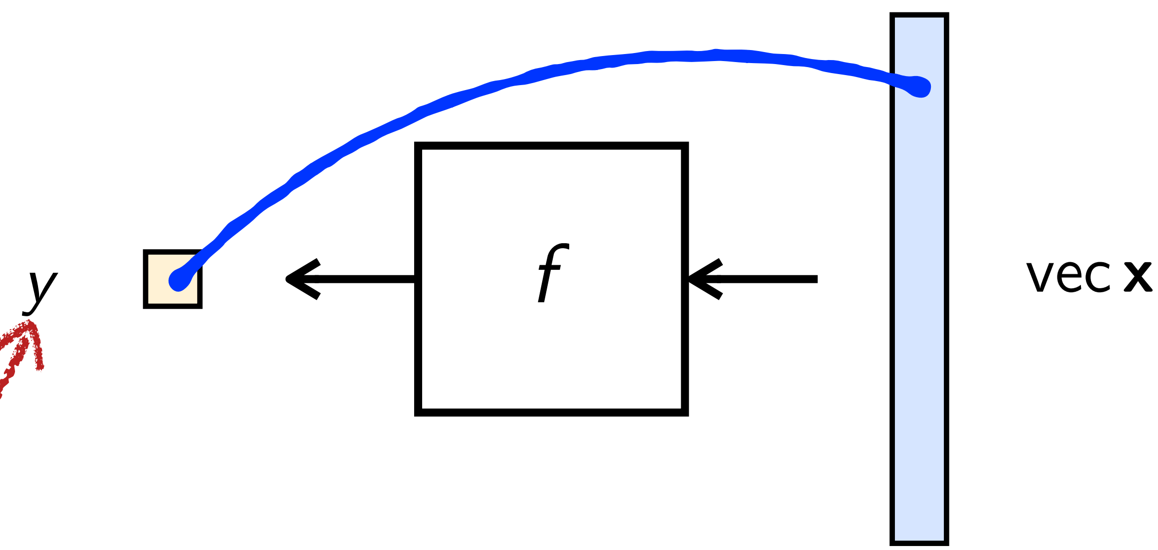
1 TB of memory required !!



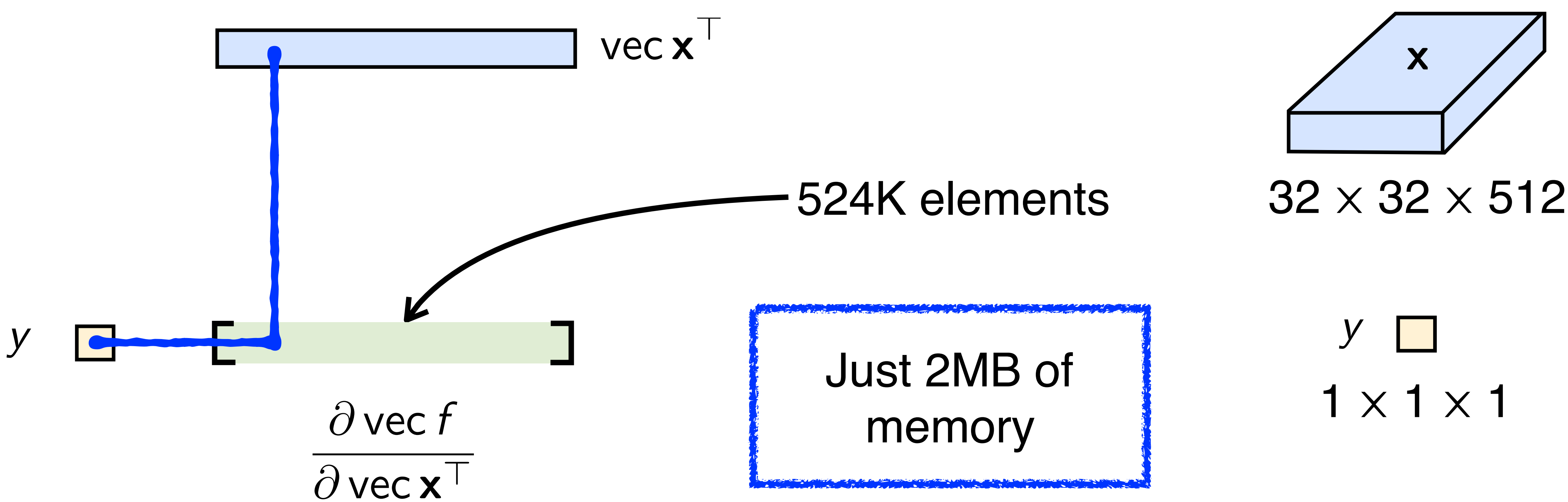
Unless the output is a scalar

Scalar

This is always the case
if the last layer
is the **loss function**

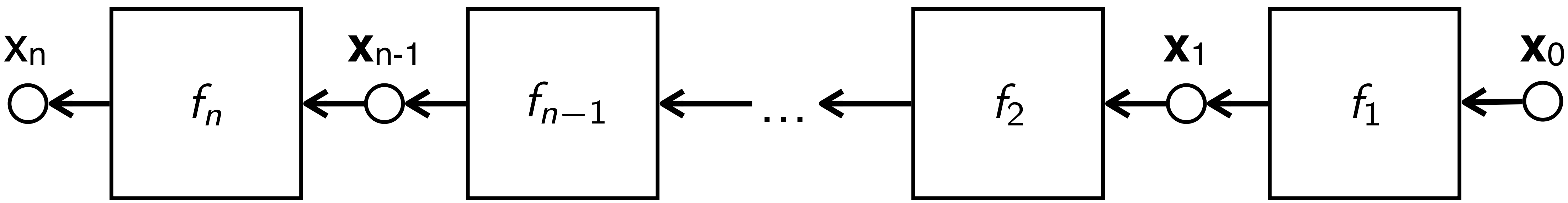


Now the Jacobian has the same size as **x**. Example:

