Lecture Series on
Hardware for Deep Learning

Part 2: Convolutional Neural Networks

Dr. Adam Teman
EnICS Labs, Bar-Ilan University

22 March 2020

EnICS
Emerging Nanoscaled
Integrated Circuits and Systems Labs

Tradition of Excellence
Bar-Ilan University
אוניברסיטת בר-אילן

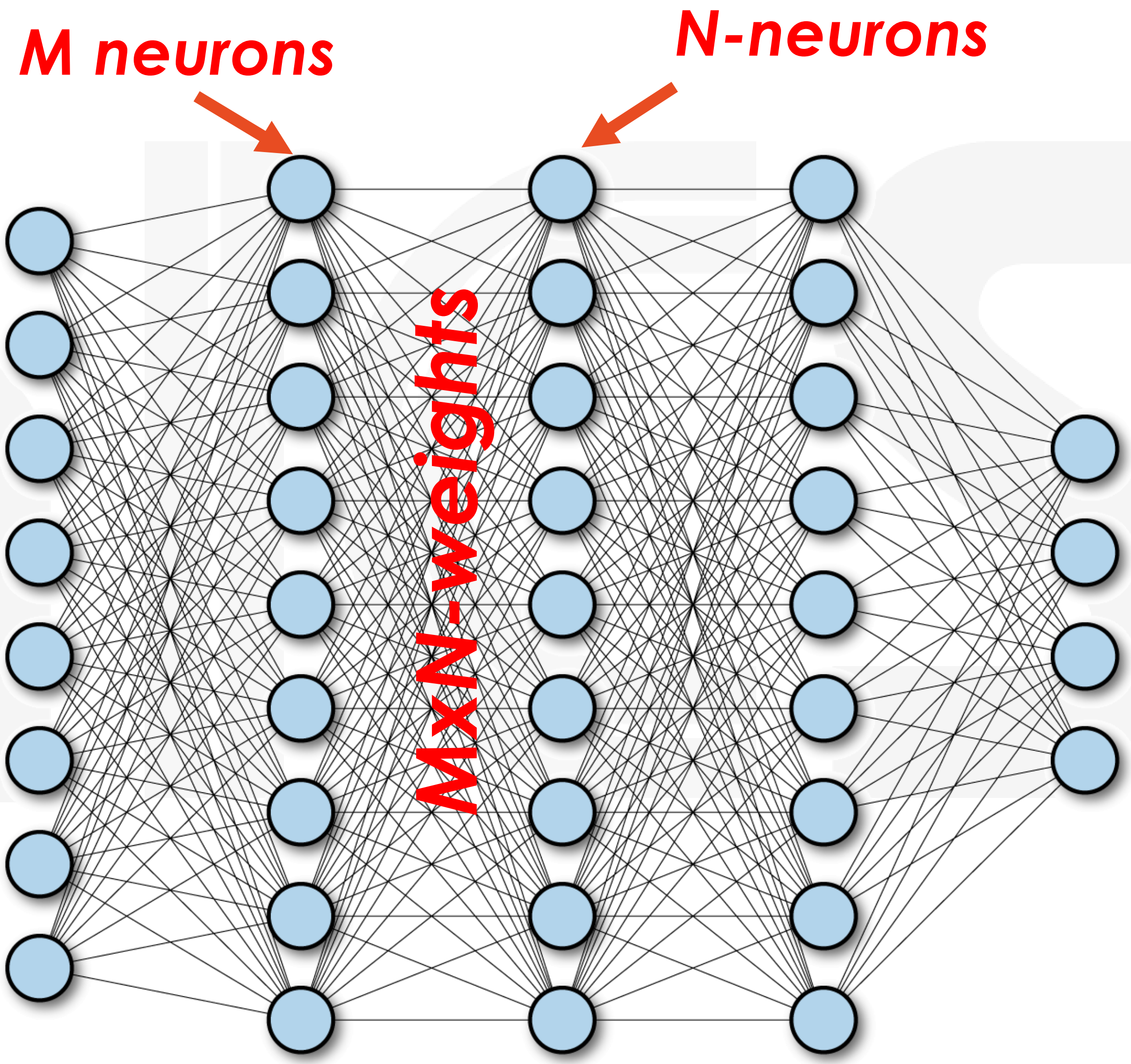
Outline


Introduction to CNNs
CNN Basics
The size of a CNN
Additional CNN Components
Famous CNNs

Introduction to CNNs

Explosion in Fully-Connected DNNs

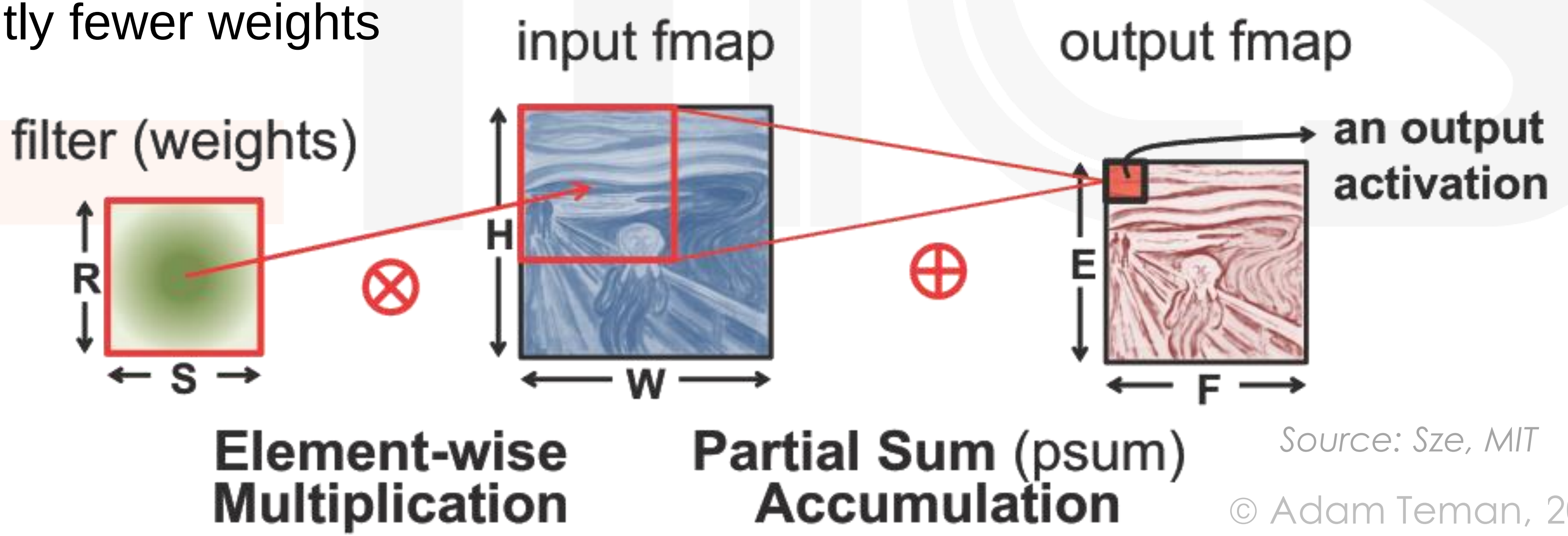
- **Fully connected layers have a synapse connecting each neuron in one layer to every neuron in the following layer.**
  - Each layer has *MxN* weights!
- **This results in huge storage and computation requirements**



Source: O'Reilly

Convolutional NN Relaxes Requirements

- **Inspired by the visual cortex:**
  - Cortical neurons respond only to stimuli in the receptive field.
- **CNNs or ConvNets are sparsely connected NNs with weight sharing.**
  - *RxS* weights/layer (*R*, *S* are small)
  - Fewer operations per output
  - Significantly fewer weights



Source: Sze, MIT

LeNet – the original CNN

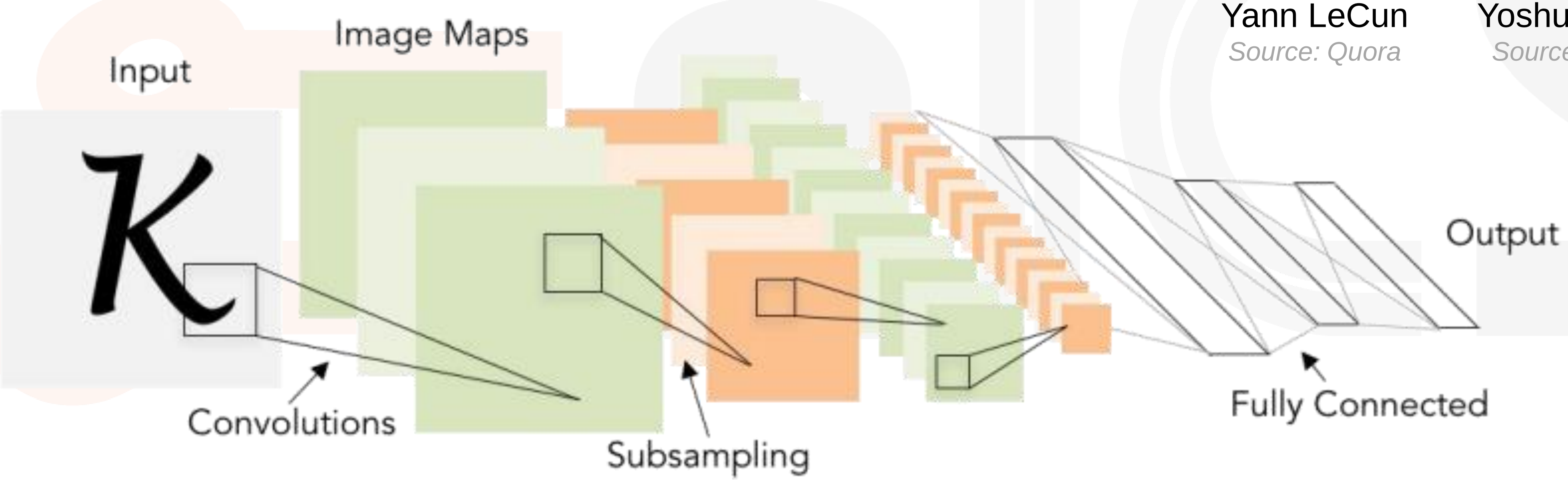
- "*Gradient-based learning applied to document recognition*", **LeCun, Bottou, Bengio, Haffner 1998**

