Convolutional Neural Networks

Convolutional neural networks are deep artificial neural networks that are used primarily to -

- classify images (e.g. name what they see),
- cluster them by similarity (image search), and
- perform object recognition within scenes etc.
- They are algorithms that can identify faces, individuals, street signs, tumors, platypuses and many other aspects of visual data.

1) Convolution and Non-linearity Blocks First

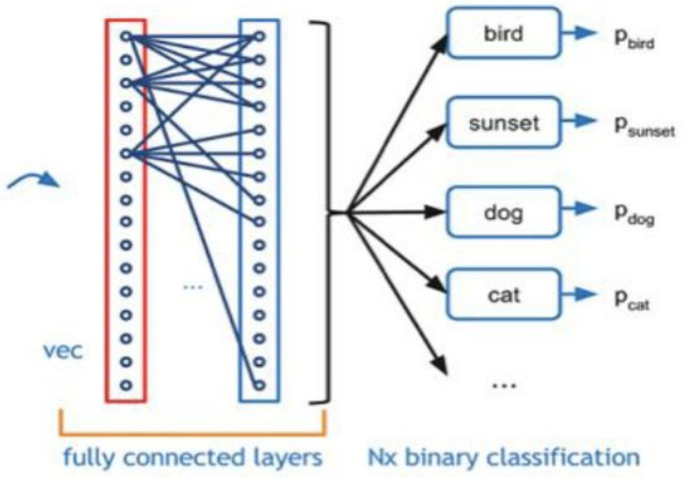
2) Then Fully Connected Layers leading to prediction

Image →* kernels → Feature Map (convolved Images) → ReLU (or other non-linearity) → Max Pooling

Convolution w/ kernels

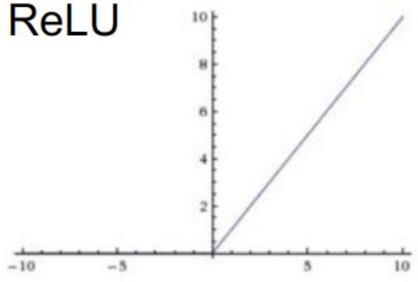
- ReLU
- Tanh
- Sigmoid

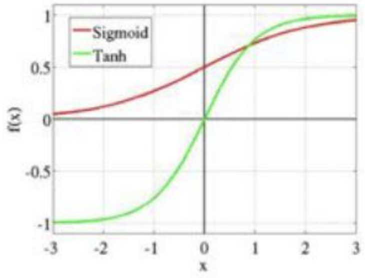
Down-sampling



Can repeat in multiple layers

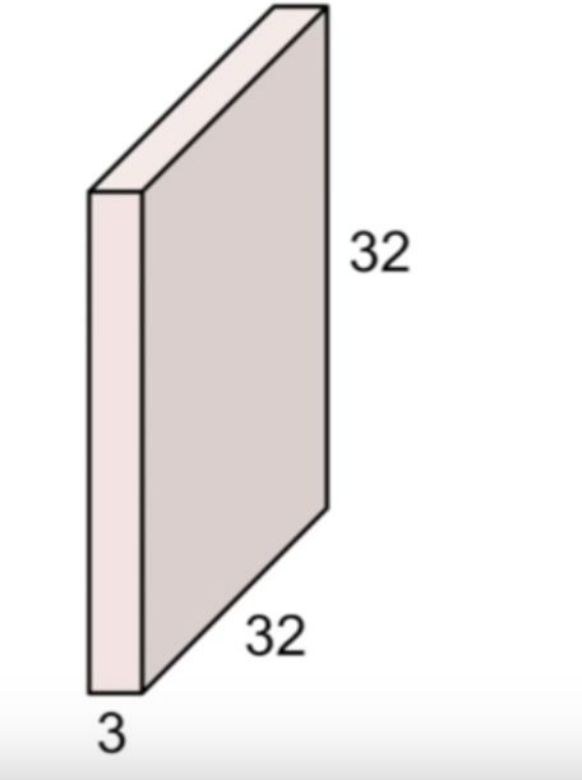
ReLU





Convolution Layer

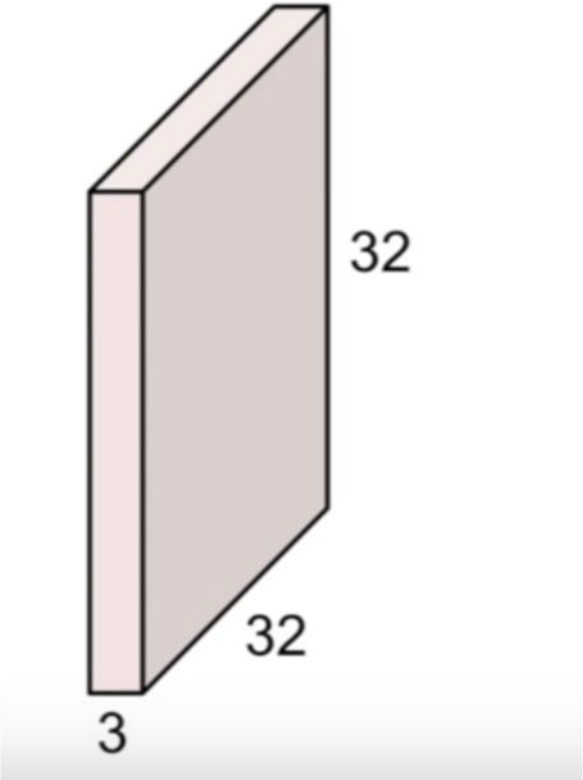
32x32x3 image -> preserve spatial structure



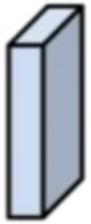
Instead of stretching the image into one long vector we are now going to keep the structure of the three dimensional input.

Convolution Layer

32x32x3 image



5x5x3 filter

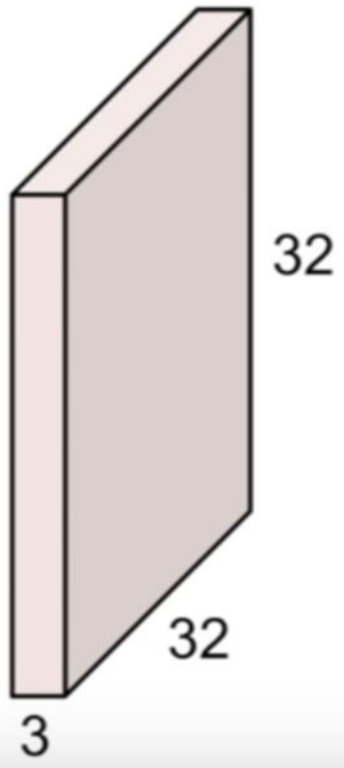


Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

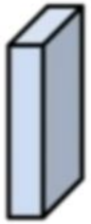
Convolution Layer

Filters always extend the full depth of the input volume

32x32x3 image



5x5x3 filter

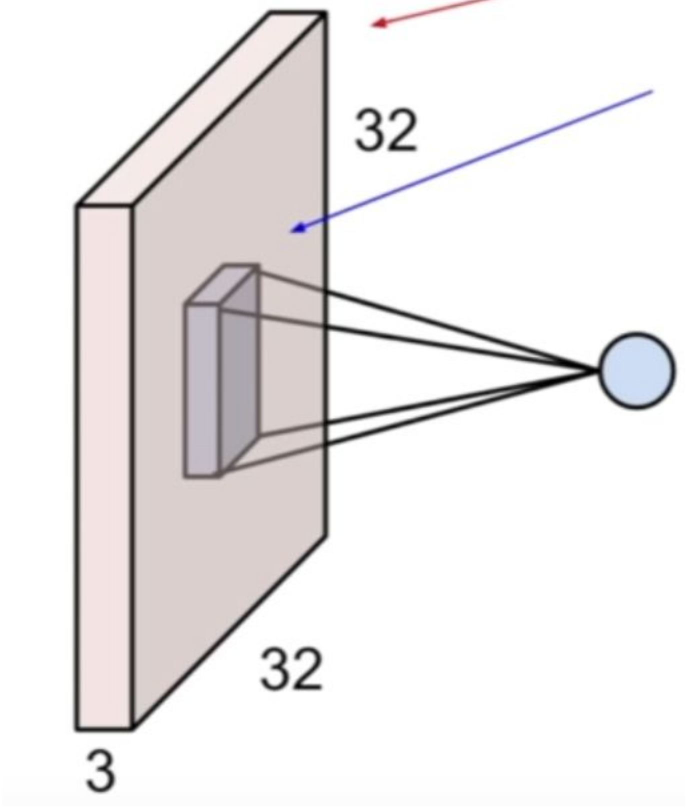


Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Convolution Layer

32x32x3 image

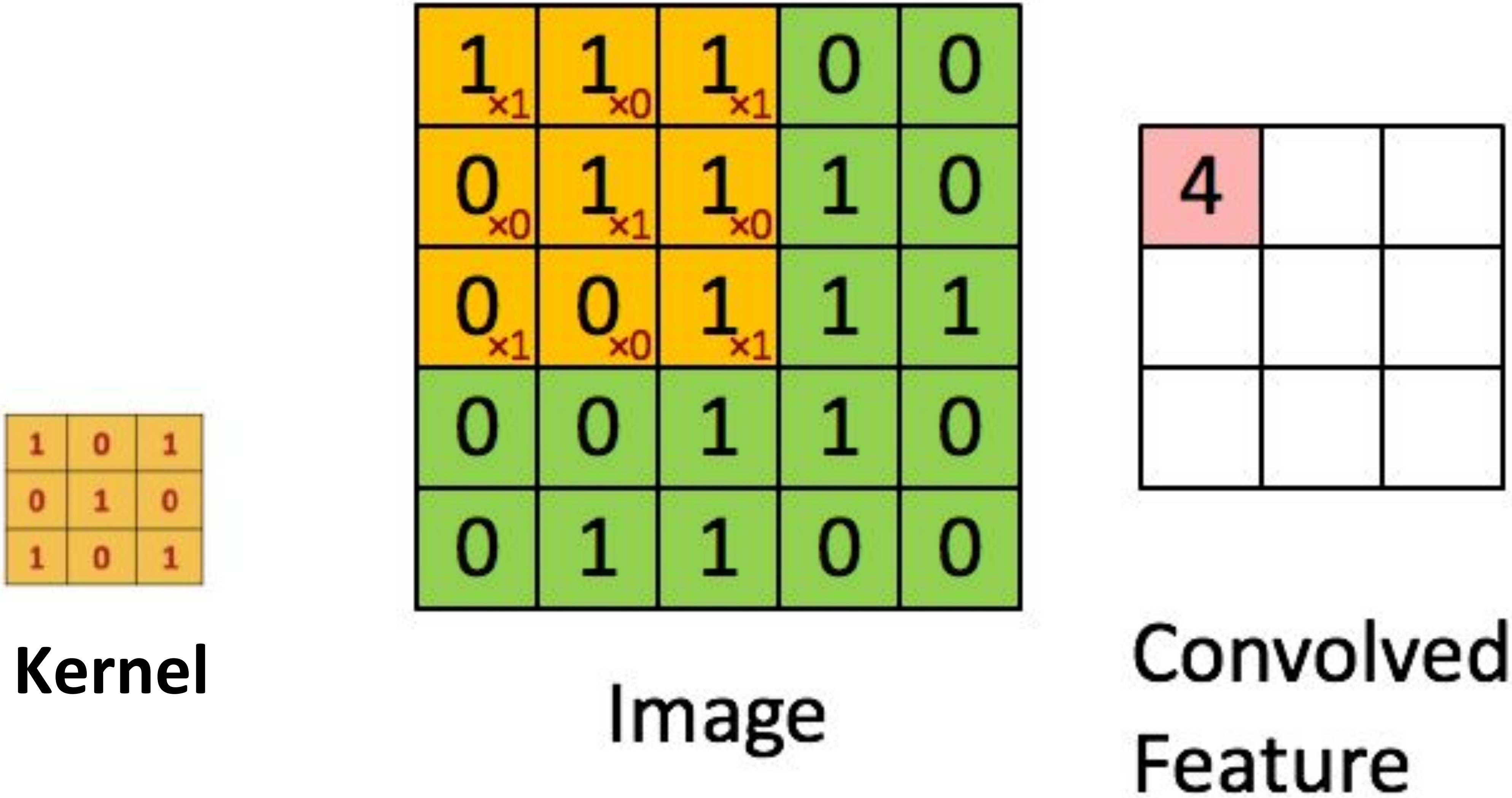
5x5x3 filter w



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)

