

TOCI: DEEP LEARNING

Convolutional Neural Networks

An overview and applications

Ms. Farzeen Ashfaq

Outline

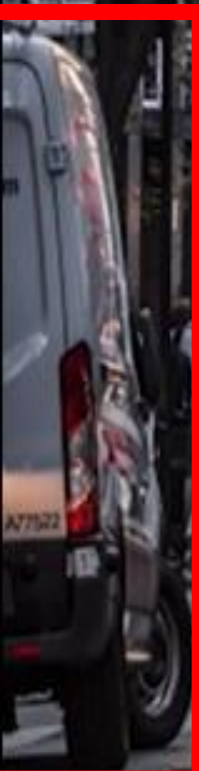
- Overview of Convolutional Neural Networks
- The Convolution operation
- A typical CNN model architecture
- Properties of CNN models
- Applications of CNN models
- Notable CNN models
- Limitations of pure CNN models

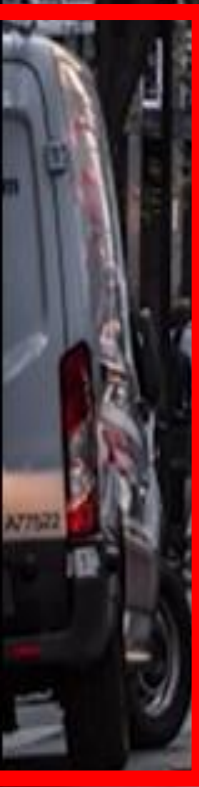
- Hands-on CNN-supported image classification













What computers See



157	153	174	168	150	182	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

157	153	174	168	150	182	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	106	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers $[0,255]$!
i.e., $1080 \times 1080 \times 3$ for an RGB image

High Level Features



High Level Features



Nose,
Eyes,
Mouth

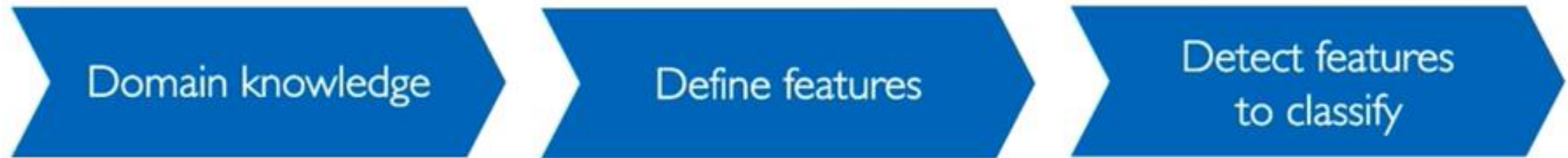


Wheels,
License Plate,
Headlights



Door,
Windows,
Steps

Manual Feature Extraction



PROBLEMS??

Manual Feature Extraction

Domain knowledge

Define features

Detect features
to classify

Viewpoint variation



Scale variation



Deformation



Occlusion



Illumination conditions



Background clutter



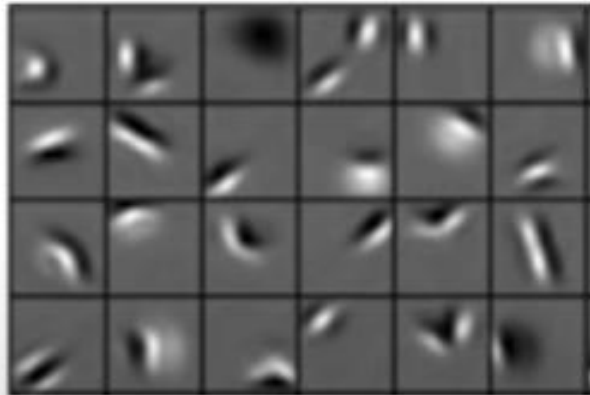
Intra-class variation



Thanks to Neural Networks

Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



Eyes, ears, nose

High level features

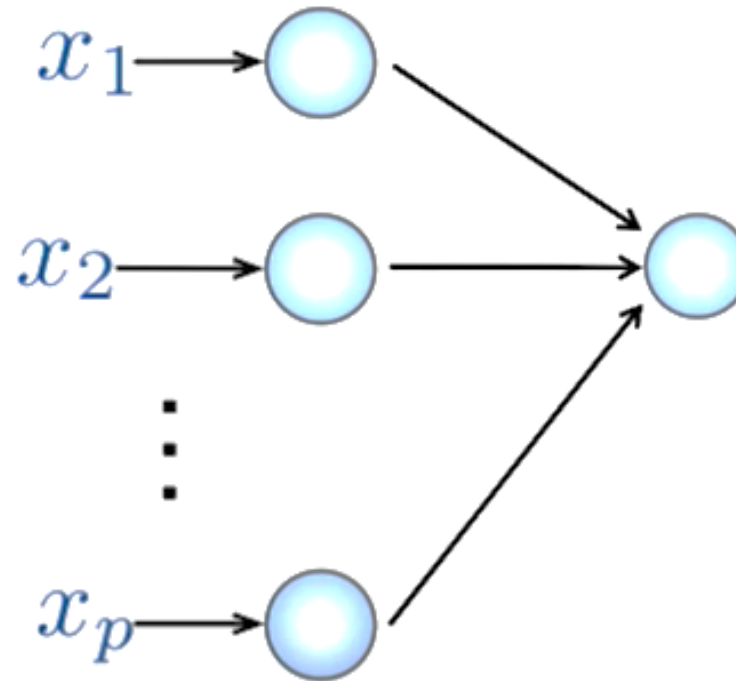


Facial structure

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values

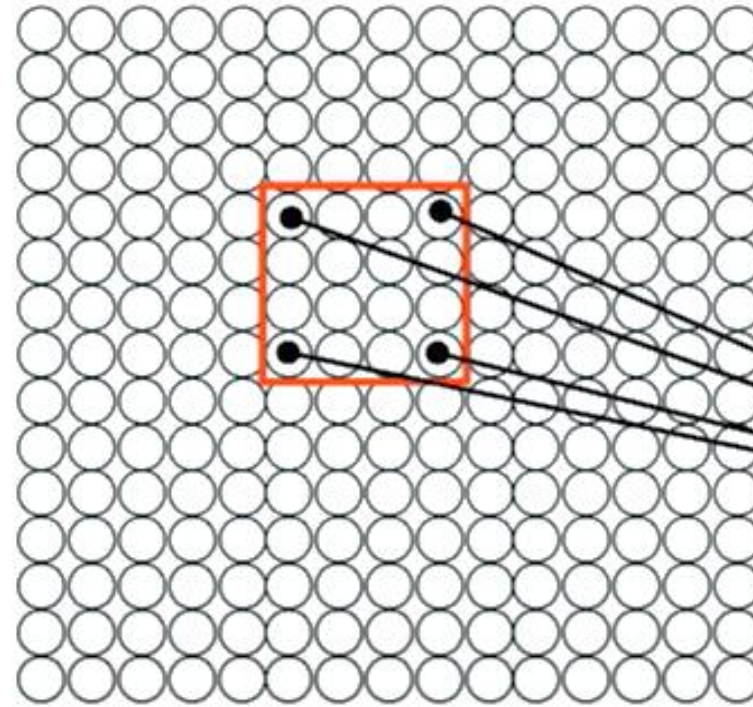


Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Using Spatial Structures

Input: 2D image.
Array of pixel values



Idea: connect patches of input
to neurons in hidden layer.
Neuron connected to region of
input. Only "sees" these values.

Pioneer in CNN



Yann LeCun

Received his PhD from MIT in 1986

Worked at AT&T Bell Labs

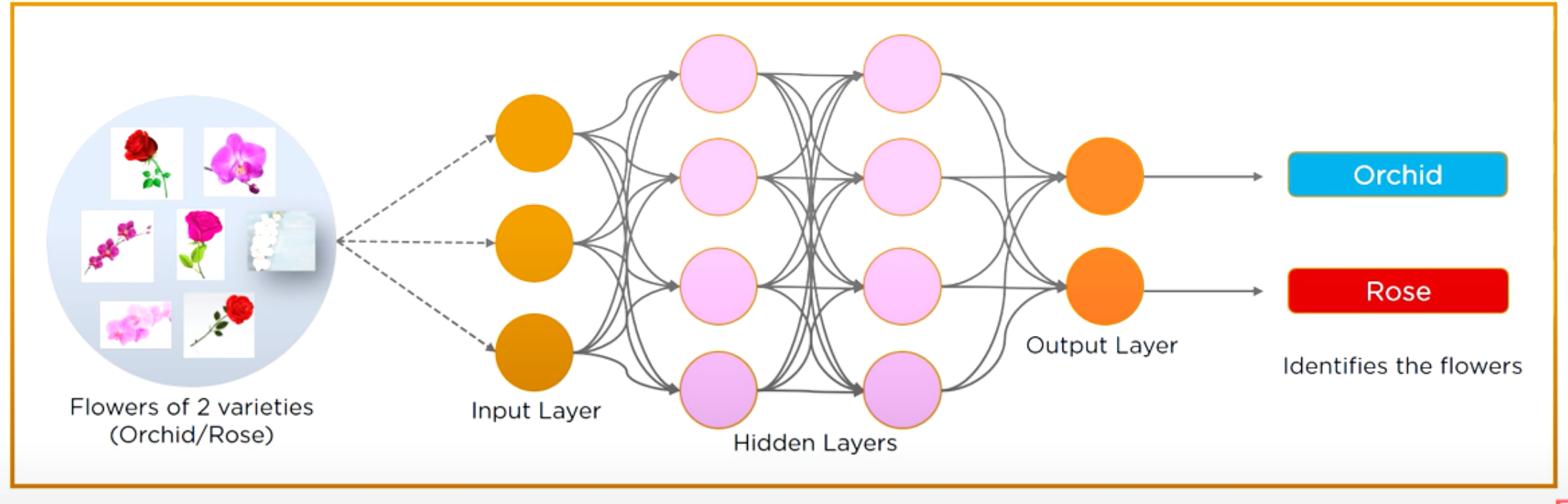
Built the first Convolution Neural Network called LeNet in 1988

Convolutional Neural Networks¹

Convolutional Neural Networks (CNNs) are neural networks that use *convolution* in place of general matrix multiplication in at least one of their layers

- they are usually employed for processing data that has a grid-like topology
 - image data → each image can be thought as a 2D grid of pixels
 - timeseries data → each timeseries can be thought as a 1D grid taking samples at regular time intervals

What is a CNN?