Yann LeCun

Source: Quora

Yoshua Bengio

Source: AI Week



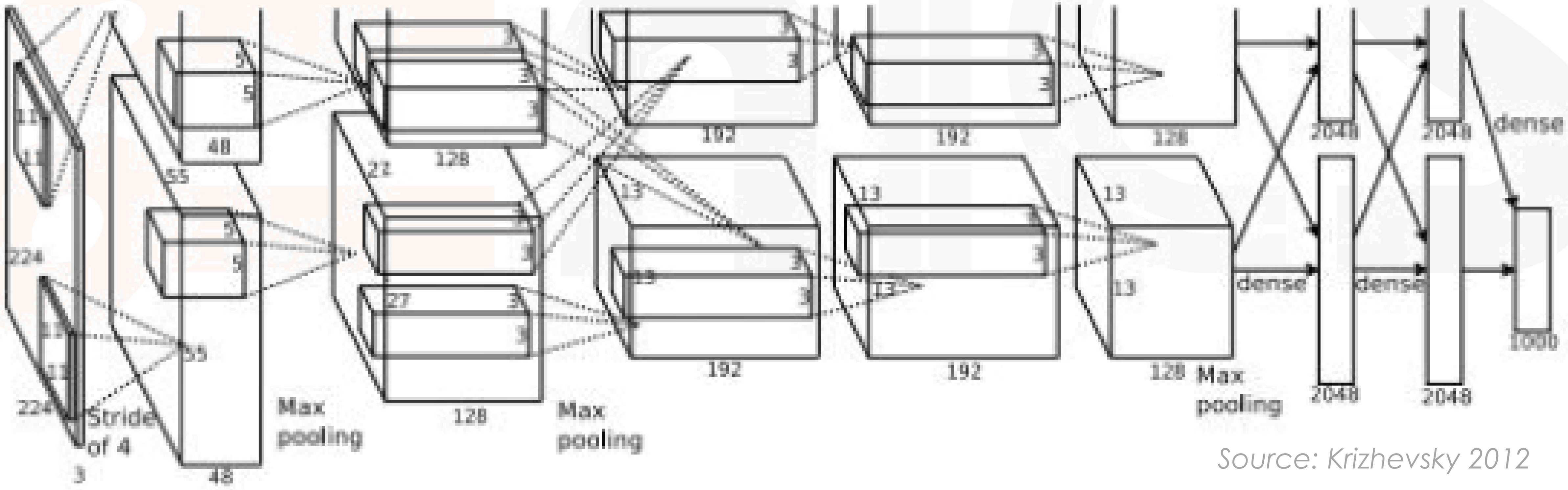
LeNet-5

Source: LeCun 1998

The AlexNet Breakthrough

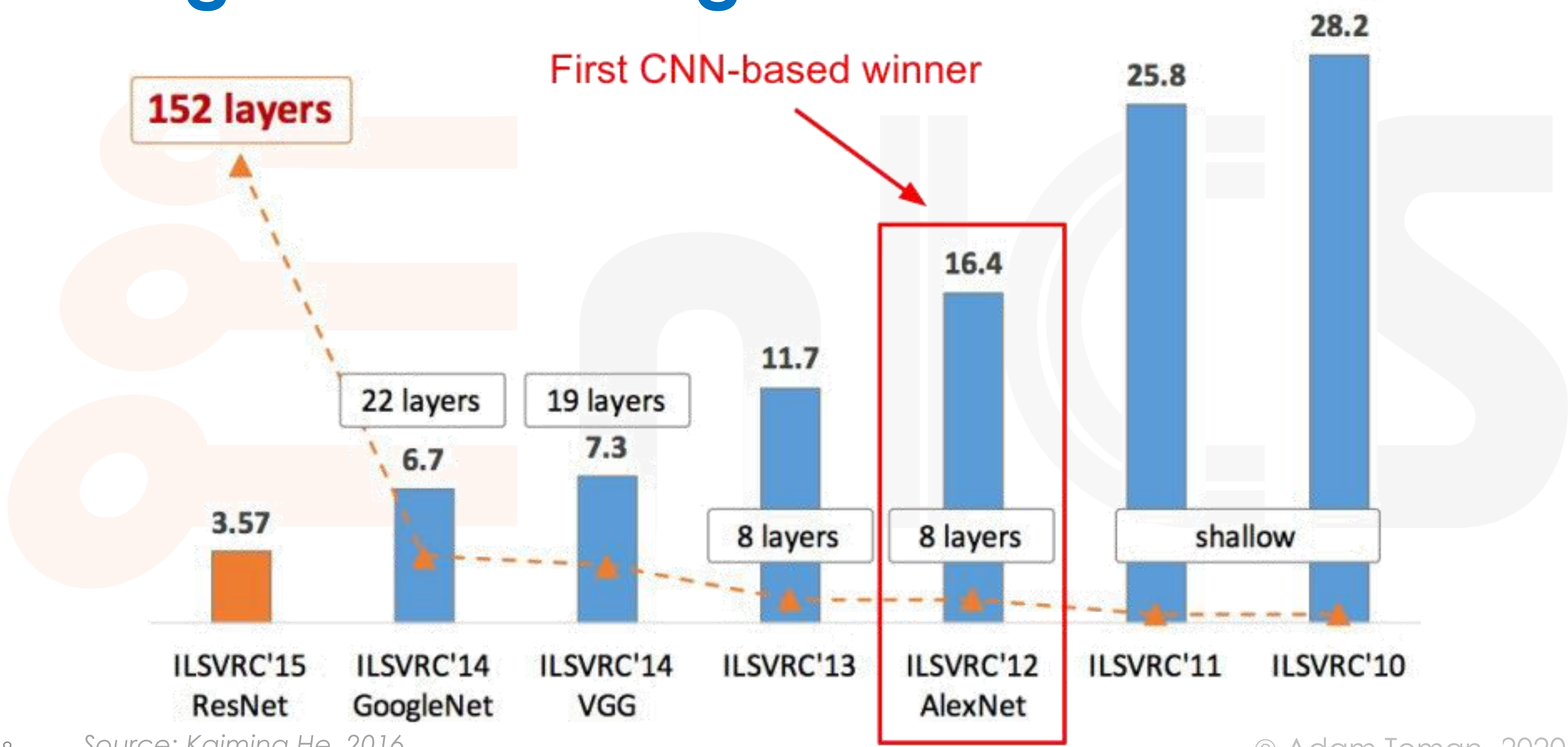
- "*ImageNet classification with deep convolutional neural networks*", Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012

