Deep Learning 1

2025-2026 – Pascal Mettes

Lecture 5

Convolutional Neural Networks

Previous lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn't work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos

Where are we

have model
- Architecture — Today
- Weights initialisation

compute loss
- Gradient Descent, dataloader
- Augmentations
- Regularisation

compute gradient
- Loss functions
- Backprop

update weights
- Gradient Descent
- Advanced Optimizers

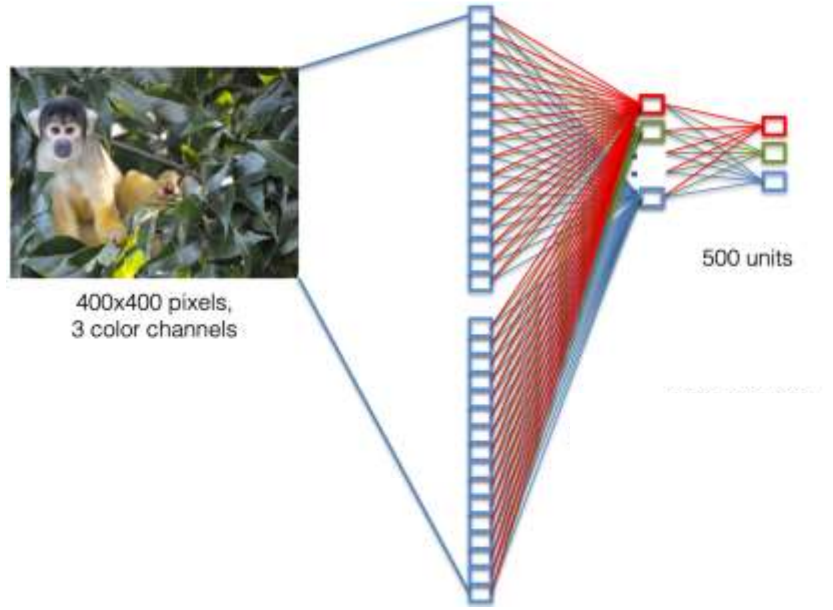
Last week

MLPs and real world data

Multi-layer perceptrons / feedforward networks assume vectorized data.

Consider an image of 400x400 pixels with 3 color channels.

How many parameters needed for a 1-layer networks with 500 hidden units?

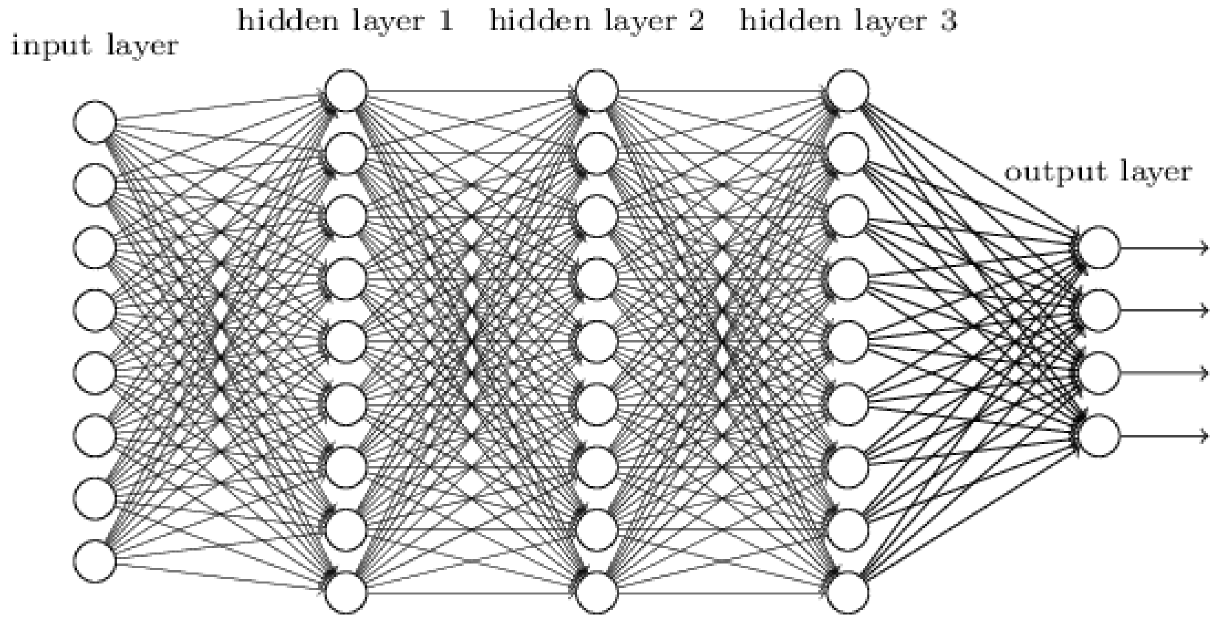


MLPs and real world data

From this fully-connected model, do we really need all the edges?

Can some of these be "shared" (equal weight)?

Can prior knowledge ("inductive biases") be incorporated into the design?



The convolution

We have a signal $x(t)$ and a weighting function $w(a)$

We can generate a new function $s(t)$ by the following equation:

$$s(t) = \int x(a)w(t-a)da$$

We refer to $w(a)$ *as a filter or a kernel.* This operation is called convolution.

The convolution operation is typically denoted with asterisk:

$$s(t) = (x * w)(t)$$

Convolution vs linear operators

Linear operation: $f(x, w) = x^T w$

Global operation, separate weight per feature

Dimensionality of w = dimensionality of x

1 output value

Convolutional operation: $f(x, w) = x * w$

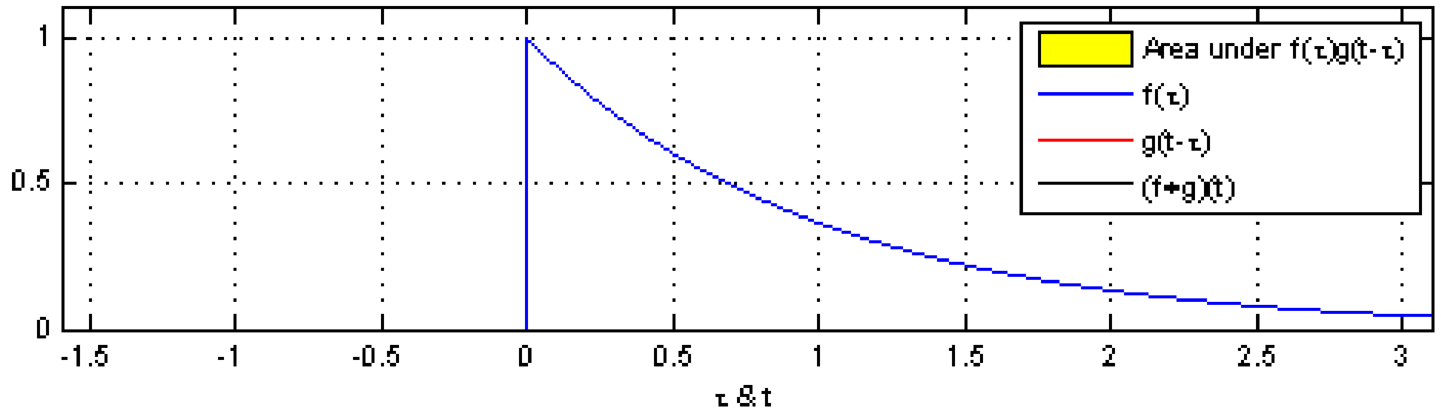
Local operation, shared weights over local regions

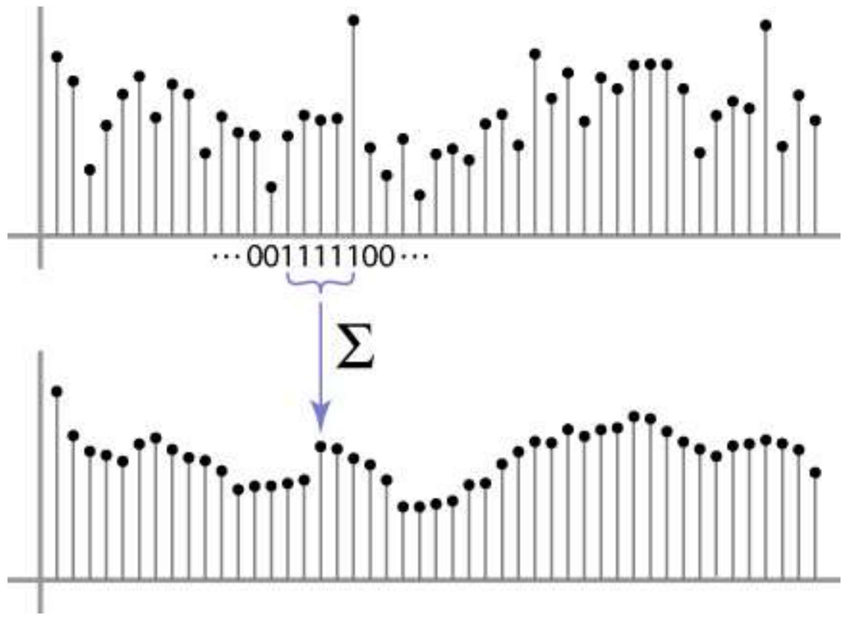
Dimensionality of w much smaller than dimensionality of x

Output (almost?) same size as input

1D convolutions

The convolution $(f * g)(t)$ of two functions $f(t)$ and $g(t)$ *computes the overlap in area.*





2D convolutions

For a two-dimensional image I as our input, we want to use a two-dimensional kernel K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

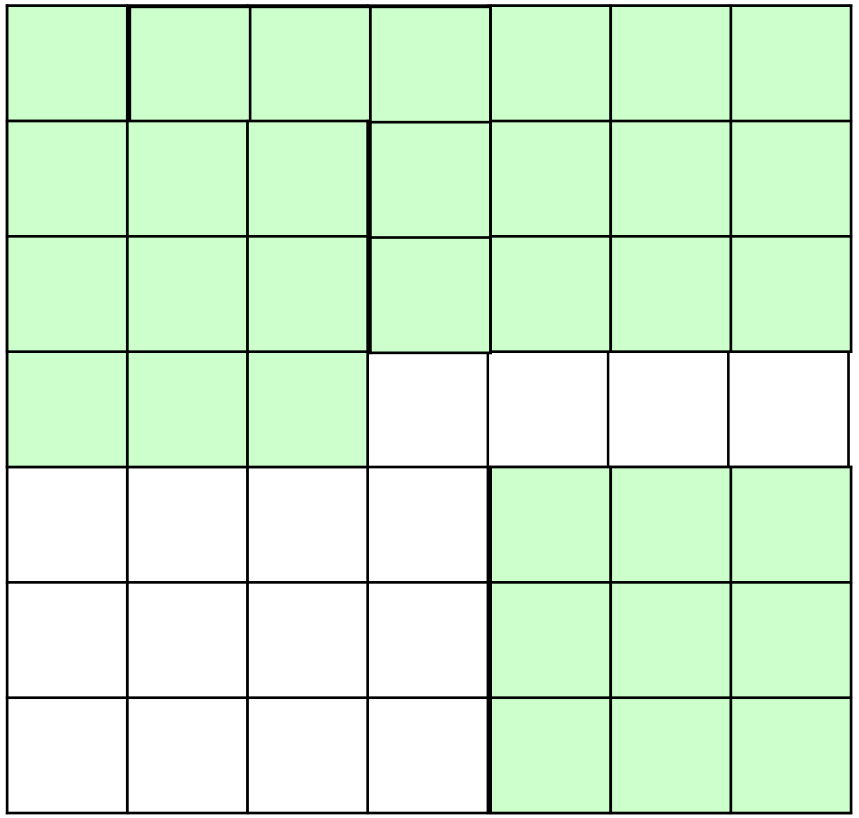
Convolution is commutative, and we can equivalently write:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Neural networks libraries implement cross-correlation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

2D convolution example



Input image: 7x7

Filter size: 3x3

Do the convolution by sliding the filter over all possible image locations.

What is the size of the output?

Other 2D convolution example

$$F[x,y] \qquad \star \qquad H[a,b] \qquad\qquad G[x,y]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

$$G = H \star F$$

Other 2D example

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

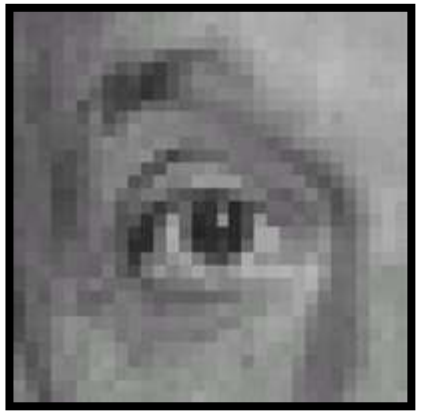
$F[x,y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

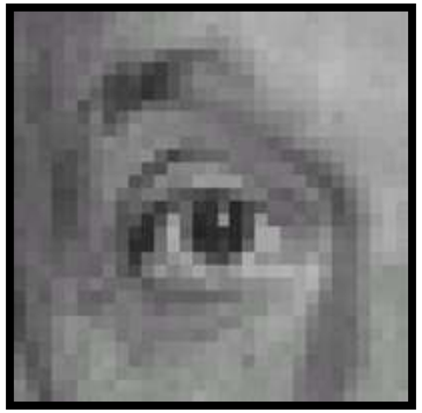
$G[x,y]$

Source: D. Lowe

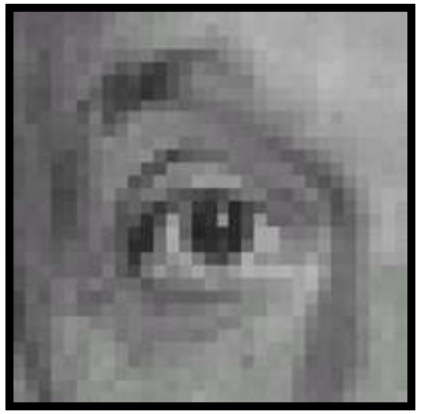
Let’s test your convolution intuition


Original

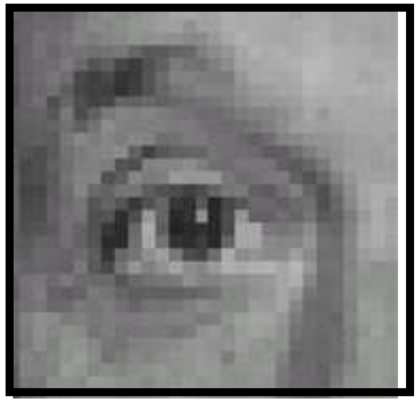
0	0	0
0	1	0
0	0	0


Filtered
(no change)

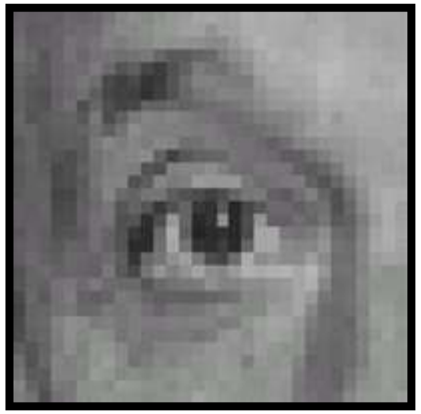
Let’s test your convolution intuition


Original

0	0	0
0	0	1
0	0	0

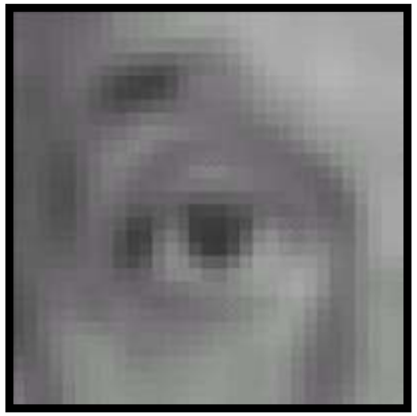

Filtered
(shift left)

Let’s test your convolution intuition



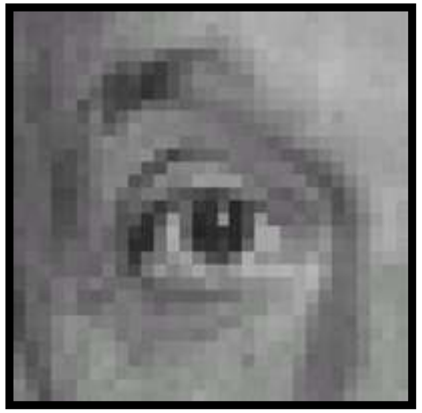
Original

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$



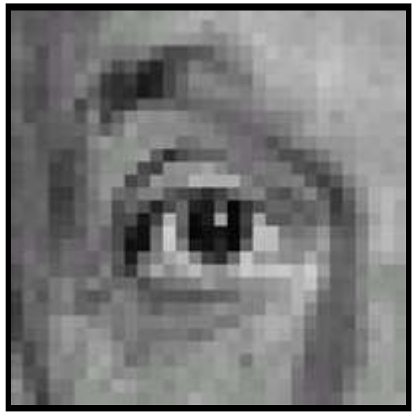
Filtered
(blur)

Let’s test your convolution intuition



Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



Filtered
(sharpening)

Motivations for convolutions

Sparse interaction, or local connectivity.

- *The receptive field of the neuron, or the filter size.*
- *The connections are local in space (width and height), but full in depth.*
- *A set of learnable filters.*

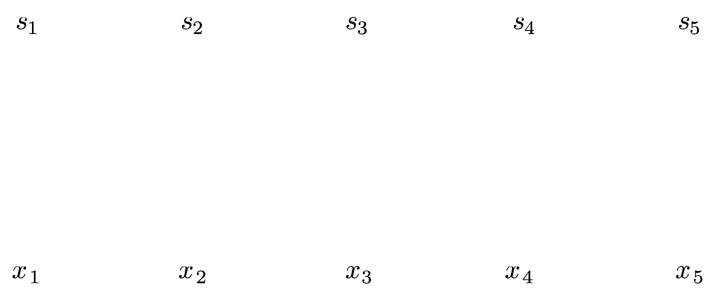
Parameters sharing, the weights are tied.

Equivariant representation (spatially): same operation at different places.

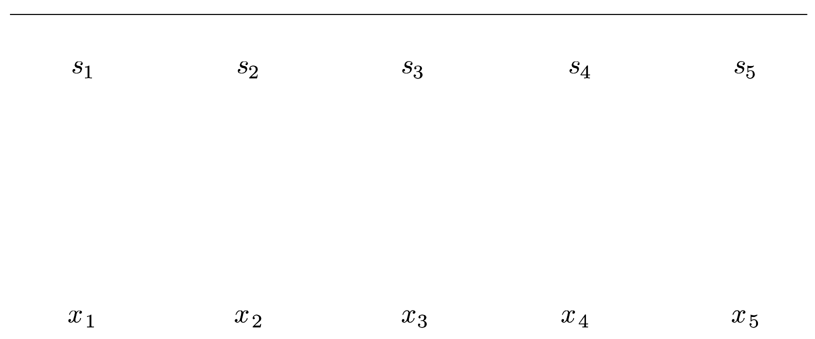
1. Sparsity

Sparse interaction, or local connectivity

- This is accomplished by making the kernel smaller than the input.
- reduces the memory requirements of the model
- improves its statistical efficiency.



sparse connection

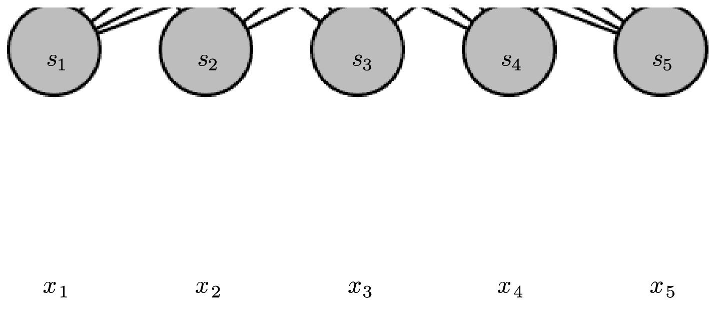


full connection

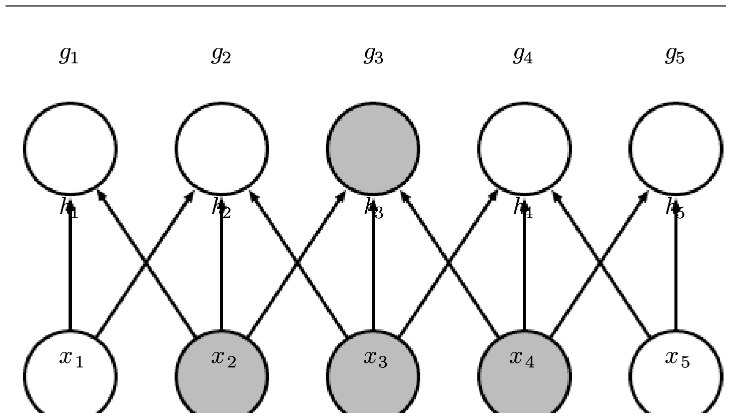
1. Sparsity

Sparse interaction, or local connectivity.

- Receptive field is the kernel/filter size.
- The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers.



convolution with a kernel of width 3

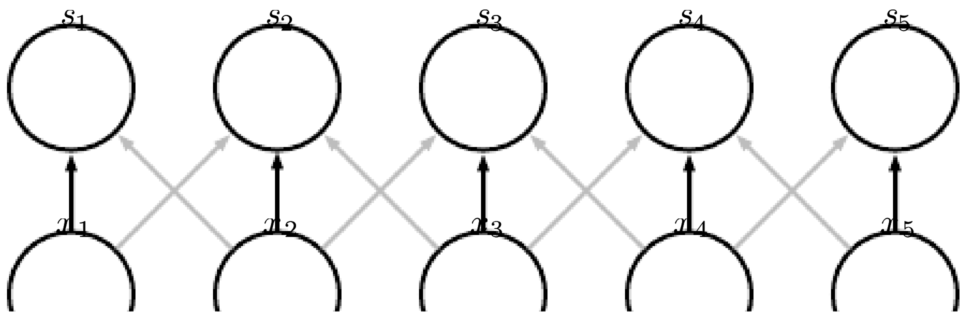


receptive field of the units in the deeper layers

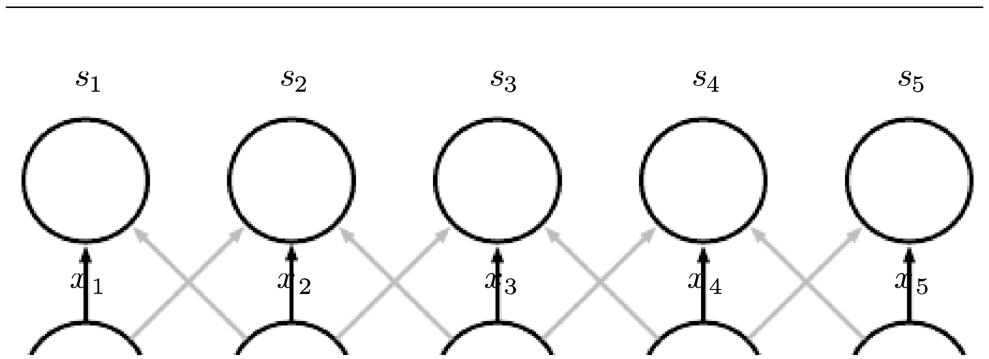
2. Parameter sharing

Parameters sharing, the weights are tied.

