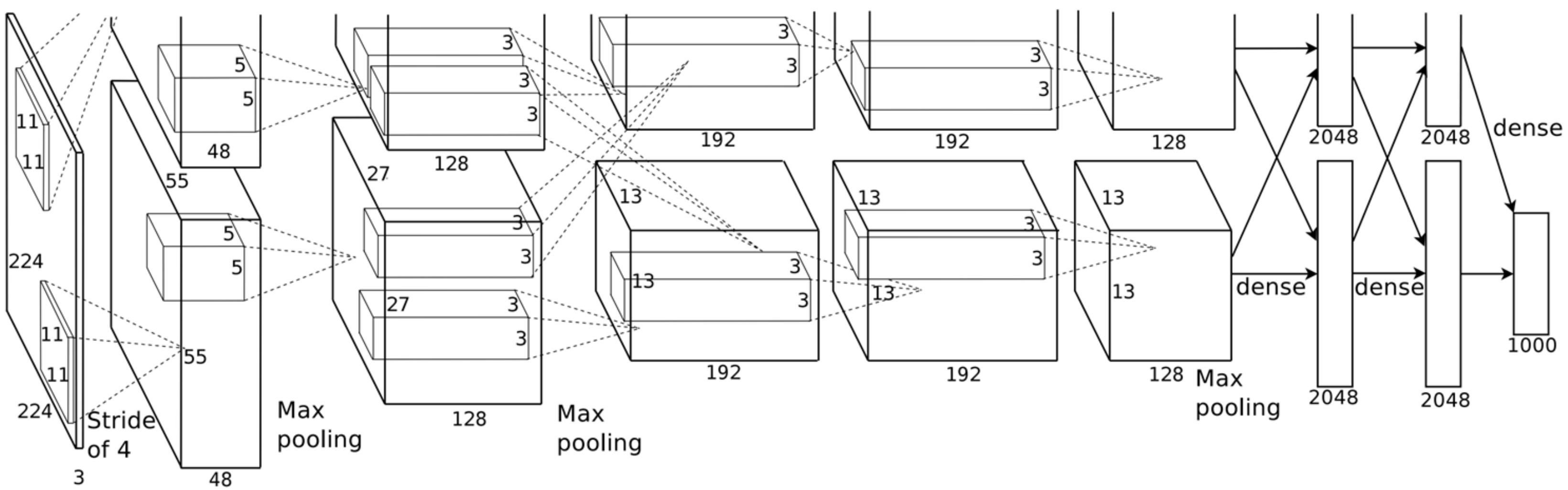
About Deep NN



2018 Turing Award — by ACM @ 2019/3/27

AWARDS & RECOGNITION

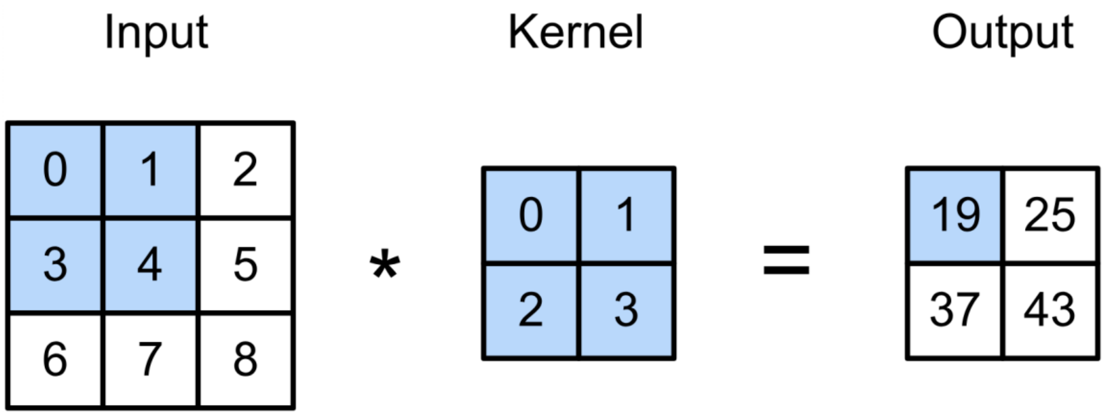
ACM Announces 2018 Turing Award Recipients

ACM has named Yoshua Bengio of the University of Montreal, Geoffrey Hinton of Google, and Yann LeCun of New York University recipients of the 2018 ACM A.M. Turing Award for conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing. Working independently and together, Hinton, LeCun and Bengio developed conceptual foundations for the field, identified surprising phenomena through experiments, and contributed engineering advances that demonstrated the practical advantages of deep neural networks.

Outline

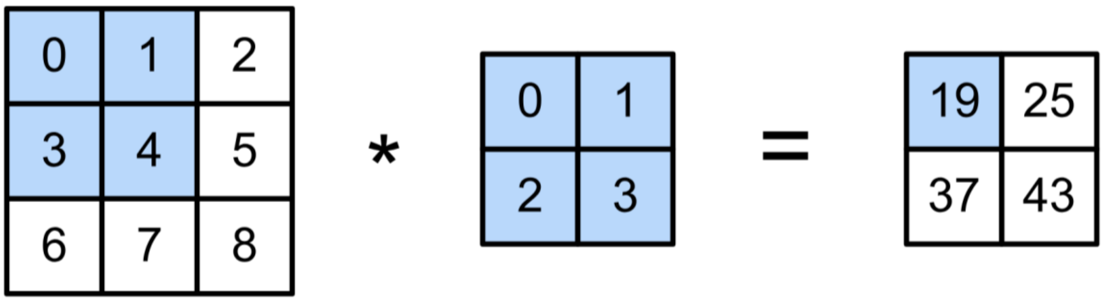
- Convolution, Padding & Stride
- Pooling
- Convolutional Neural Network (LeNet)
- Deep Neural Networks
- Deep Learning Frameworks

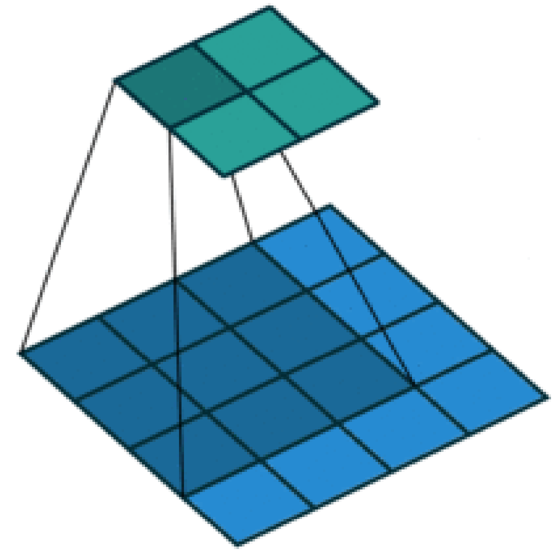
2-D Cross Correlation



$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$
$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$
$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$
$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$

2-D Convolution Layer





(vdumoulin@ Github)

$\mathbf{X}: n_h \times n_w$

input matrix

$\mathbf{W}: k_h \times k_w$

kernel matrix

$\mathbf{Y}: (n_h - k_h + 1) \times (n_w - k_w + 1)$

output matrix feature map

b: scalar bias

$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

W and *b* are learnable parameters.

Examples

(wikipedia)

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

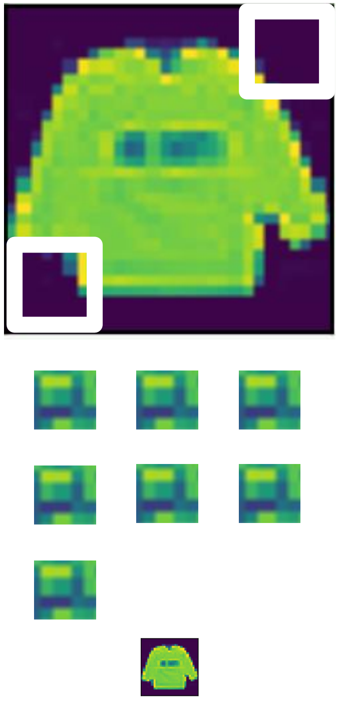
Sharpen

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

Gaussian Blur

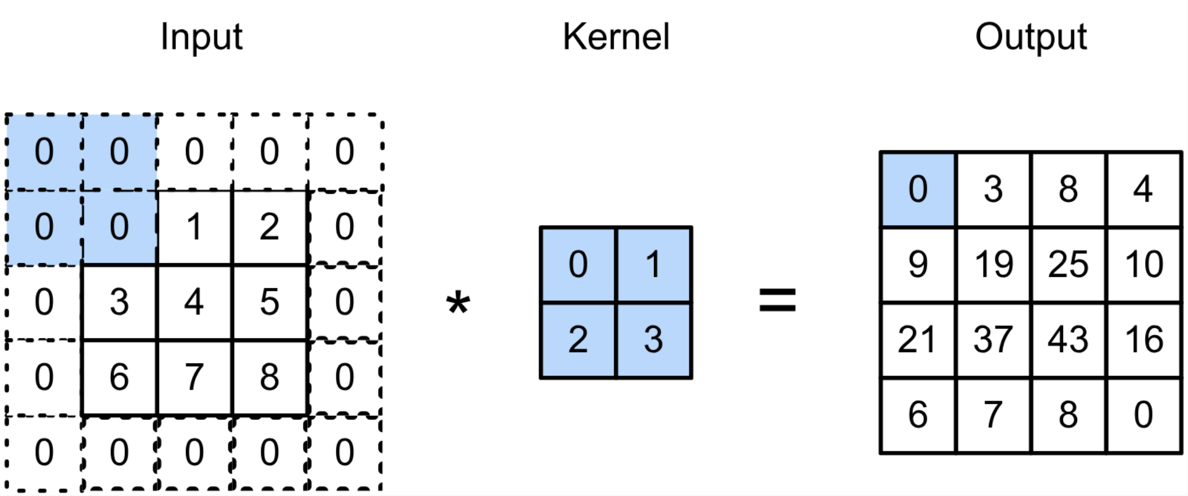
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to

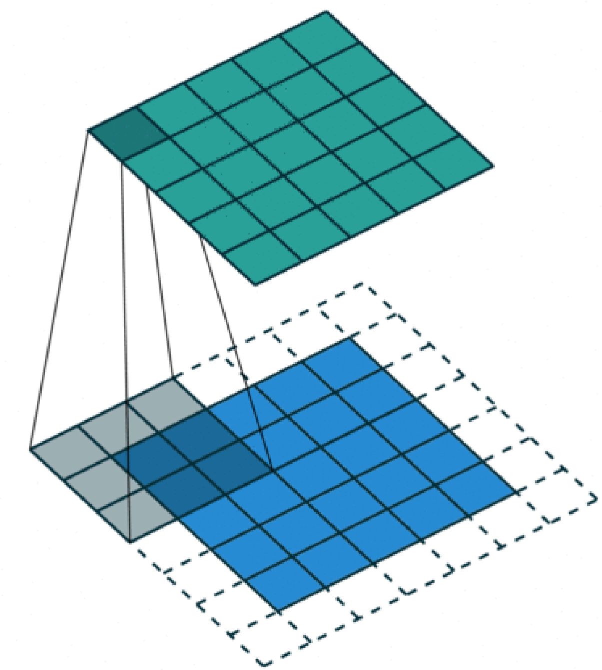
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



Padding

Padding adds rows/columns around input





$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

- Padding p_h rows and p_w columns, output shape will be

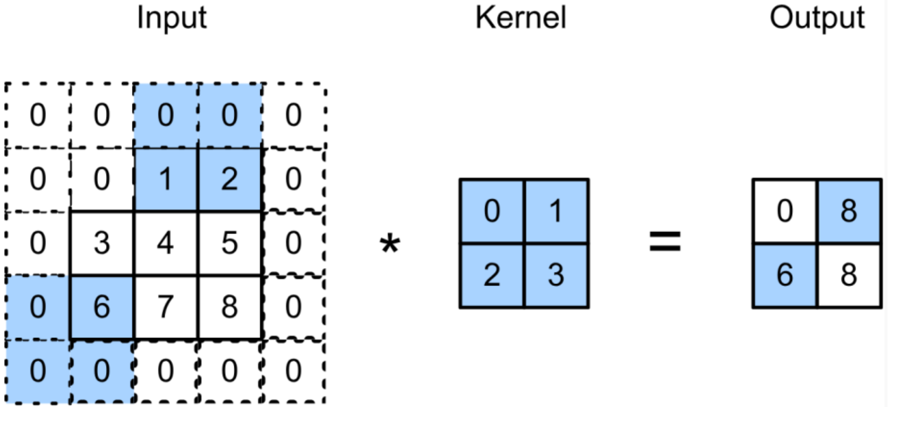
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- A common choice is $p_h = k_h - 1$ and $p_w = k_w - 1$
  - Odd k_h: pad $p_h/2$ on both sides
  - Even k_h: pad $\lceil p_h/2 \rceil$ on top, $\lfloor p_h/2 \rfloor$ on bottom

Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$

Stride

- Given stride s_h for the height and stride s_w for the width, the output shape is

$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

- With $p_h = k_h - 1$ and $p_w = k_w - 1$:

$$\lfloor (n_h + s_h - 1)/s_h \rfloor \times \lfloor (n_w + s_w - 1)/s_w \rfloor$$

- If input height/width are divisible by strides:

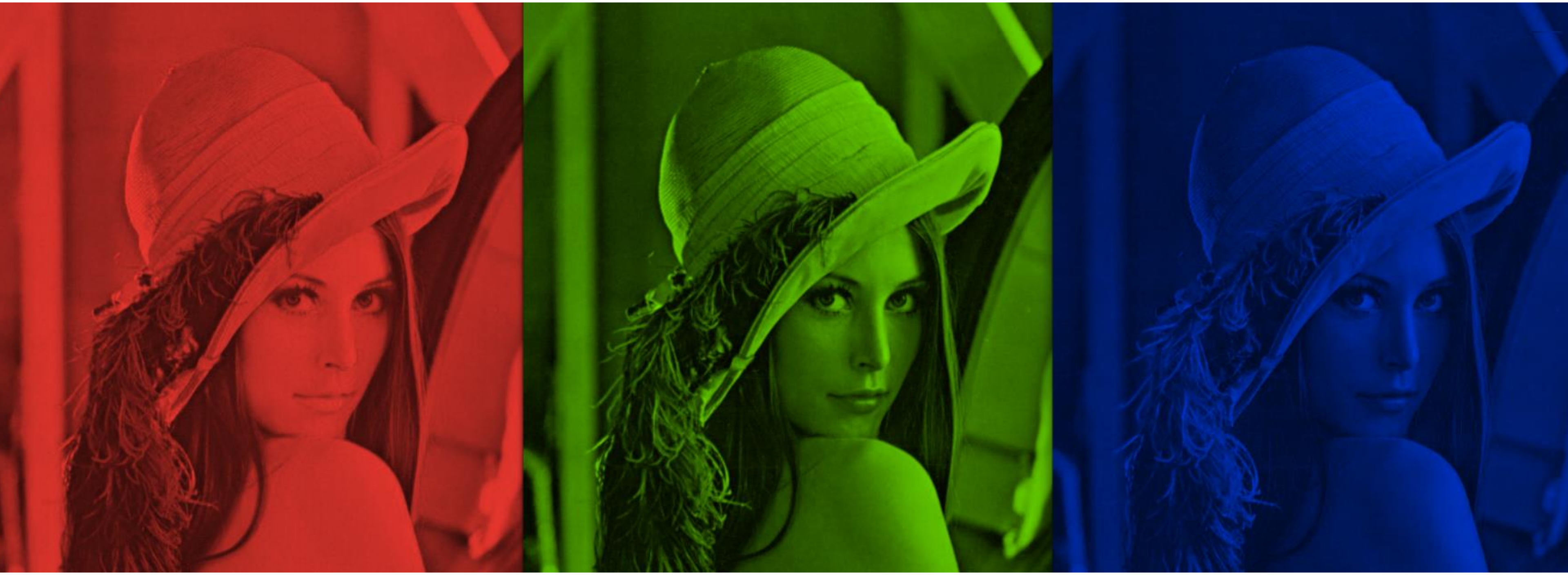
$$(n_h/s_h) \times (n_w/s_w)$$

Multiple Input Channels

- Color image may have three RGB channels

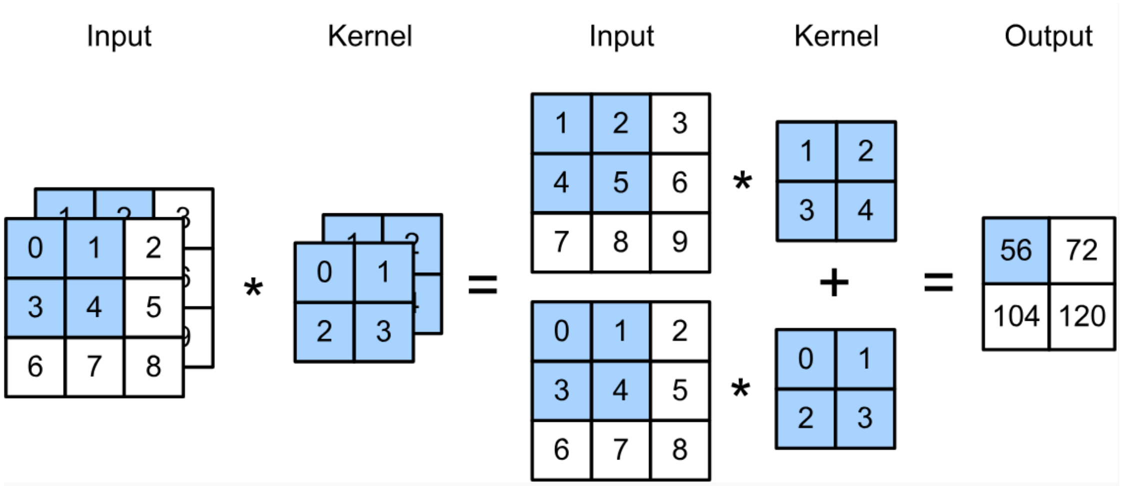
Multiple Input Channels

- Color image may have three RGB channels
- Converting to grayscale loses information



Multiple Input Channels

- Have a kernel for each channel, and then sum results over channels



$$
\begin{gathered}
(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4) \\
+(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3) \\
= 56
\end{gathered}
$$

Multiple Input Channels

- **X**: $c_i \times n_h \times n_w$ input
- **W**: $c_i \times k_h \times k_w$ kernel
- **Y**: $m_h \times m_w$ output

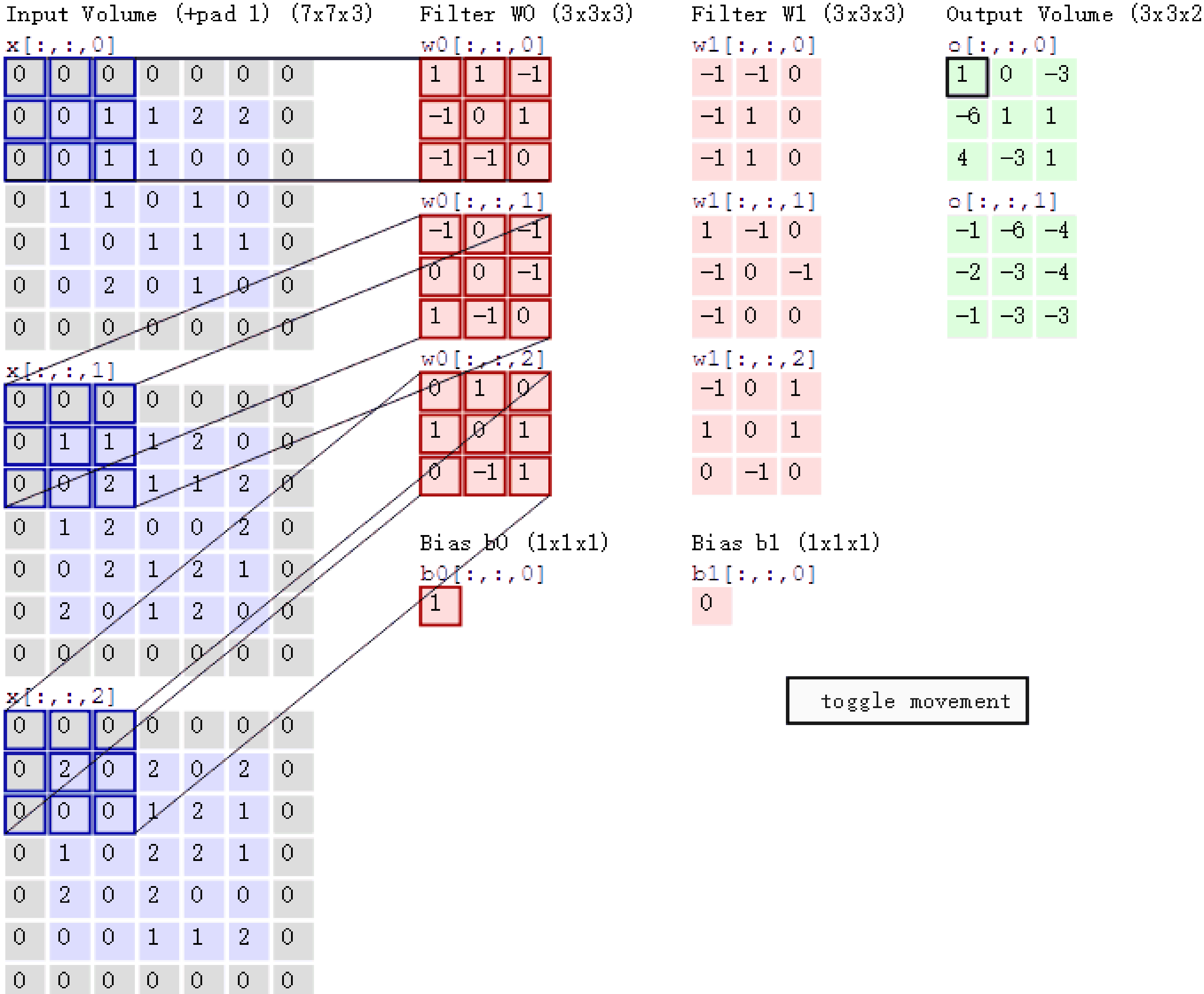
$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