Alex Krizhevsky
Source: Quartz



Source: Krizhevsky 2012

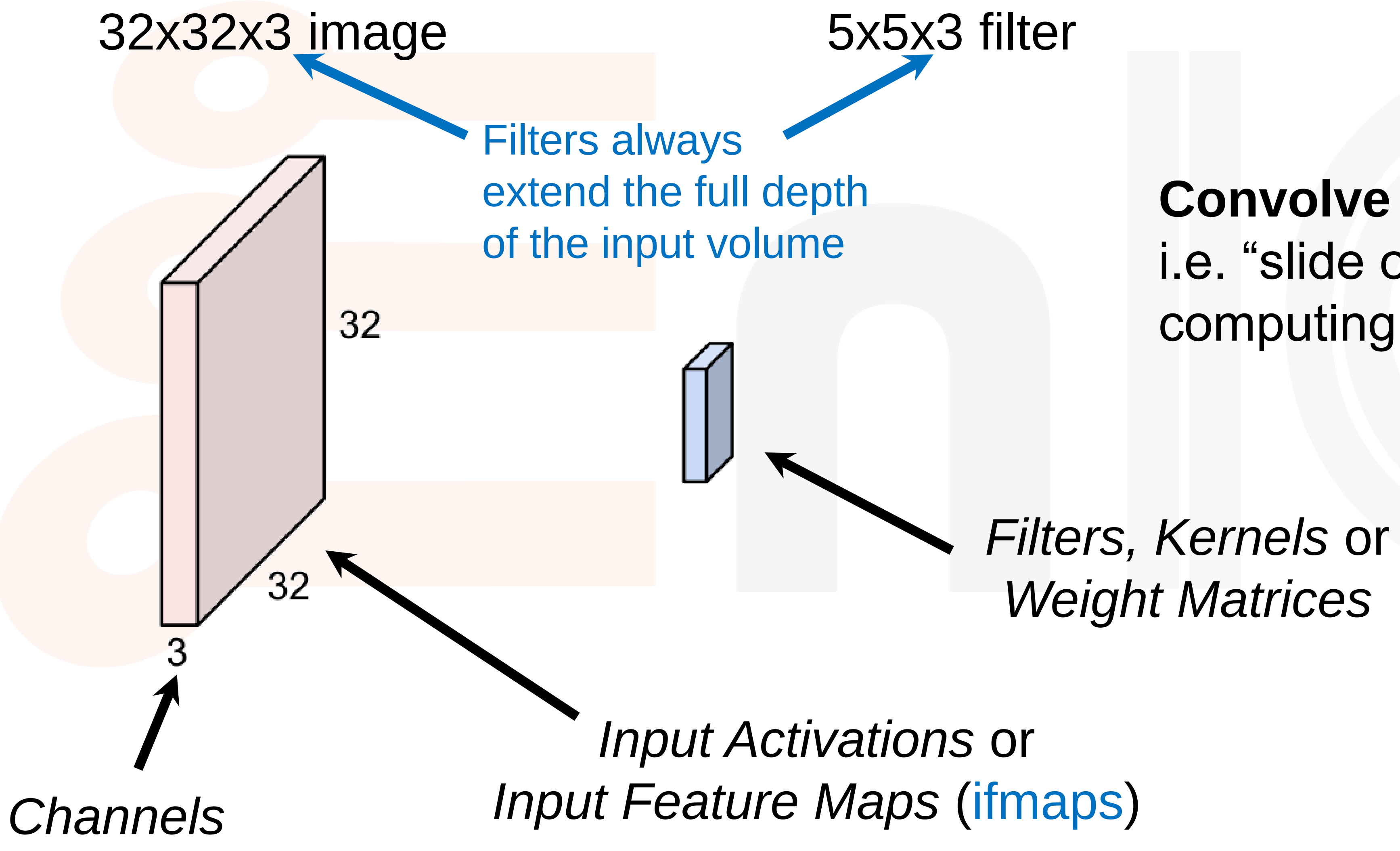
ImageNet Challenge Winners



Source: Kaiming He, 2016

CNN Basics

Convolution Layer

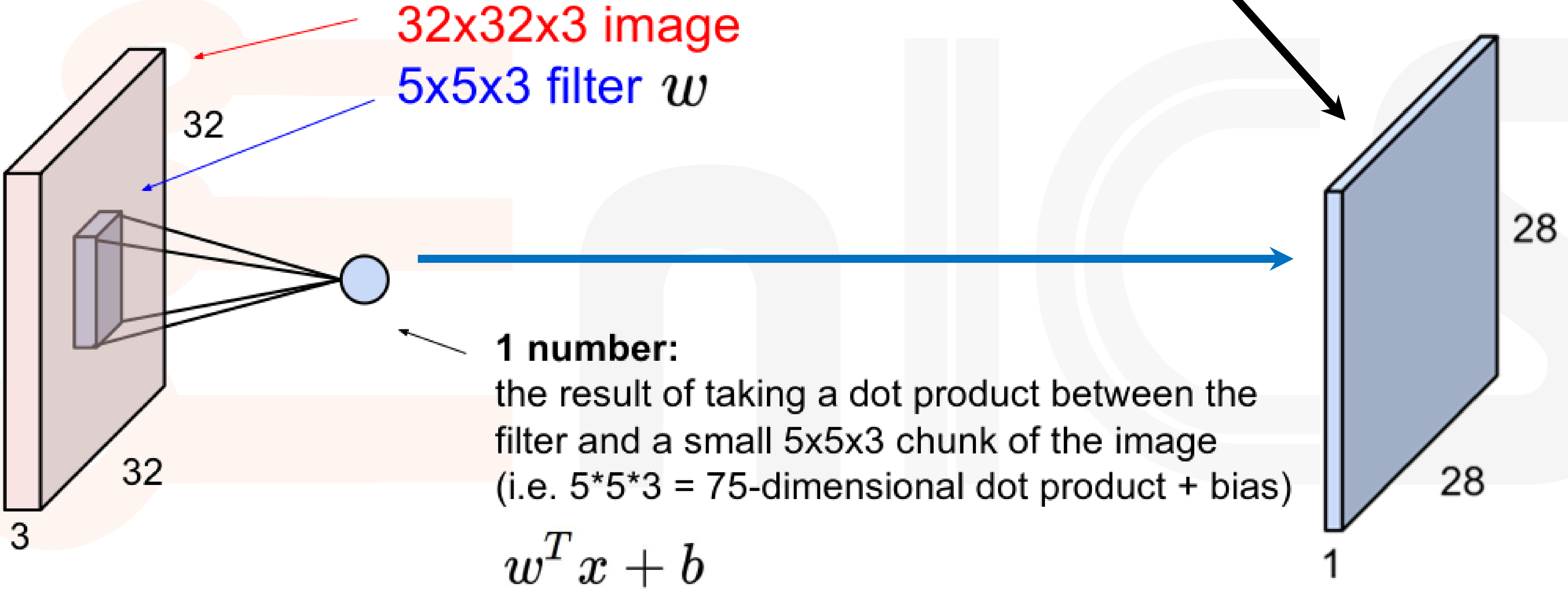


Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Source: Stanford 321n

Convolution Layer

Output Activations or
Output Feature Maps (ofmaps)



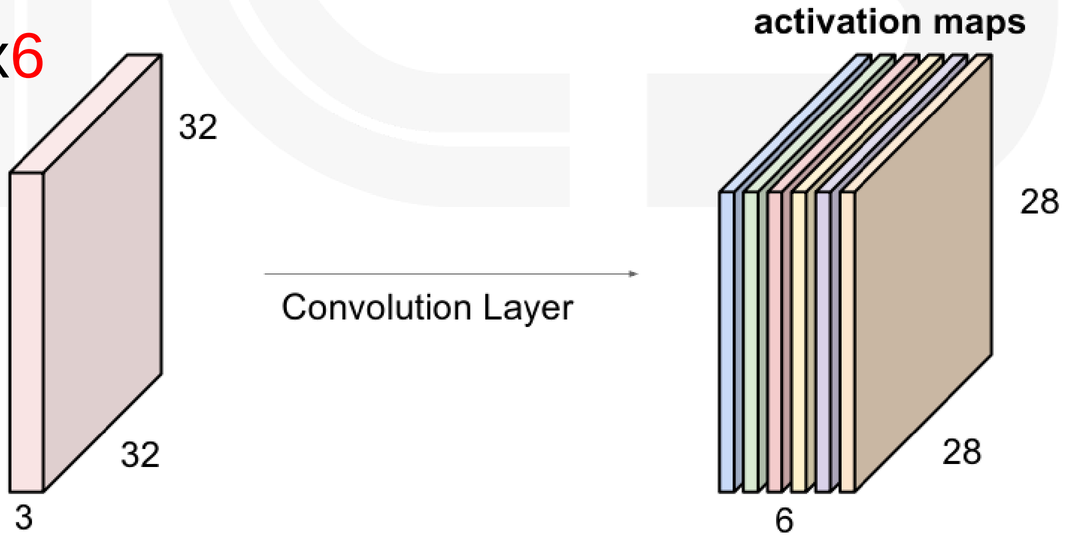
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Source: Stanford 321n

Multiple Filters

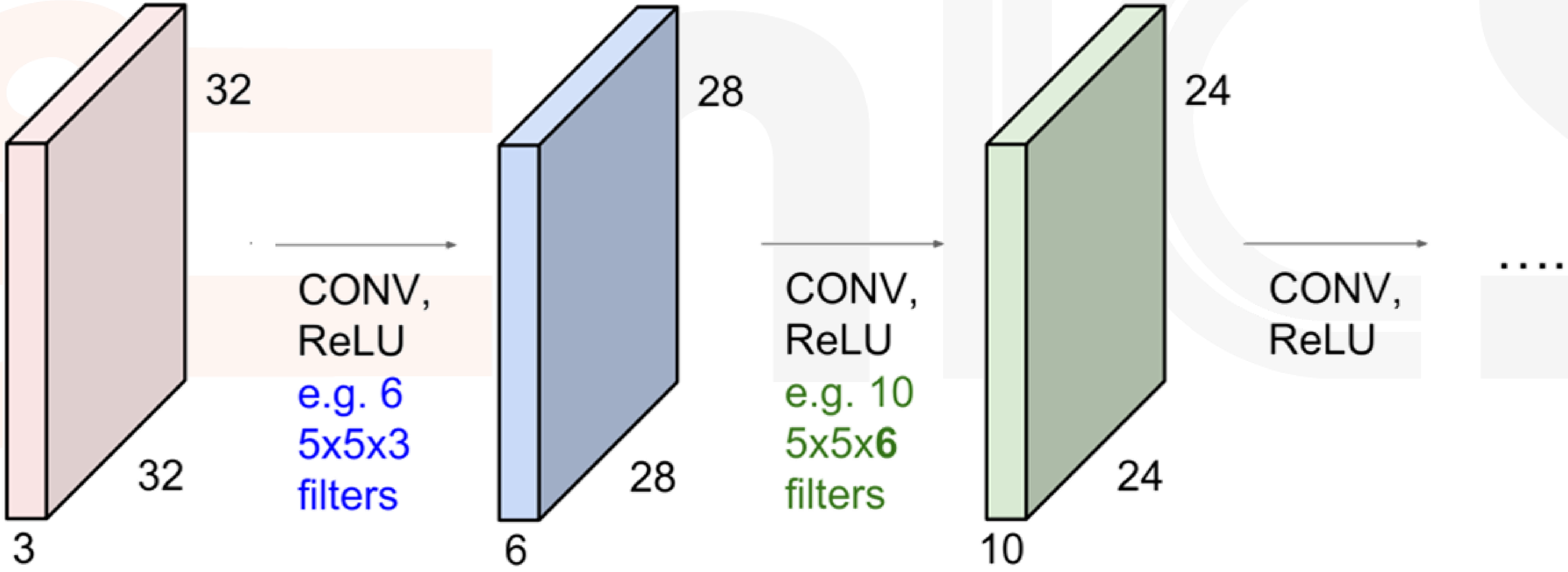
- **Each filter has the <u>same depth</u> as the input map.**
  - For example, if the input is 32x32x3, the filter size could be 5x5x3.
- **But there can be (are) <u>many filters</u> in a convolution layer**
  - Each filter will result in another <u>separate activation map</u>.
  - This will be the channel depth of the next input layer.
- **For example, given 6 filters, we get 6 image maps**
  - And the input to the next layer is now 28x28x6