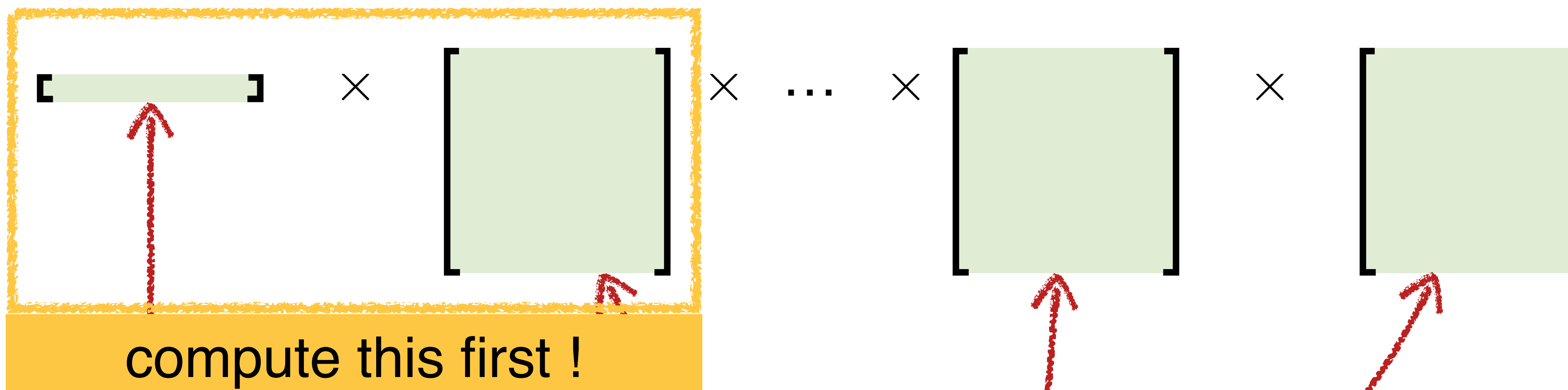


Backpropagation

Assumed x_n is a scalar (e.g. loss)



$$\frac{df_n}{d \operatorname{vec} \mathbf{x}_{n-1}^\top} \times \frac{d \operatorname{vec} f_{n-1}}{d \operatorname{vec} \mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d \operatorname{vec} f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

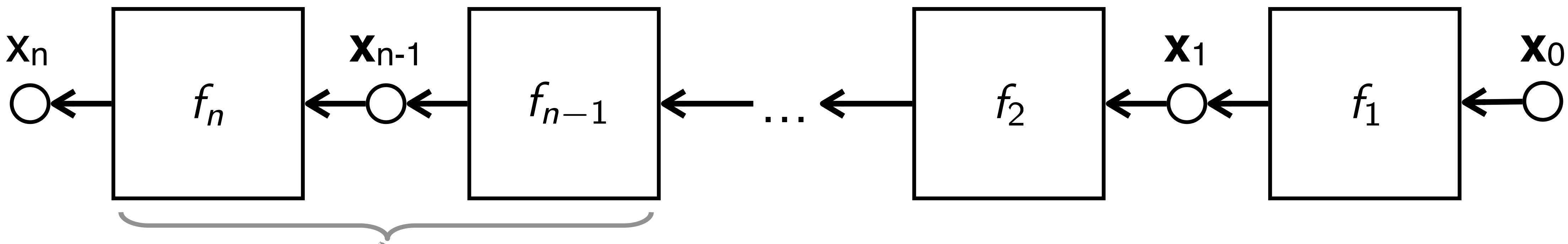


small
explicitly compute

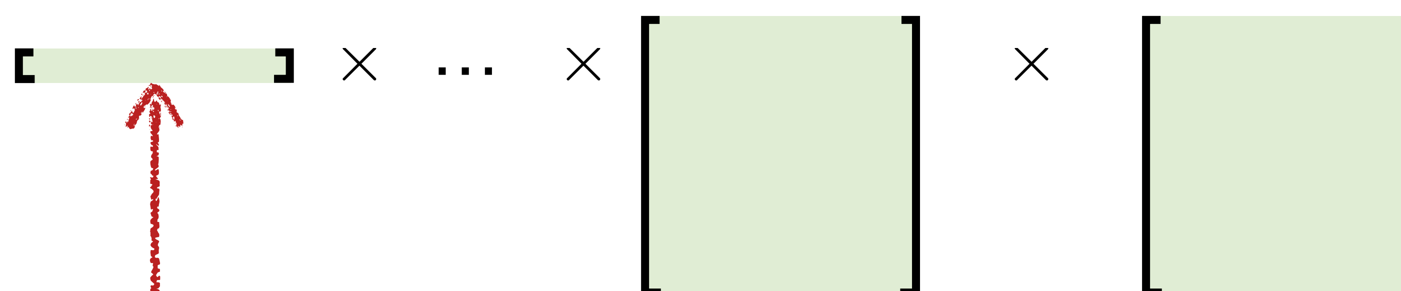
uber matrices
do **not** explicitly compute

Backpropagation

Assumed x_n is a scalar (e.g. loss)



$$\frac{d f_n \circ f_{n-1}}{d \operatorname{vec} \mathbf{x}_{n-2}^\top} \times \cdots \times \frac{d \operatorname{vec} f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