$$w^T x + b$$

Convolution Layer


Kernel
Image
Convolved Feature

Before We Go On

X is an image

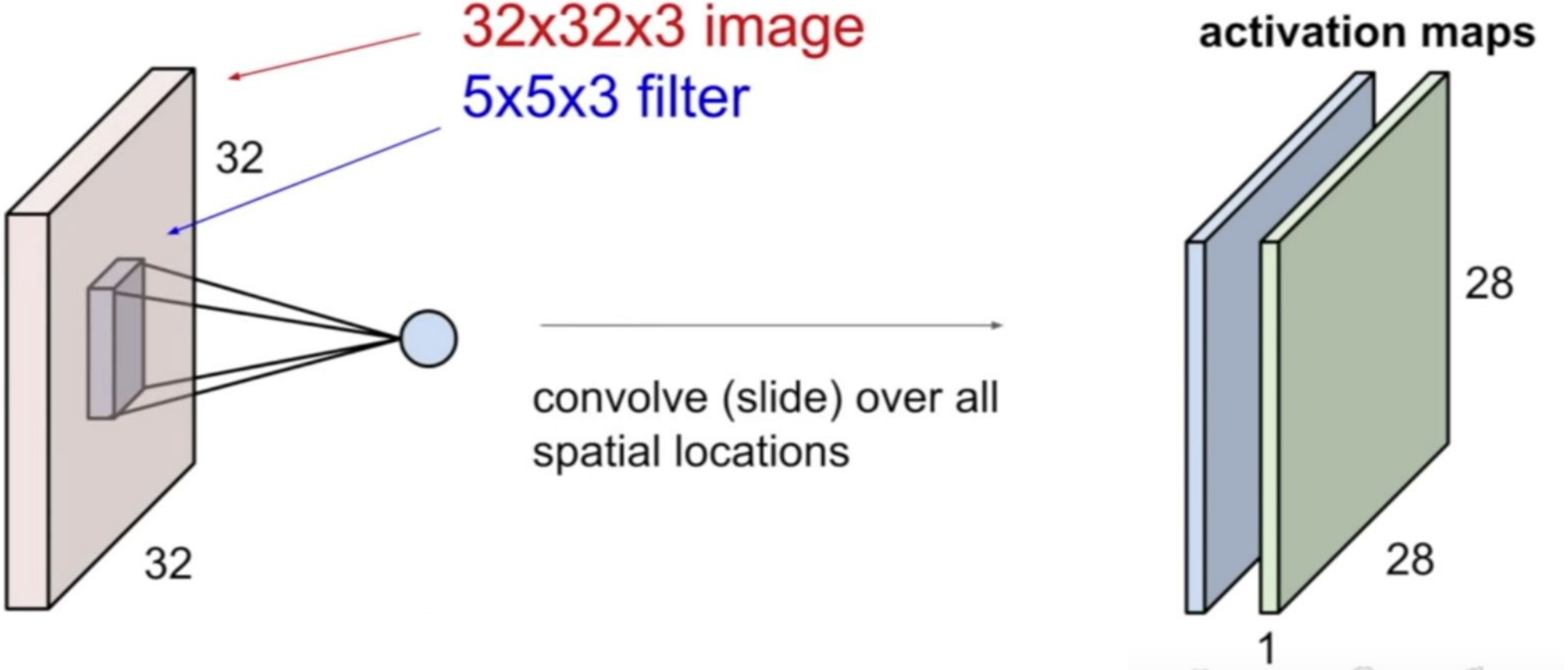
h is filter (or kernal)

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

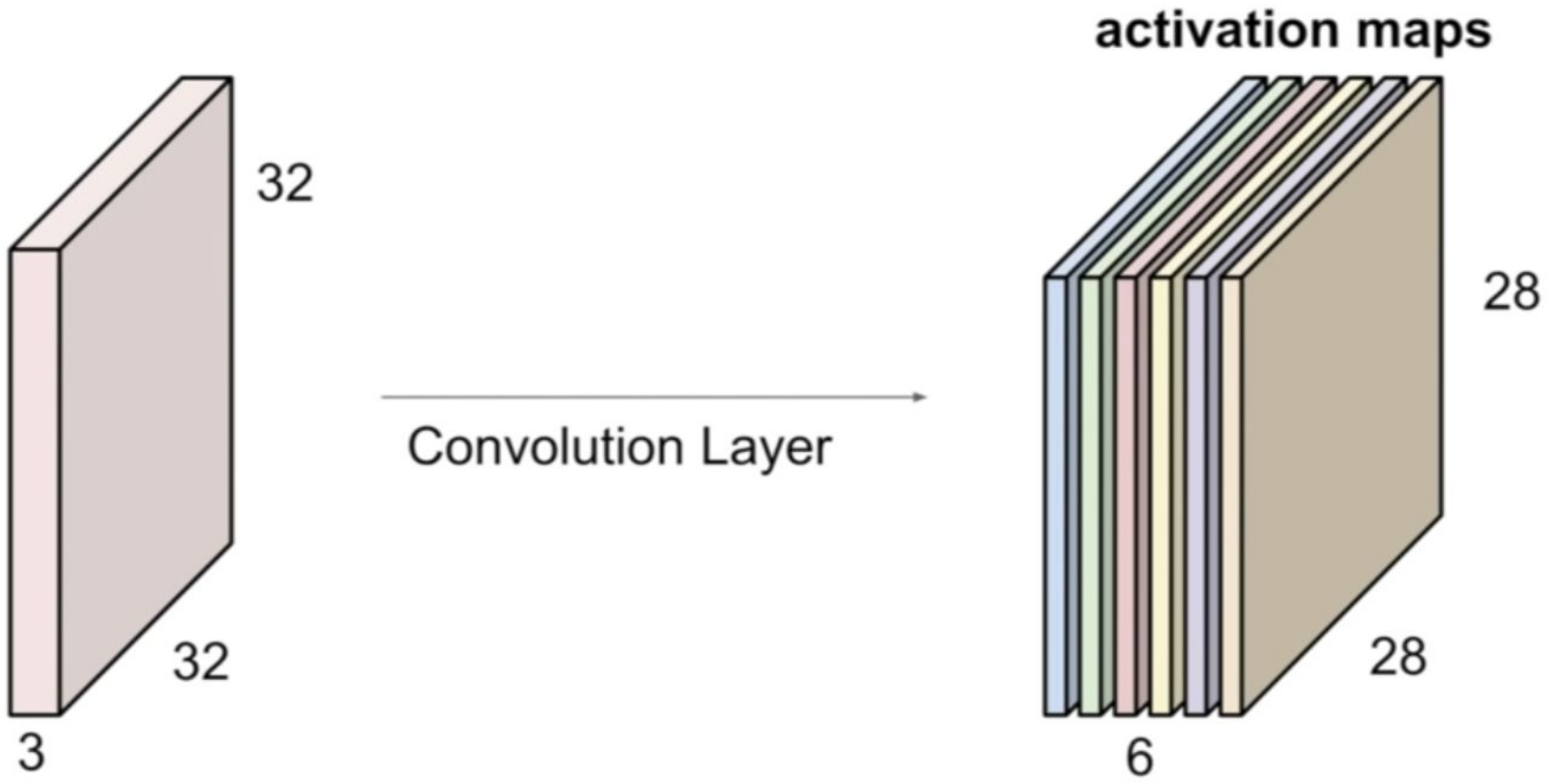
What is X * h?