Source: Stanford 321n

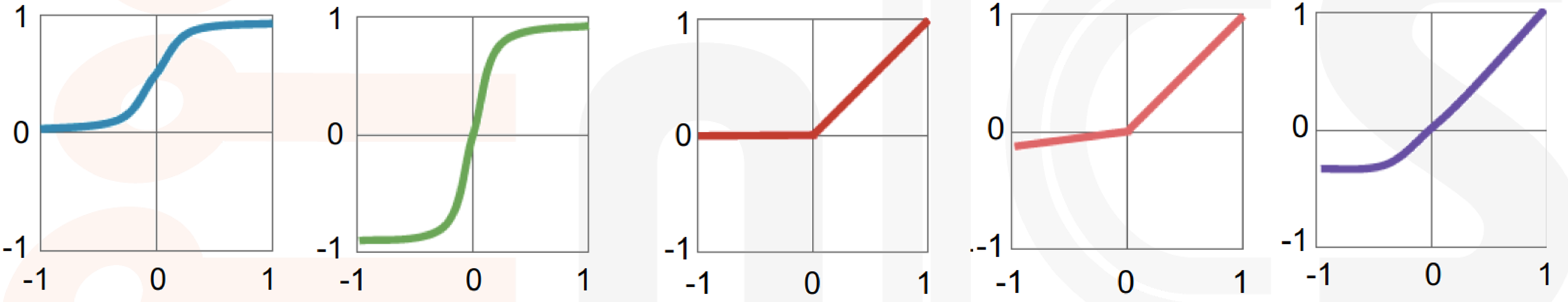
A basic CNN

- **A basic CNN is a sequence of Convolution Layers, interspersed with Activation Functions**



Source: Stanford 321n

Activation Functions



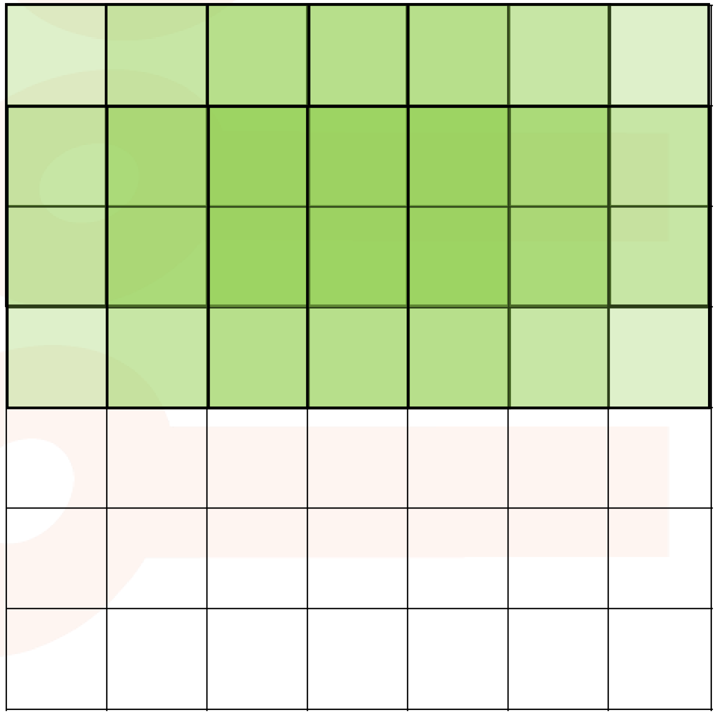
Sigmoid	Hyperbolic Tangent	Rectified Linear Unit (ReLU)	Leaky ReLU	Exponential LU
$y=1/(1+e^{-x})$	$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$	$y=\max(0,x)$	$y=\max(\alpha x,x)$ α = small const. (e.g. 0.1)	$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$

Source: Stanford 321n

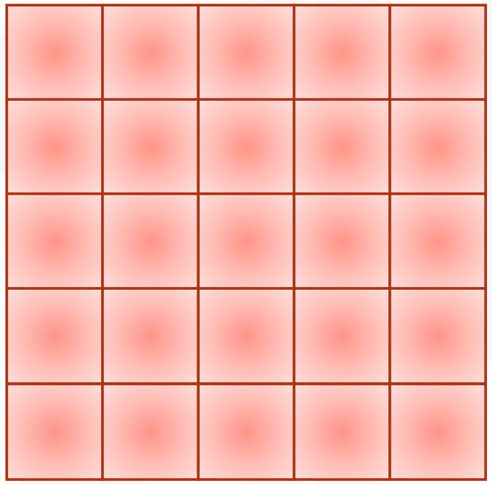
The size of a CNN

How large is the output of a Conv layer?

- **Given a 7x7x1 input and a 3x3x1 filter:**

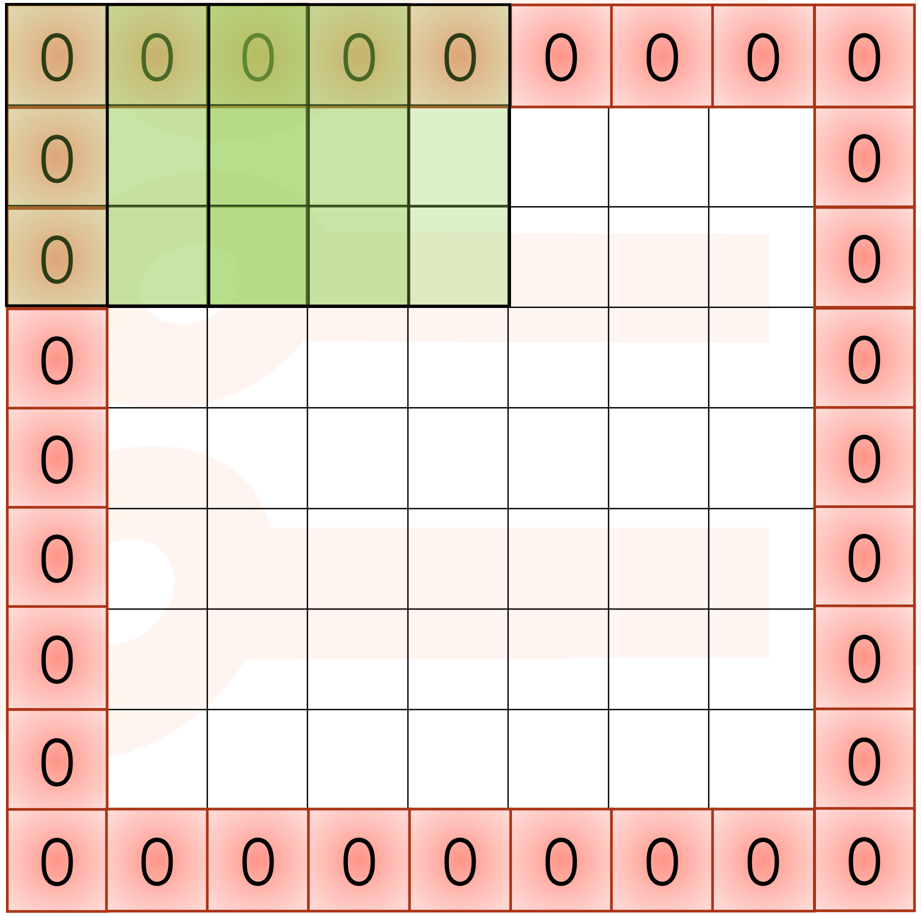


5x5x1 Activation Map

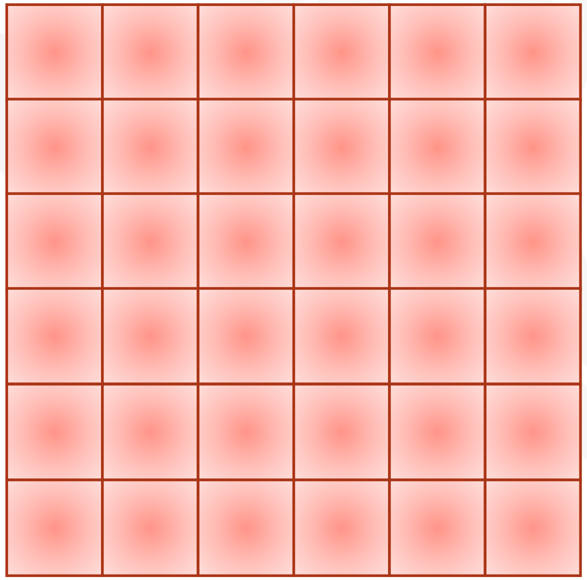


Padding with Zeros

- **Pad with zeros to account for borders**

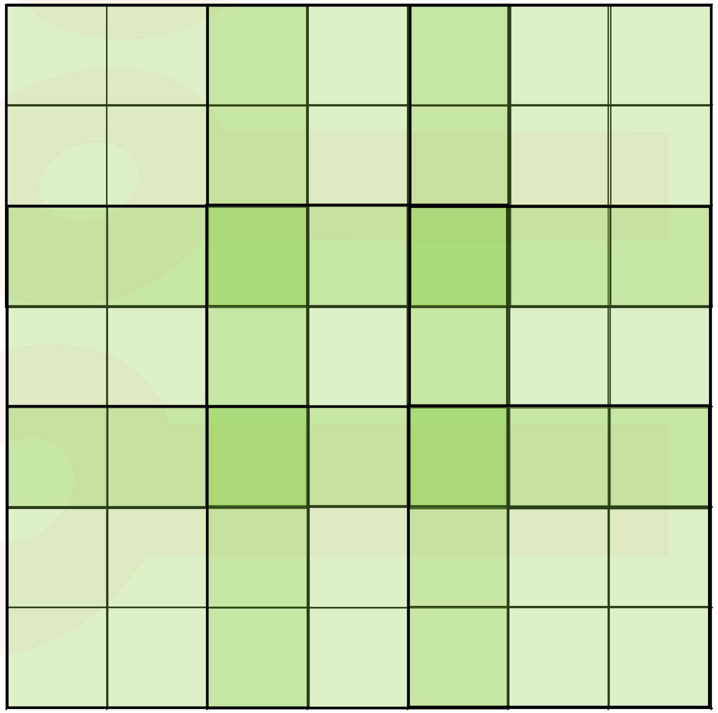


7x7x1 Activation Map

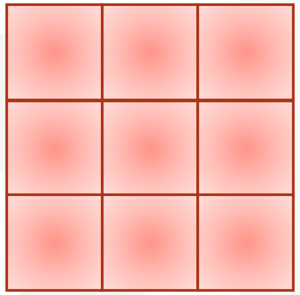


Use Stride to Downsample

- **Stride is the distance between convolutional steps**
  - For example, stride = 2



