Procesamiento de Señal y Transformadas

Convolutional Neural Networks

Prof. Rubén Vera Rodríguez
ruben.vera@uam.es
BiDA Lab, EPS
http://atvs.ii.uam.es/atvs/

Based on material by Fei-Fei Li & Justin Johnson & Serena Yeung from Stanford University

Escuela Politécnica Superior UAM Universidad Autónoma de Madrid BiDA Lab

Bit of history…

Frank Rosenblatt, ~1957: Perceptron

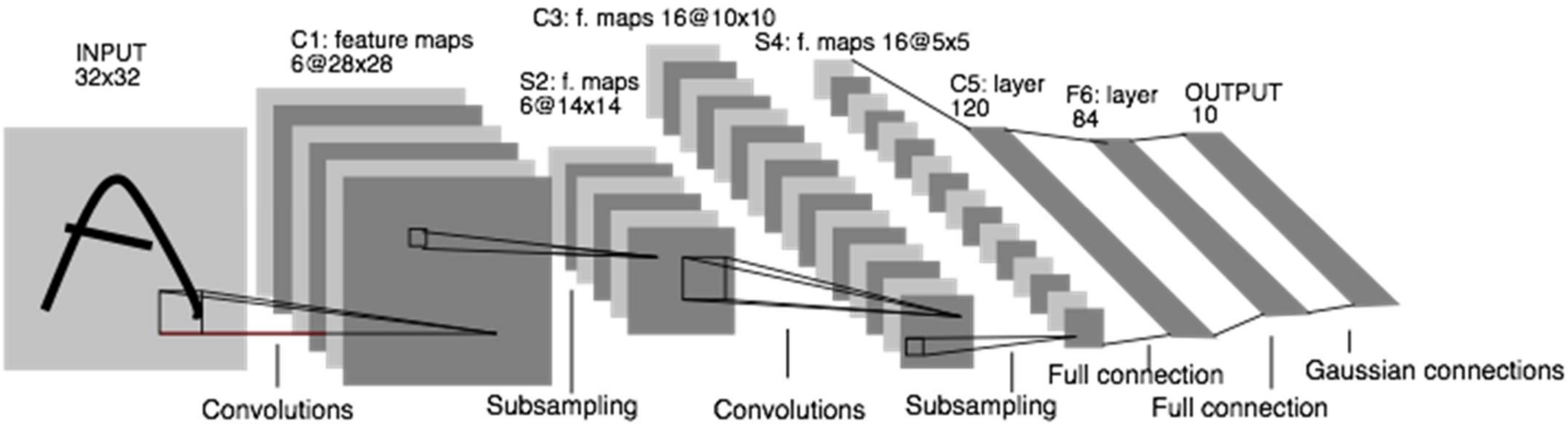
Widrow and Hoff, ~1960: Adaline/Madaline: Multilayer perceptron networks

Rumelhart et al., 1986: First time back-propagation became popular

Bit of history…

Convolutional Networks: 1998

LeNet: a layered model composed of convolution and subsampling operations followed by a holistic representation and ultimately a classifier for handwritten digits. *[LeCun, Bottou, Bengio, Haffner 1998]*



Bit of history…

Hinton and Salakhutdinov 2006, Reinvigorated research in Deep Learning

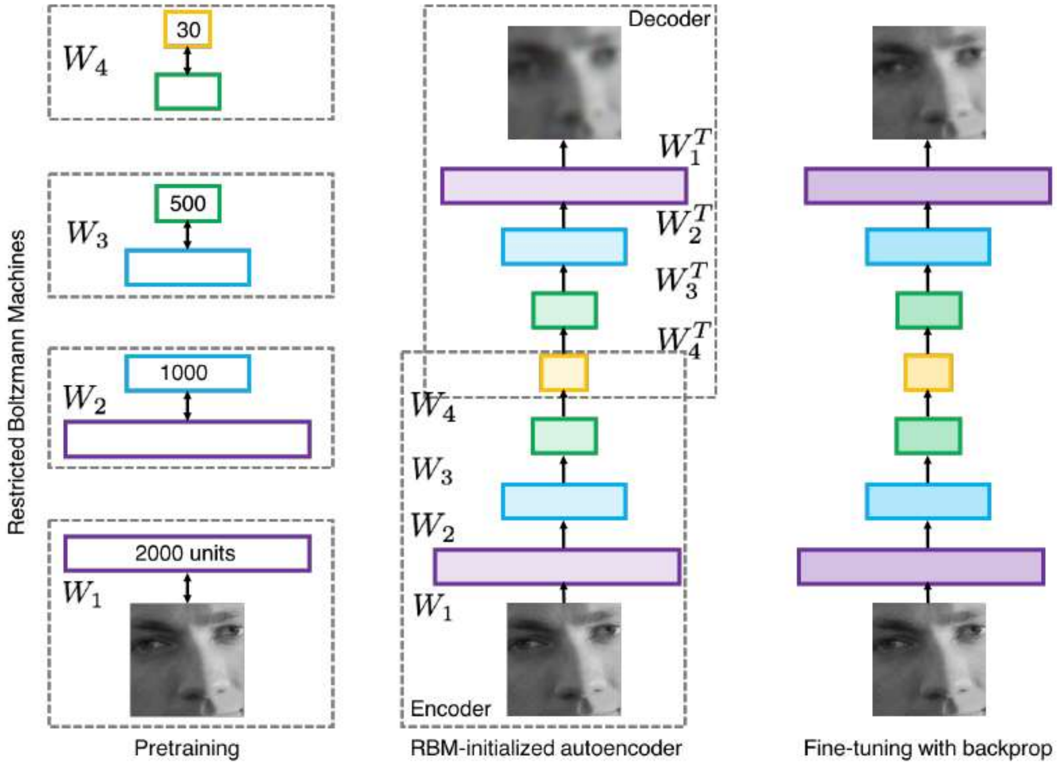


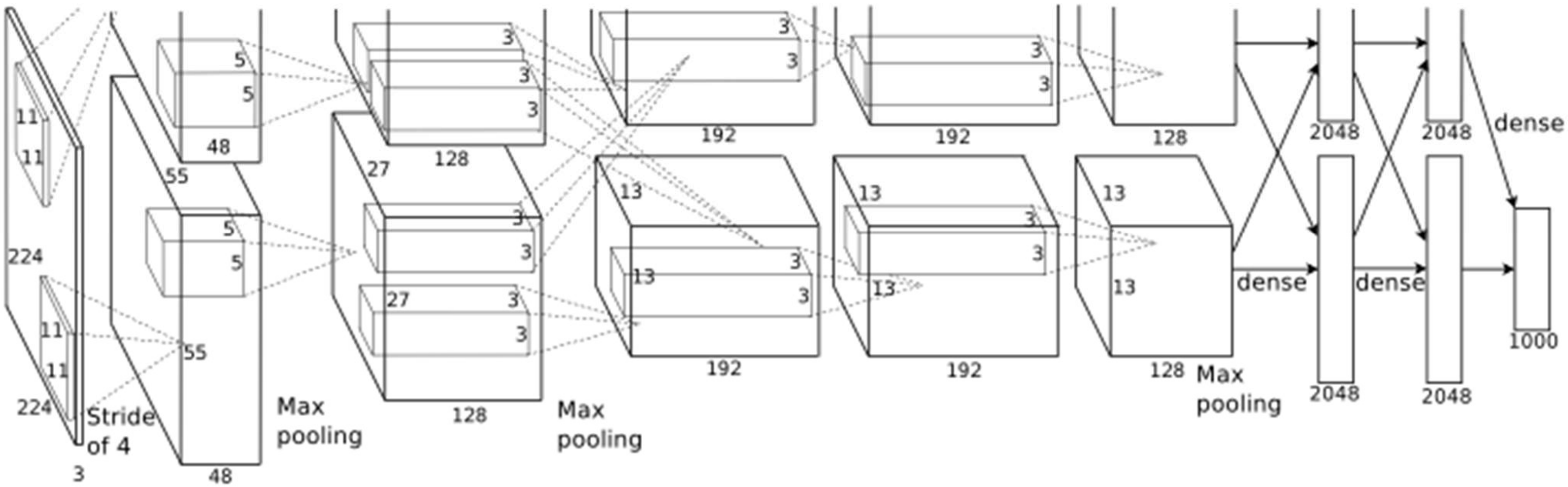
Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017

Bit of history…

Convolutional Nets: 2012

AlexNet: a layered model composed of convolution, subsampling, and further operations followed by a holistic representation and a final classification on ILSVRC12.

+ data + gpu + non-saturating nonlinearity + regularization

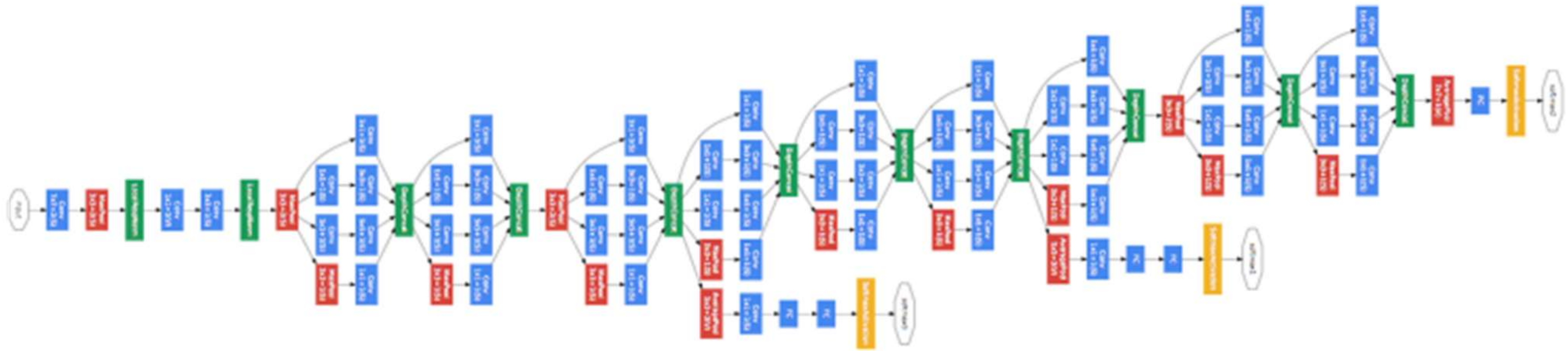


ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

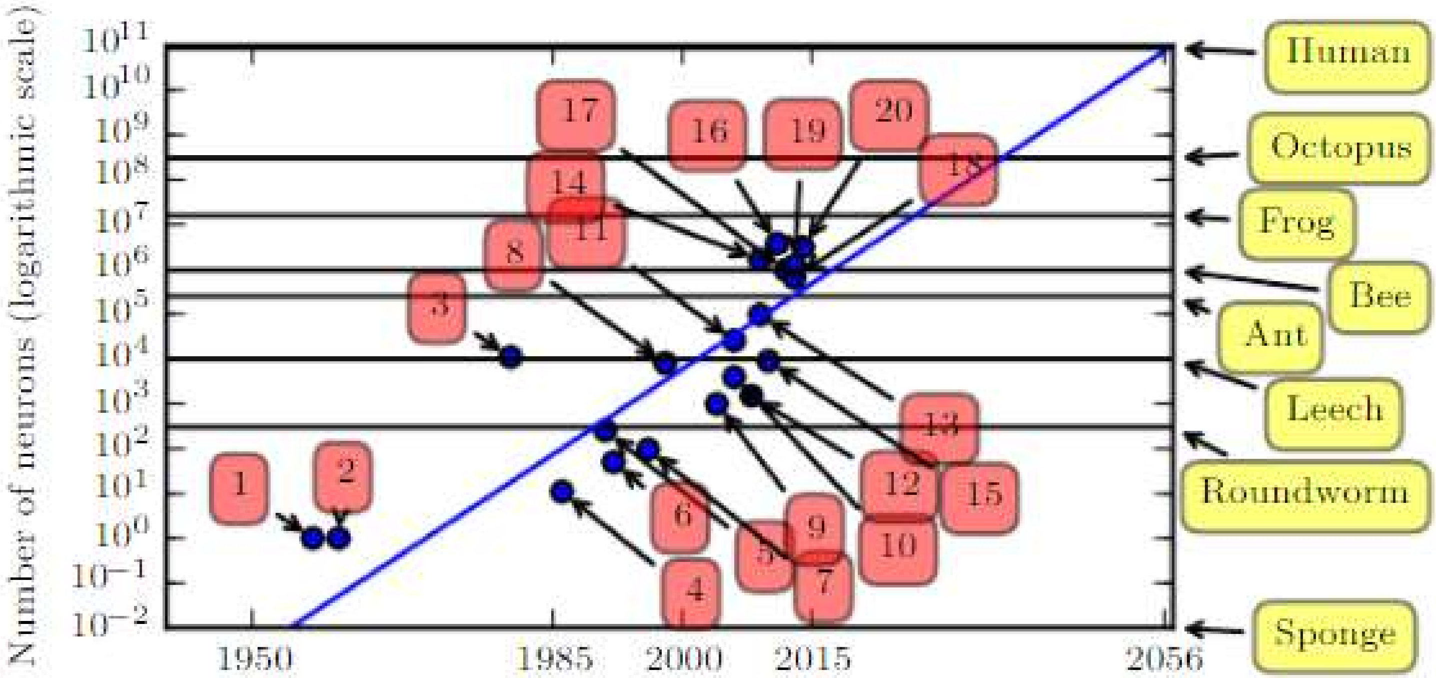
Convolutional Nets: 2014

ILSVRC14 Winners: ~6.6% Top-5 error
- **GoogLeNet**: composition of multi-scale dimension-reduced modules (pictured)
- VGG: 16 layers of 3x3 convolution interleaved with max pooling + 3 fully-connected layers

+ data + gpu + non-saturating nonlinearity + regularization

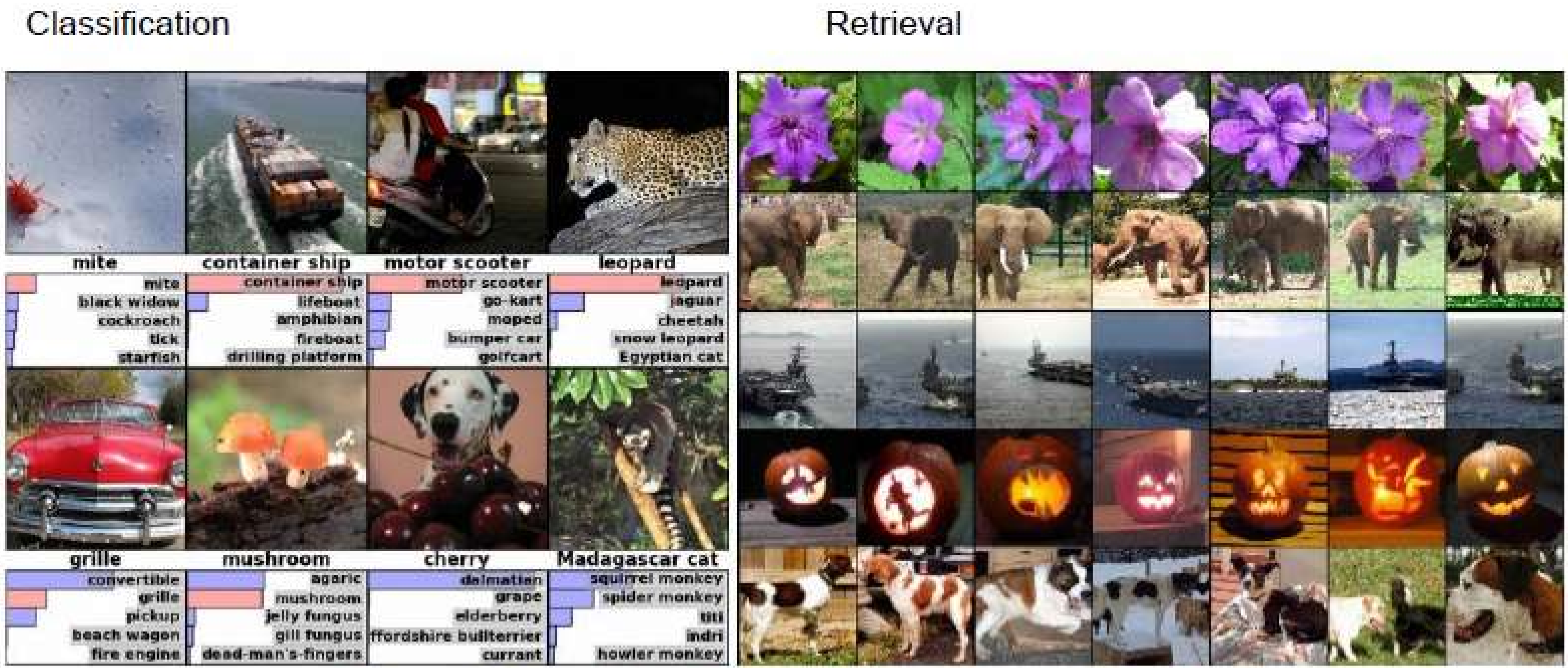


Evolution of Neural Networks



1. Perceptron (Rosenblatt, 1958, 1962)
2. Adaptive linear element (Widrow and Hoff, 1960)
3. Neocognitron (Fukushima, 1980)
4. Early back-propagation network (Rumelhart *et al.*, 1986b)
5. Recurrent neural network for speech recognition (Robinson and Fallside, 1991)
6. Multilayer perceptron for speech recognition (Bengio *et al.*, 1991)
7. Mean field sigmoid belief network (Saul *et al.*, 1996)
8. LeNet-5 (LeCun *et al.*, 1998b)
9. Echo state network (Jaeger and Haas, 2004)
10. Deep belief network (Hinton *et al.*, 2006)
11. GPU-accelerated convolutional network (Chellapilla *et al.*, 2006)
12. Deep Boltzmann machine (Salakhutdinov and Hinton, 2009a)
13. GPU-accelerated deep belief network (Raina *et al.*, 2009)
14. Unsupervised convolutional network (Jarrett *et al.*, 2009)
15. GPU-accelerated multilayer perceptron (Ciresan *et al.*, 2010)
16. OMP-1 network (Coates and Ng, 2011)
17. Distributed autoencoder (Le *et al.*, 2012)
18. Multi-GPU convolutional network (Krizhevsky *et al.*, 2012)
19. COTS HPC unsupervised convolutional network (Coates *et al.*, 2013)
20. GoogLeNet (Szegedy *et al.*, 2014a)

ConvNets Applications

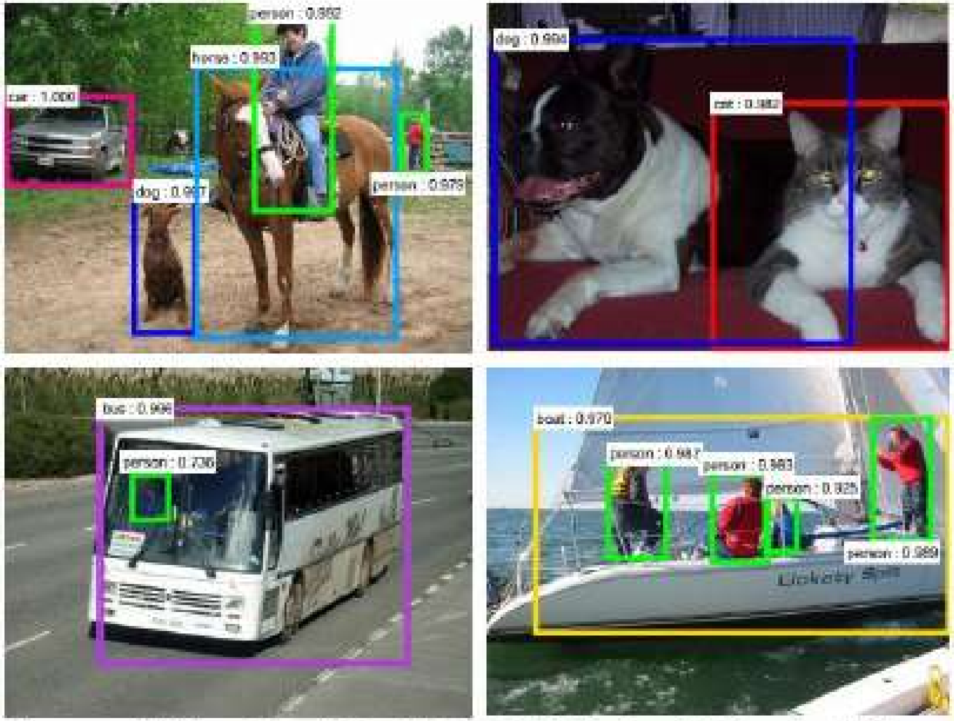


Figures copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

ImageNet Classification with Deep Convolutional Neural Networks *[Krizhevsky, Sutskever, Hinton, 2012]*

ConvNets Applications

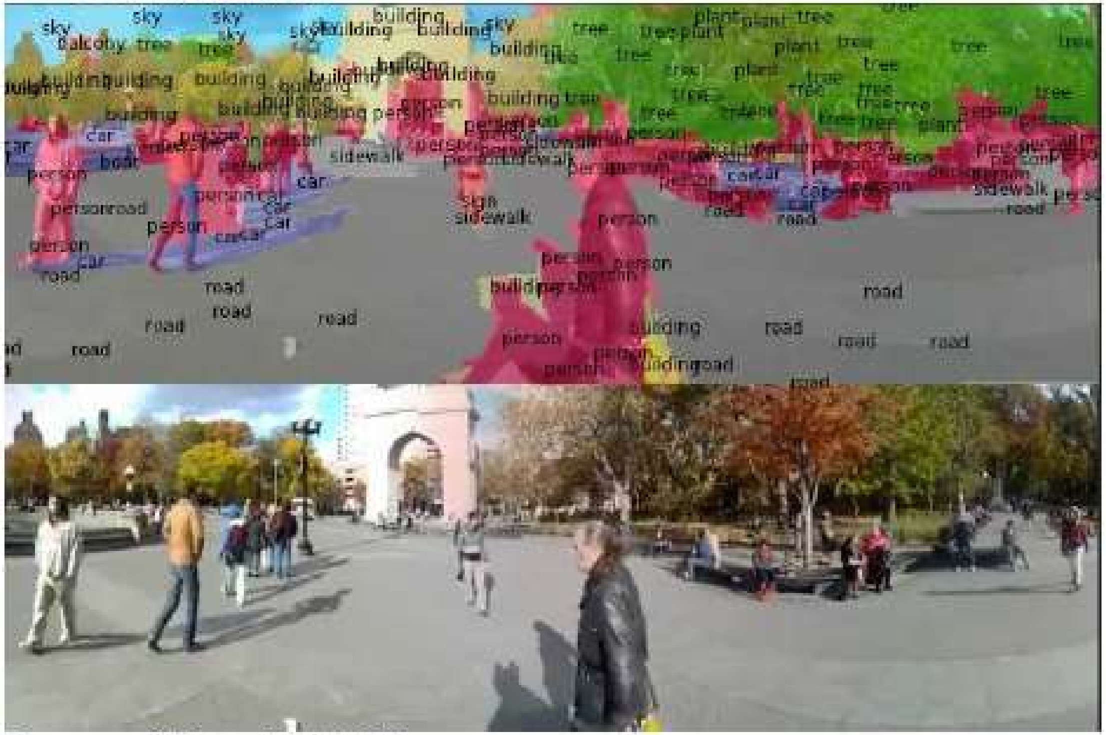
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



Figures copyright Clement Farabet, 2012. Reproduced with permission.