- refers to using the same parameter for more than one function in a model.
- each member of the kernel is used at every position of the input.



parameter sharing

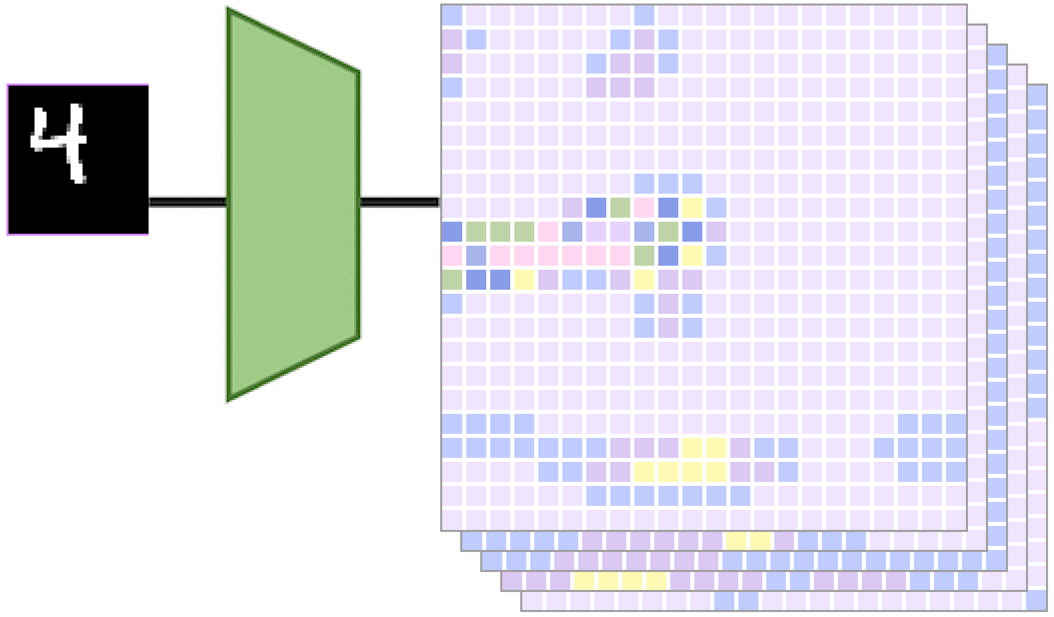


no parameter sharing

3. Equivariance

Equivariant representation

- *Parameter sharing causes the layer to have equivariance to translation.*
- *A function is equivariant if the input changes, the output changes in the same way.*
- *Why do we want equivariance to translation?*

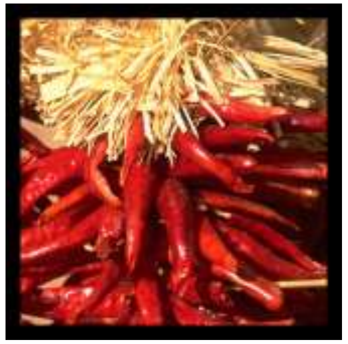


Dealing with borders

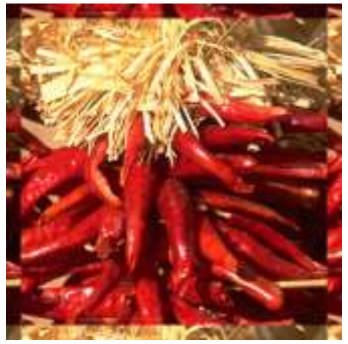
We can’t convole border pixels, as filter windows extends beyond the image.

As is, this means that each convolution makes the image output smaller.

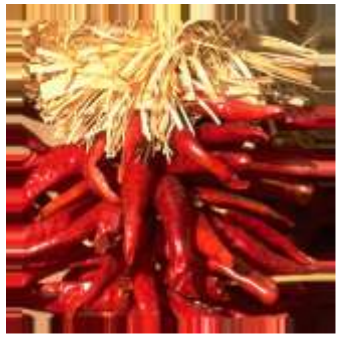
Solution: re-introduce a set of border pixels for every convolution.



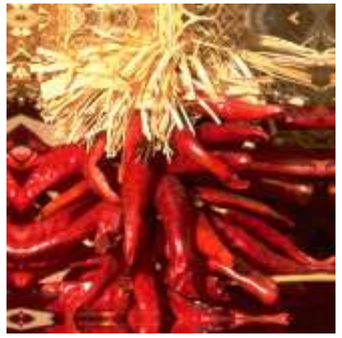
Zero padding



Wrap around



Copy



Reflect

Convolutional networks pipeline

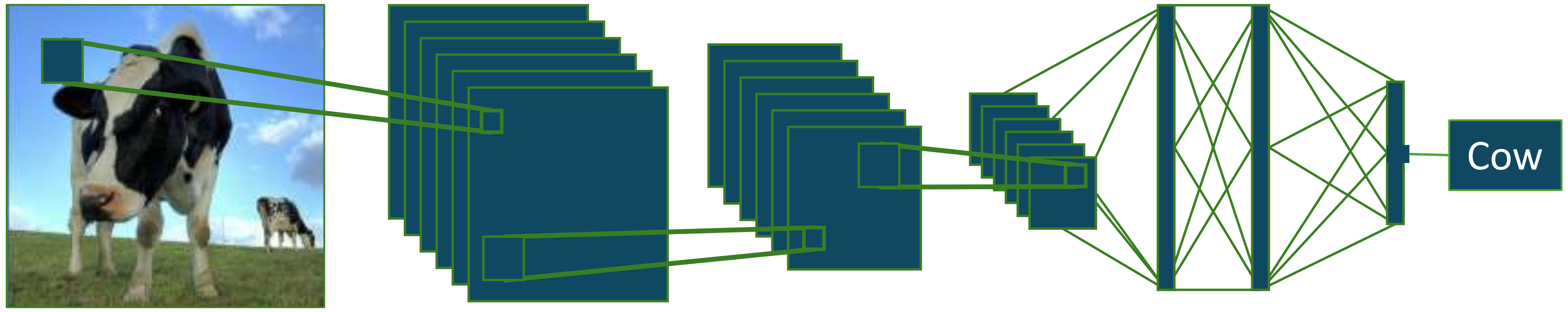
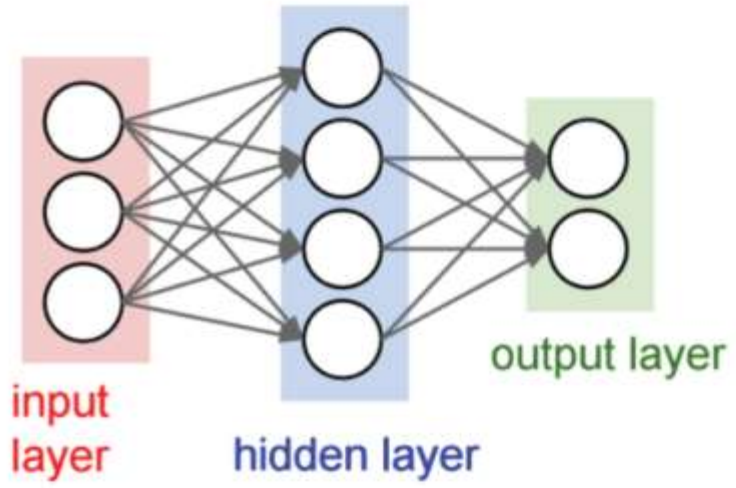


Image is the input, which should go through layers and ultimately predict label.

We want to *learn* the filter values to help us recognize classes.

3D Activations

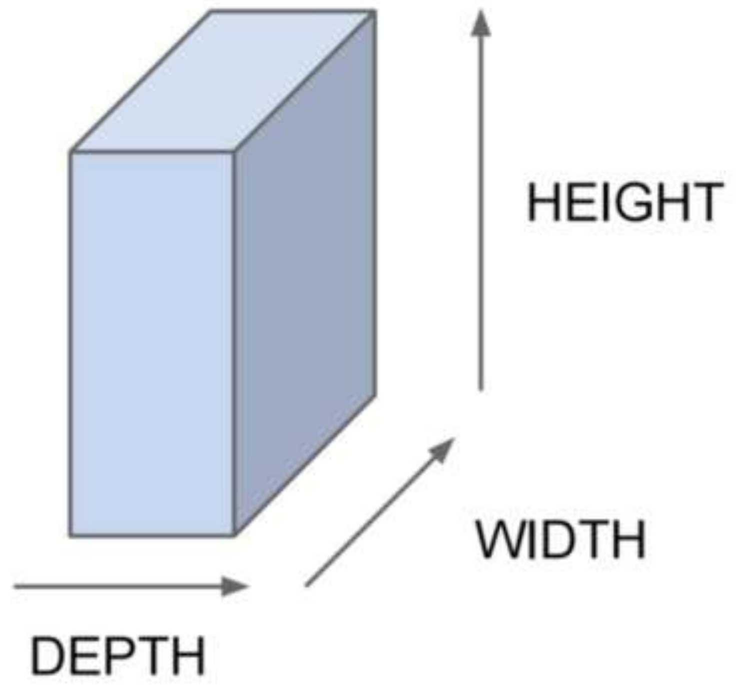
before:



(1D vectors)

3D Activations

All Neural Net activations arranged in **3 dimensions:**

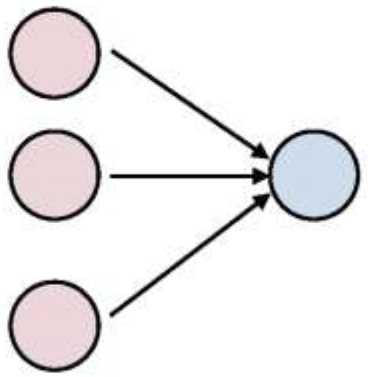


For example, a CIFAR-10 image is a 3x32x32 volume (3 depth — RGB channels, 32 height, 32 width)

Figure: Andrej Karpathy

3D Activations

1D Activations:



3D Activations:

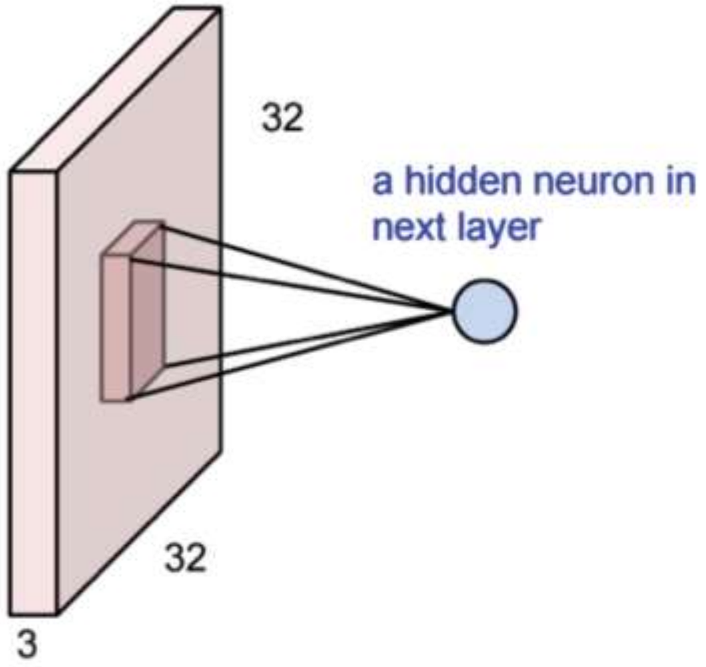


Figure: Andrej Karpathy

3D Activations

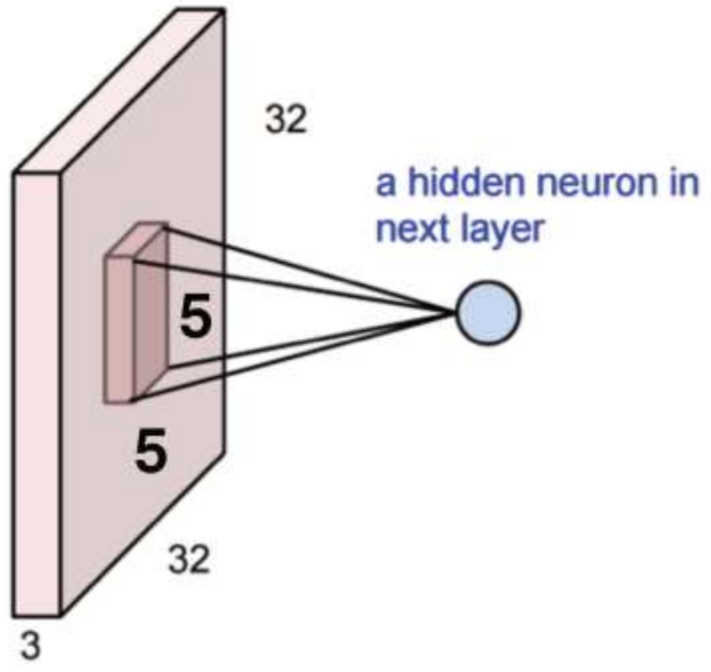


Figure: Andrej Karpathy

- The input is 3x32x32
- This neuron depends on a 3x5x5 chunk of the input
- The neuron also has a 3x5x5 set of weights and a bias (scalar)

3D Activations

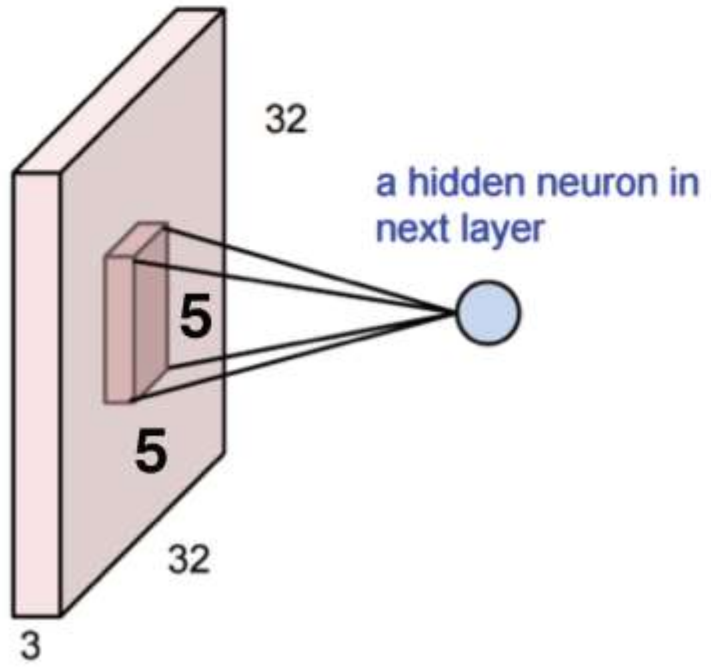


Figure: Andrej Karpathy

- The input is 3x32x32

- This neuron depends on a 3x5x5 chunk of the input

- The neuron also has a 3x5x5 set of weights and a bias (scalar)

3D Activations

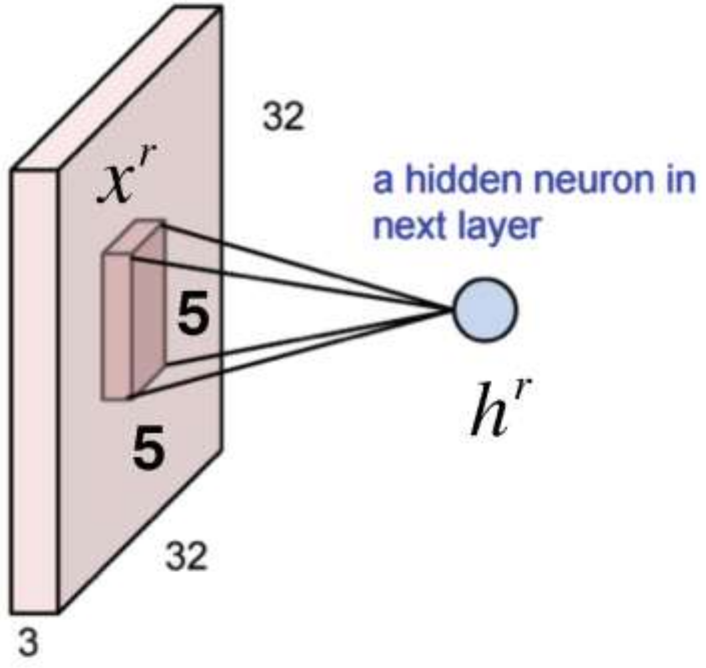


Figure: Andrej Karpathy

Example: consider the region of the input "x^r"

With output neuron h^r

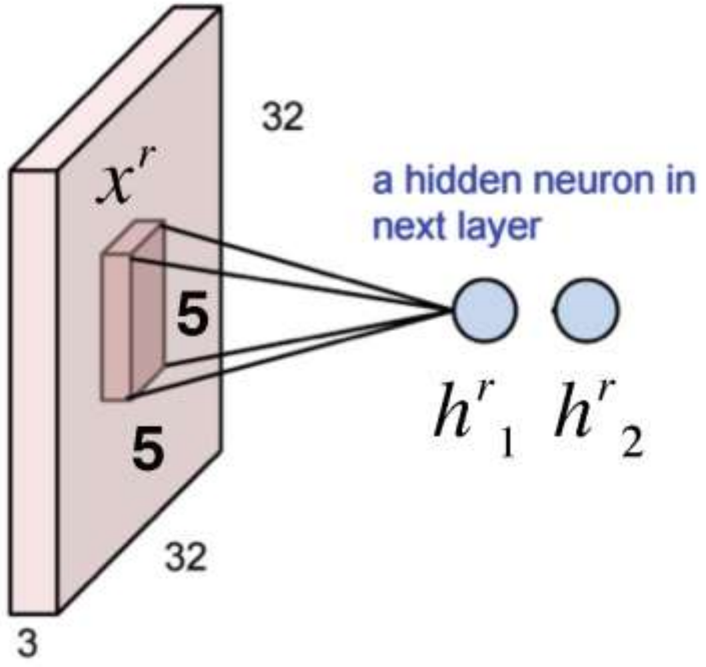
Then the output is:

$$h^r = \sum_{ijk} x^r{}_{ijk} W_{ijk} + b$$

↑

Sum over 3 axes

3D Activations



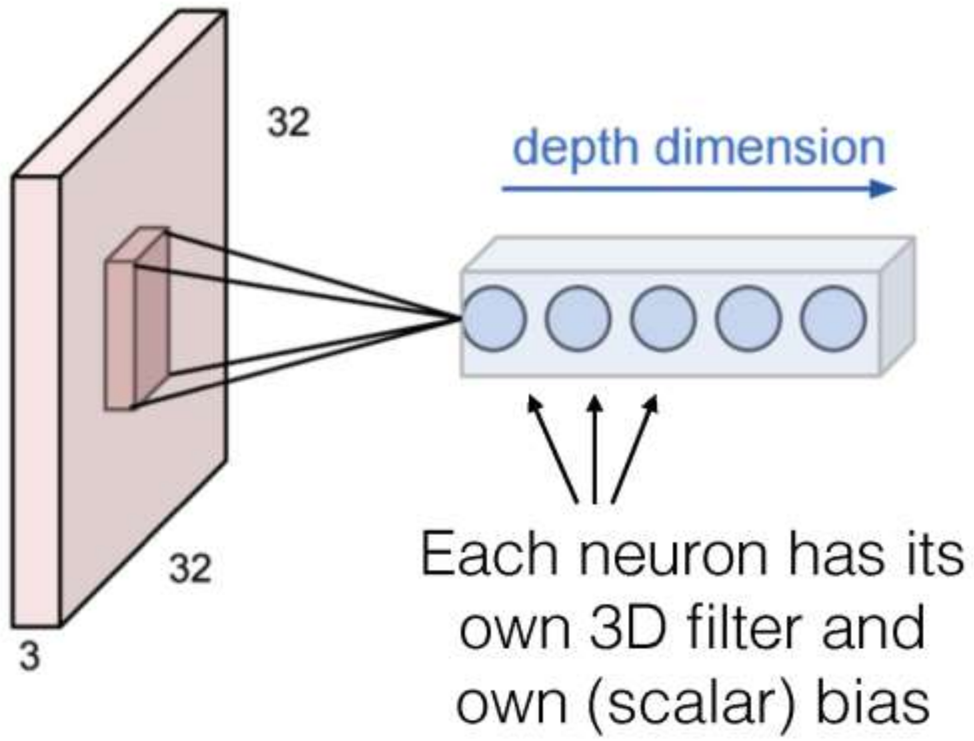
With **2** output neurons

$$h^r{}_1 = \sum_{ijk} x^r{}_{ijk} W_{1ijk} + b_1$$

$$h^r{}_2 = \sum_{ijk} x^r{}_{ijk} W_{2ijk} + b_2$$

Figure: Andrej Karpathy

3D Activations

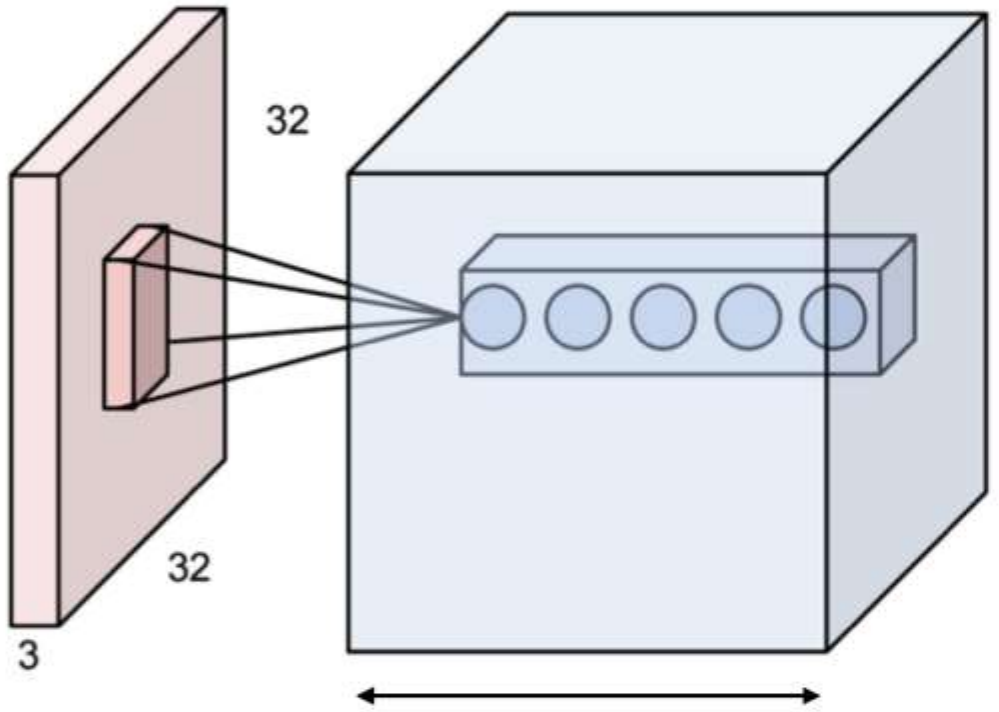


We can keep adding more outputs

These form a column in the output volume: [depth x 1 x 1]

Figure: Andrej Karpathy

3D Activations



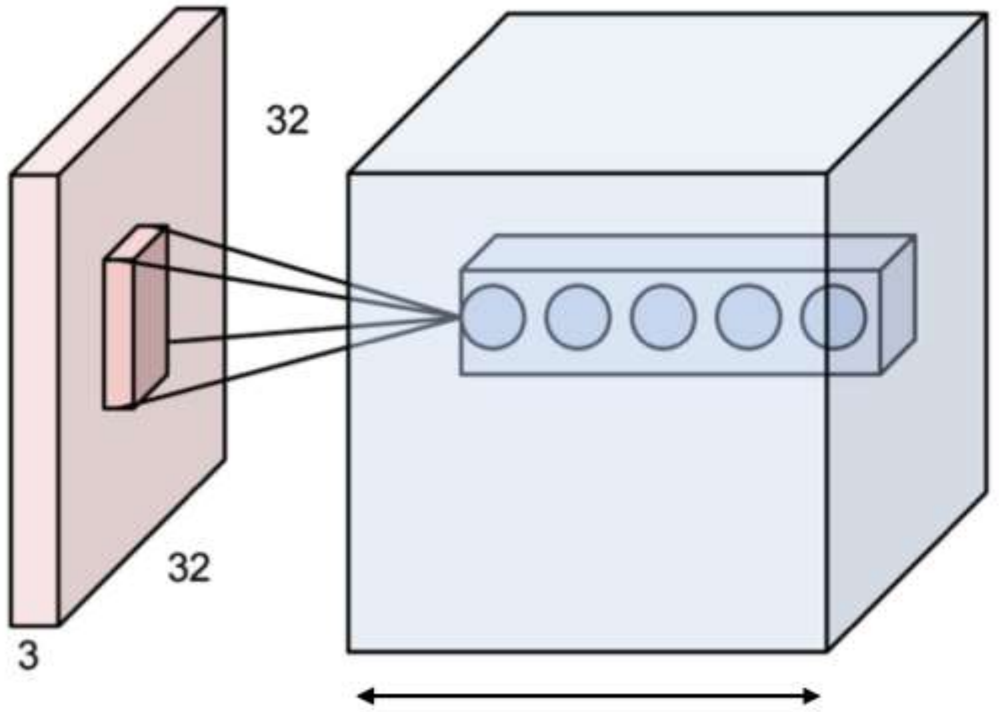
D sets of weights
(also called filters)

Now repeat this across the input

Weight sharing:
Each filter shares the same weights (but each depth index has its own set of weights)