3x3x1 Activation Map



Example

- **Input image: 32x32x3**
- **10 filters of 5x5**
- **Stride=1, No padding**
- **First question:**
  - What is the output size?

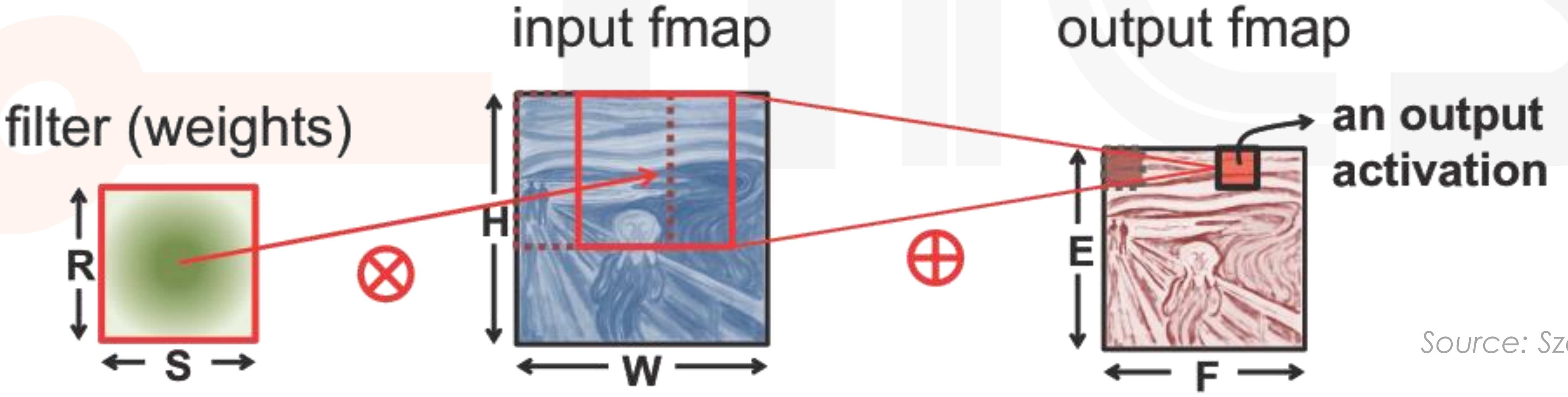
Output maps are 28x28
10 filters, so 10 output maps
Total size: 28x28x10

- **Second question:**
  - How many parameters (weights) are in this layer?

Each filter: 5x5x3 = 75 weights
+1 bias → 76 parameters
10 filters, so 760 parameters

Counting the size of a Conv Layer

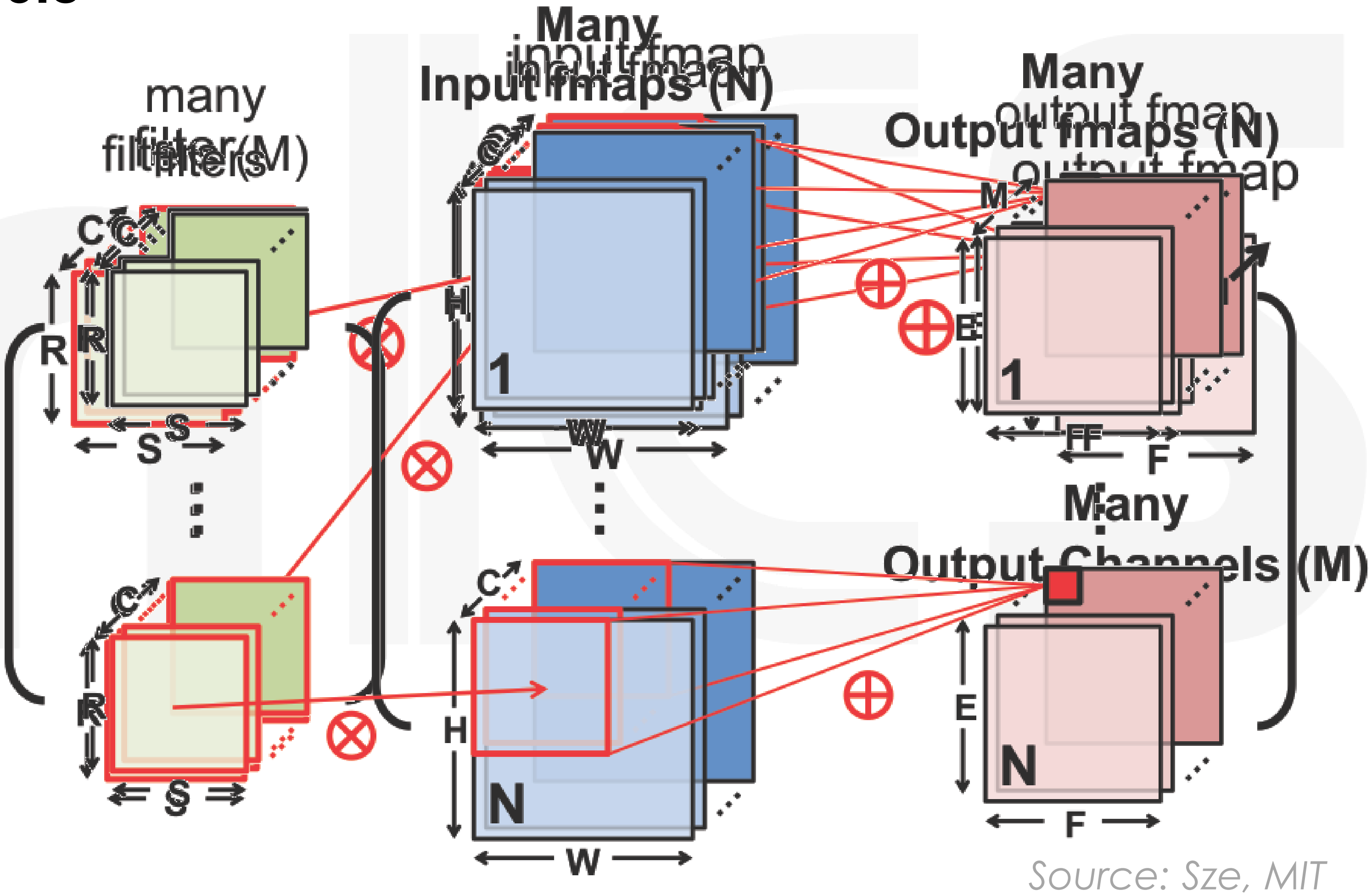
- **Start with a (2-D) WxH input feature map (fmap)**
- **Apply convolution with an SxR filter (weights)**
  - Each step is SxR multiplications and additions (MACs)
- **The output map is of size ExF**
  - So a total of (SxR) x (ExF) MACs



Source: Sze, MIT

Counting the size of a Conv Layer

- But, there can be C input channels
  - RxSxC weights
  - (RxSxC)x(ExF) MACs
- And there can be M filters
  - MxRxSxC weights
  - (MxRxSxC)x(ExF) MACs
- And this can be applied to a batch of N inputs
  - MxRxSxC weights
  - Nx(MxRxSxC)x(ExF) MACs
  - Nx(MxExF) stored outputs



Source: Sze, MIT

Let's summarize the parameters…

- **N** – Number of input fmaps/output fmaps (batch size)
- **C** – Number of 2-D input fmaps /filters (channels)
- **H,W** – Height/Width of input fmap (activations)
- **R,S** – Height/Width of 2-D filter (weights)
- **M** – Number of 2-D output fmaps (channels)
- **E,F** – Height/Width of output fmap (activations)

Output fmaps (O)

Biases (B)

Input fmaps (I)

Filter weights (W)

Stride Size (U)

Source: Sze, MIT

$$\underline{\mathbf{O}[n][m][x][y]} = \text{Activation}(\underline{\mathbf{B}[m]} + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \underline{\mathbf{I}[n][k][Ux+i][Uy+j]} \times \underline{\mathbf{W}[m][k][i][j]}),$$