Layers in CNN

- Convolution Layer
- ReLU Layer
- Pooling Layer
- Fully Connected Layer

Simple Convolution Operation

$a = [5, 3, 7, 5, 9, 7]$
 $b = [1, 2, 3]$

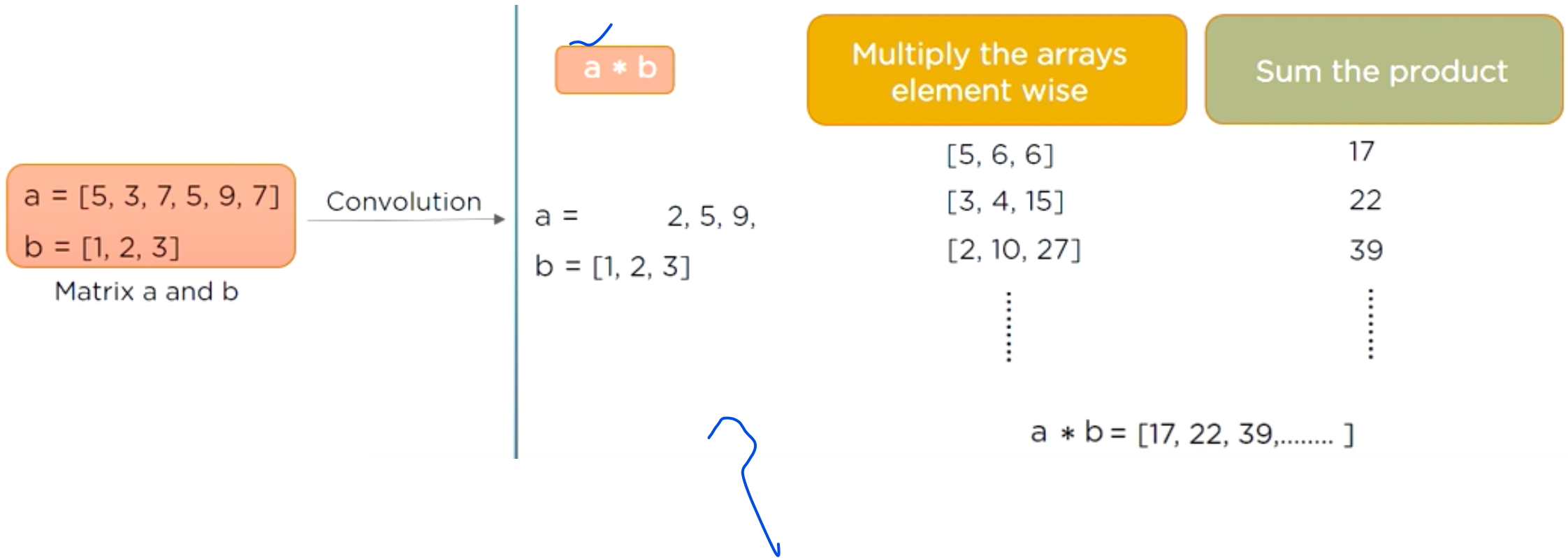
Matrix a and b

Convolution

$a * b$

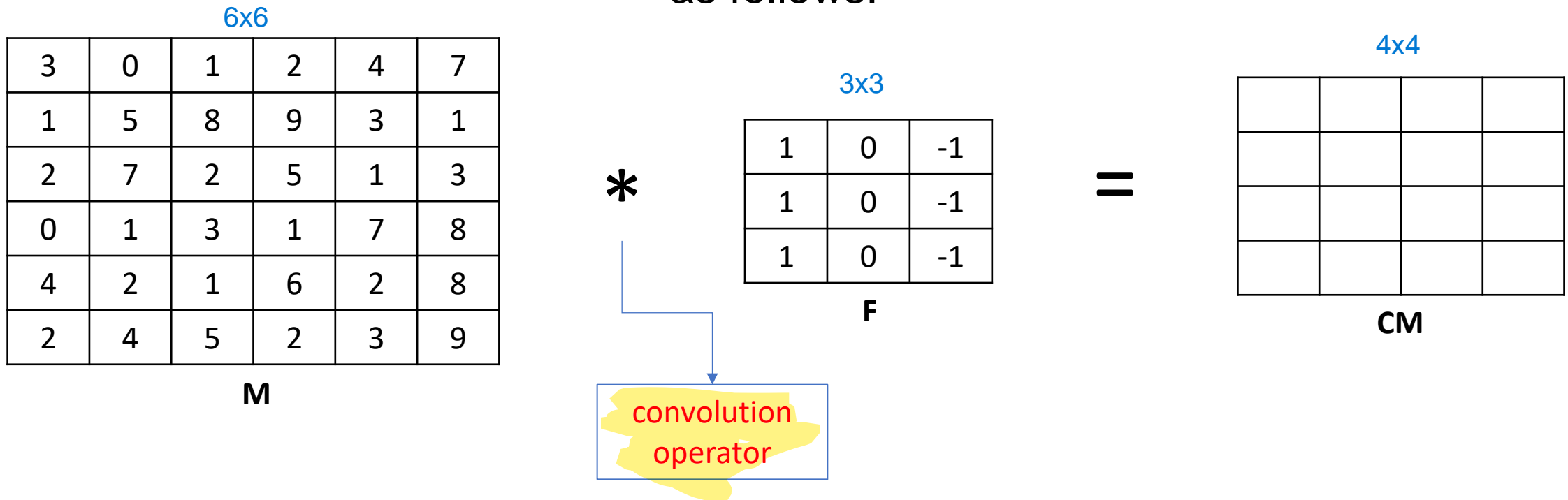
$a = [5, 3, 2, 5, 9, 7]$
 $b = [1, 2, 3]$

Simple Convolution Operation



The Convolution operation¹

Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride 1* is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



The Convolution operation¹

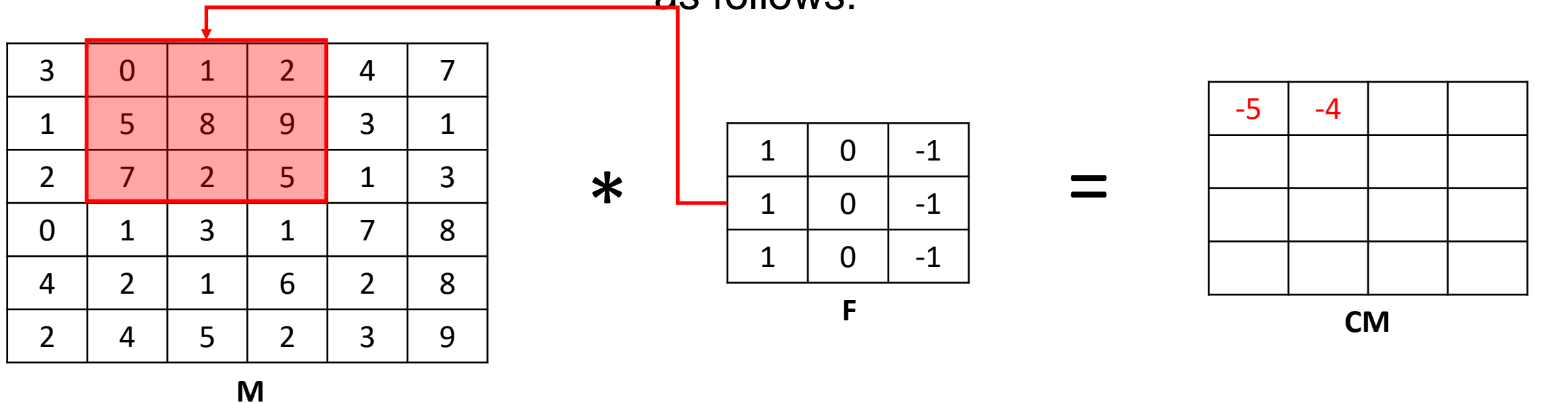
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:

The diagram illustrates the 2D convolution operation. It shows a 6x6 input matrix **M**, a 3x3 filter matrix **F**, and a 4x4 output matrix **CM**. A red box highlights a 3x3 region in **M** (top-left corner), and a red arrow points from this region to the filter **F**. The filter **F** is a 3x3 matrix with values 1, 0, -1 in each row. The output matrix **CM** is a 4x4 matrix with the top-left element highlighted in red as -5. Below the matrices, the calculation for the first element of **CM** is shown with blue brackets and numbers:

$$\underbrace{3*1 + 1*1 + 2*1}_{6} + \underbrace{0*0 + 5*0 + 7*0}_{0} + \underbrace{1*(-1) + 8*(-1) + 2*(-1)}_{-11} = -5$$

The Convolution operation¹

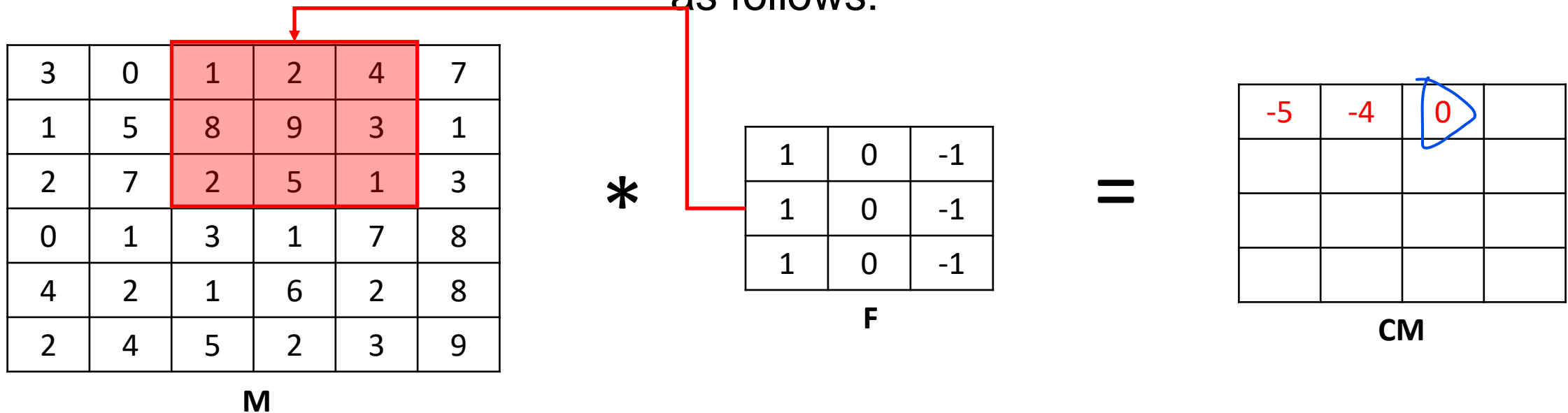
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$\underbrace{0*1 + 5*1 + 7*1}_{12} + \underbrace{1*0 + 8*0 + 2*0}_{0} + \underbrace{2*(-1) + 9*(-1) + 5*(-1)}_{-16} = -4$$

The Convolution operation¹

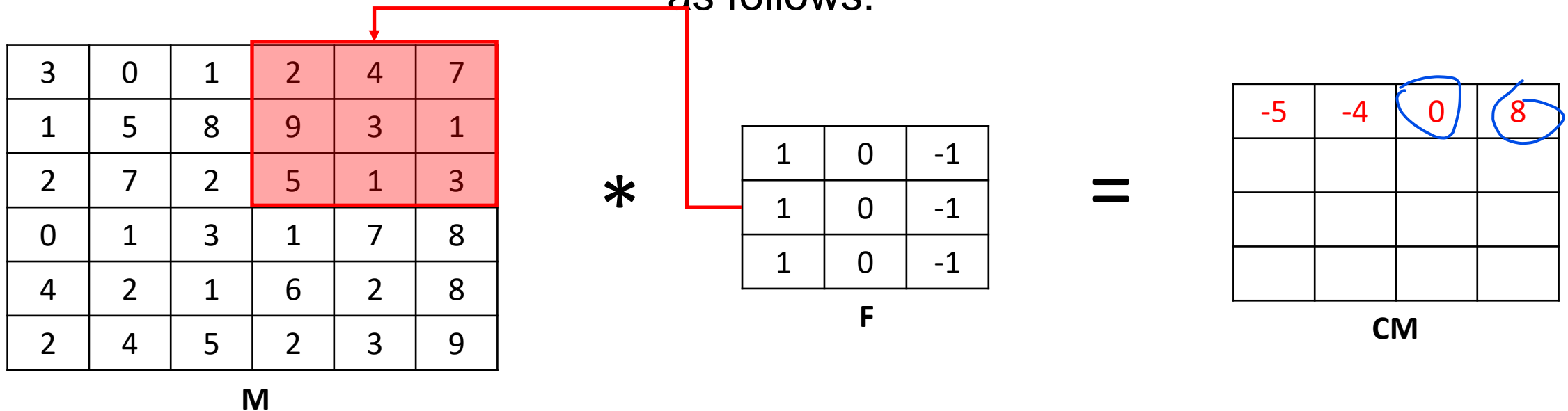
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$\underbrace{1*1 + 8*1 + 2*1}_{11} + \underbrace{2*0 + 9*0 + 5*0}_3 + \underbrace{4*(-1) + 3*(-1) + 1*(-1)}_{-8}$$