Figure: Andrej Karpathy

3D Activations



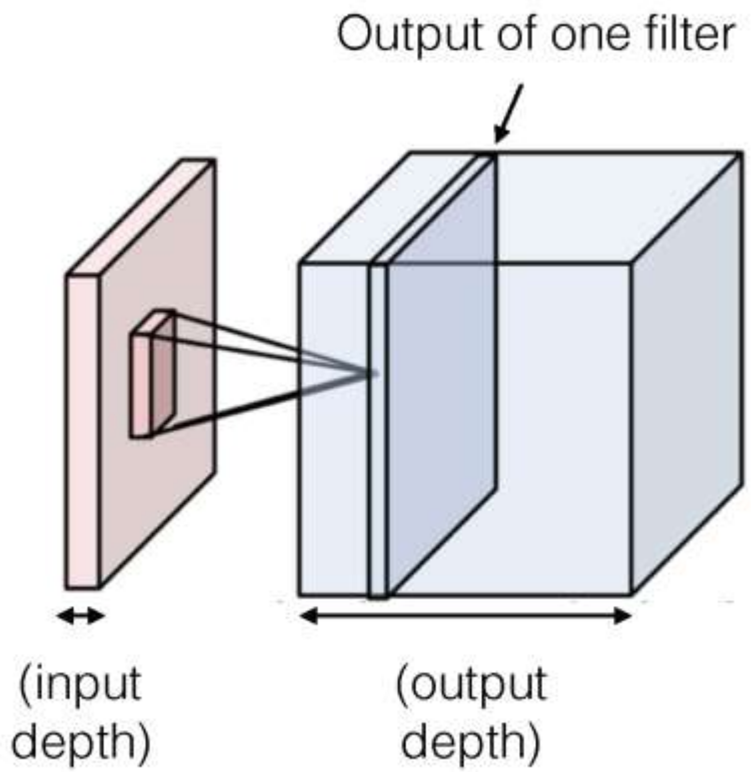
D sets of weights
(also called filters)

With weight sharing, this is called **convolution**

Without weight sharing, this is called a **locally connected layer**

Figure: Andrej Karpathy

3D Activations



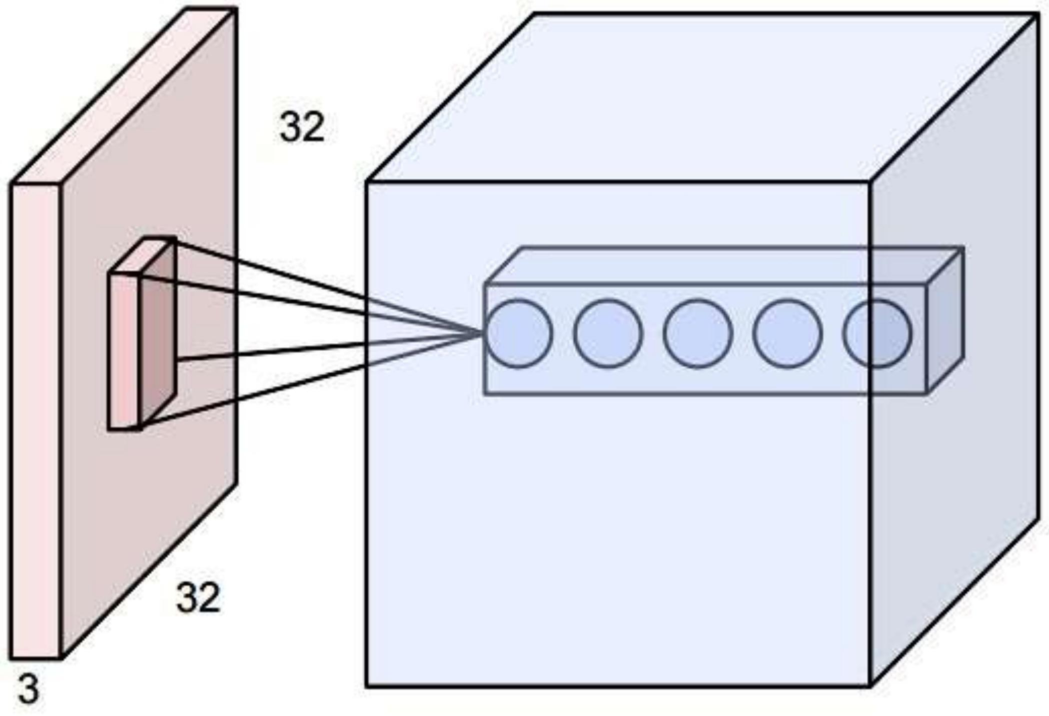
One set of weights gives one slice in the output

To get a 3D output of depth *D*, use *D* different filters

In practice, ConvNets use many filters (~64 to 1024)

All together, the weights are **4** dimensional:
(output depth, input depth, kernel height, kernel width)

Quick test



Q1: What does the 3 mean?

RGB

Q2: What is the output size for D filters (with padding)?

32x32xD

Q3: Does the output size depend on the input size or the filter size (with padding)?

Input size

Filters and layer dimensionality

First layer:

Input: 32x32x3 and A filters of size 5x5x3

Output: 32x32xA

Second layer:

Input: 32x32xA and B filters of size 5x5x[?]

Output size: 32x32x[?]

3D Activations

We can unravel the 3D cube and show each layer separately:

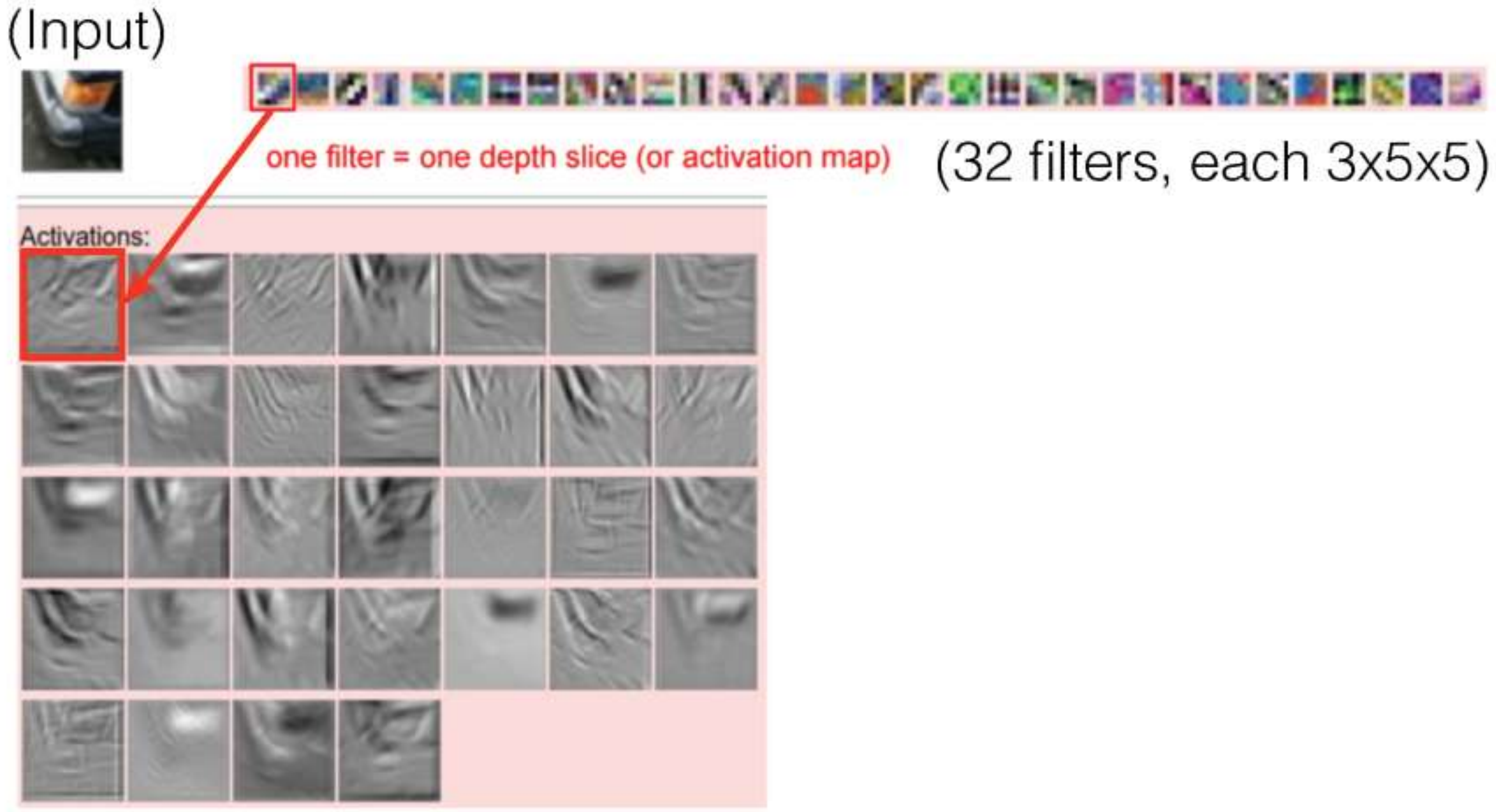


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

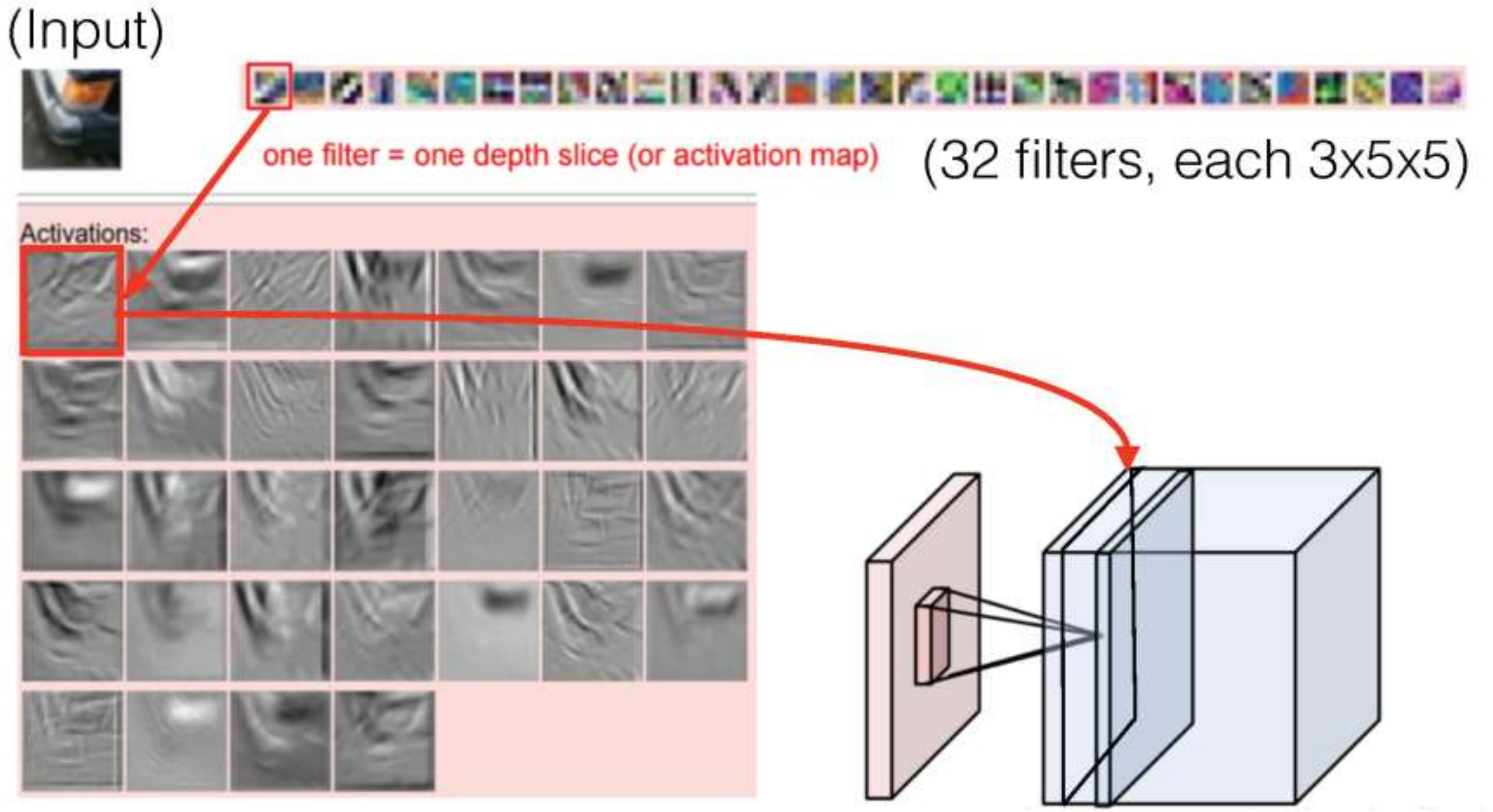


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

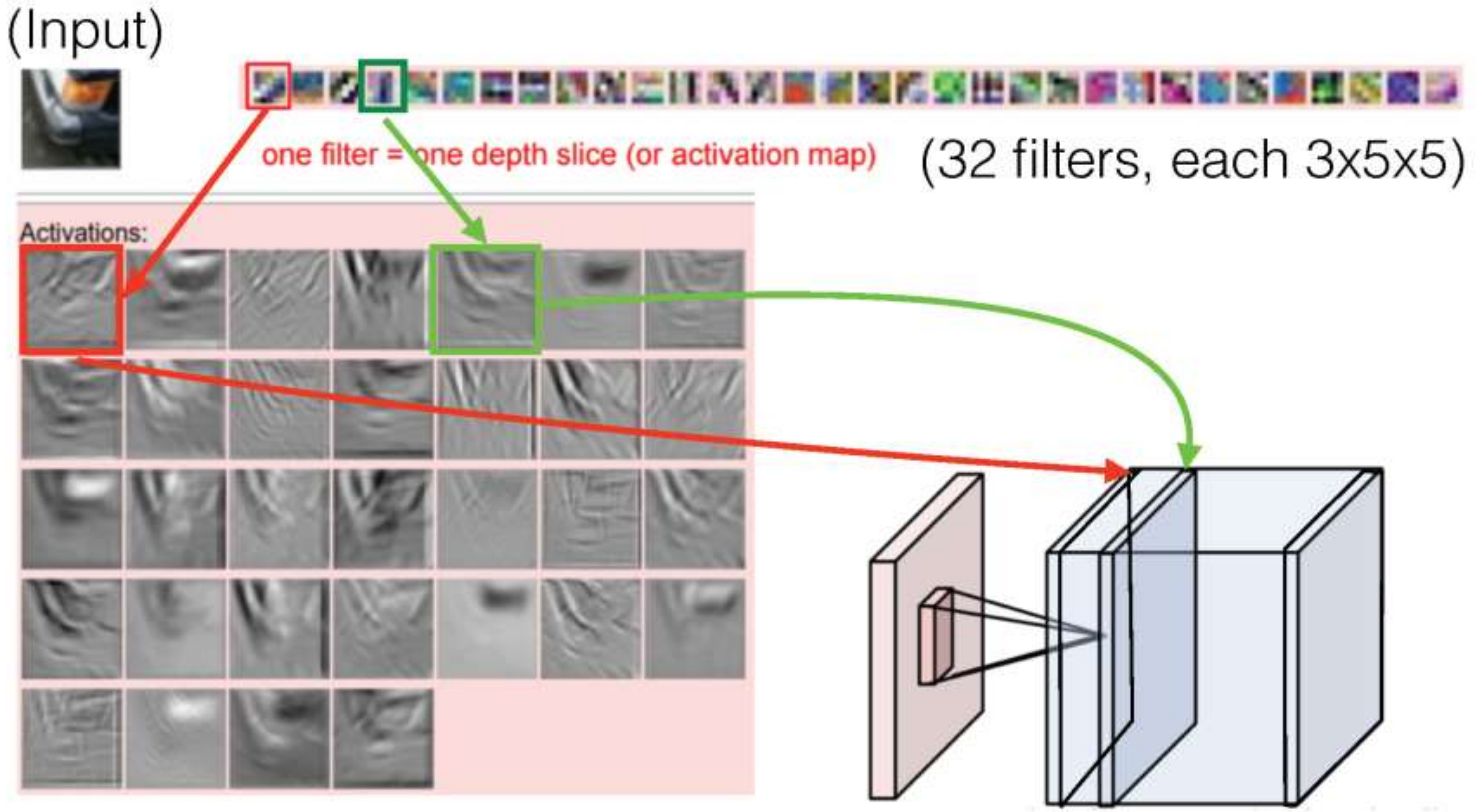


Figure: Andrej Karpathy

3D Activations

We can unravel the 3D cube and show each layer separately:

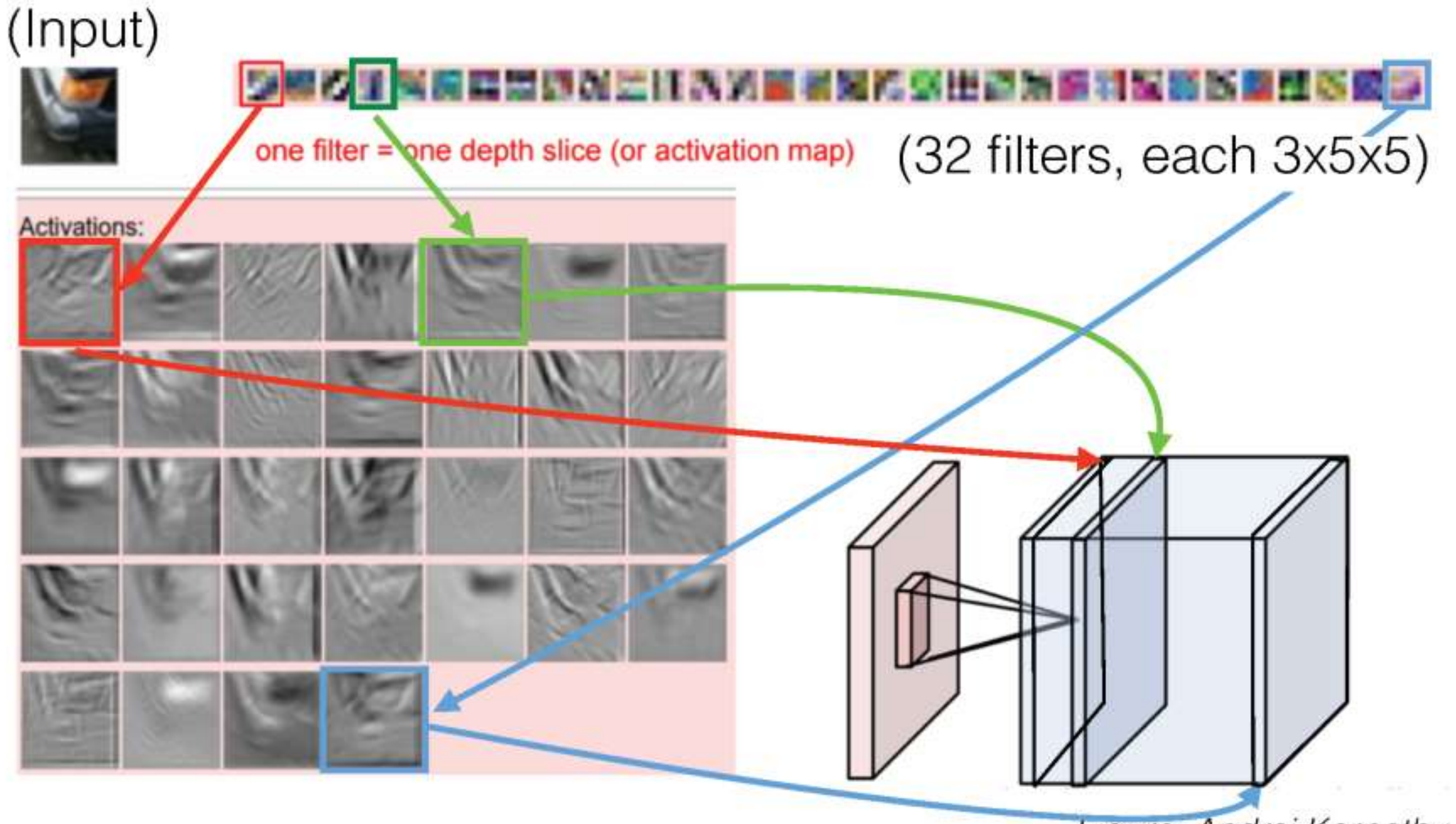
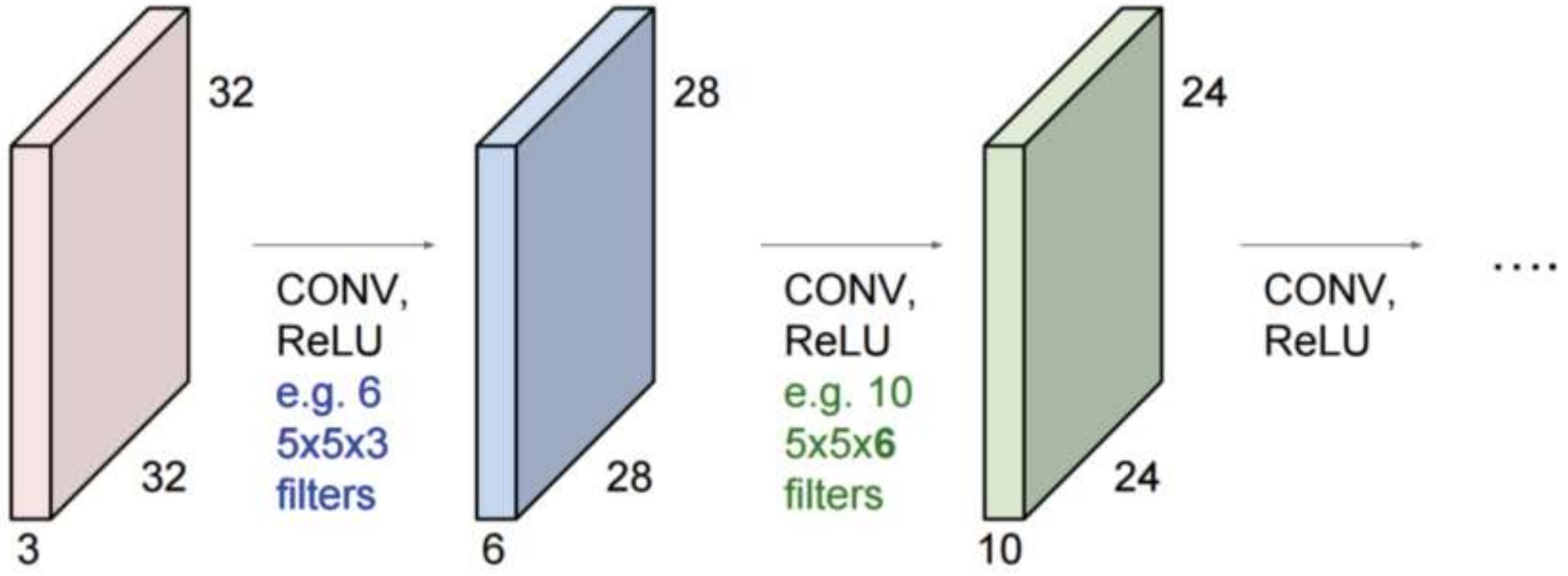


Figure: Andrej Karpathy

Putting it together

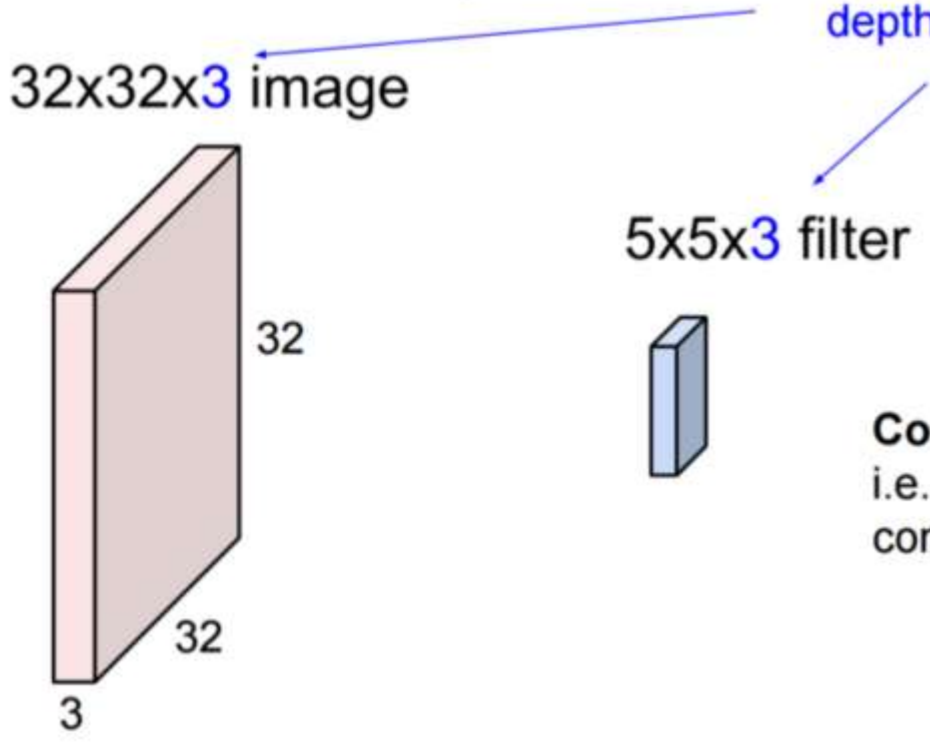
A **ConvNet** is a sequence of convolutional layers, interspersed with activation functions (and possibly other layer types)