[Farabet et al., 2012]

ConvNets Applications

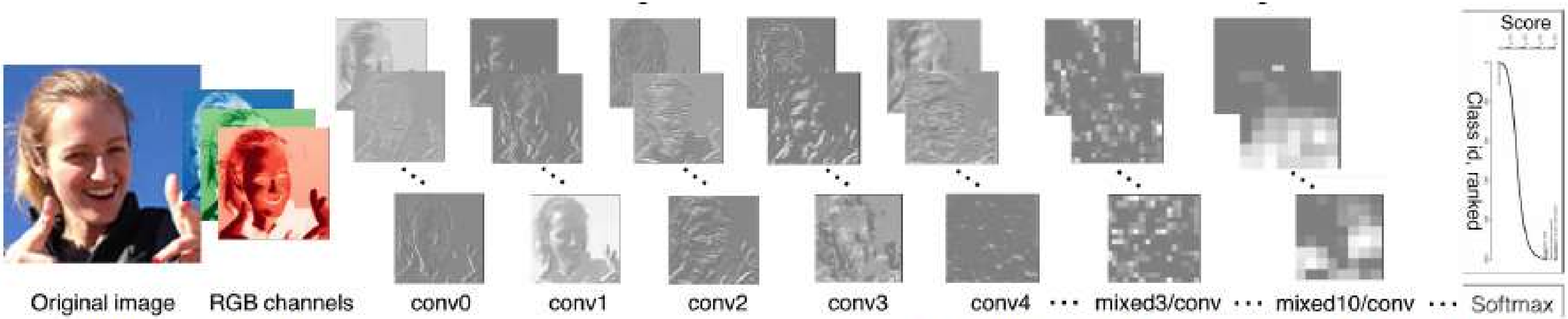
Photo by Lane McIntosh. Copyright CS231n 2017.

self-driving cars

NVIDIA Tesla line
(these are the GPUs on rye01.stanford.edu)

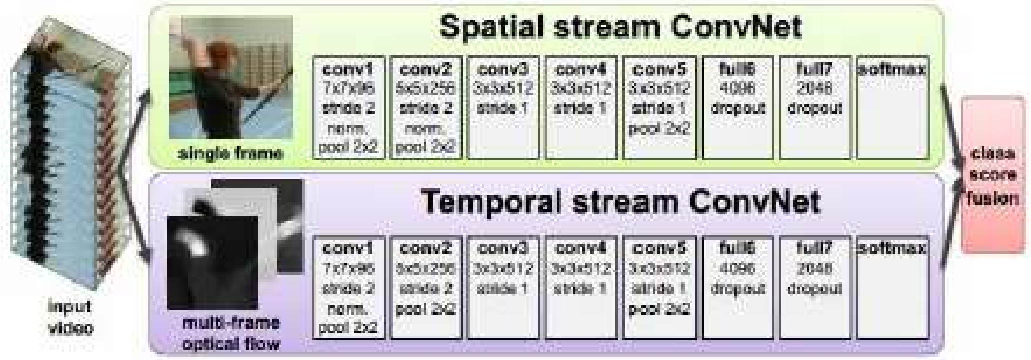
Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.

ConvNets Applications



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.

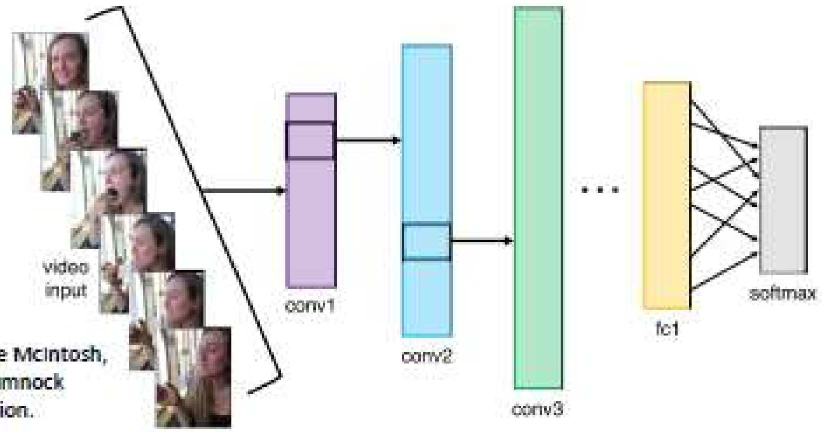


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

ConvNets Applications

[Toshev, Szegedy 2014]

[Guo et al. 2014]

How can we extract features from unstructured data?

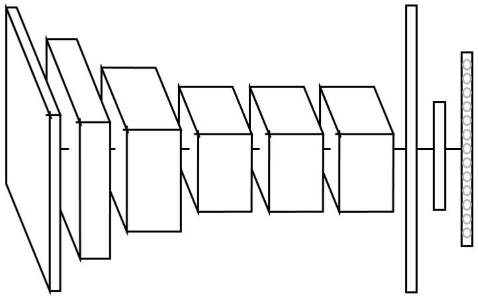
Image Domain	Translator	Feature Domain

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{128} \end{bmatrix}$$

How can we extract features from unstructured data?

Image Domain

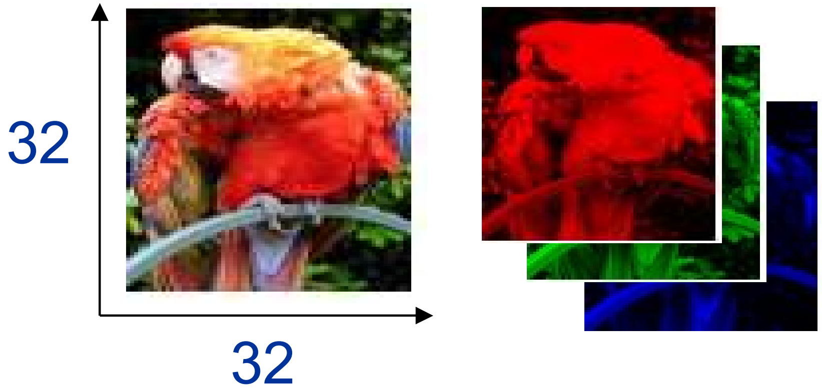
Deep Learning algorithm



Feature Domain

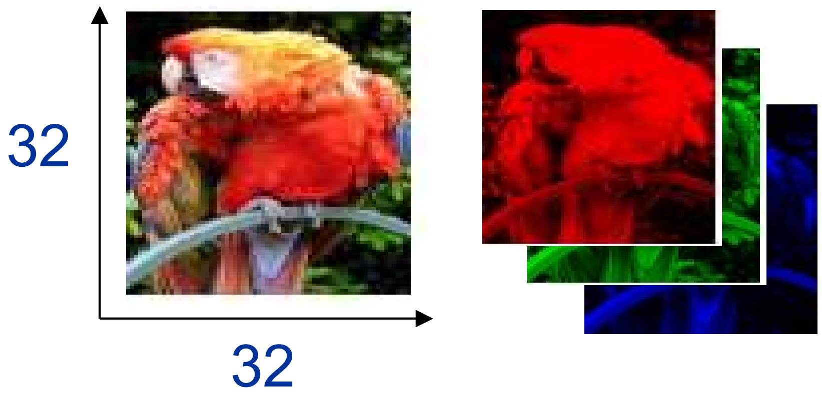
$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{128} \end{bmatrix}$$

Fully Connected Layer

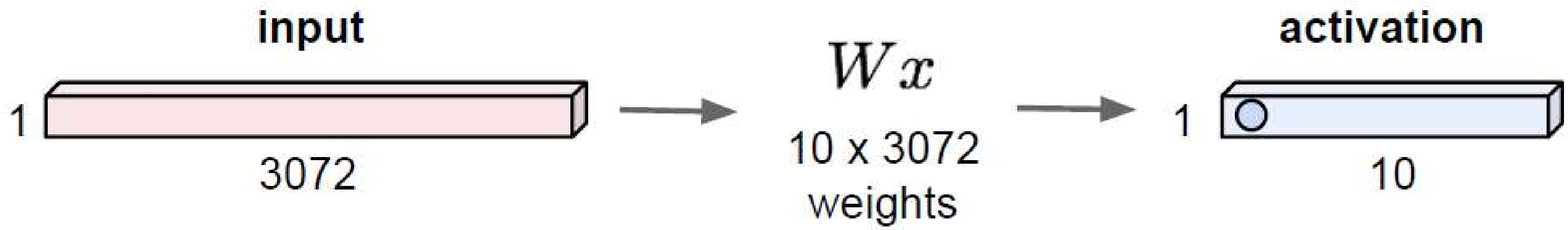


32x32x3 image -> stretch to 3072 x 1

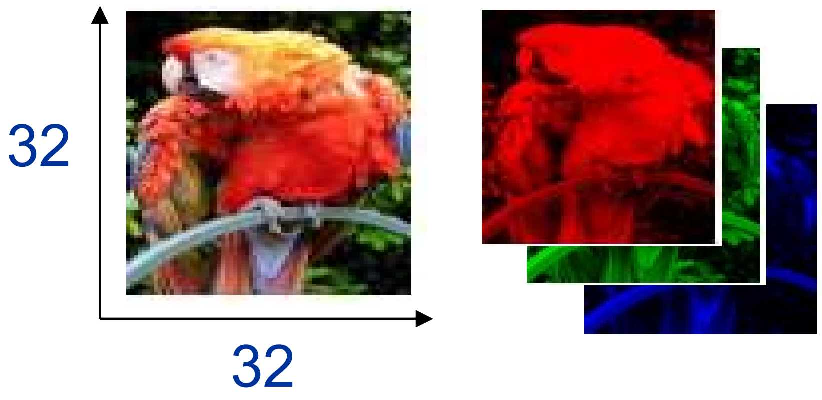
Fully Connected Layer



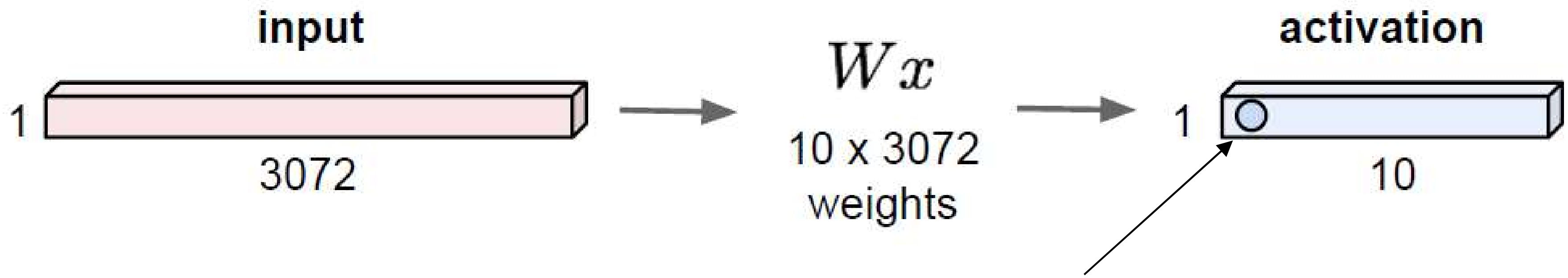
32x32x3 image -> stretch to 3072 x 1



Fully Connected Layer



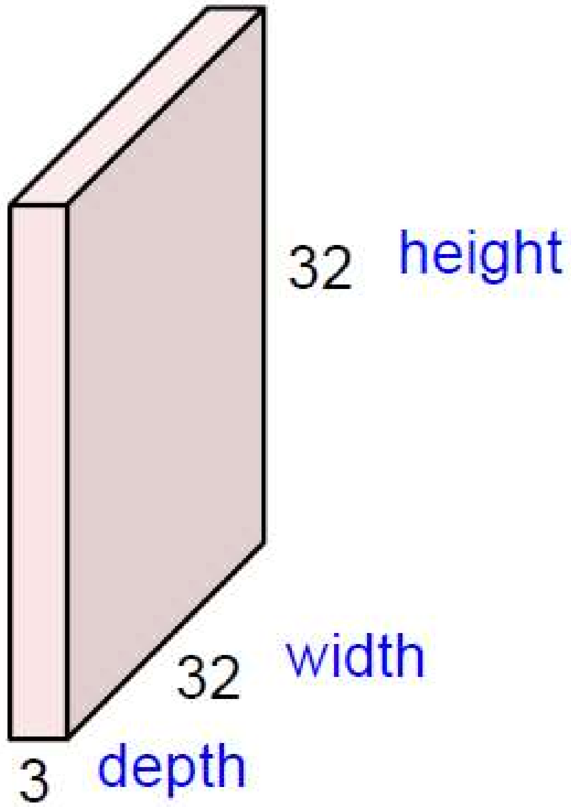
32x32x3 image -> stretch to 3072 x 1



1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

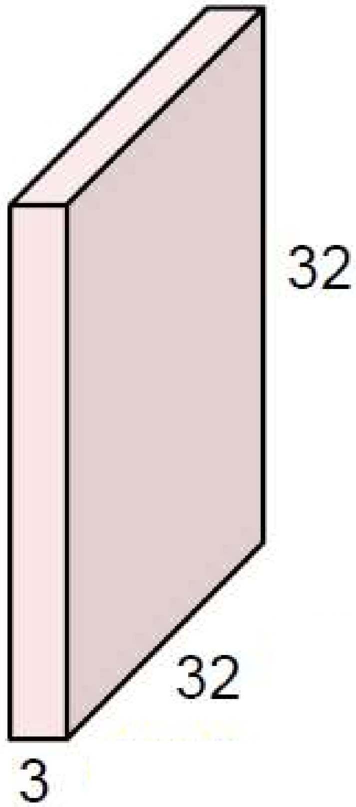
Conv Layer

32x32x3 image -> preserve spatial structure

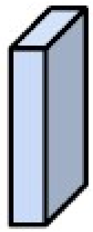


Conv Layer

32x32x3 image -> preserve spatial structure



5x5x3 filter

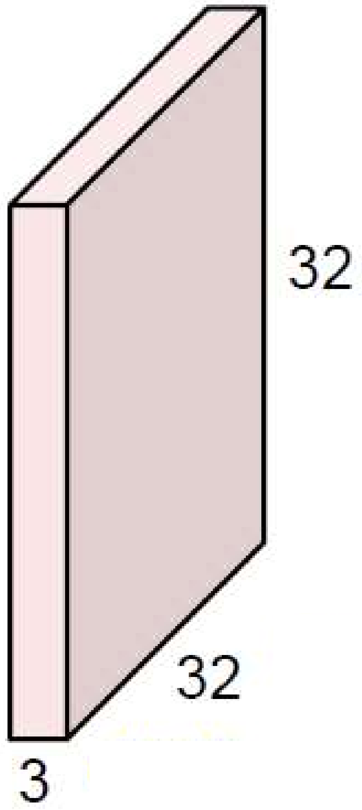


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

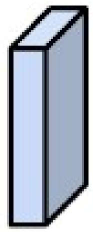
Conv Layer

32x32x3 image -> preserve spatial structure

Filters always extend the full depth of the input volume

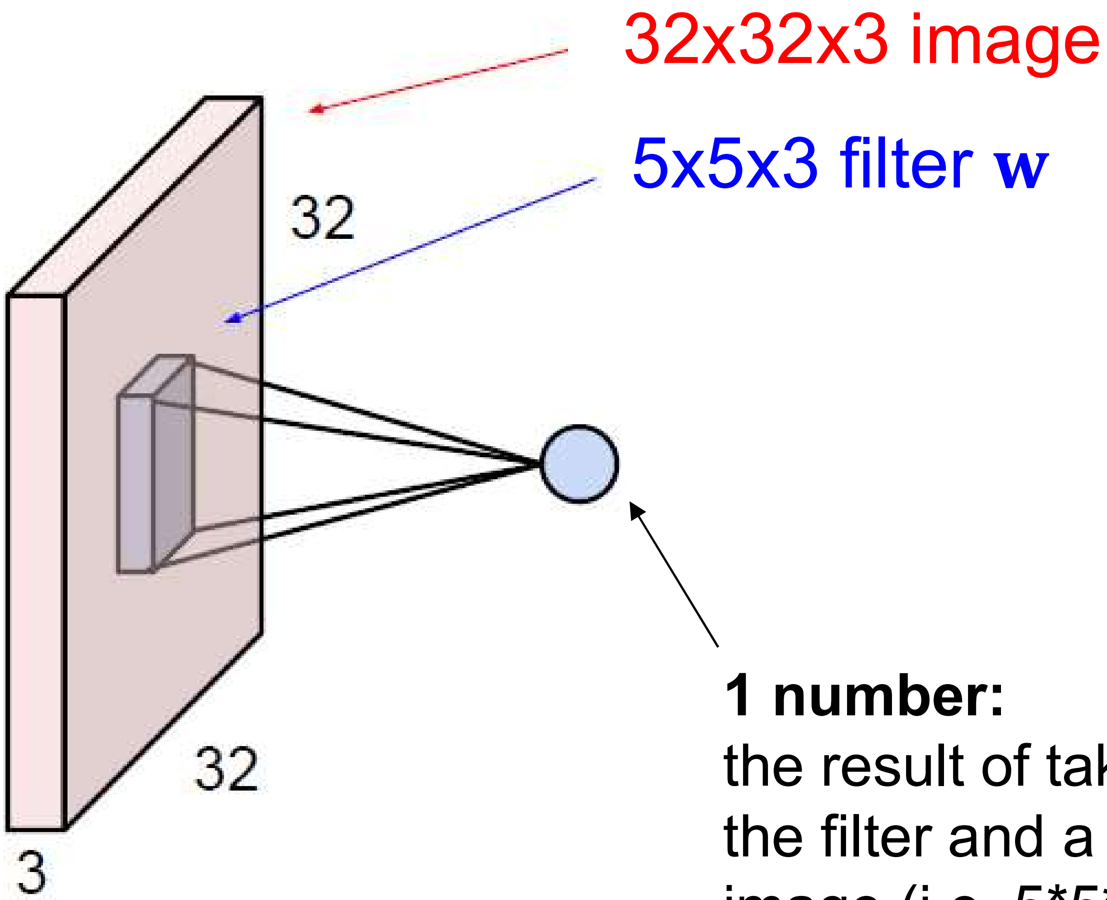


5x5x3 filter



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

Conv Layer

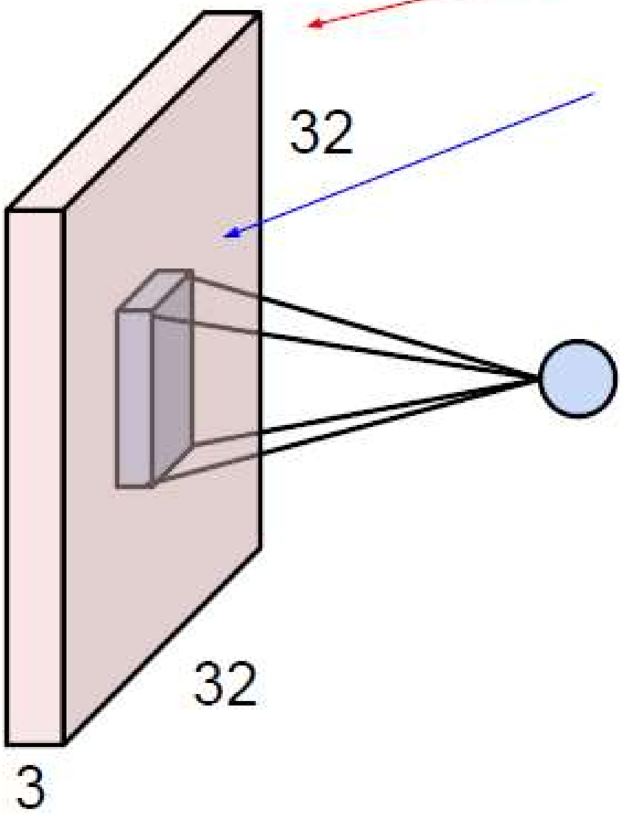


1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

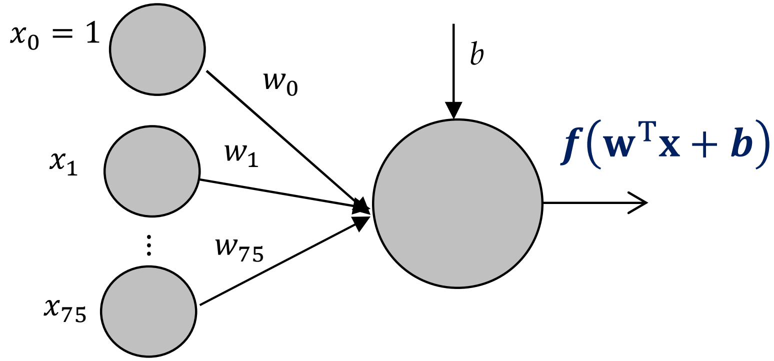
Conv Layer

32x32x3 image

5x5x3 filter $\mathbf{w}$

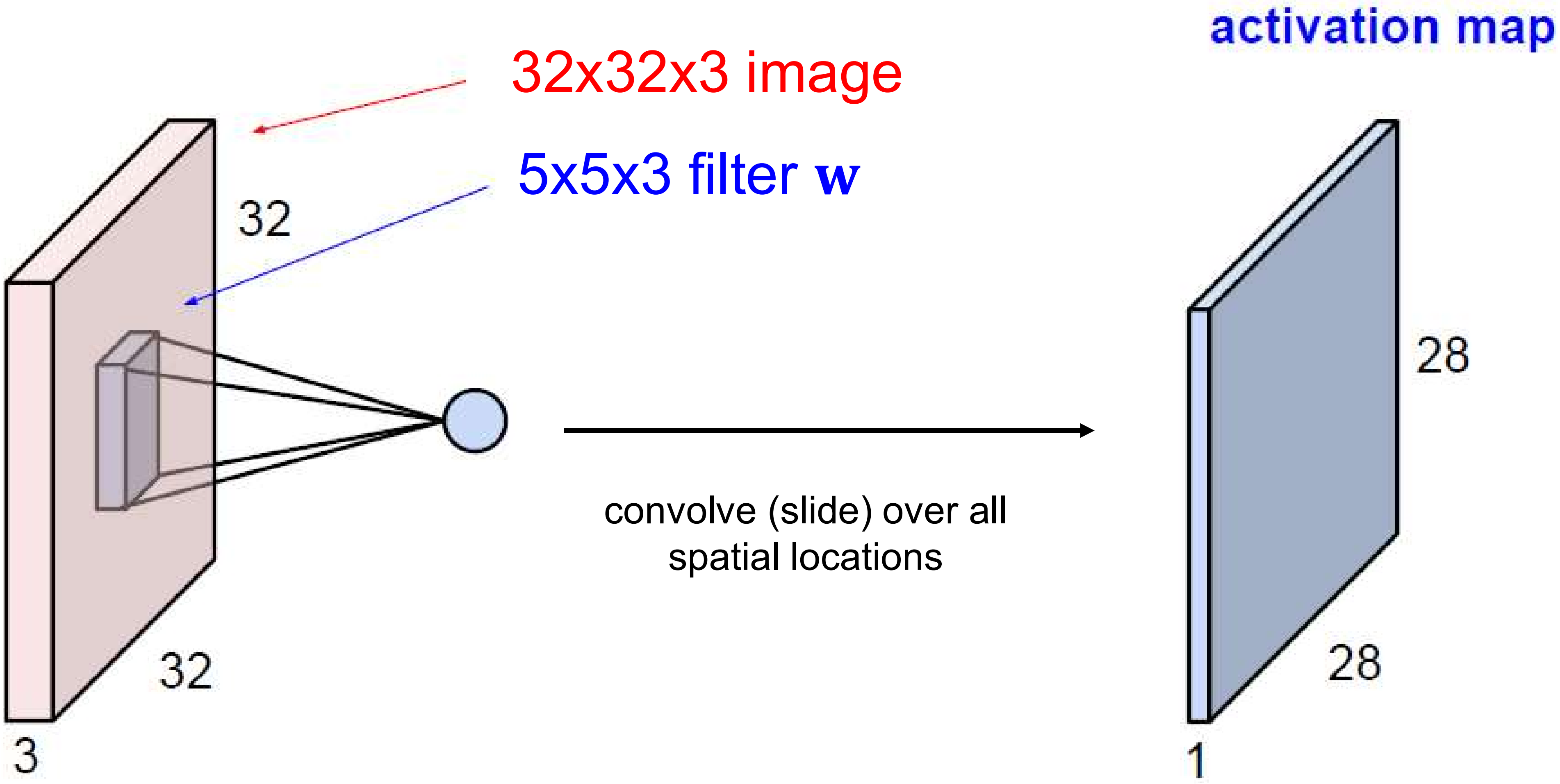


The filter is just a neuron with local connectivity...