The Convolution operation¹

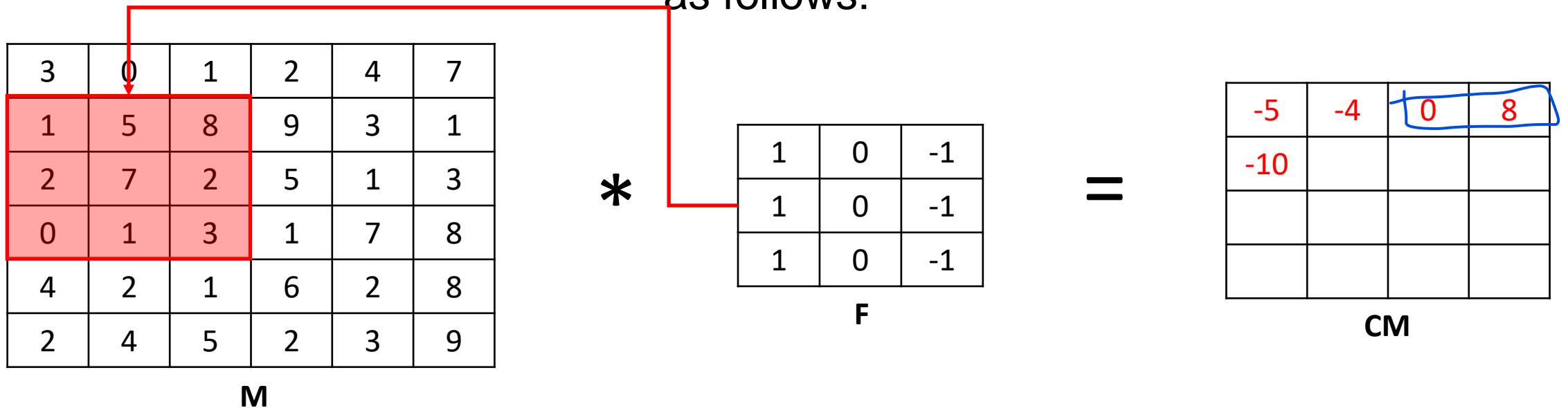
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$\underbrace{2*1 + 9*1 + 5*1}_{16} + \underbrace{4*0 + 3*0 + 1*0}_{0} + \underbrace{7*(-1) + 1*(-1) + 3*(-1)}_{-11}$$

The Convolution operation¹

Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:

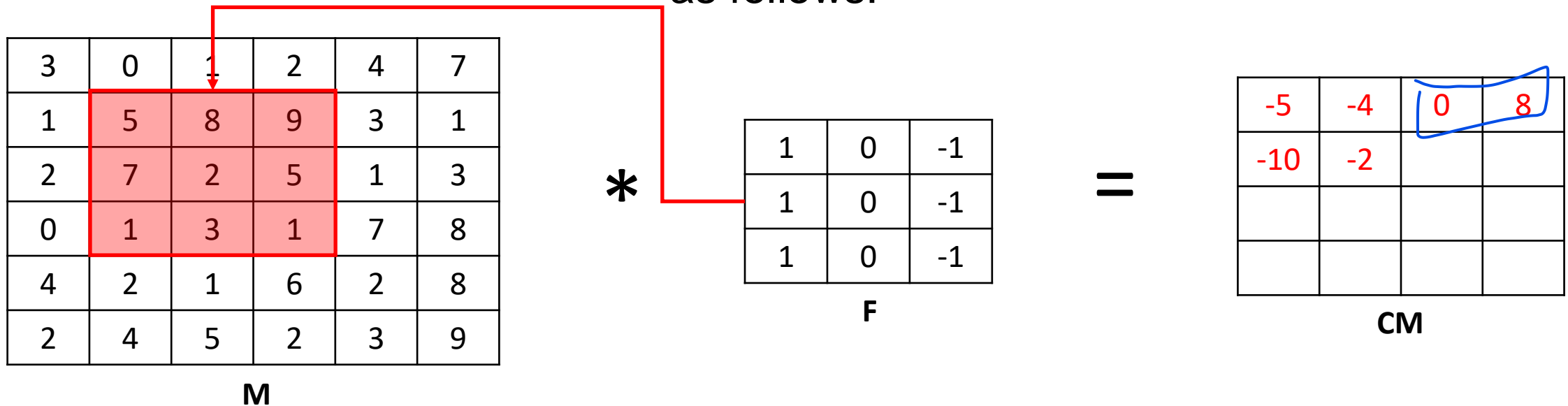


$$\underbrace{1*1 + 2*1 + 0*1}_{3} + \underbrace{5*0 + 7*0 + 1*0}_{-13} + \underbrace{8*(-1) + 2*(-1) + 3*(-1)}_{-10}$$

-10

The Convolution operation¹

Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



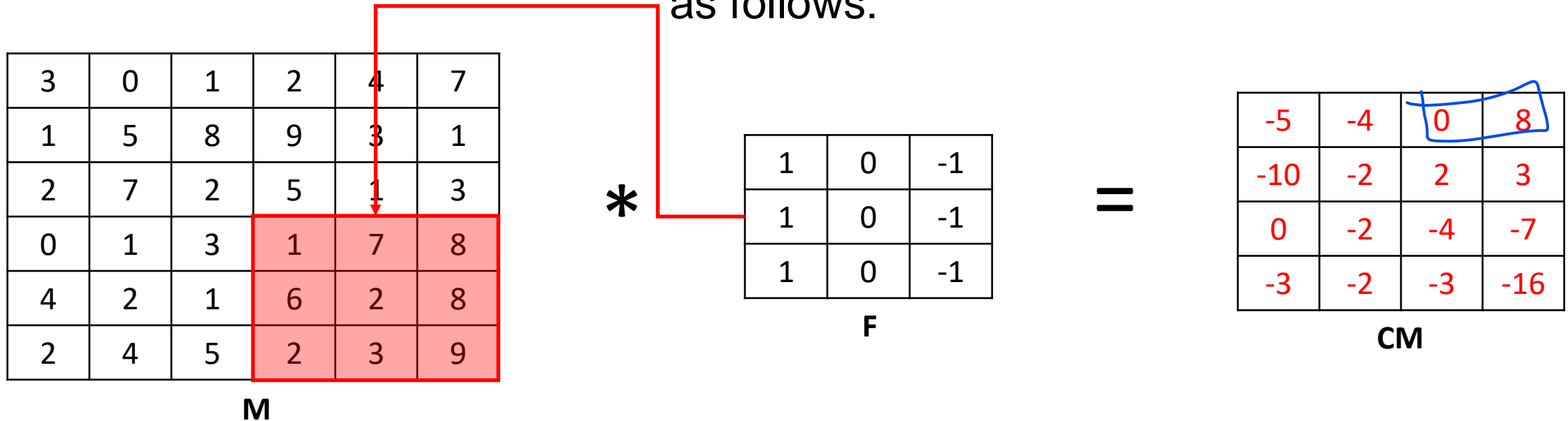
$$5*1 + 7*1 + 1*1 + 8*0 + 2*0 + 3*0 + 9*(-1) + 5*(-1) + 1*(-1)$$

13

-15

The Convolution operation¹

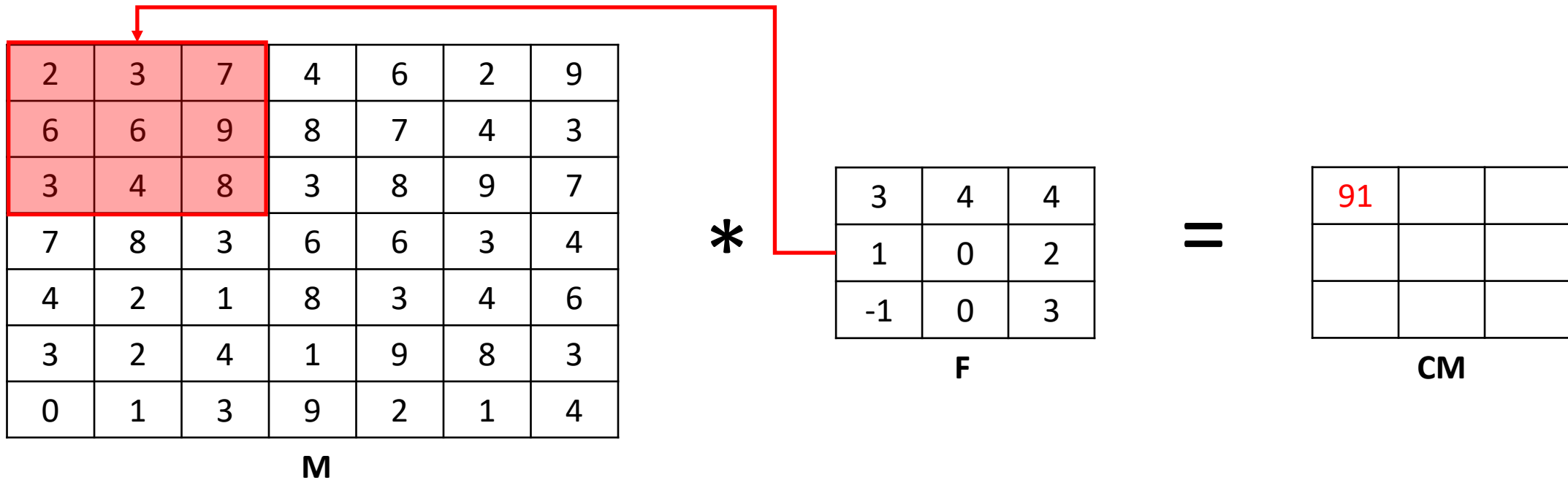
Assume a 6x6 matrix **M** as input. The 2D convolution of **M** with *filter (or kernel)* **F** and *stride* 1 is a 4x4 matrix **CM** (sometimes called *feature map*) computed as follows:



$$1*1 + 6*1 + 2*1 + 7*0 + 2*0 + 3*0 + 8*(-1) + 8*(-1) + 9*(-1)$$

Larger Strides

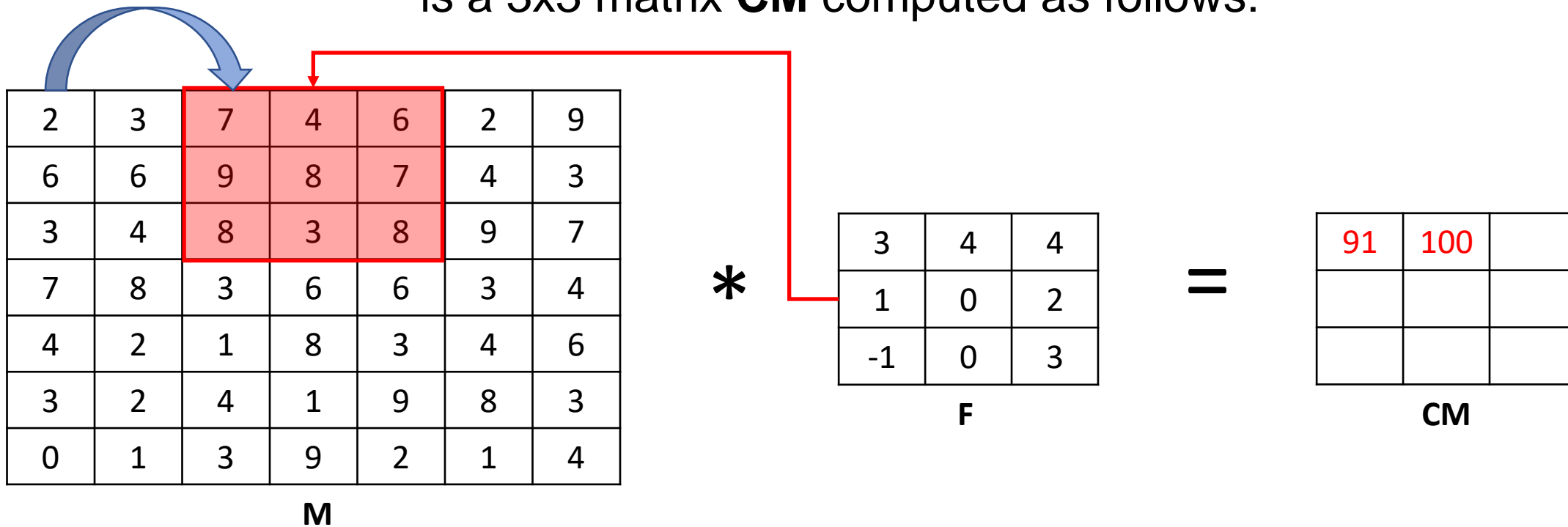
Assume a M, a 7x7 matrix. The 2D convolution of M with *filter* F and **stride 2** is a 3x3 matrix **CM** computed as follows:



$$\underbrace{2*3 + 6*1 + 3*(-1)}_{\text{Row 1}} + \underbrace{3*4 + 6*0 + 4*0}_{\text{Row 2}} + \underbrace{7*4 + 9*2 + 8*3}_{\text{Row 3}}$$