Putting it together

Convolution Layer

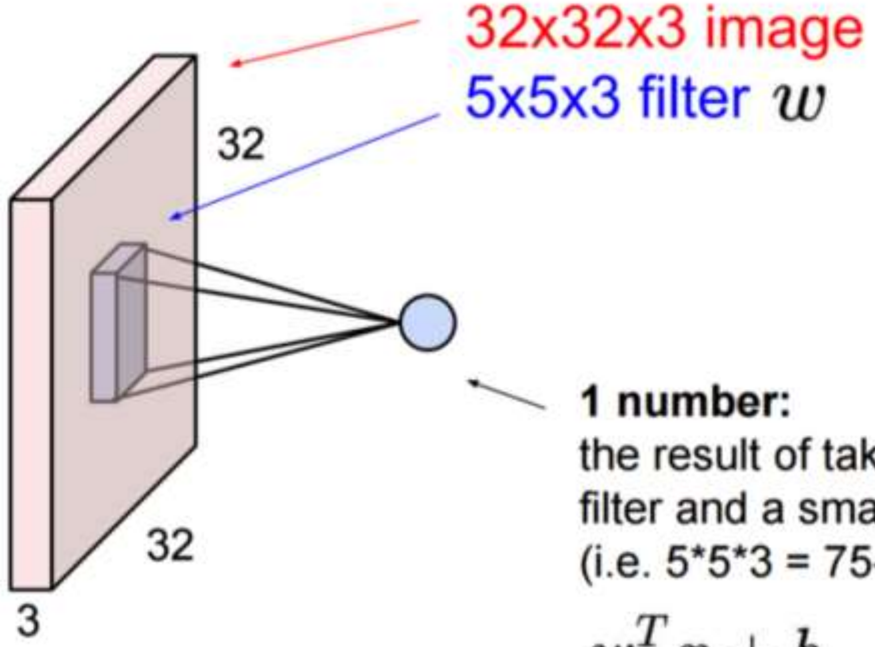
Filters always extend the full depth of the input volume



Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Putting it together

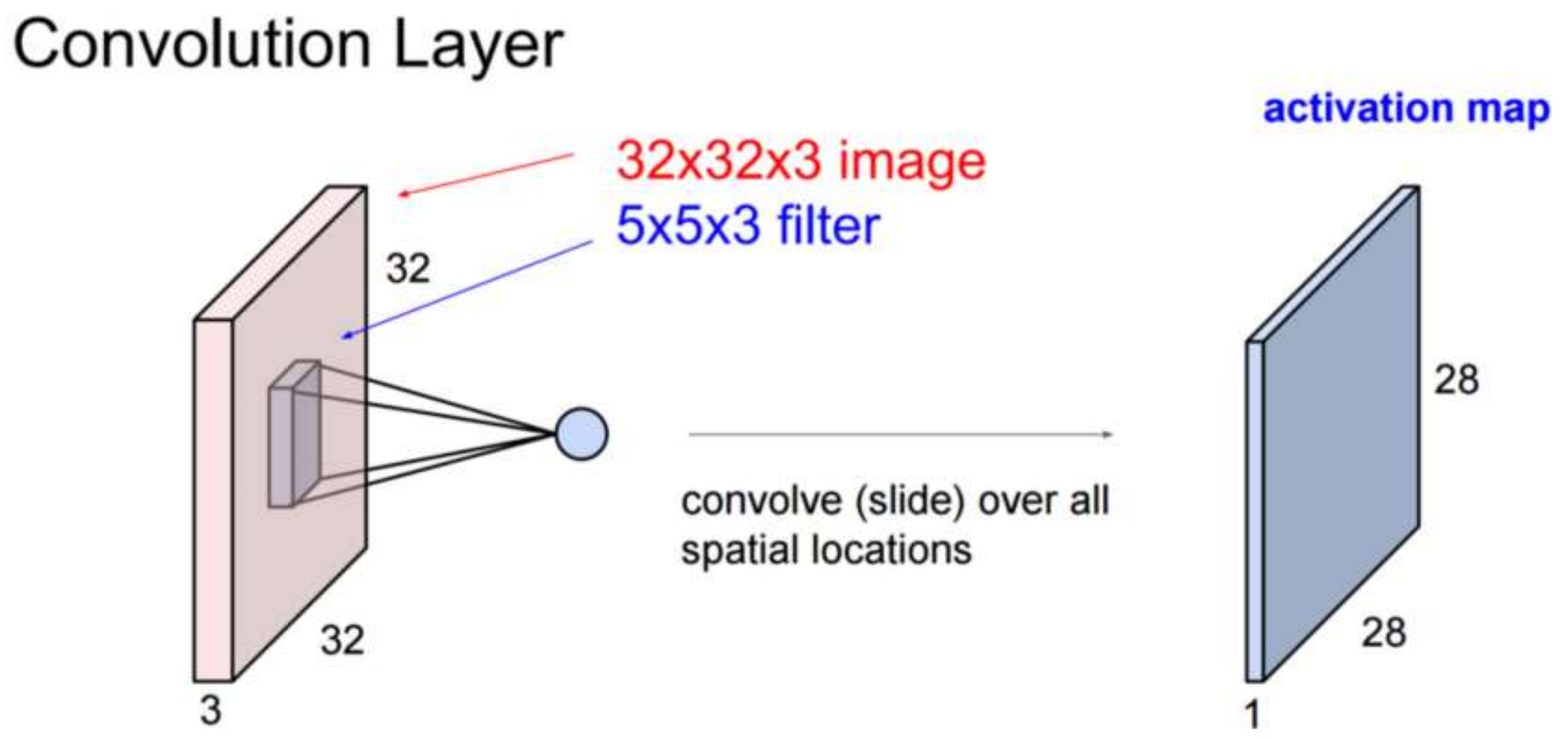
Convolution Layer



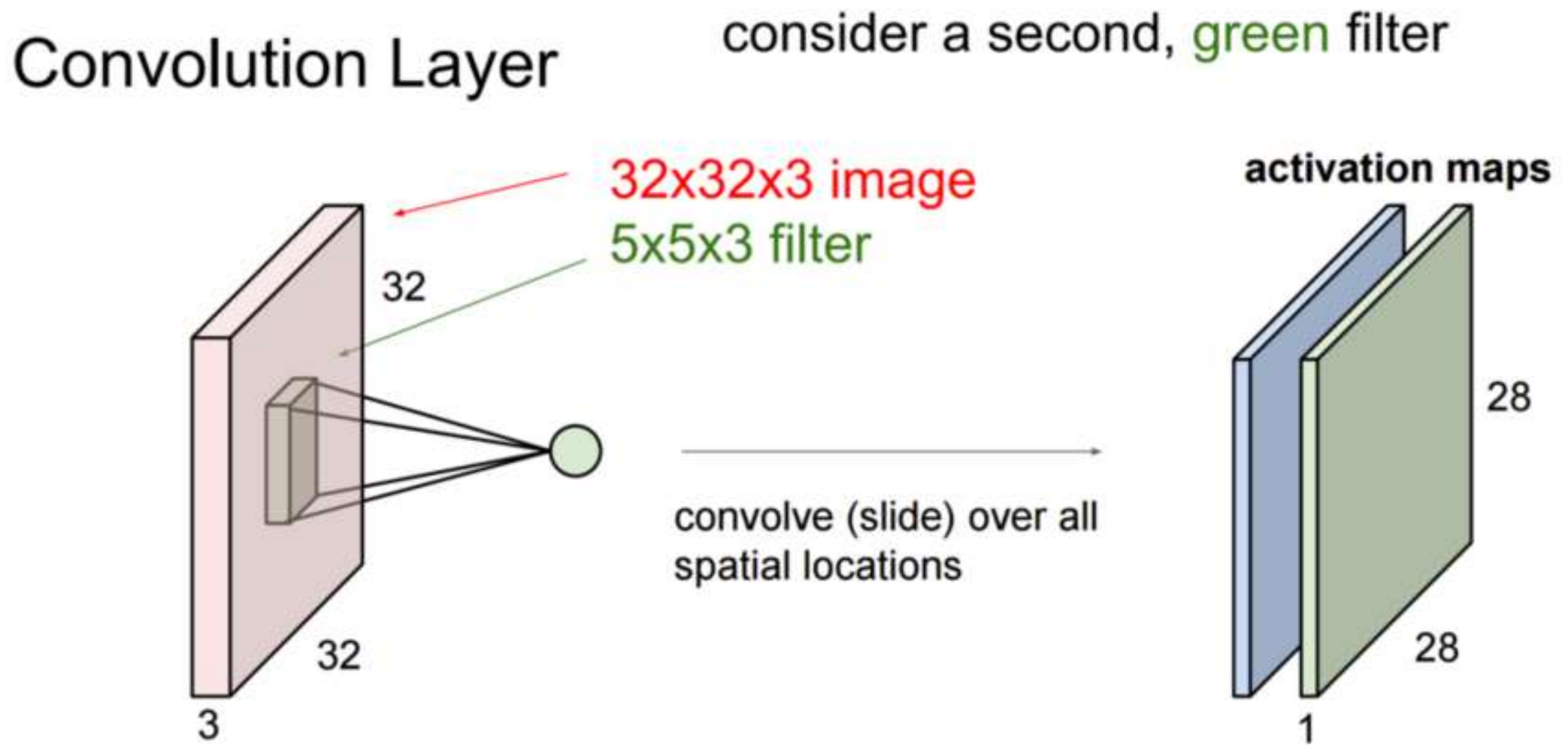
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Putting it together

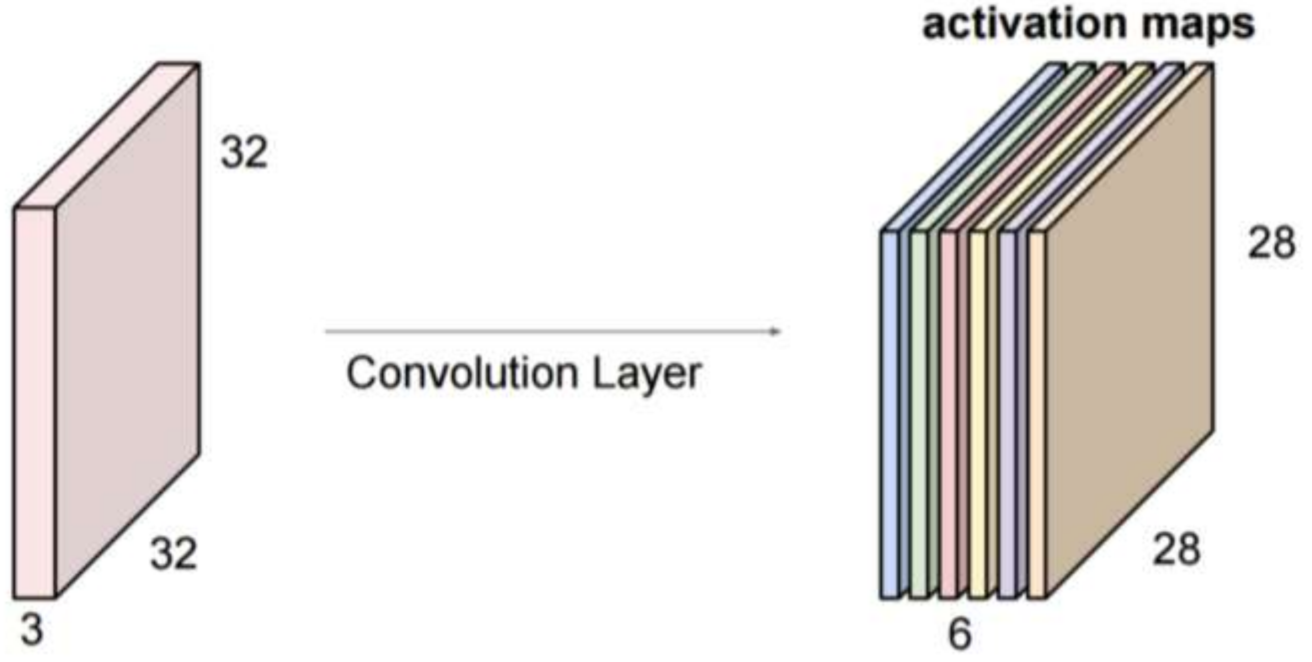


Putting it together


Convolution Layer
consider a second, green filter
32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all spatial locations
activation maps
28
28
1

Putting it together

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

Break

Pooling

For most ConvNets, **convolution** is often followed by **pooling**:

- Creates a smaller representation while retaining the most important information
- The "max" operation is the most common
- Why might "avg" be a poor choice?

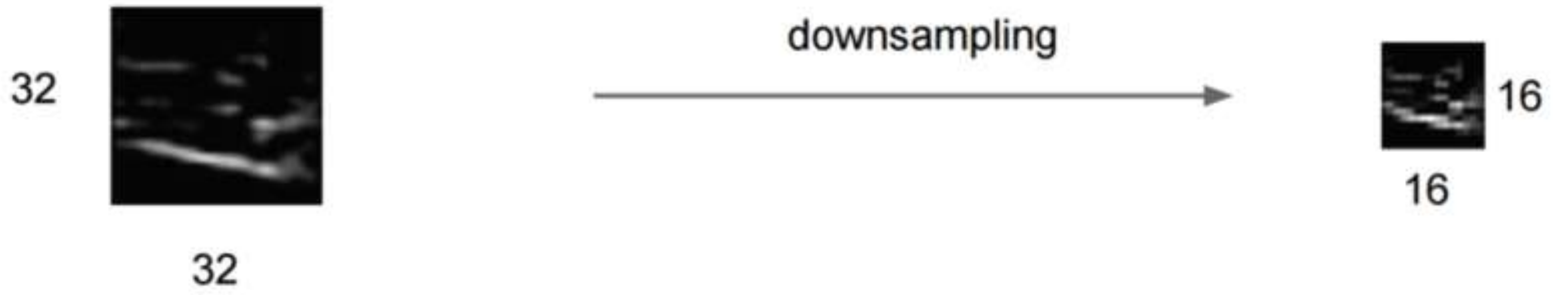
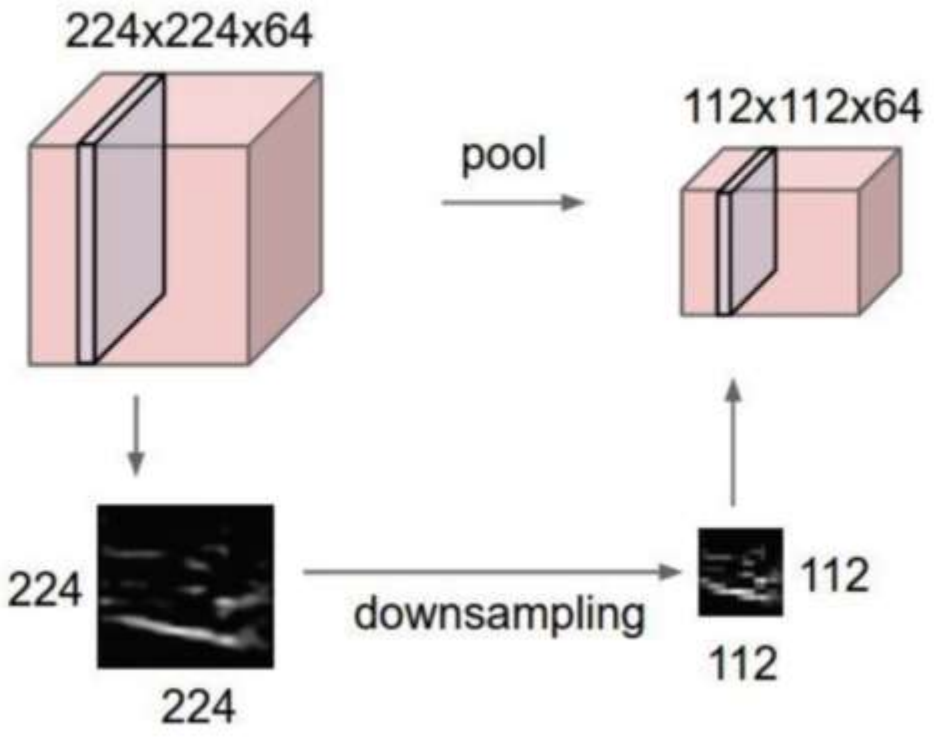


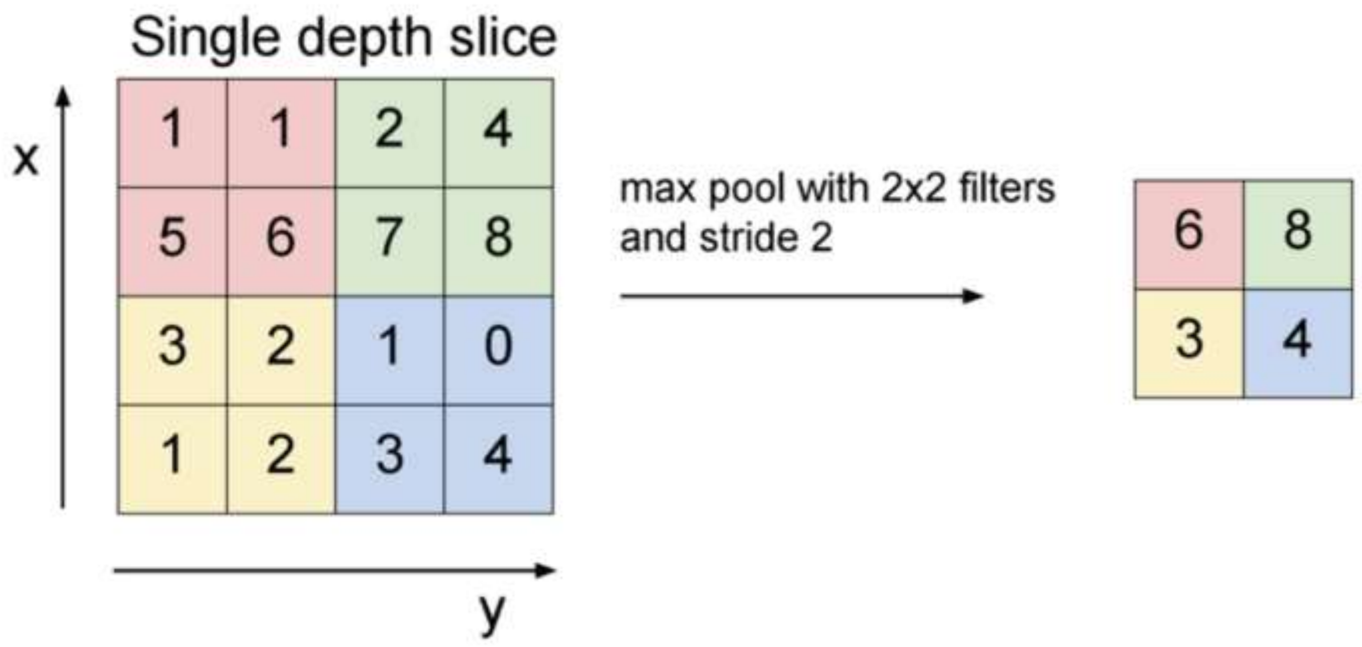
Figure: Andrej Karpathy

Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently:



Max Pooling



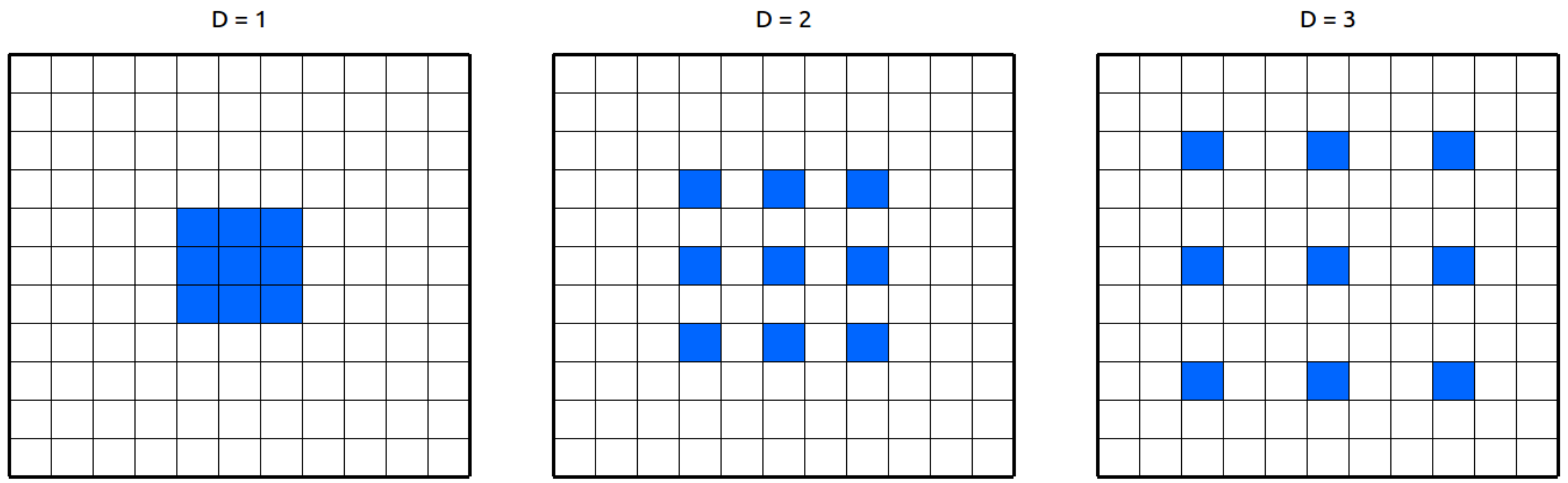
What's the backprop rule for max pooling?

- In the forward pass, store the index that took the max
- The backprop gradient is the input gradient at that index

Figure: Andrej Karpathy

Dilated convolutions

Enlarge receptive field without increasing parameter count.



1x1 convolutions

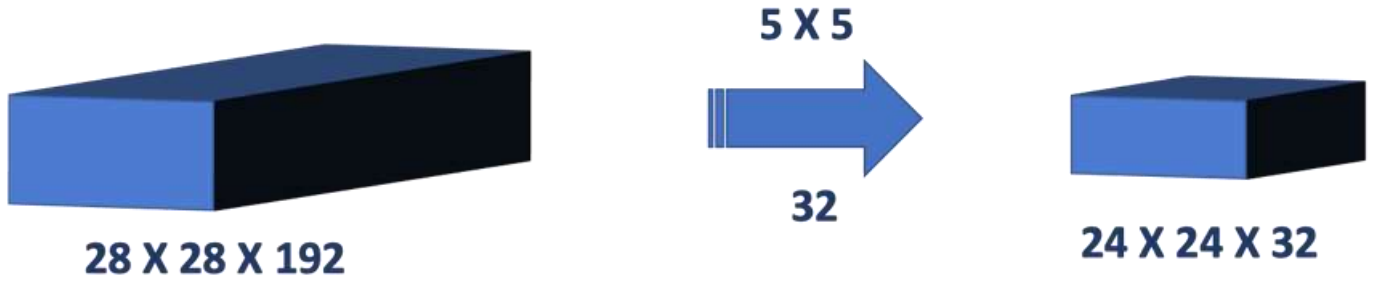
Generally, the goal of a convolutional layer is to learn local spatial filters.

Special case: 1x1 convolutions.

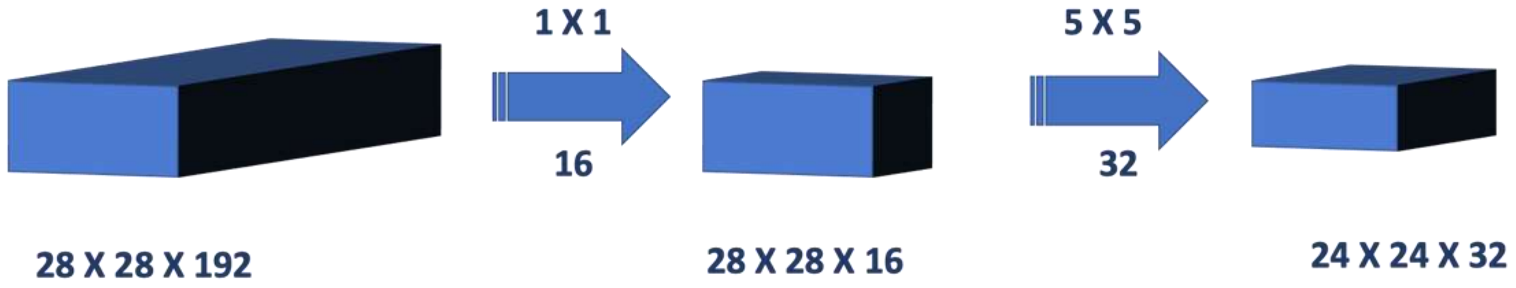
Example: input is [32x32x3], then a 1x1 convolution is a 3-D dot product.