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)

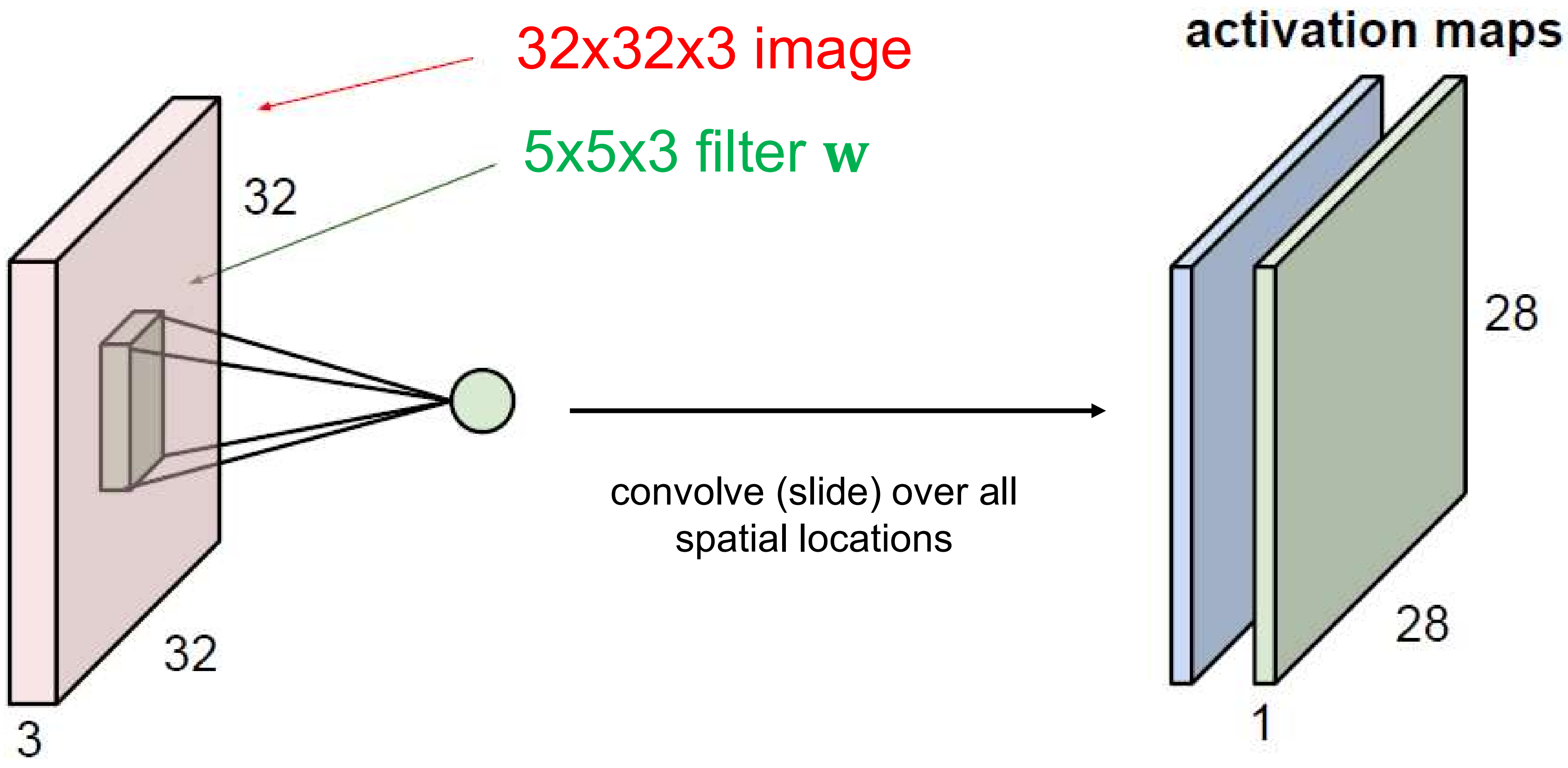
Conv Layer



activation map
32x32x3 image
5x5x3 filter **w**
32
32
3
convolve (slide) over all spatial locations
28
28
1

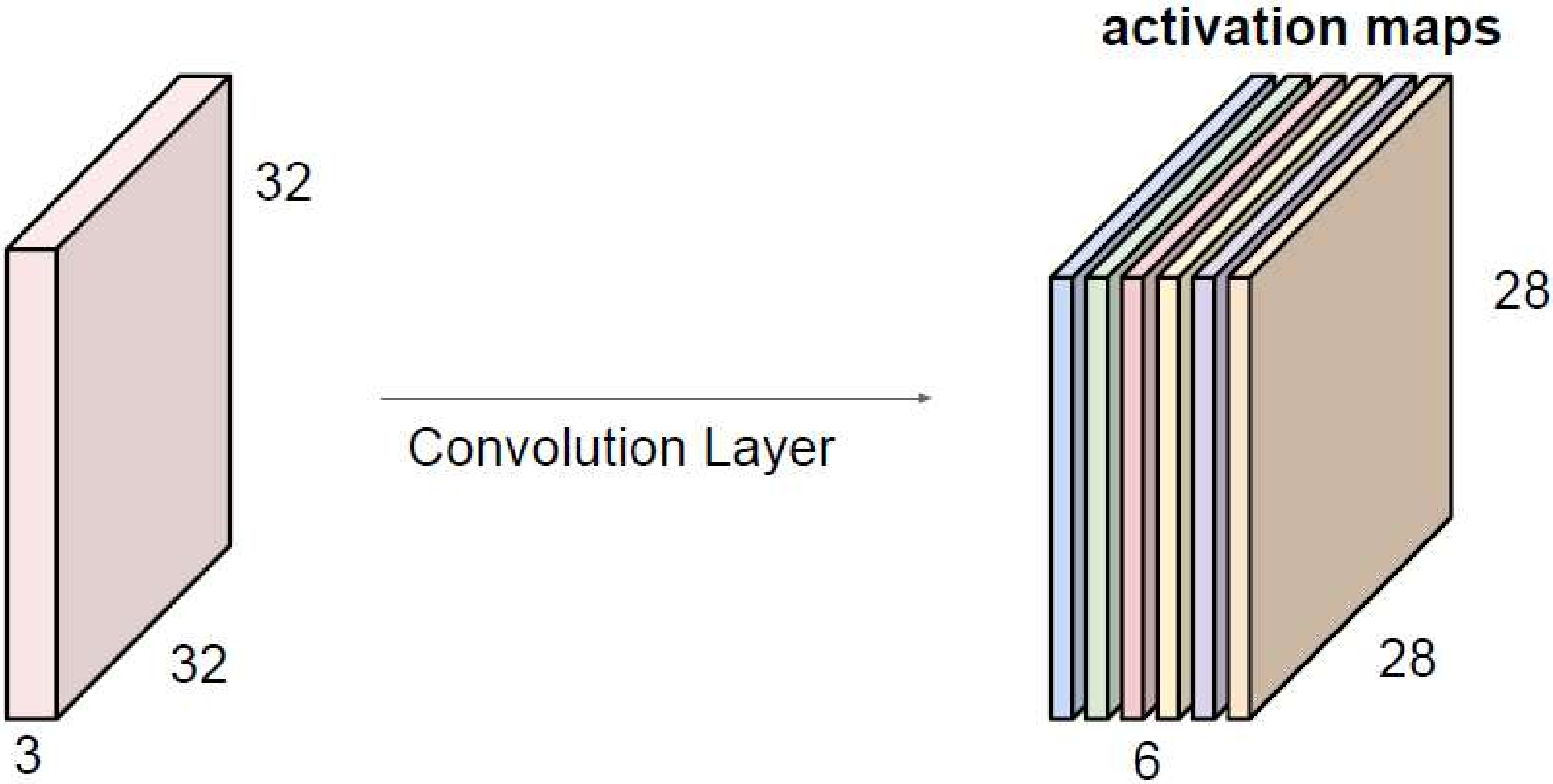
Conv Layer

Consider a second filter



Conv Layer

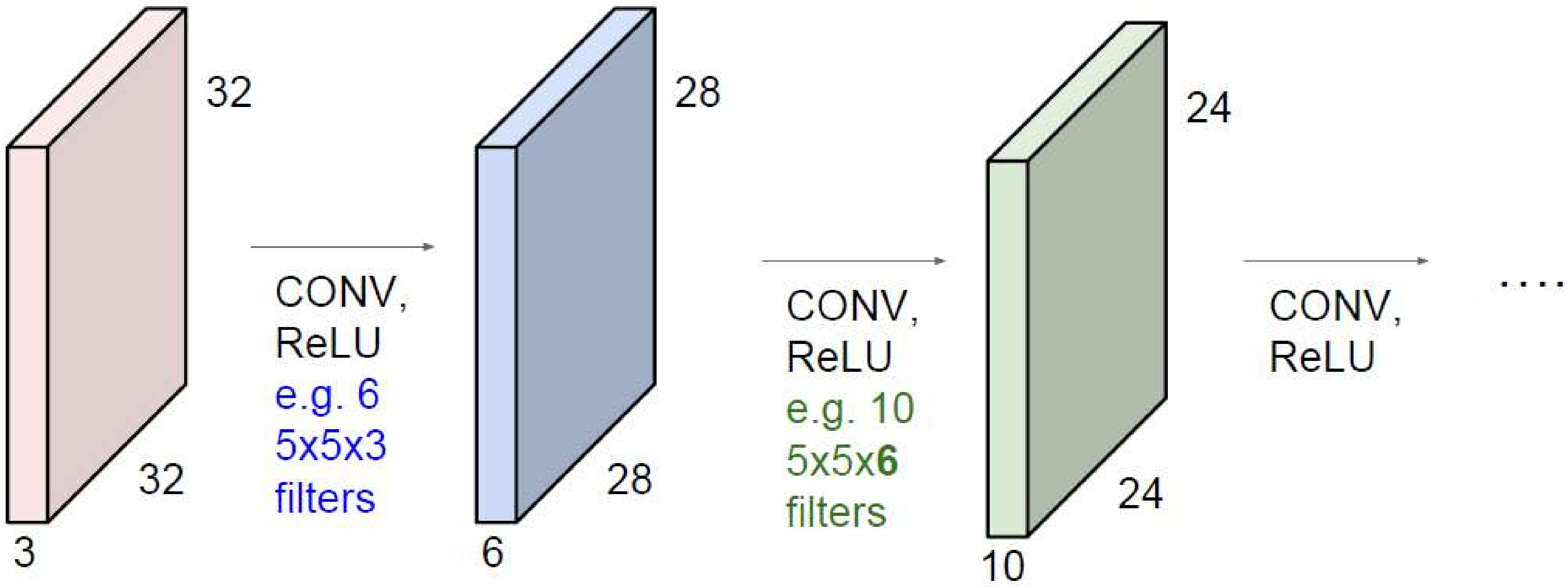
For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



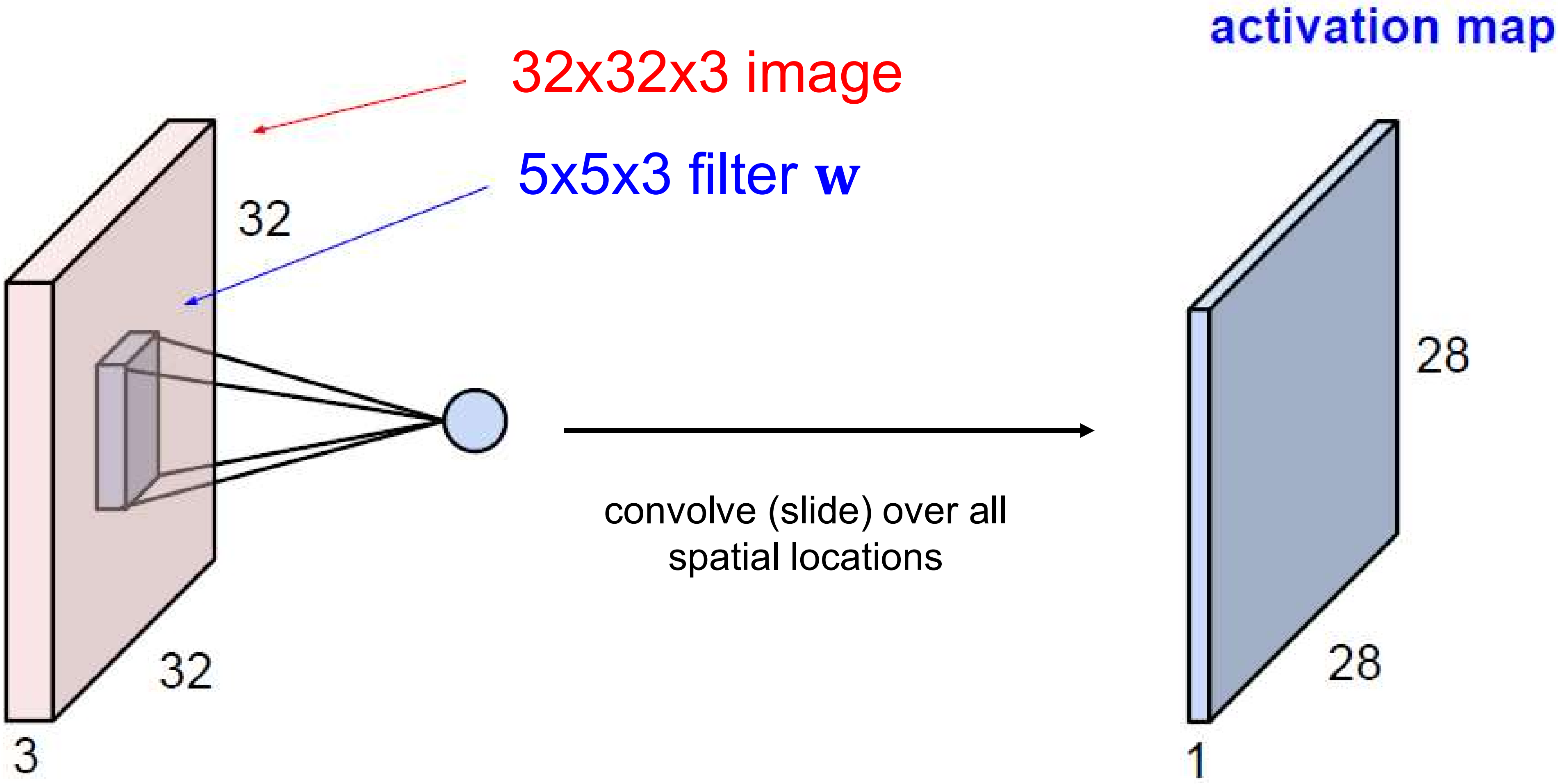
We stack these up to get a "new image" of size 28x28x6!

ConvNet

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

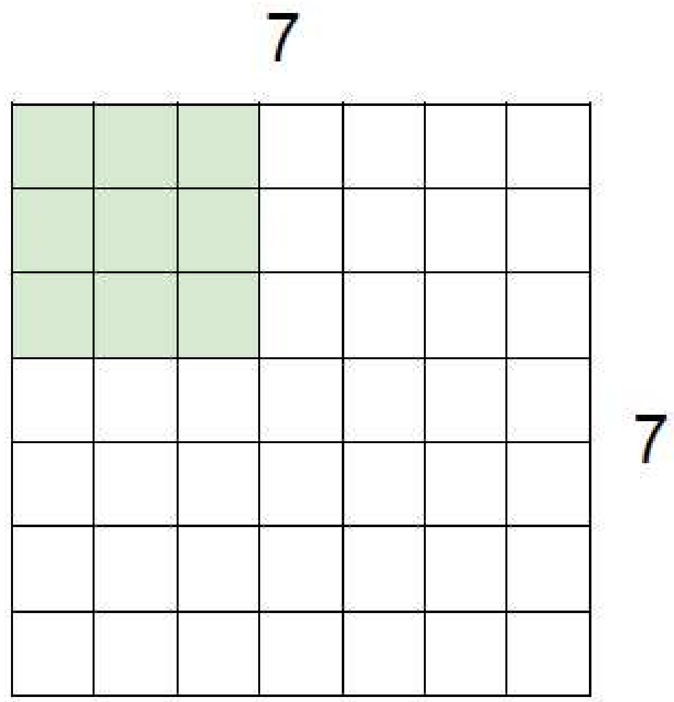


Spatial Dimensions



Spatial Dimensions

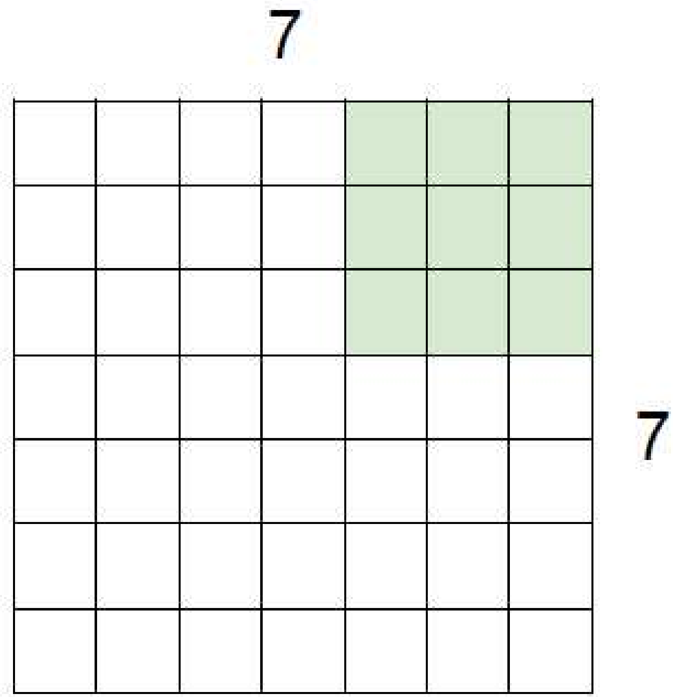
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions

A closer look at spatial dimensions:

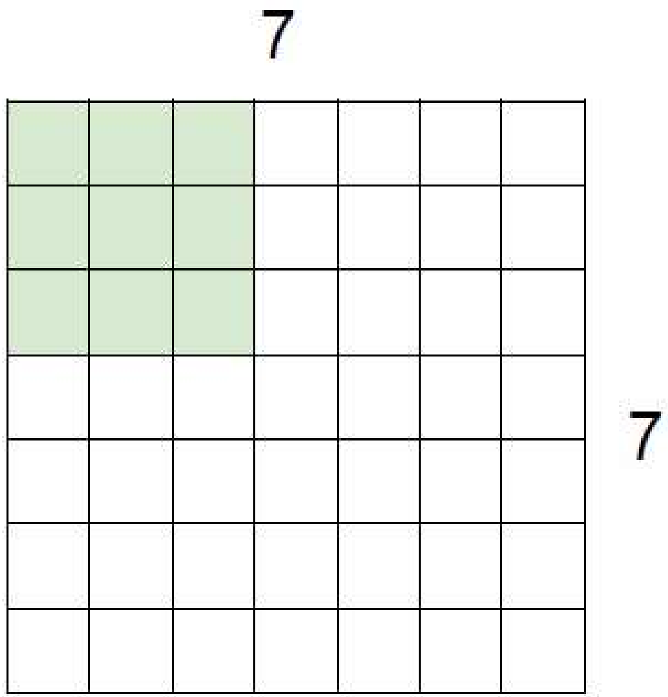


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Spatial Dimensions

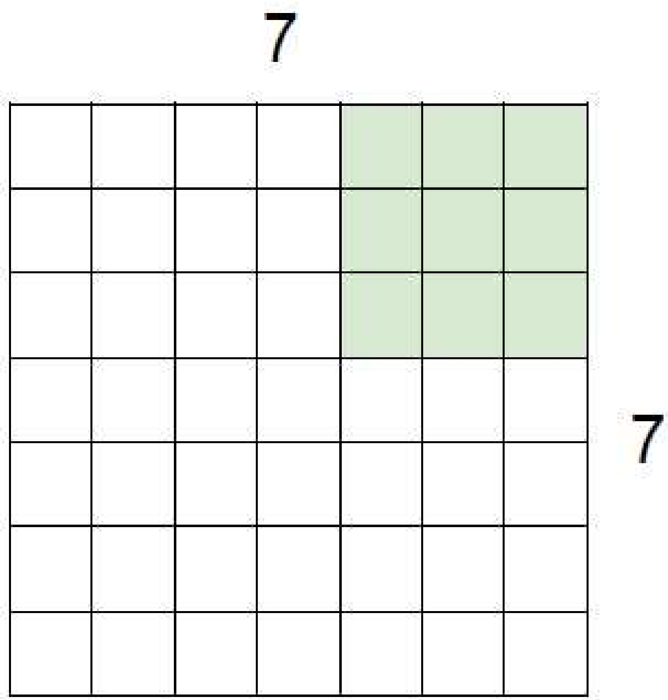
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Spatial Dimensions

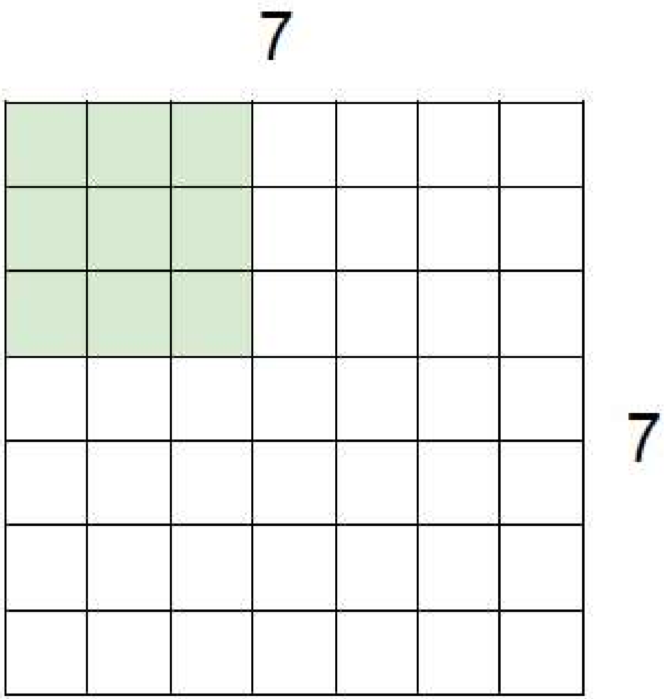
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Spatial Dimensions

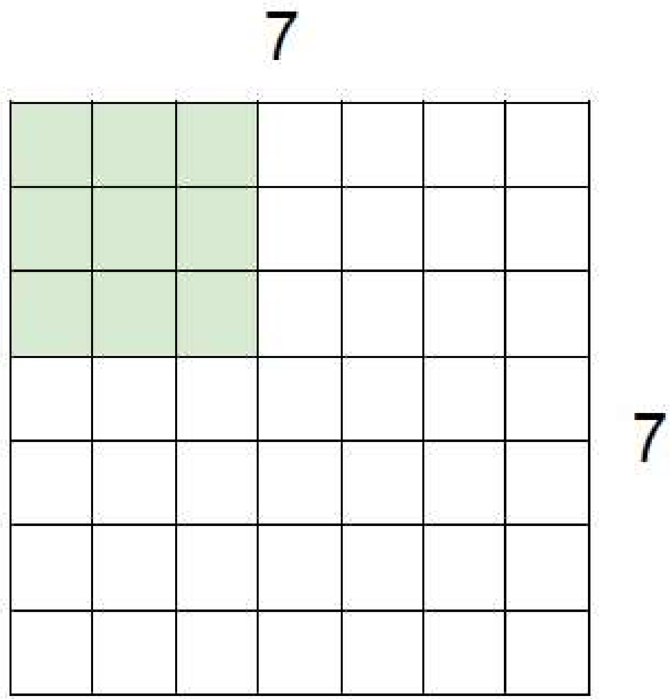
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Spatial Dimensions