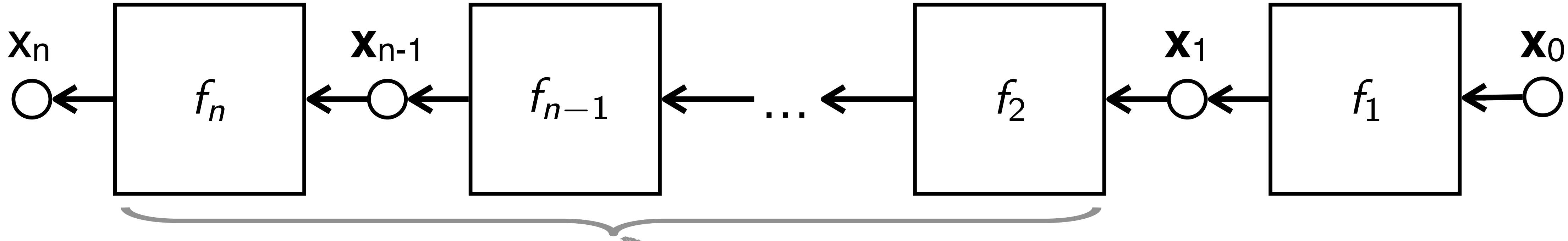


small
explicitly compute

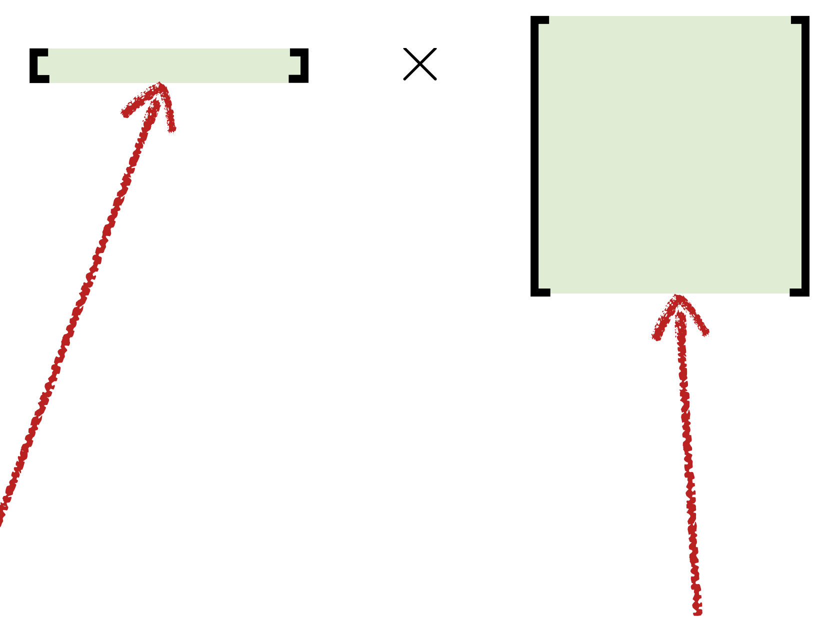
uber matrices
do **not** explicitly compute

Backpropagation

Assumed x_n is a scalar (e.g. loss)



$$\frac{d f_n \circ \cdots \circ f_2}{d \operatorname{vec} \mathbf{x}_1^\top} \times \frac{d \operatorname{vec} f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$

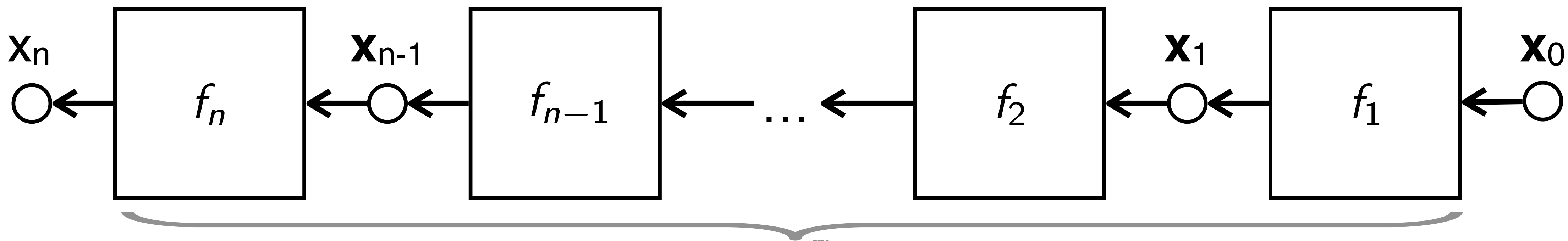


small
explicitly compute

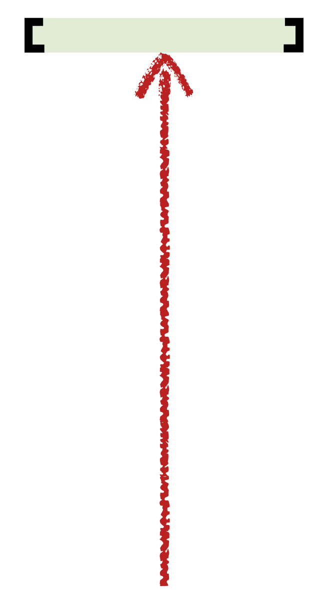
uber matrix
do **not** explicitly compute

Backpropagation

Assumed x_n is a scalar (e.g. loss)