1x1 convolutions learn to mix information across channels.

1x1 convolutions

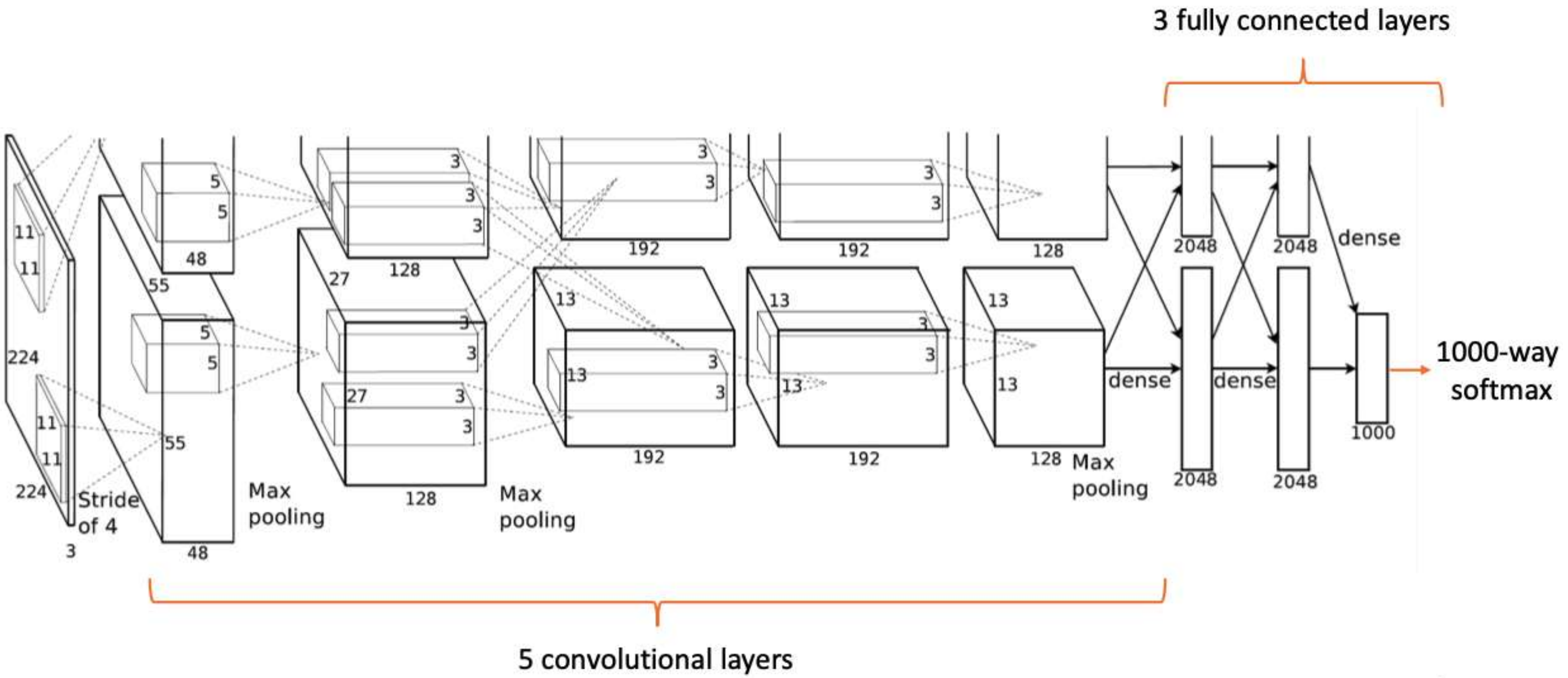


Number of Operations : (28X28X32) X (5X5X192) = 120.422 Million Ops



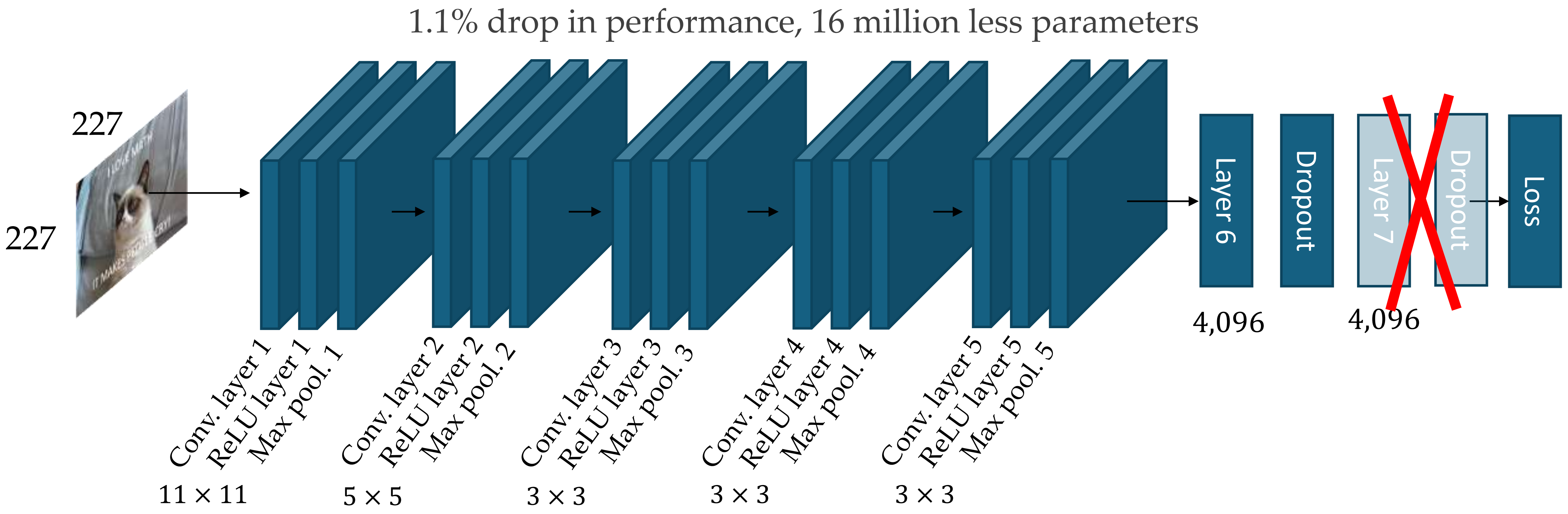
Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

AlexNet

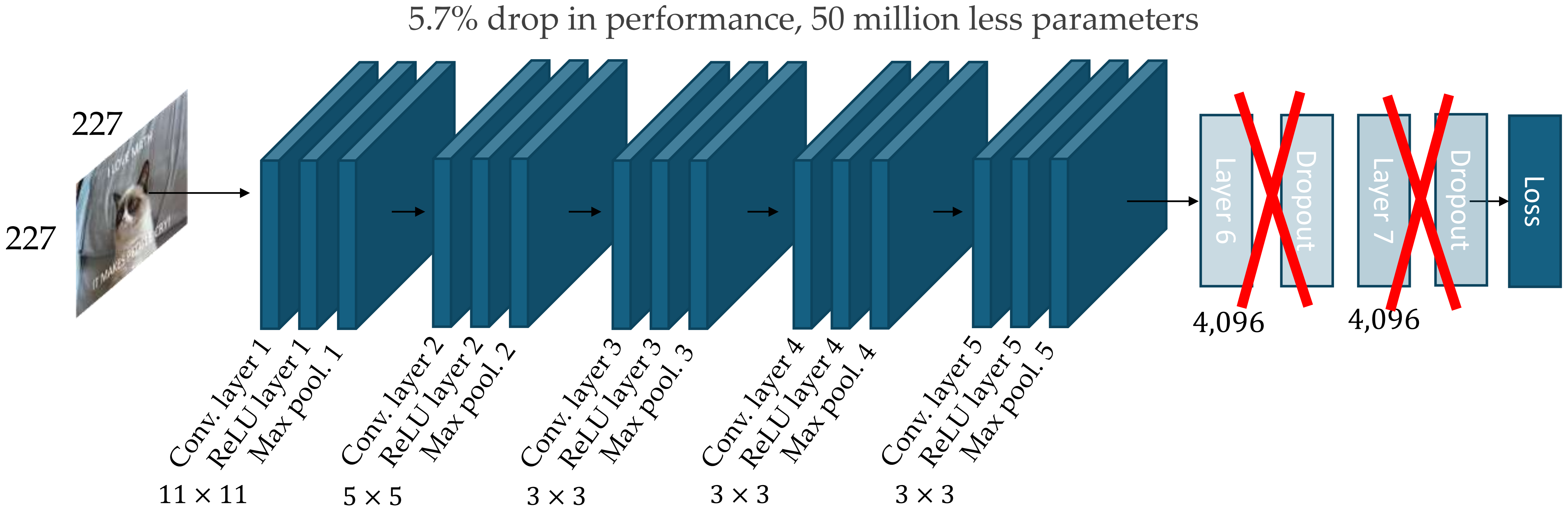


Where are most of the parameters?

Removing parameters

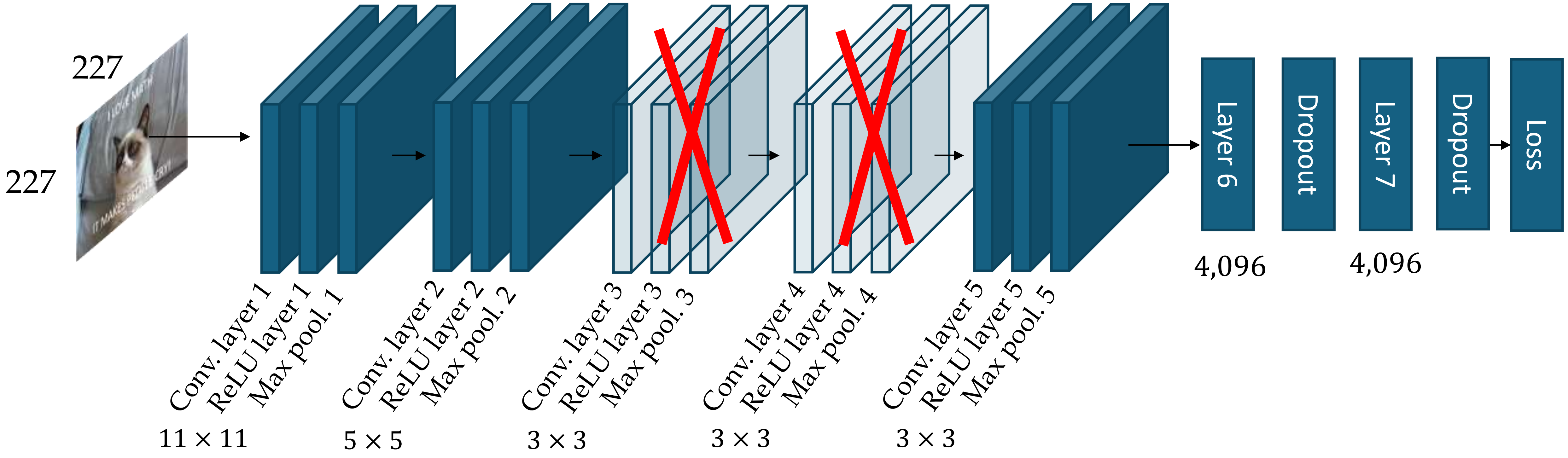


Removing parameters



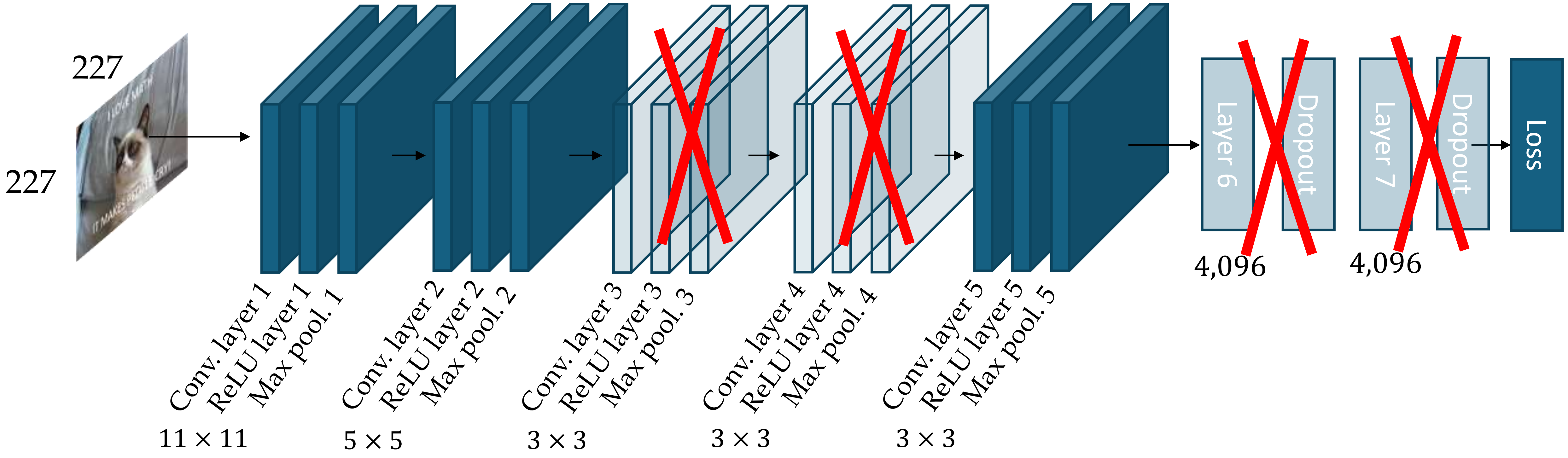
Removing parameters

3.0% drop in performance, 1 million less parameters. Why?



Removing parameters

33.5% drop in performance. Depth is crucial.



Key ingredient for deeper networks: residuals

Let’s say we have the neural network nonlinearity $\mathrm{a} = F(x)$.

Perhaps easier to learn a function $\mathrm{a} = F(x)$ to model differences $\mathrm{a} \sim \delta \mathrm{y}$ than to model absolutes $\mathrm{a} \sim \mathrm{y}$.

Otherwise, you may need to model the magnitude and the direction of activations

Think of it like in input normalization → you normalize around 0

Think of it like in regression → you model differences around the mean value

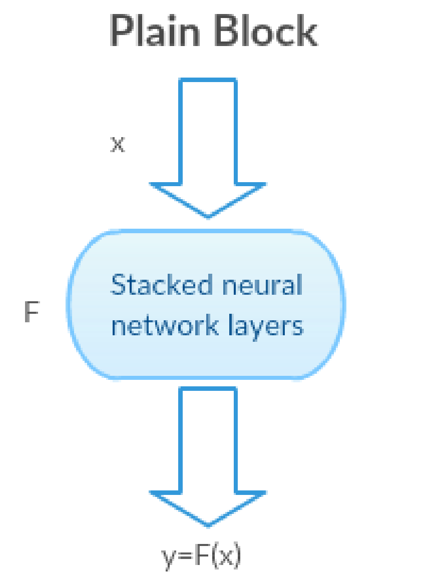
“Residual idea”: Let neural networks explicitly model difference mappings

$$F(x) = H(x) - x \Rightarrow H(x) = F(x) + x$$

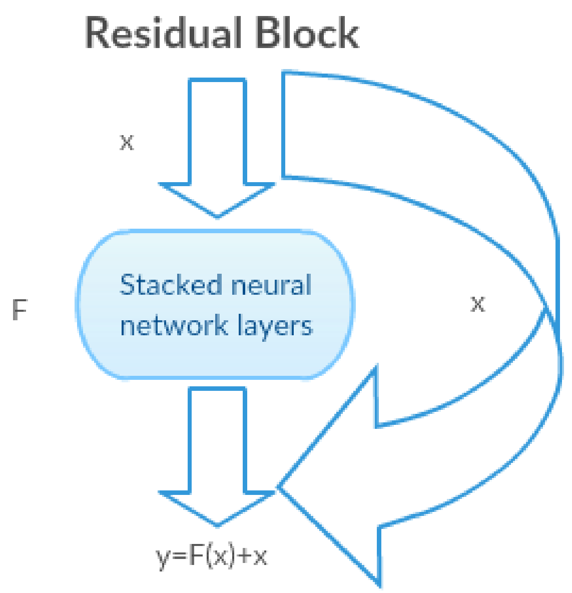
$F(x)$ are the stacked nonlinearities

x is the input to the nonlinear layer.

The residual block



Hard to get F(x)=x and make y=x an identity mapping



Easy to get F(x)=0 and make y=x an identity mapping

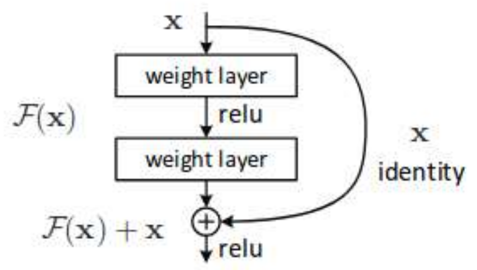
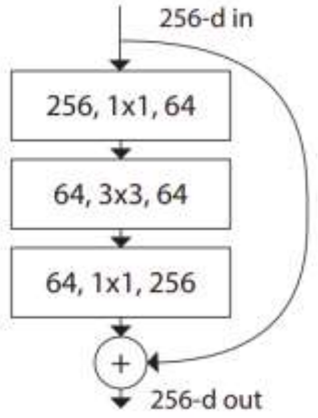
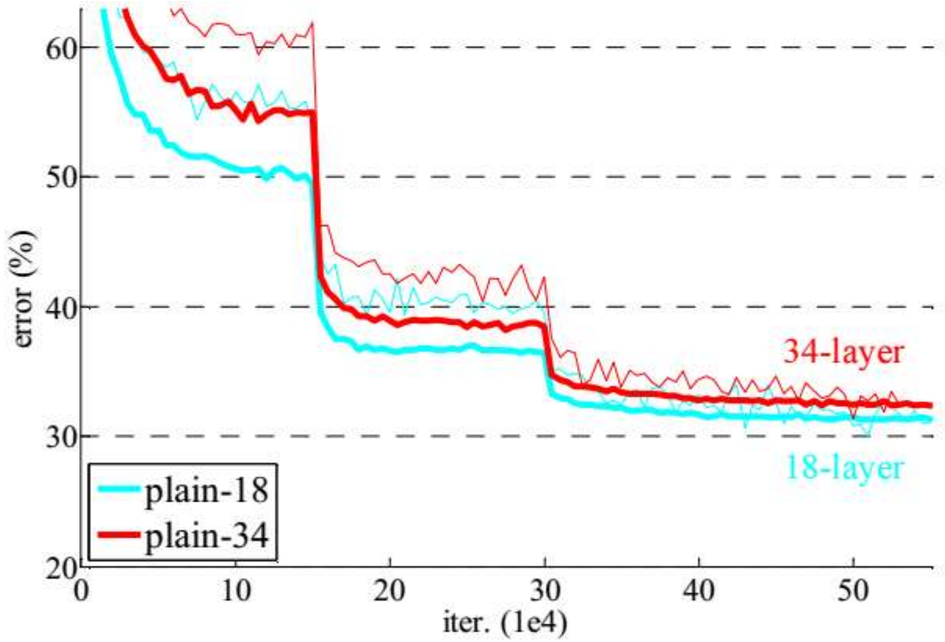

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

Example



Performance degration gone



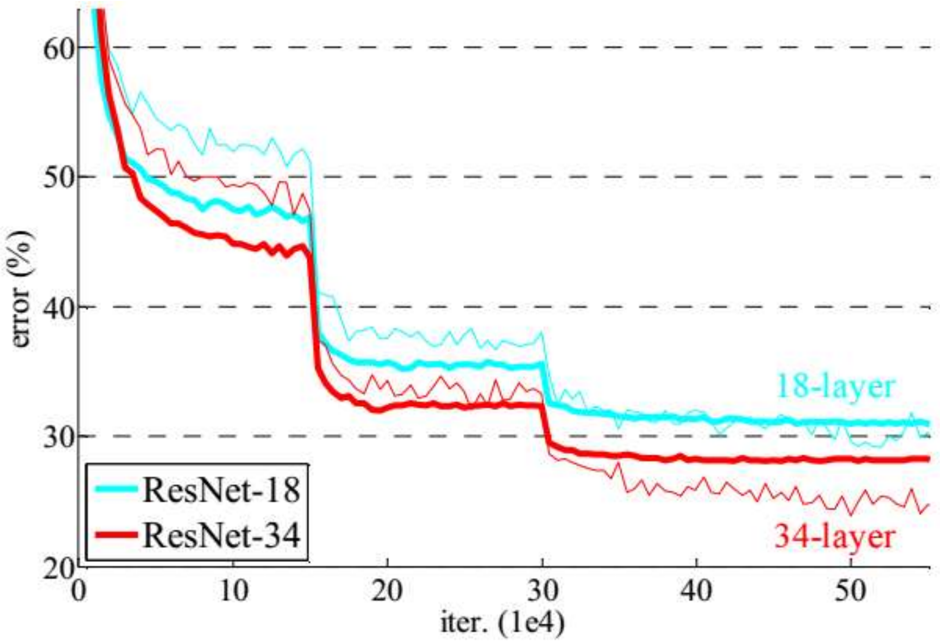
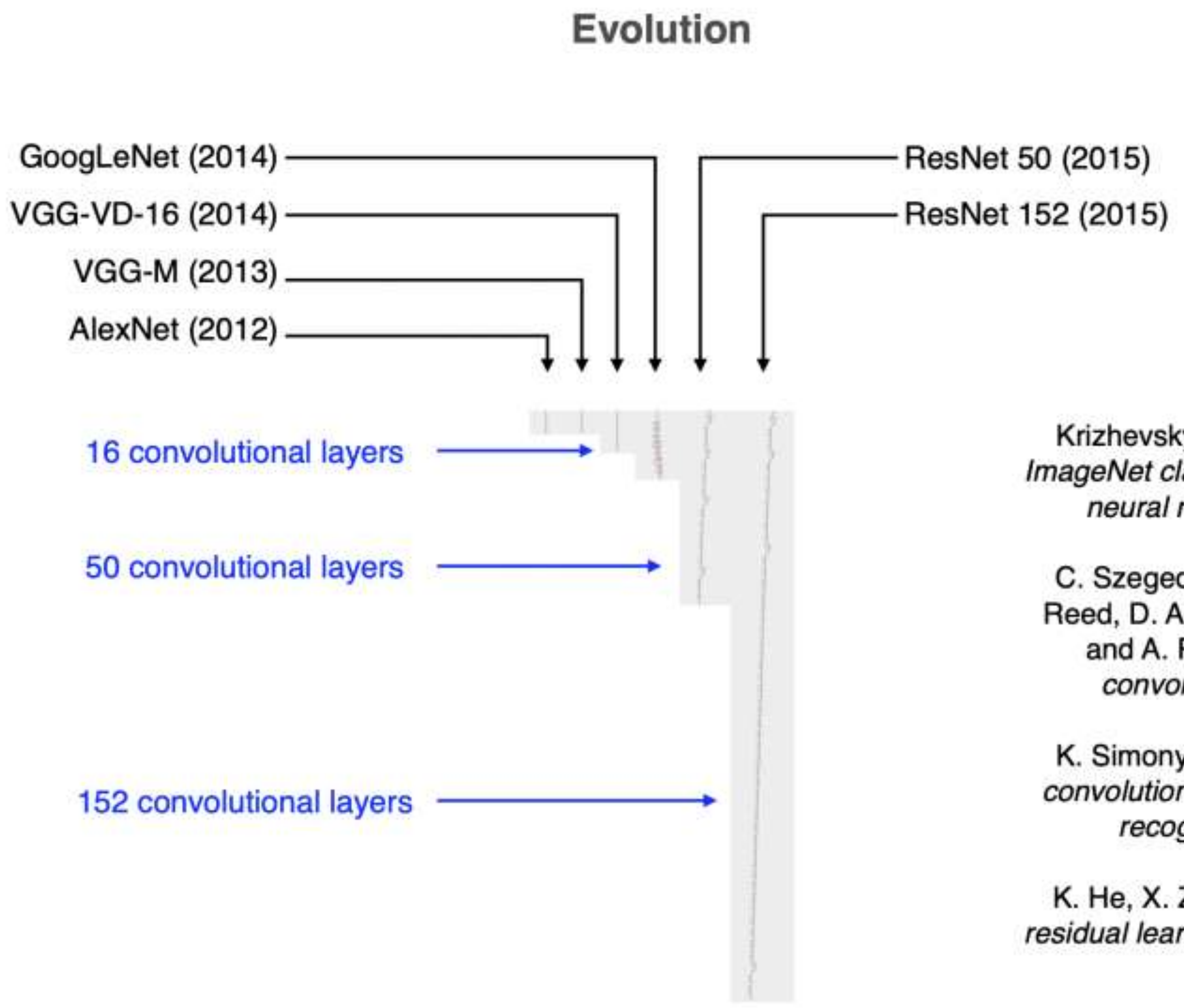


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

ResNet is a record-breaker: (1) depth



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.