Larger Strides

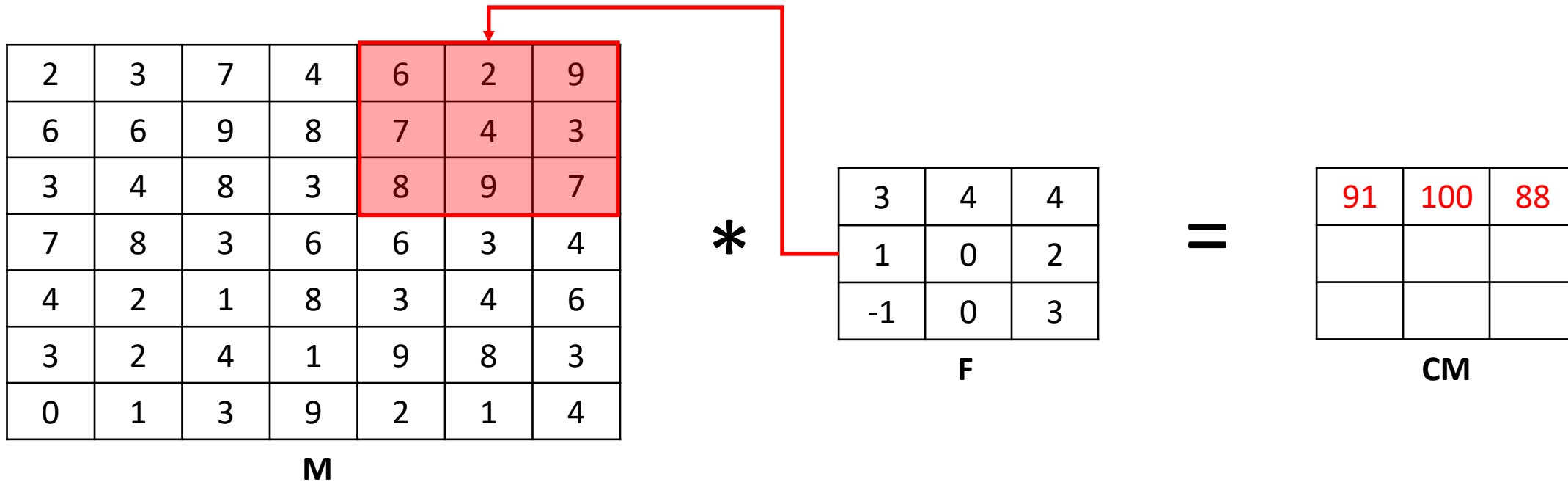
Assume a M , a 7×7 matrix. The 2D convolution of M with *filter* F and **stride 2** is a 3×3 matrix **CM** computed as follows:



$$7*3 + 9*1 + 8*(-1) + 4*4 + 8*0 + 3*0 + 6*4 + 7*2 + 8*3$$

Larger Strides

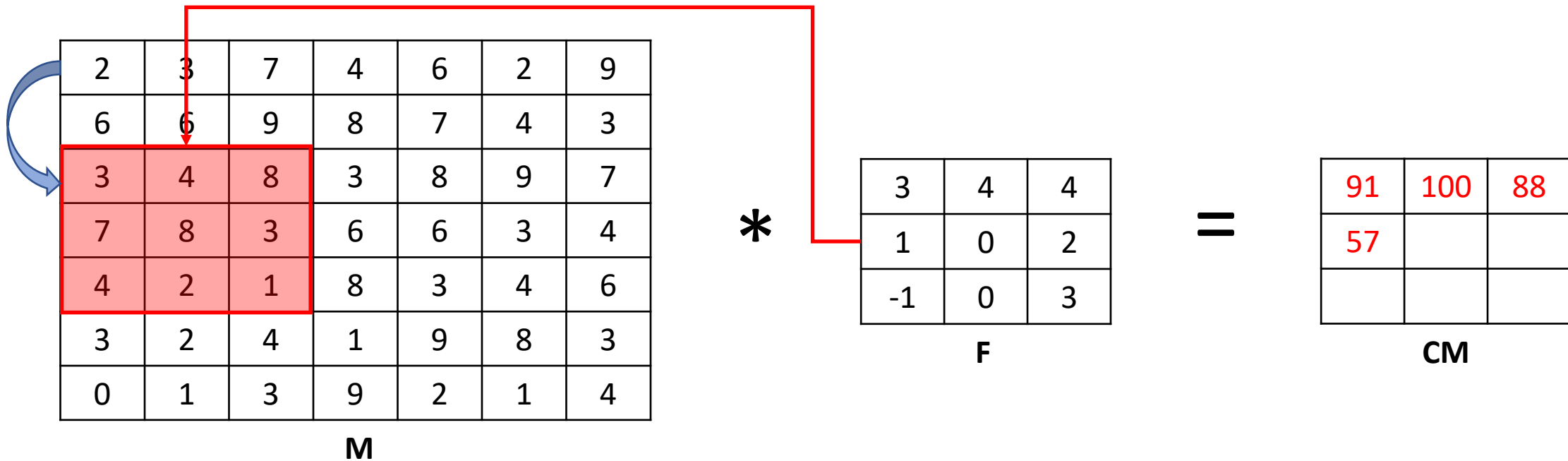
Assume a M , a 7×7 matrix. The 2D convolution of M with *filter* F and **stride 2** is a 3×3 matrix **CM** computed as follows:



$$6*3 + 7*1 + 8*(-1) + 2*4 + 4*0 + 9*0 + 9*4 + 3*2 + 7*3$$

Larger Strides

Assume a M, a 7x7 matrix. The 2D convolution of M with *filter* F and **stride 2** is a 3x3 matrix **CM** computed as follows:

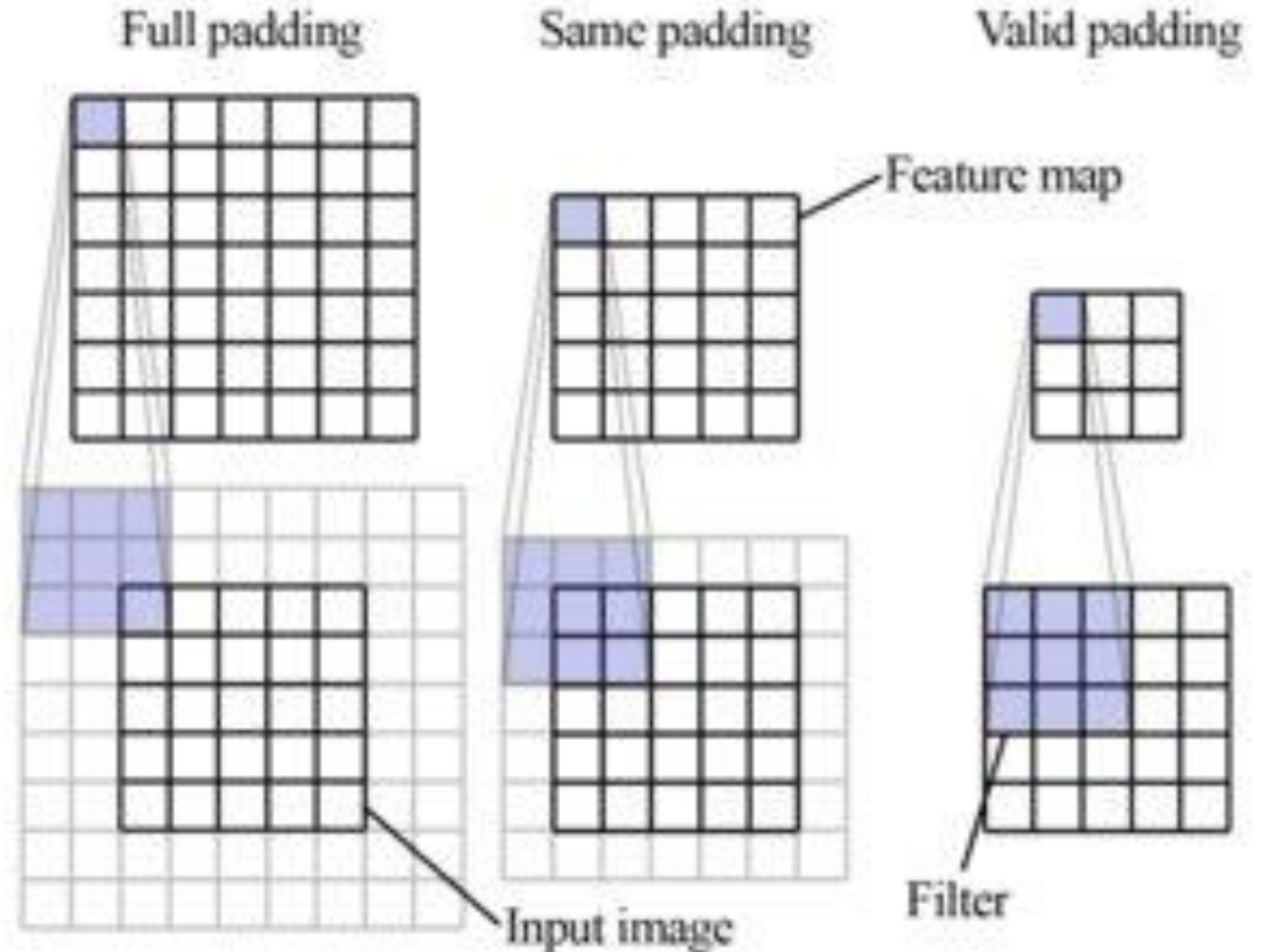


$$3*3 + 7*1 + 4*(-1) + 4*4 + 8*0 + 2*0 + 8*4 + 3*2 + 1*3$$

Padding^{1,4}





Usually, the convolution results in shrinking outputs and/or loss of information at the corners of the grid. We tackle that have the following options:

- **Full-padding**: for kernel of size k , $p = k-1$ where p is the number of zero rows/columns added
- **Same-padding**: for kernel of size k , $p = \text{ceil}(\frac{k}{2})$ where p is the number of zero rows/columns added
- **Valid-padding**: the convolution kernel is only allowed to visit positions where the entire kernel is contained entirely within the input