Multiple Output Channels

- No matter how many inputs channels, so far we always get single output channel.
- We can have multiple 3-D kernels, each one generates a output channel.
  - Input $\mathbf{X}$: $c_i \times n_h \times n_w$
  - Kernel $\mathbf{W}$: $c_o \times c_i \times k_h \times k_w$
  - Output $\mathbf{Y}$: $c_o \times m_h \times m_w$

$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

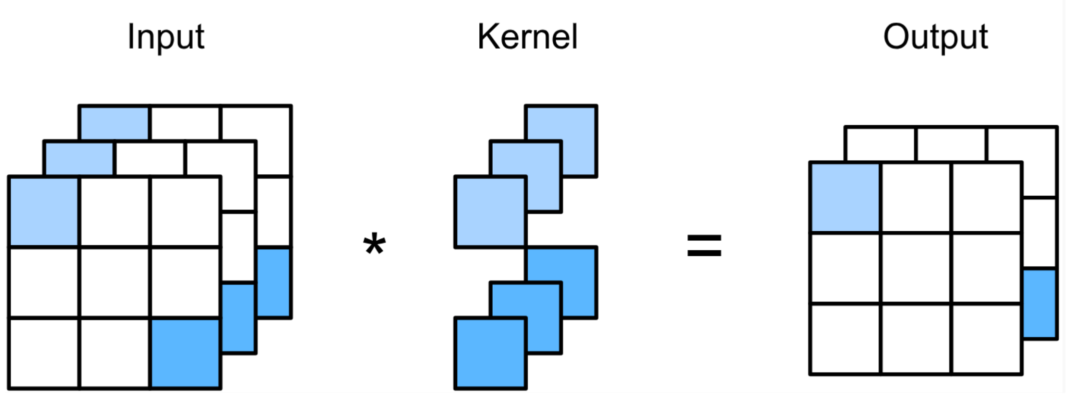
$$for\ i = 1, \ldots, c_o$$


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 0 1 1 2 2 0
0 0 1 1 0 0 0
0 1 1 0 1 0 0
0 1 0 1 1 1 0
0 0 2 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 1 1 2 0 0
0 0 2 1 1 2 0
0 1 2 0 0 2 0
0 0 2 1 2 1 0
0 2 0 1 2 0 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 2 0 2 0 2 0
0 0 0 1 2 1 0
0 1 0 2 2 1 0
0 2 0 2 0 0 0
0 0 0 1 1 2 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
1 1 -1
-1 0 1
-1 -1 0
w0[:,:,1]
-1 0 -1
0 0 -1
1 -1 0
w0[:,:,2]
0 1 0
1 0 1
0 -1 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 -1 0
-1 1 0
-1 1 0
w1[:,:,1]
1 -1 0
-1 0 -1
-1 0 0
w1[:,:,2]
-1 0 1
1 0 1
0 -1 0
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
1 0 -3
-6 1 1
4 -3 1
o[:,:,1]
-1 -6 -4
-2 -3 -4
-1 -3 -3
toggle movement

1 x 1 Convolutional Layer

$k_h = k_w = 1$ is a popular choice.

It doesn't recognize spatial patterns, but fuse channels.



Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

Outline

- Convolution, Padding & Stride
- Pooling
- Convolutional Neural Networks (LeNet)
- Deep Neural Networks
- Deep Learning Frameworks

Pooling

- Why pooling:
  - We want to reduce the dimensionality when processing data
  - Alleviate the excessive sensitivity of the convolutional layer to location
    - Lighting, object positions, scales, appearance vary among images
    - Convolution is sensitive to position
      - e.g. Detect vertical edges

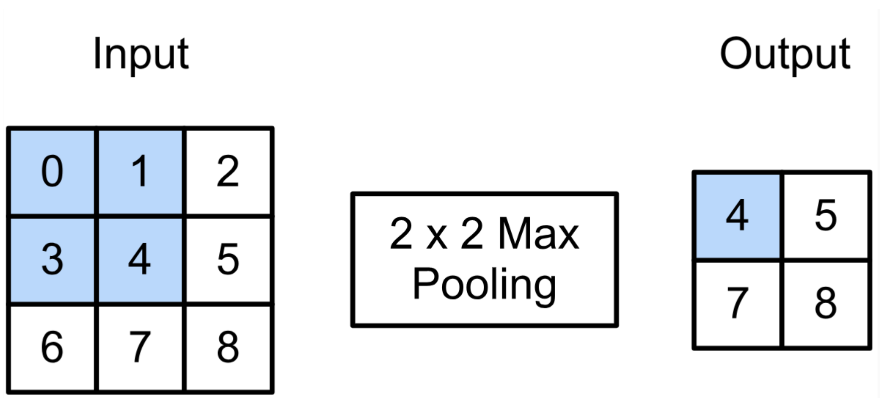
```
X  [[1. 1. 0. 0. 0.        Y  [[ 0.  1.  0.  0.
    [1. 1. 0. 0. 0.            [ 0.  1.  0.  0.
    [1. 1. 0. 0. 0.            [ 0.  1.  0.  0.
    [1. 1. 0. 0. 0.            [ 0.  1.  0.  0.
```

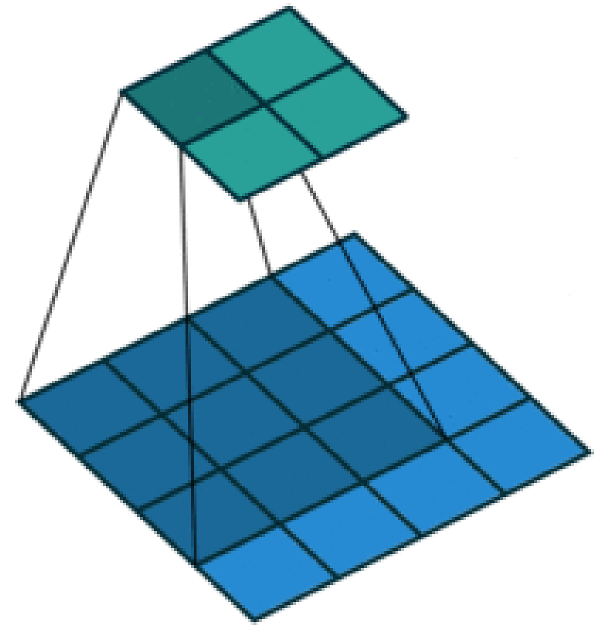
Pooling

- Commonly used
  - Max Pooling
  - Average Pooling

2-D Max Pooling

- Returns the maximal value in the sliding window

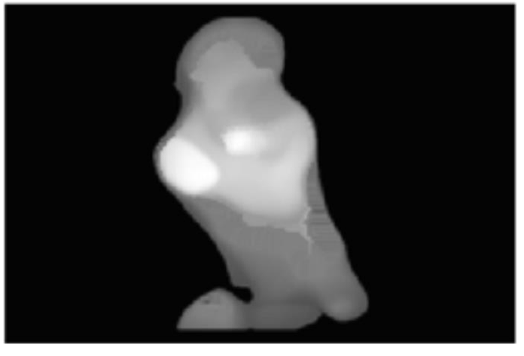




$$max(0,1,3,4) = 4$$

Average Pooling

- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window



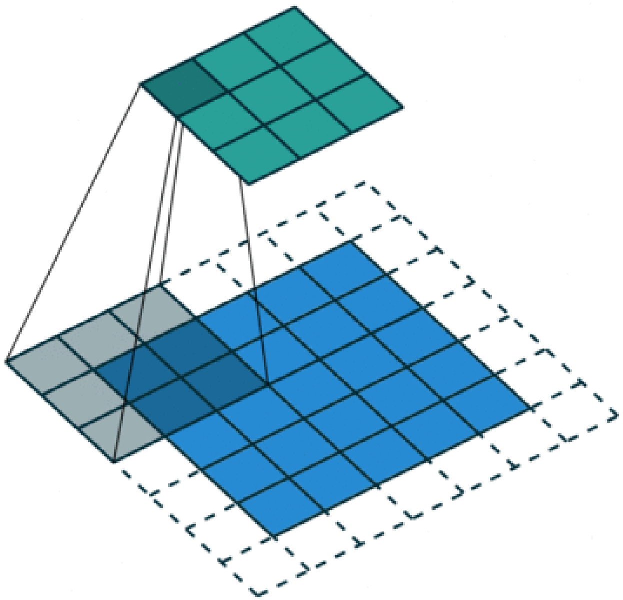
Max pooling

Average pooling

Padding, Stride, and Multiple Channels

- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel

  #output channels = #input channels



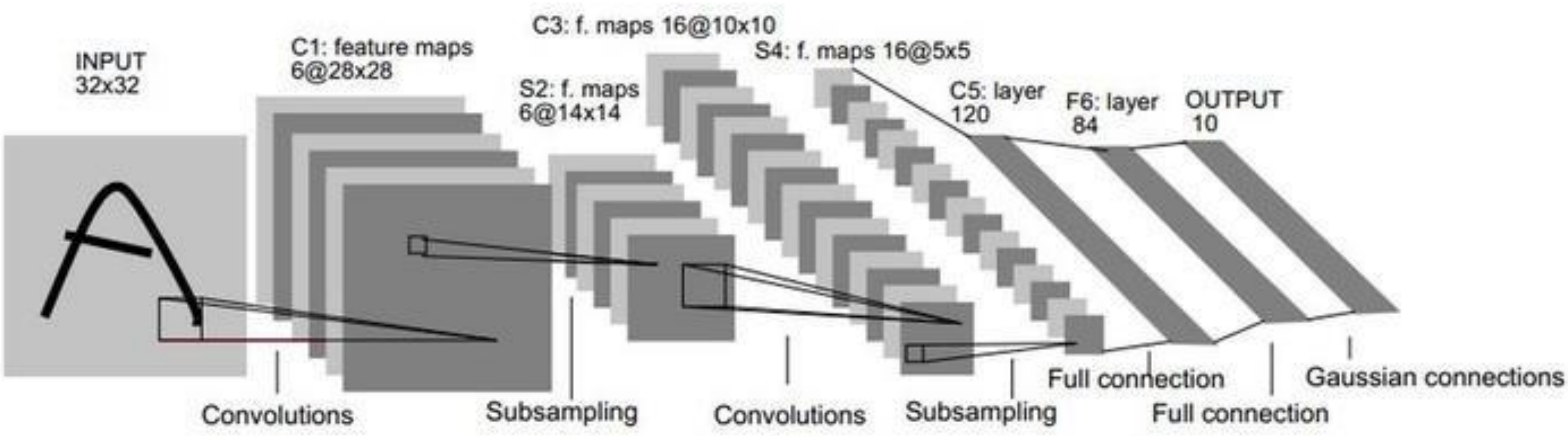
Summary

- Convolutional layer
  - Reduced model capacity compared to dense layer
  - Efficient at detecting spatial patterns
  - High computation complexity
  - Control output shape via padding, strides and channels
- Max/Average Pooling layer
  - Reduce the dimension
  - Provides some degree of invariance

Outline

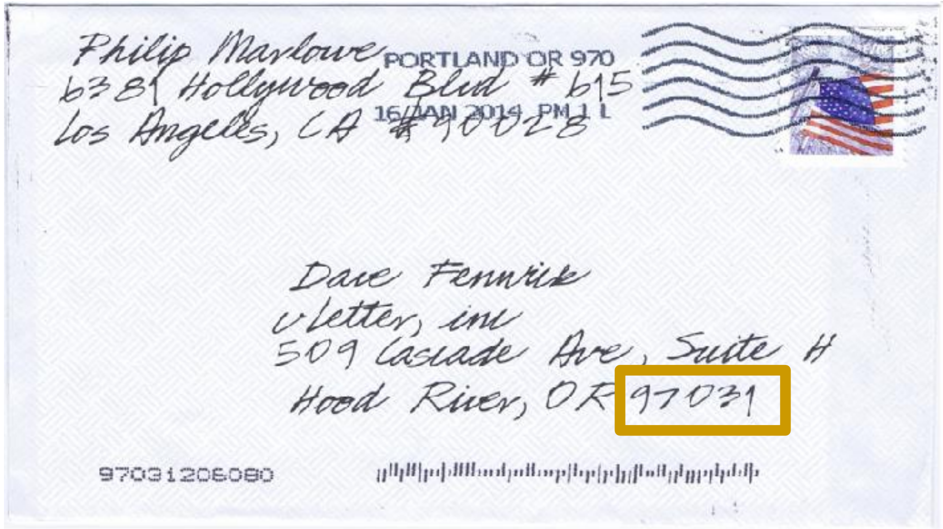
- Convolution, Padding & Stride
- Pooling
- Convolutional Neural Network (LeNet)
- Deep Neural Networks
- Deep Learning Frameworks

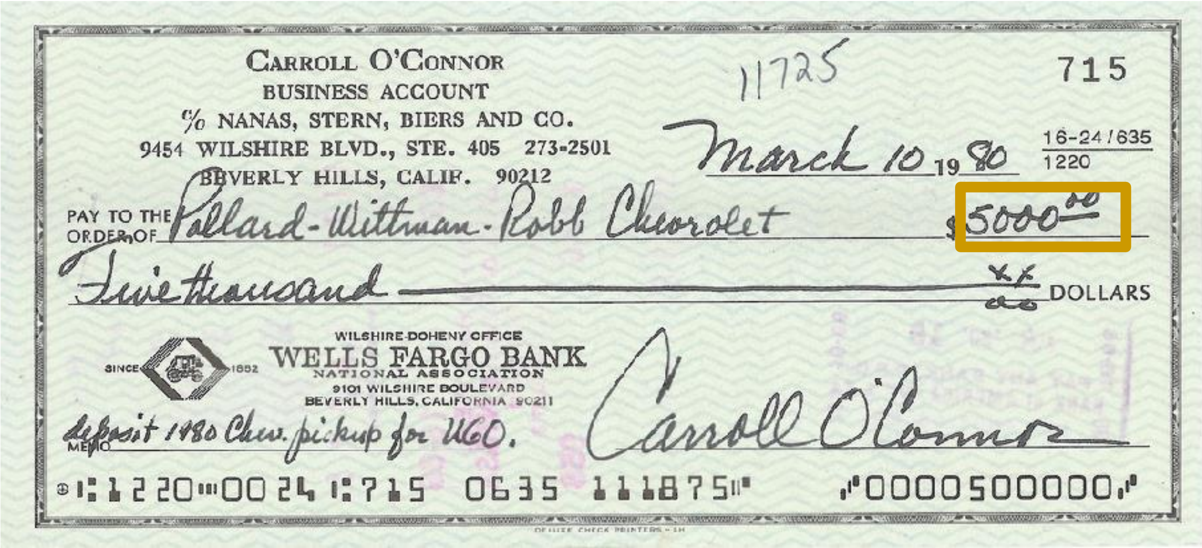
LeNet

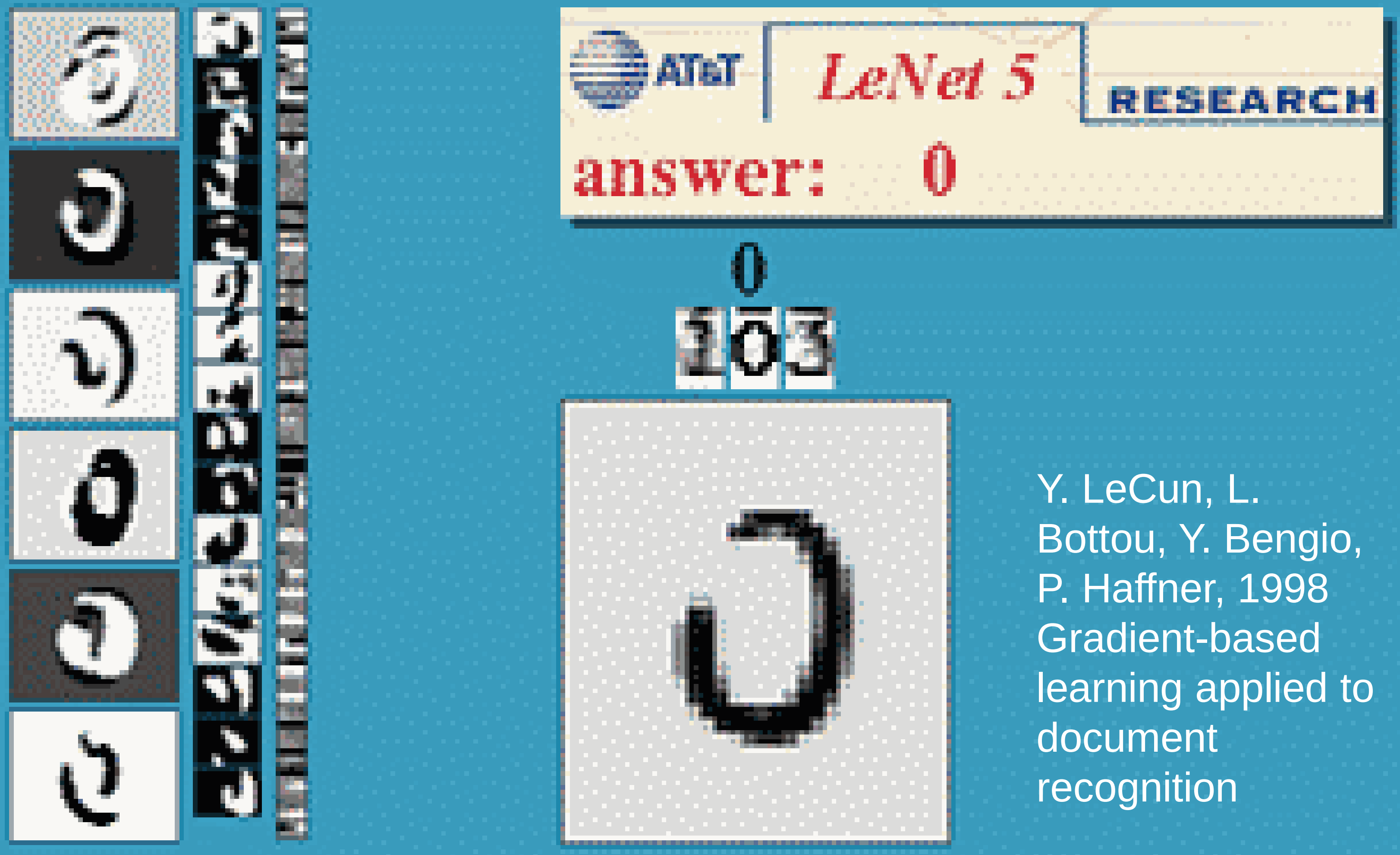


LeCun, Y., Bottou, L., **Bengio**, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.