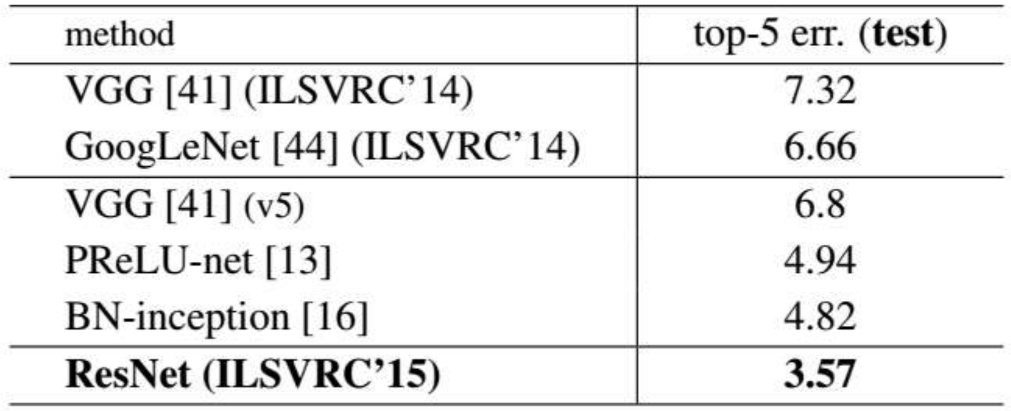
K. Simonyan and A. Zisserman. *Very deep convolutional networks for large-scale image recognition*. In Proc. ICLR, 2015.

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

Image credit: Andrea Vedaldi

ResNet is a record-breaker: (2) performance

Very low ImageNet errors with networks that can grow to 1000 layers.

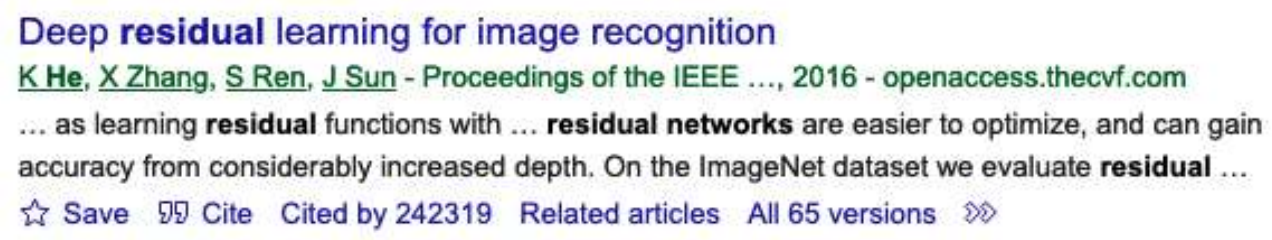


method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

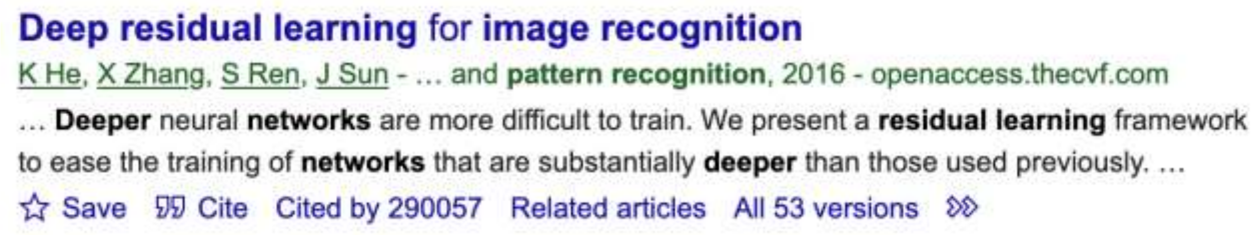
ResNet is a record-breaker: (3) influence

DL1 24/25:



Deep **residual** learning for image recognition
K He, X Zhang, S Ren, J Sun - Proceedings of the IEEE …, 2016 - openaccess.thecvf.com
… as learning **residual** functions with … **residual networks** are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate **residual** …

DL1 25/26:



Deep residual learning for **image recognition**
K He, X Zhang, S Ren, J Sun - … and **pattern recognition**, 2016 - openaccess.thecvf.com
… **Deeper** neural **networks** are more difficult to train. We present a **residual learning** framework to ease the training of **networks** that are substantially **deeper** than those used previously. …

Convolutional networks beyond classification

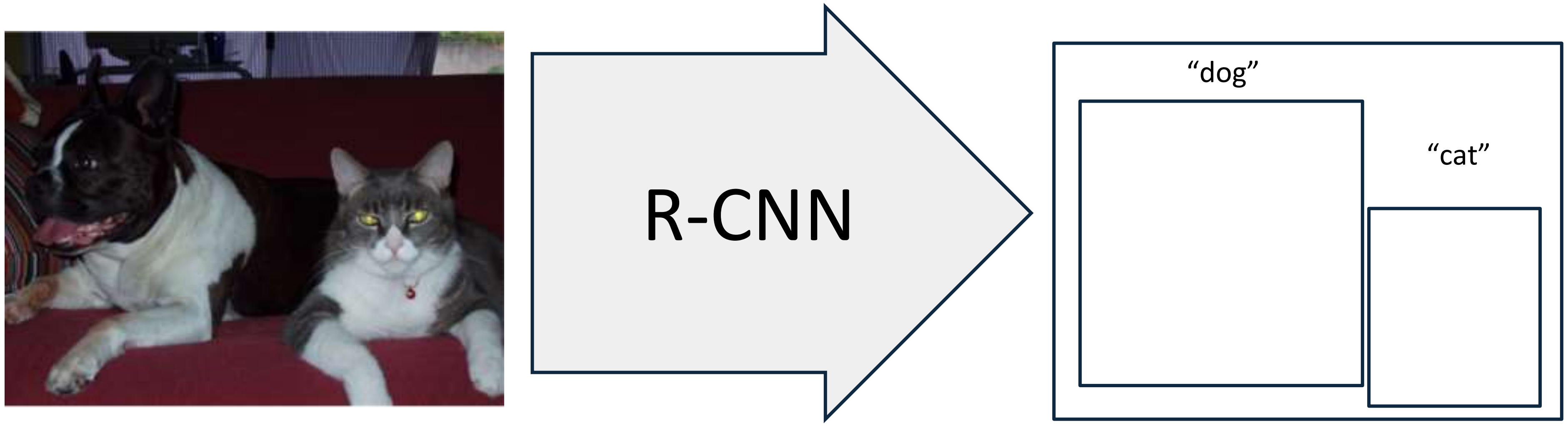
Detection, localization, and explanation

Object detection

Goal: correctly identify where the main objects are.

This task is called *object detection.*

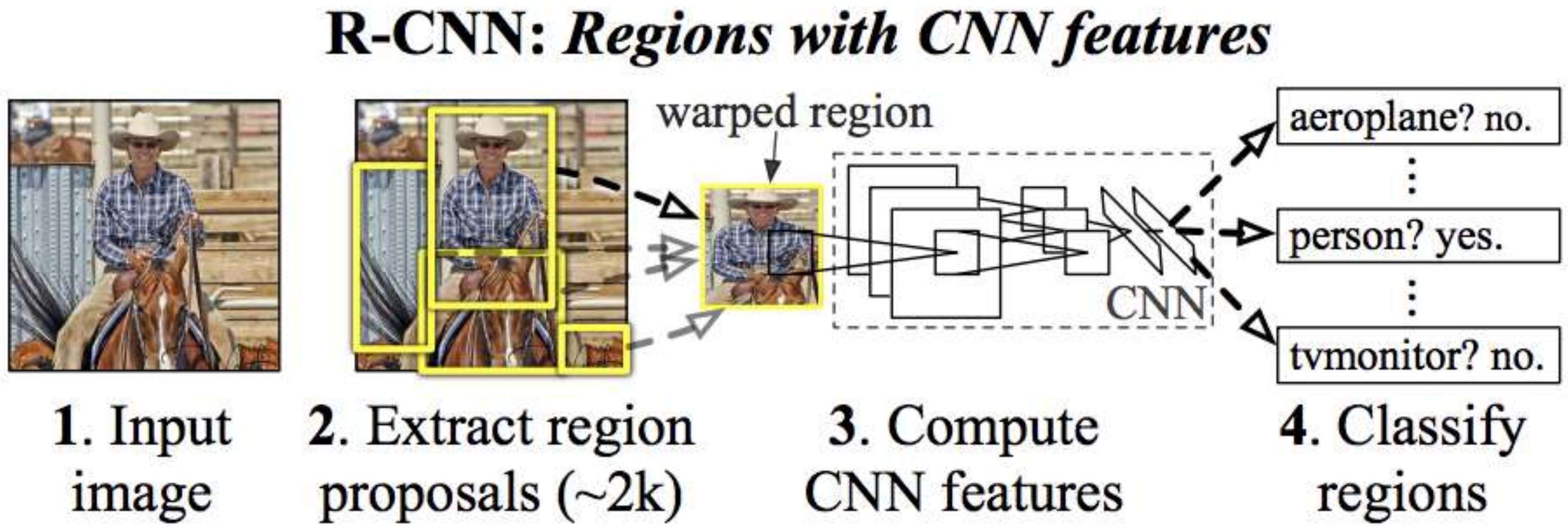
Output: bounding boxes with object classification scores.



Region-Based ConvNets (R-CNN) [2014]

Idea: each box in an image is its own image.

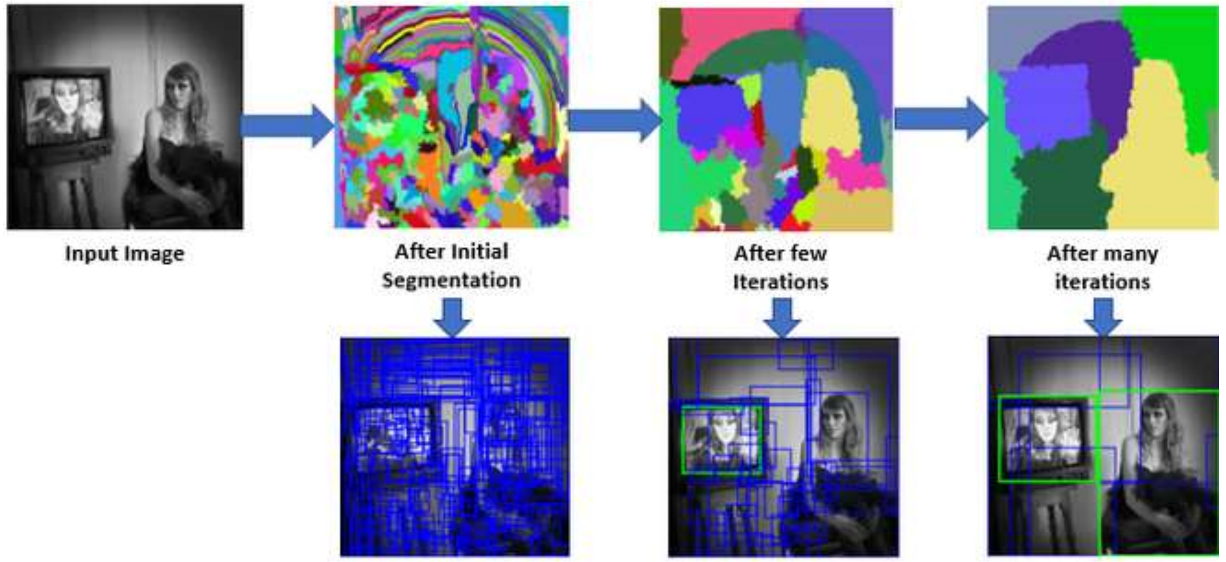
How: (1) find interesting boxes, (2) rescale each box and feed through a CNN.



Extracting region proposals

An idea from before the deep learning era. R-CNN uses an algorithm designed by the UvA: selective search.

Selective search: start from superpixels, group one-by-one. Each new grouping is a new region proposal.



Extra step: improving the region locations

Now, having found the object in the box, can we tighten the box to fit the true dimensions of the object?

We can, and this is the final step of R-CNN.

R-CNN runs a simple linear regression on the region proposal to generate tighter bounding box coordinates to get the final result.

- Inputs: sub-regions of the image corresponding to objects.
- Outputs: New bounding box coordinates for the object in the sub-region.

R-CNN summarized

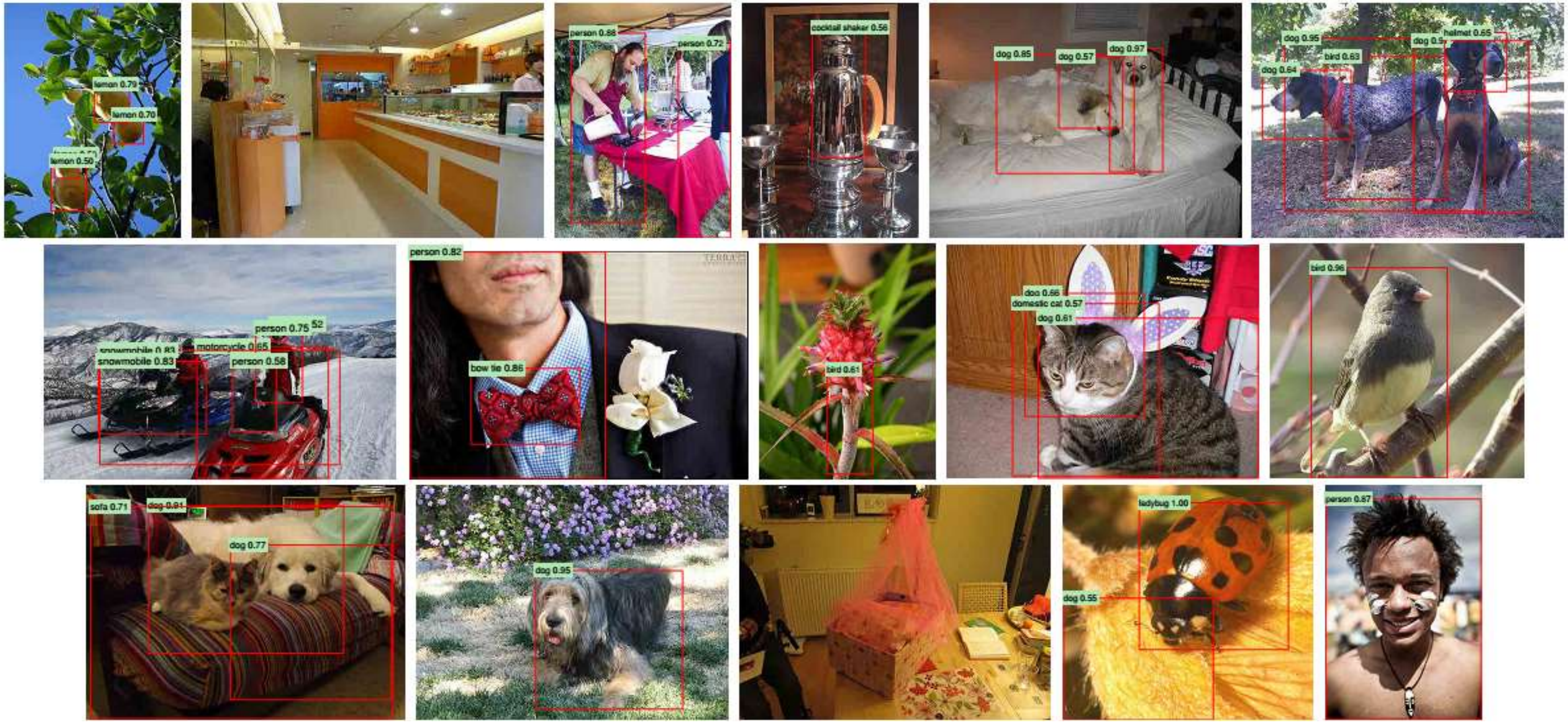
R-CNN is just the following steps:

1. Generate a set of proposals for bounding boxes.

2. Run the images in the bounding boxes through a pre-trained AlexNet and finally an SVM to see what object the image in the box is.

3. Run the box through a linear regression model to output tighter coordinates for the box once the object has been classified.

R-CNN results

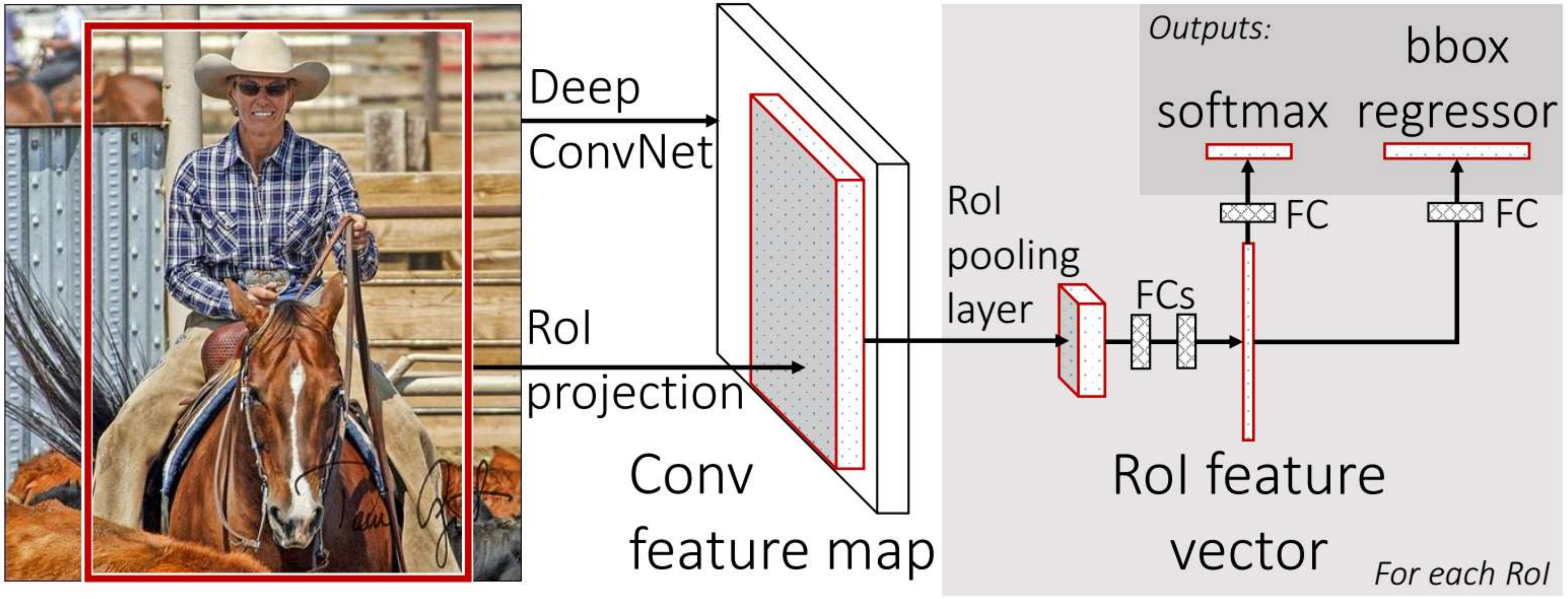


What are the two major downsides to R-CNN?

1. Each region proposal requires a new pass through the network.

2. Region proposals need to be determined a priori.

Fast R-CNN [2015]



Key to Fast R-CNN: RIOPooling

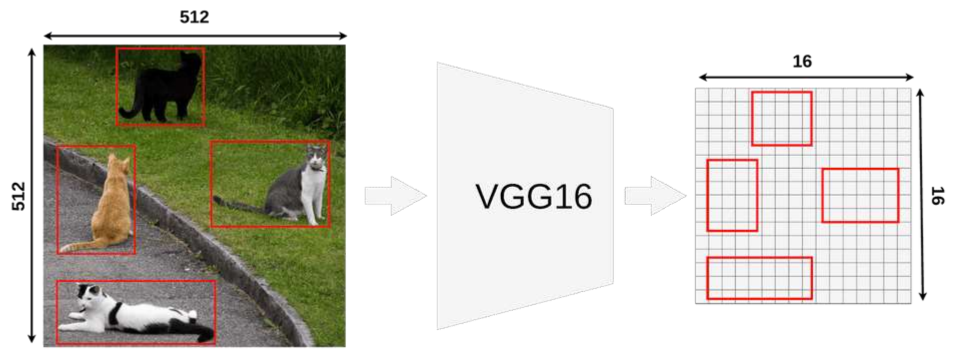
For the forward pass of the CNN, a lot of proposed regions for the image invariably overlapped …

… causing us to run the same CNN computation again and again (~2000 times!).

Why not run the CNN just once per image and then find a way to share that computation across the ~2000 proposals?

Region of Interest pooling

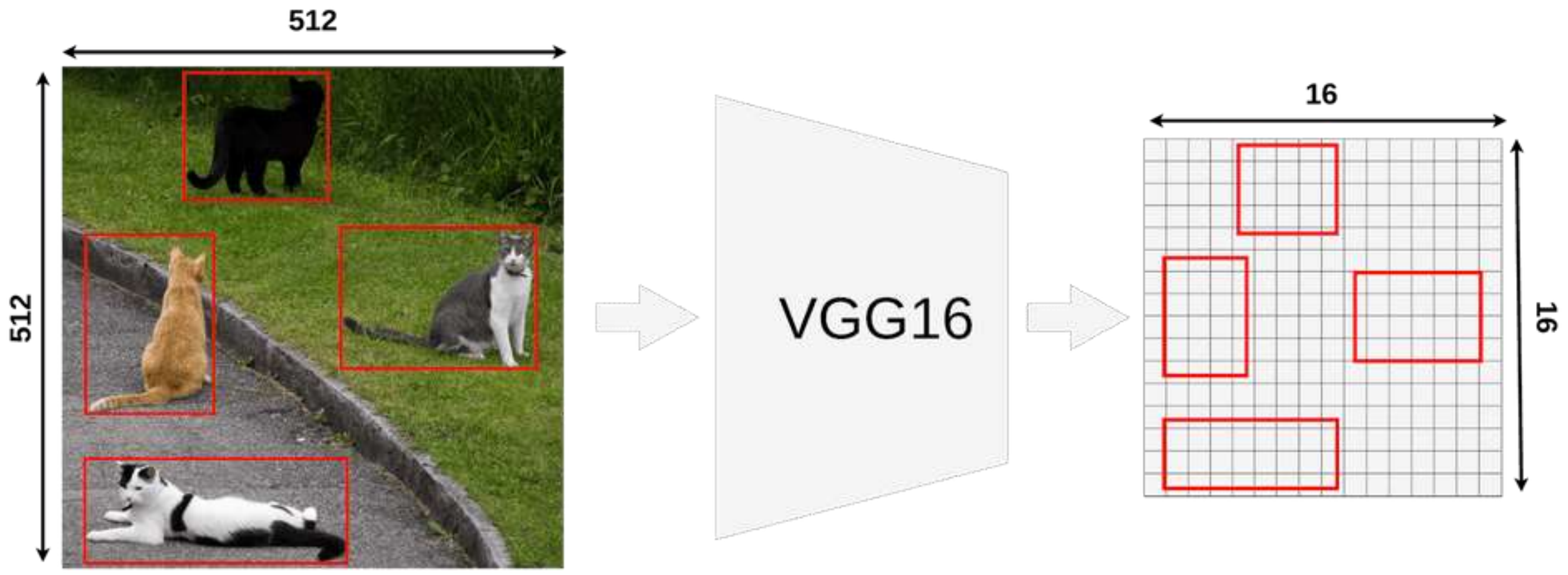
RoIPool shares the forward pass of a CNN for an image across its subregions.



In the image above, notice how the CNN features for each region are obtained by selecting a corresponding region from the CNN's feature map.

Then, the features in each region are pooled (usually using max pooling). So, all it takes us is one pass of the original image as opposed to ~2000!