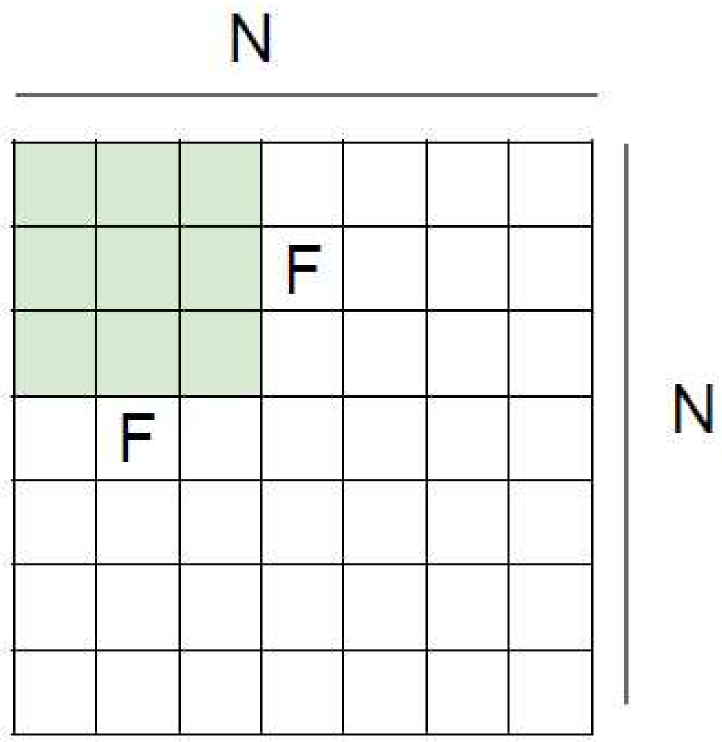
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

Spatial Dimensions

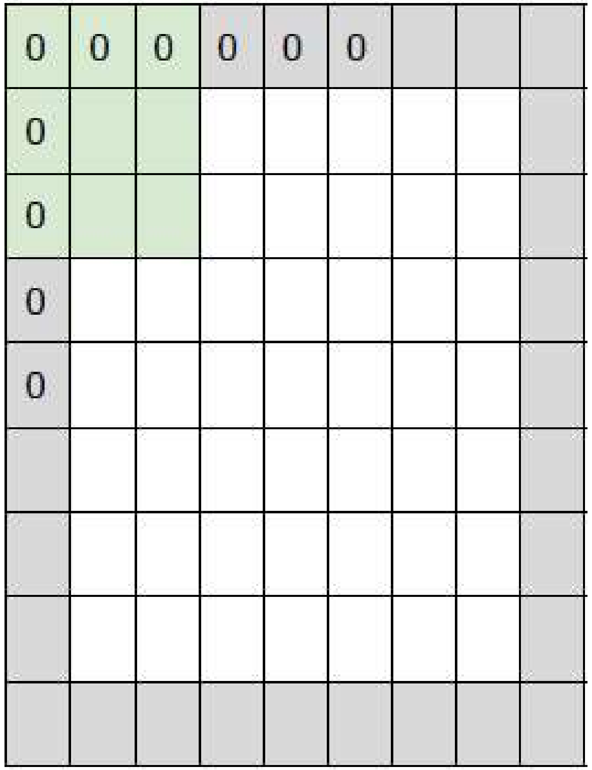


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Spatial Dimensions

In practice: Common to zero pad the border

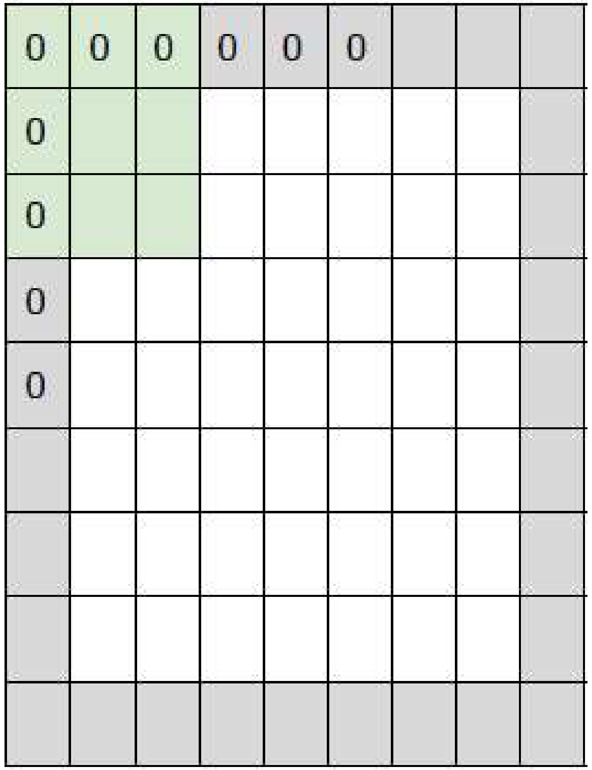


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Spatial Dimensions

In practice: Common to zero pad the border



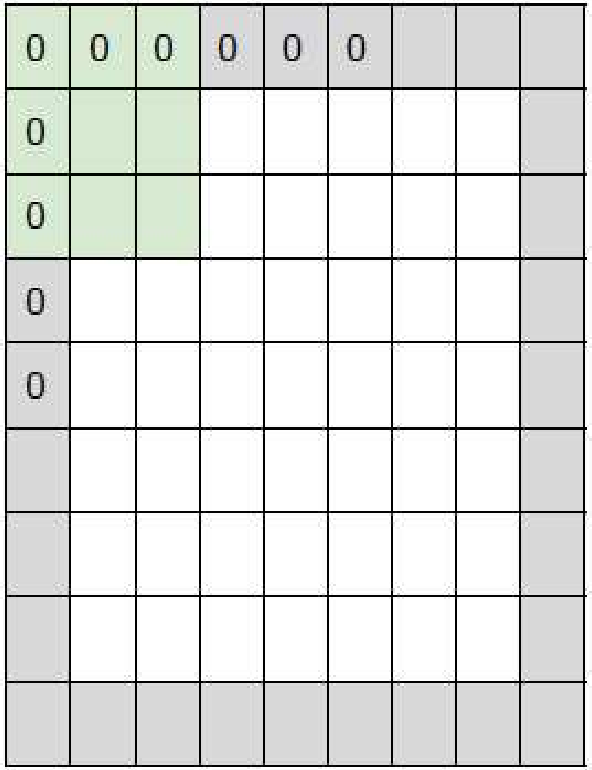
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

(recall:)
(N + 2P - F) / stride + 1

Spatial Dimensions

In practice: Common to zero pad the border



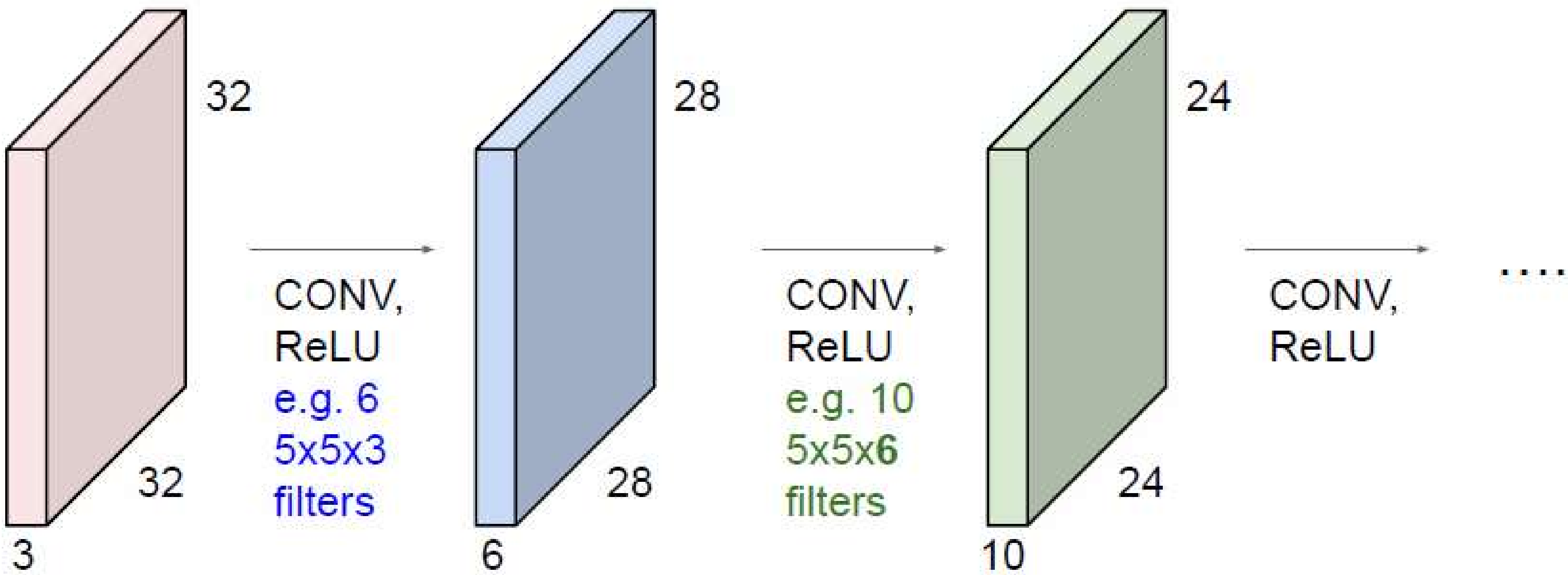
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

Spatial Dimensions

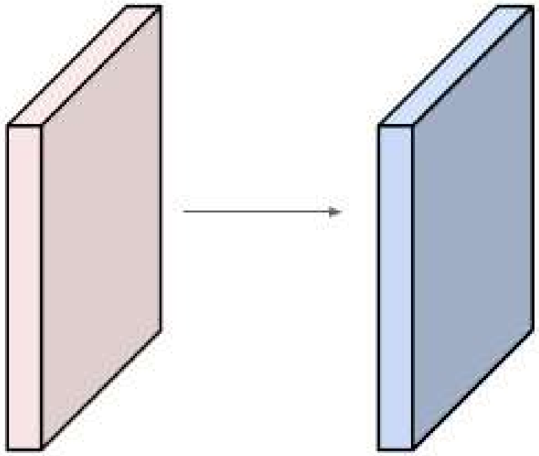
Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Spatial Dimensions

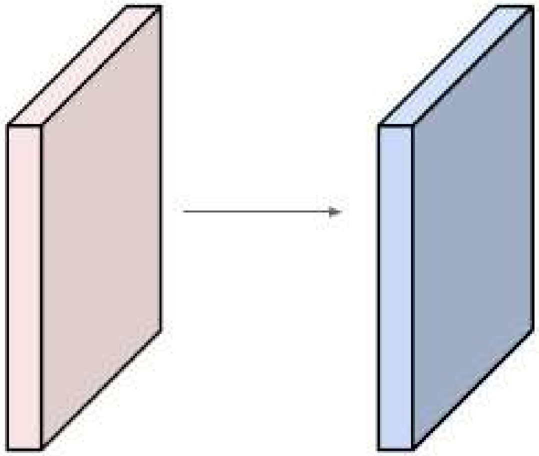
Examples time:



Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Spatial Dimensions



Examples time:

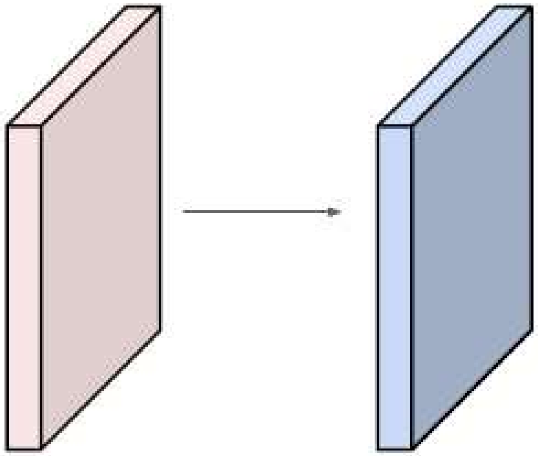
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

(32+2*2-5)/1+1 = 32 spatially, so
32x32x10

Spatial Dimensions

Examples time:



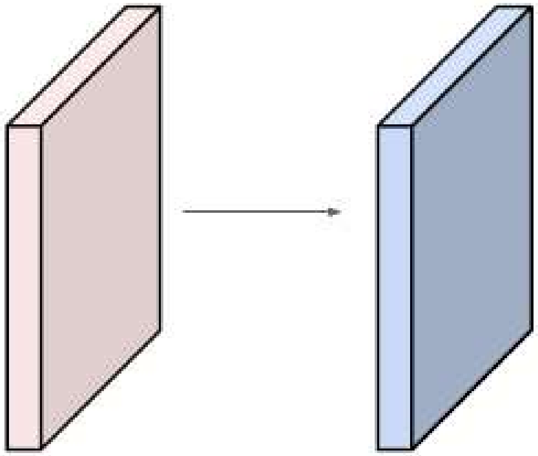
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

Spatial Dimensions

Examples time:

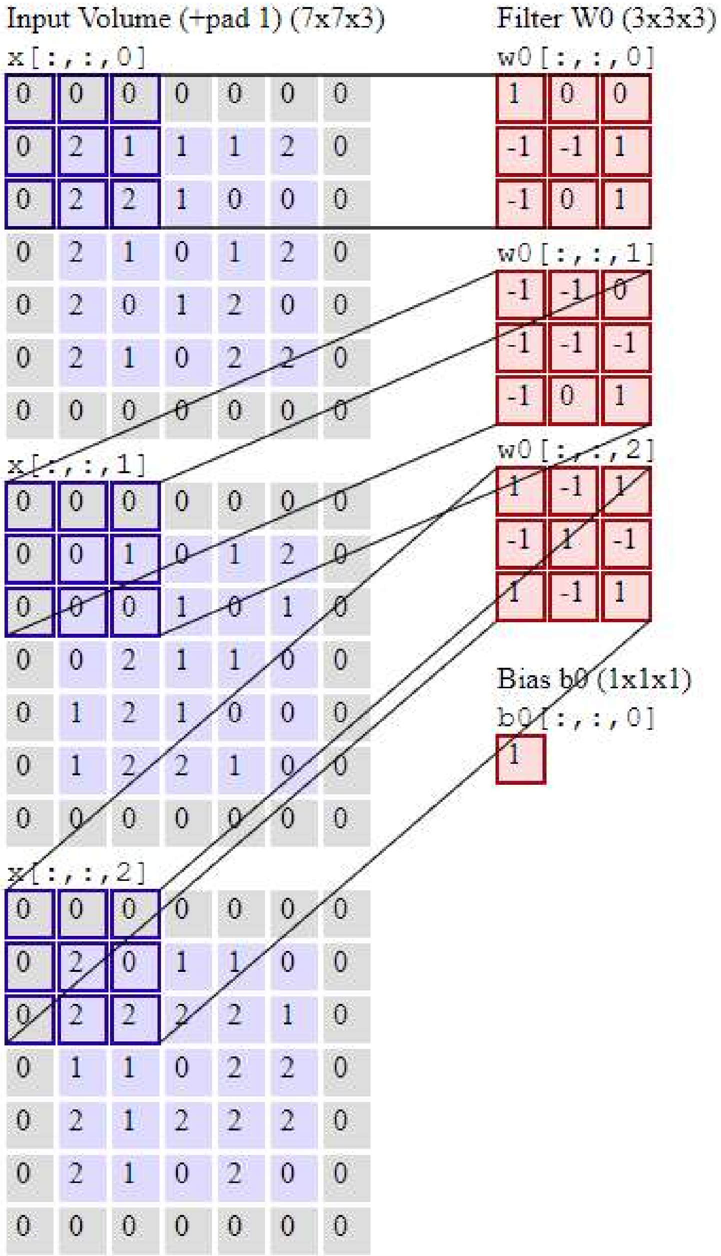
Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2



Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Spatial Dimensions



Input volume=5x5x3

Filter 3x3x3

Stride =2

Output

o[:,:,0]

3	-2	-5
-1	-2	-5
-4	-8	-4

Escuela Pol

BiDA Lab

Spatial Dimensions

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Spatial Dimensions

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$
where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

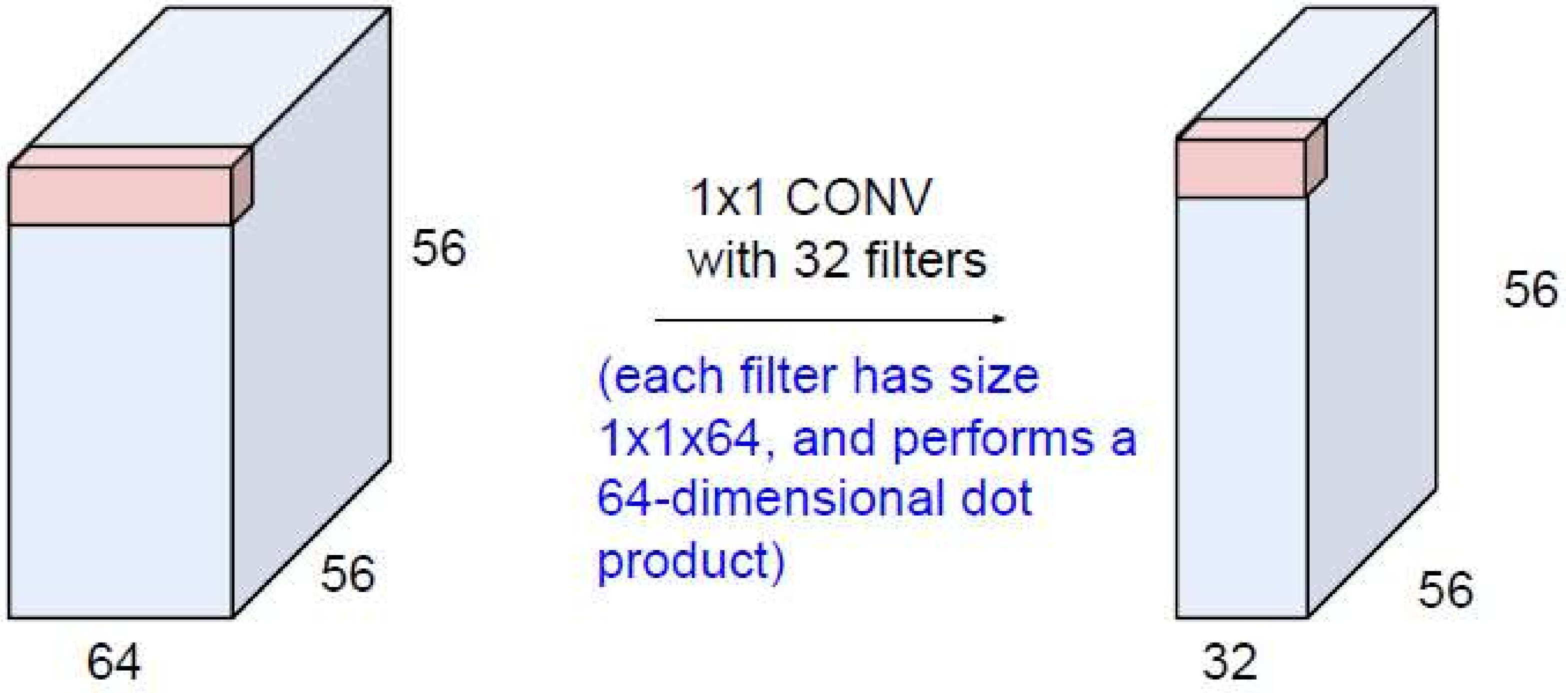
Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

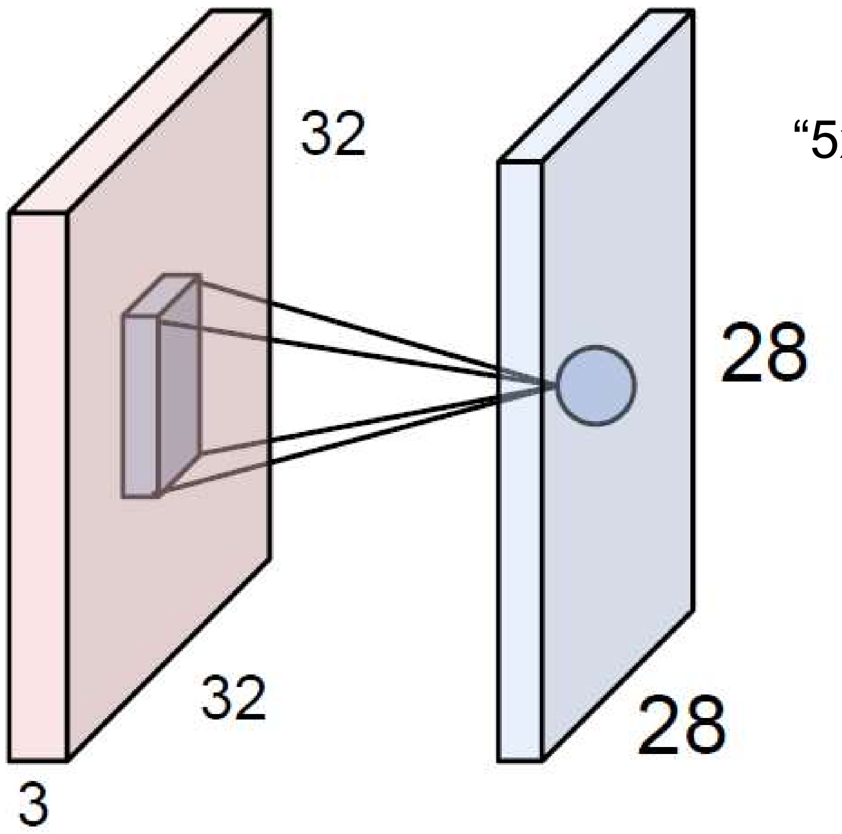
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Spatial Dimensions

(btw, 1x1 convolution layers make perfect sense)



Conv Layer



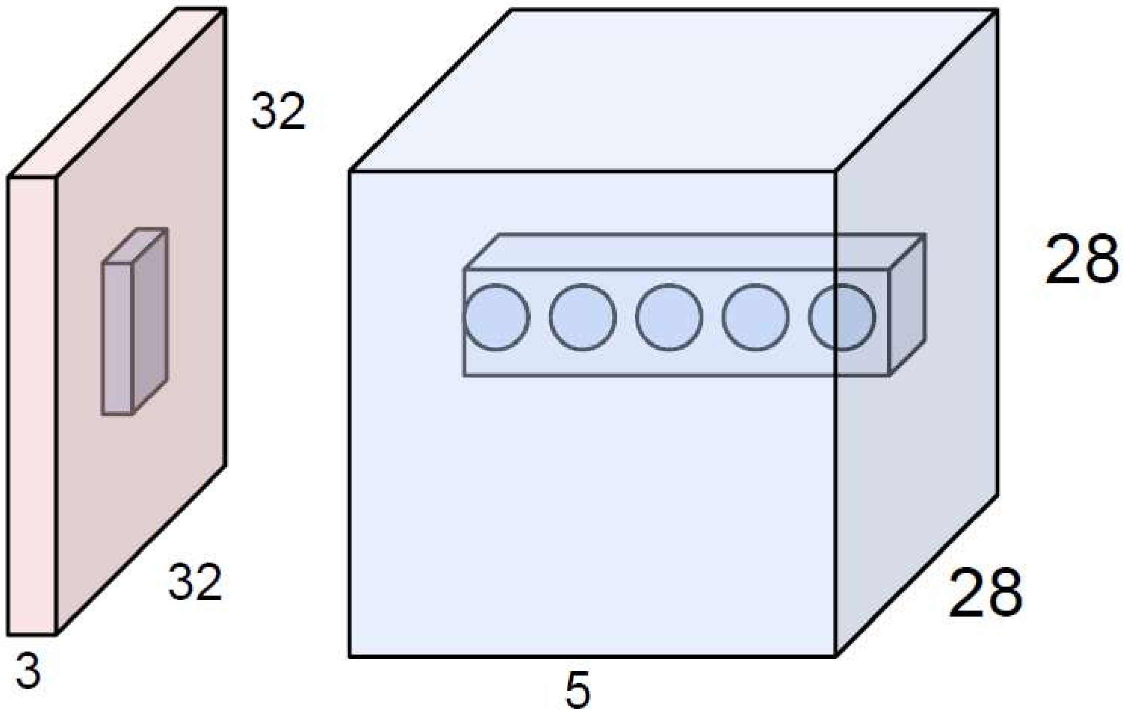
An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”