Handwritten Digit Recognition

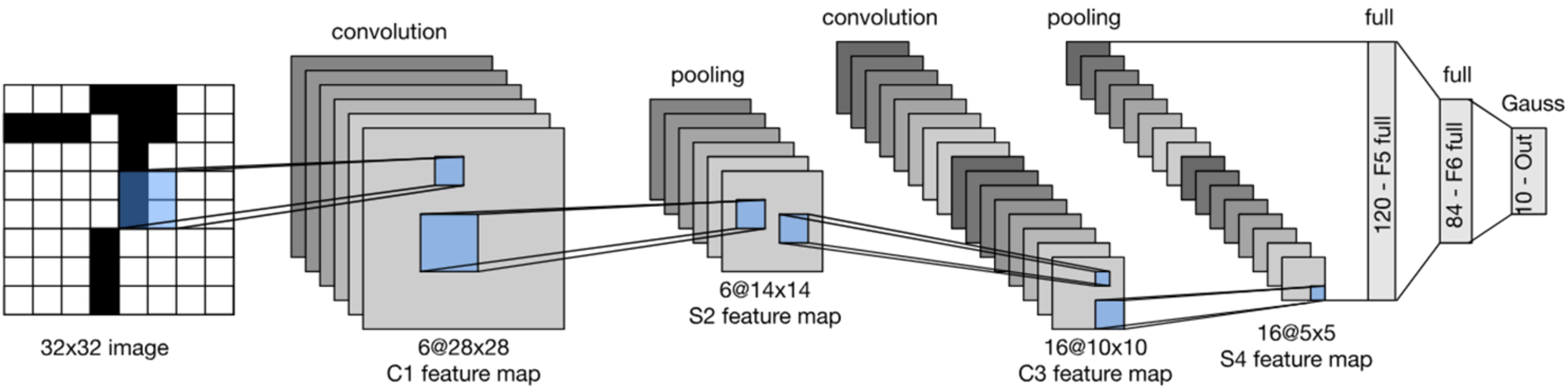






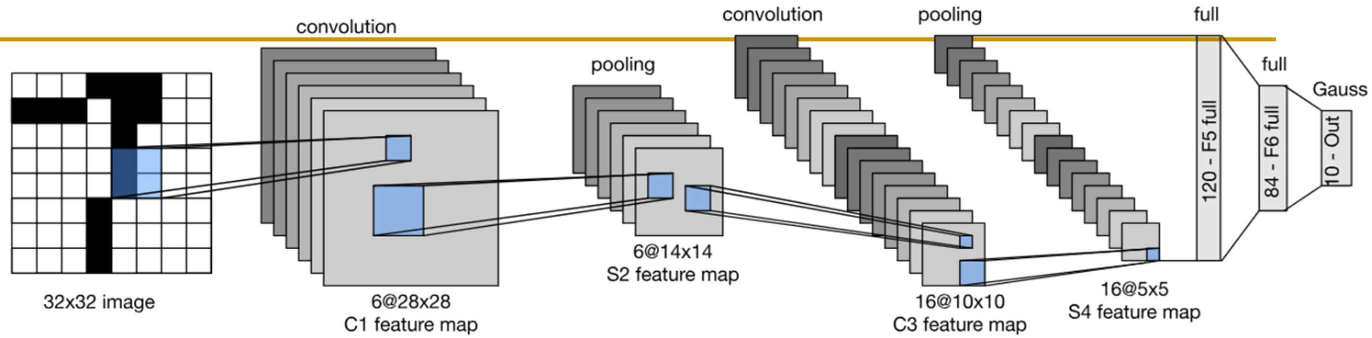
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

LeNet Architecture



- convolutional layers labeled Cx,
- subsampling layers labeled Sx
- fully connected layers labeled Fx

LeNet



- convolutional block
  - convolutional layer
    - used to recognize the spatial patterns in the image, such as lines and the parts of objects
    - each convolutional layer uses a 5*5 window and a sigmoid activation function for the output
  - pooling layer
    - average pooling layer is used to reduce the dimensionality
    - the window shape is 2 x 2 and the stride is 2

Outline

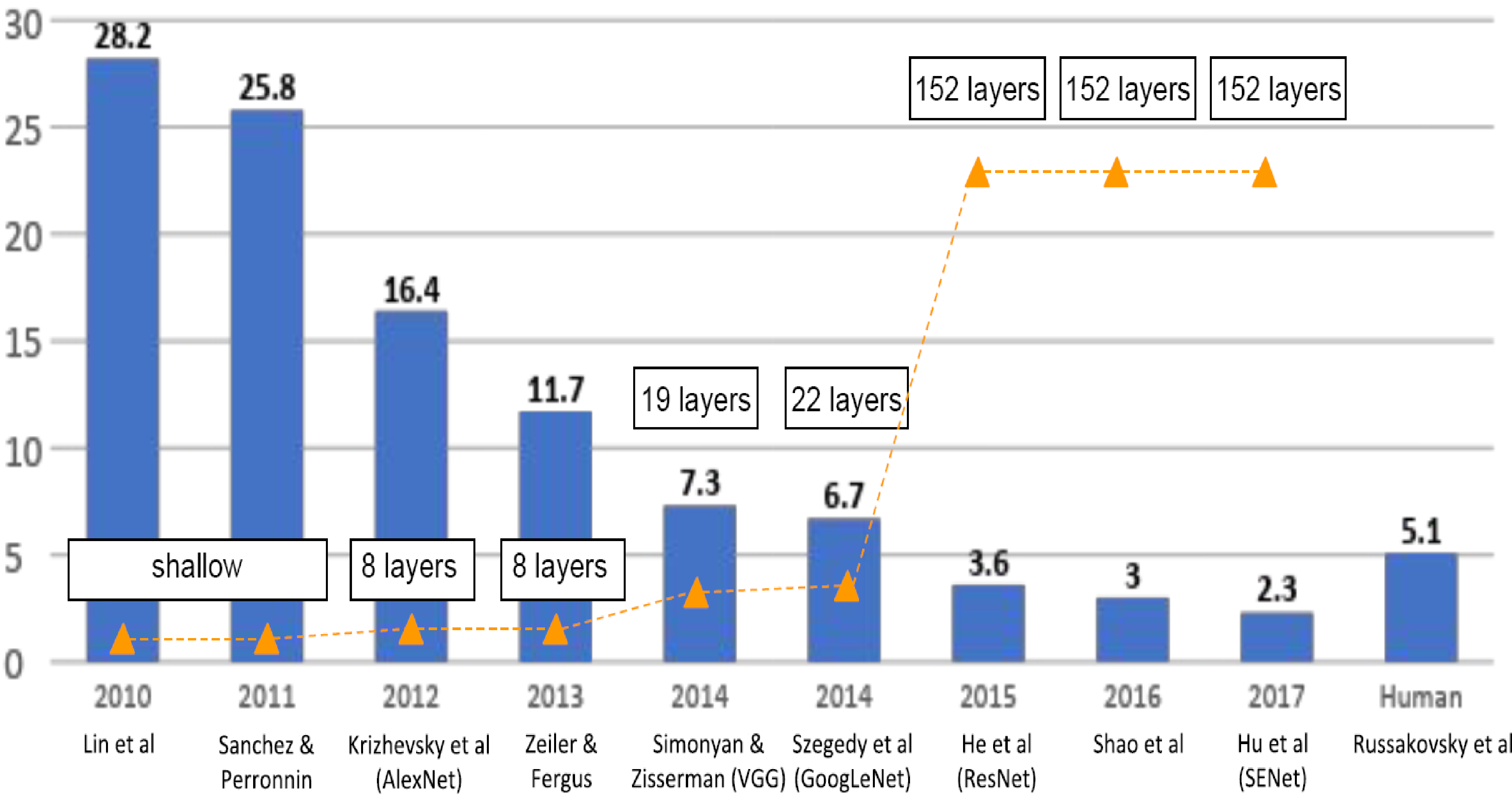
- Convolution, Padding & Stride
- Pooling
- Convolutional Neural Network (LeNet)
- Deep Neural Networks
- Deep Learning Frameworks

Progress

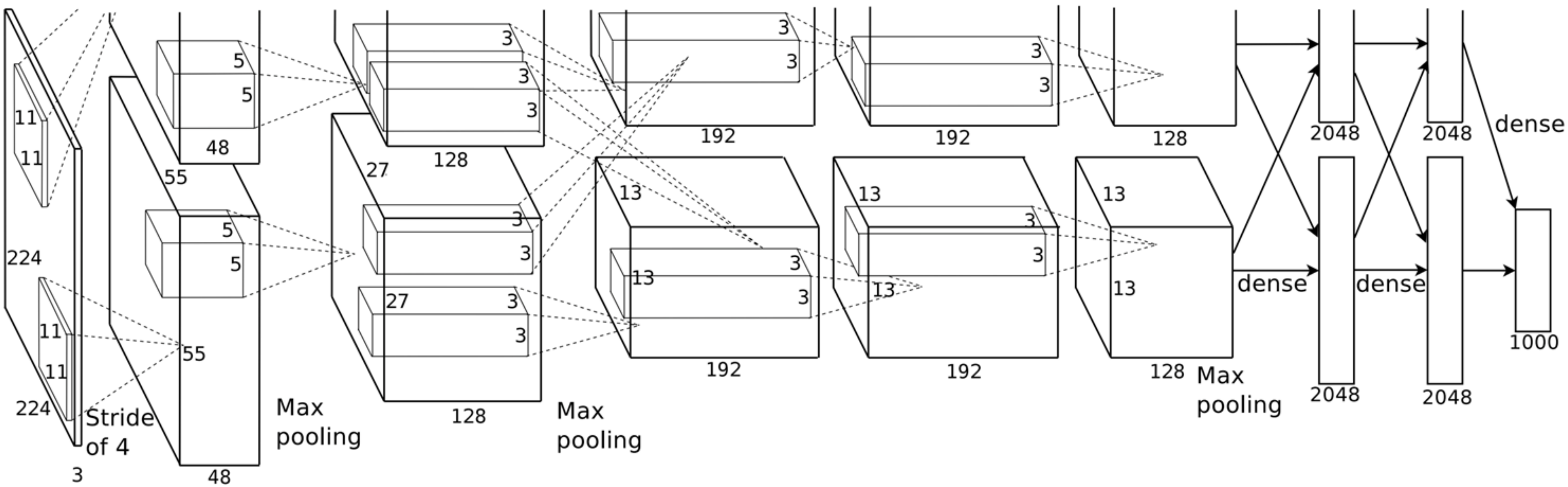
- LeNet
  - 2 convolution + pooling layers
  - 2 hidden dense hidden layers
- AlexNet(2012)
  - Bigger and deeper LeNet
  - ReLu, Dropout, preprocessing
- NiN (2013)
  - 1x1 convolutions + global pooling instead of dense
- GoogLeNet(2014)
  - Incetption
- ResNet (2015)
  - Residual blocks

。。。。。。

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



AlexNet

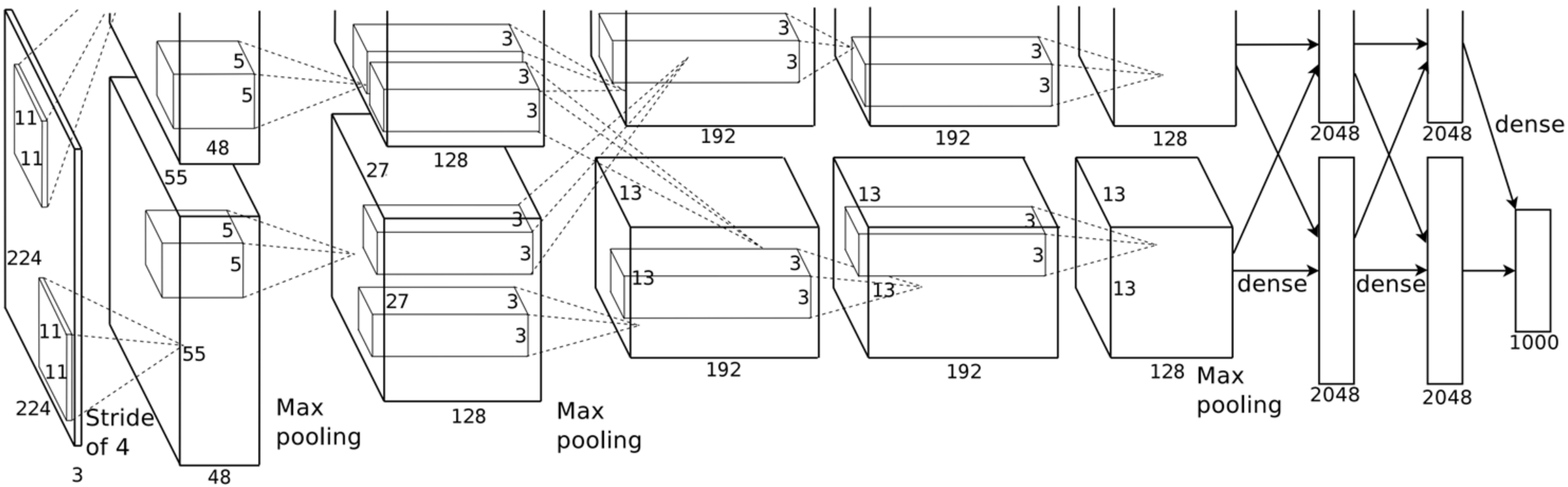


Krizhevsky, A., Sutskever, I., & **Hinton**, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

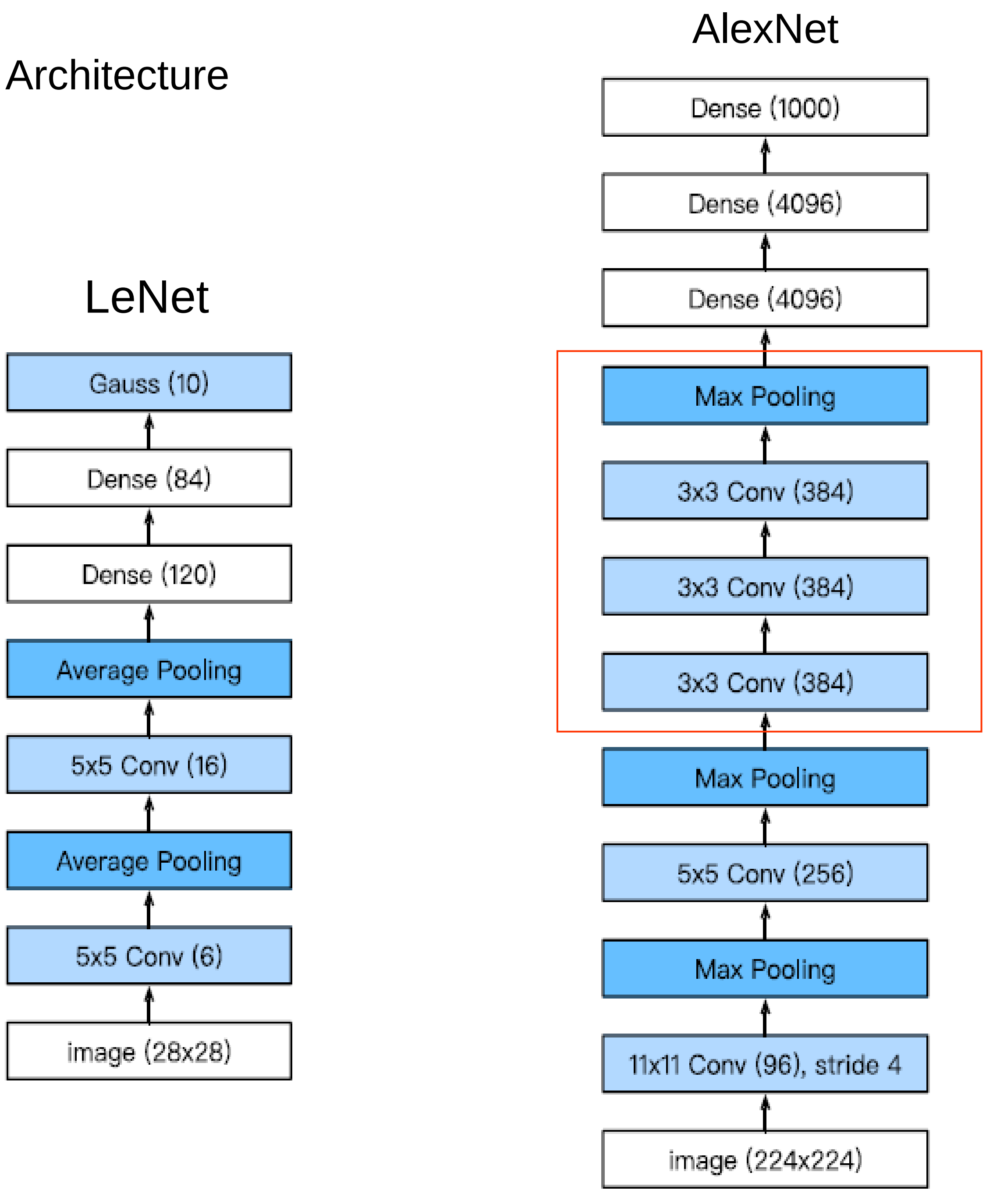
AlexNet

- A key step from shallow to deep networks！
- AlexNet won ImageNet competition in 2012
- Deeper and bigger LeNet
- Key modifications
  - Dropout (regularization)
  - ReLu (training)
  - Data augmentation、Max Pooling

AlexNet Architecture



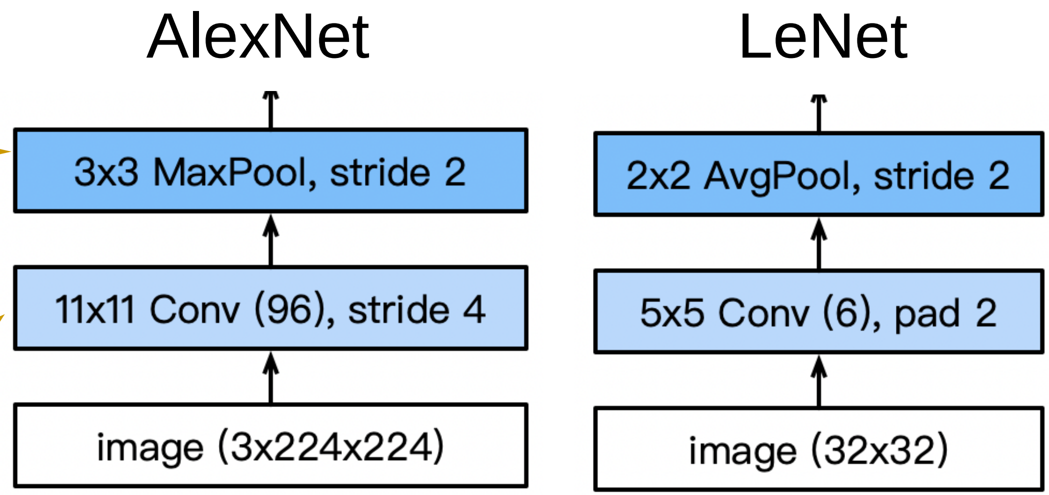
AlexNet Architecture



AlexNet Architecture

Larger pool size, change to max pooling

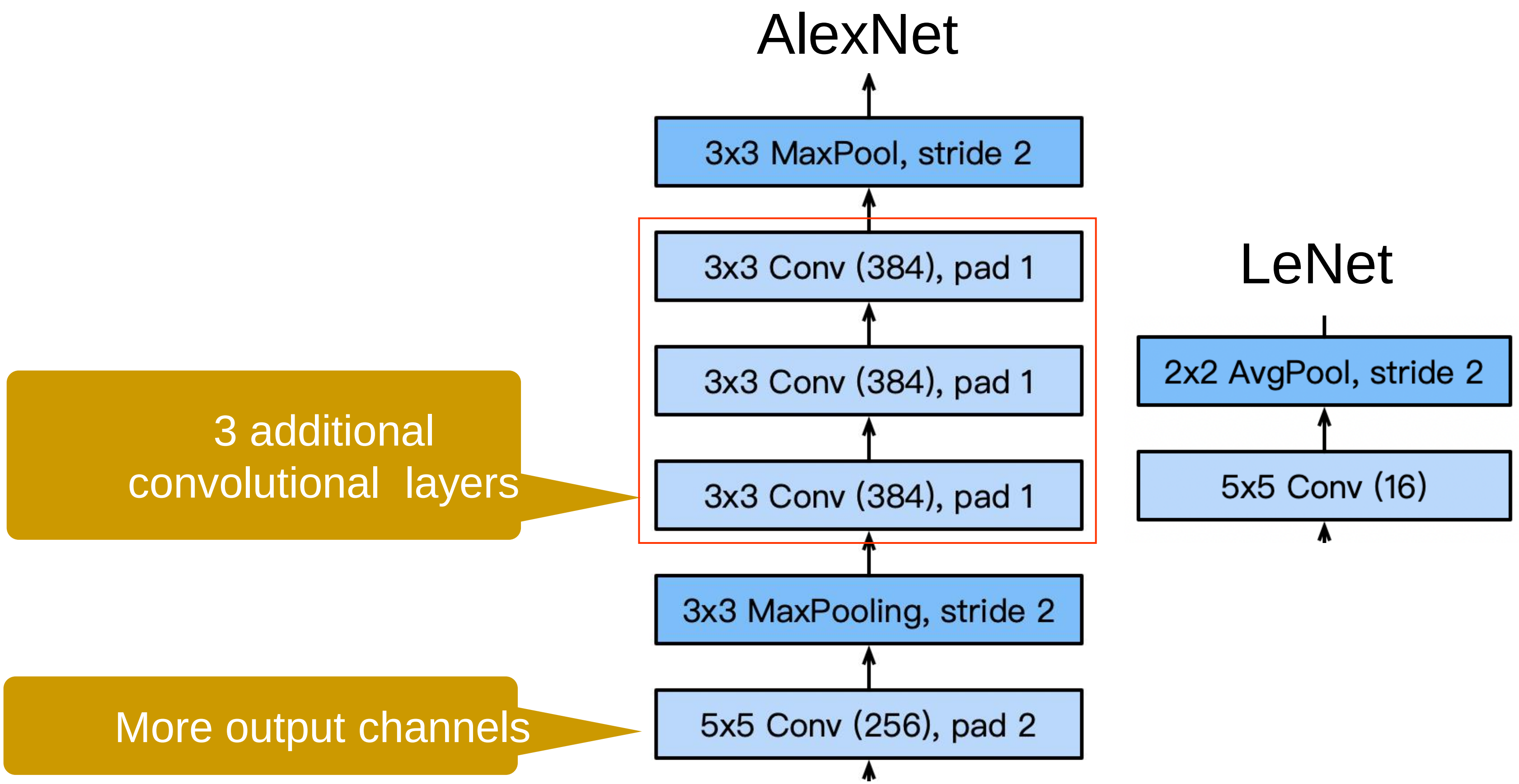
Larger kernel size, stride because of the increased image size, and more output channels.