Convolutional Layer Implementation

- **Naïve 7-layer for-loop implementation:**

Output fmaps (O)
Biases (B)
Input fmaps (I)
Filter weights (W)

$$\mathbf{O}[n][m][x][y] = \text{Activation}(\mathbf{B}[m] + \sum_{i=0}^{R-1}\sum_{j=0}^{S-1}\sum_{k=0}^{C-1} \mathbf{I}[n][k][Ux+i][Uy+j] \times \mathbf{W}[m][k][i][j]),$$

for each output fmap value

convolve a window and apply activation

```
for (n=0; n<N; n++) {
    for (m=0; m<M; m++) {
        for (x=0; x<F; x++) {
            for (y=0; y<E; y++) {

                O[n][m][x][y] = B[m];
                for (i=0; i<R; i++) {
                    for (j=0; j<S; j++) {
                        for (k=0; k<C; k++) {
                            O[n][m][x][y] += I[n][k][Ux+i][Uy+j] × W[m][k][i][j];
                        }
                    }
                }

                O[n][m][x][y] = Activation(O[n][m][x][y]);
            }
        }
    }
}
```

Source: Sze, MIT

Additional CNN Components

Additional Components: Pooling layer

- **Stride>1 reduces the size of the activation maps, but skips part of the input.**
  - Instead (or in addition), use a pooling layer
- **Pooling layers apply a function to a window of the activation map.**
  - Pooling is applied to *each channel separately*.
  - For example, max pooling 2x2

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

2x2 Output

6	8
3	4

Additional Components: Batch Normalization

- Normalize activations towards mean=0 and std. dev.=1 based on the statistics of the training dataset
- Calculated per channel in a layer.
- put in between CONV/FC and Activation function

data mean

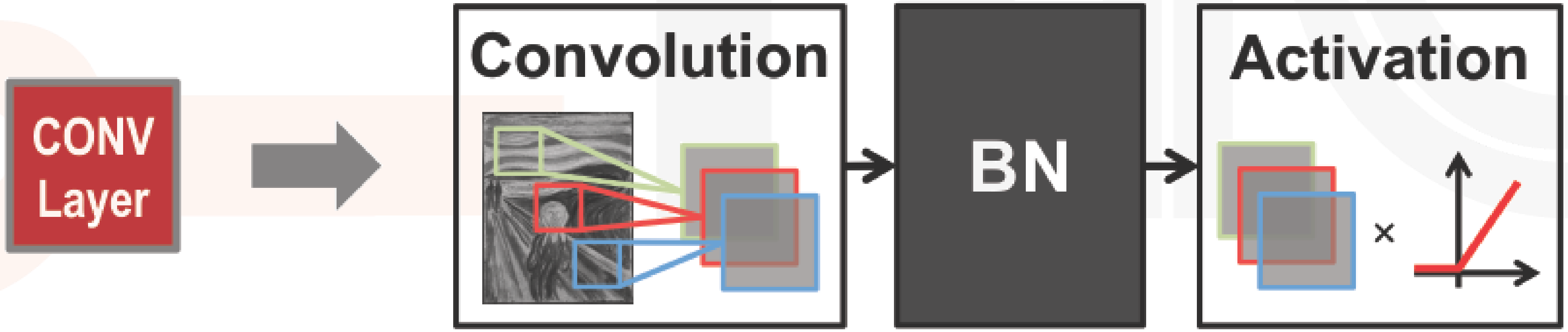
learned scale factor

$$y = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}\gamma + \beta$$

learned shift factor

data std. dev.

small const. to avoid numerical problems

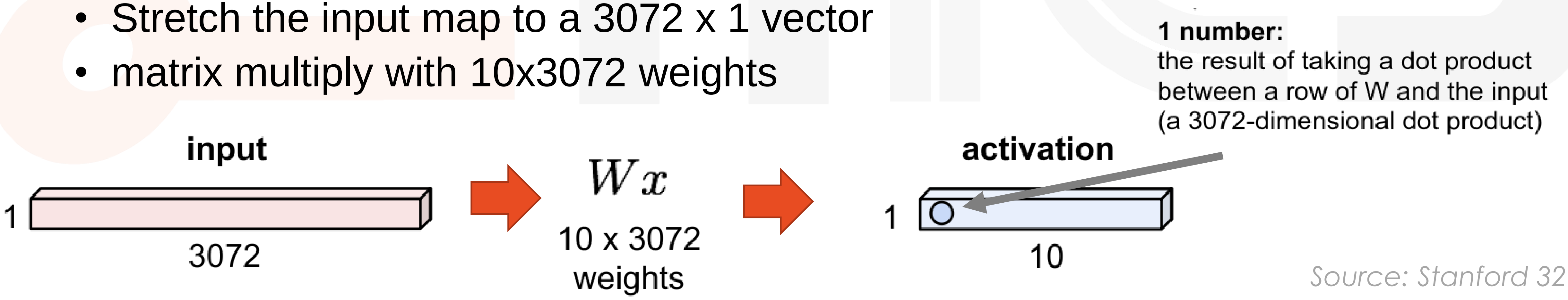


Source: Sze, MIT

- Believed to be key to getting faster training and high accuracy on very deep neural networks.

Fully-Connected (FC) Layer

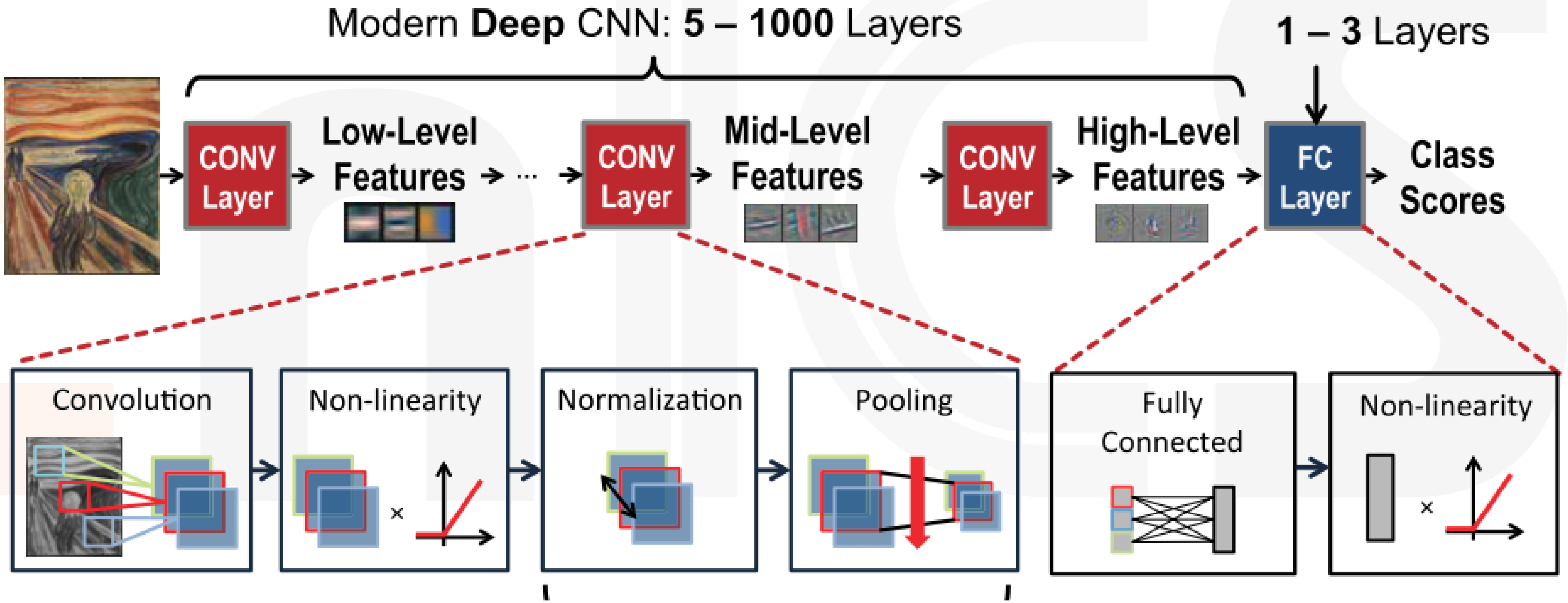
- **Height and width of output fmaps are 1 (E = F = 1)**
- **Filters as large as input fmaps (R = H, S = W)**
- **Implementation: Matrix Multiplication**
- **Example:**
  - 32x32x3 input
  - 10 output categories
  - Stretch the input map to a 3072 x 1 vector
  - matrix multiply with 10x3072 weights



Source: Stanford 321n

Putting it all together

- **Drive the input through**
  - Convolution Layers (filters)
  - Activations
  - Pooling Layers
  - Normalization



- **And finally use a fully connected layer to provide output scores**

Source: Sze, MIT

Famous CNNs

And some famous examples: **LeNet-5**

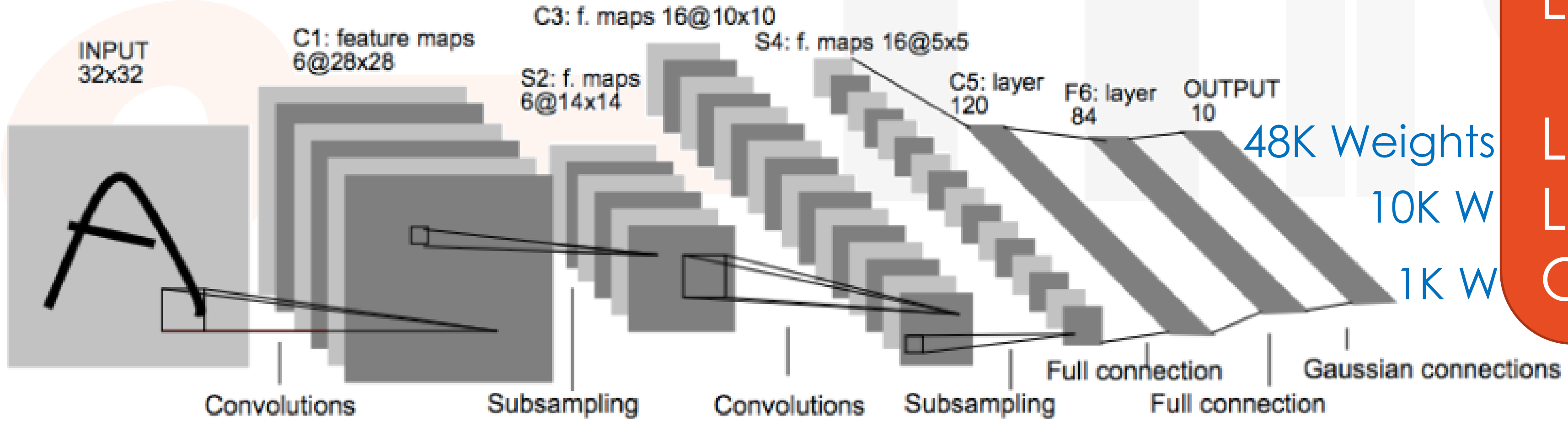
- **Digit classification (MNIST Dataset)**
  - 2xCONV, 2xPOOL, 2xFC
  - Sigmoid activation
  - 60K Weights
  - 340K MACs
  - Used in ATMs for checks