The Convolution operation on Images³

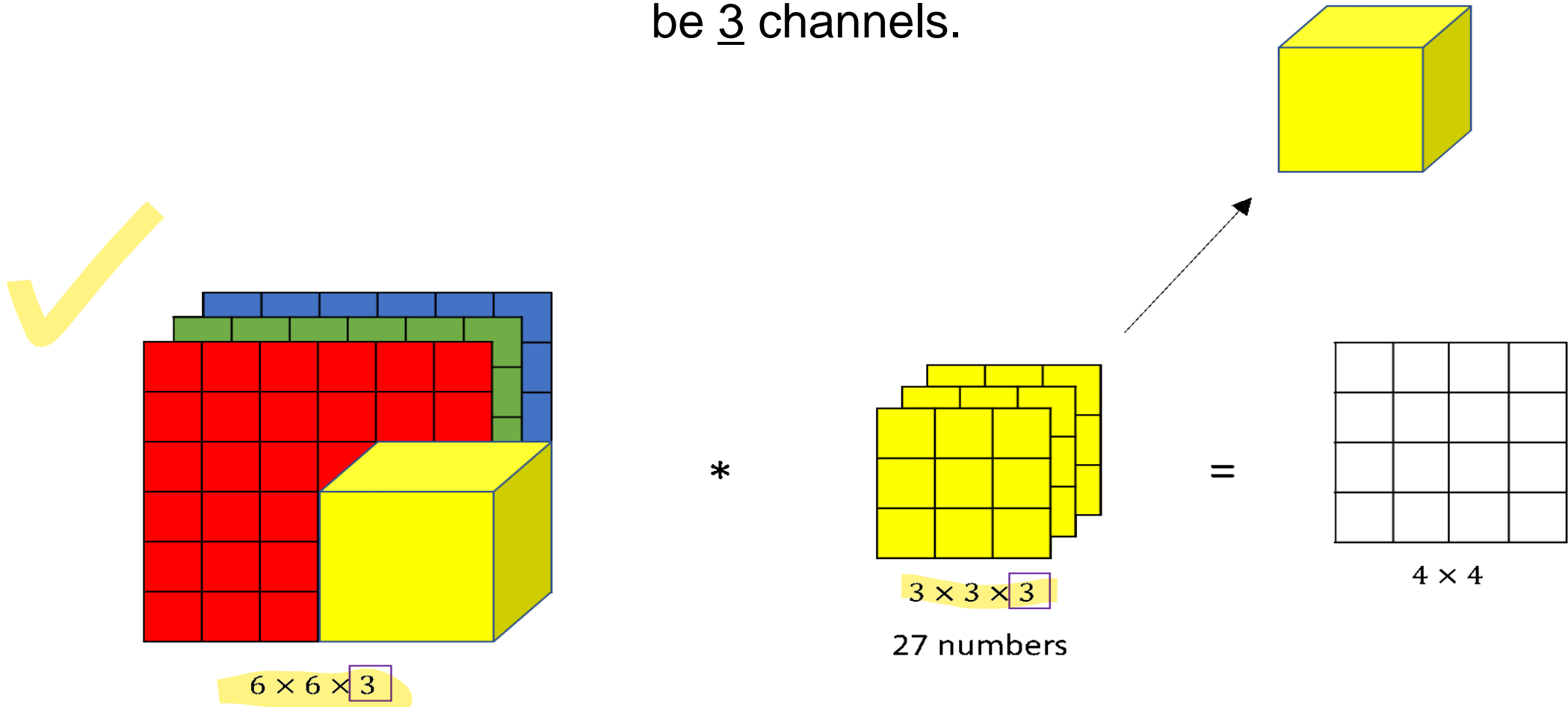
Convolutions are frequently used in image processing, to manipulate the image or detect different features in the image

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)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

The Convolution operation on RGB Images⁵

An RGB image is represented with a 3D tensor: $(x, y, 3)$, where x, y are the pixel dimensions and 3 corresponds to the 3 color channels. The filter will also be 3 channels.



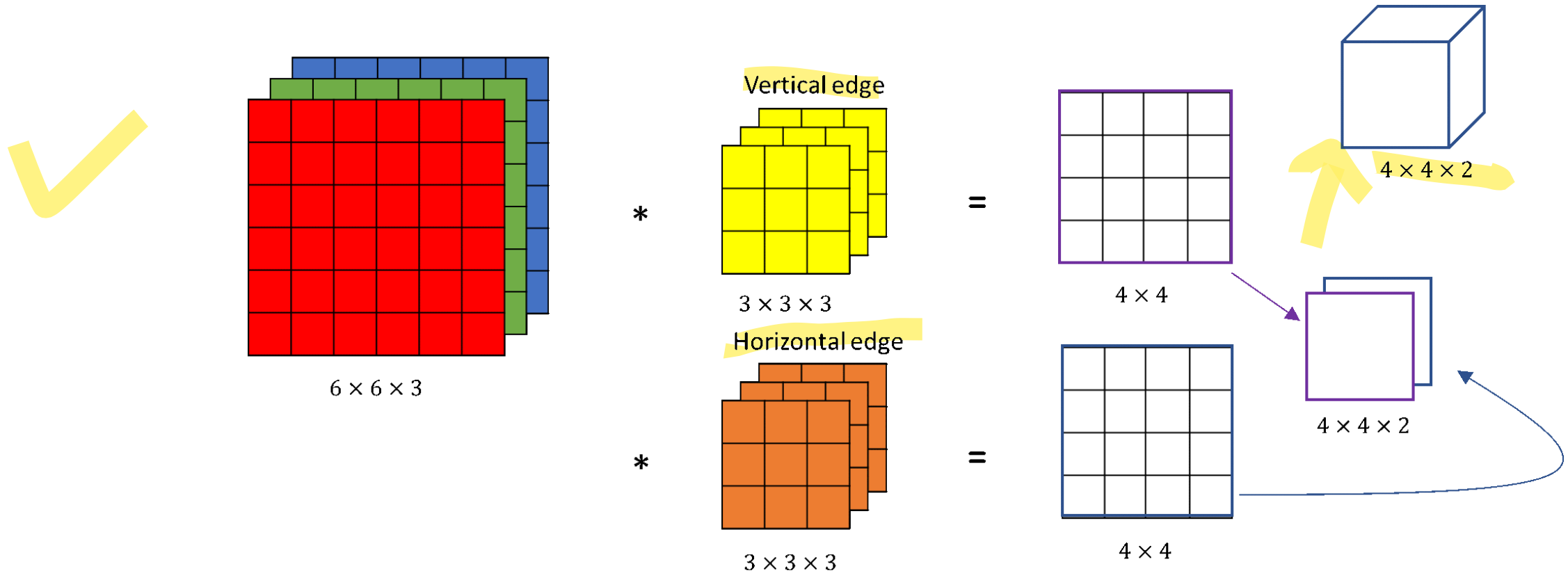
The Convolution operation on RGB Images⁵

The *channel* dimension in the kernel permits to detect features in only a subset of the dimensions, e.g.

- vertical edges only in the red dimension
 - set all cells in the blue and green channels to zero
- horizontal edges in the red and blue dimensions
- vertical edges no matter the color dimension

The Convolution operation on RGB Images⁵

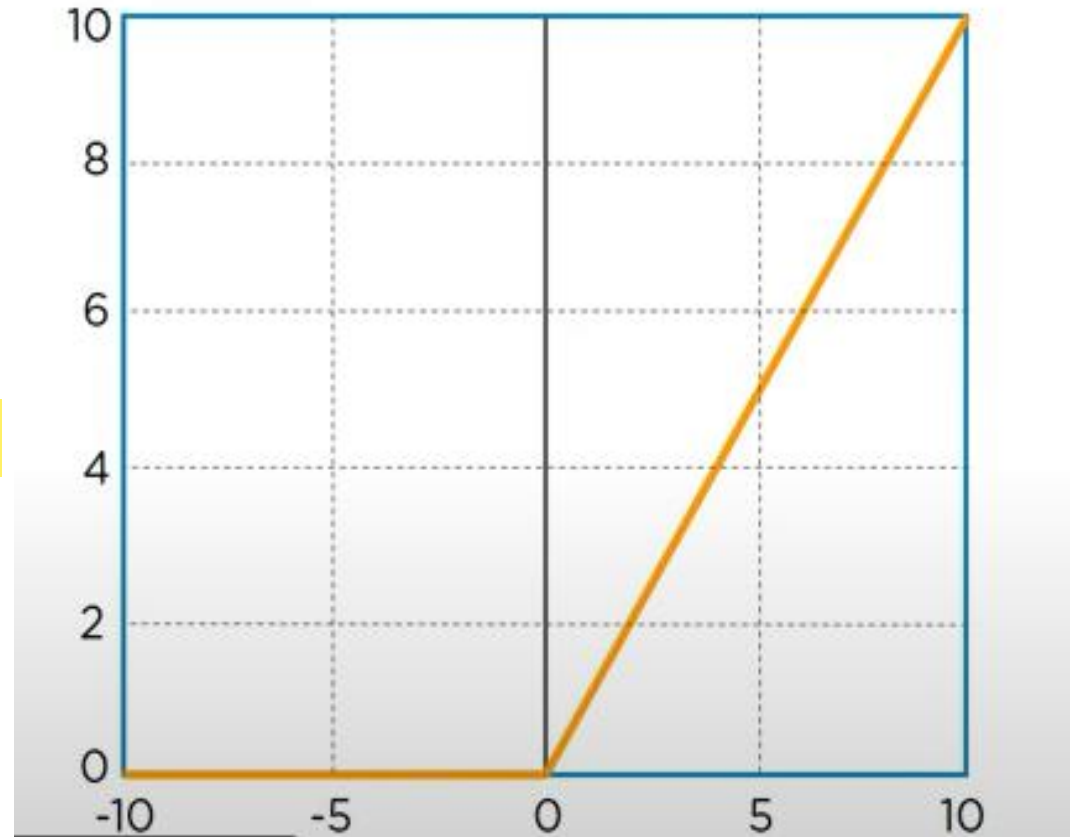
We can also detect different features at the same time, by employing multiple filters



The output will have a number of channels equal to the number of features we are trying to detect

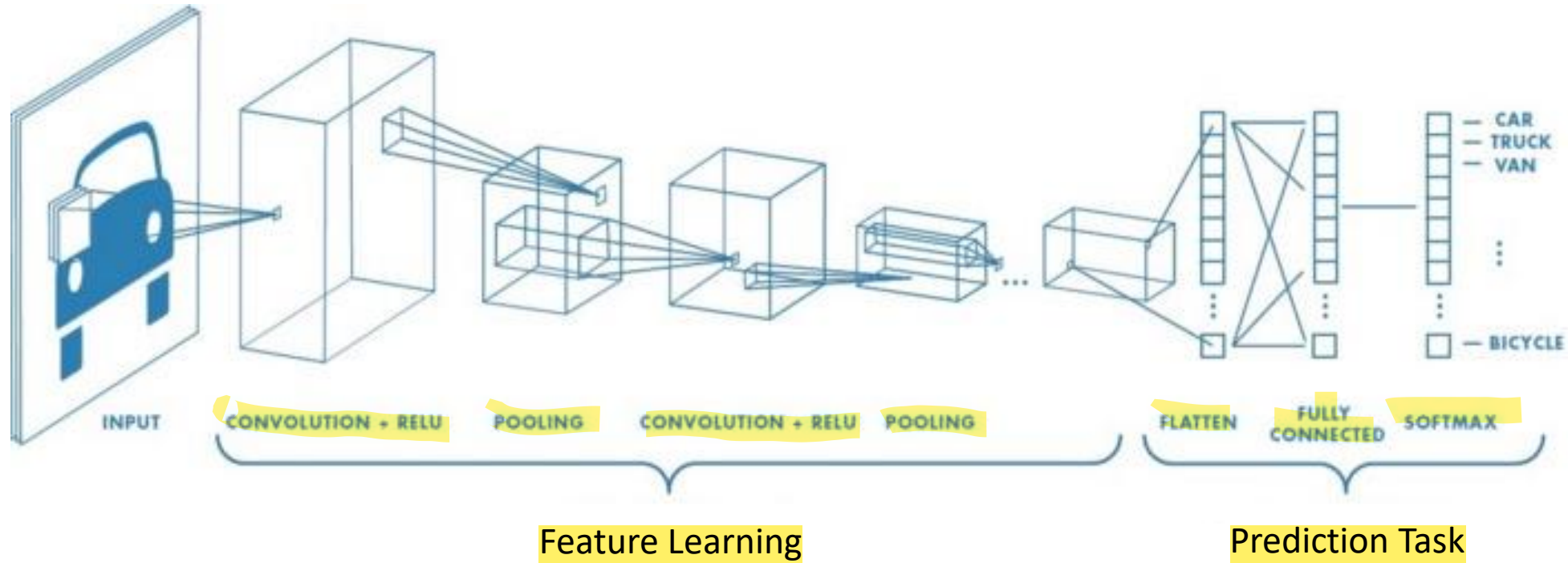
The ReLU Layer

- Performs element wise operation
- Sets all negative pixels to 0
- Introduce non-linearity to the network
- The output is a rectified linear map



Typical CNN model architecture³

Typically, a CNN model consists of convolution layers, for feature selection, followed by fully connected layers that perform the prediction task




Typical CNN model architecture¹

A typical feature learning layer of a convolutional network consists of three stages:

- the first stage performs several convolution operations in parallel to produce a set of linear activations
 - each convolution employs a different kernel to learn different features
 - the input is usually a grid of vector-valued observations
- in the second stage, each linear activation is run through a nonlinear activation function (e.g., ReLU)
- In the third stage, a *pooling function* is employed to modify the output of the layer further