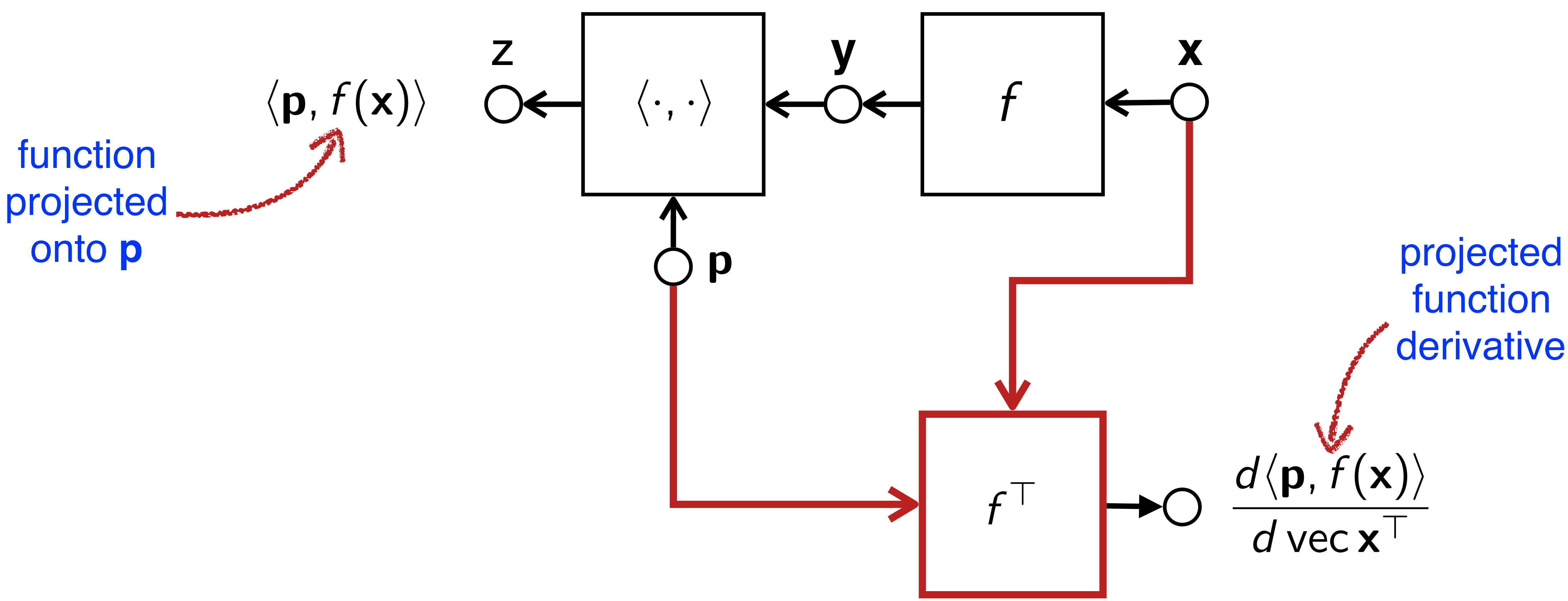
$$\frac{d f_n \circ \cdots \circ f_1}{d \operatorname{vec} \mathbf{x}_0^\top}$$