156 Weights
122K MACs

2416 Weights
151K MACs

48K Weights
10K W
1K W

Input: 32x32x1
L1: CONV: 6x(5x5x1)
ofmaps: (28x28)x6
L2: 2x2 POOL:
ofmaps: (14x14)x6
L3: CONV: 16x(5x5x6)
ofmaps: (10x10)x16
L4: 2x2 POOL:
ofmaps: (5x5)x16
L5: FC – 120 Neurons
L6: FC – 84 Neurons
Output: 10 categories



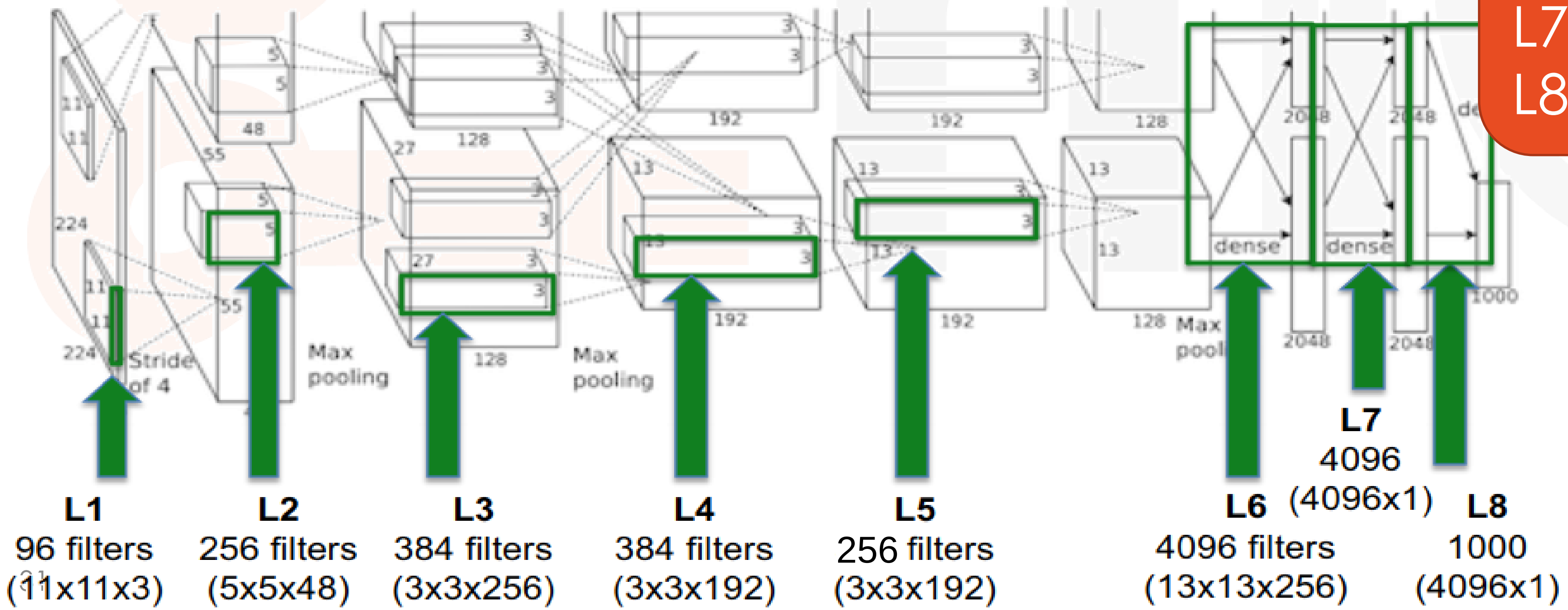
six 5x5 filters
2x2 average pooling
sixteen 5x5 filters
2x2 average pooling

Source: Lecun 1998
Calculation: engmrk.com/lenet-5-a-classic-cnn-architecture/

And some famous examples: **AlexNet**

- **AlexNet: ILSCVR Winner 2012** (16.4% Top-5 error)
  - 5xCONV, 3xFC, various stride, filter sizes
  - 61M Weights, 724M MACs
  - ReLu activation, LRN Normalization, SoftMax out
  - Total: 61M Weights, 724M MACs

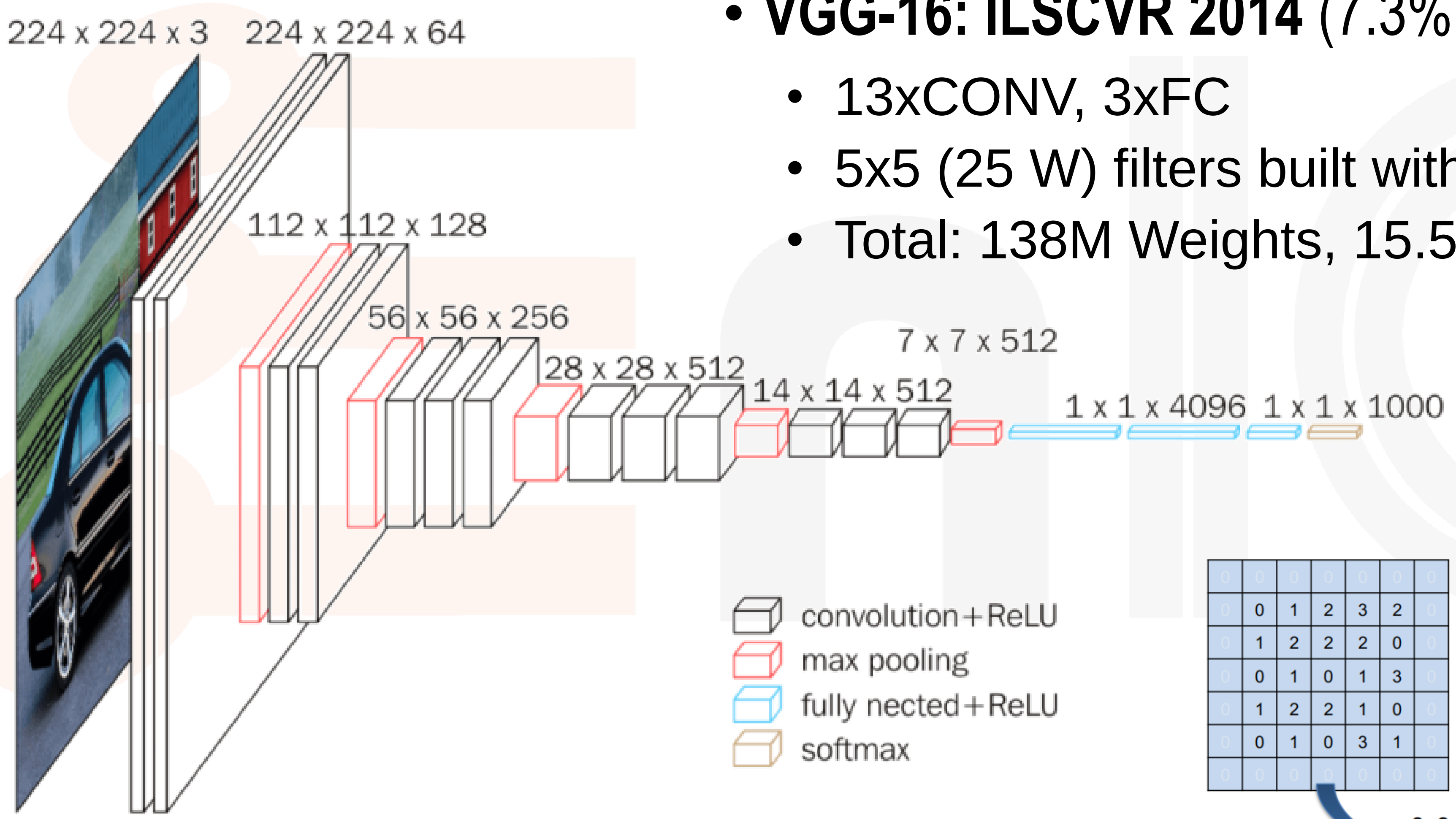
L1: 34K Wgt, 105M MACs
L2: 307K Wgt, 224M MACs
L3: 885K Wgt, 150M MACs
L4: 664K Wgt, 112M MACs
L5: 442K Wgt, 75M MACs
L6: 38M Wgt/MACs
L7: 17M Wgt/MACs
L8: 4M Wgt/MACs