Convolution Layer

consider a second, green filter

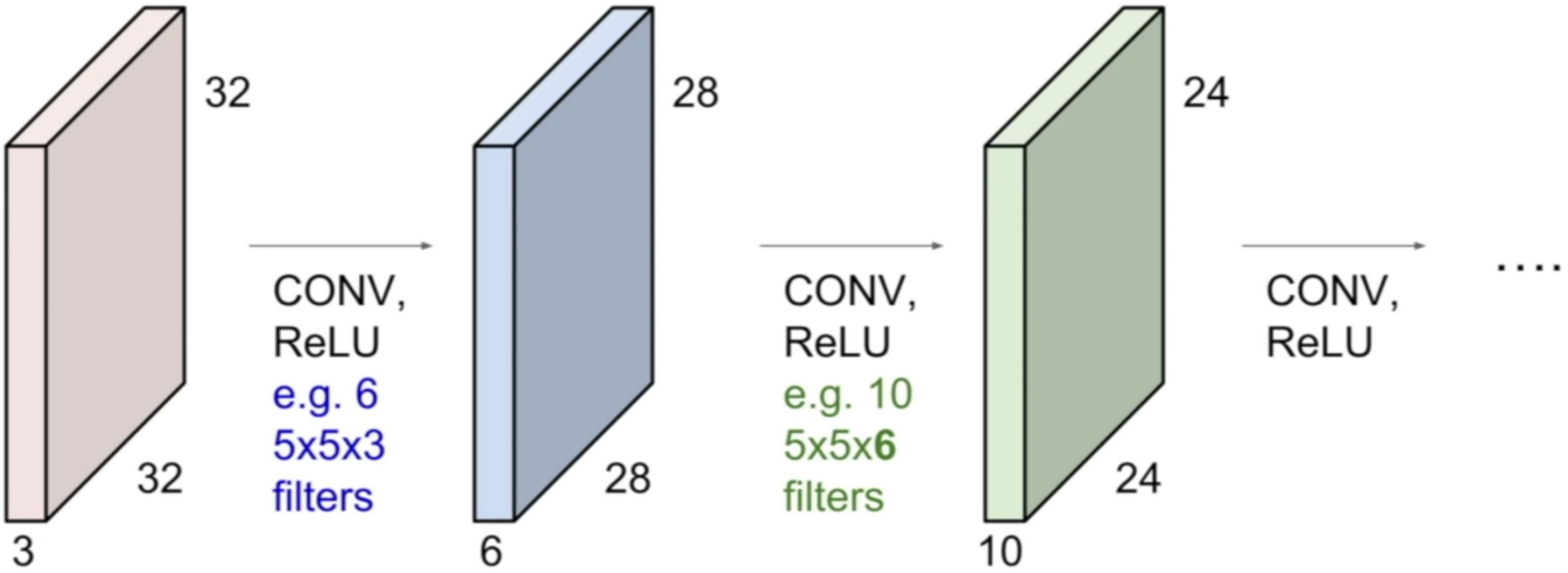


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

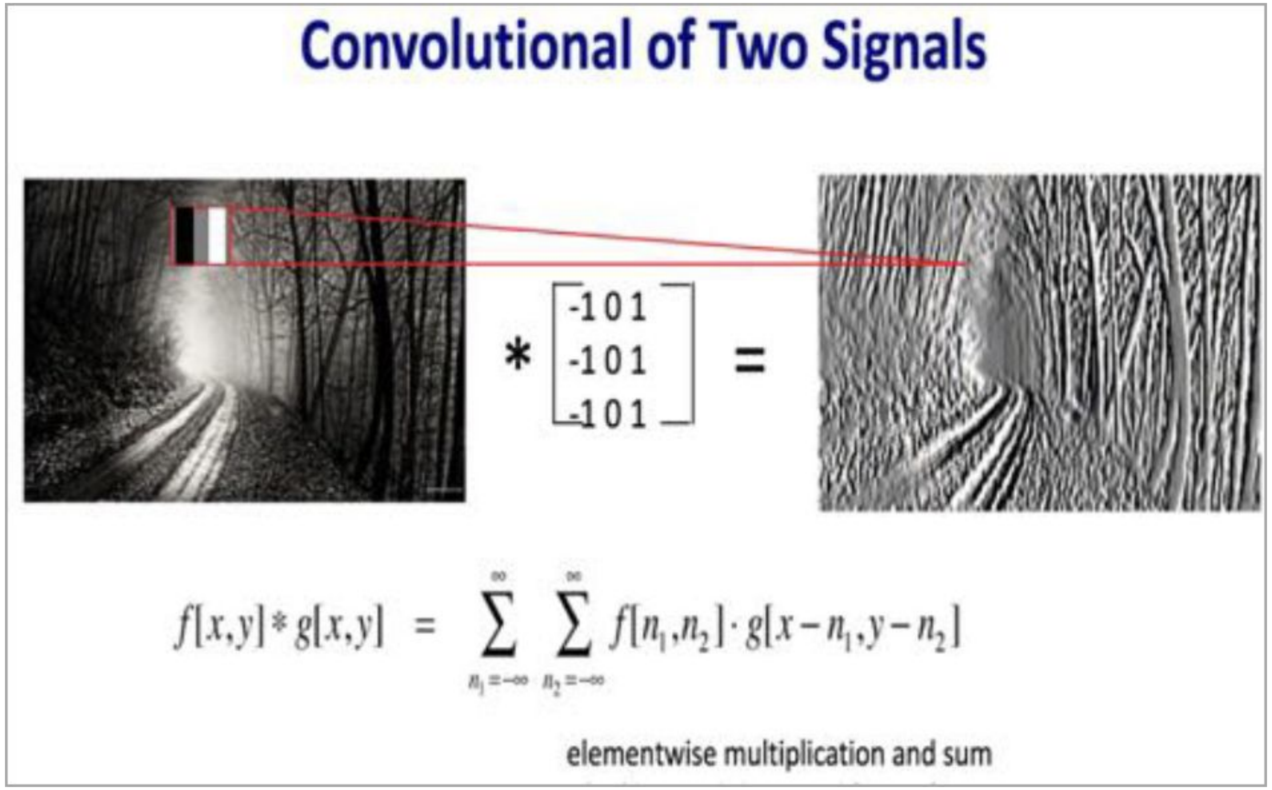


We stack them up and get a "New Image" of size 28 x 28 x 6

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



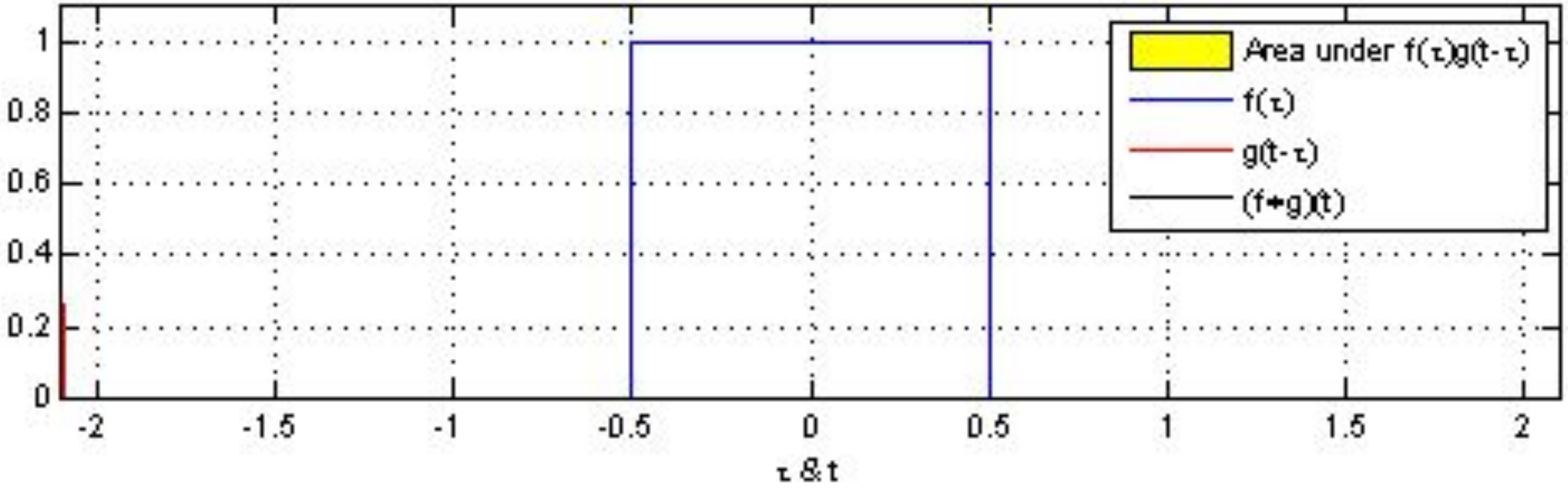
Using different types of filters affects the information extracted from an image- blur, edges, texture.



The picture represents what the Filter/Kernel was looking for

Input

What is convolution?



In mathematics (in particular, functional analysis) **convolution** is a mathematical operation on two functions (f and g) to produce a third function that expresses how the shape of one is modified by the other.

Source - wikipedia

Different Filter Types

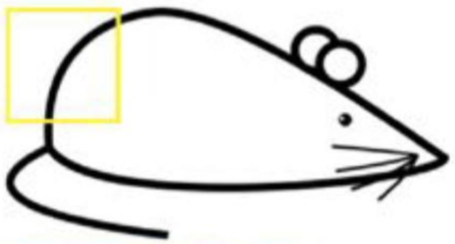
These filter are more typical for image processing,
 but not feature extraction.

In CNNs, we are looking to extract features

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Feature Extraction

Original image



Visualization of the filter on the image

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

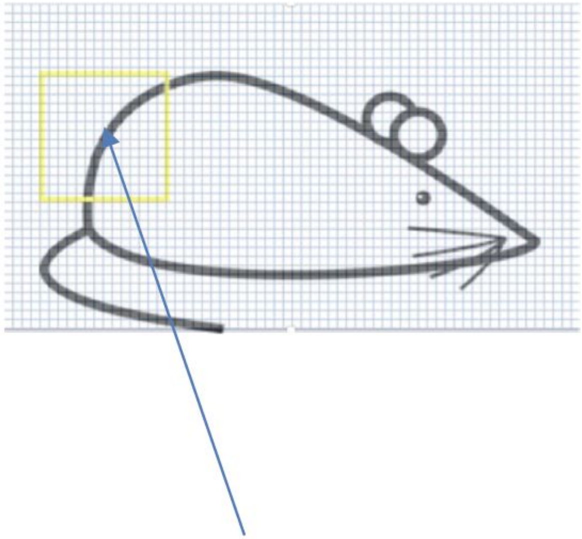
Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

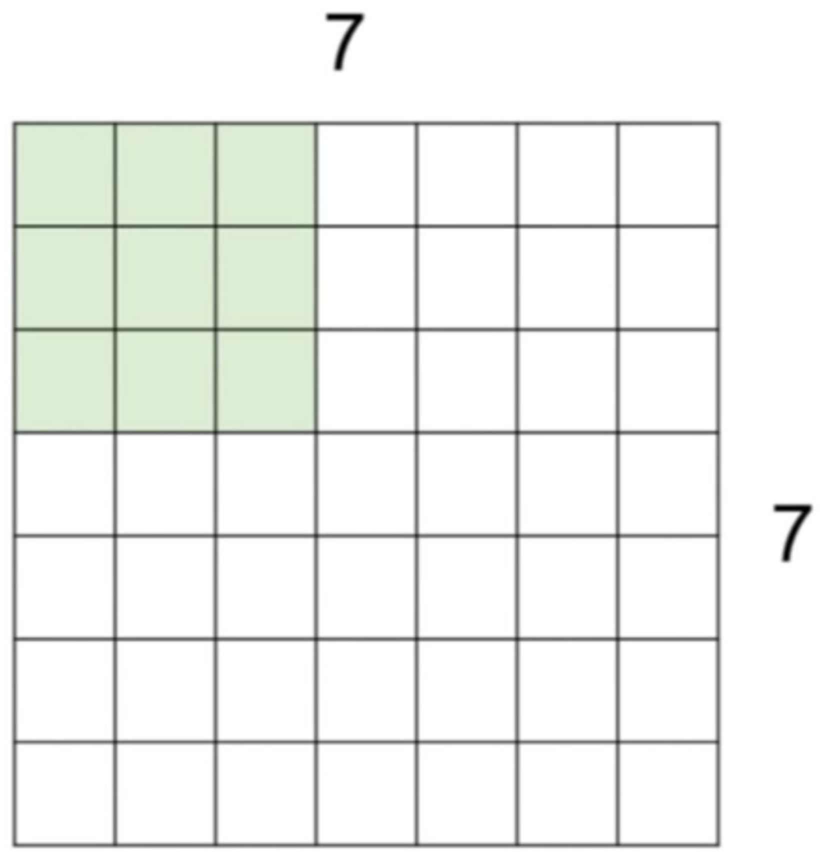


Value of convolution at this spot
is large because the sum of products

Think of this like a "moving dot product" of the kernel across the original image

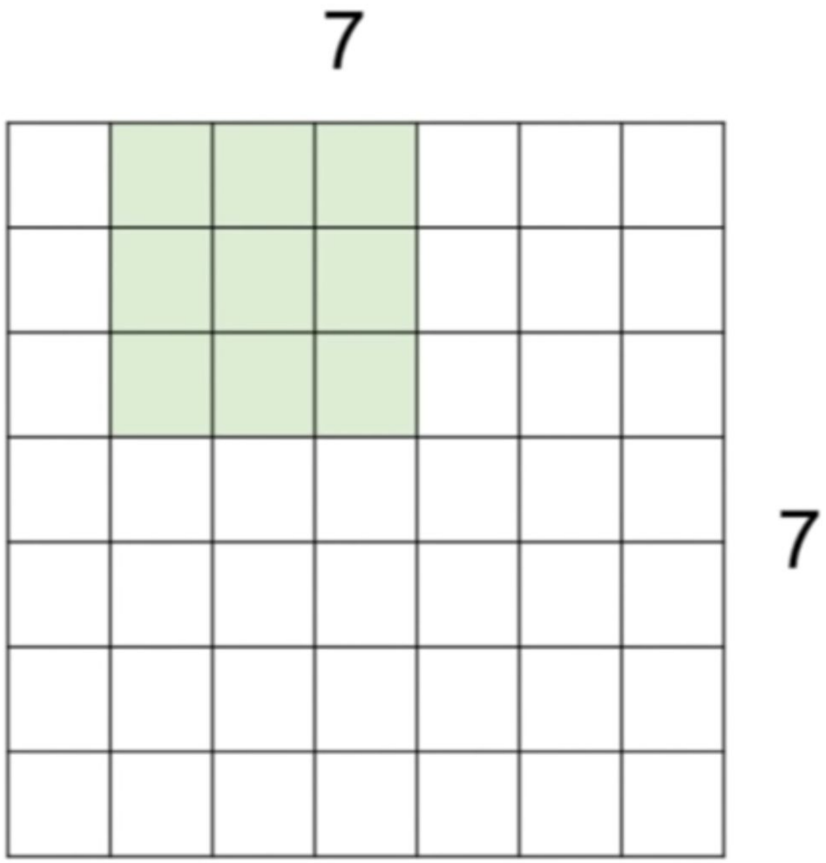
Spatial Dimensions:

Spatial Dimensions:



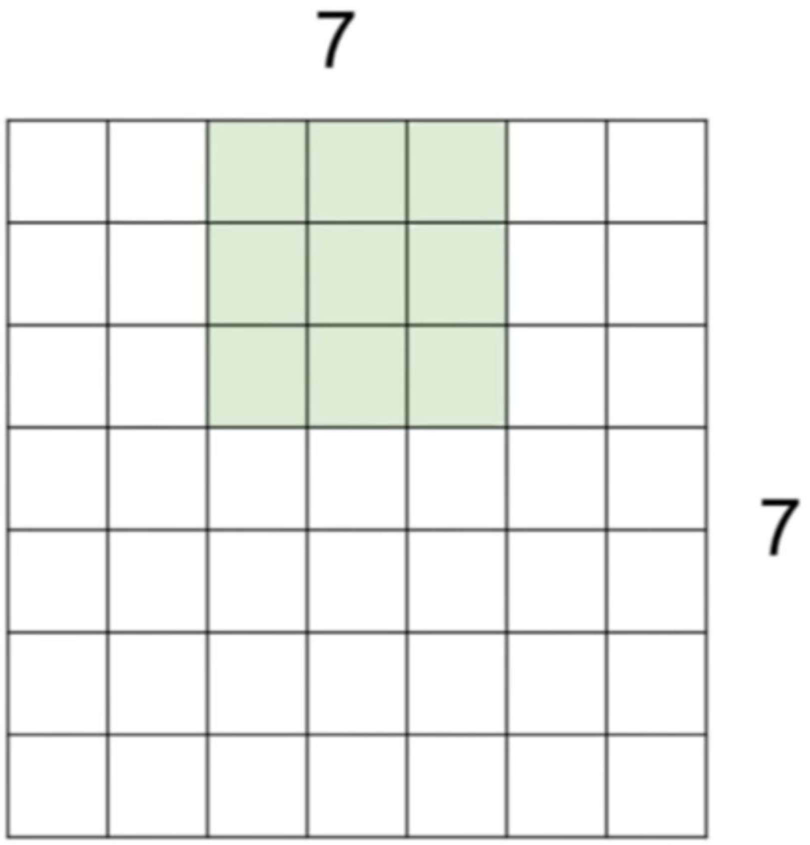
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions:



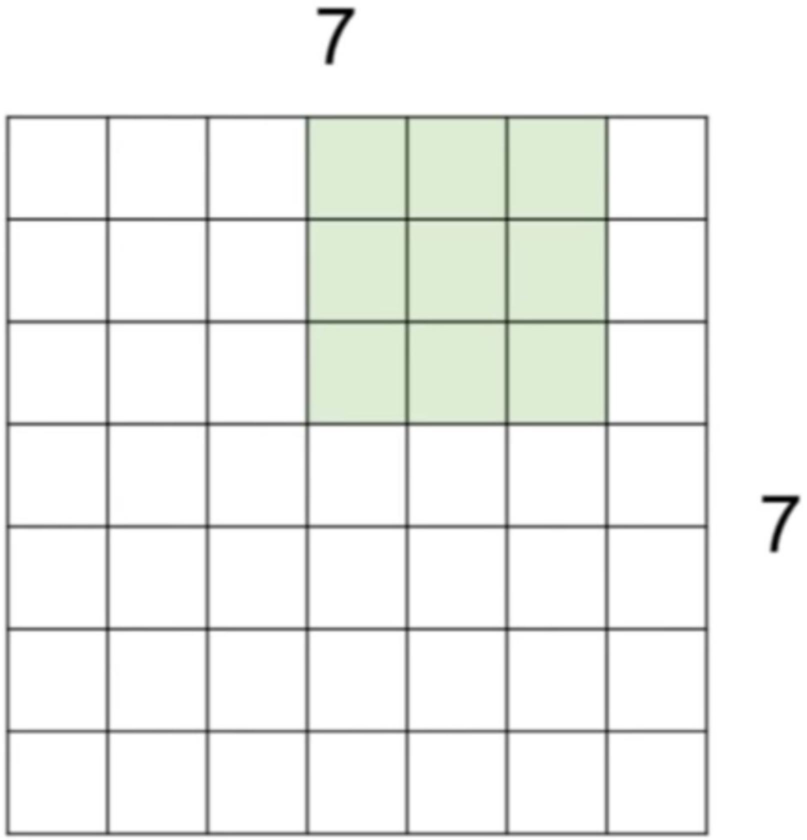
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions



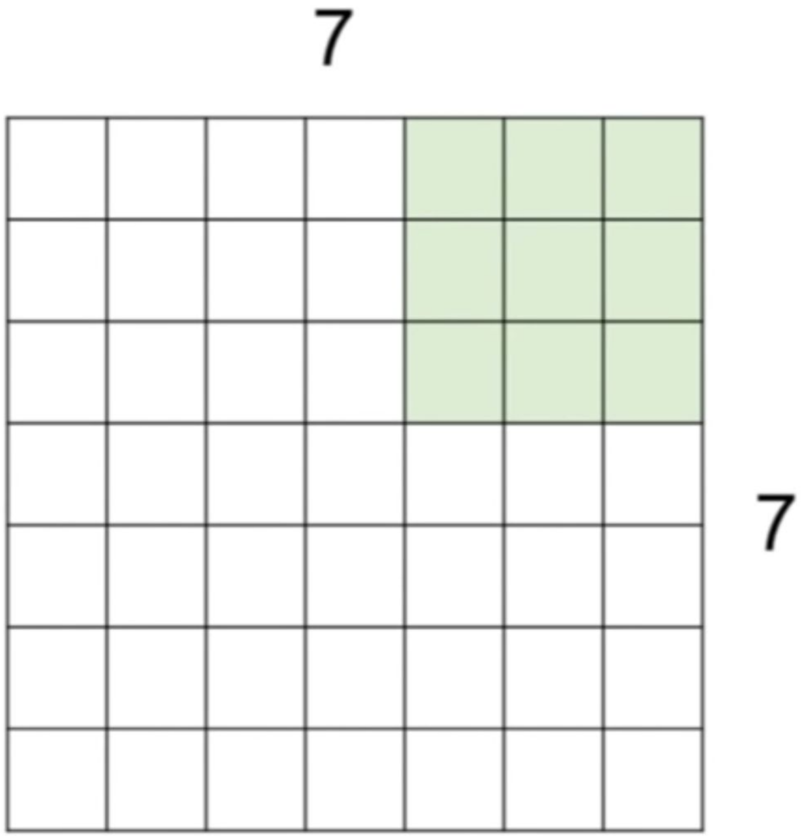
7x7 input (spatially)
assume 3x3 filter

Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter

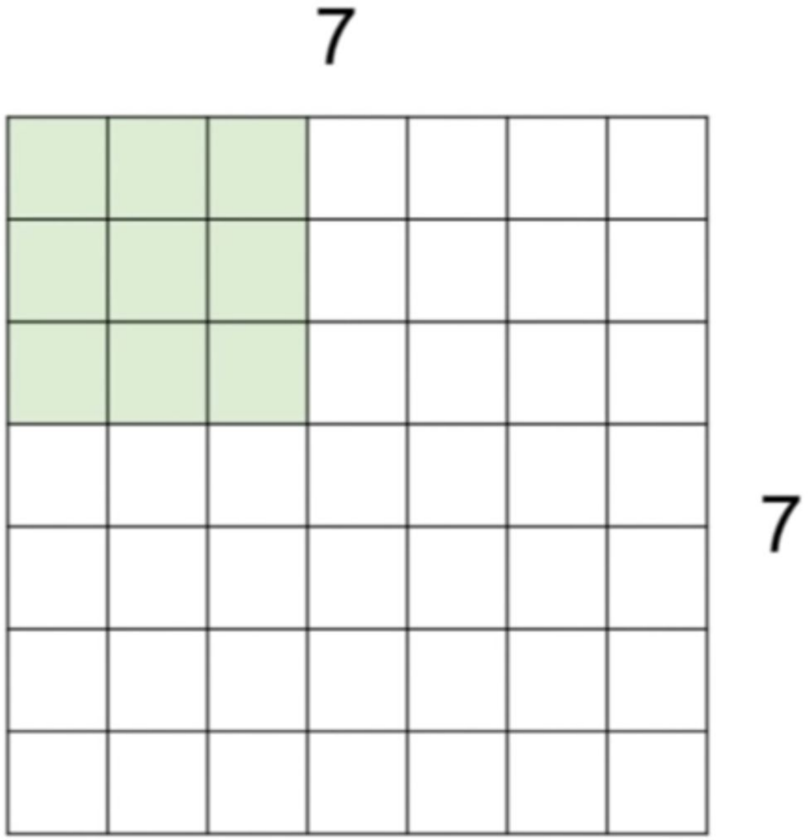
Spatial Dimensions



7x7 input (spatially)
assume 3x3 filter

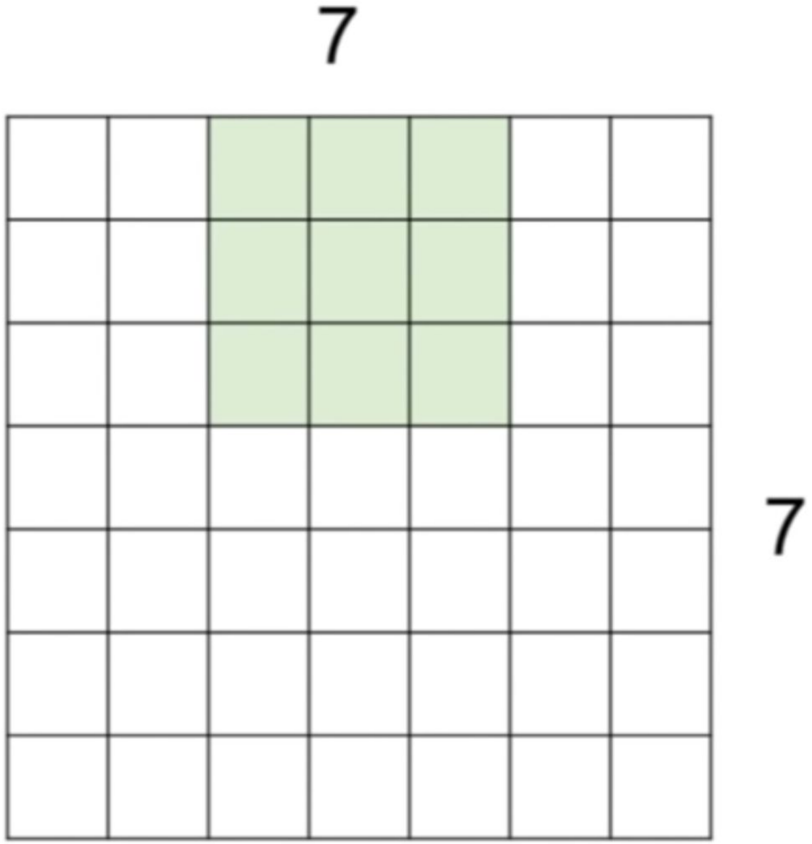
=> 5x5 output

Stride 2:



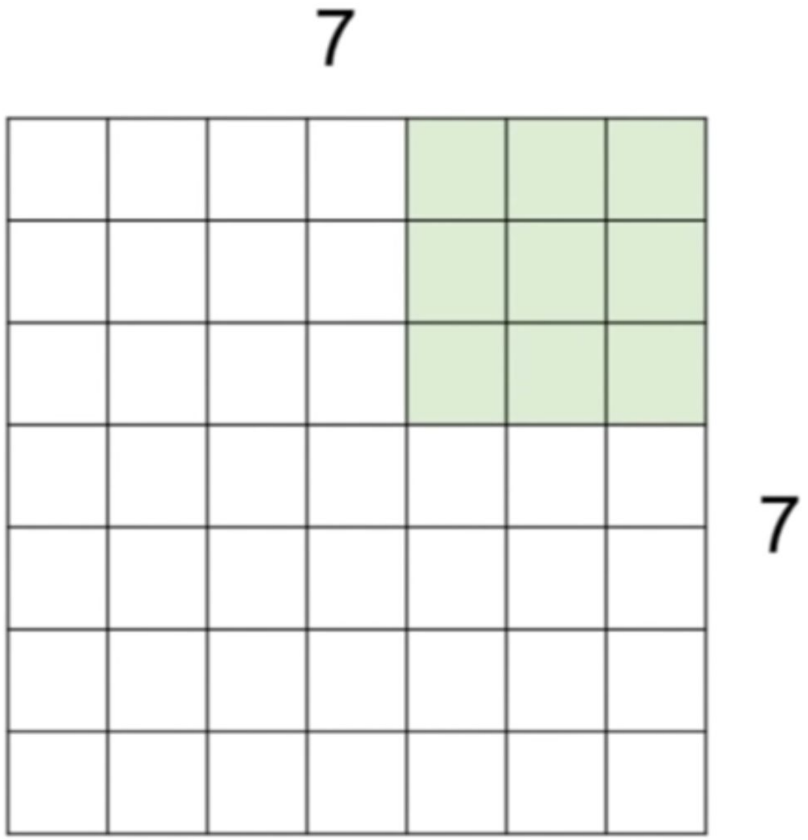
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride 2:



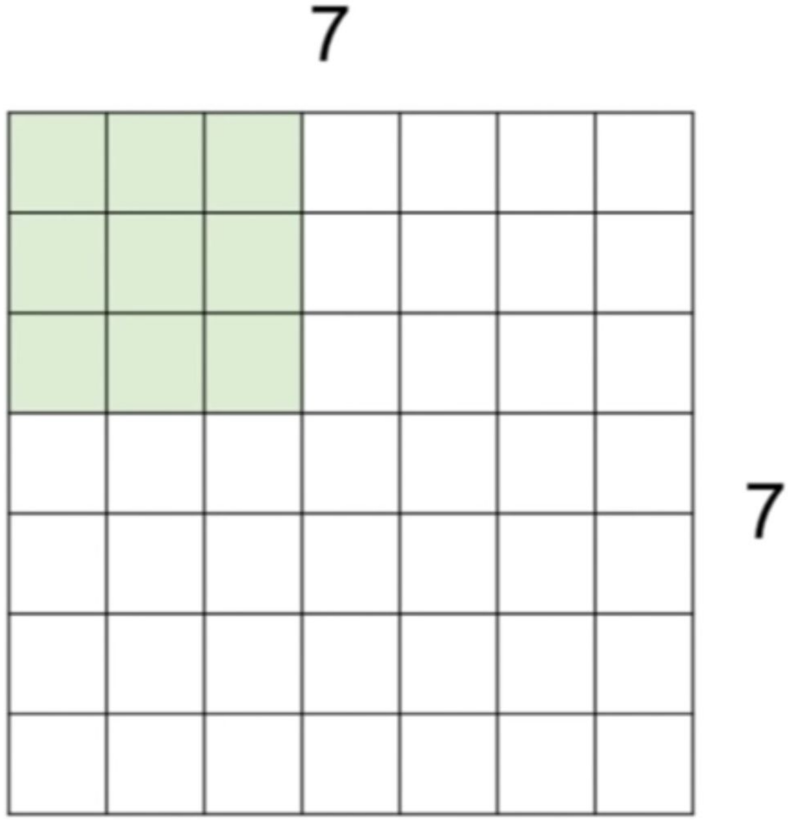
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

Stride 2:

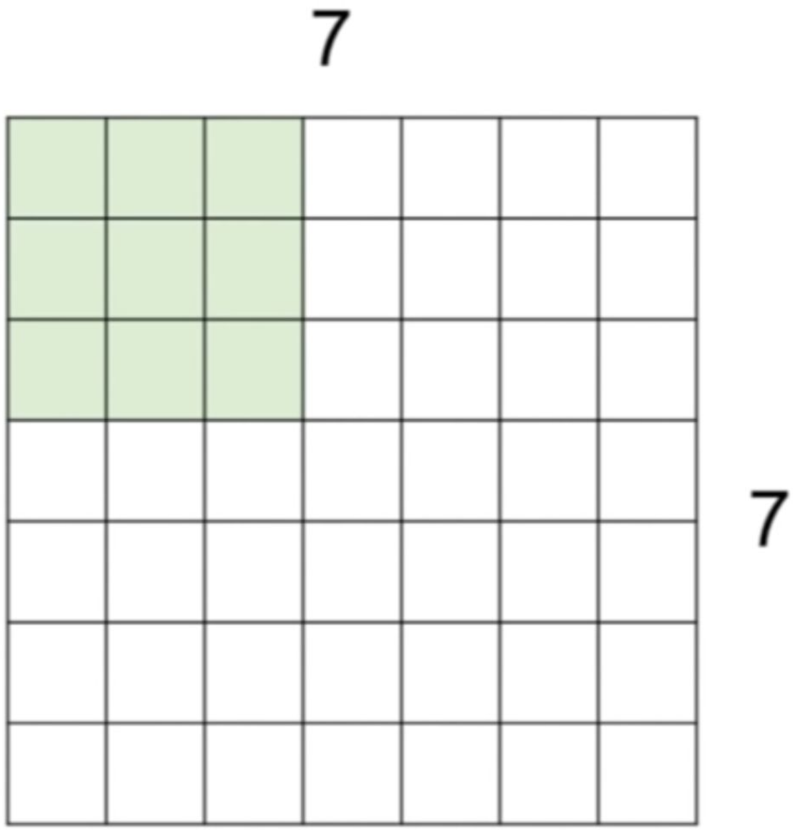


7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Stride 3:



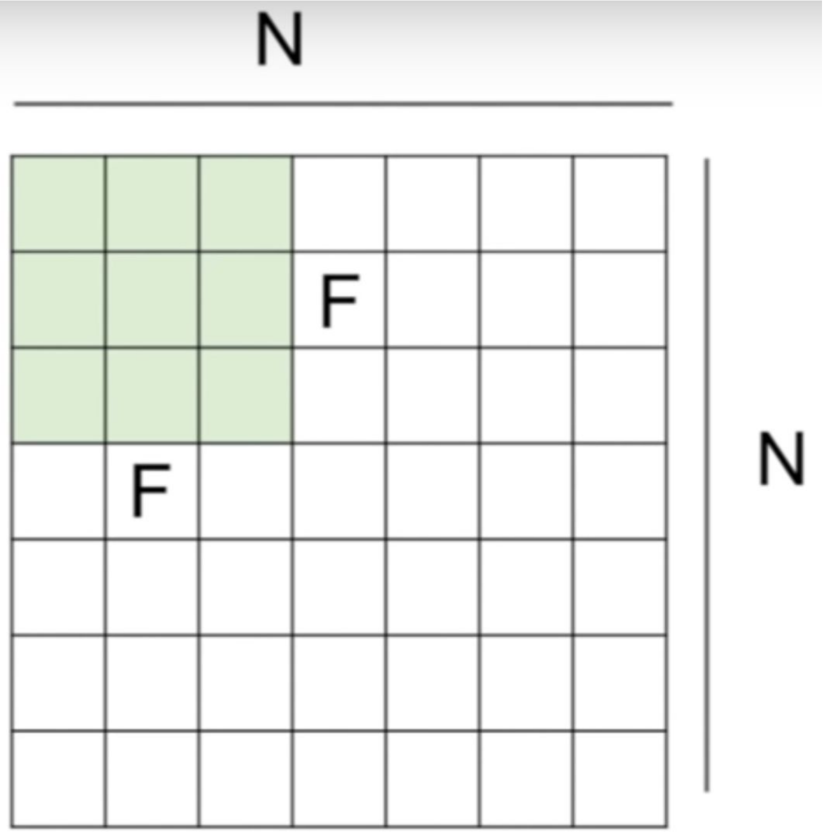
7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**



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

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

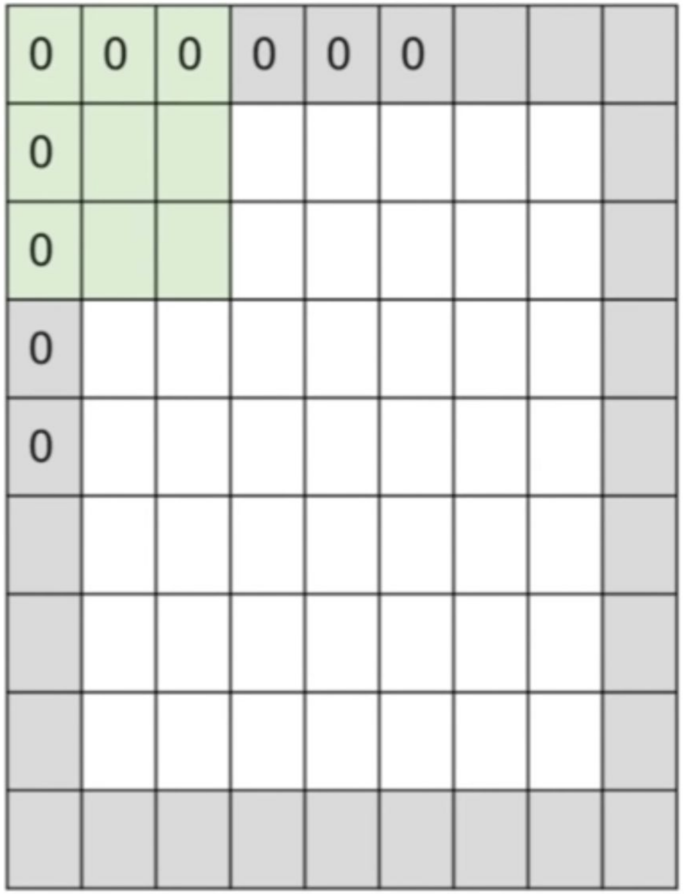
Calculating output size:



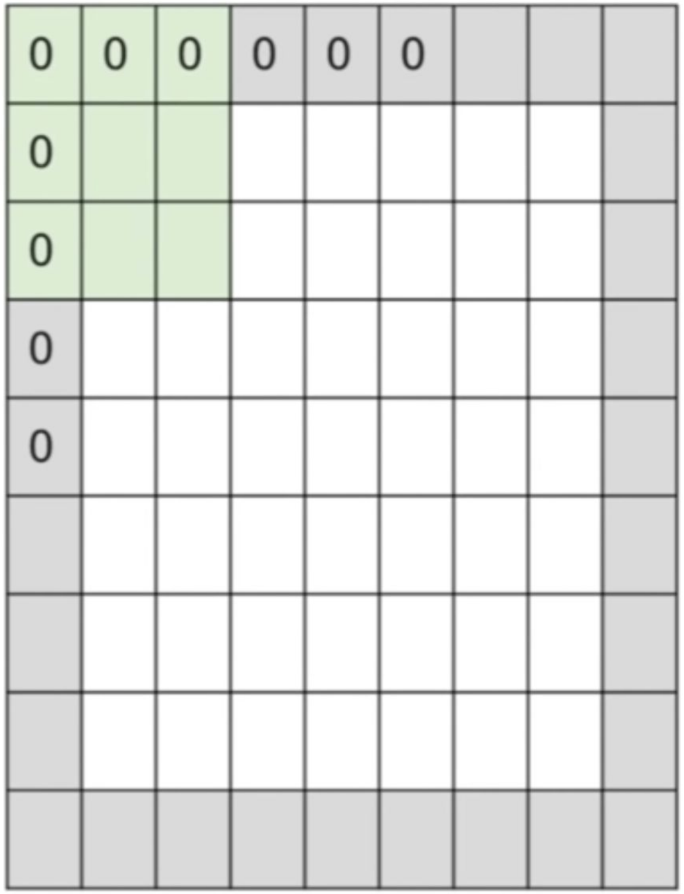
Output size:
(N - F) / stride + 1

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 :\

In practice: Common to zero pad the border



In practice: Common to zero pad the border

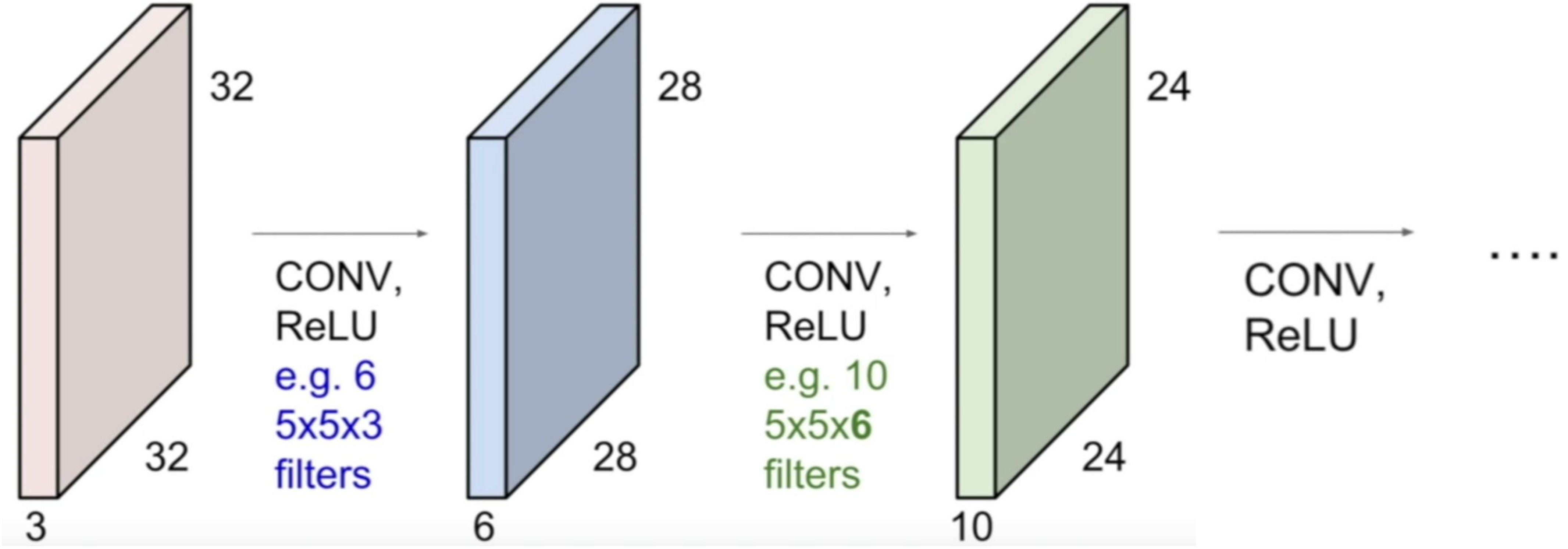


e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

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

Remember back to…

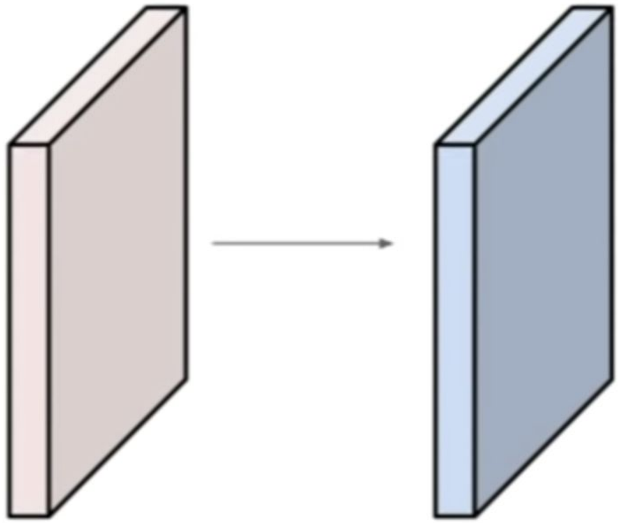
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.



Examples time:

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

Output volume size: ?

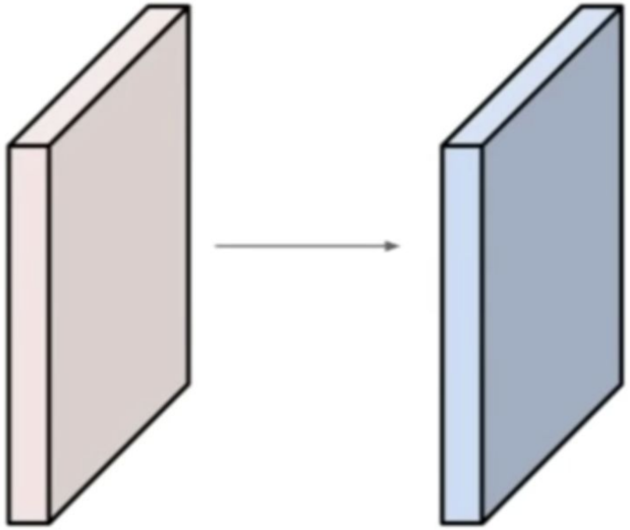


Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
$(32+2*2-5)/1+1 = 32$ spatially, so
32x32x10

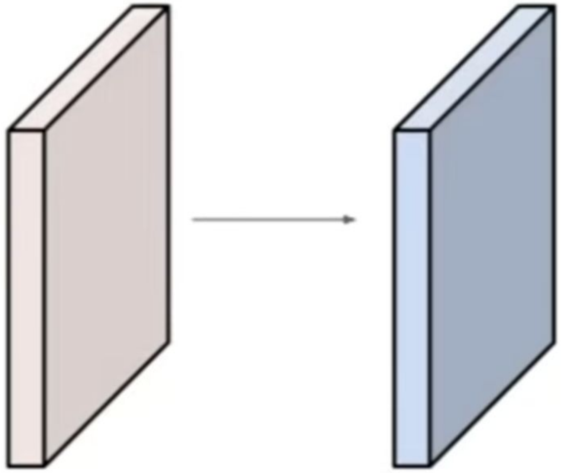
Examples time:

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



Number of parameters in this layer?