Conv Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

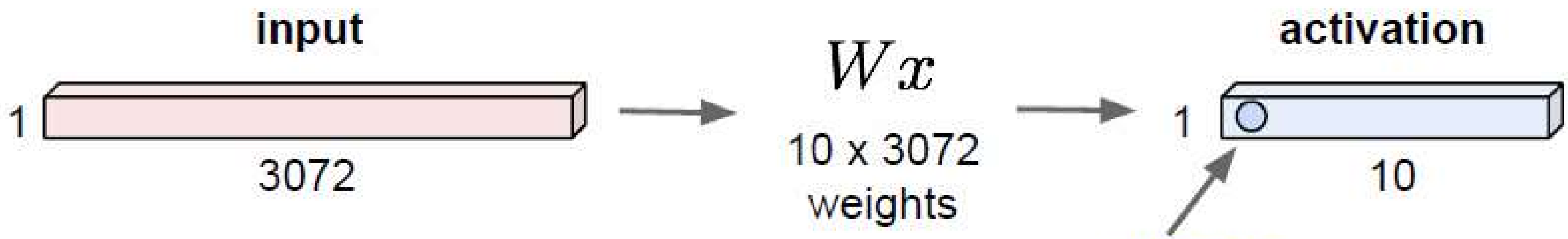
There will be 5 different neurons all looking at the same region in the input volume



Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

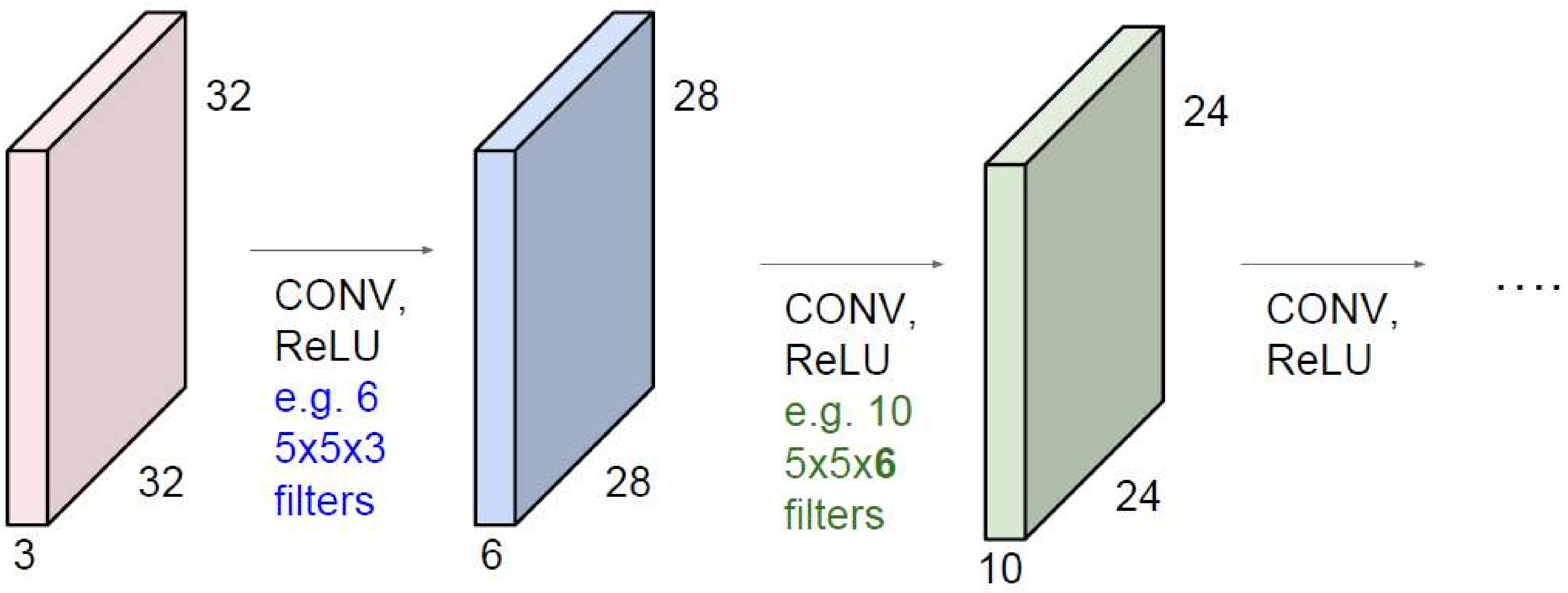
Each neuron looks at the full input volume



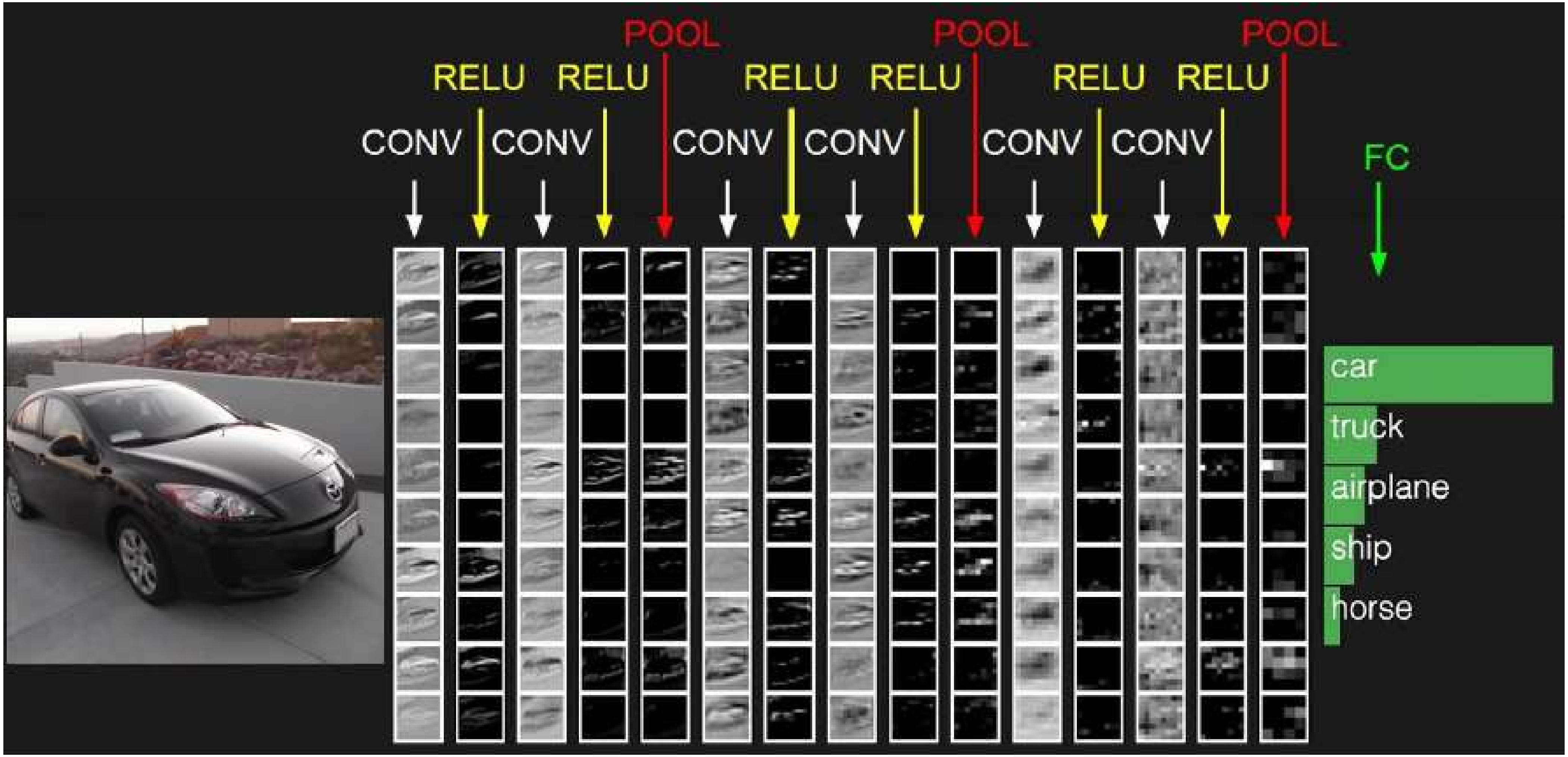
1 number:
the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)

ConvNet

Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

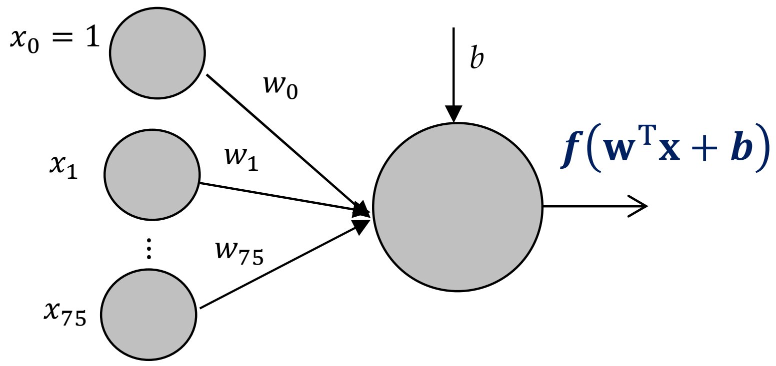


ConvNet

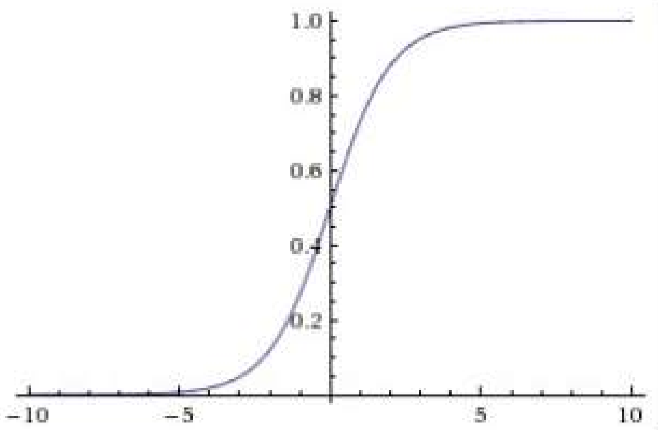


Activation Functions

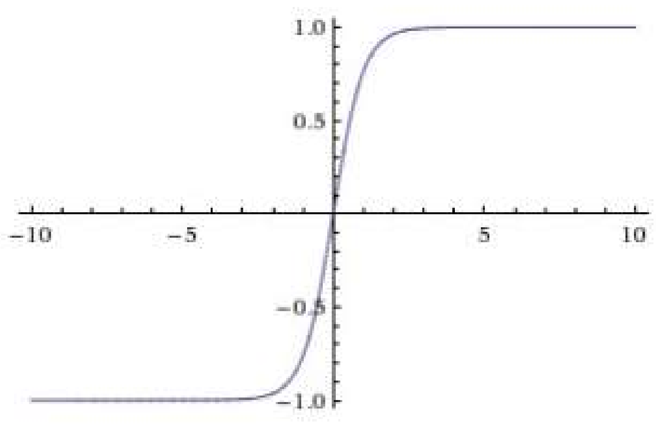
f is the activation function:



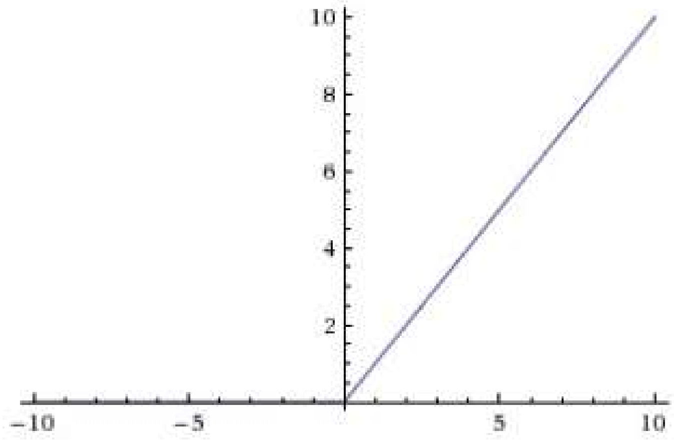
These are non-linear functions applied at the output values



Sigmoid



Tanh



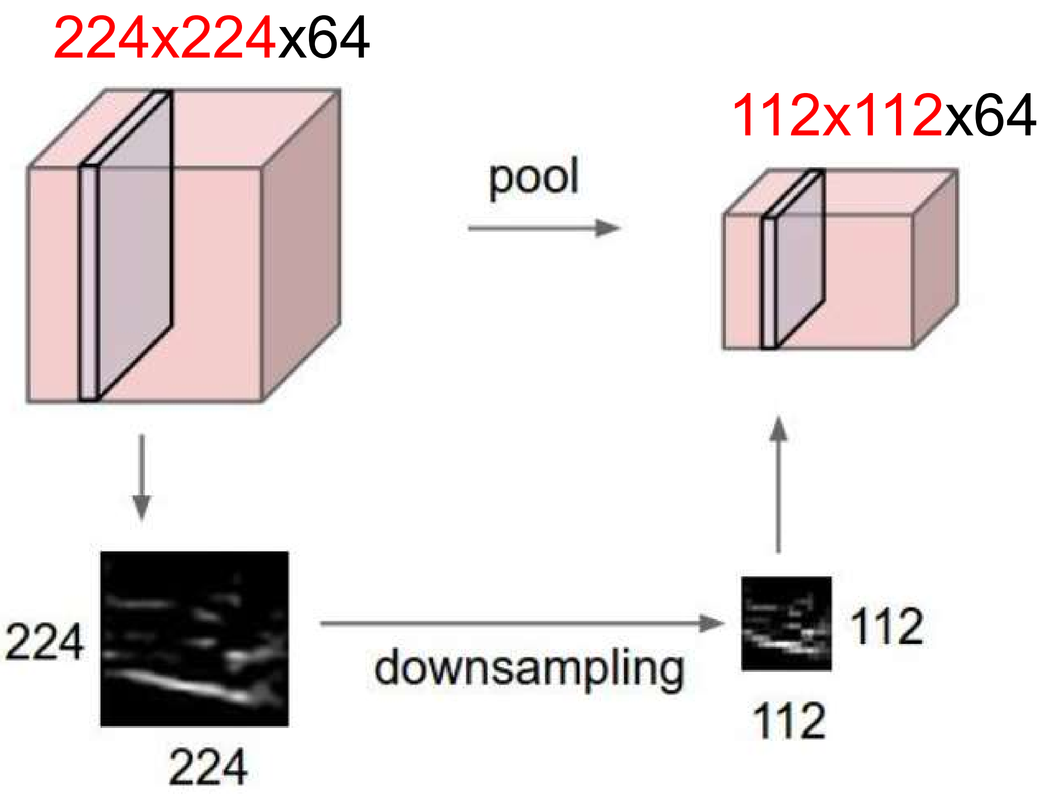
ReLu

Very popular, simple implementation

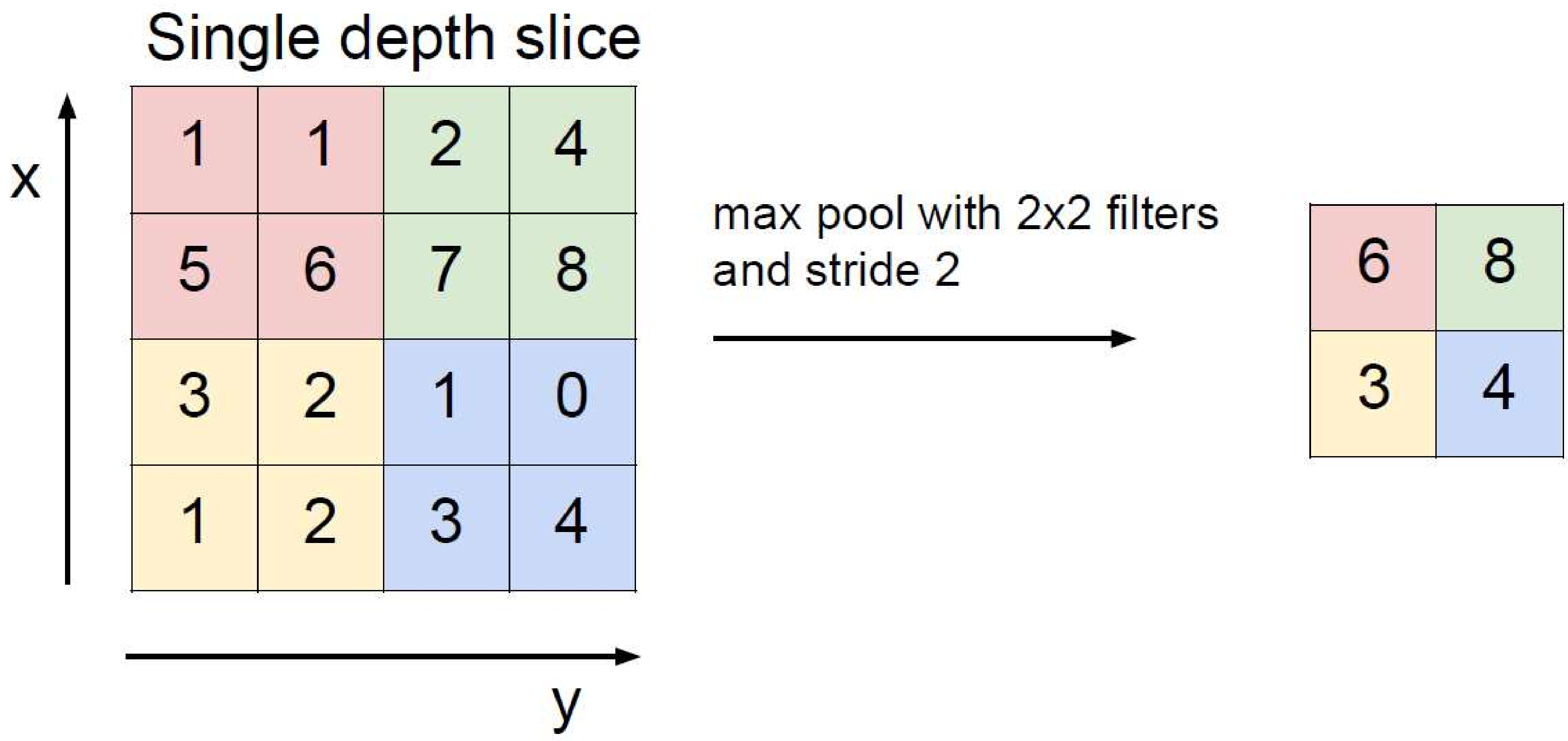
Pooling Layer

Makes the representations smaller and more manageable

Operates over each activation map independently:

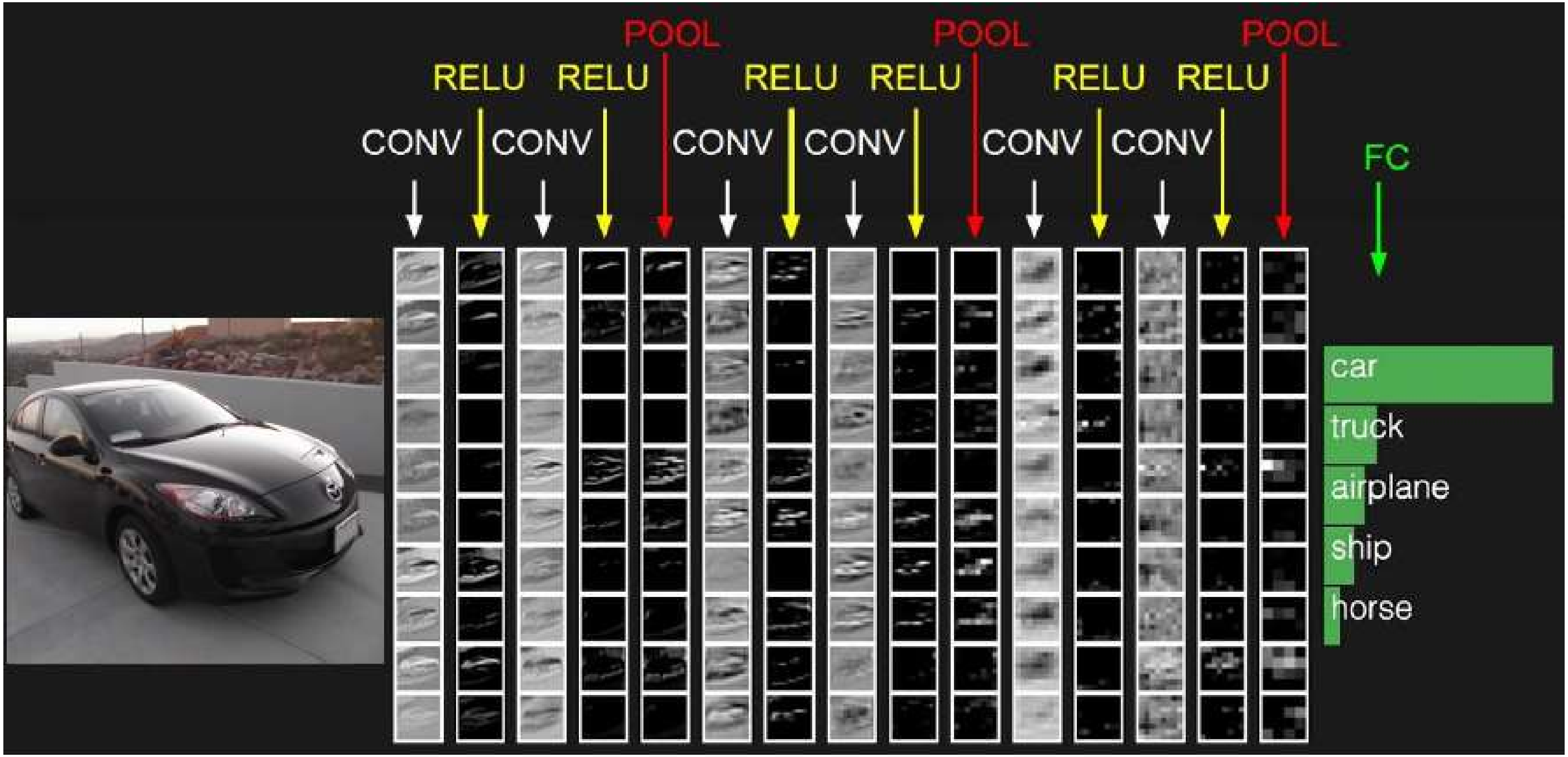


Pooling Layer: Max Pooling



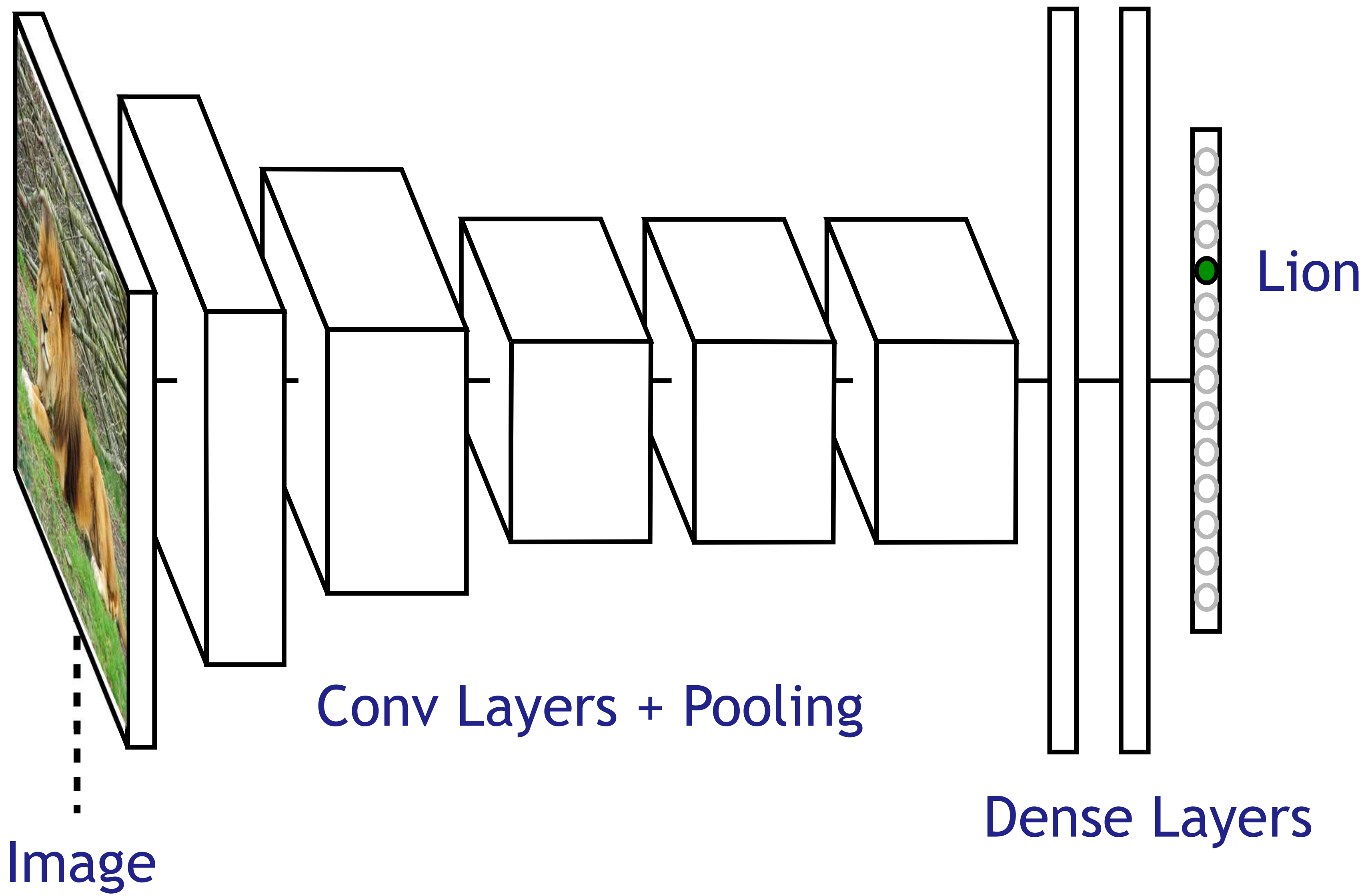
Fully Connected Layer

Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Convolutional Neural Networks: AlexNet