Pooling¹

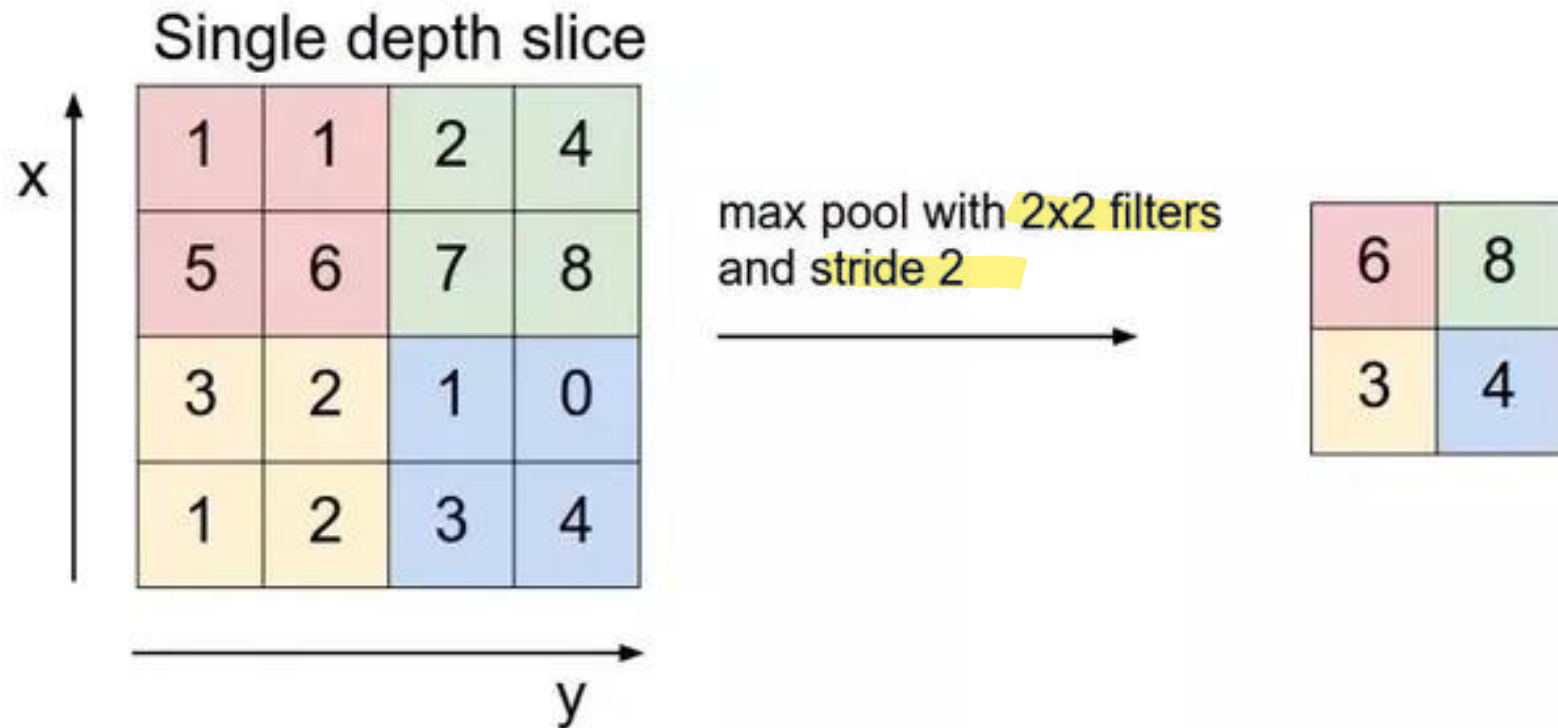
A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs

- 
- max pooling
 - average
 - weighted average
 - sum
 - L2 norm

pooling helps make the representation become approximately invariant to small translations of the input

Pooling^{1,3}

Example of max pooling



Properties of CNN models

- Sparse interactions between NN units (through kernels of small size)
 - fewer parameters to learn
 - less computation resources are required
- Parameter sharing (same kernel is applied throughout the input)
 - Maintain the same feature detection throughout the input
- Ability to (automatically) learn local structure
- Can handle variable-sized inputs

Applications of CNN models^{6,7}

- Image Processing
 - image classification
 - object detection
 - image segmentation
 - object tracking
 - visual saliency recognition
 - face recognition
- Histopathology
- Speech Processing
- Text Detections and Recognition (OCR)

Applications of CNN models^{6,7}

- Natural Language Processing
- Drug Discovery
- Timeseries Analysis
 - Health risk assessment and biomarkers of aging discovery
 - Forecasting
 - Electromyography (EMG) recognition
- Go

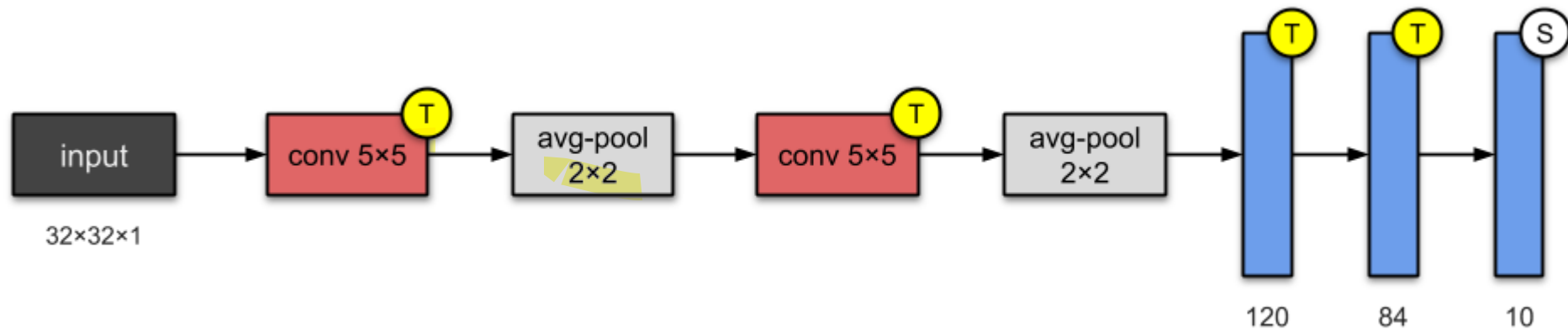


Notable CNN models^{11, 12}

- LeNet
- AlexNet
- VGG-16
- Inception-based architectures
- Xception
- ResNet

Notable CNN models^{11,12}

LeNet (1998)⁸

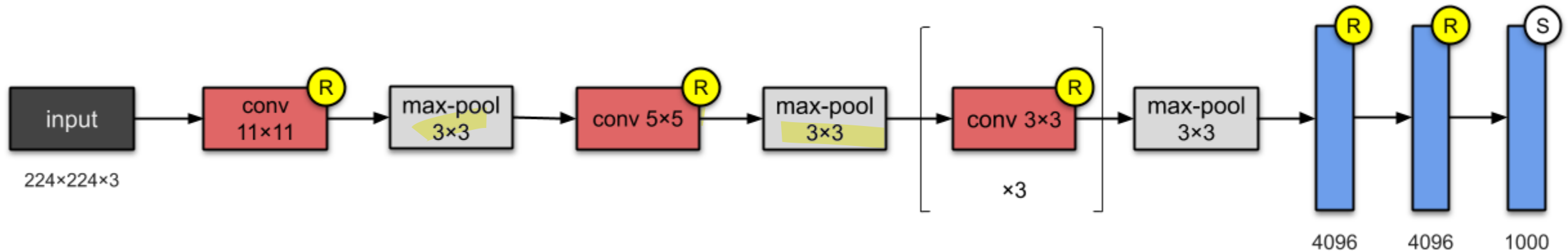


The architecture has become the standard 'template' architecture:

- (1) stacking convolutions and pooling layers (7 levels)
- (2) ending the network with one or more fully-connected layers

Notable CNN models^{11,12}

AlexNet (2012)⁹



(1) deeper and with more filters than LeNet

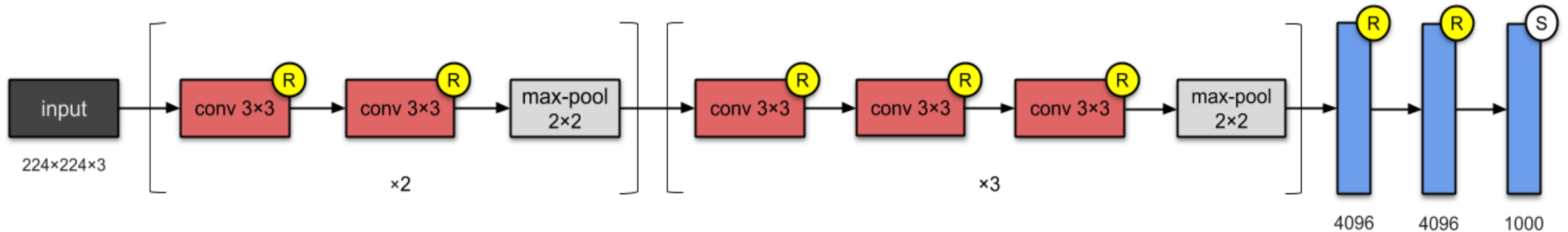
(2) employed stacked convolutional layers

(3) was first to implement Rectified Linear Units (ReLU) as activation functions

→ in 2012 outperformed all prior competitors and won the ImageNet challenge

Notable CNN models^{11,12}

VGG-16/VGG-19 (2014)¹⁰



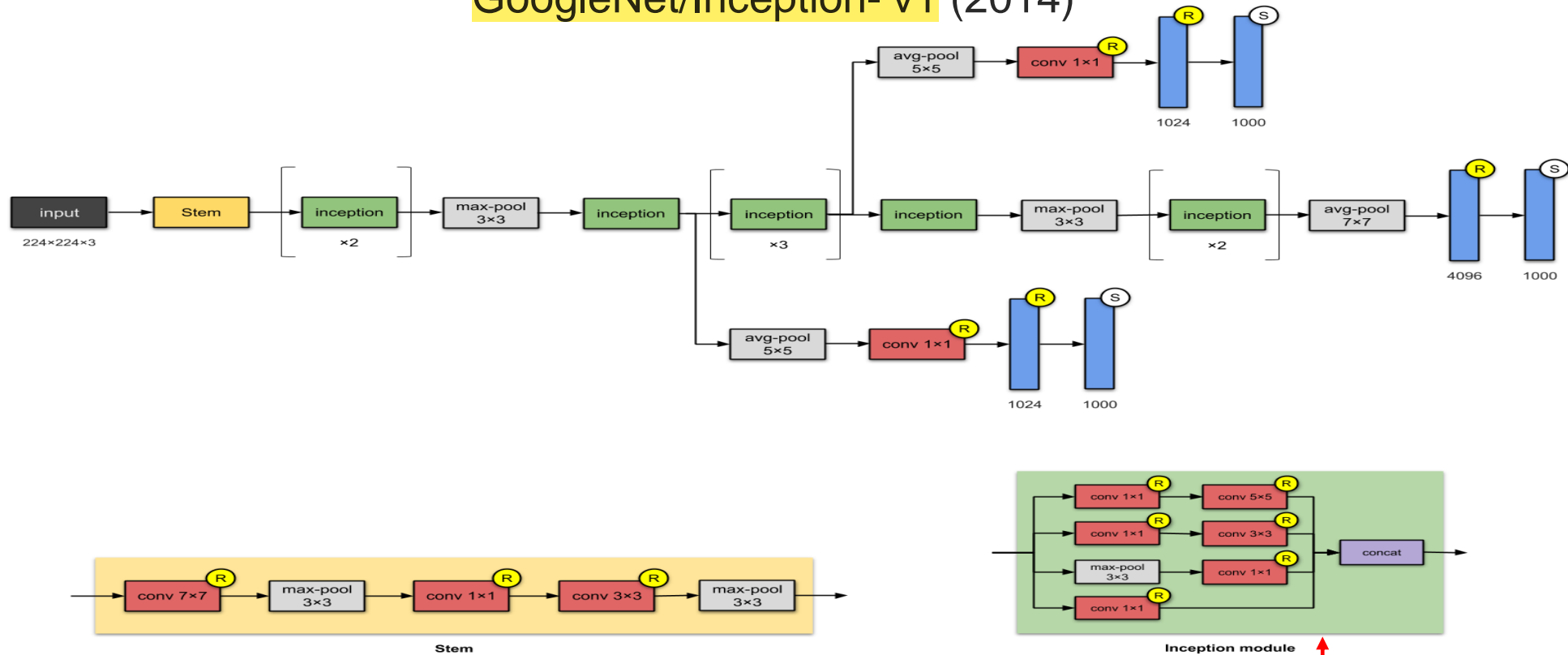
(1) significantly deeper network than the previous ones (16 to 19 layers)