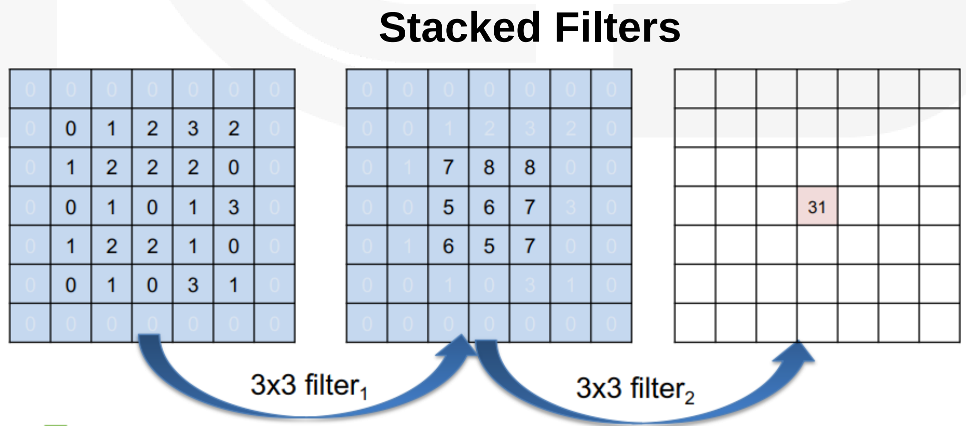


Source: Krizhevsky, 2012

And some famous examples: **VGG-16**

- **VGG-16: ILSCVR 2014** (7.3% Top-5 Error)
  - 13xCONV, 3xFC
  - 5x5 (25 W) filters built with 2 stacked 3x3 (18 W)
  - Total: 138M Weights, 15.5G MACs





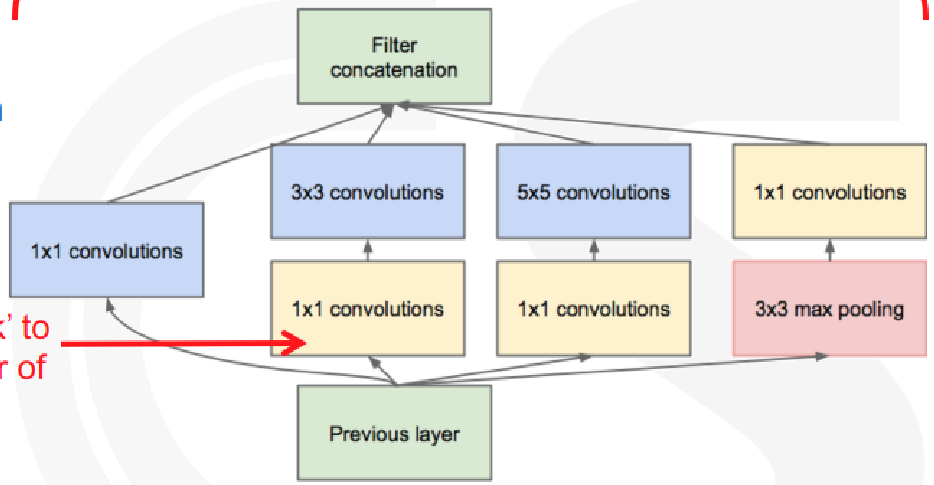
Source: https://neurohive.io/en/popular-networks/vgg16/

And some famous examples: **GoogLeNet**

- **GoogLeNet (Inception): ILSCVR Winner 2014** (6.7%)
  - 57 Layers (21 layers deep), 1xFC
  - Total: 7M Weights, 1.43G MACs
  - Inception modules, 1x1 bottlenecks
  - Entire CNN can be trained on one GPU

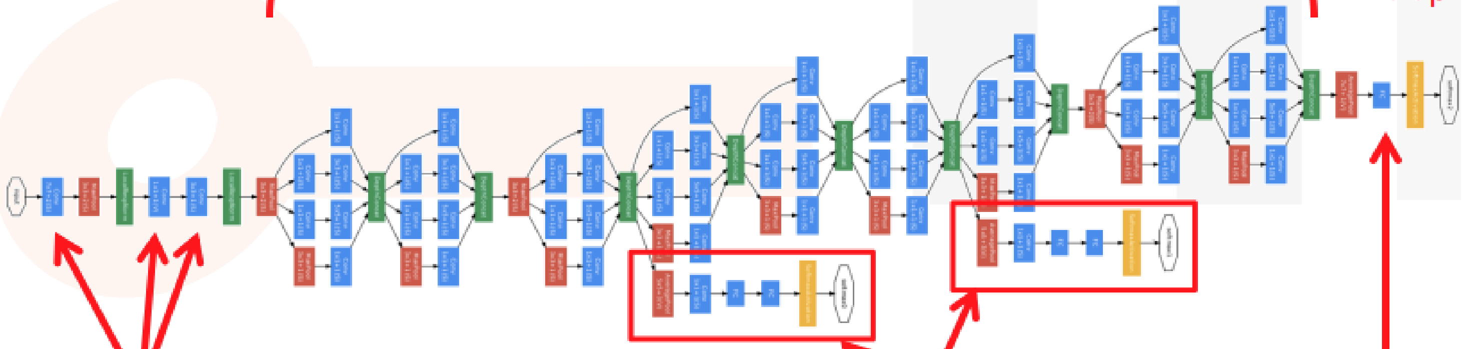
parallel filters of different size have the effect of processing image at different scales

Inception Module



1x1 'bottleneck' to reduce number of weights and multiplications

9 Inception Layers



3 CONV layers

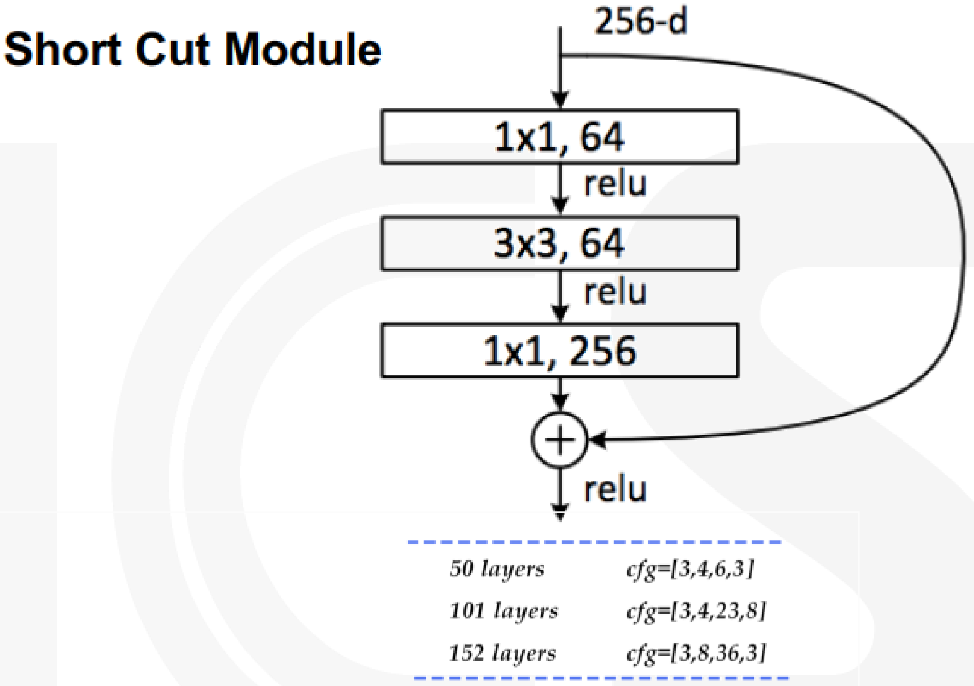
Auxiliary Classifiers
(helps with training, not used during inference)

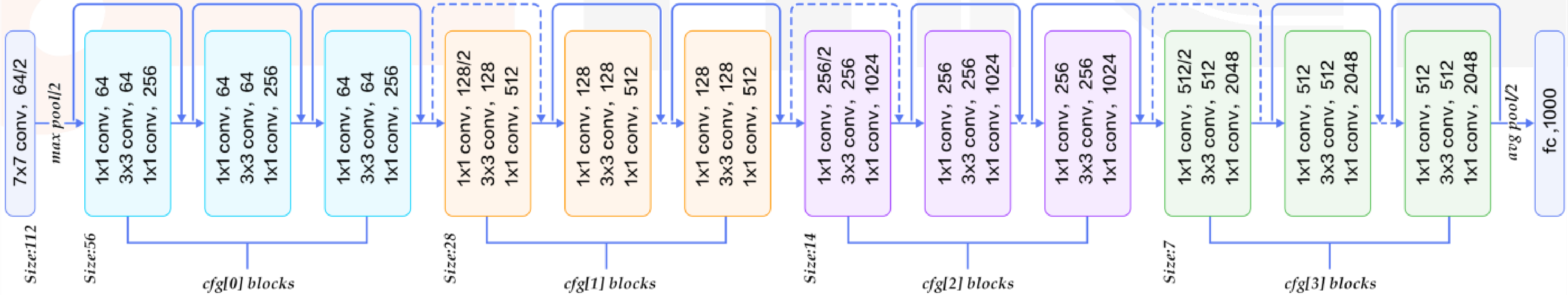
1 FC layer
(reduced from 3)

Source: Szegedy, 2015

And some famous examples: **ResNet**

- **ResNet-50: ILSCVR Winner 2015** (3.57% Error)
  - Better than human level accuracy!
  - 49xCONV, 1xFC
  - Total: 25.5M Weights, 3.9G MACs
  - Short Cut Modules (skip connections)





Main References

- Stanford C231n, 2017
- Sze, et al. “Efficient Processing of Deep Neural Networks: A Tutorial and Survey”, Proceedings of the IEEE, 2017
- Sze, et al. ISCA Tutorial 2019