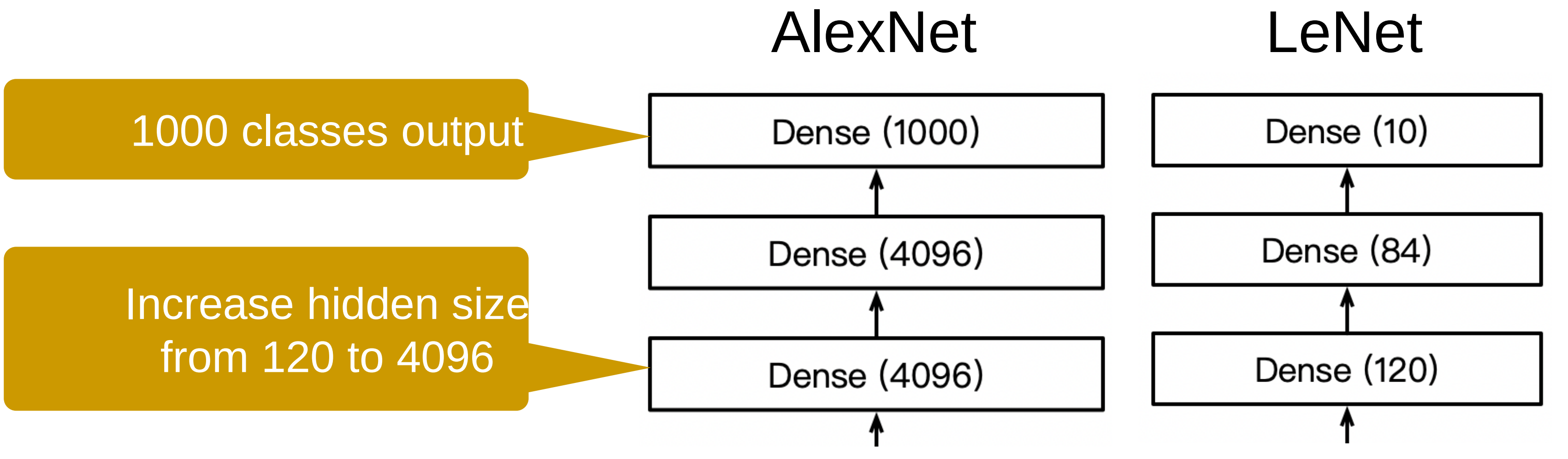
ImageNet (2010)

Images	Color images with nature objects	Gray image for hand-written digits
Size	469 x 387	28 x 28
# examples	1.2 M	60 K
# classes	1,000	10

AlexNet Architecture



AlexNet Architecture

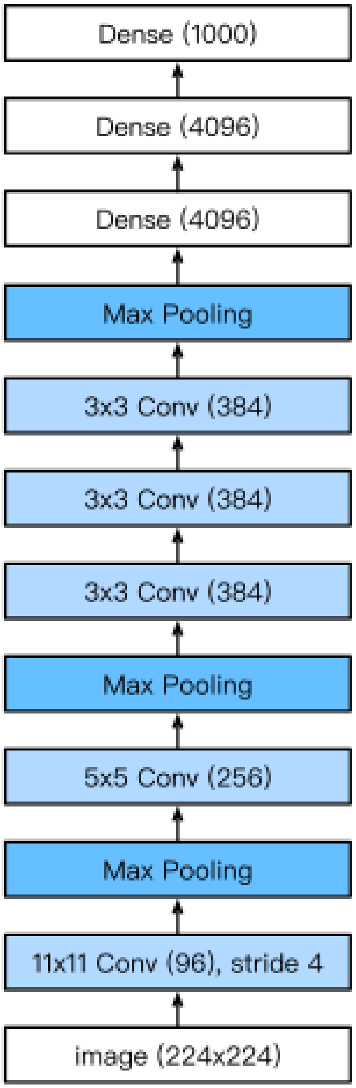


More Tricks

- Change activation function from sigmoid to ReLu (no more vanishing gradient)
- Add a dropout layer after two hidden dense layers (better robustness / regularization)
- Data augmentation

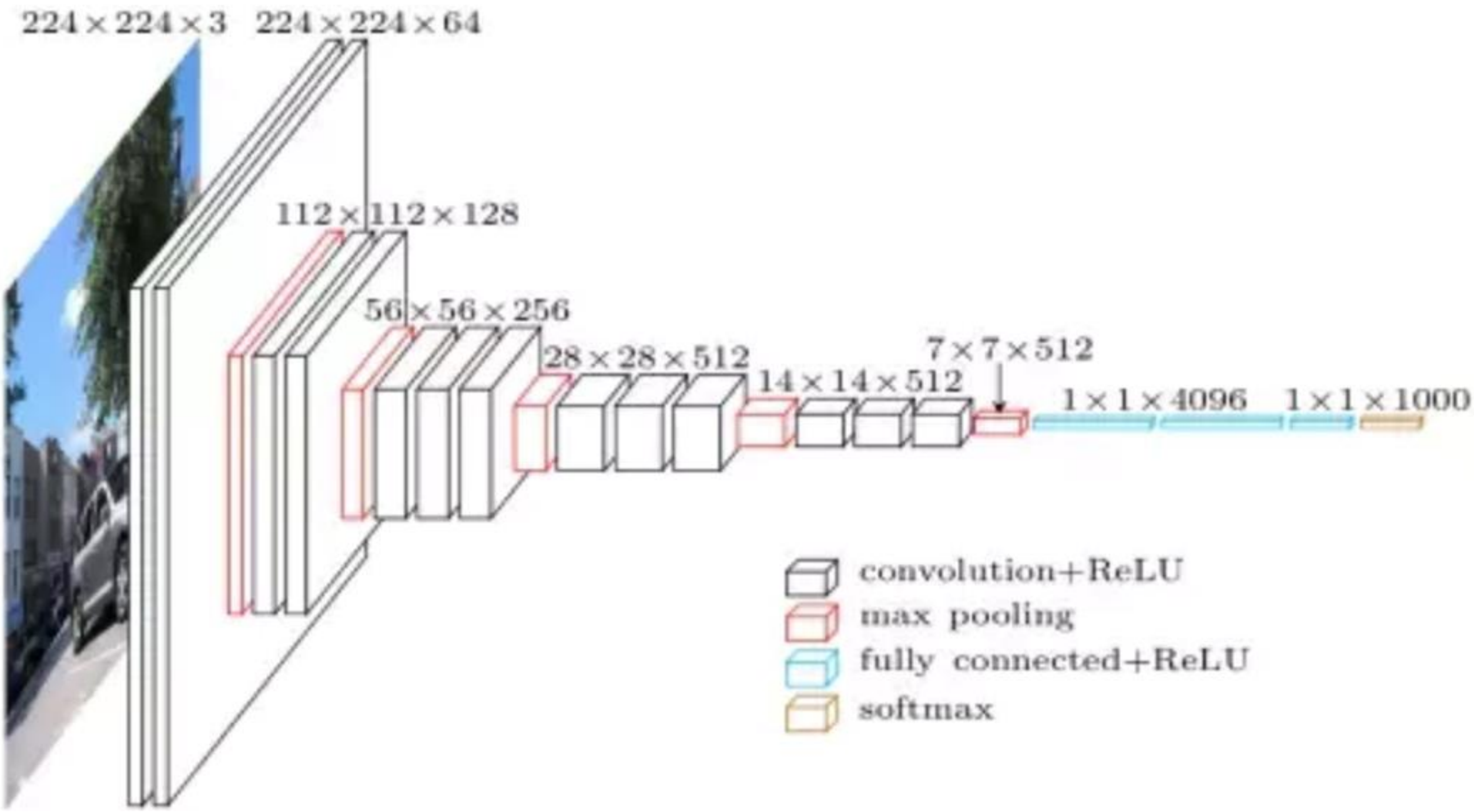
Complexity

	#parameters		FLOP	
	AlexNet	**LeNet**	**AlexNet**	**LeNet**
Conv1	35K	150	101M	1.2M
Conv2	614K	2.4K	415M	2.4M
Conv3-5	3M		445M	
Dense1	26M	0.48M	26M	0.48M
Dense2	16M	0.1M	16M	0.1M
Total	46M	0.6M	1G	4M
Increase	11x	1x	250x	1x



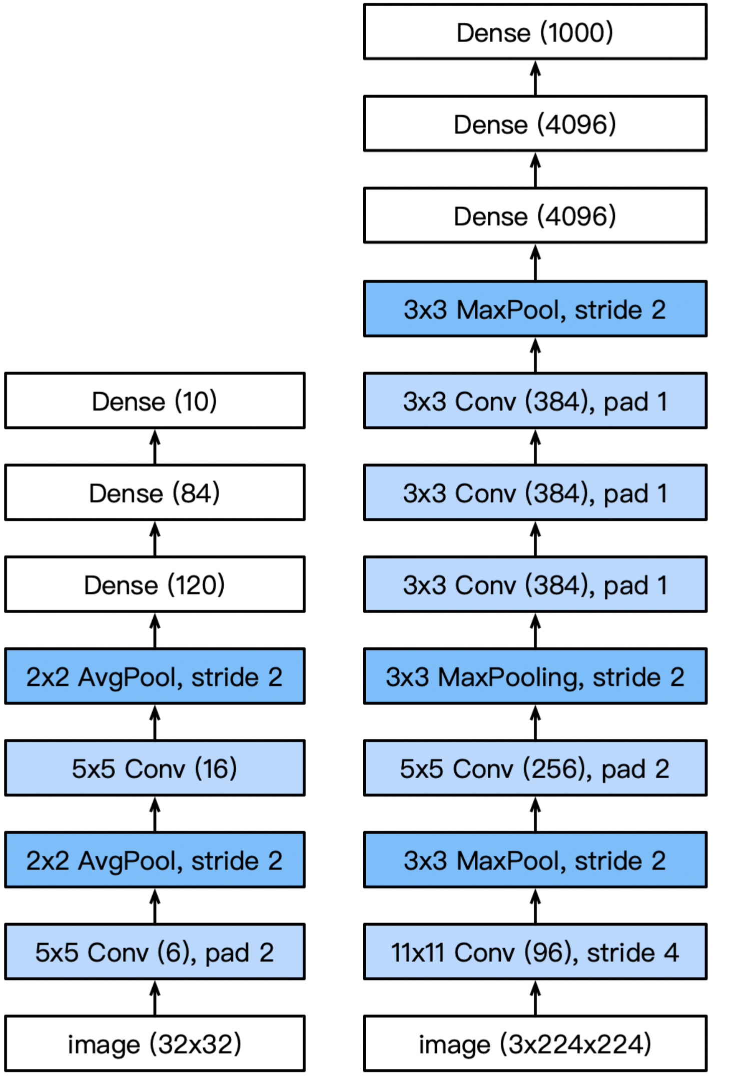
VGG

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

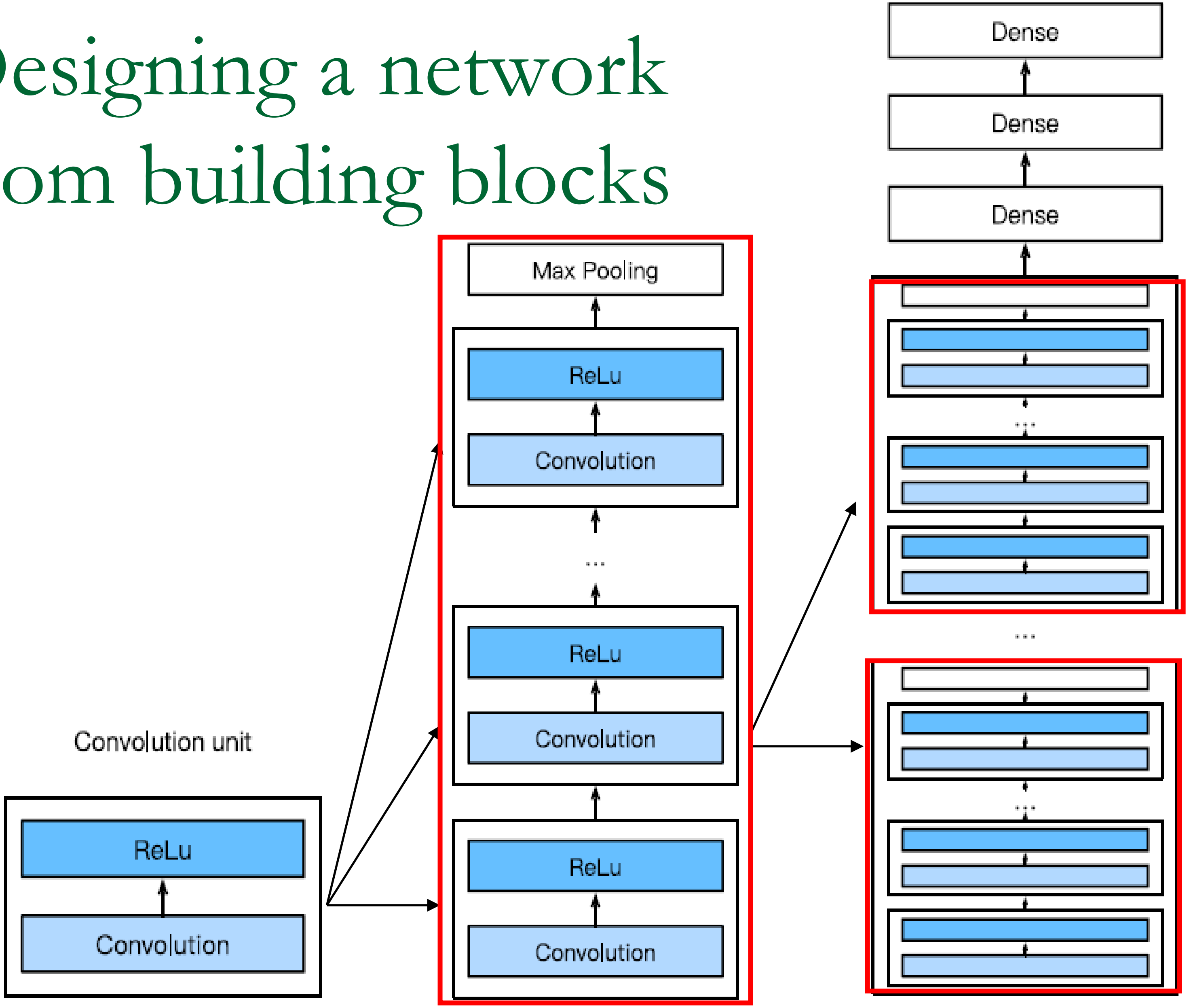


How to design new networks

- AlexNet is deeper and bigger than LeNet to get performance
- Go even bigger & deeper?
- Options
  - **More** dense layers (too expensive)
  - **More** convolutions
  - Group into **blocks**

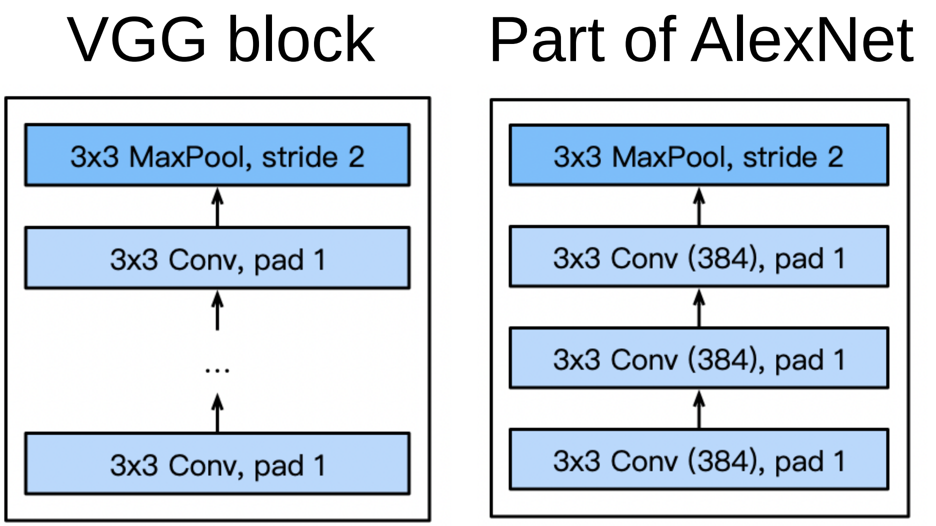


Designing a network from building blocks



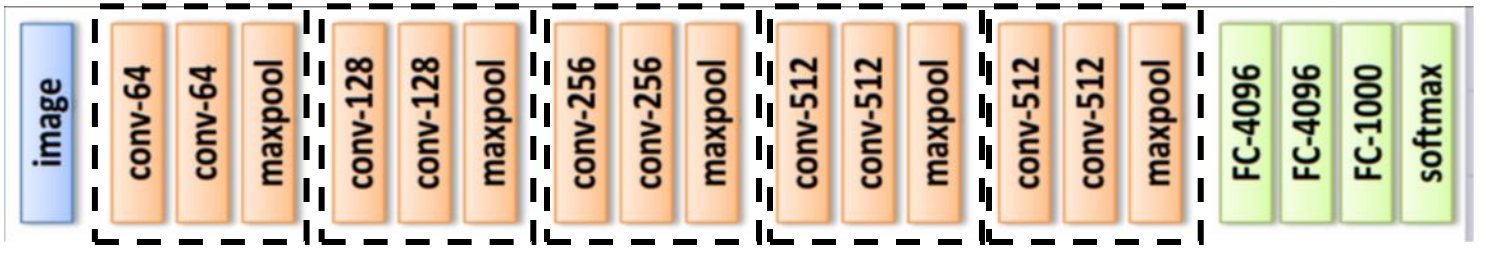
VGG Blocks

- Deeper vs. wider?
  - 5x5 convolutions
  - 3x3 convolutions (more)
  - **deep & narrow better**
- VGG block
  - *3x3* convolutions (pad 1)
    (*n* layers, *m* channels)
  - 2x2 max-pooling
    (stride 2)



VGG Architecture

- Multiple VGG blocks followed by dense layers



- Vary the repeating number to get different architectures, such as VGG-16, VGG-19, …

VGG

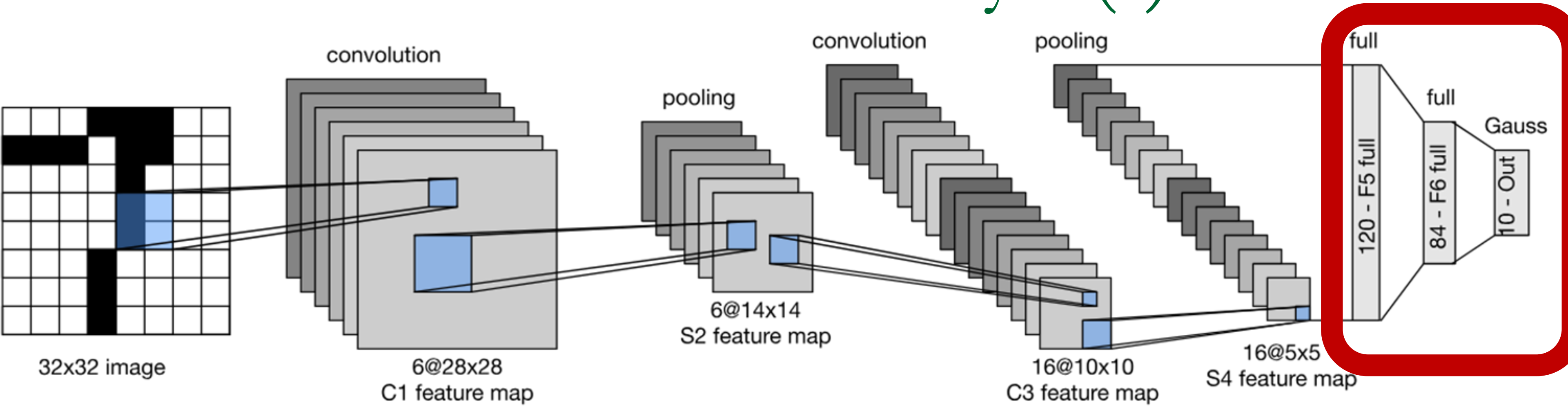
Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Design pattern

- The design pattern of LeNet, AlexNet, and VGG：
  - extract the spatial features through a sequence of convolutions and pooling layers
  - post-process the representations via fully connected layers

The Curse of the Last Layer(s)



- Convolution layers need relatively few parameters

$$c_i \times c_o \times k^2$$

- Last layer needs many parameters for n classes

$$c \times m_w \times m_h \times n$$

- LeNet: 16x5x5x120 = 48k
- VGG: 512x7x7x4096 = 102M

Design pattern

- An alternative design pattern：to use fully connected layers much earlier in the process
  - A careless use of a dense layer would destroy the spatial structure of the data entirely

The inputs and outputs of convolutional layers are usually four-dimensional arrays (example, channel, height, width）

Not Match！

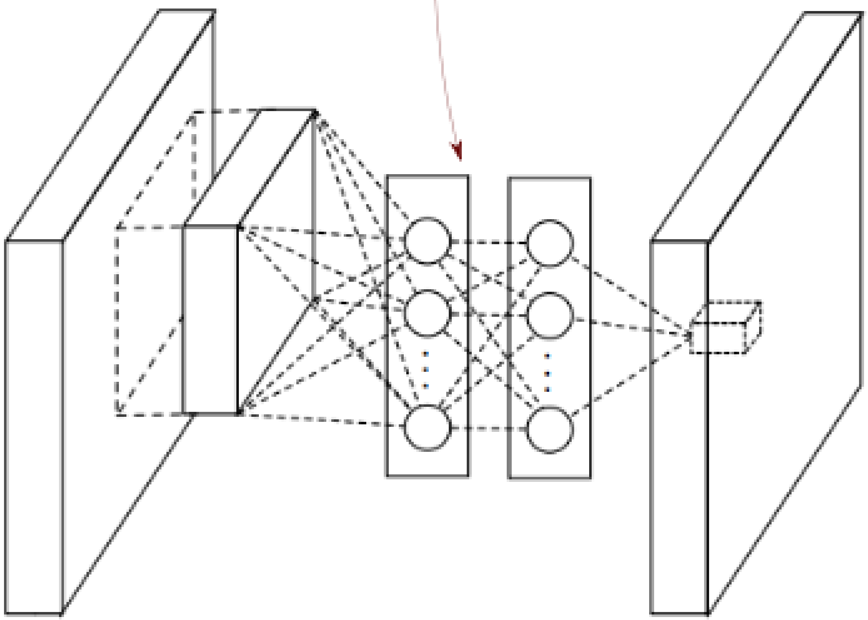
Once we process data by a fully connected layer, it’s virtually impossible to recover the spatial structure of the representation.