Examples time:



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

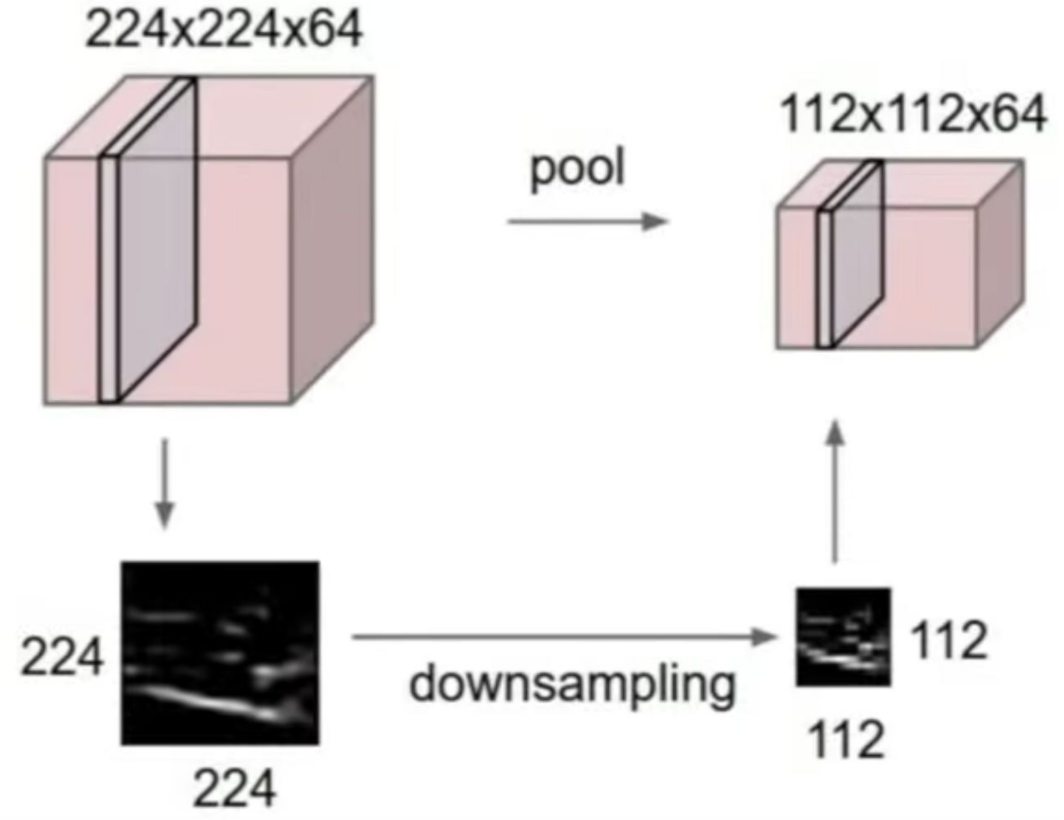
Number of parameters in this layer?
each filter has 5*5*3 + 1 = 76 params (+1 for bias)
=> 76*10 = **760**

Hyperparameters

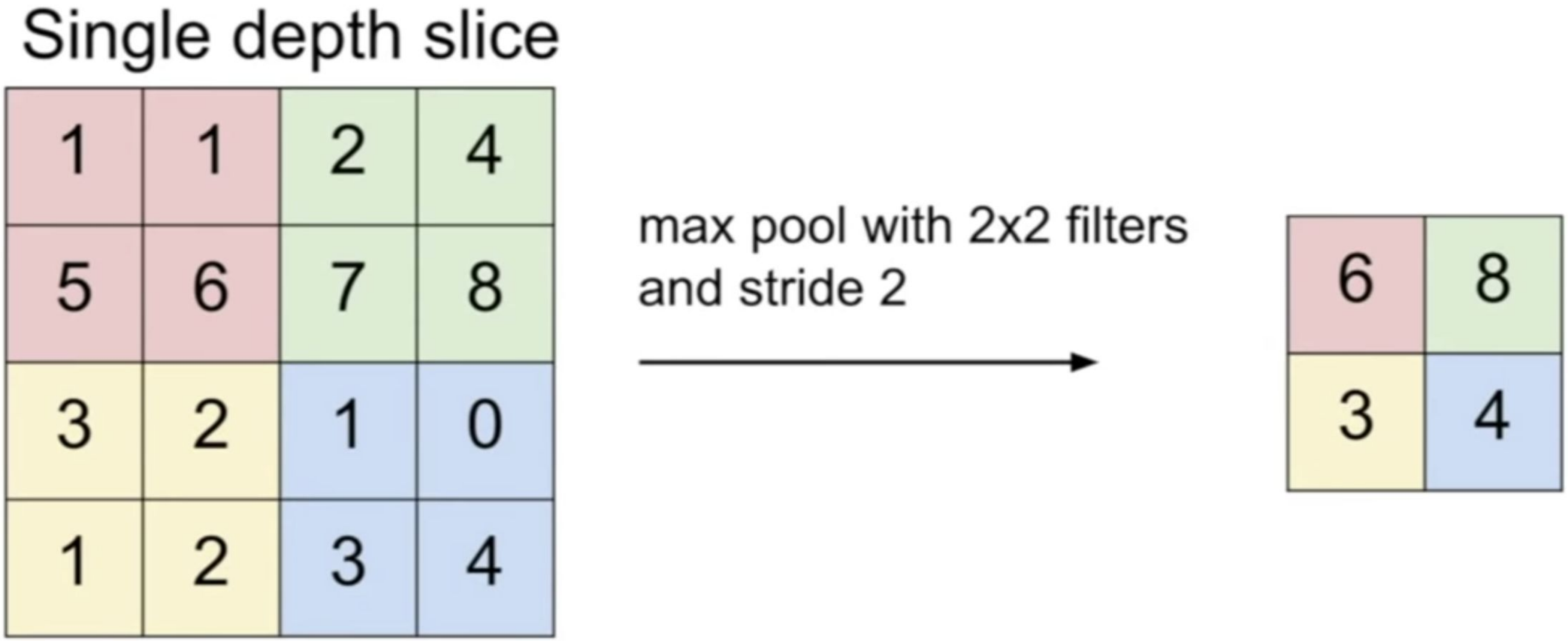
- Hyperparameters can be tuned to change complexity and size of extracted features
- Filter Size, Stride (step size of the filter), Zero-padding (to maintain dimensions)

Pooling Layer

- Makes the representations smaller and more manageable
- Operates over each activation map independently

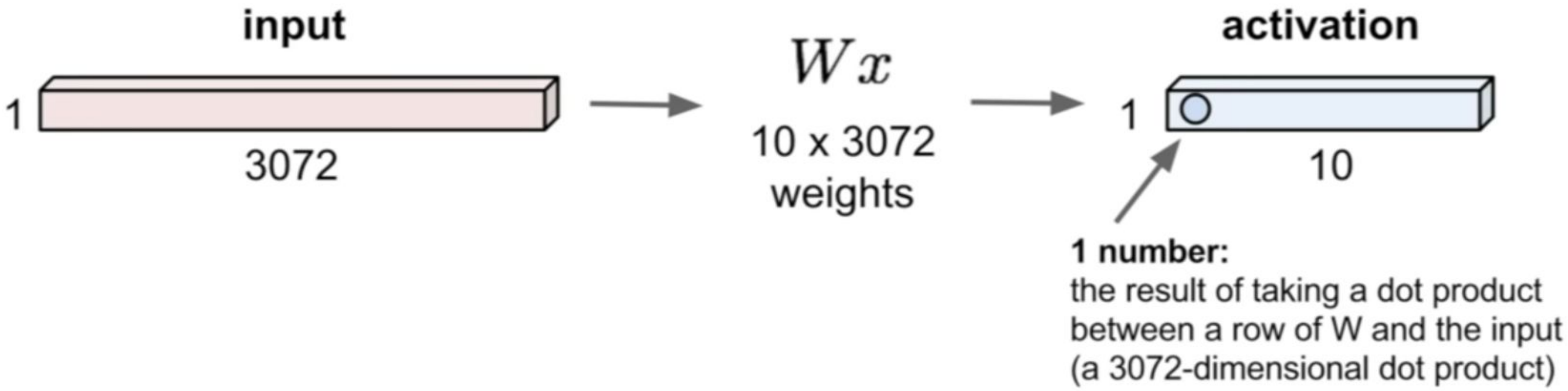


Max Pooling


Single depth slice
1
1
2
4
5
6
7
8
3
2
1
0
1
2
3
4
max pool with 2x2 filters
and stride 2
6
8
3
4