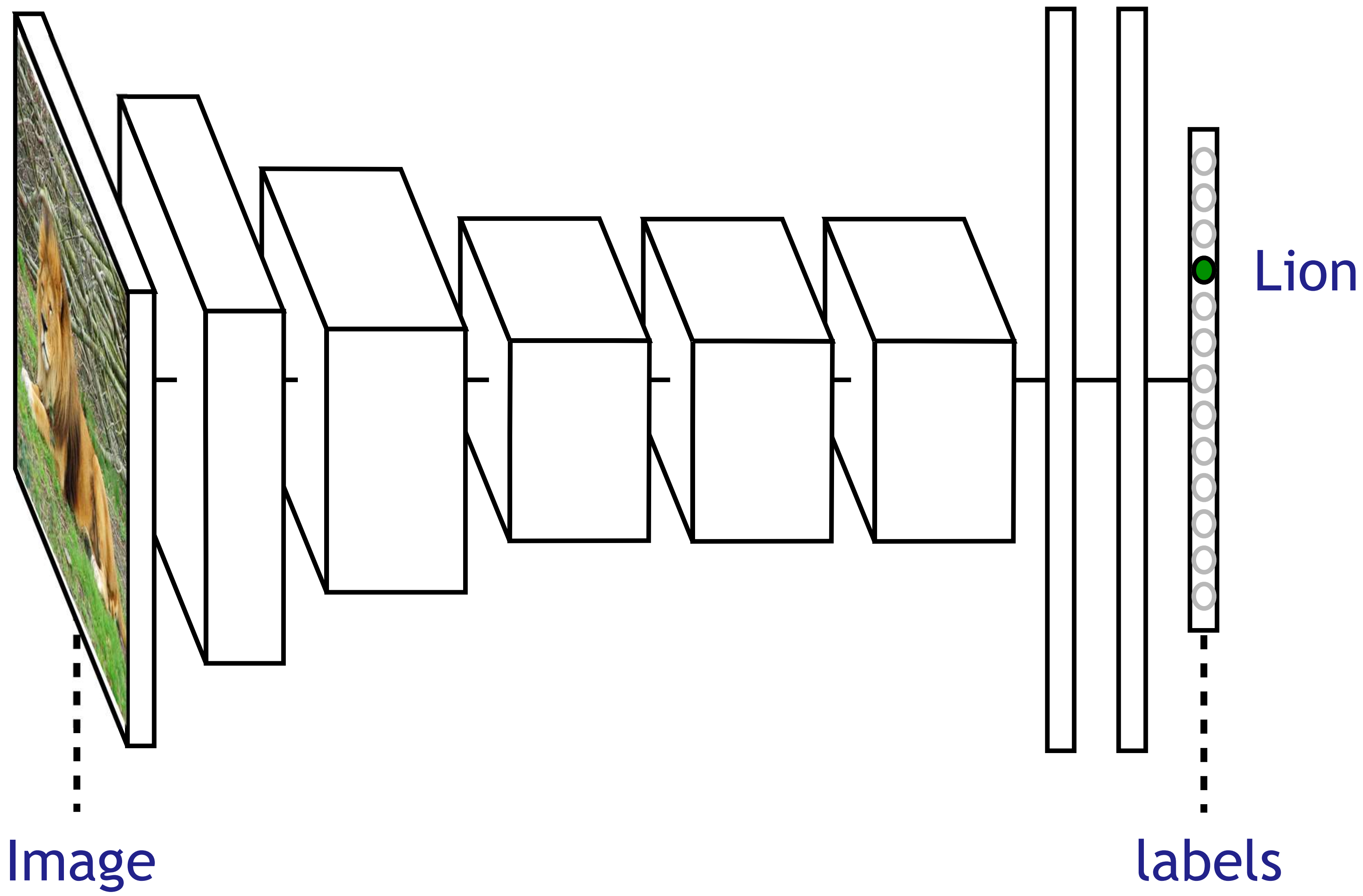
Slides from Jason Yosinski



Krizhevsky, Sutskever, Hinton — NIPS 2012

Convolutional Neural Networks: AlexNet

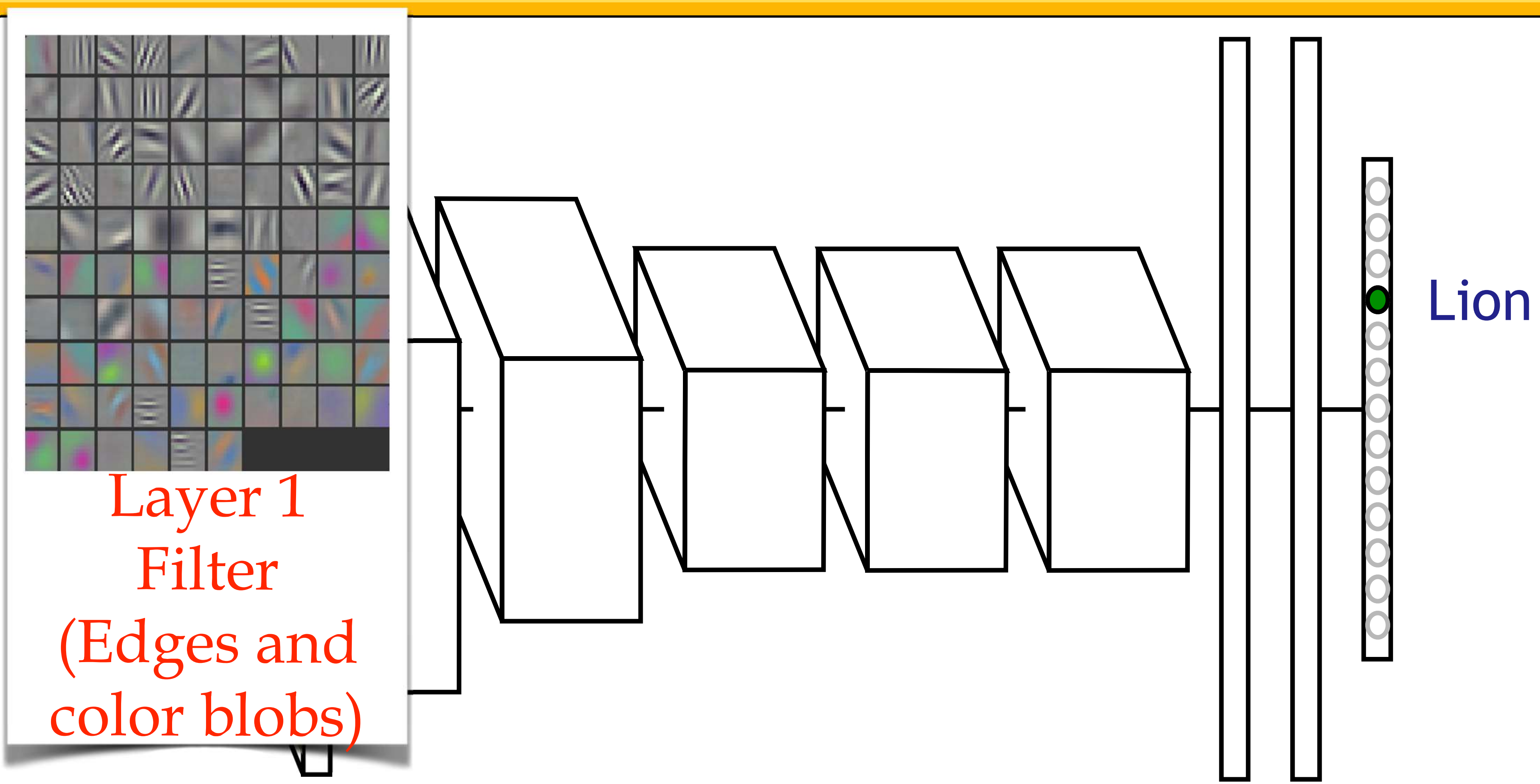
Slides from Jason Yosinski



Krizhevsky, Sutskever, Hinton — NIPS 2012

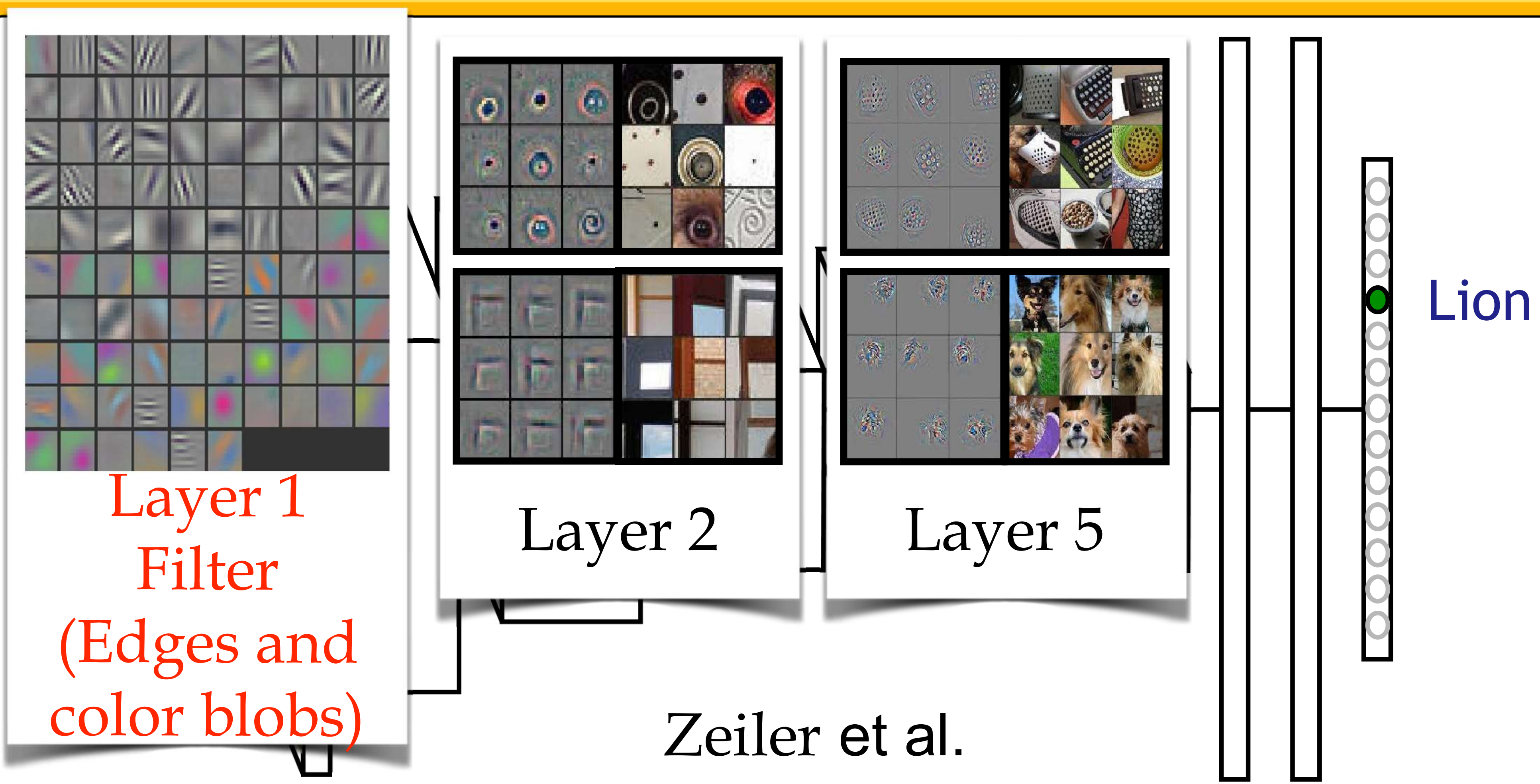
Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski



Convolutional Neural Networks: AlexNet

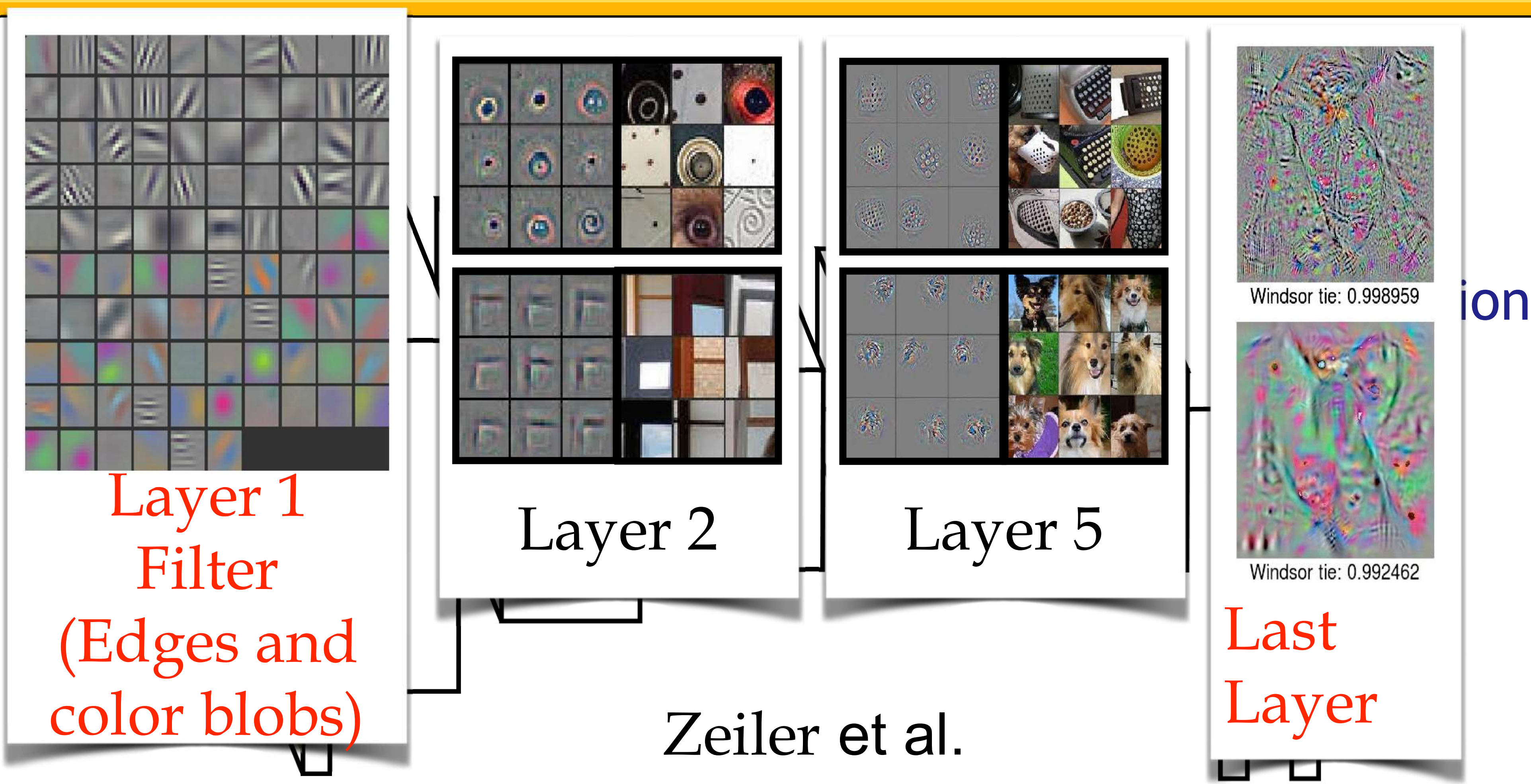
Slides from Jason Yosinski



Zeiler et al.
arXiv 2013

Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski

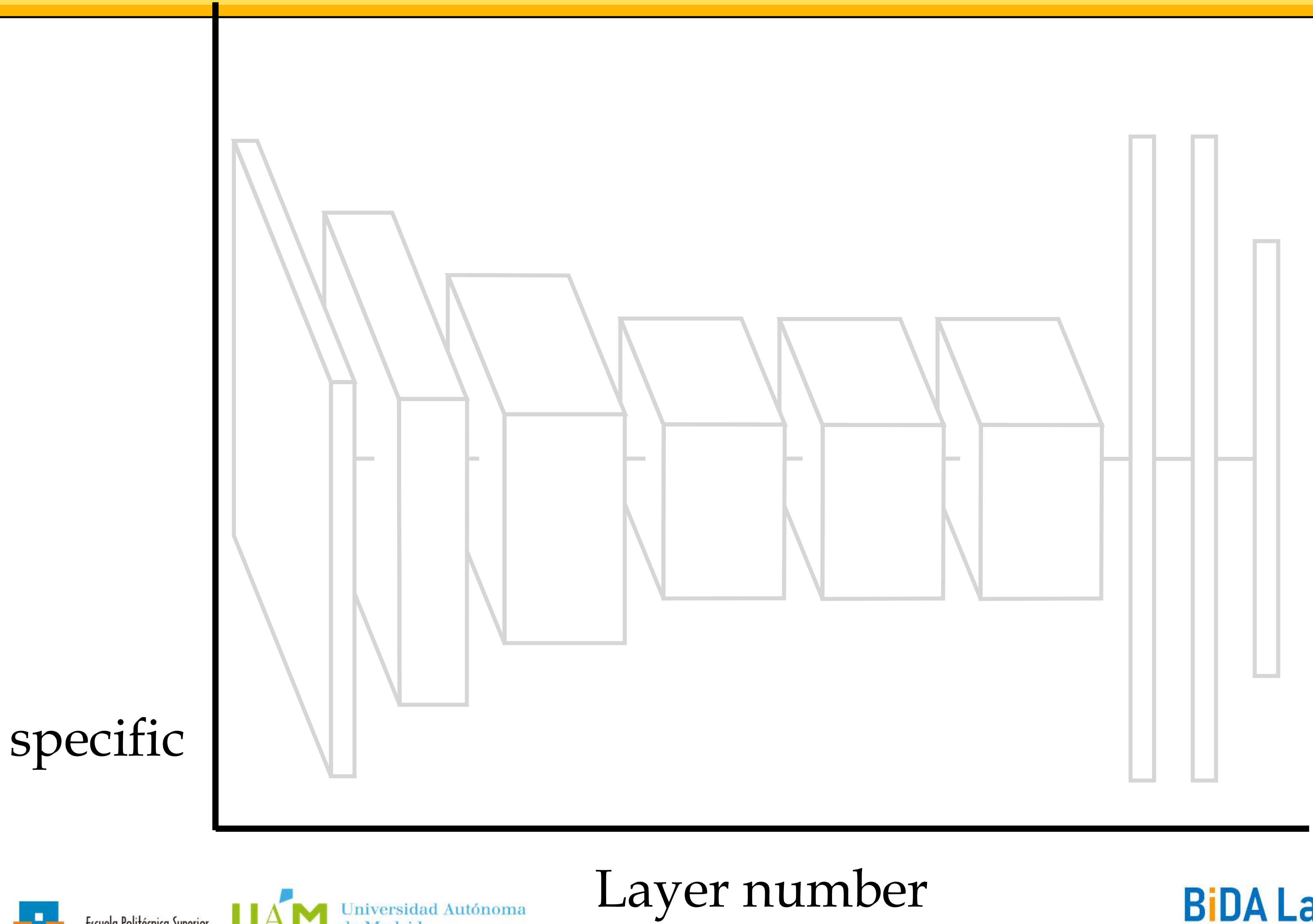


Zeiler et al.
arXiv 2013

Nguyen et al.
arXiv 2014

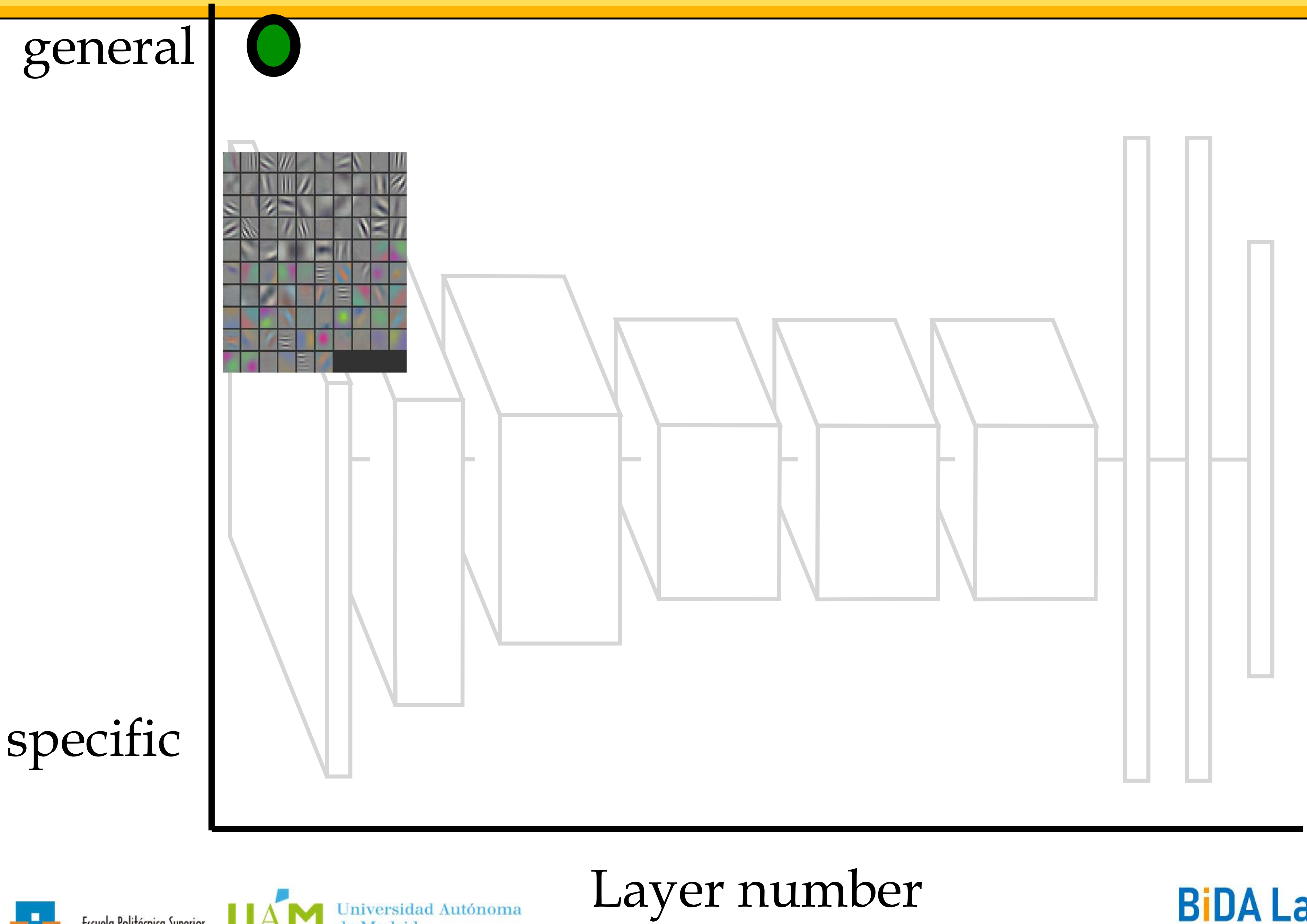
Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski



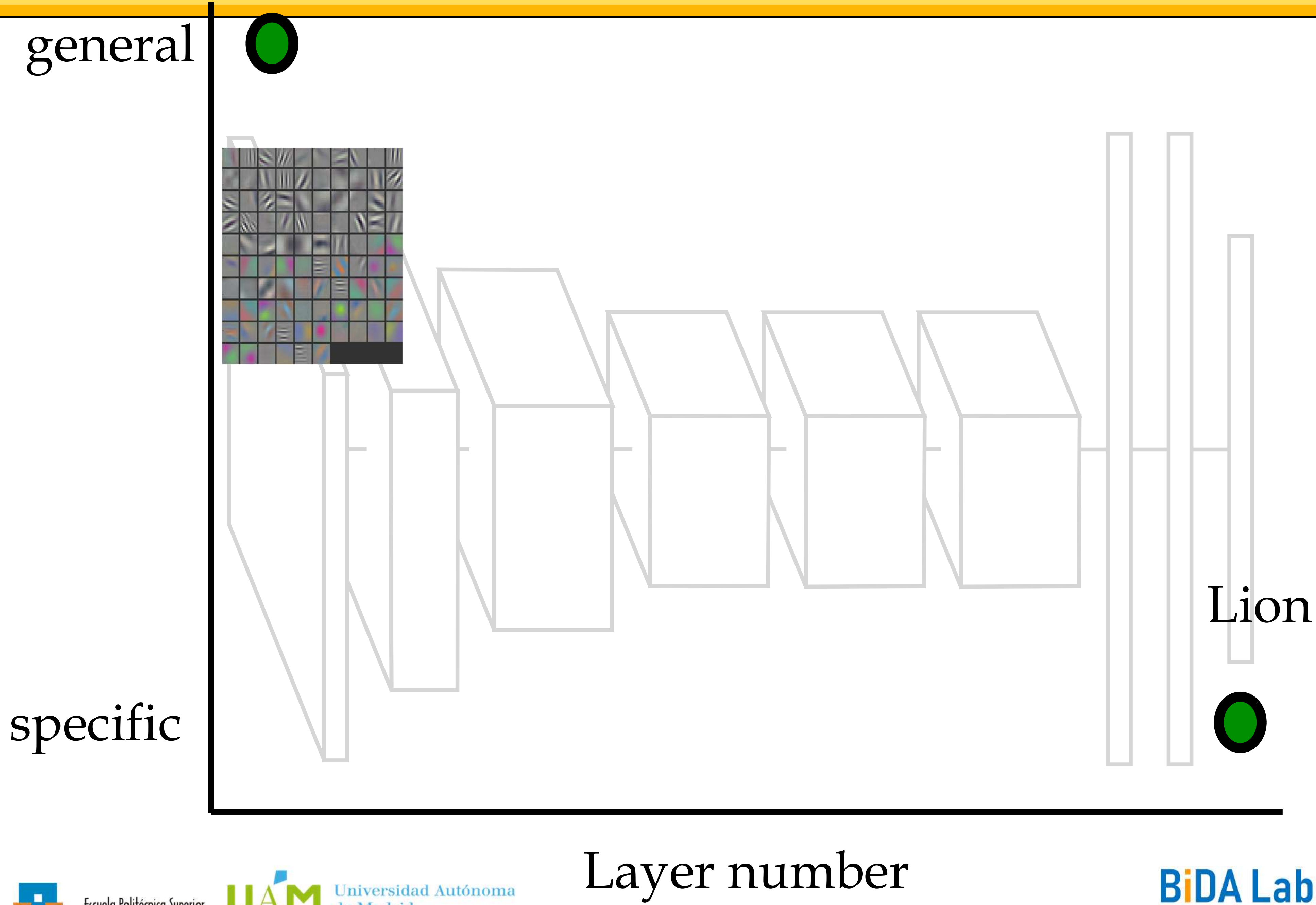
Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski



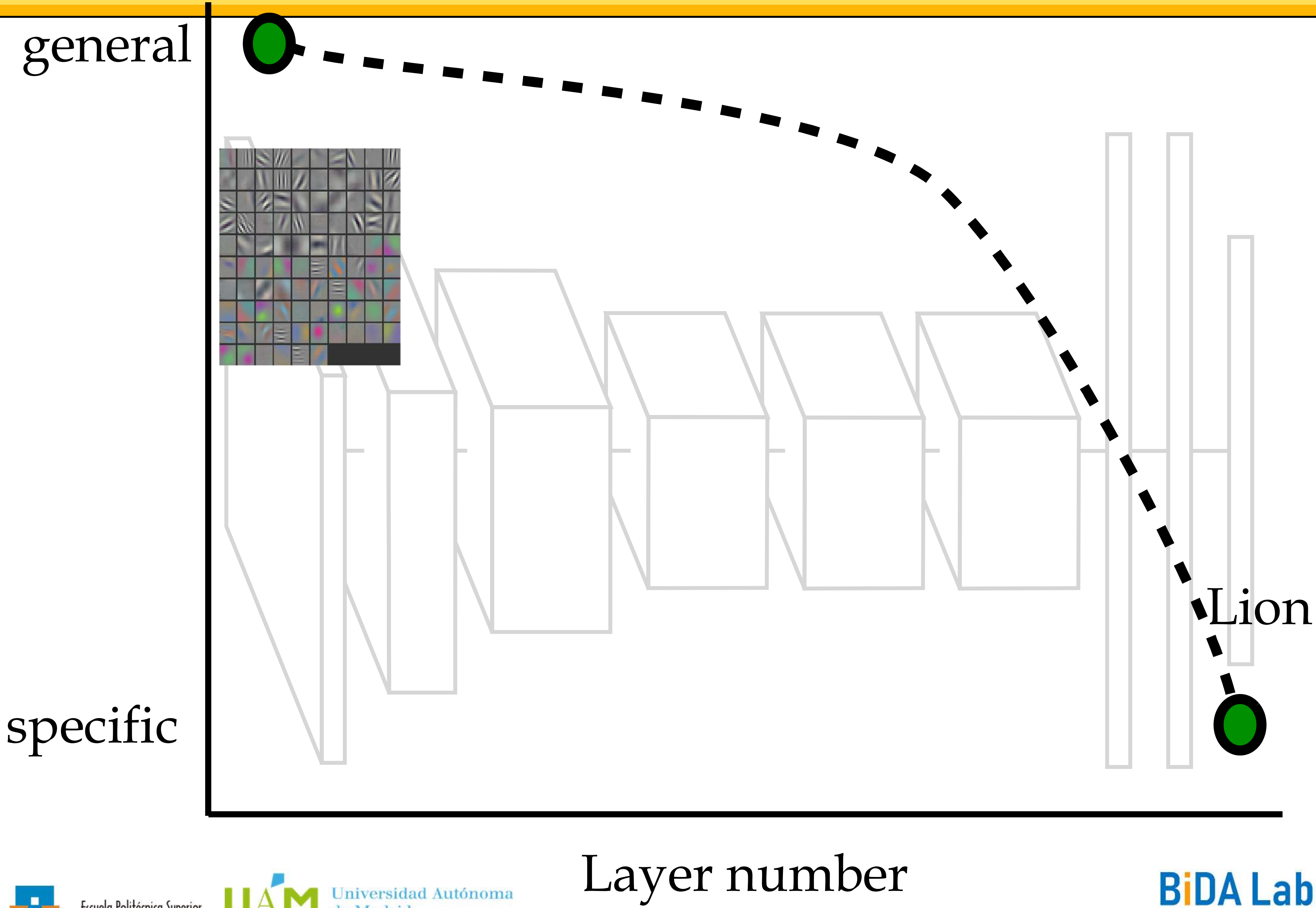
Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski



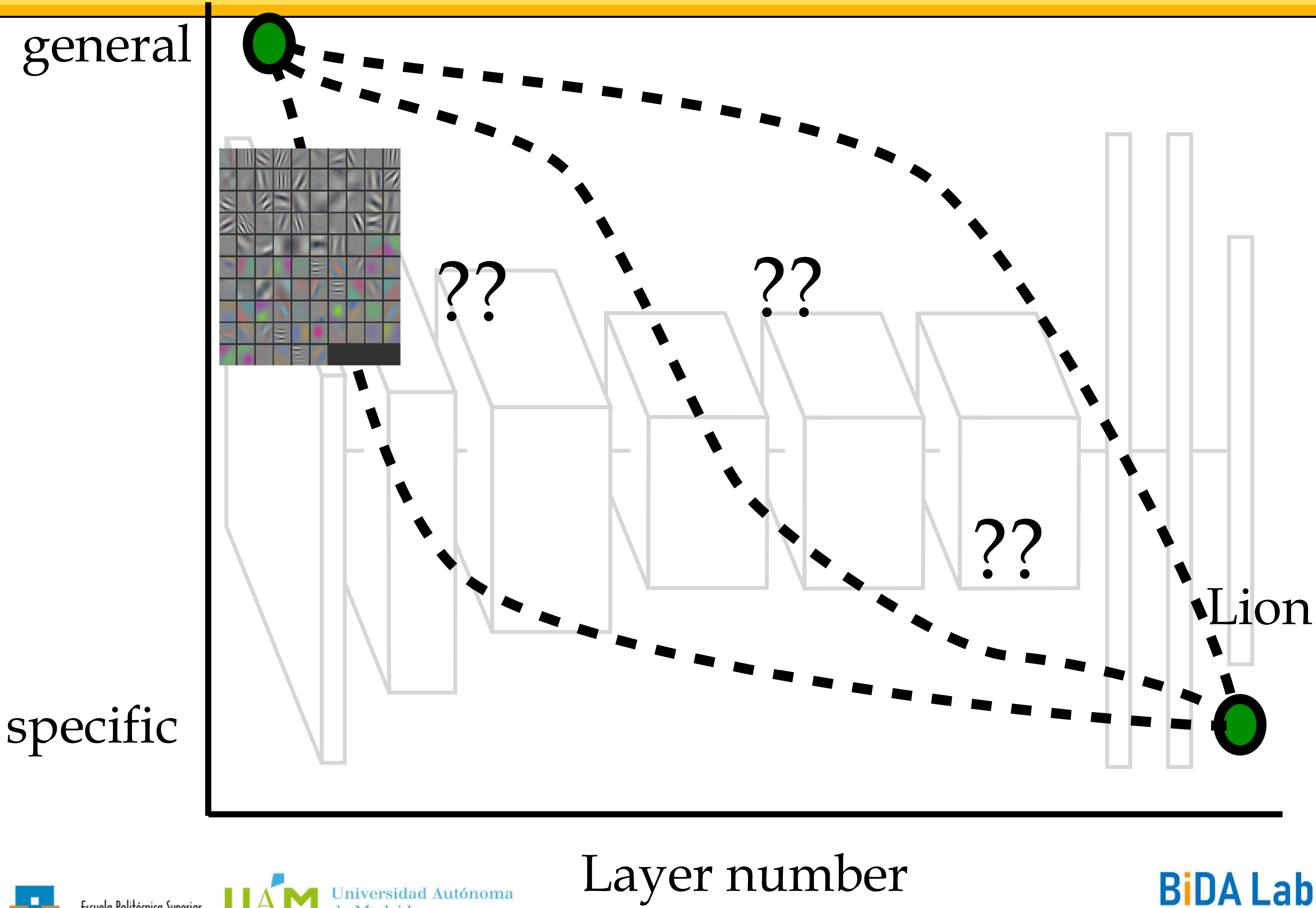
Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski

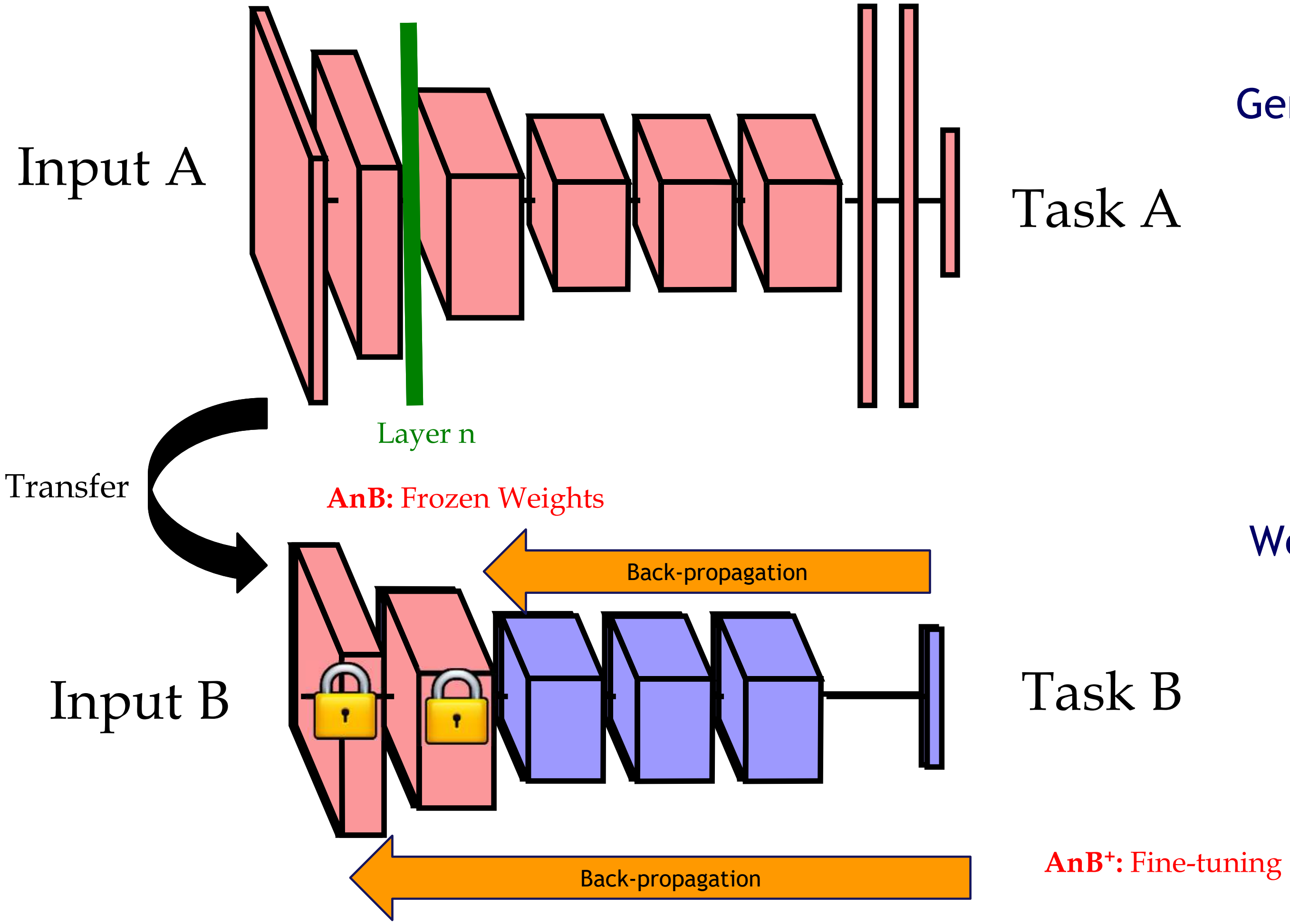


Convolutional Neural Networks: AlexNet

Slides from Jason Yosinski



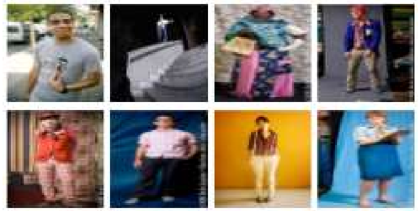
Transfer learning



General purpose pre-trained network

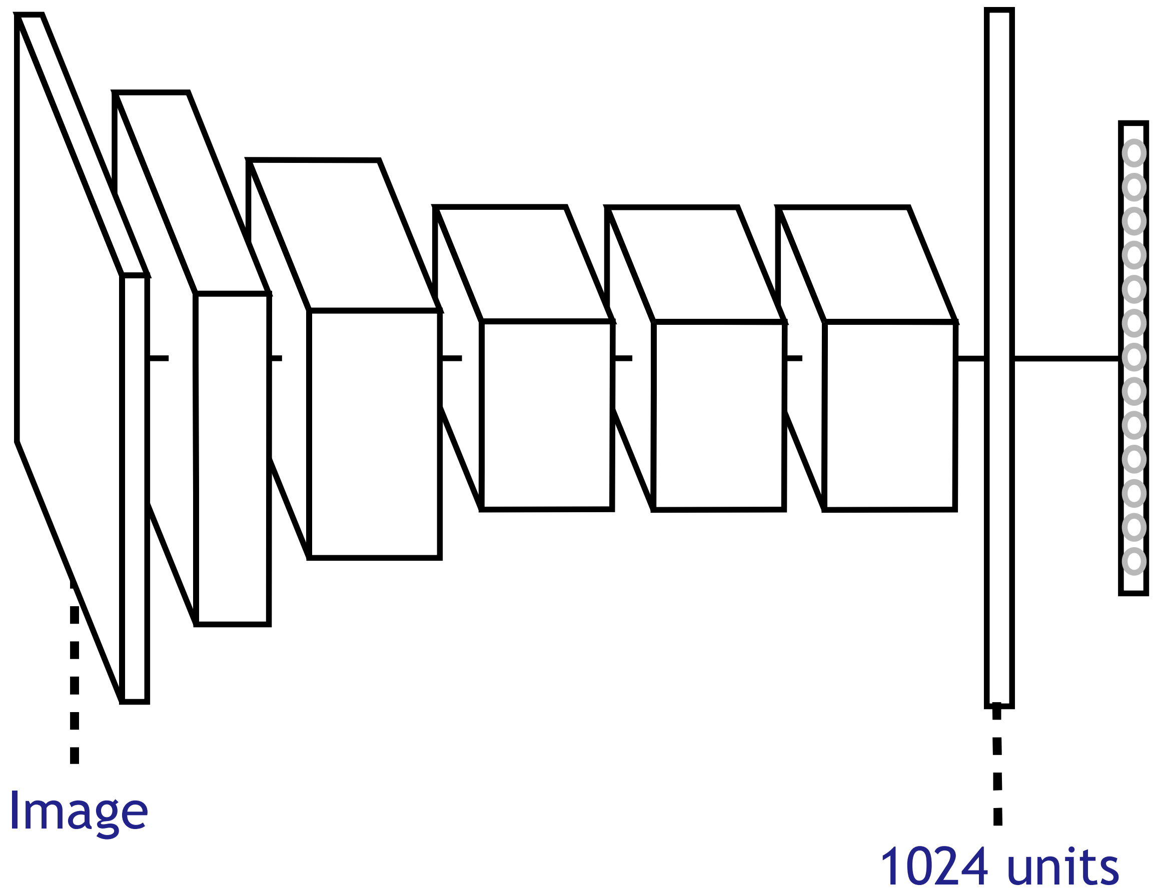
Inception from Google

Weights adapted to fashion images



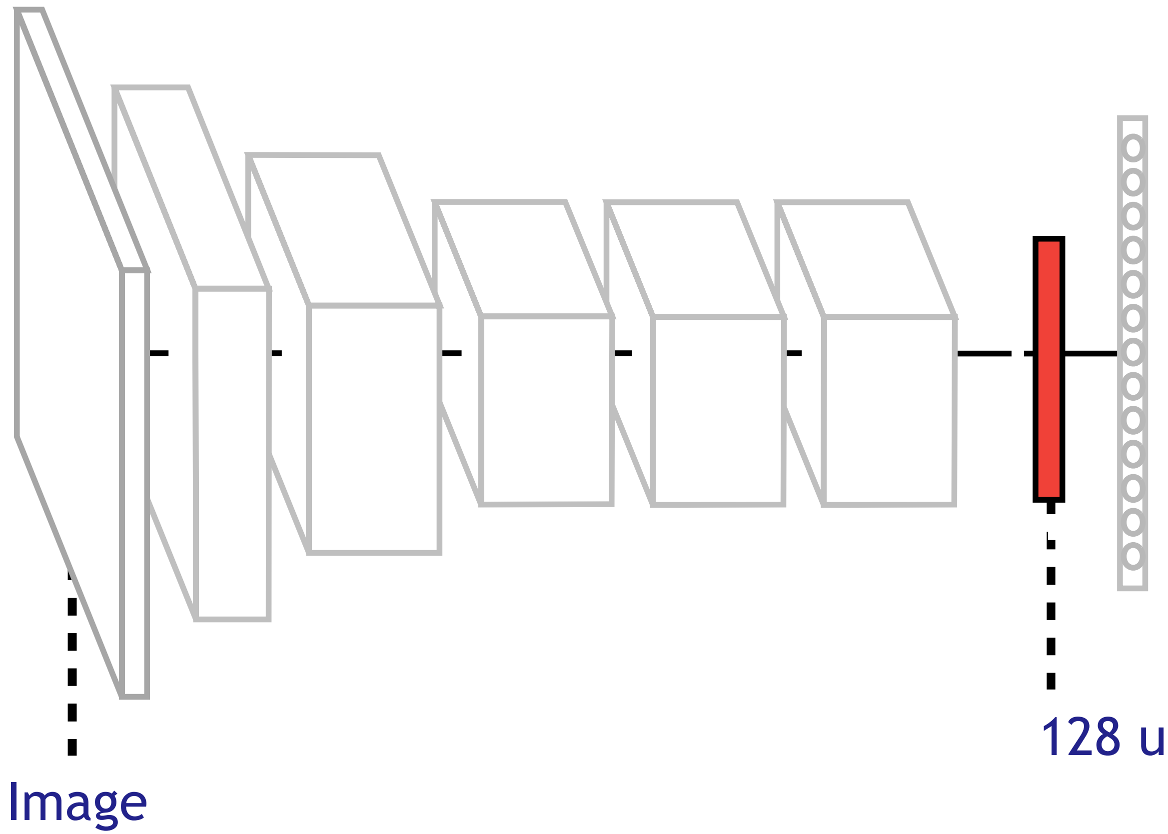
Fashion-net

Transfer learning: embeddings



Tuned network for fashion applications

Transfer learning: embeddings



$$\mathbf{f} = \begin{bmatrix} f^1 \\ f_2 \\ \vdots \\ f_{128} \end{bmatrix}$$

We can add a new layer with low number of units to compress the information with high specificity