Apply a fully connected layer at a pixel level

The inputs and outputs of fully connected layers are usually two dimensional arrays (example, feature)

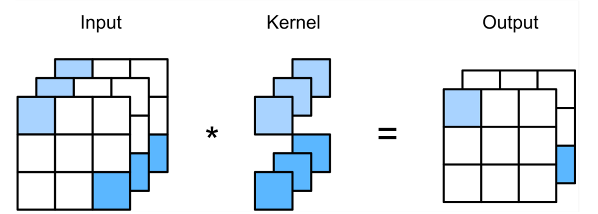
Network in Network

Non linear mapping
introduced by mlpconv
layer consisting of multiple
fully connected layers with
non linear activation
function.



Breaking the Curse of the Last Layer

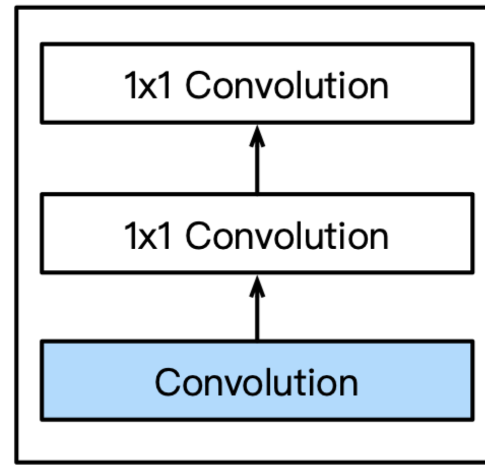
- Key Idea
  - **Get rid of the fully connected layer(s)**
  - Convolutions and pooling reduce resolution (e.g. stride of 2 reduces resolution 4x)
- Implementation details



  - Reduce resolution progressively
  - Increase number of channels
  - Use **1x1 convolutions** (they only act per pixel)
- **Global average pooling in the end**

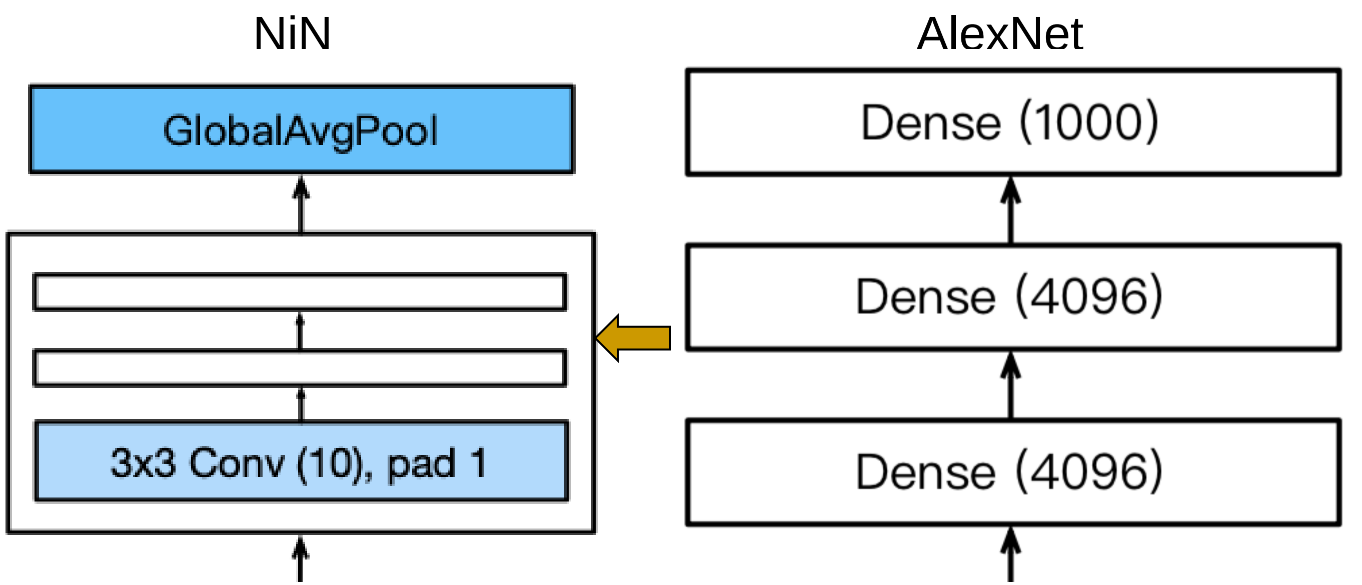
NiN Block

- A convolutional layer
  - kernel size, stride, and padding are hyper-parameters
- Followed by two 1x1 convolutions
  - 1 stride and no padding, share the same output channels as the first layer
  - Act as dense layers

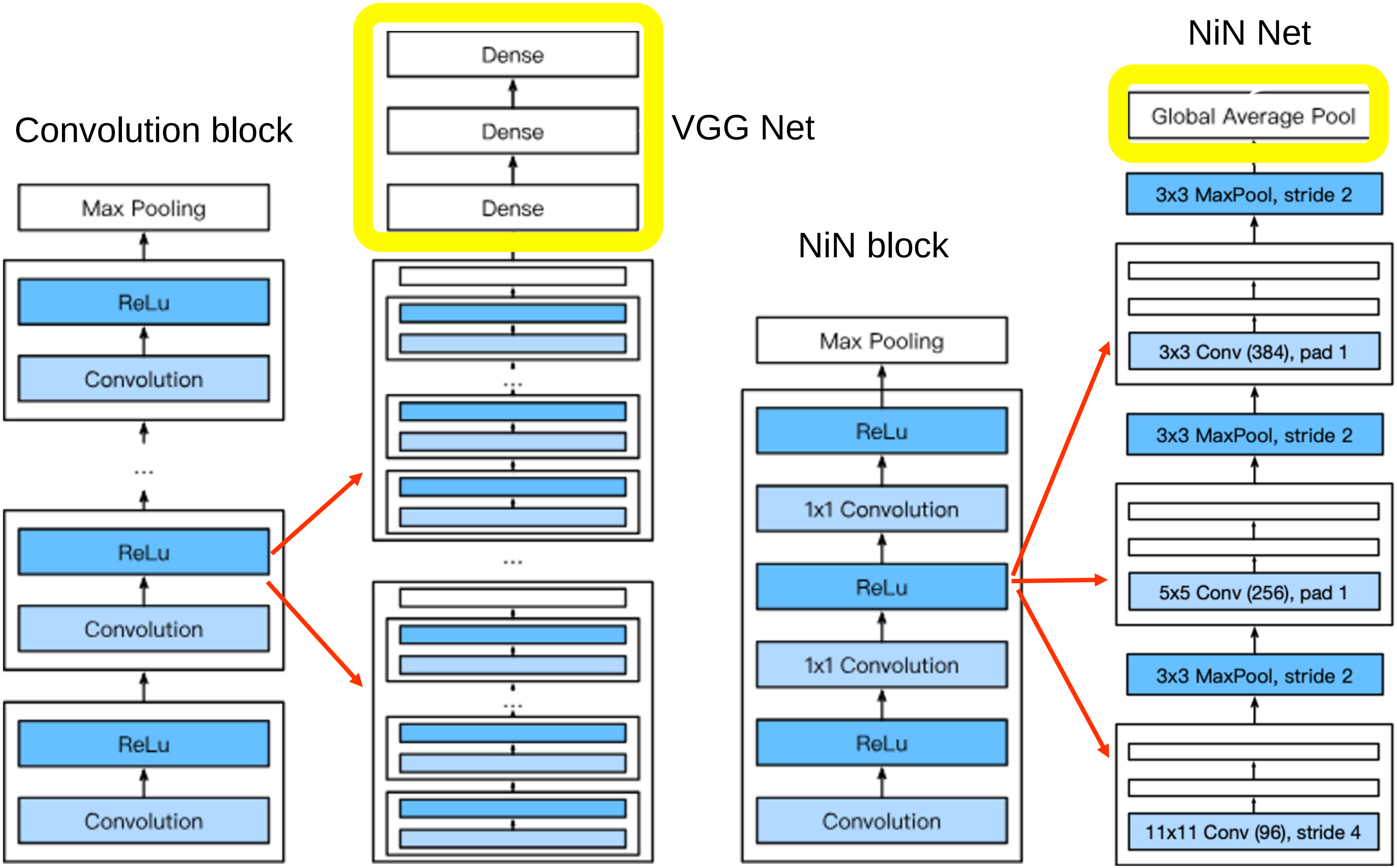


NiN Last Layers

- Replace AlexNet's dense layers with a NiN block
- Global average pooling layer to combine outputs



NiN Networks

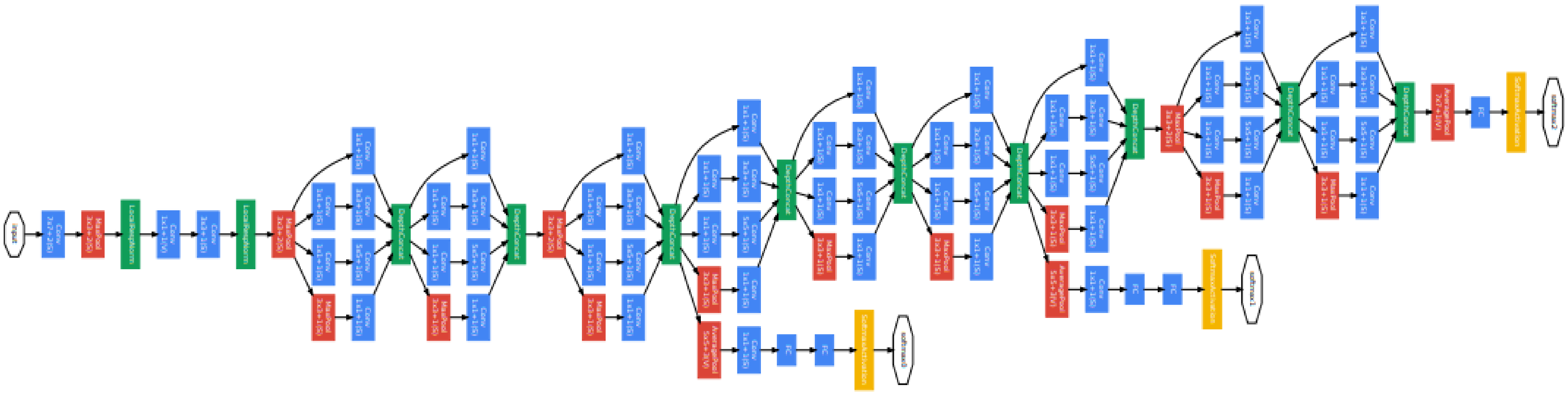


NiN Networks Summary

- Reduce image resolution progressively
- Increase number of channels
- Global average pooling for given numbers of classes

GoogLeNet(2014)

- Networks with Parallel Concatenations

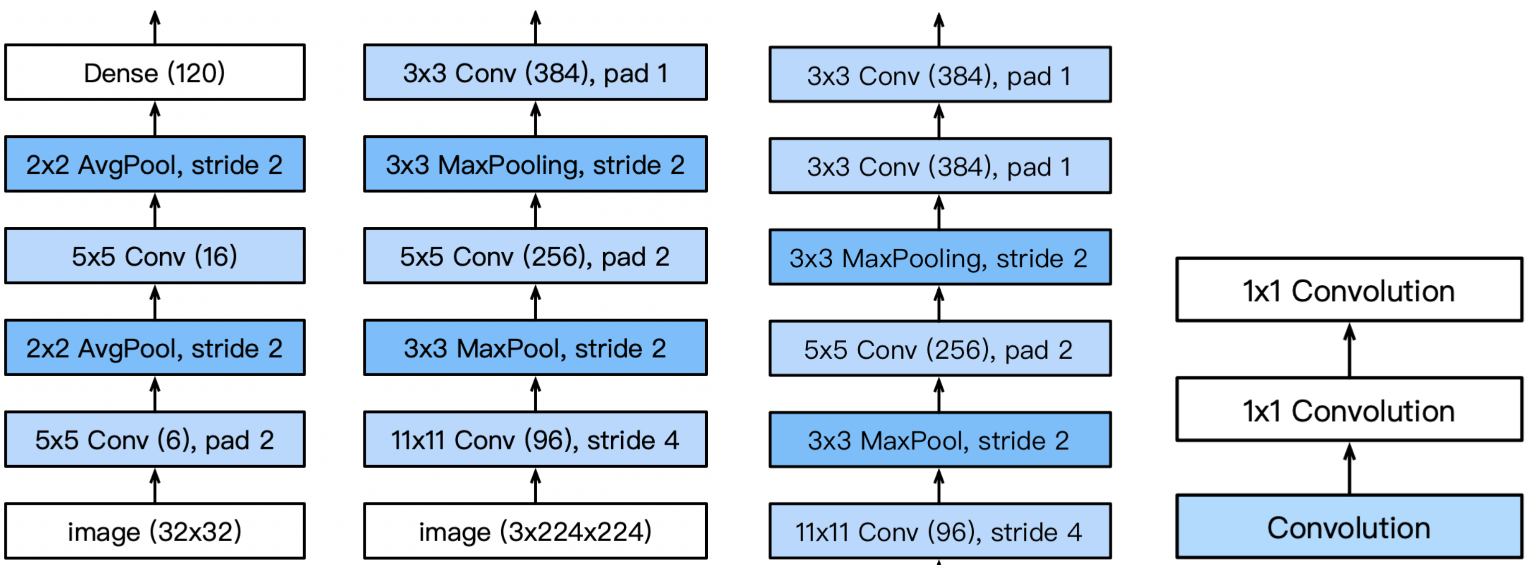


Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., & Anguelov, D. & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Picking the best convolution …

1x1 | 3x3 | 5x5 | 11x11 | Multiple 1x1

LeNet | AlexNet | VGG | NiN

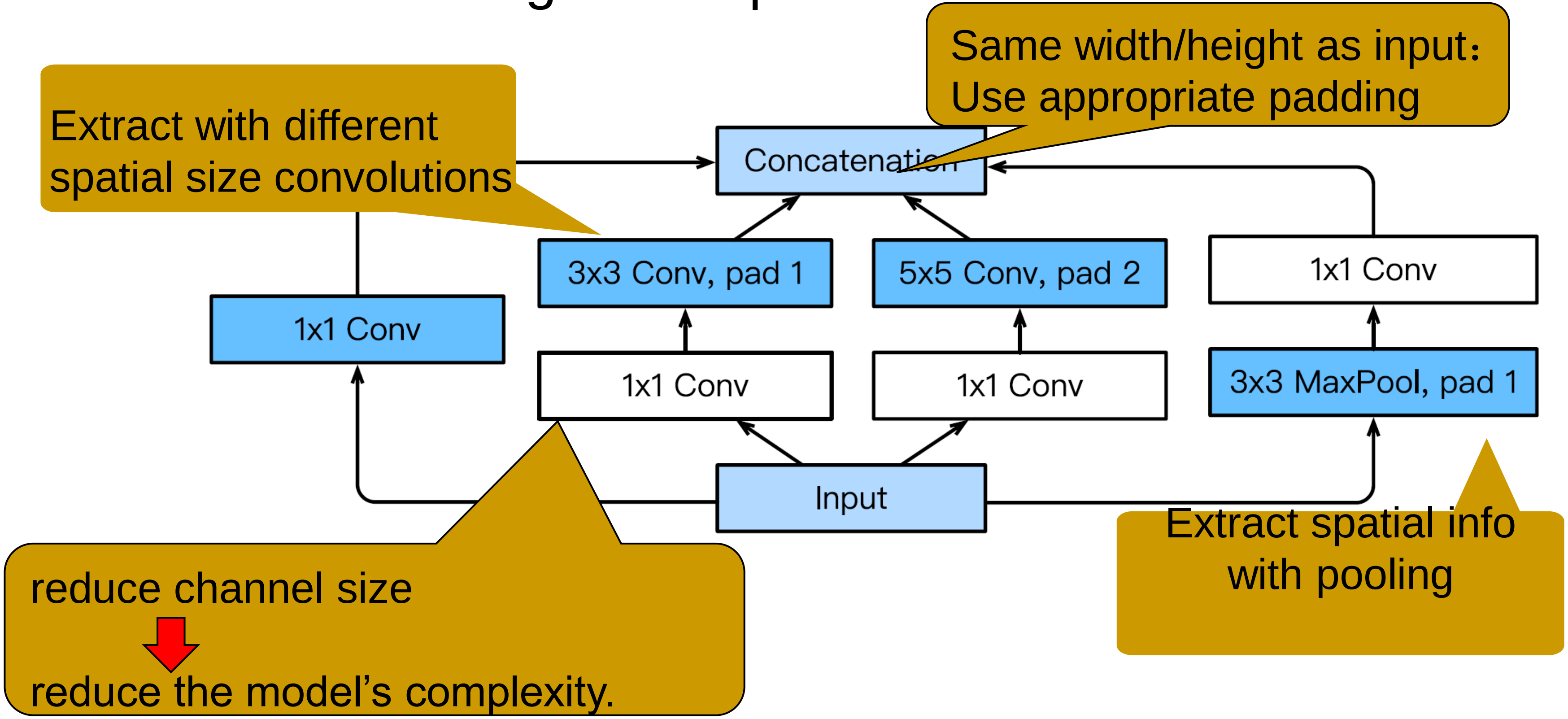


Why choose? Just pick them all !

Inception Blocks

- ✓ The basic convolutional block in GoogLeNet.