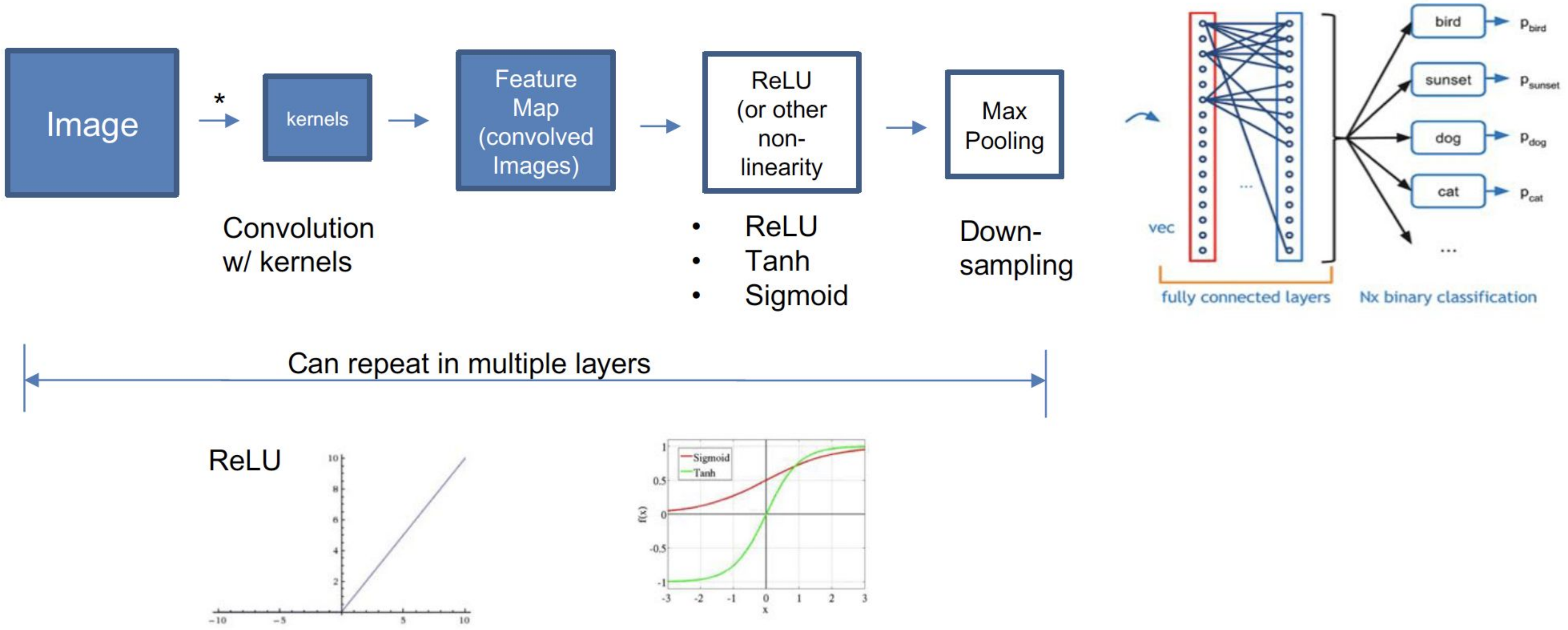
Fully Connected Layer

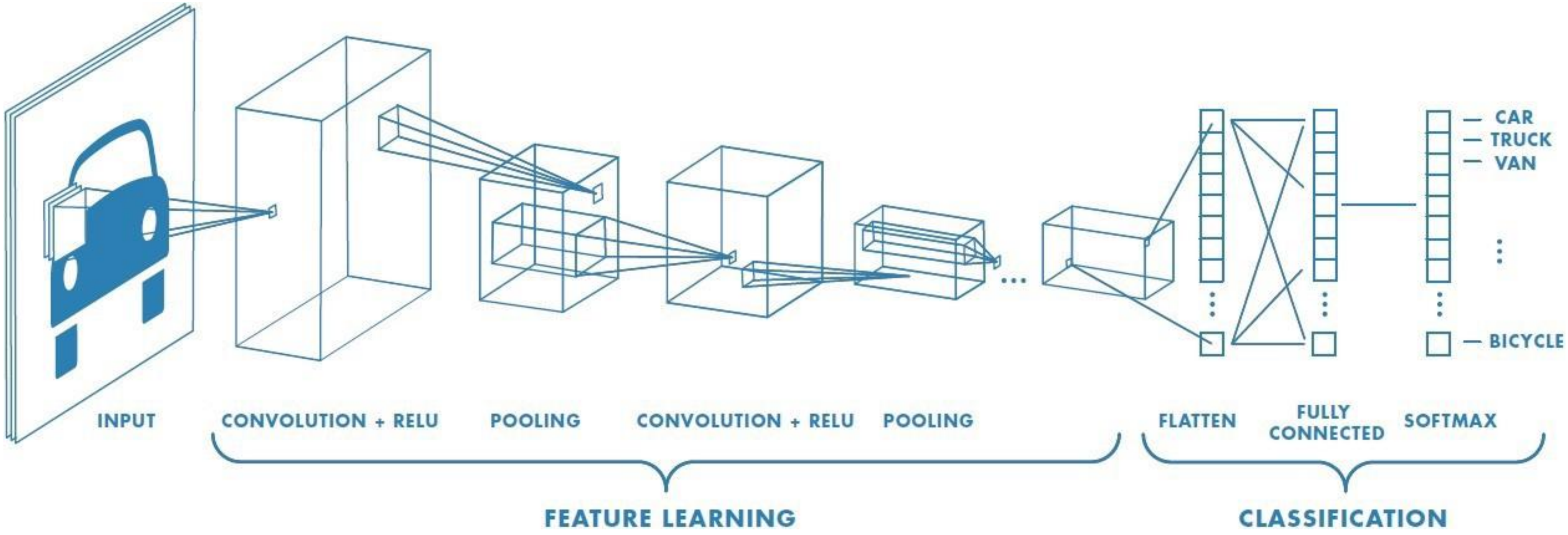
- 32 x 32 x 3 image => stretch 3072 x 1



1) Convolution and Non-linearity Blocks First

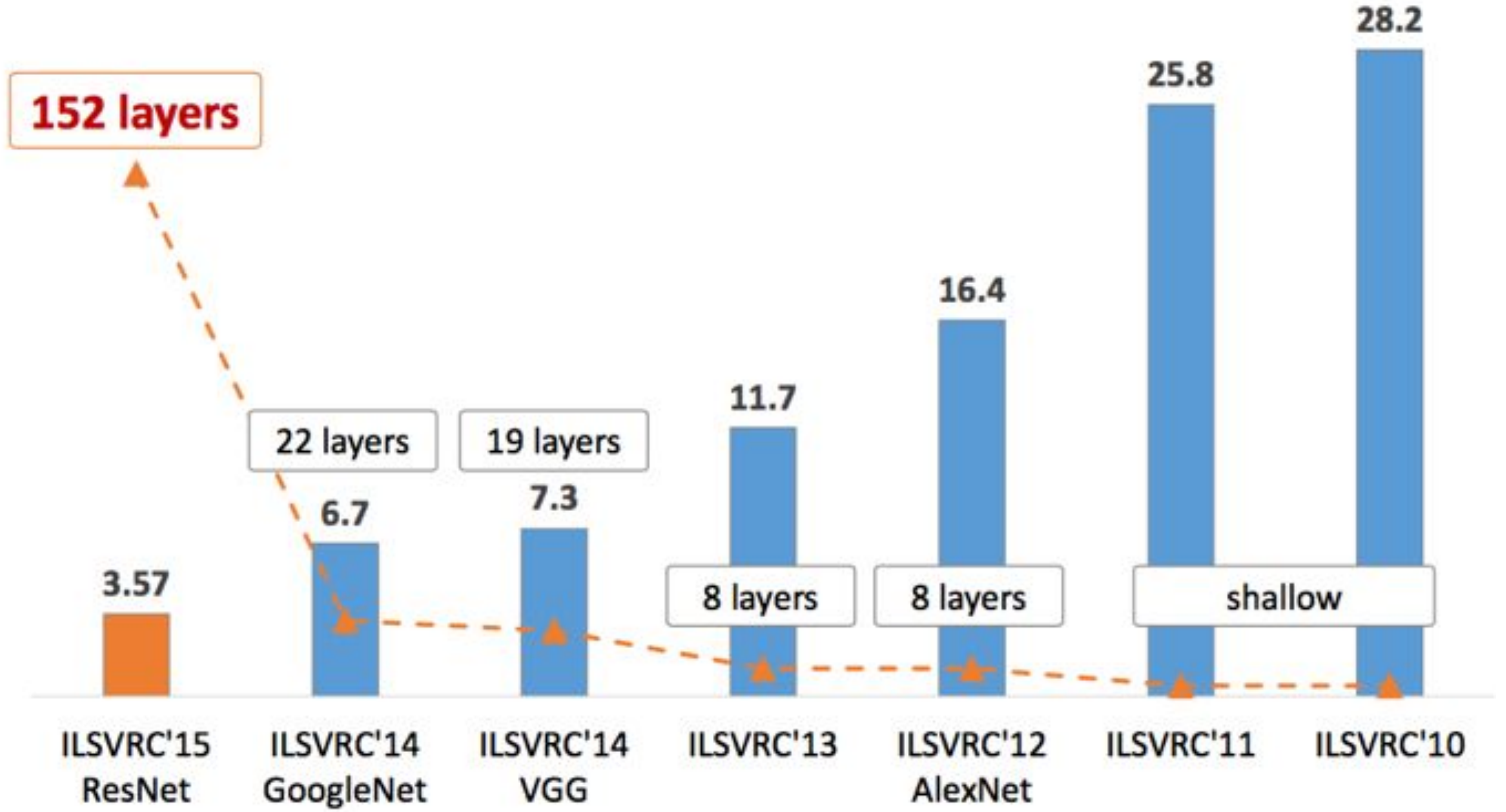
2) Then Fully Connected Layers leading to prediction



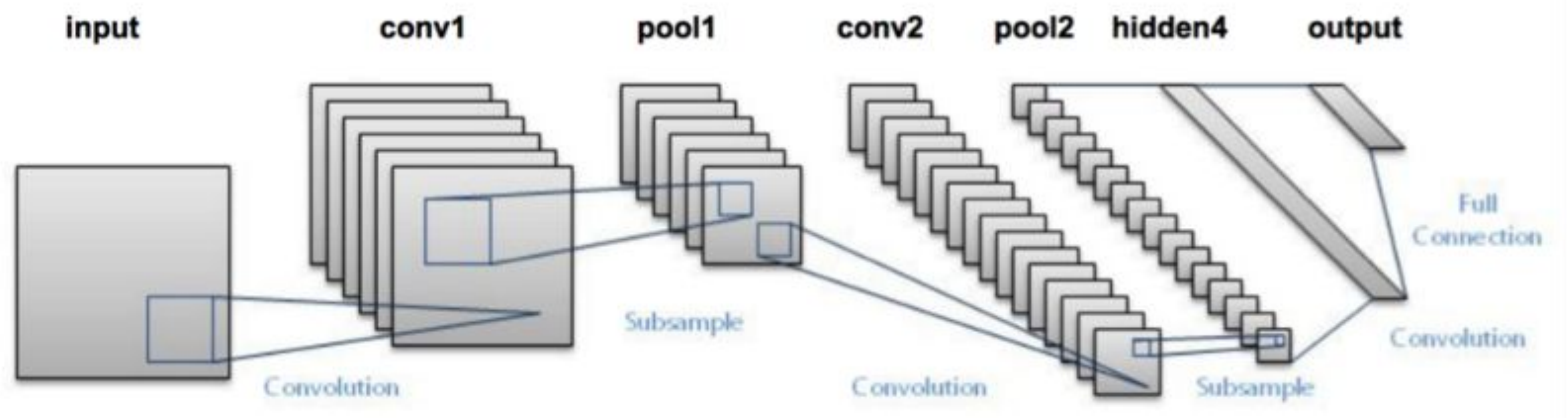

CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

LeNet vs ResNet vs VGGNet vs GoogLeNet

....

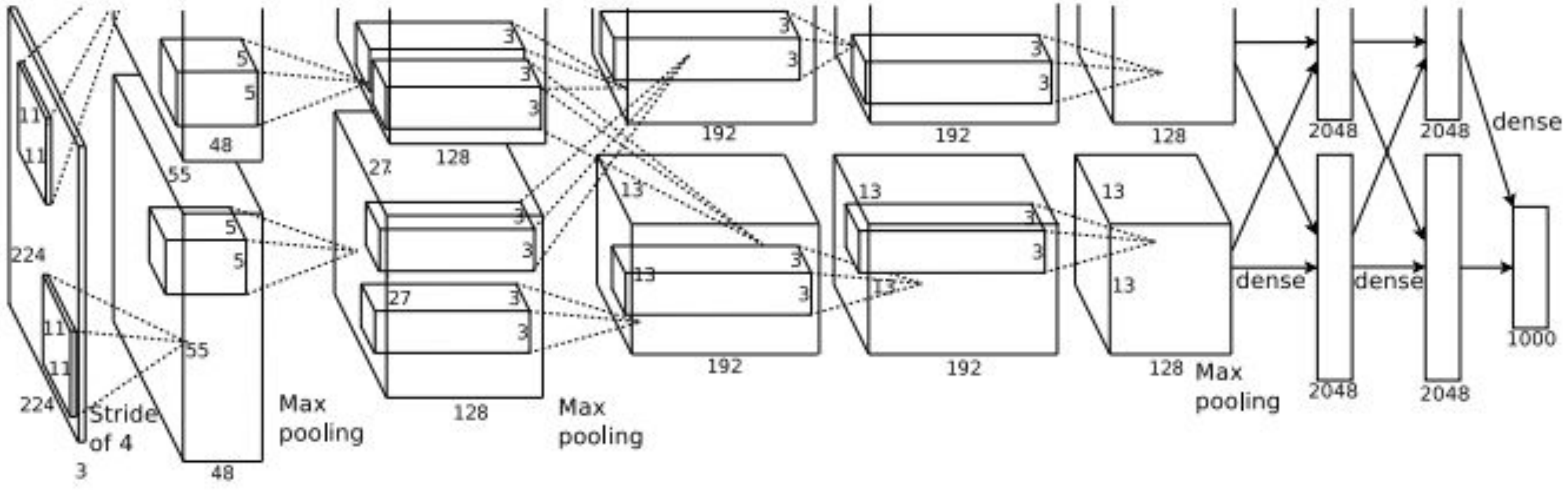


LeNet-5



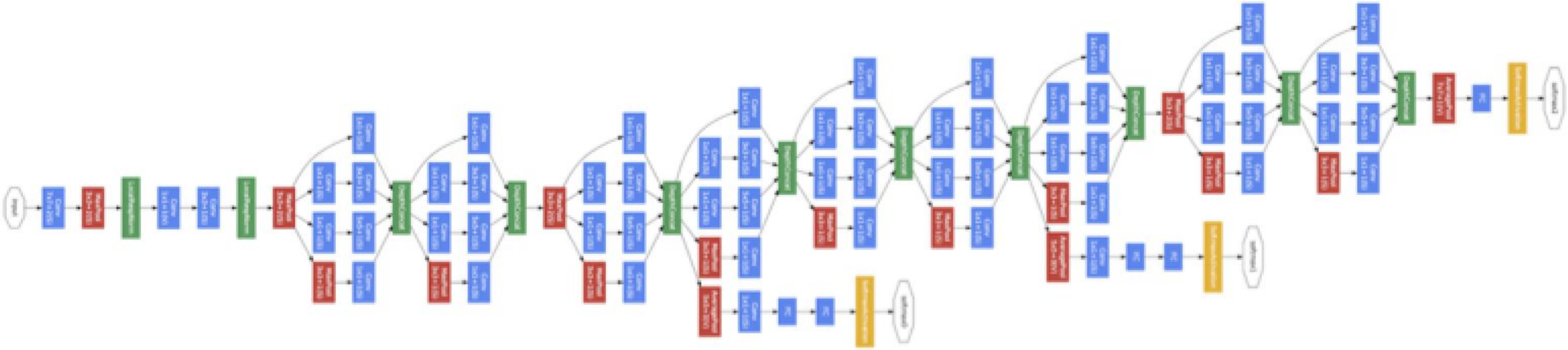
1. A pioneering 7-level convolutional network by LeCun et al in 1998
2. that classifies digits
3. 32x32 pixel grey scale input images

AlexNet



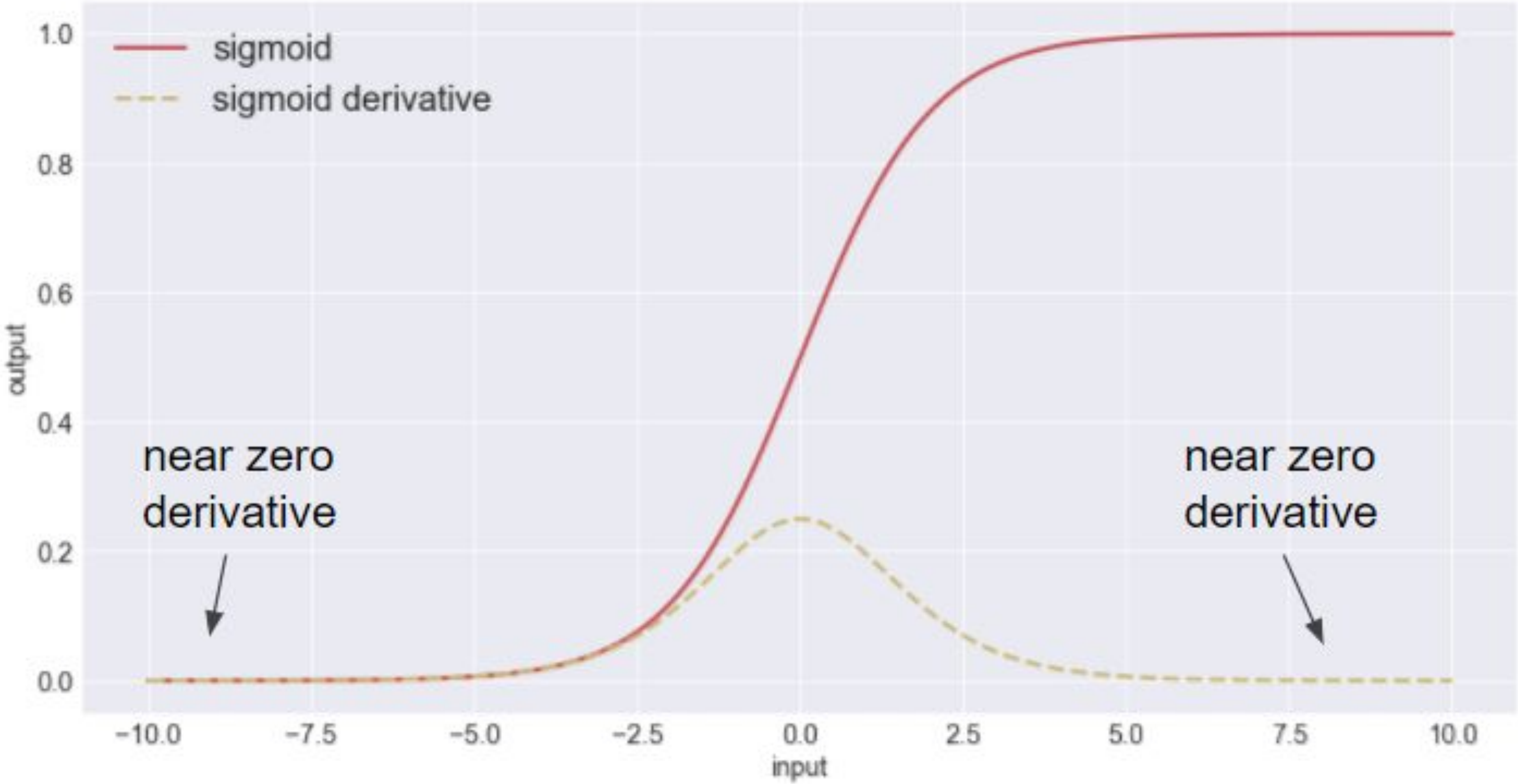
1. similar architecture as LeNet by Yann LeCun et al but was deeper, with more filters per layer, and with stacked convolutional layers.
2. It consisted 11x11, 5x5,3x3, convolutions, max pooling, dropout, data augmentation, ReLU activations, SGD with momentum.
3. Trained for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs
4. Reducing the top-5 error from 26% to 15.3% on ImageNet challenge.