small
explicitly compute

Projected function derivative

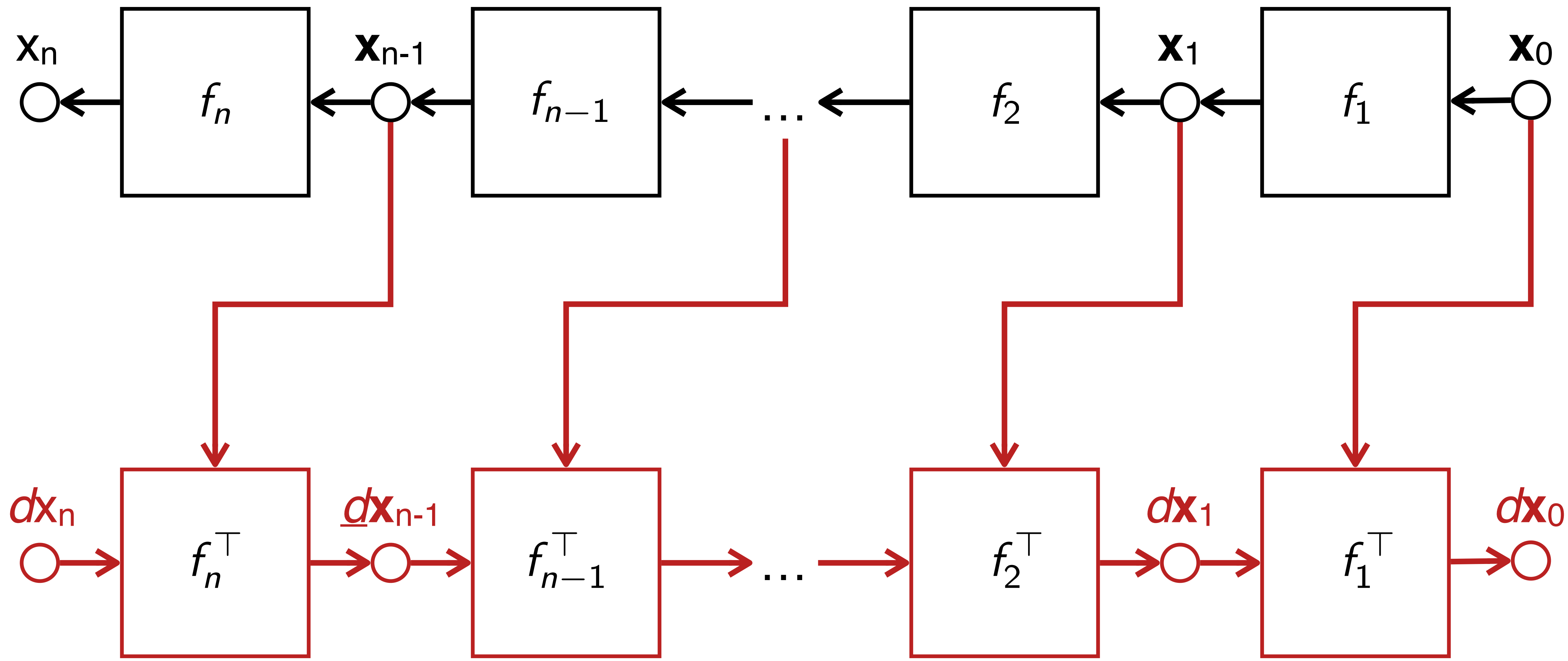
The “BP-transpose” function



An “equivalent circuit” is obtained by introducing a transposed function $f^\top$

Backpropagation network

BP induces a "transposed" network

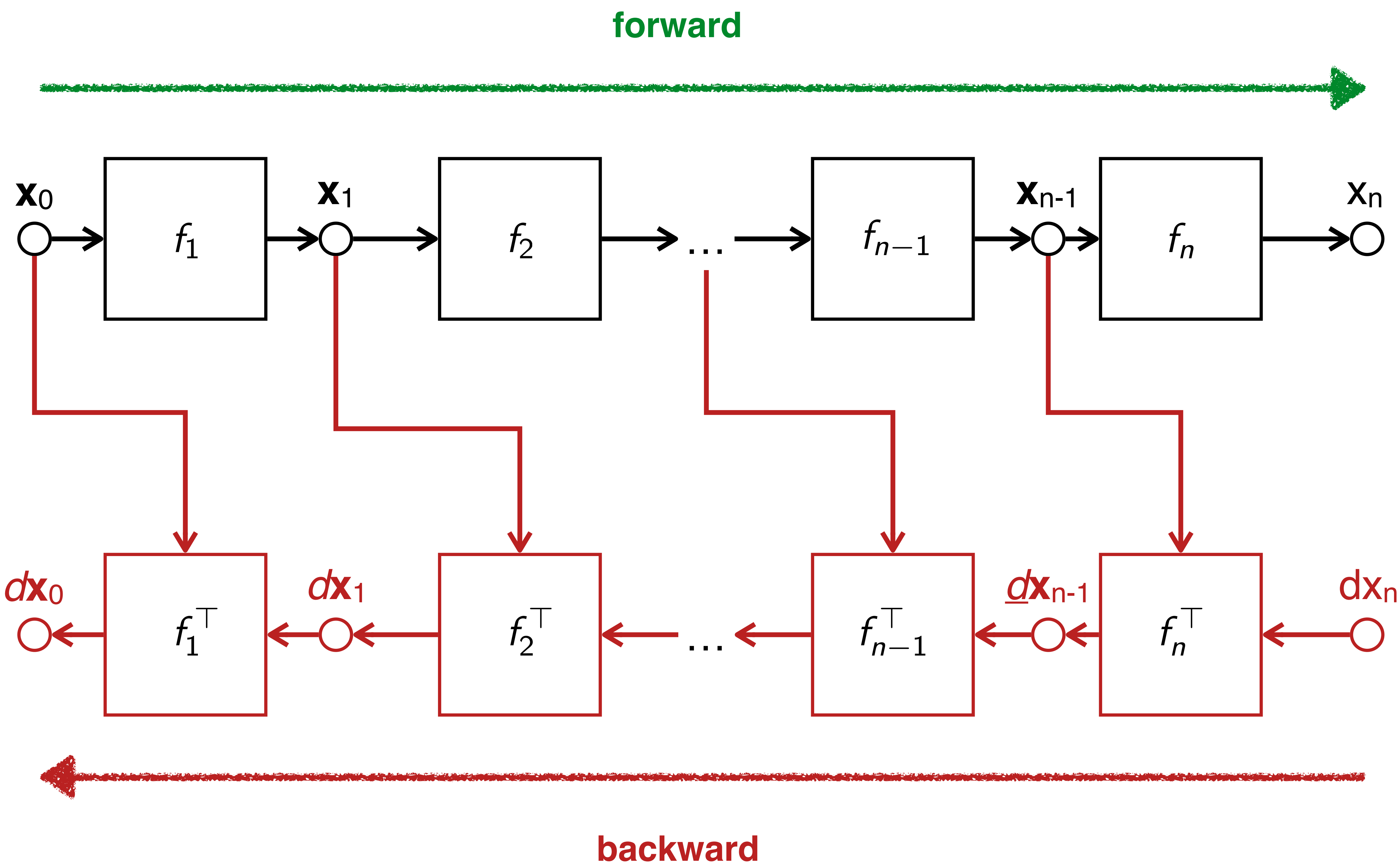


where $$d\mathbf{x}_i = \frac{d f_n \circ \cdots \circ f_{i+1}}{d \operatorname{vec} \mathbf{x}_i}$$

Note: the BP network is linear in $d\mathbf{x}_1, \ldots, d\mathbf{x}_{n-1}, d\mathbf{x}_n$. Why?