4 paths extract information from different aspects, then concatenate along the output channel

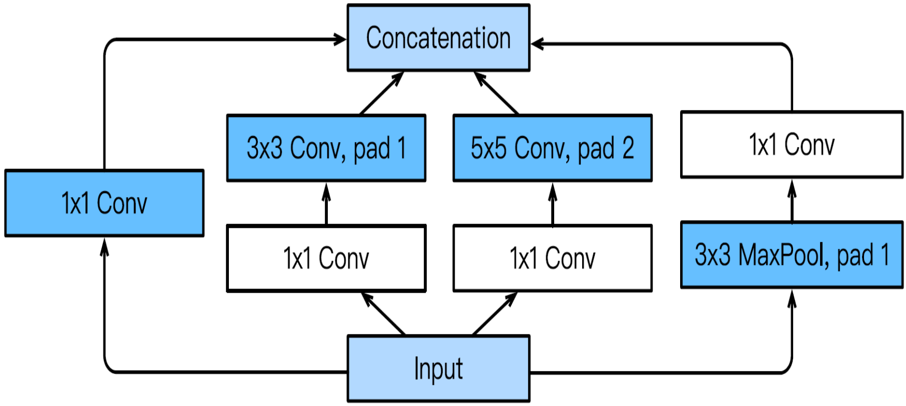


Inception Blocks

Inception blocks have fewer parameters and less computation complexity than a single 3x3 or 5x5 convolutional layer

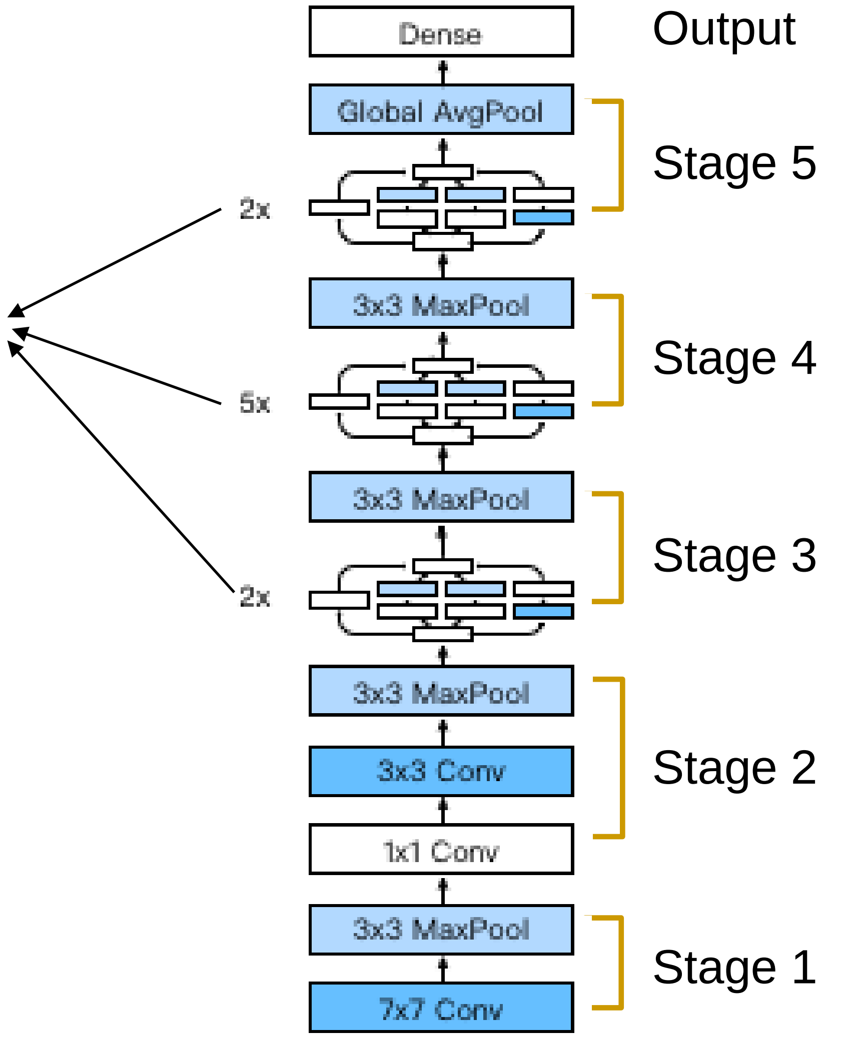
- Mix of different functions (powerful function class)
- Memory and compute efficiency (good generalization)

	#parameters	FLOPS
Inception	0.16 M	128 M
3x3 Conv	0.44 M	346 M
5x5 Conv	1.22 M	963 M



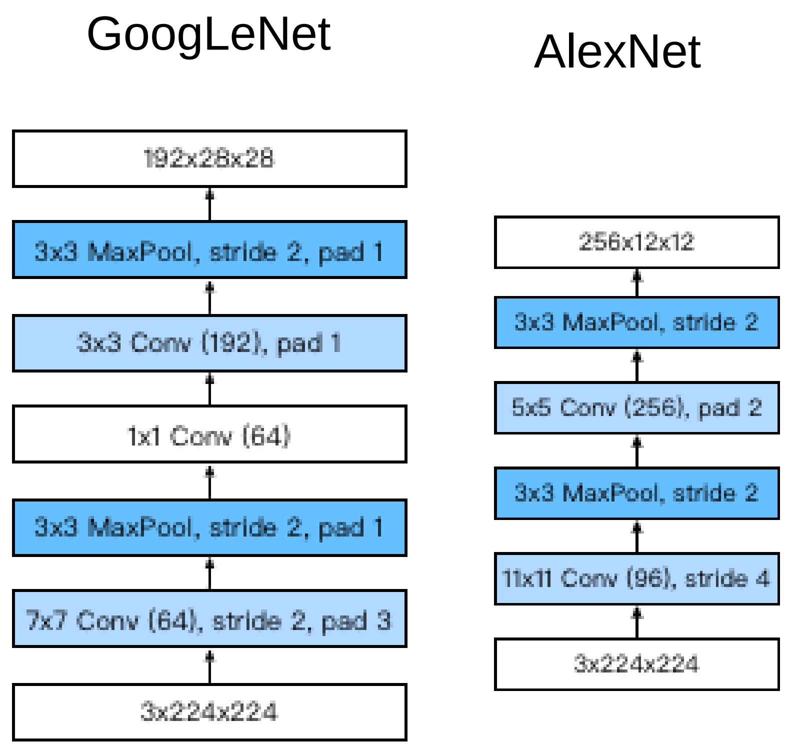
GoogLeNet

- 5 stages with 9 inception blocks

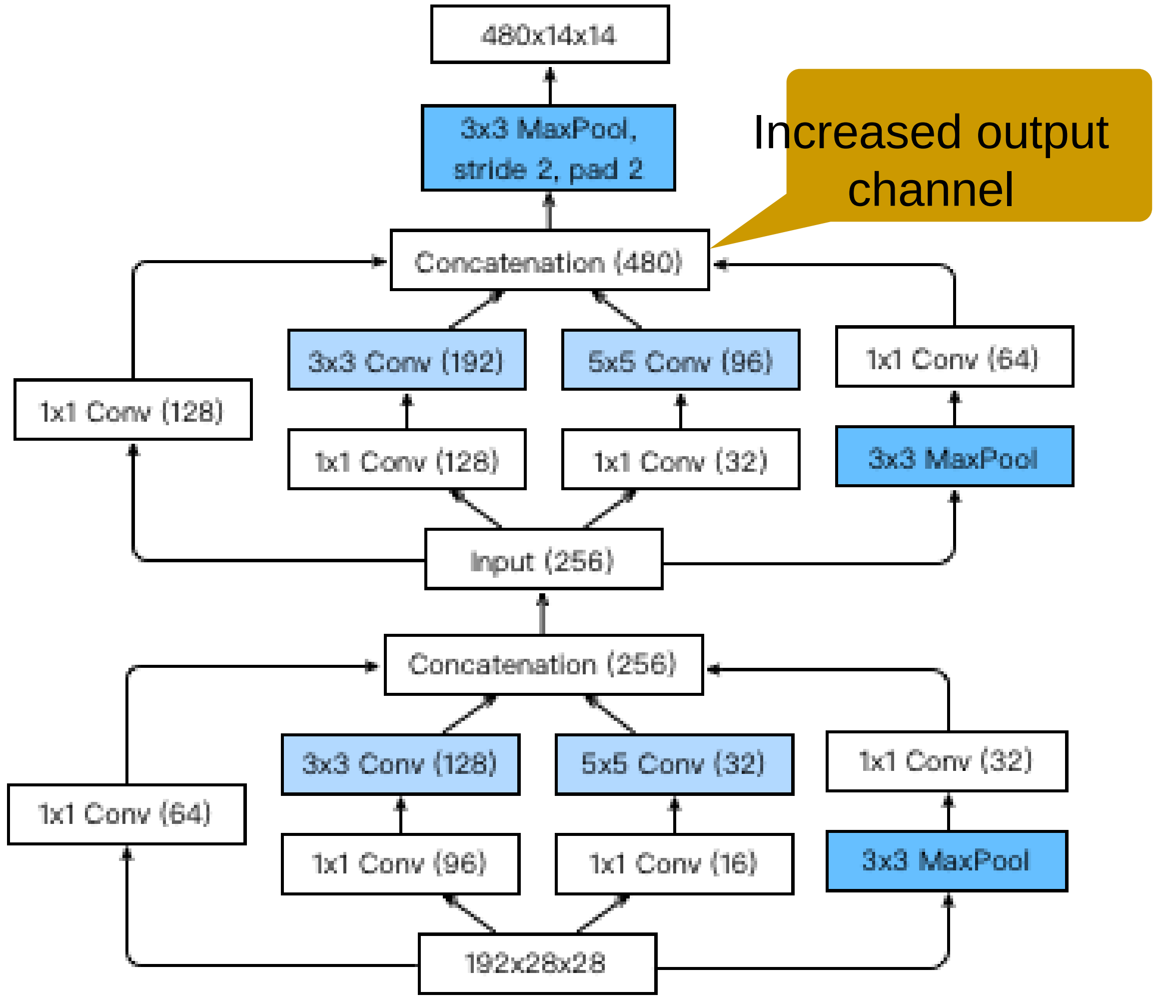


Stage 1 & 2

- Smaller kernel size and output channels due to more layers

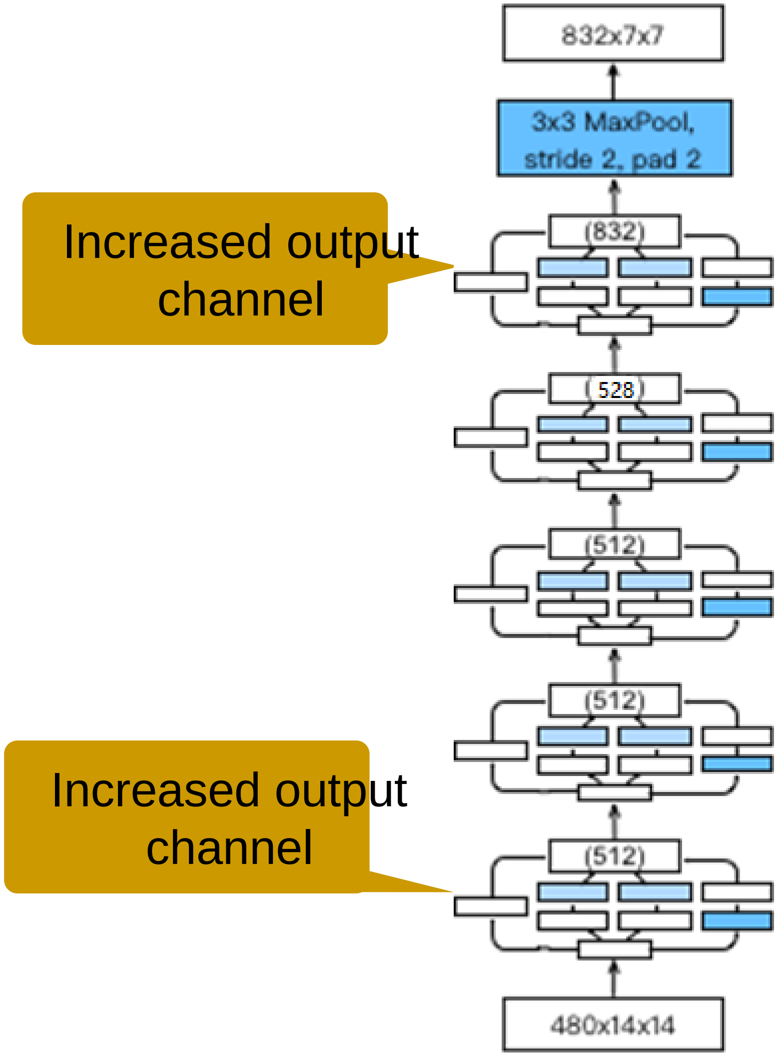


Stage 3



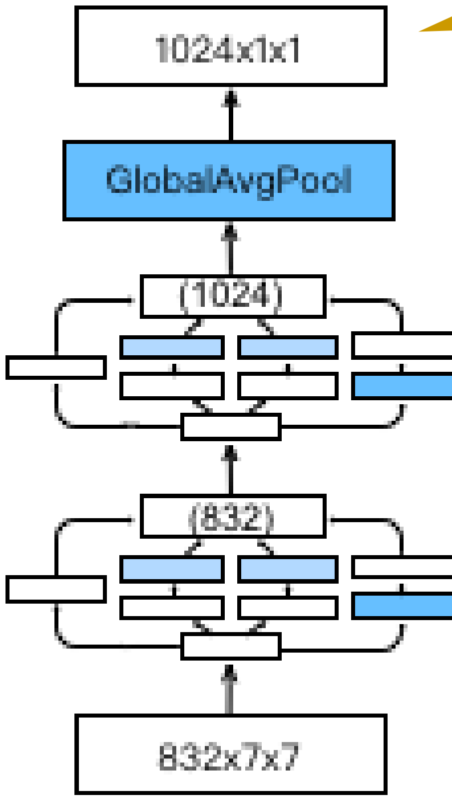
480x14x14
3x3 MaxPool, stride 2, pad 2
Increased output channel
Concatenation (480)
3x3 Conv (192)
5x5 Conv (96)
1x1 Conv (64)
1x1 Conv (128)
1x1 Conv (128)
1x1 Conv (32)
3x3 MaxPool
Input (256)
Concatenation (256)
3x3 Conv (128)
5x5 Conv (32)
1x1 Conv (32)
1x1 Conv (64)
1x1 Conv (96)
1x1 Conv (16)
3x3 MaxPool
192x28x28

Stage 4


832x7x7
3x3 MaxPool, stride 2, pad 2
(832)
(528)
(512)
(512)
(512)
480x14x14
Increased output channel
Increased output channel

Stage 5

1024-dim feature to output layer



Increased output channel

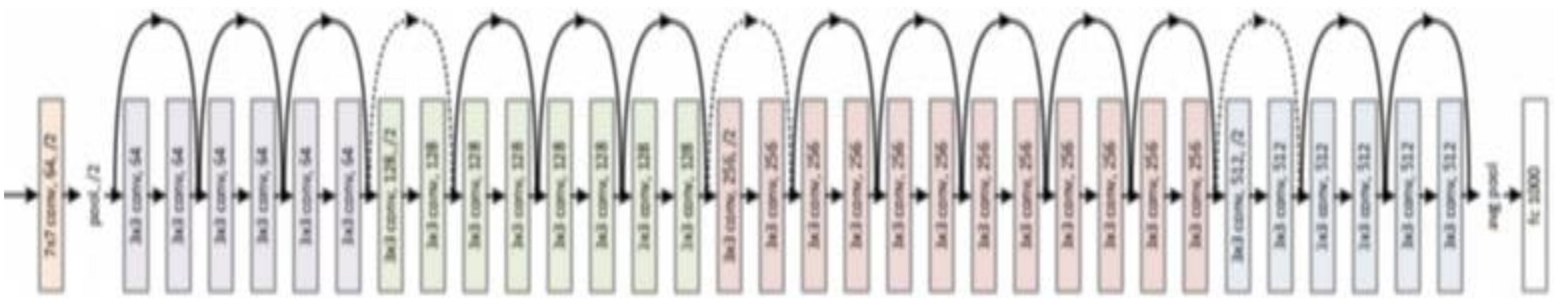
GoogLeNet Incarnation of the Inception Architecture

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

The many flavors of Inception Networks

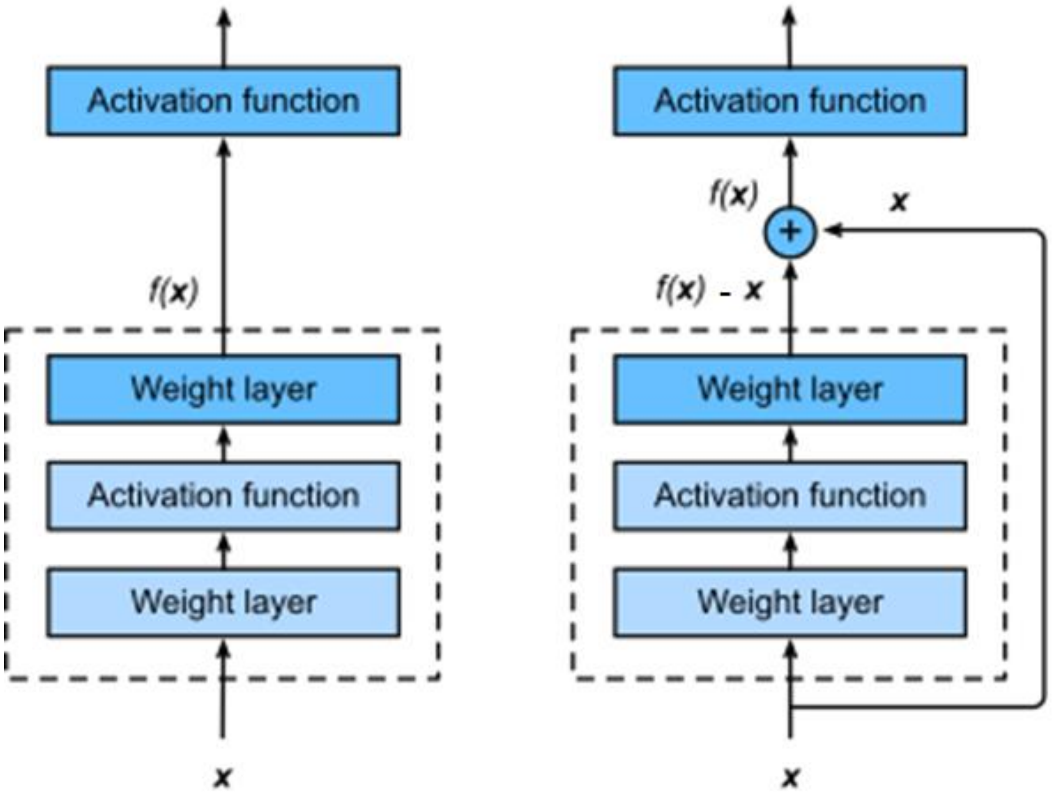
- Inception-BN (v2) - Add batch normalization
- Inception-V3 - Modified the inception block
  - Replace 5x5 by multiple 3x3 convolutions
  - Replace 5x5 by 1x7 and 7x1 convolutions
  - Replace 3x3 by 1x3 and 3x1 convolutions
  - Generally deeper stack
- Inception-V4 - Add residual connections (more later)

Residual Networks (ResNet)

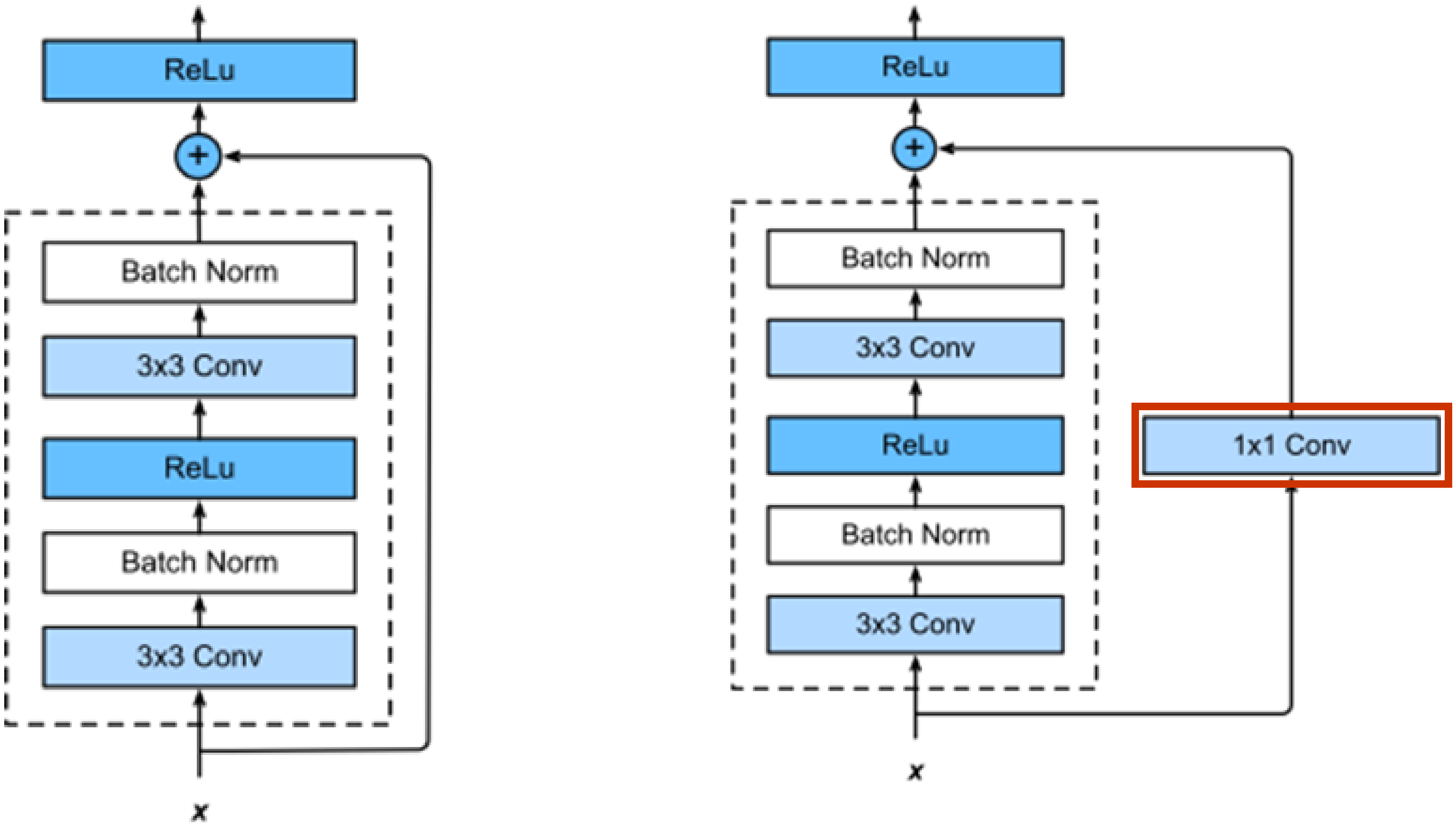


He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. CVPR (pp. 770-778).