GoogleNet / Inception(2014)



1. It achieved a top-5 error rate of 6.67%
2. It used batch normalization, image distortions and RMSprop
3. Their architecture consisted of a 22 layer deep CNN.
4. Reduced the number of parameters from 60 million (AlexNet) to 4 million.

Batch-Normalization??

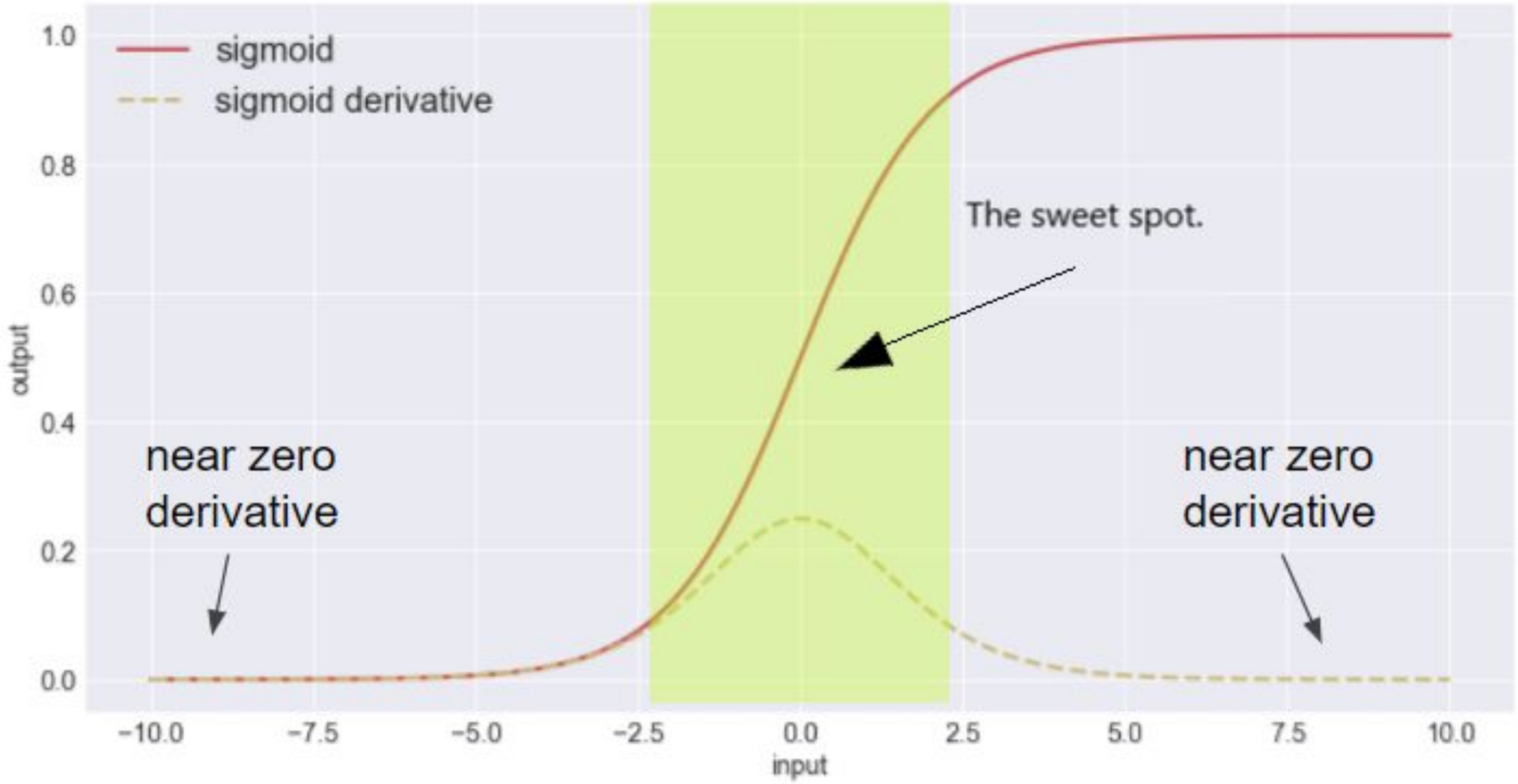


Batch-Normalization??

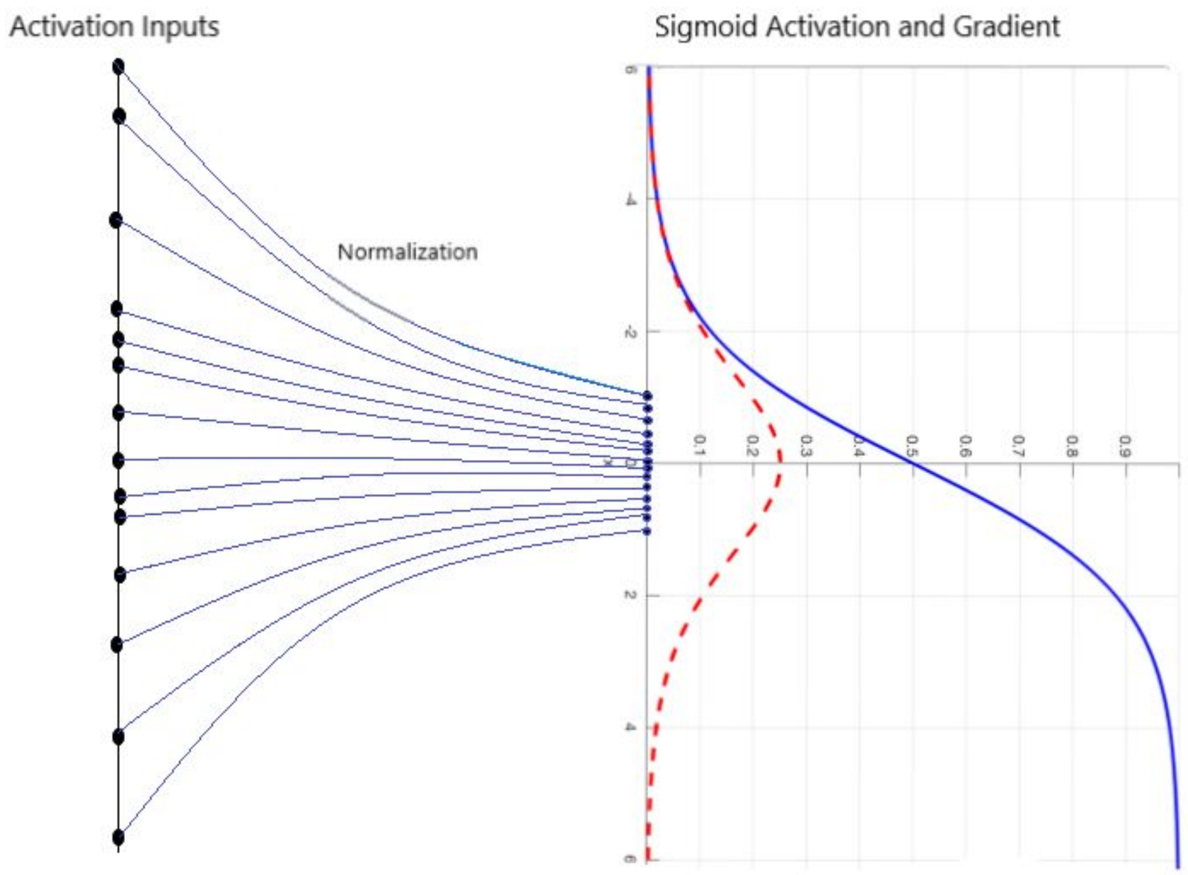
Problem 1: Optimization / Feature learning
As a network trains, weights in early layers change and as a result, the inputs of later layers vary wildly. Each layer must readjust its weights to the varying distribution of every batch of inputs. This slows model training. If we could make layer inputs more similar in distribution, the network could focus on learning the difference between classes.

Problem 2. Vanishing Gradients
When input distribution varies, so does neuron output. This results in neuron outputs that occasionally fluctuate into the sigmoid function's saturable regions. Once there, neurons can neither update their own weights nor pass a gradient back to prior layers. How can we keep neuron outputs from varying into saturable regions?

Batch-Normalization??



Batch-Normalization??

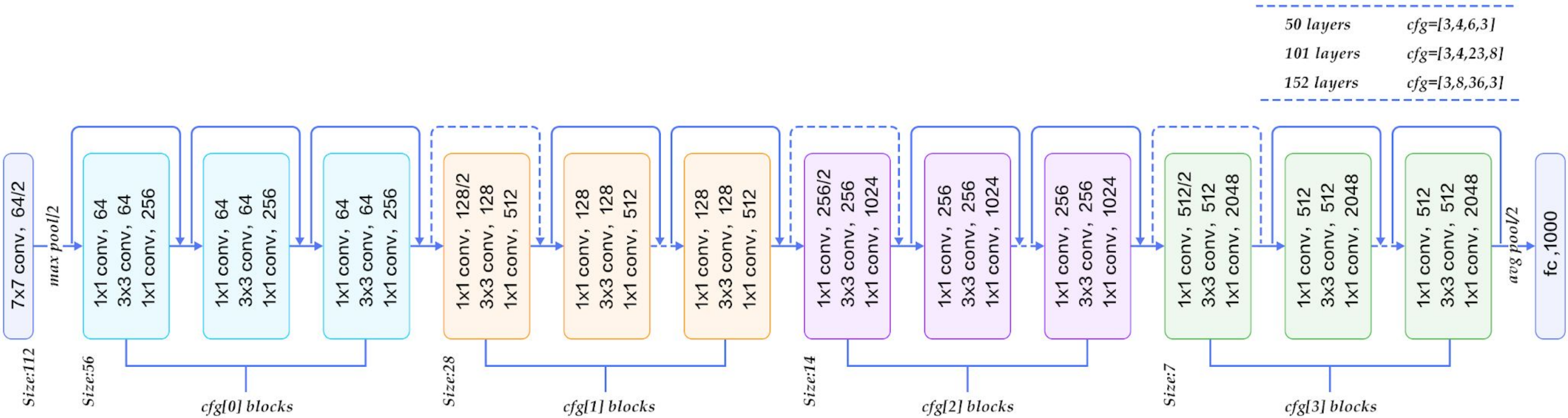


Batch normalization mitigates the effects of a varied layer inputs. By normalizing the output of neurons, the activation function will only receive inputs close to zero. This ensures a non-vanishing gradient, solving the second problem.

VGGNet (2014)

1. runner-up at the ILSVRC 2014
2. Similar to AlexNet, only 3x3 convolutions, but lots of filters.
3. Trained on 4 GPUs for 2–3 weeks.
4. It is currently the most preferred choice in the community for extracting features from images.
5. VGGNet consists of 138 million parameters, which can be a bit challenging to handle.

ResNet(2015)



1. novel architecture with "skip connections" and features heavy batch normalization.
2. 152 layers while still having lower complexity than VGGNet.
3. top-5 error rate of 3.57% which beats human-level performance on this dataset.

Resources

https://skymind.ai/wiki/convolutional-network

http://deeplearning.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/

http://cs231n.stanford.edu/

Berkeley CS182/282A

Deep Learning book by Ian Goodfellow, Yoshua Bengio, Aaron Courville