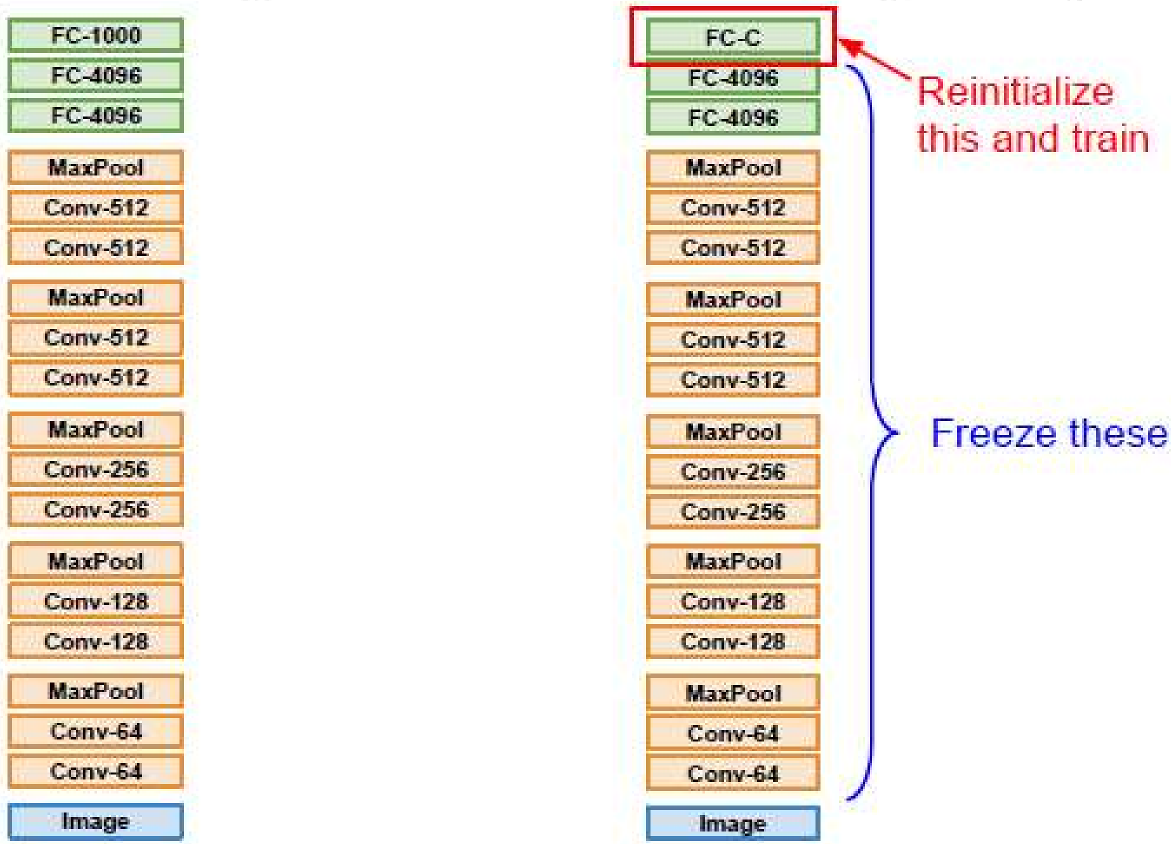
Transfer learning

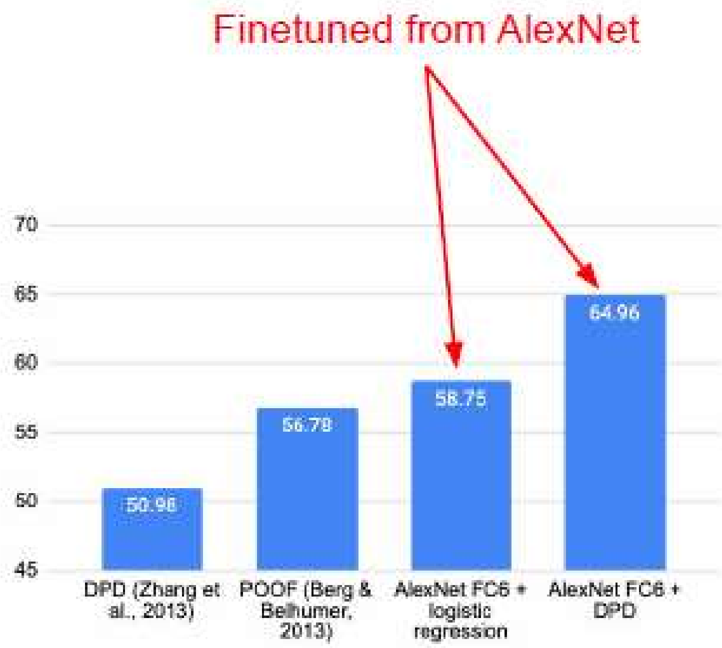
Transfer Learning with CNNs

Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014
Razavian et al, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition", CVPR Workshops 2014

1. Train on Imagenet

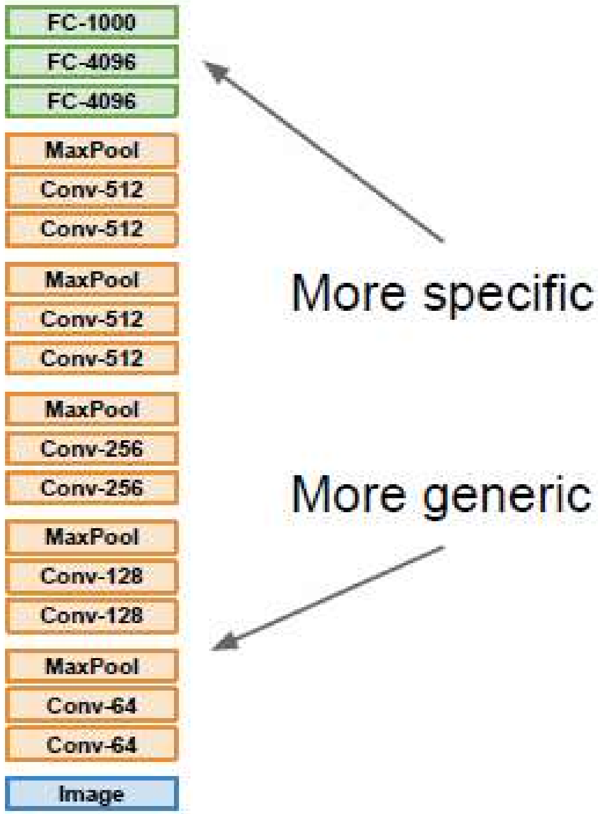
2. Small Dataset (C classes)





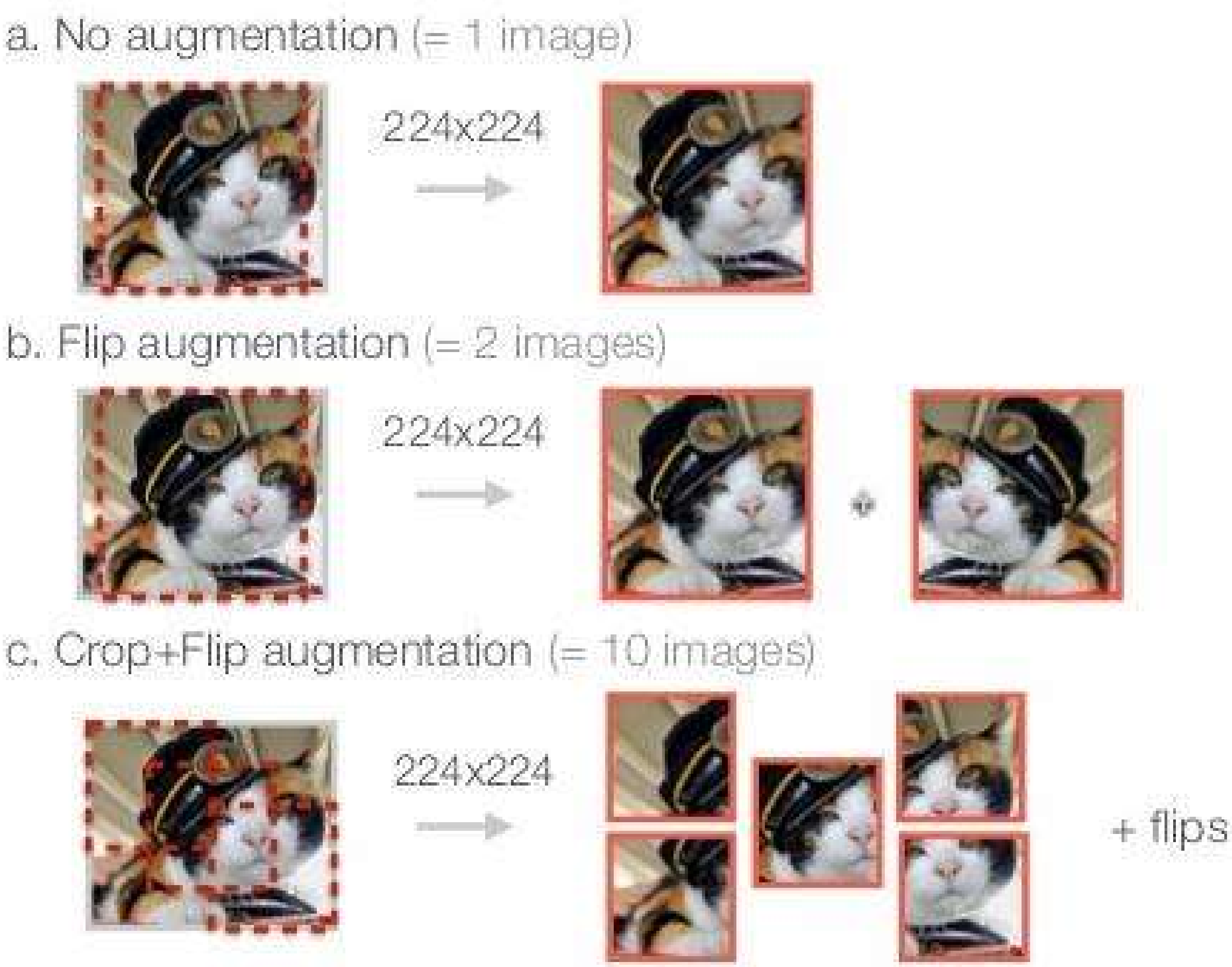
Donahue et al, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition", ICML 2014

Transfer learning

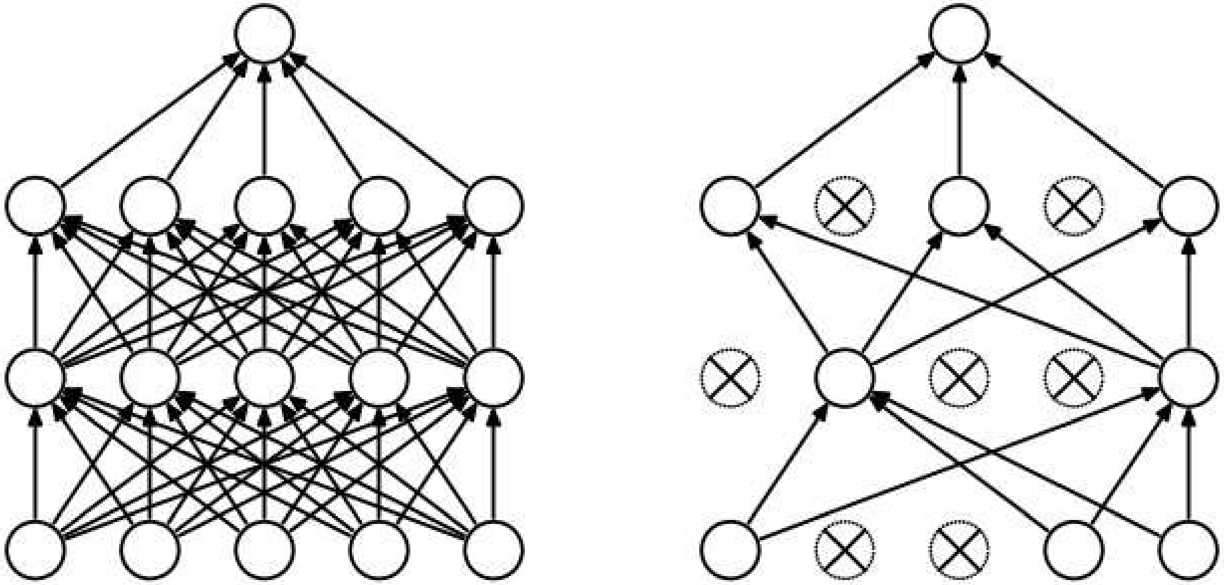


	very similar dataset	**very different dataset**
very little data	Finetune linear classifier on top layer	You're in trouble… Try data augmentation / collect more data
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Data Augmentation

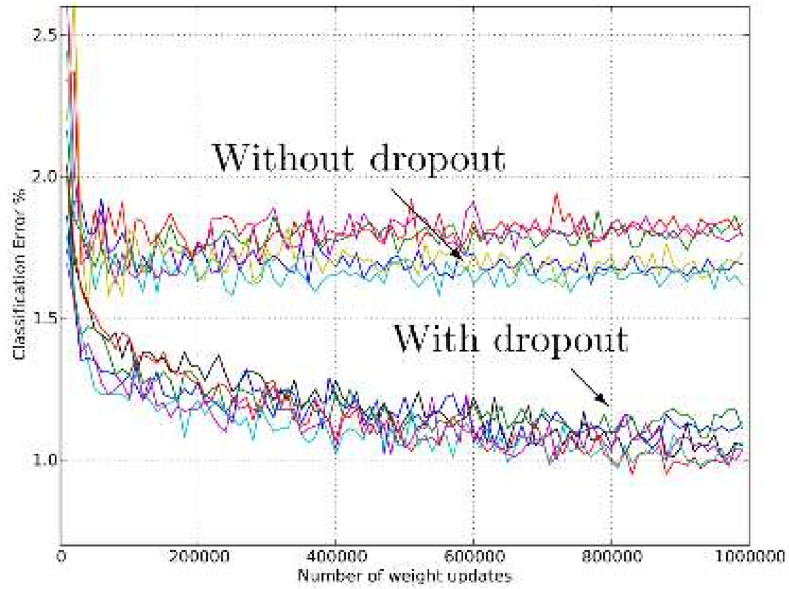


Dropout



(a) Standard Neural Net

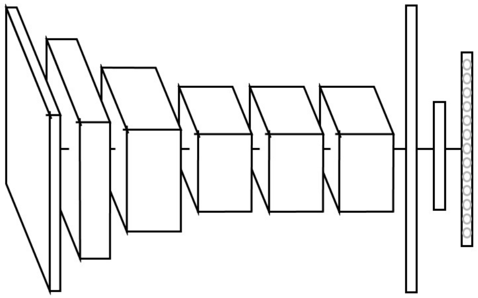
(b) After applying dropout.



Dropout: A Simple Way to Prevent Neural Networks from Overfitting
(N Srivastava et al. 2015)

How can we extract features from unstructured data?

Image Domain | Deep Learning algorithm | Feature Domain



$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{128} \end{bmatrix}$$

We can compare images in the feature domain

$$\mathbf{f}^1 = \begin{bmatrix} f_1^1 \\ f_2^1 \\ \vdots \\ f_{128}^1 \end{bmatrix}$$

$$\mathbf{f}^2 = \begin{bmatrix} f_1^2 \\ f_2^2 \\ \vdots \\ f_{128}^2 \end{bmatrix}$$

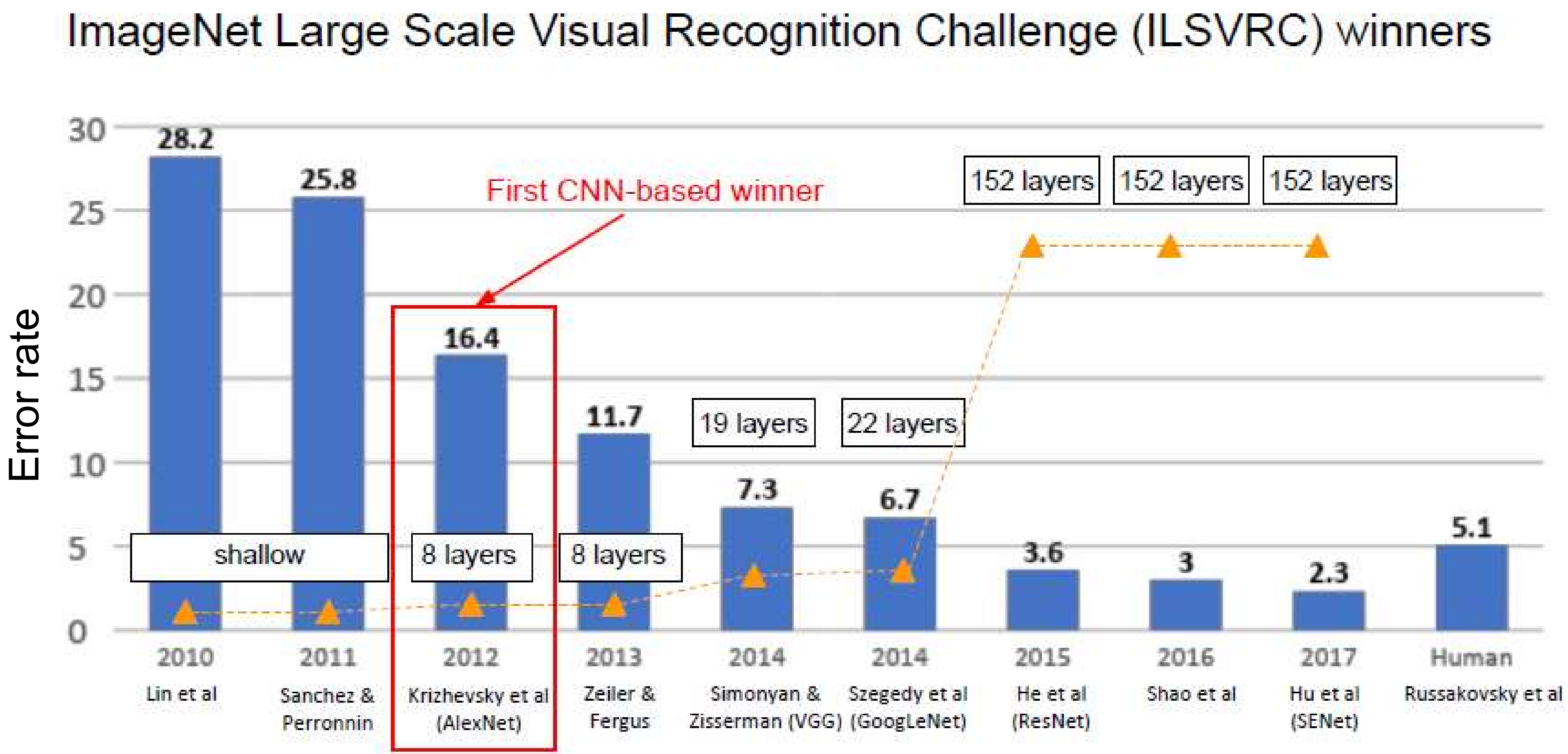
$$d(\mathbf{f}^1, \mathbf{f}^2) = 1{,}85$$

$$\mathbf{f}^1 = \begin{bmatrix} f_1^1 \\ f_2^1 \\ \vdots \\ f_{128}^1 \end{bmatrix}$$

$$\mathbf{f}^3 = \begin{bmatrix} f_1^3 \\ f_2^3 \\ \vdots \\ f_{128}^3 \end{bmatrix}$$

$$d(\mathbf{f}^1, \mathbf{f}^3) = 2{,}12$$

CNNs Evolution



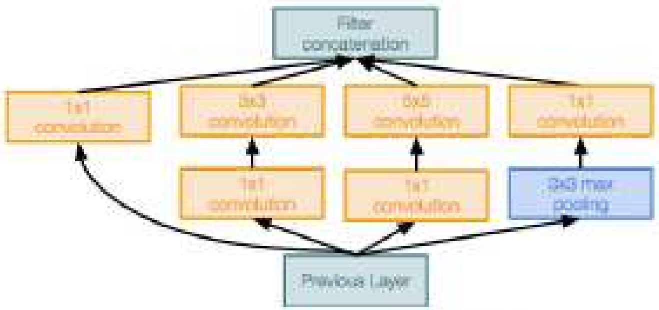
CNNs Evolution

Case Study: GoogLeNet

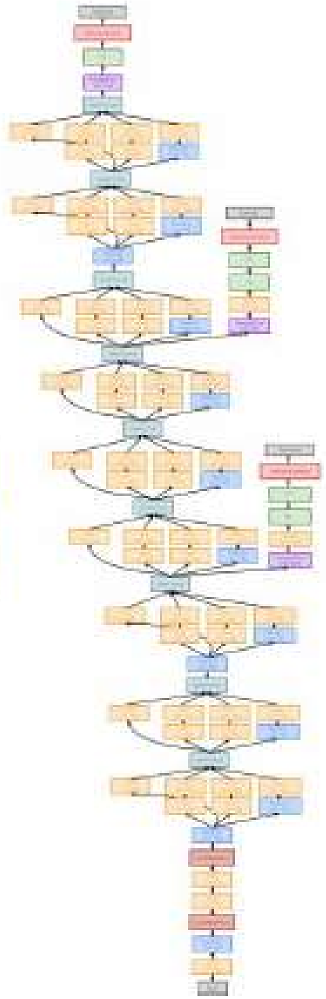
[Szegedy et al., 2014]

Deeper networks, with computational efficiency

- ILSVRC'14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient "Inception" module
- No FC layers



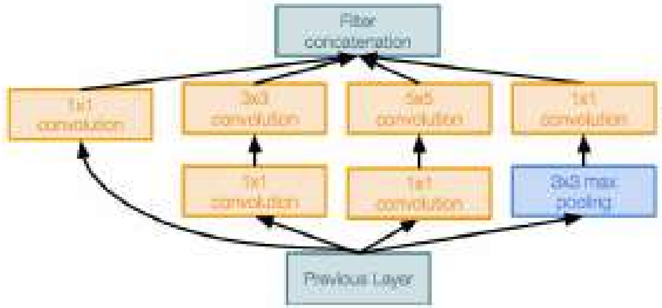
Inception module



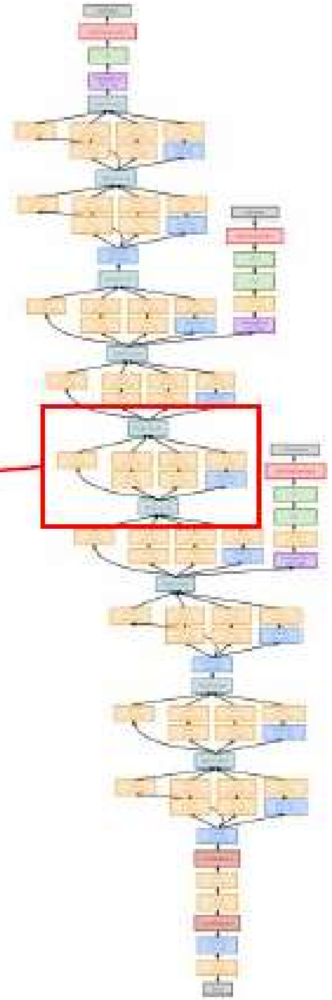
Case Study: GoogLeNet

[Szegedy et al., 2014]

"Inception module": design a good local network topology (network within a network) and then stack these modules on top of each other



Inception module



CNNs Evolution

Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!