Backpropagation network

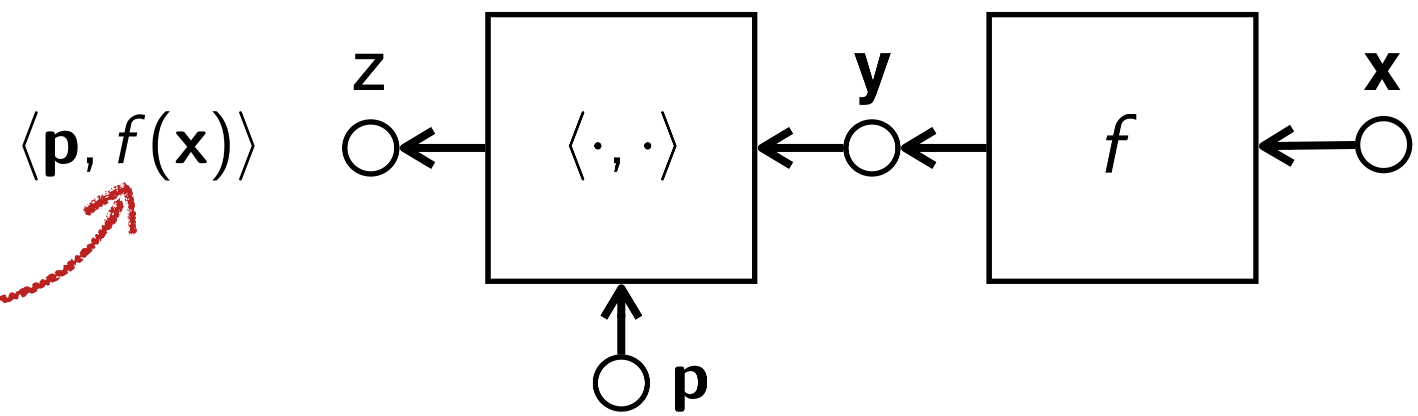
BP induces a “transposed” network



Projected function derivative

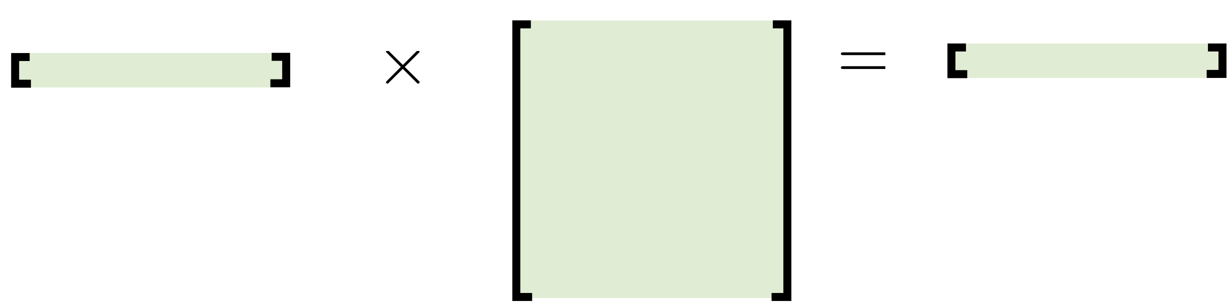
Interpretation of vector-matrix product in BP

function
projected
onto **p**



projected
function
derivative

$$\mathbf{p}^\top \times \frac{d \operatorname{vec} f}{d \operatorname{vec} \mathbf{x}^\top} = \frac{d \langle \mathbf{p}, f(\mathbf{x}) \rangle}{d \operatorname{vec} \mathbf{x}^\top}$$



Example: MatConvNet

Modular: every part can be used directly

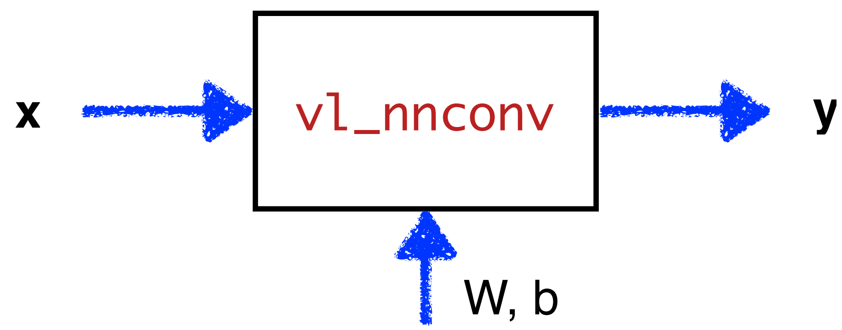
Portable C++ / CUDA Core	**Building Blocks**	**Wrappers**

Convolution, pooling, normalization, cuDNN, …	`vl_nnconv()`
	`vl_nnpool()`
	`vl_nnnormalize()`
	…
		SimpleNN
		DagNN

Could be used directly or in other languages (e.g. Python or Lua)	Atomic operations Reusable Flexible GPU support	Pack a CNN model Simple to use

Anatomy of a building block

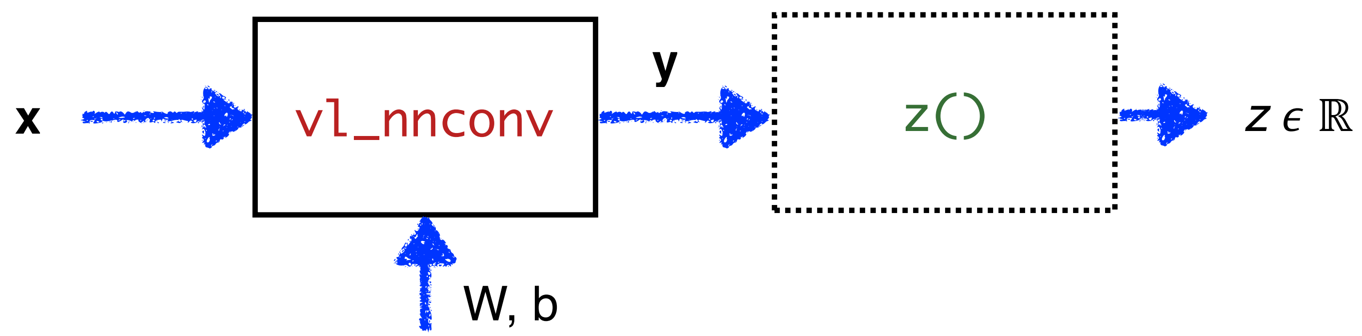
forward (eval)




```
y = vl_nnconv(x, W, b)
```