(2) more and smaller filters (3x3)

→ was the runner-up in 2014 ImageNet challenge

Notable CNN models^{11,12}

GoogleNet/Inception-v1 (2014)¹³



(1) even deeper network (22 layers)

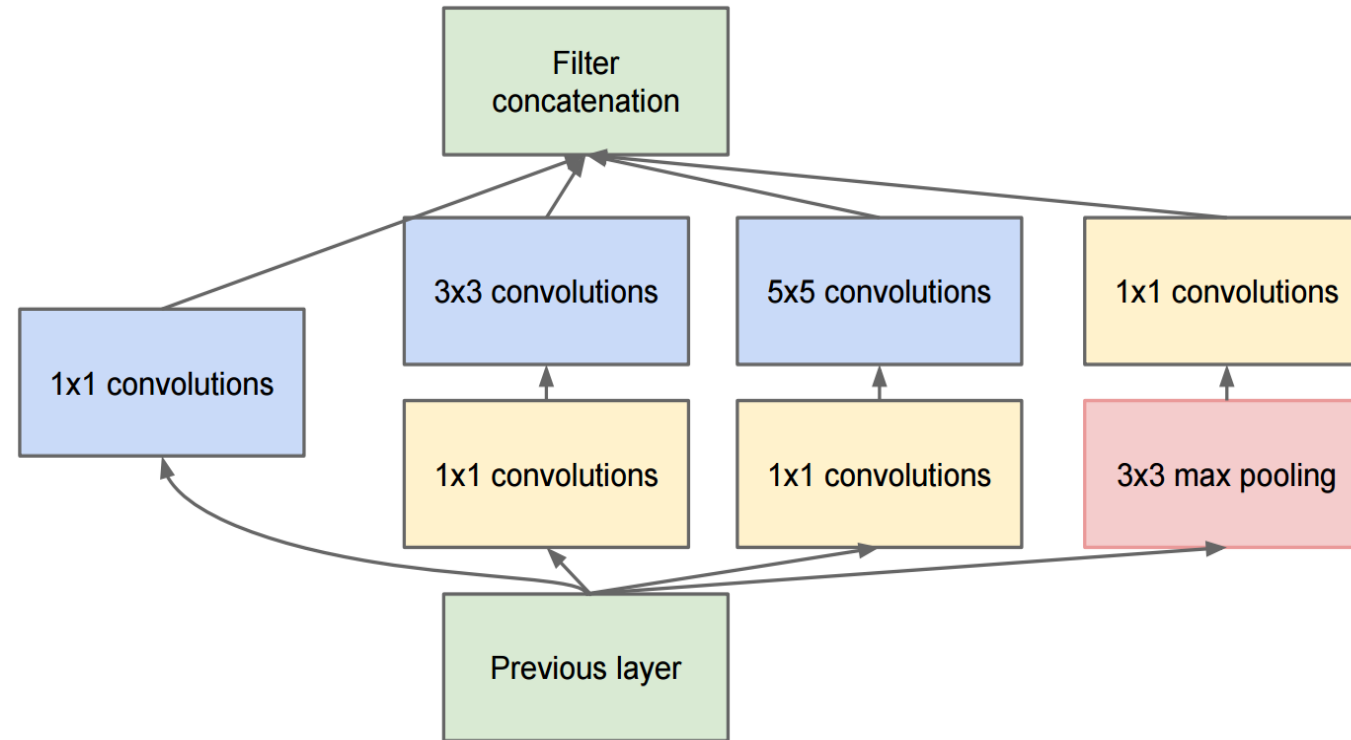
(2) implemented the inception module to drastically reduce the number of parameters

(3) employed batch normalization

→ was the winner in 2014 ImageNet challenge

Notable CNN models^{11,12}

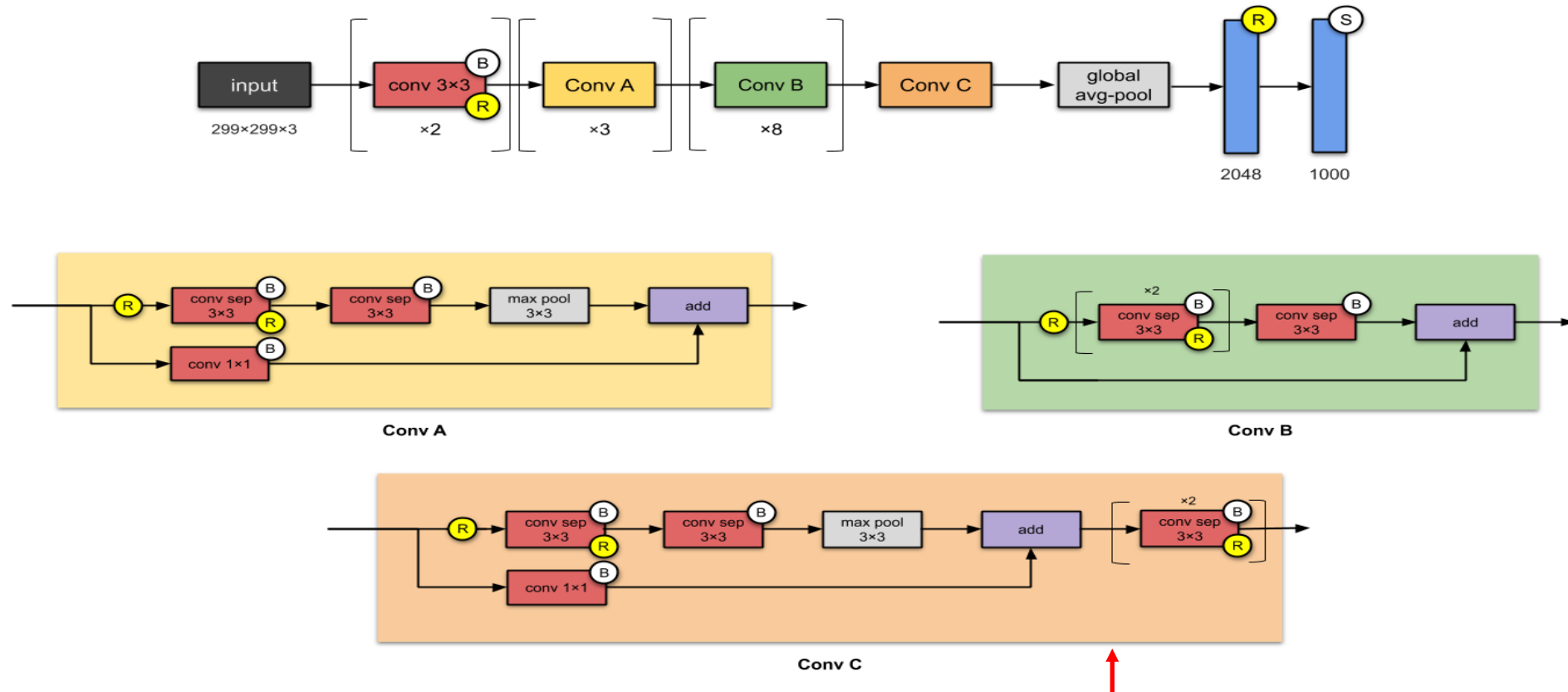
Inception- 1 module



Enhancements of the original inception module (e.g., Inception- v3¹⁴, Inception- v4¹⁸) have improved the performance of the inception-supported models, most notably by refactoring larger convolutions into consecutive smaller ones that are easier to learn

Notable CNN models^{11,12}

Xception (2016)¹⁶



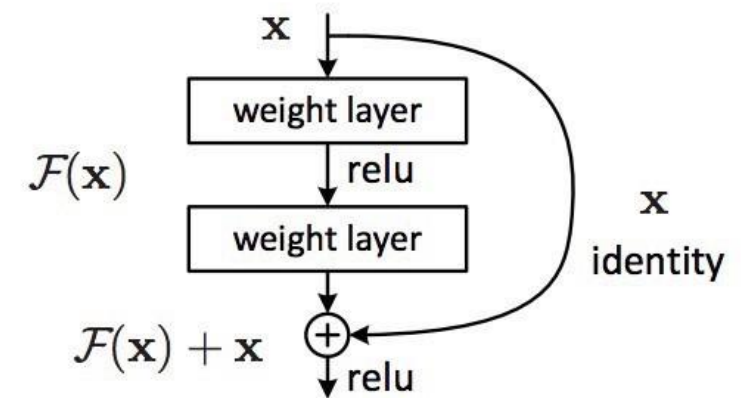
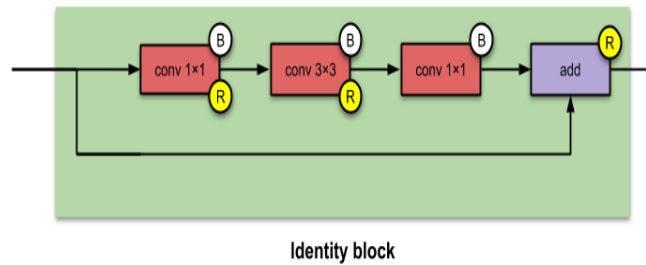
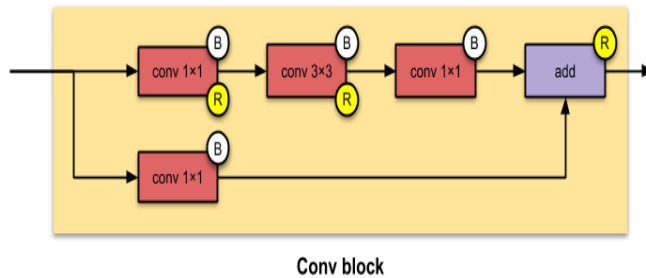
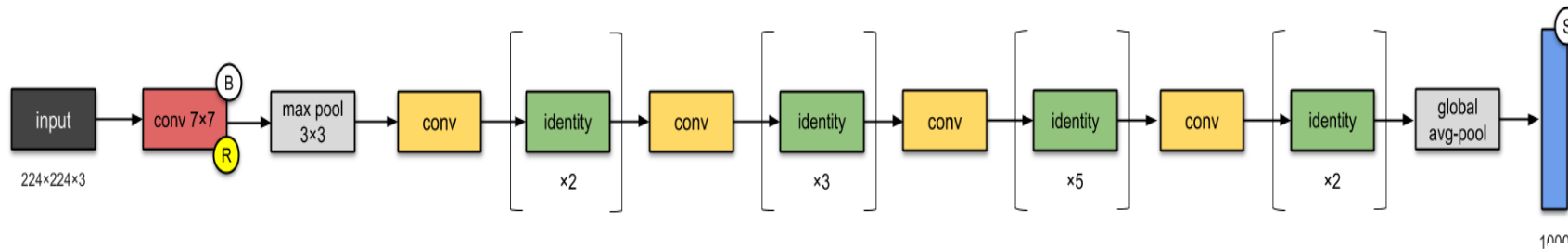
(1) inception modules are replaced with depth-wise separable convolutions

(2) same number of parameters as Inception

→ slightly outperforms Inception v3 on the ImageNet dataset, and vastly outperforms it on a larger image classification dataset with 17,000 classes