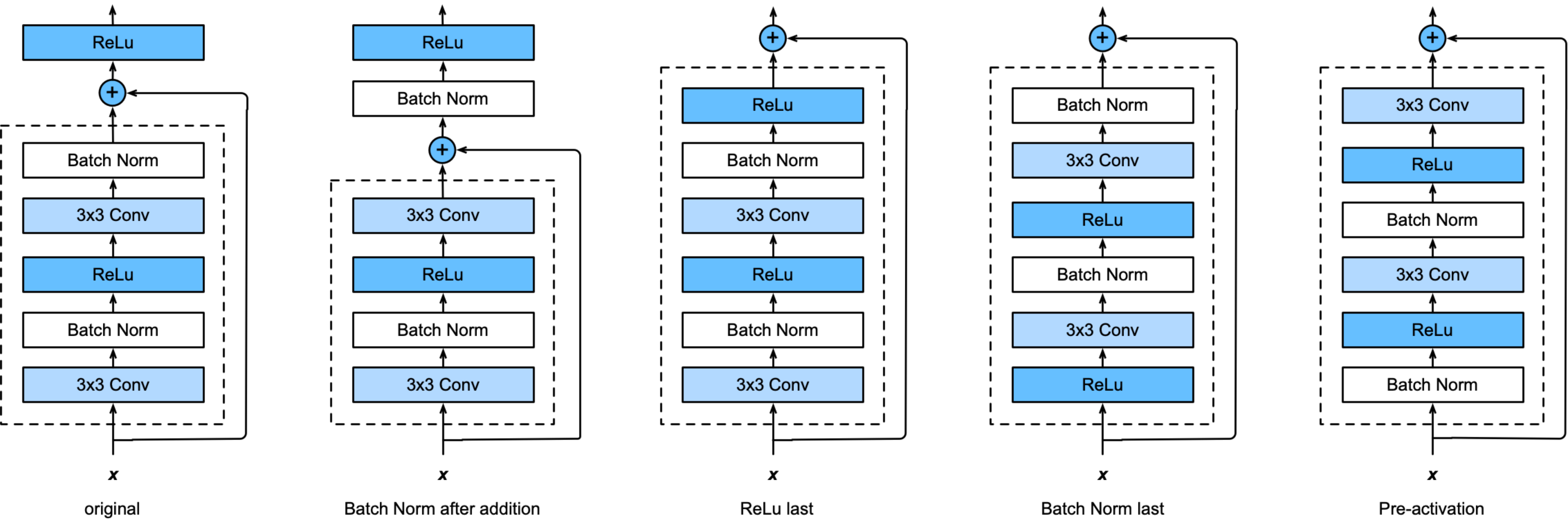
Residual Networks (ResNet)



ResNet Block in detail



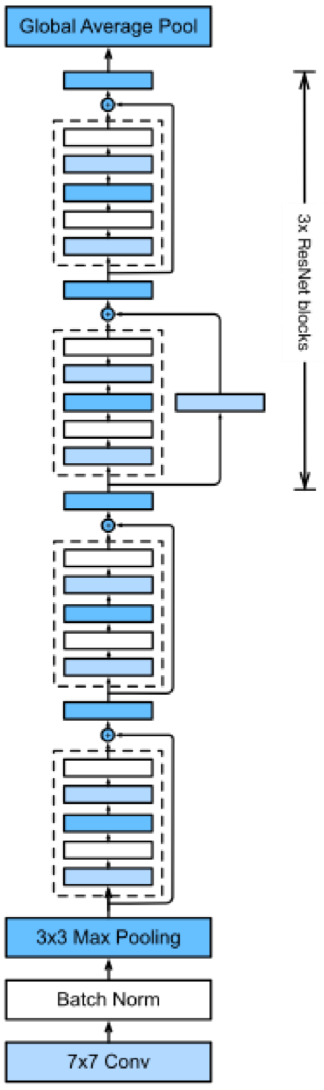
The many flavors of ResNet blocks



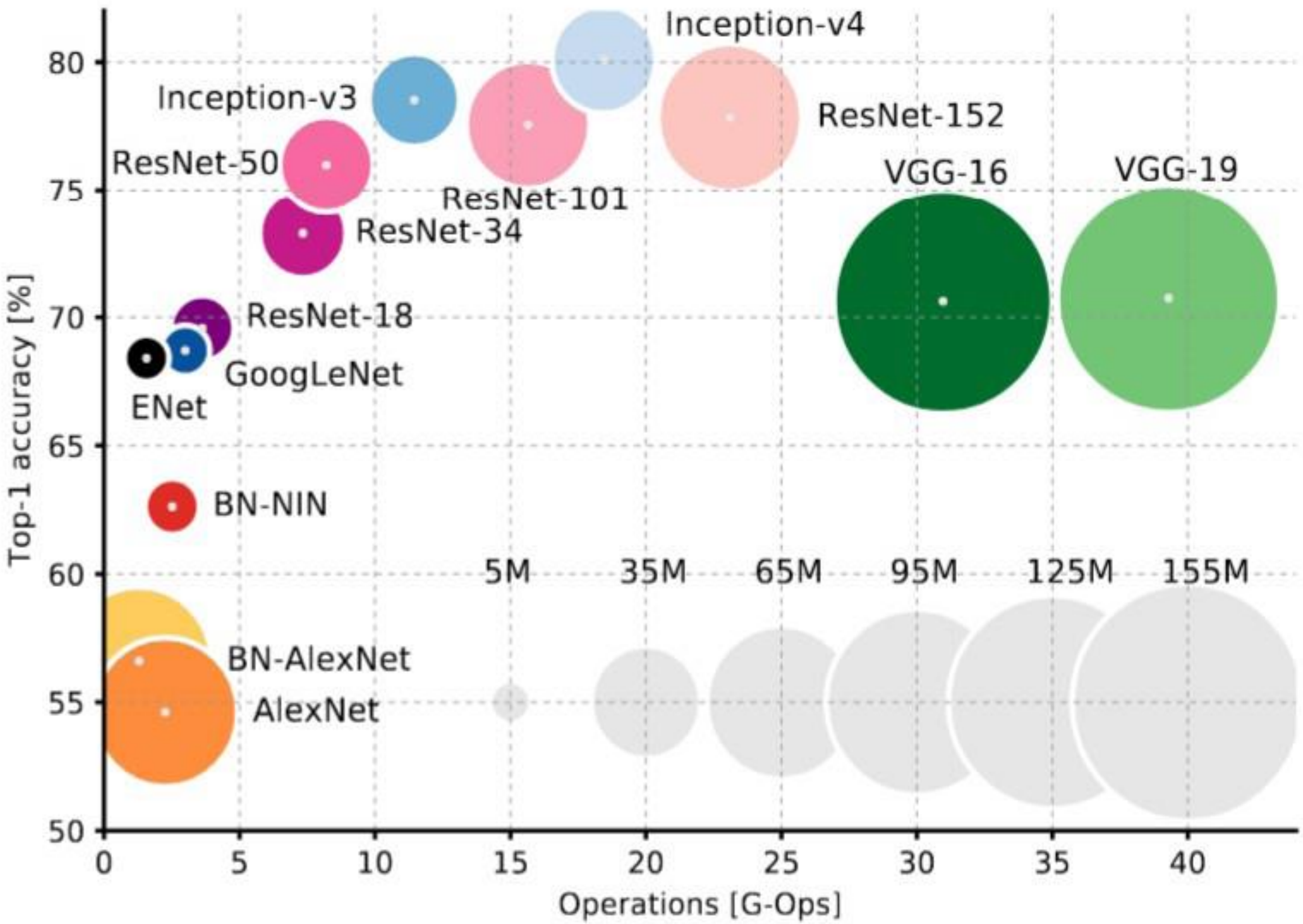
Try every permutation

ResNet

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to the expressiveness
- Pooling/stride for dimensionality reduction
  - Down sample per module (stride=2)
- Batch Normalization for capacity control



GOPS vs. Accuracy on ImageNet vs. #Parameters



Another Scenario

- Data is not always generated i.i.d., all drawn from some distribution, but follows sequential order

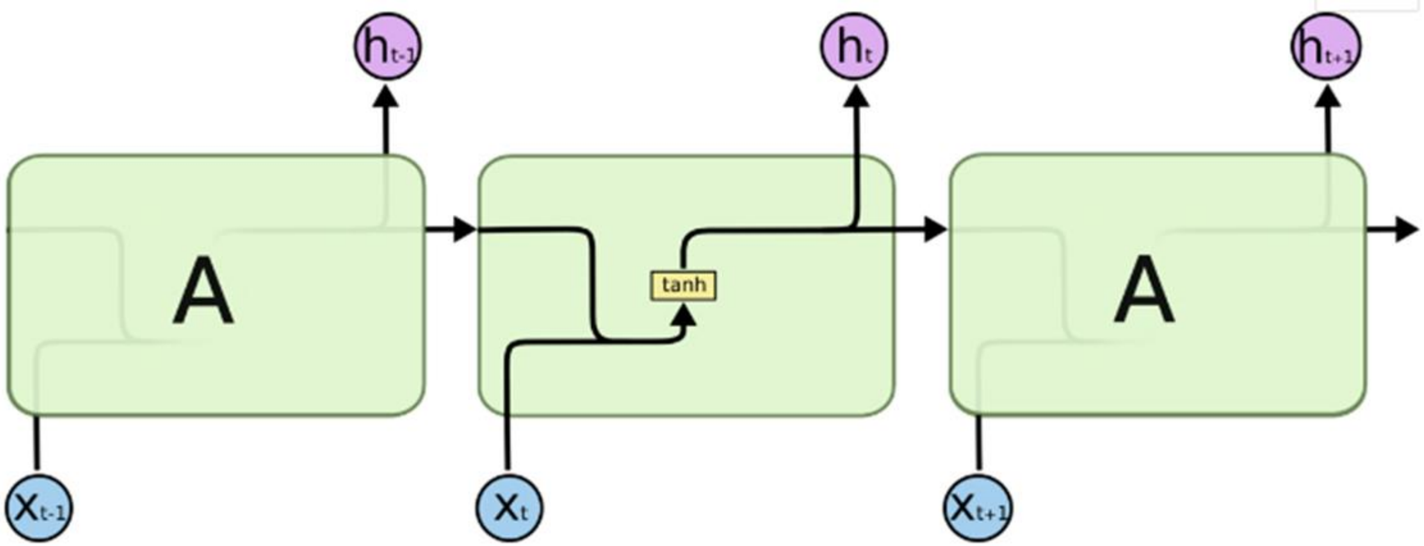
  e.g.

  - the words in a paragraph are written in sequence
  - image frames in a video
  - the audio signal in a conversation
  - the browsing behavior on a website
- Not only receive a sequence as an input, but rather might be expected to continue the sequence.

When the order of data matters……

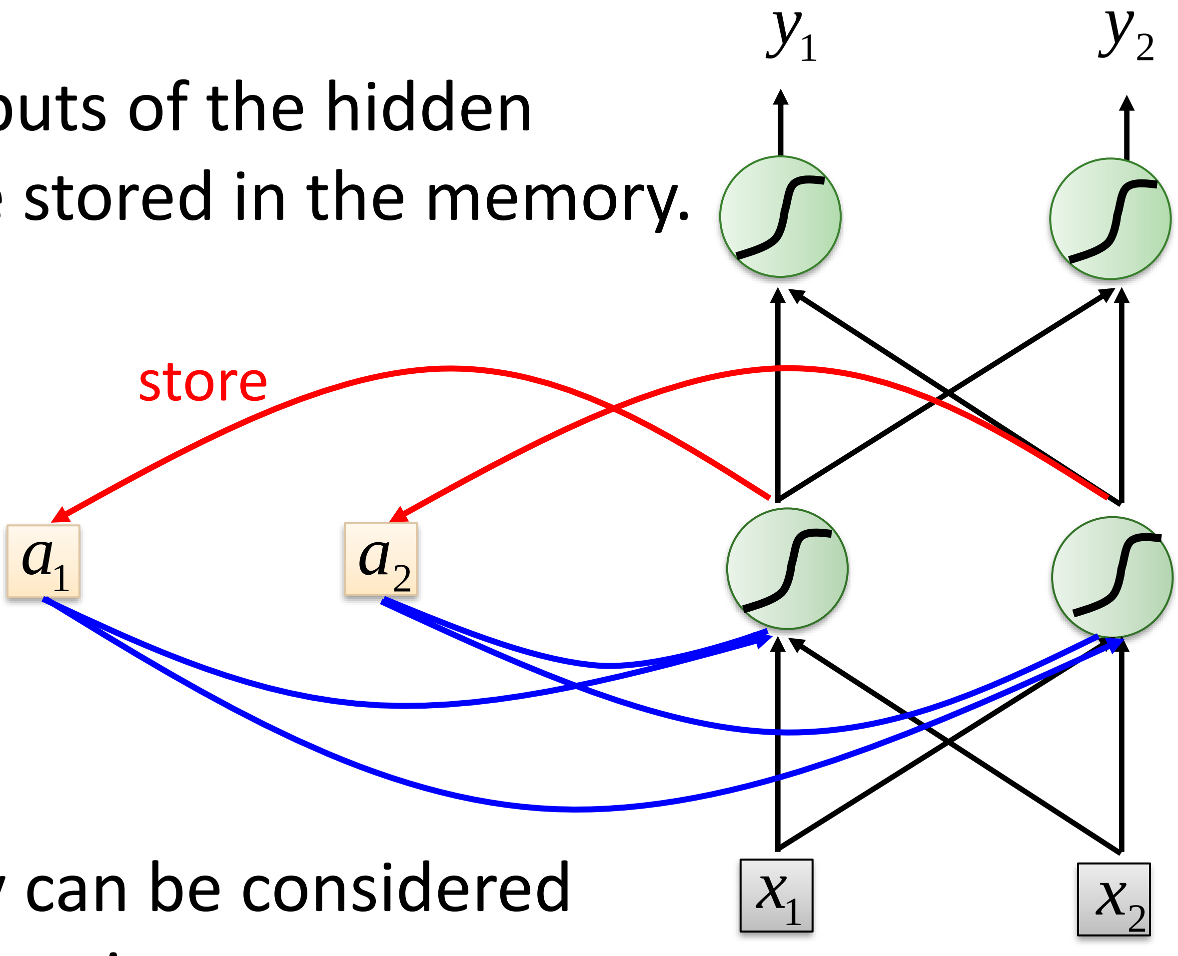
Sequence Models !

Recurrent Neural Networks



Recurrent Neural Network (RNN)

The outputs of the hidden layer are stored in the memory.

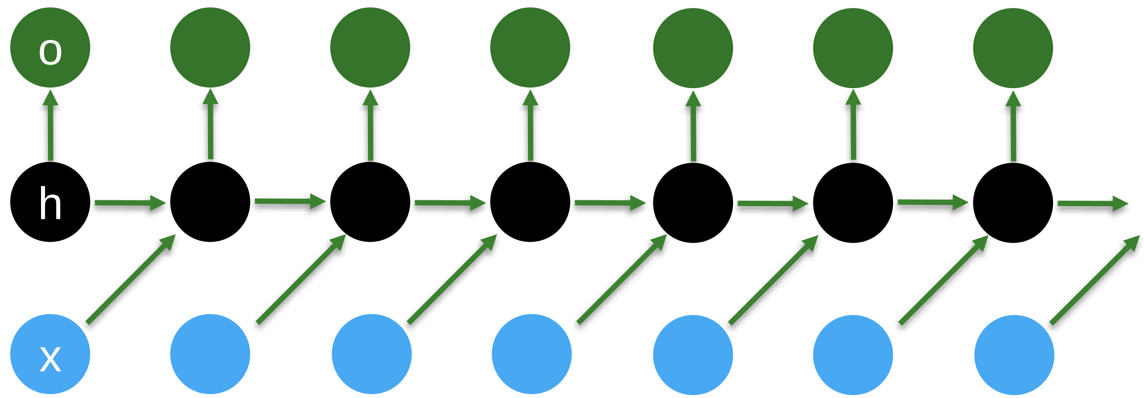


Memory can be considered as another input.

Recurrent Neural Networks

(with hidden state)

Explanation



- Hidden State update

$$\mathbf{h}_t = \phi(\mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{W}_{hx}\mathbf{x}_t + \mathbf{b}_h)$$

- Observation update

$$\mathbf{o}_t = \phi(\mathbf{W}_{ho}\mathbf{h}_t + \mathbf{b}_o)$$

Gradient Explode/ Vanishing

$w = 1 \Rightarrow y^{1000} = 1$

$w = 1.01 \Rightarrow y^{1000} \approx 20000$

Large $\partial L / \partial w$ → Small Learning rate?

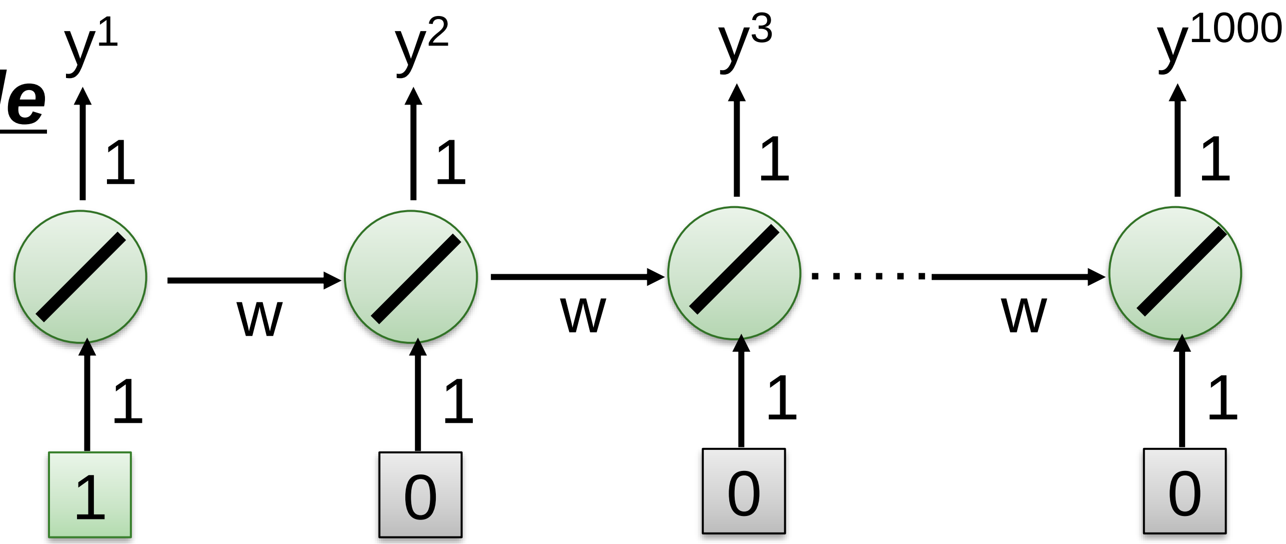
$w = 0.99 \Rightarrow y^{1000} \approx 0$

$w = 0.01 \Rightarrow y^{1000} \approx 0$

small $\partial L / \partial w$ → Large Learning rate?