Anatomy of a building block

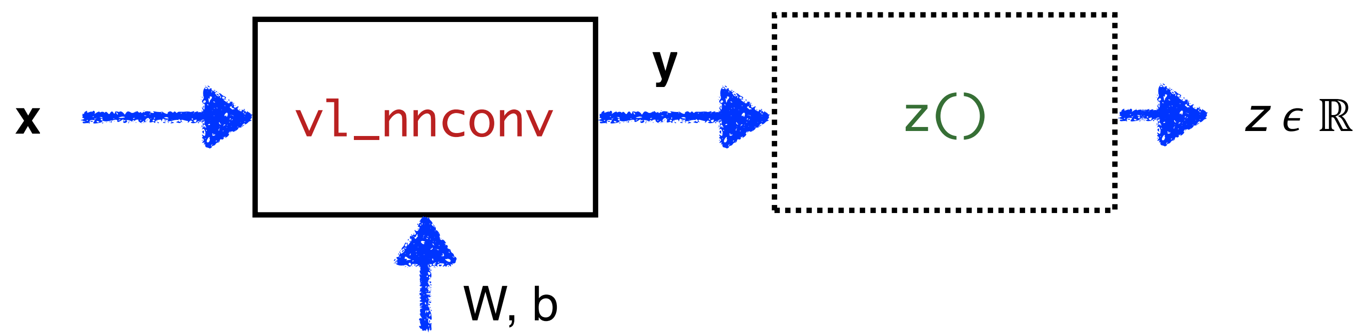
forward (eval)




```
y = vl_nnconv(x, W, b)
```

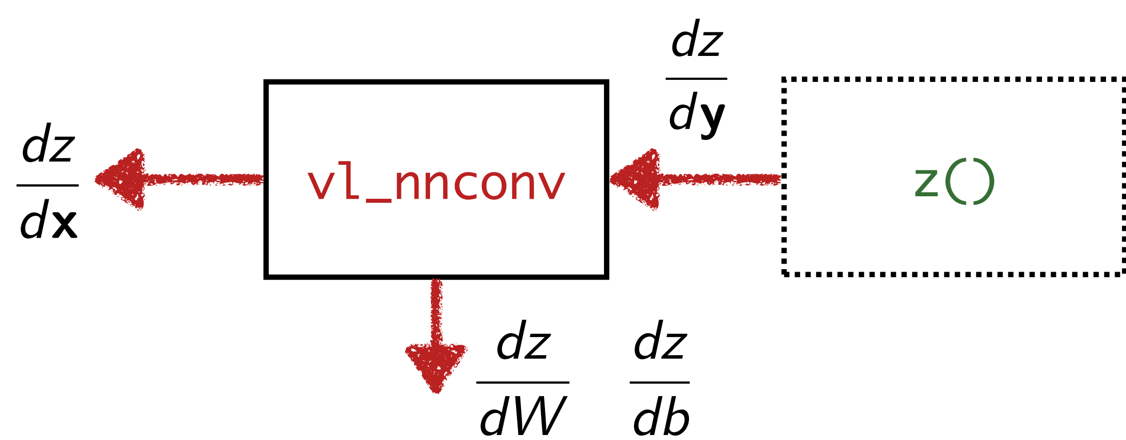
Anatomy of a building block

forward (eval)




```
y = vl_nnconv(x, W, b)
```

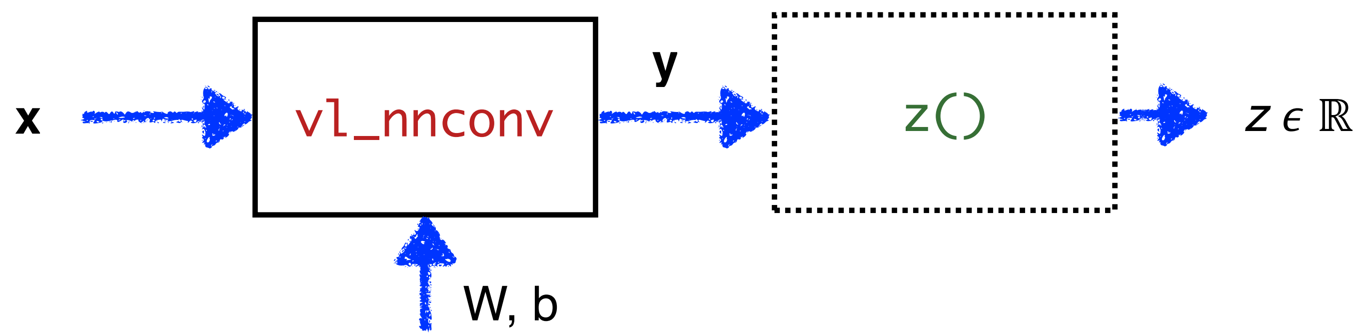
backward (backprop)




```
dzdx = vl_nnconv(x, W, b, dzdy)
```