Notable CNN models^{11,12}

Resnet (2015)¹⁵

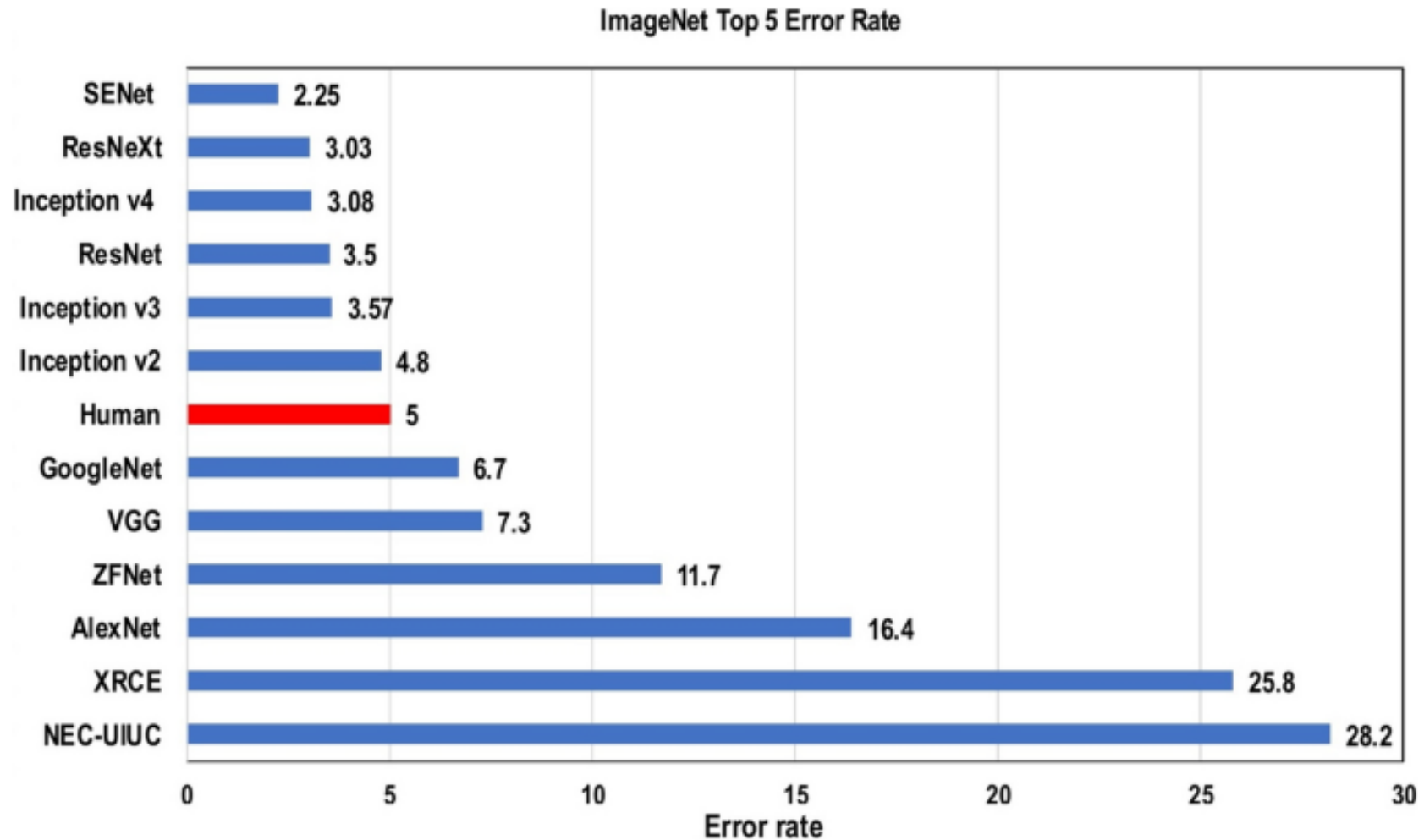


(1) instead of learning a mapping from x to $H(x)$, learn the difference $F(x) = H(x) - x \rightarrow$ creates skip connections

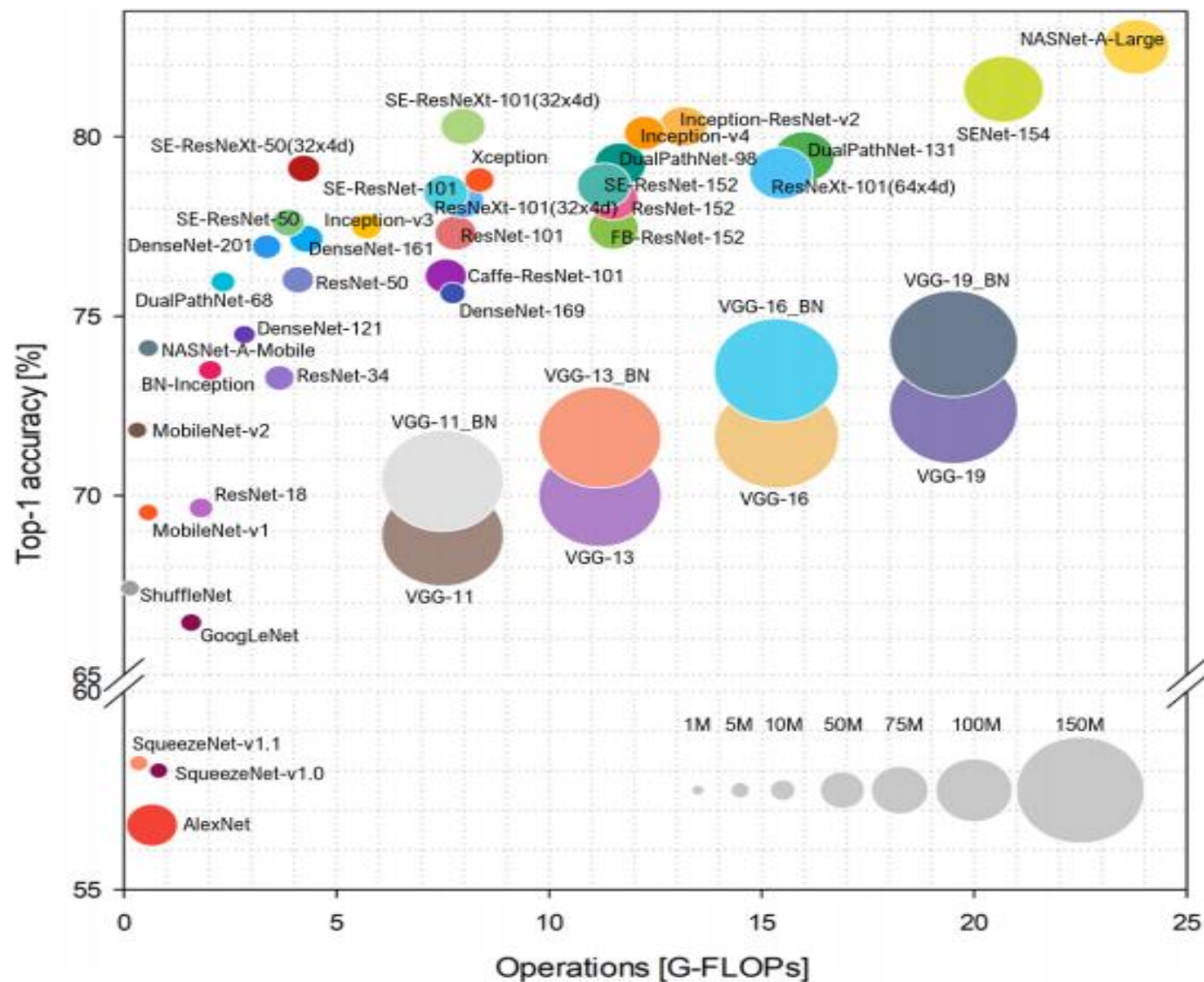
(2) skip connections stacked on top of each other

\rightarrow enables deeper architectures without loss of performance

Notable CNN models¹⁹



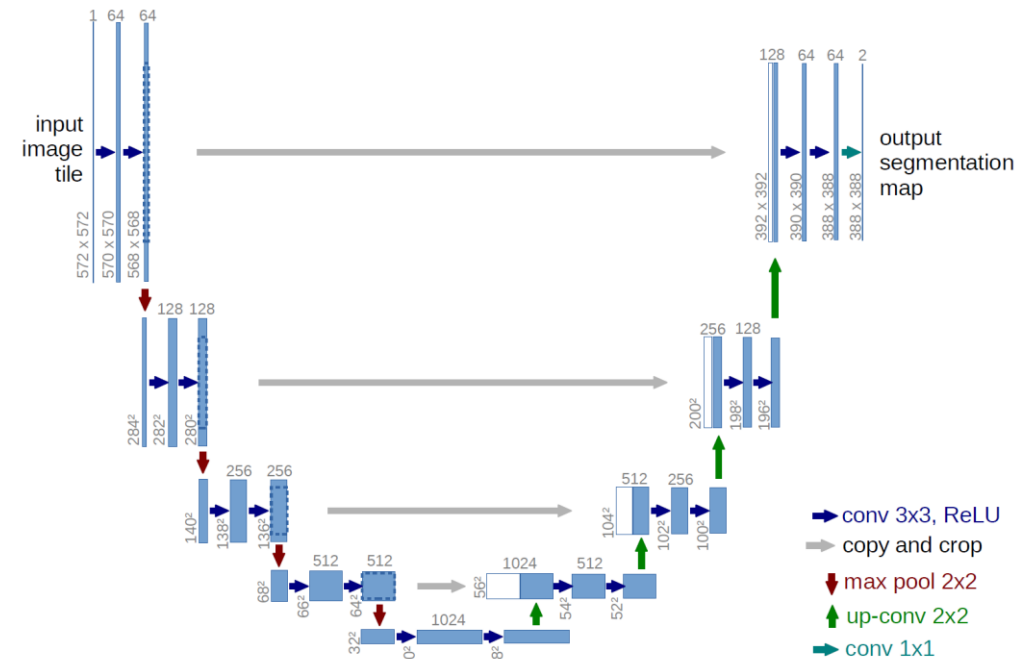
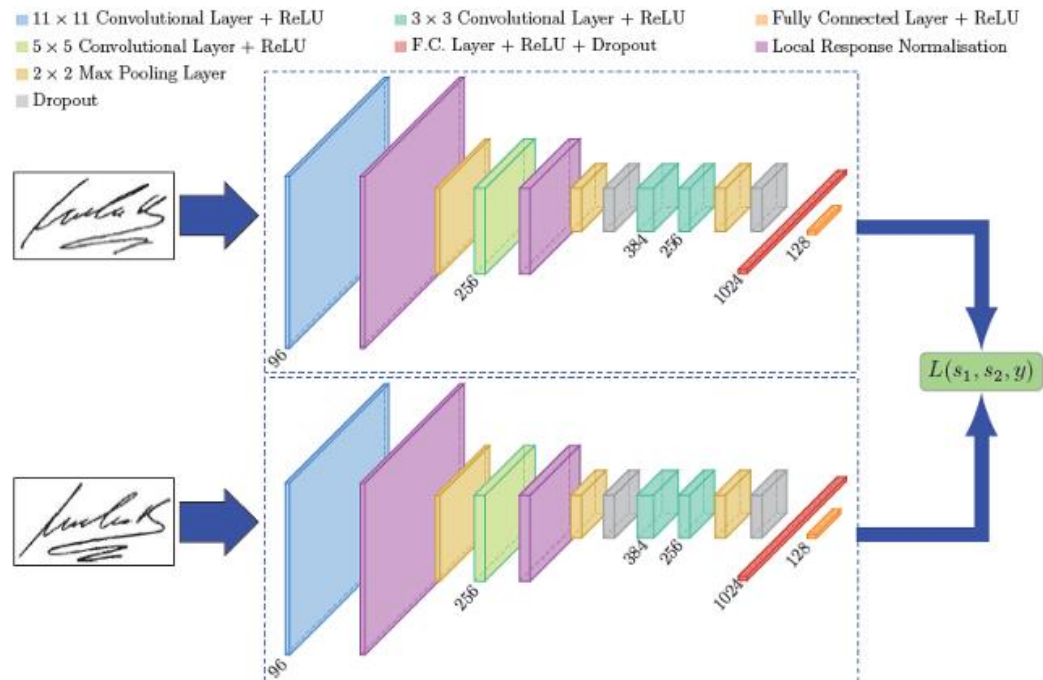
Notable CNN models²⁰



Application Architectures based on CNN models

- **UNET:** semantic segmentation

- **Siamese Network:**
 - learns from very little data



All is great then?²¹

CNN models can be easily fooled



All is great then?²¹

CNNs are bad at generalizing across scaling, shifts and rotations in images, as well as different angles of three-dimensional viewing.