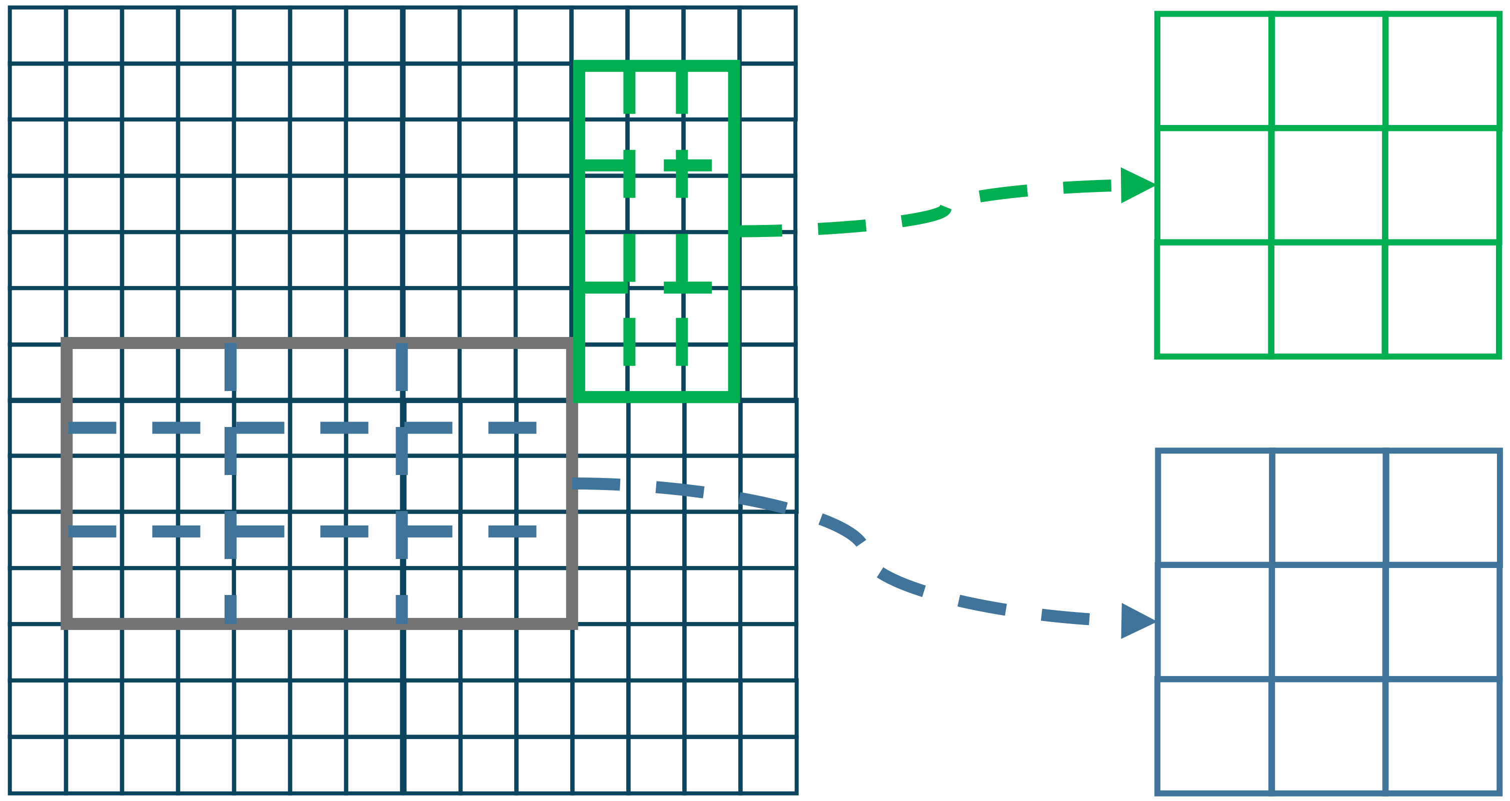
Region of Interest pooling



Region of Interest pooling

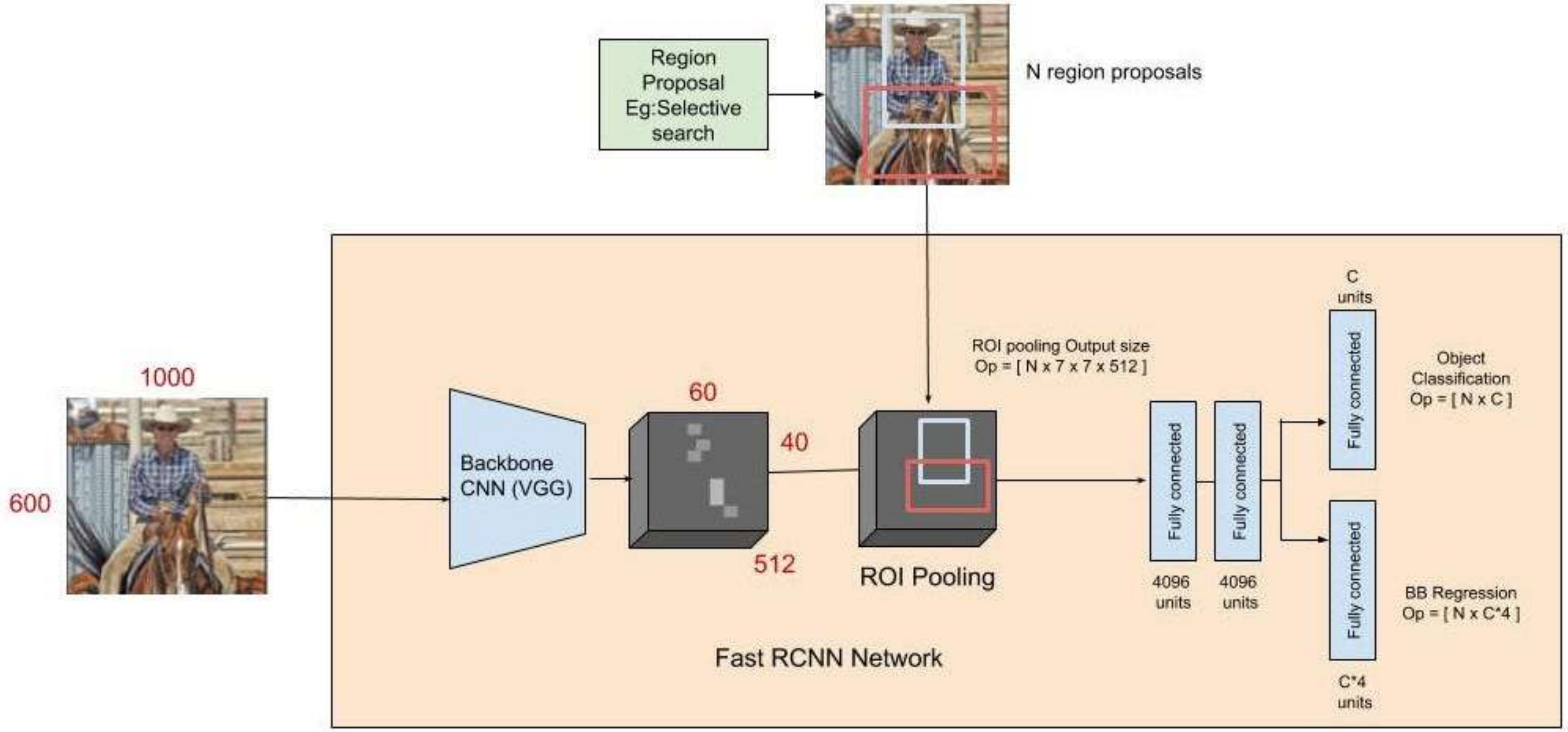
Divide feature map in TxT cells.

The cell size will change depending on the size of the candidate location.



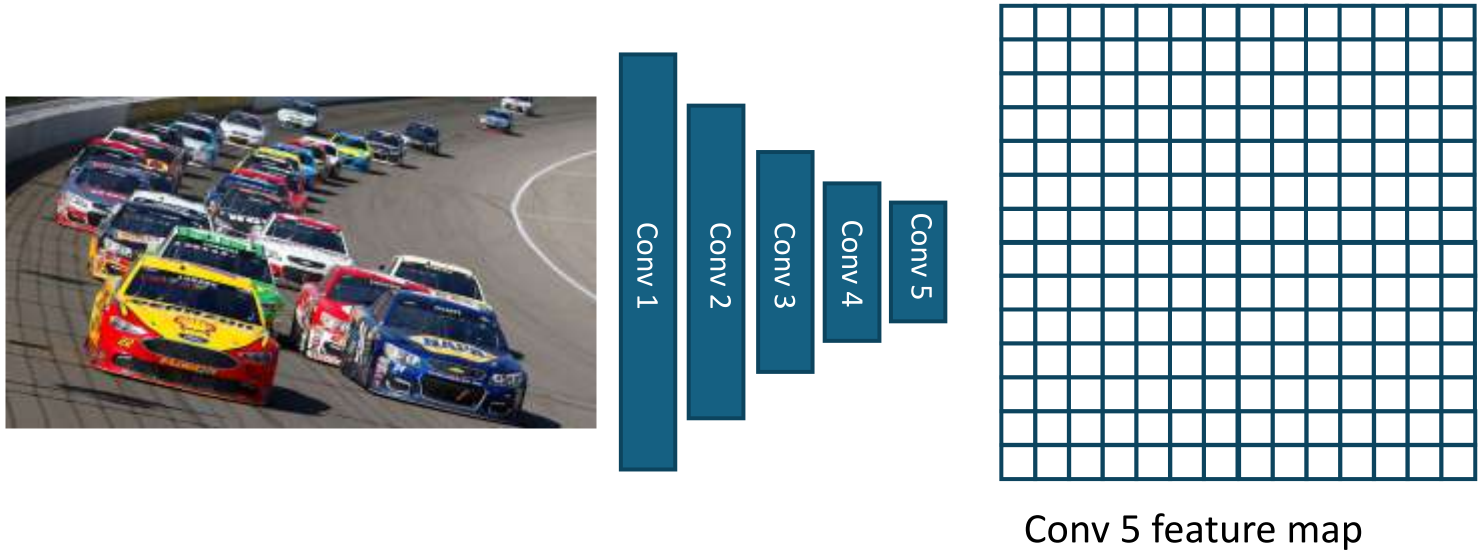
Always 3x3 no matter the size of candidate location

Second key to Fast R-CNN: joint training



Fast R-CNN: step-by-step

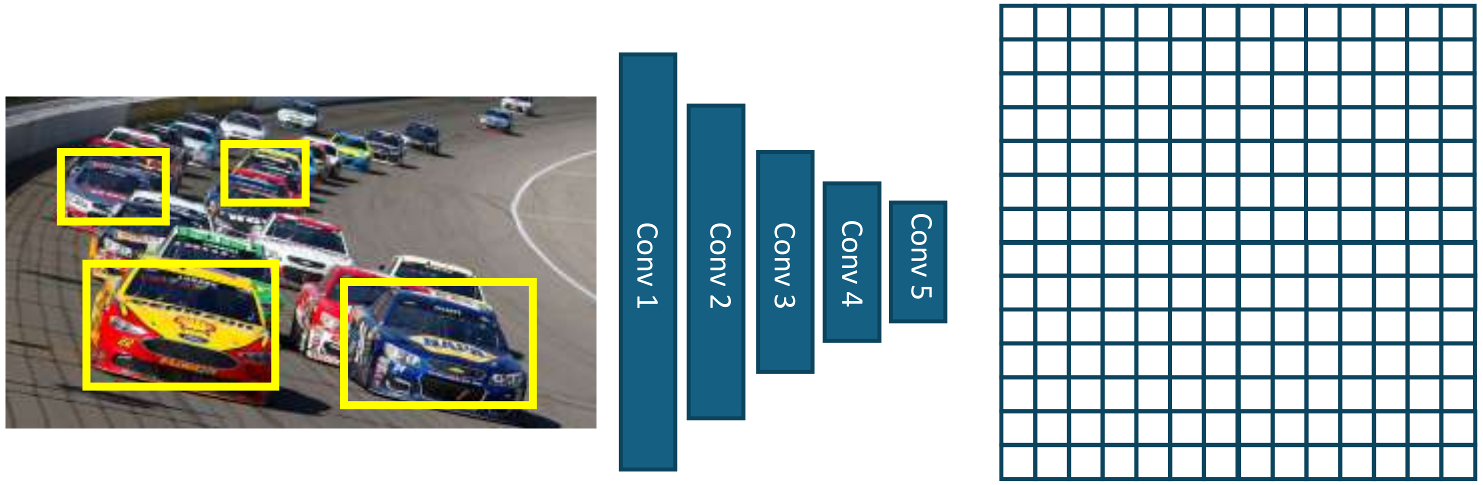
Process the whole image up to conv5.



Fast R-CNN: step-by-step

Process the whole image up to conv5.

Compute possible locations for objects.

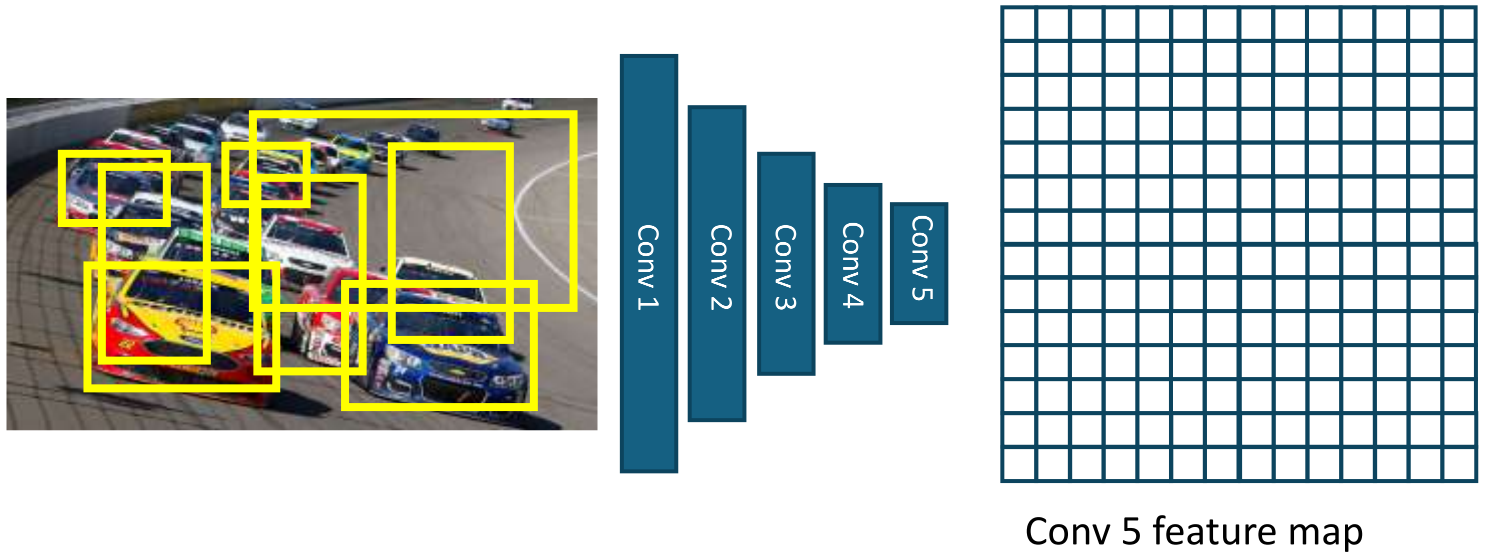


Conv 5 feature map

Fast R-CNN: step-by-step

Process the whole image up to conv5.

Compute possible locations for objects (some correct, most wrong).

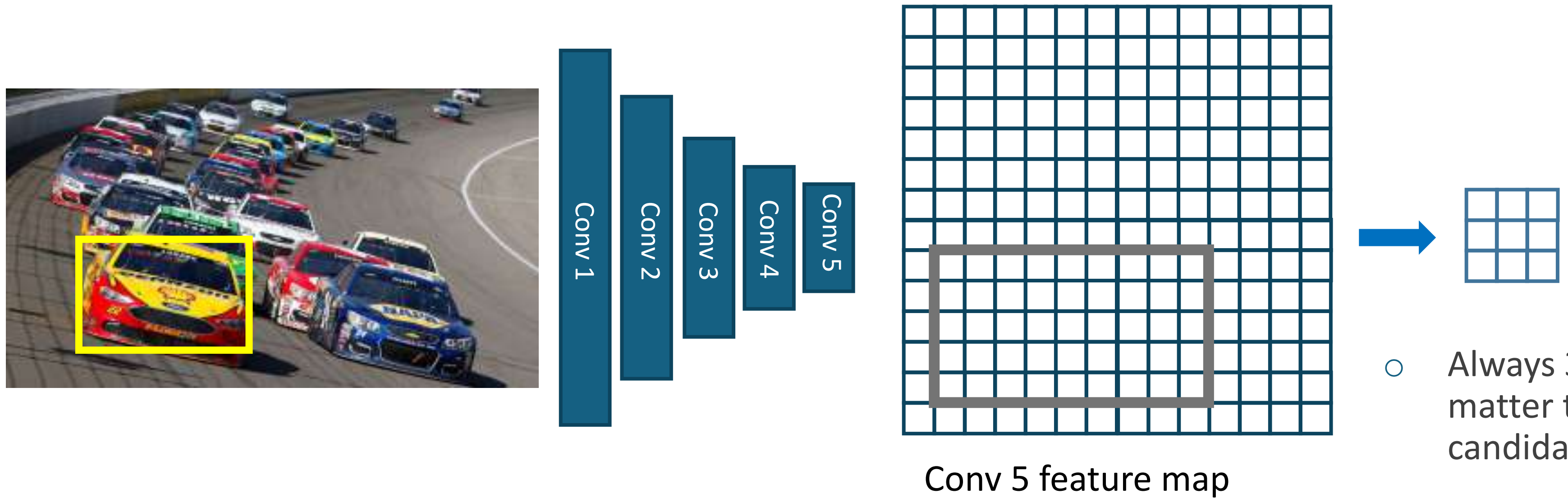


Fast R-CNN: step-by-step

Process the whole image up to conv5.

Compute possible locations for objects.

Given single location → ROI pooling module extracts fixed length feature.



- Always 3x3 no matter the size of candidate location

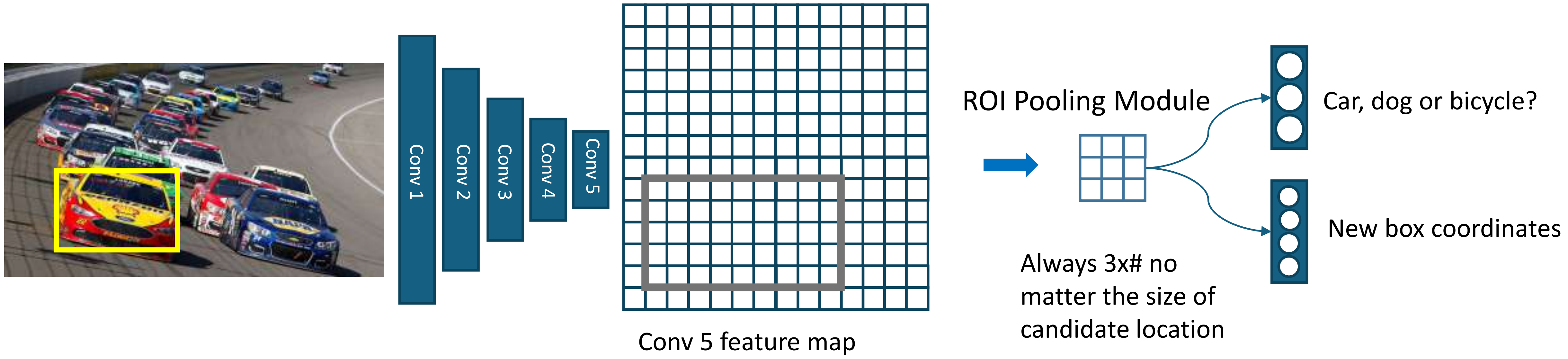
Fast R-CNN: step-by-step

Process the whole image up to conv5.

Compute possible locations for objects.

Given single location → ROI pooling module extracts fixed length feature.

Connect to two final layers, 1 for classification, 1 for box refinement.



Pros and cons of Fast R-CNN

Reuse convolutions for different candidate boxes

Compute feature maps only once

Region-of-Interest pooling

Define stride relatively → box width divided by predefined number of "poolings" T

Fixed length vector

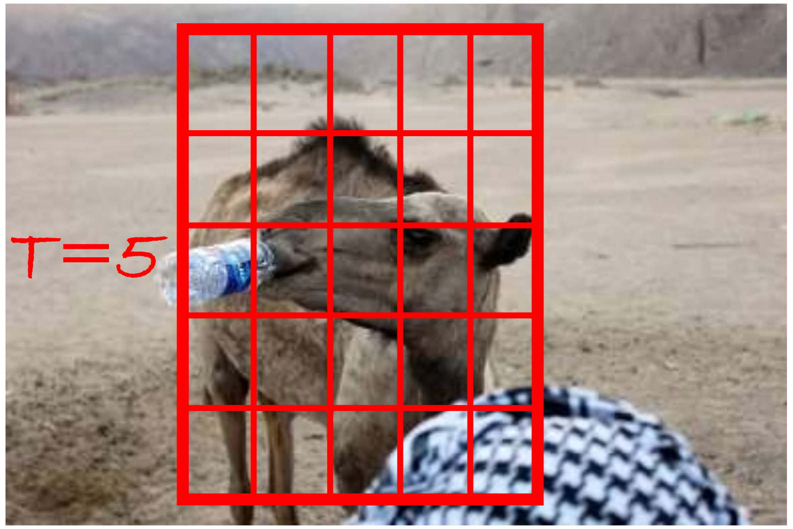
"End-to-end" training!

(Very) Accurate object detection

(Very) Faster

- Less than a second per image

External box proposals needed



Faster R-CNN [2017]

Still a bottleneck: the region proposer.

First step: generating a bunch of potential bounding boxes/regions.

In Fast R-CNN, these proposals were created using Selective Search, a fairly slow process that was found to be the bottleneck of the overall process…

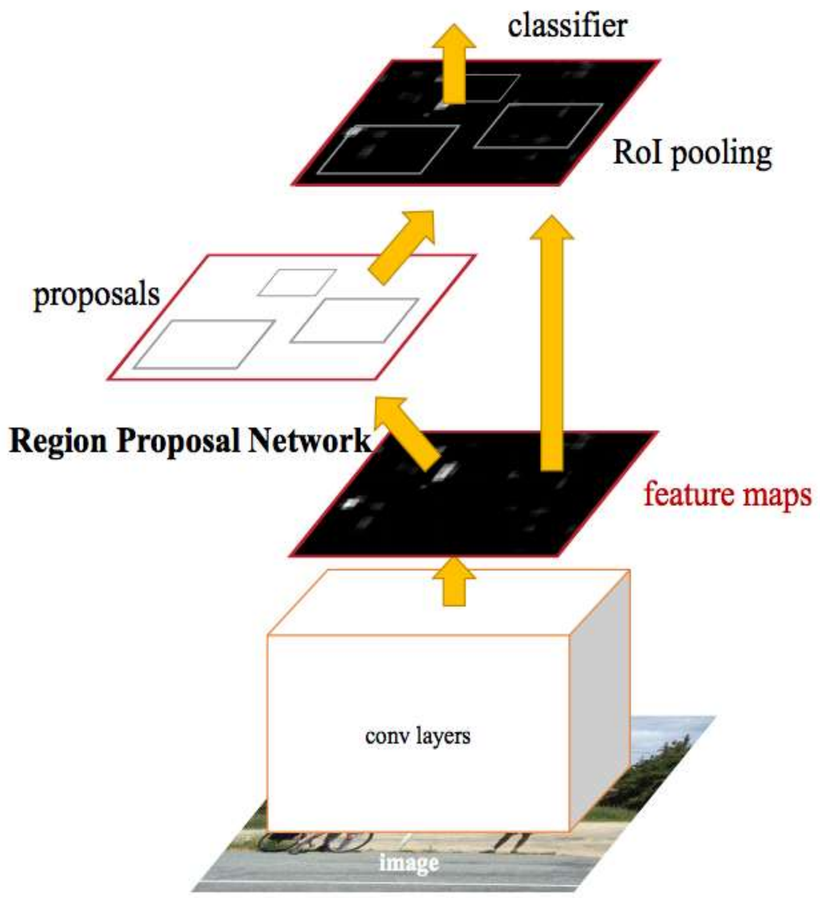
Faster R-CNN

Faster R-CNN adds a Fully Convolutional Network on top of the features of the CNN creating what's known as the Region Proposal Network.

The **Region Proposal Network** works by passing a sliding window over the CNN feature map and at each window, outputting *k* potential bounding boxes and scores for how good each of those boxes is expected to be.

In such a way, we create *k* such common aspect ratios we call anchor boxes. For each such anchor box, we output one bounding box and score per position in the image.

We then pass each such bounding box that is likely to be an object into Fast R-CNN to generate a classification and tightened bounding boxes.