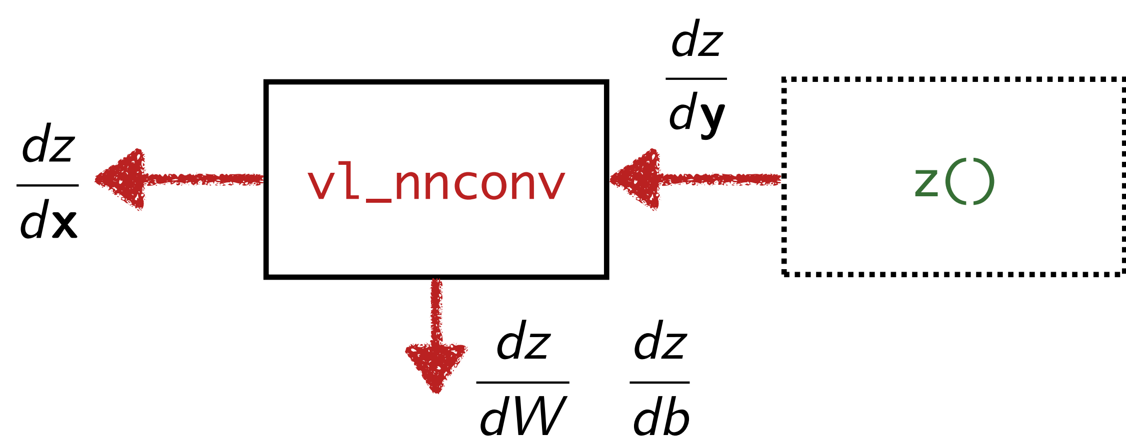
Anatomy of a building block

forward (eval)




```
y = vl_nnconv(x, W, b)
```

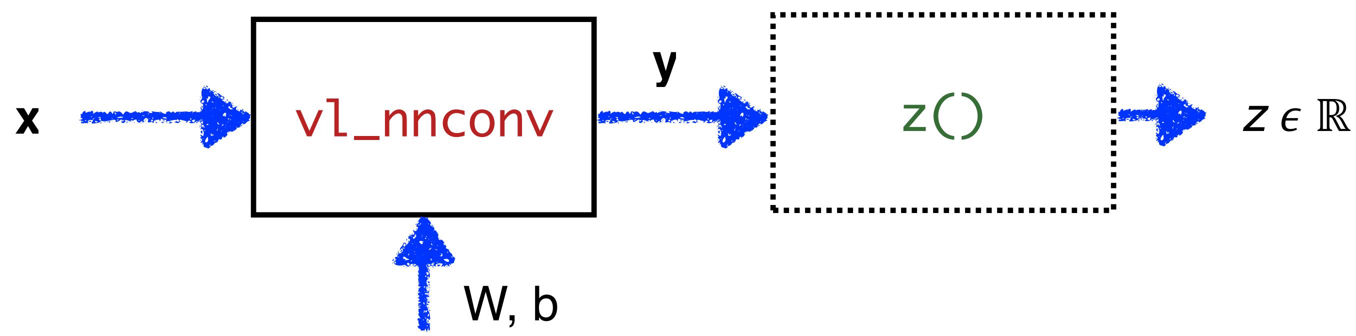
backward (backprop)




```
dzdx = vl_nnconv(x, W, b, dzdy)
```

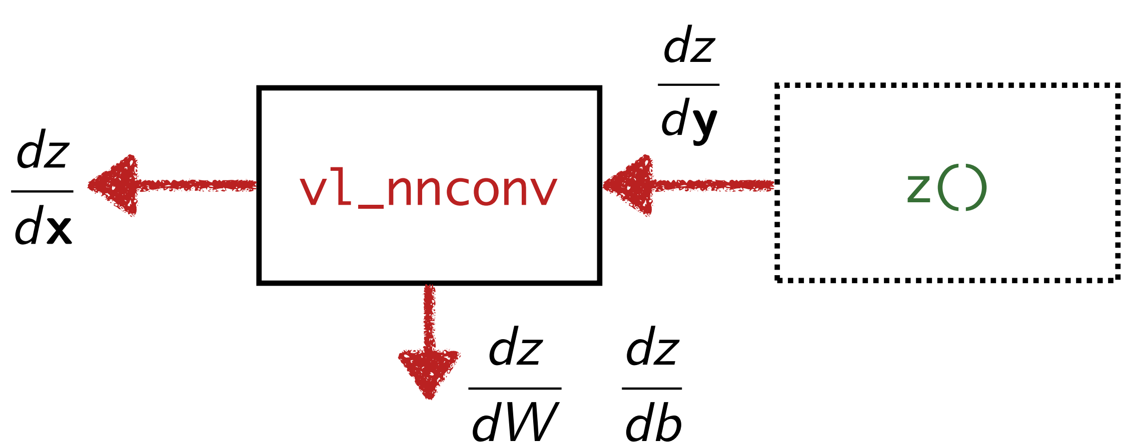
Anatomy of a building block

forward (eval)




```
y = vl_nnconv(x, W, b)
```

backward (backprop)




```
dzdx = vl_nnconv(x, W, b, dzdy)
```

Very fast implementations

Native MATLAB GPU support

Summary

Progress

- CNNs are still new, potential still being unveiled
- Depth, architectures, batch normalization, residual connections

Image segmentation

- Fully-convolutional nets: a label for each pixel
- Deconvolution, U-architectures, skip layers

Object detection

- Region nets: from pixels to a list of objects
- R-CNN, Fast R-CNN, R-CNN minus R, Faster R-CNN

Text spotting

- Brute force from synthetic data

Backpropagation revisited