$=w^{999}$

Toy Example



Recurrent Neural Networks(RNN)

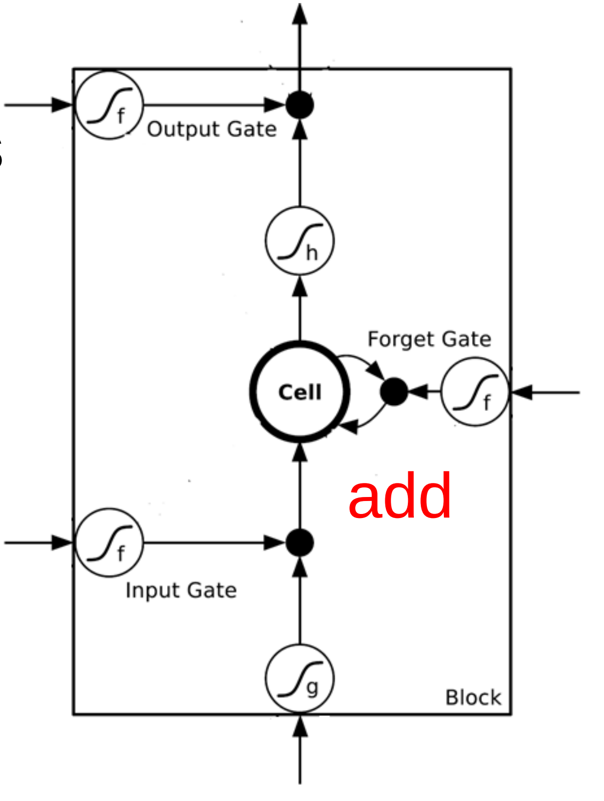
- ✓ Suitable for processing sequences, often applied to the processing of text.
- ❖ Has a problem about gradient vanishing
- ❖ Can not store long-term memory

Helpful Techniques

- Long Short-term Memory (LSTM)
  - Can deal with gradient vanishing (not gradient explode)
    - Memory and input are ***added***
    - The influence never disappears unless the forget gate is closed

  ⇒ No Gradient vanishing

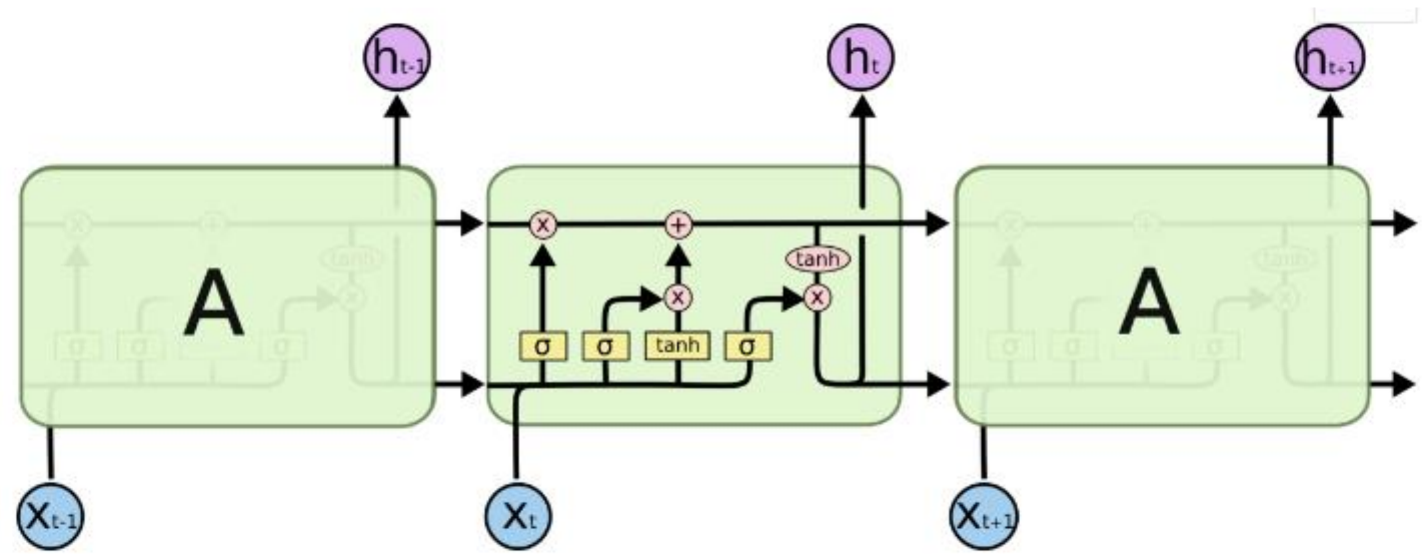
  (If the forget gate is opened.)



- Gated Recurrent Unit(GRU):

  simpler than LSTM

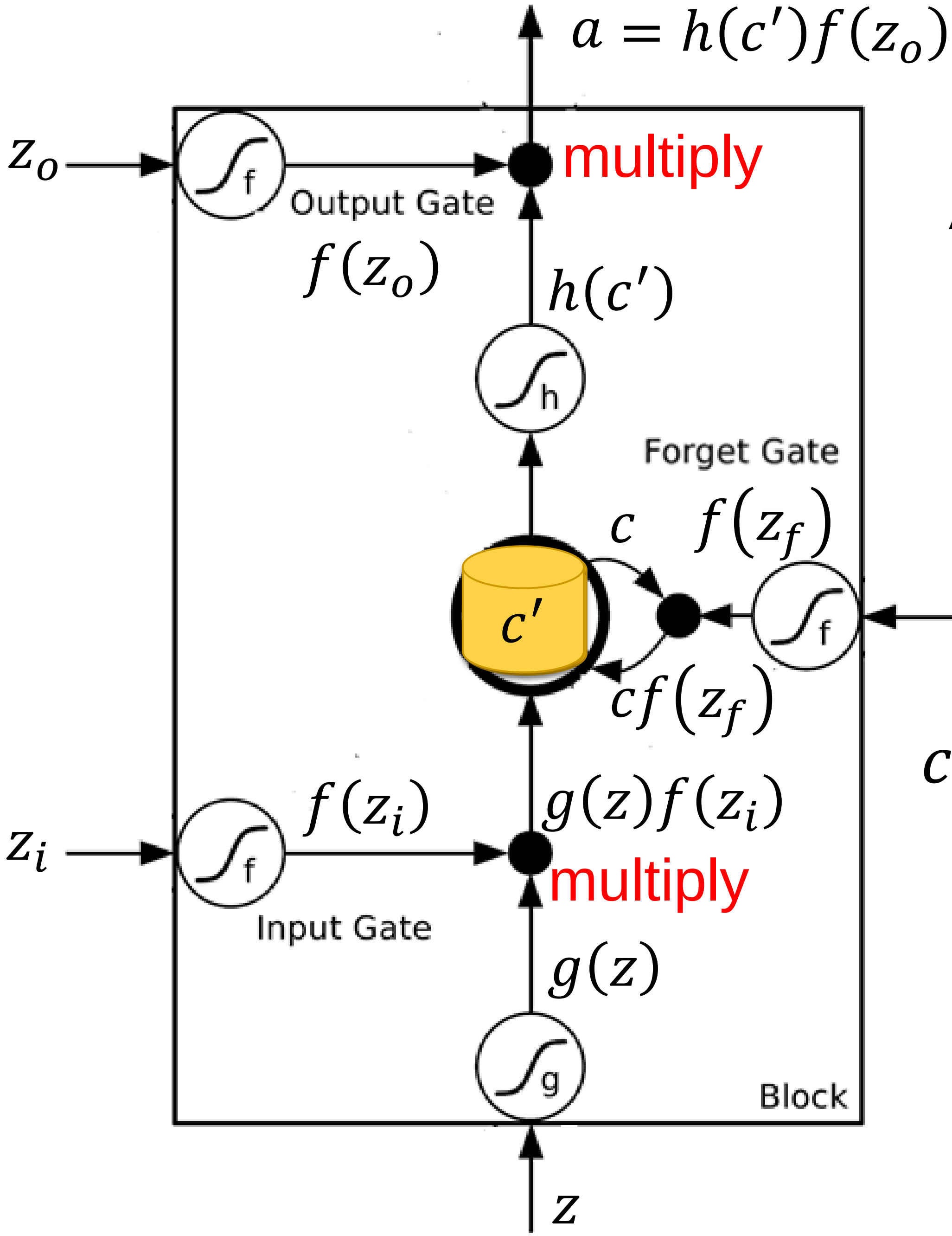
Long Short Term Memory



Hochreiter & Schmidhuber Long Short-term Memory[J]. Neural Computation, 1997,9(8):1735-1780.

Long Short Term Memory

- **Forget gate**
  - Shrink values towards zero
- **Input gate**
  - Decide whether we should ignore the input data
- **Output gate**
  - Decide whether the hidden state is used for the output generated by the LSTM
- **Hidden state** and **Memory cell**



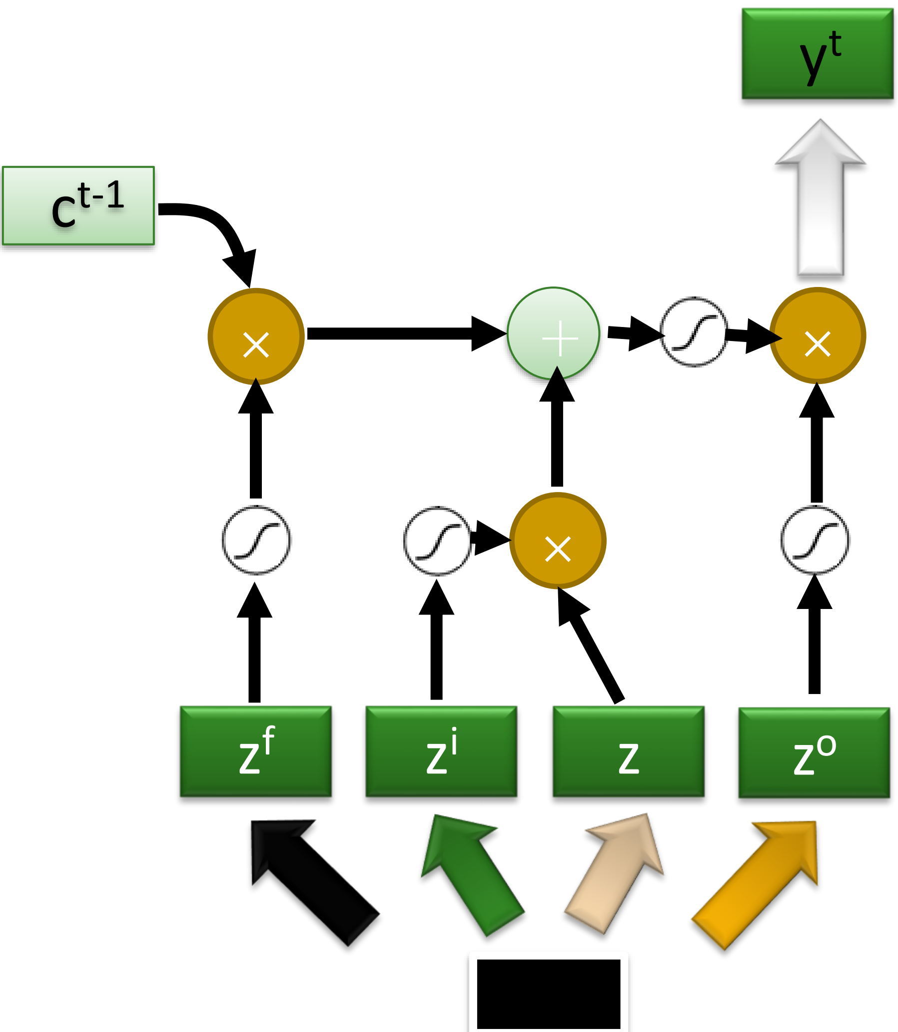
Activation function f is usually a sigmoid function

Between 0 and 1

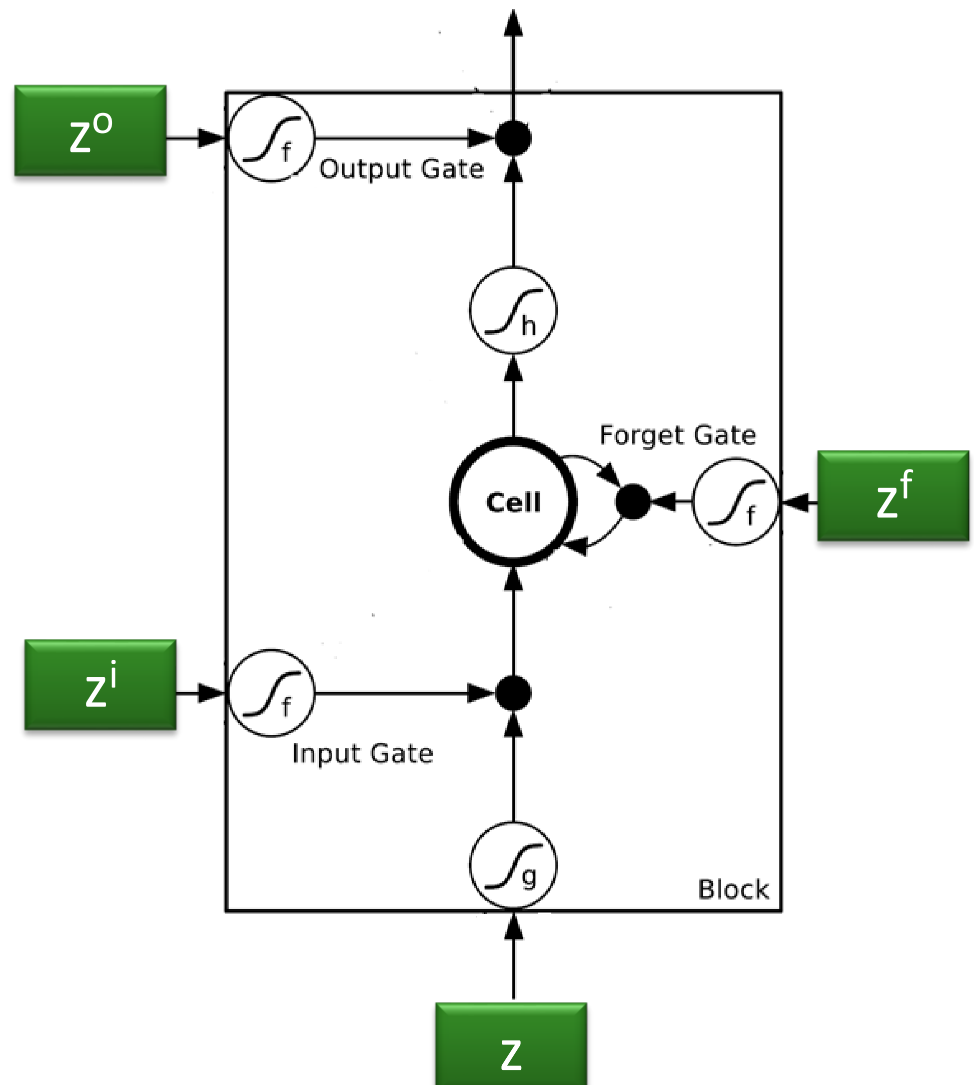
Mimic open and close gate

$$c' = g(z)f(z_i) + cf(z_f)$$

LSTM



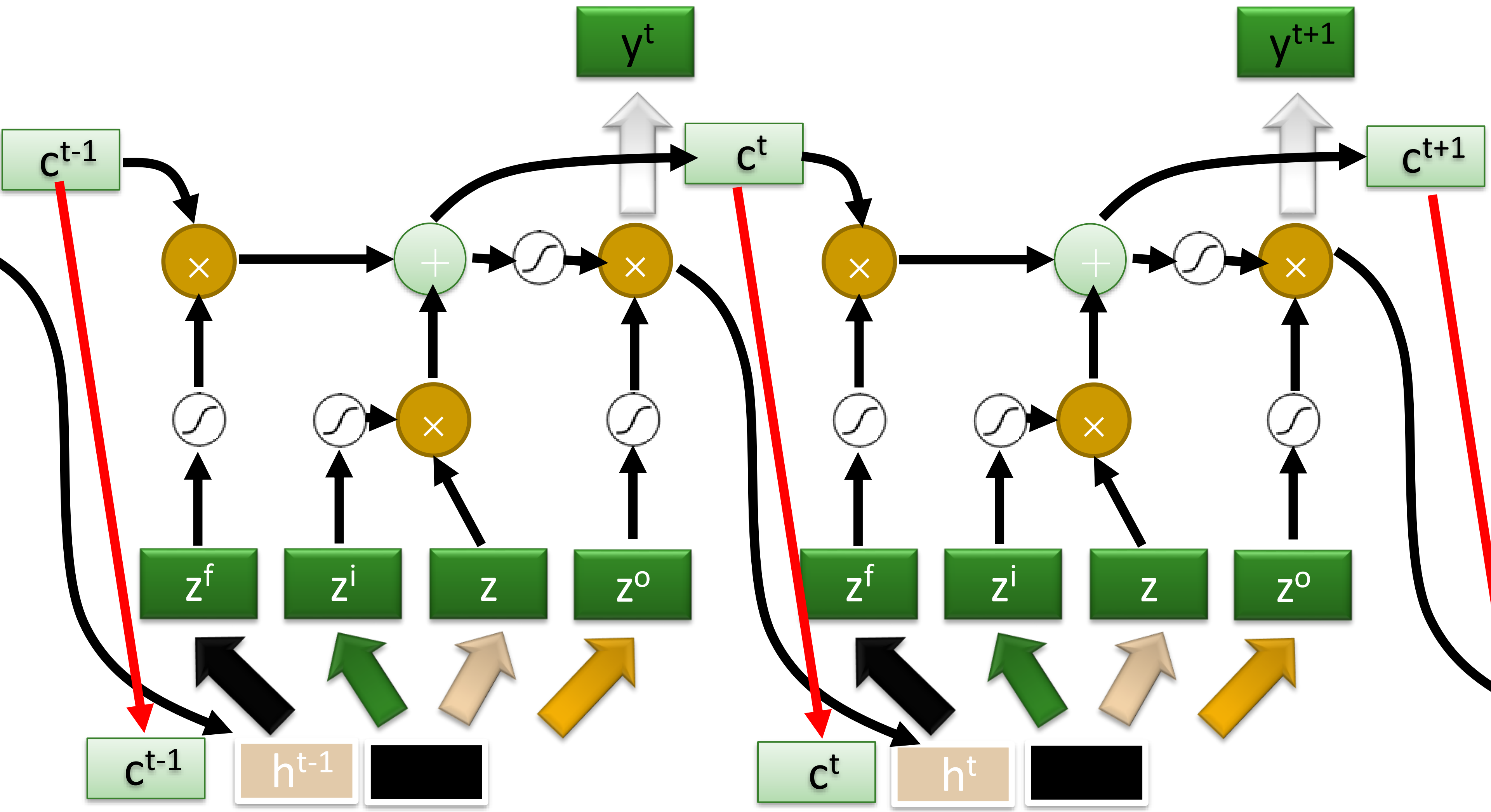
c^t-1
y^t
z^f
z^i
z
z^o



z^o
Output Gate
Forget Gate
Cell
z^f
z^i
Input Gate
Block
z

LSTM

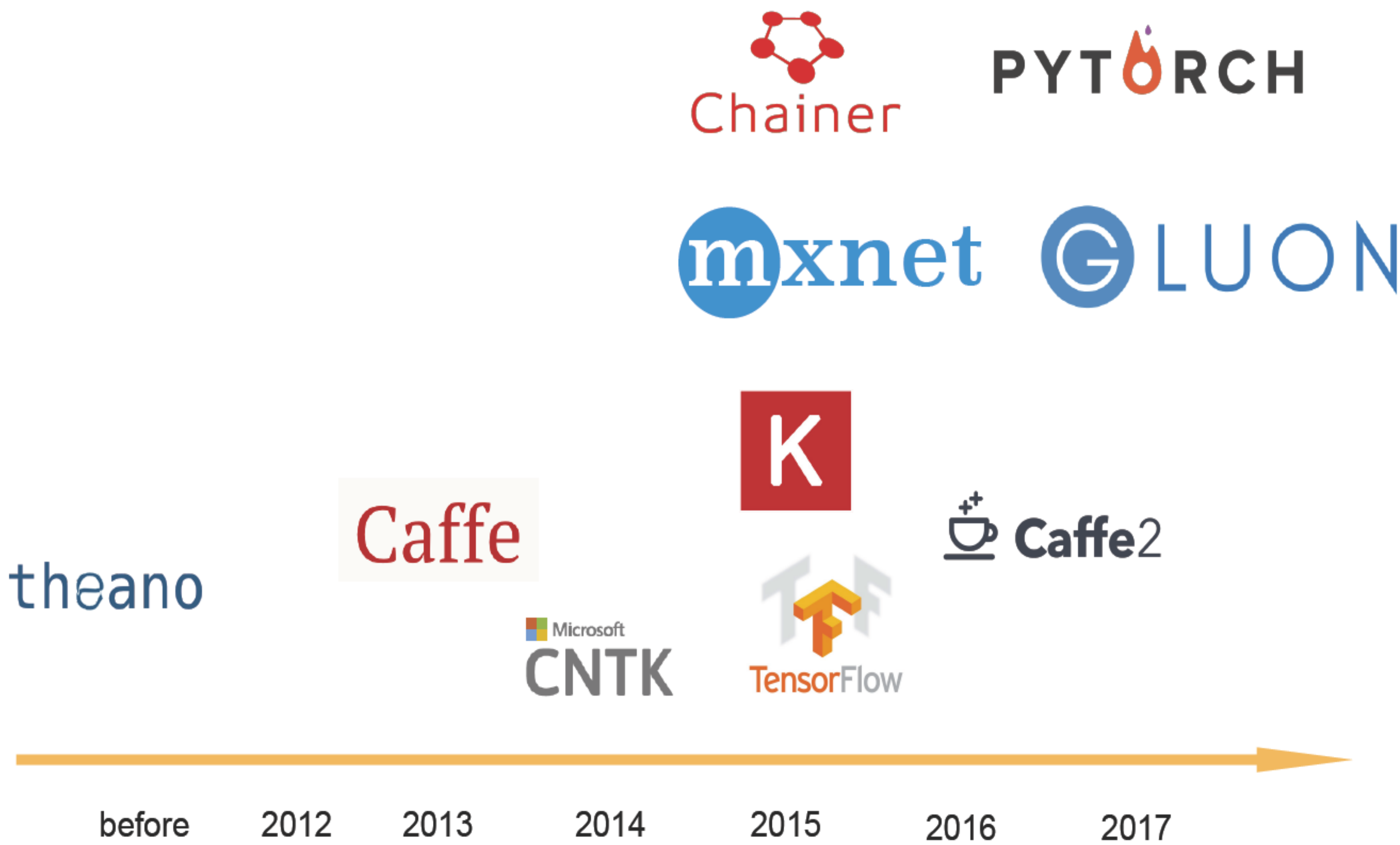
Extension: “peephole”



Outline

- Convolution, Padding & Stride
- Pooling
- Convolutional Neural Network (LeNet)
- Deep Neural Networks
- Deep Learning Frameworks

Deep Learning Frameworks



课程部分材料来自他人

和网络，仅限教学使用

，请勿传播，谢谢！