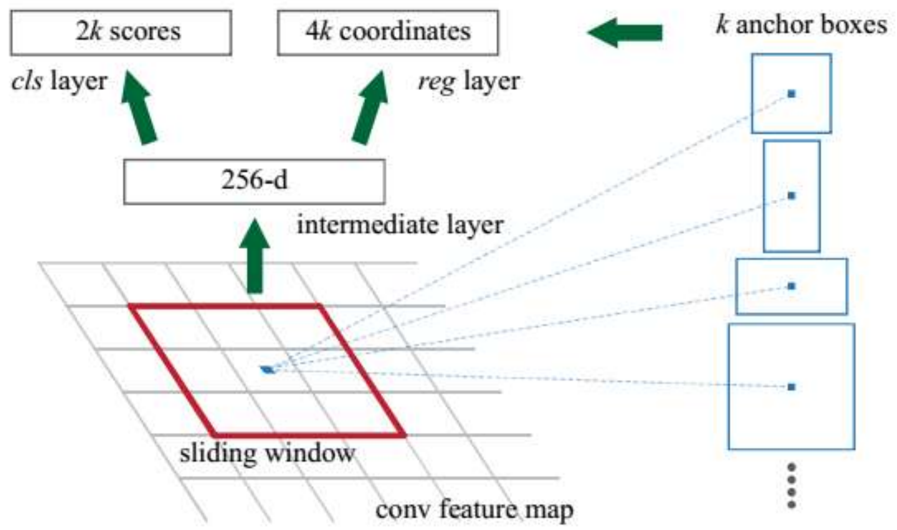


Faster R-CNN

Fast R-CNN → external candidate locations.

Faster R-CNN → deep network proposes candidate locations.

Slide the feature map → k anchor boxes per slide.



Region Proposal Network

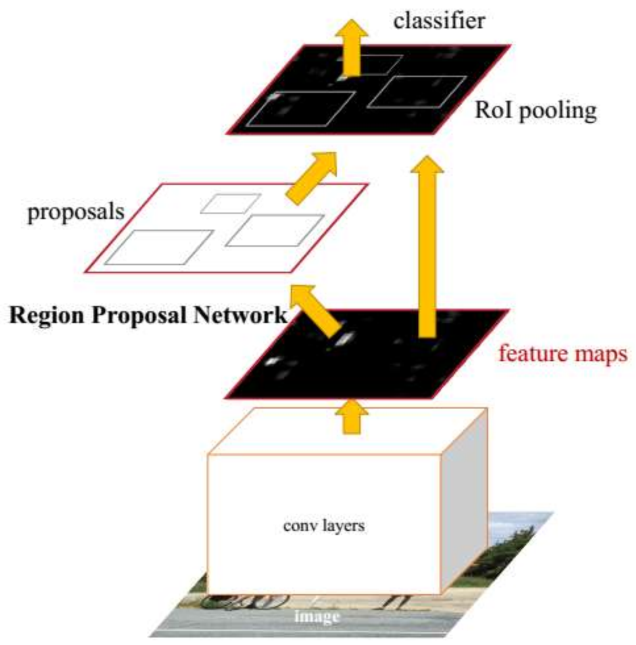


Figure 2: Faster R-CNN is a single, unified network for object detection. The RPN module serves as the 'attention' of this unified network.

Focal loss

Cross-entropy is a problem for object detection.

Many boxes evaluated, a lot of small losses lead to a huge bias.

Solution: add an exponent to the cross-entropy term!

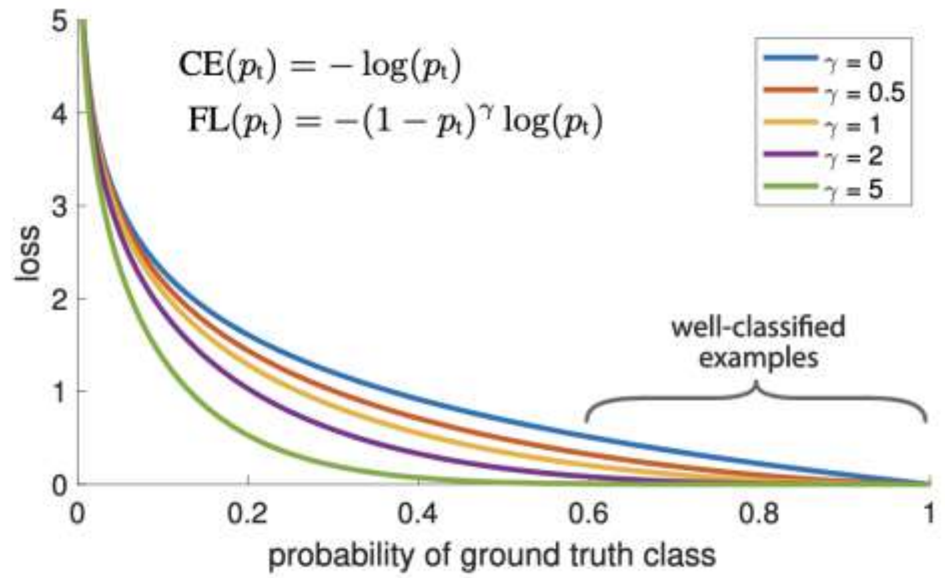


Figure 1. We propose a novel loss we term the *Focal Loss* that adds a factor $(1 - p_t)^\gamma$ to the standard cross entropy criterion. Setting $\gamma > 0$ reduces the relative loss for well-classified examples ($p_t > .5$), putting more focus on hard, misclassified examples. As our experiments will demonstrate, the proposed focal loss enables training highly accurate dense object detectors in the presence of vast numbers of easy background examples.

Mask R-CNN

Extending Faster R-CNN for Pixel Level Segmentation.

So far, we’ve seen how we’ve been able to use CNN features in many interesting ways to effectively locate different objects in an image with bounding boxes.

Can we extend such techniques to go one step further and locate exact pixels of each object instead of just bounding boxes?

This problem is known as *image segmentation.*

Mask R-CNN

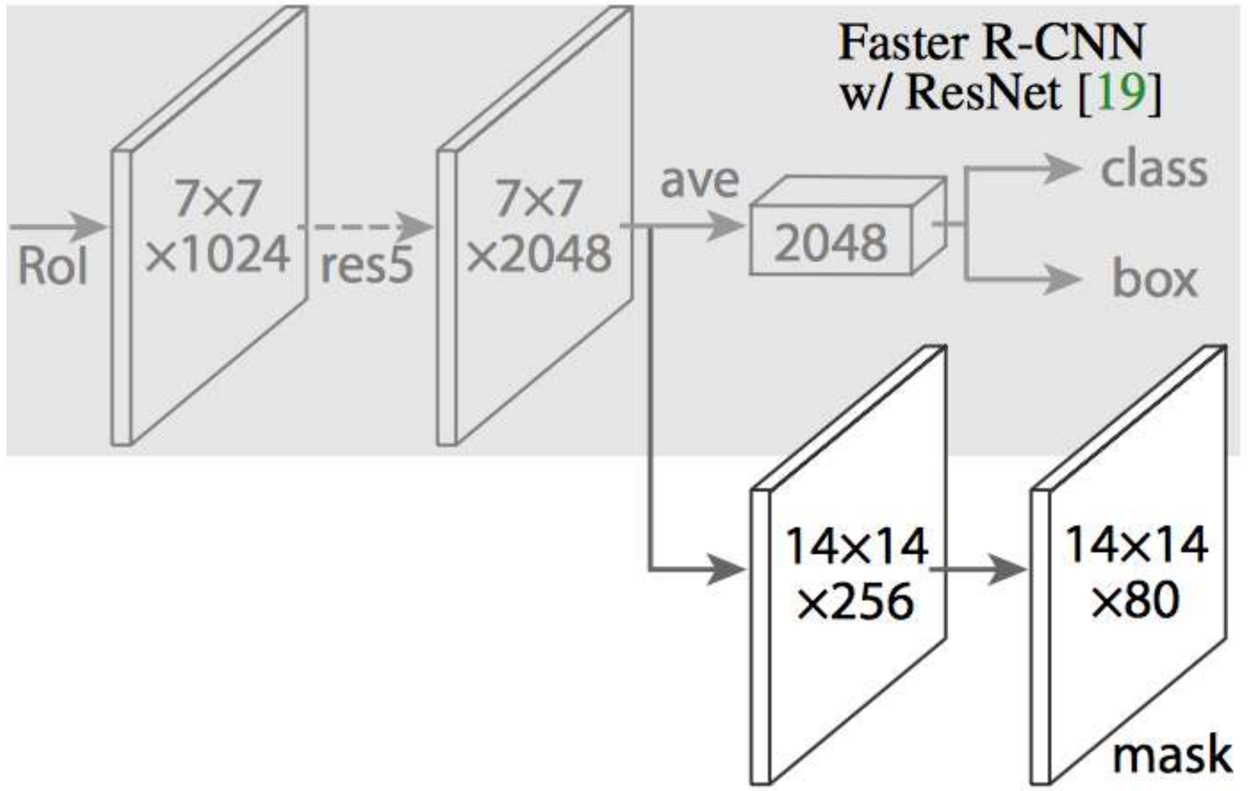
Given that Faster R-CNN works so well for object detection, could we extend it to also carry out pixel level segmentation?

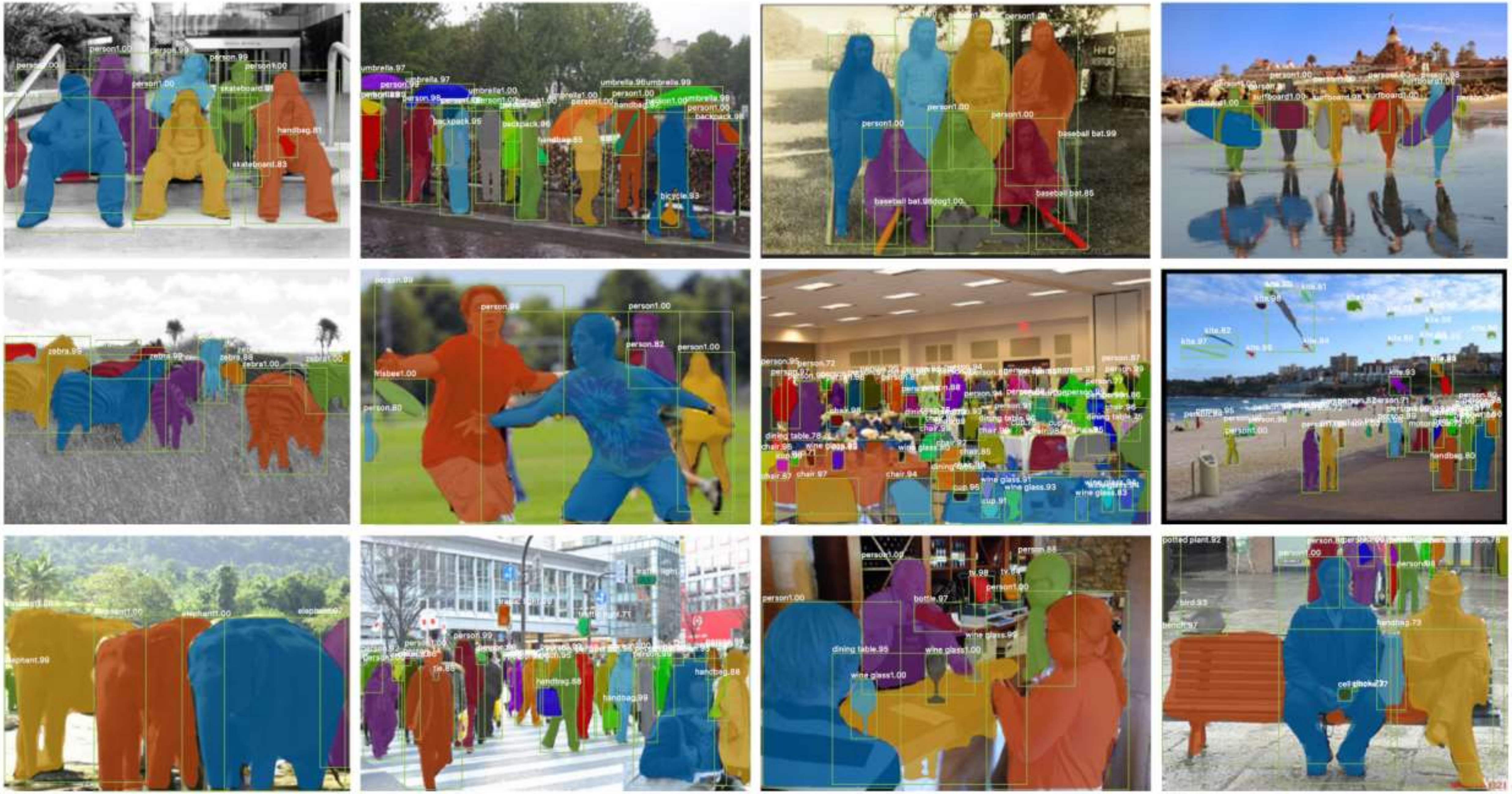
Mask R-CNN does this by adding a branch to Faster R-CNN that outputs a binary mask that says whether or not a given pixel is part of an object.

Here are its inputs and outputs:

- Inputs: CNN Feature Map.
- Outputs: Matrix with 1s on all locations where the pixel belongs to the object and 0s elsewhere (this is known as a binary mask).

Mask R-CNN

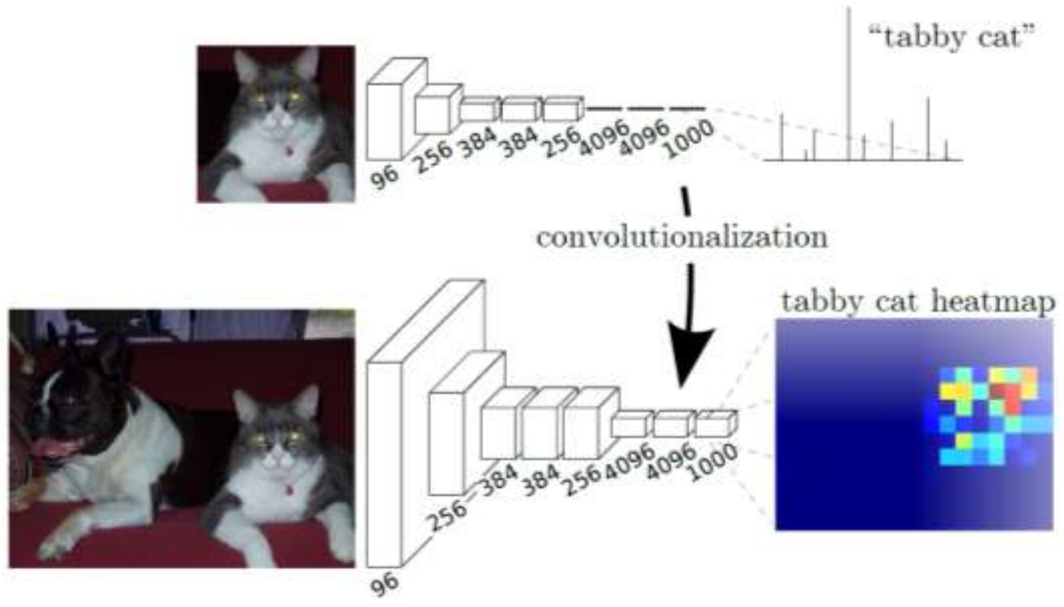

Faster R-CNN
w/ ResNet [19]
RoI
7×7
×1024
res5
7×7
×2048
ave
2048
class
box
14×14
×256
14×14
×80
mask



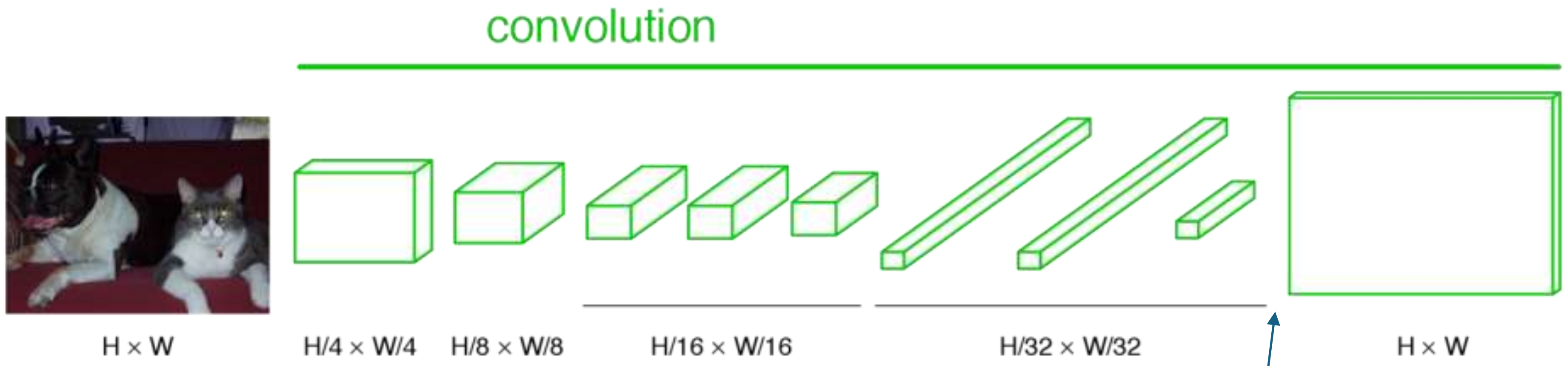
Towards pixel-level classification

Mask R-CNN classifies pixels within boxes.

Instead of first detecting objects and then segmenting them, can we also just classify each pixel directly to get a full segmentation mask?

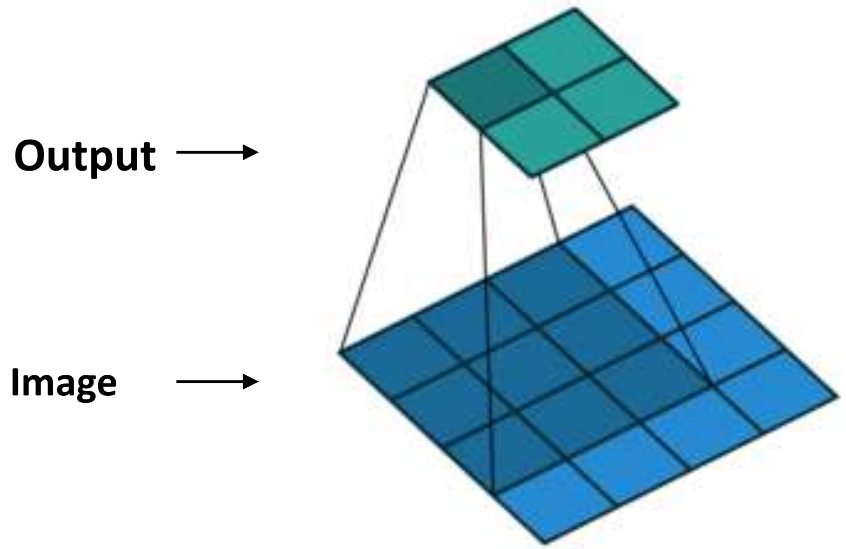


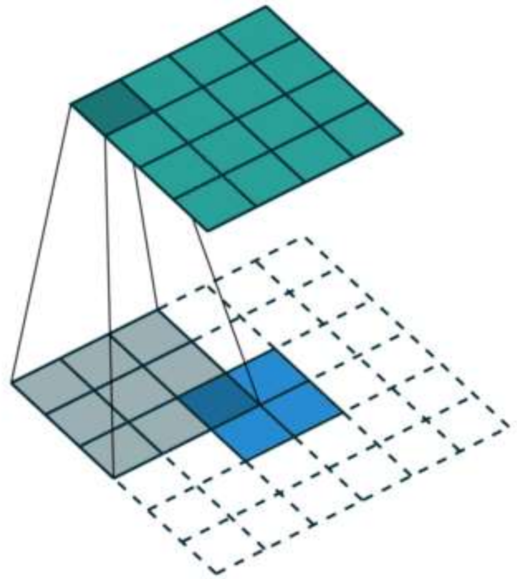
From a bottleneck back to a full image

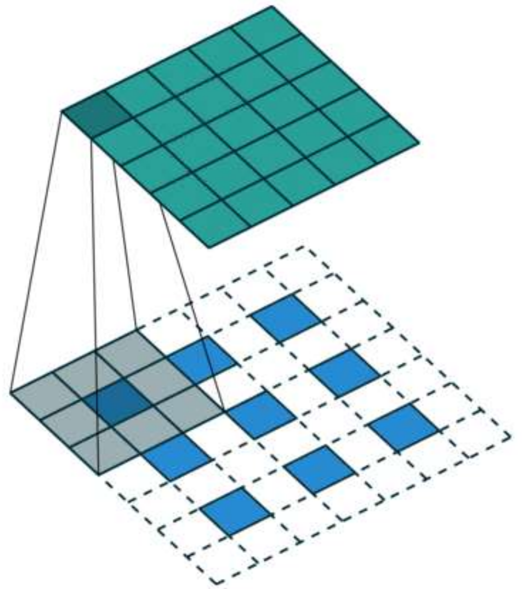


Some upsampling algorithm to return us to H x W

Deconvolutions







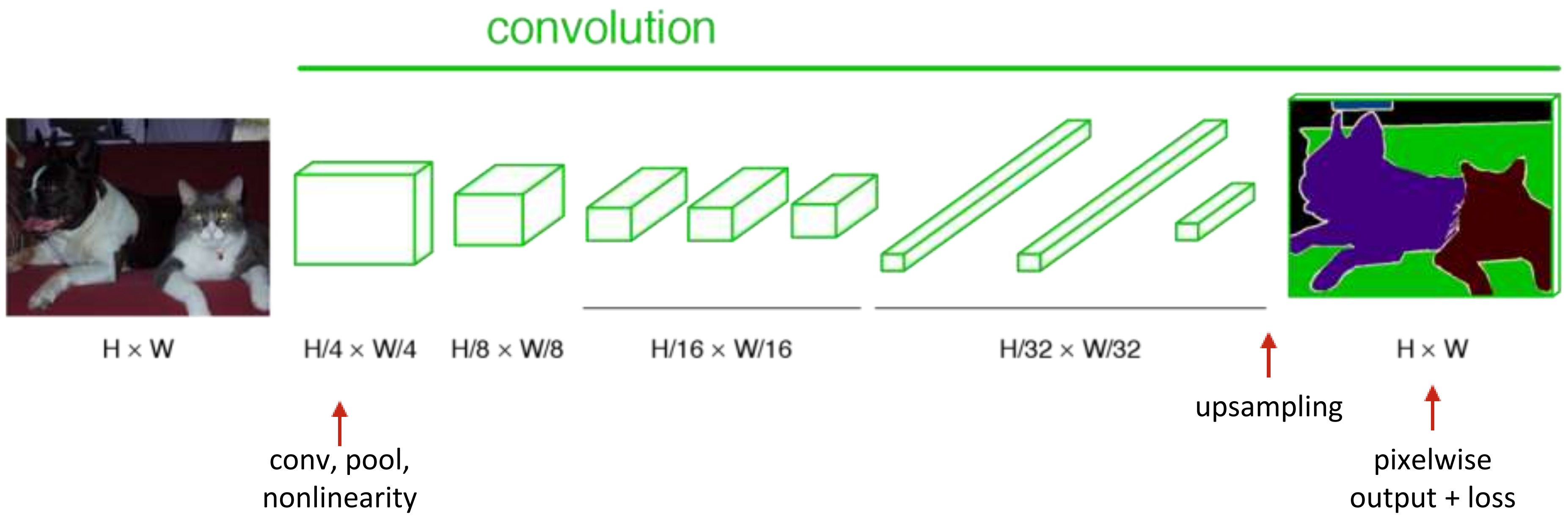
Convolution
No padding, no strides

Upconvolution
Padding, no strides

Upconvolution
Padding, strides

More visualizations: https://github.com/vdumoulin/conv_arithmetic

End-to-end pixel-level optimization



This lecture

Convolutions.

Convolutional networks.

Convolutional networks for detection and segmentation.

Learning and reflection

Chapter 9, Deep Learning Book

Chapter 10, Understanding Deep Learning Book

Learning and reflection

Understanding Deep Learning: Chapter 10

Understanding Deep Learning: Chapter 11

Next lecture

Lecture	Title
1	Intro and history of deep learning
3	Deep learning optimization I
5	Convolutional deep learning
7	Graph deep learning
9	Multi-modal deep learning
11	What doesn't work in deep learning
13	Q&A

Lecture	Title
2	AutoDiff
4	Deep learning optimization II
6	Attention-based deep learning
8	From supervised to unsupervised deep learning
10	Generative deep learning
12	Non-Euclidean deep learning
14	Deep learning for videos

Thank you