

Convolutional Neural Network

GTC 2020

Rachel Hu

Applied Scientist

Amazon AI



Outline

- Convolutions
- Pooling, Padding, Stride and Multi-layer
- Convolutional Neural Networks (LeNet)
- Deep ConvNets (AlexNet)
- Residual Neural Networks (ResNet)
- Other ideas

A large, dense crowd of people is shown from the waist up or full body. They are all wearing matching red and white horizontally striped clothing, including hats and shirts. Many individuals are also wearing black-rimmed glasses. The crowd is in motion, with many people raising their hands, clapping, or cheering. The background is slightly blurred, suggesting a busy outdoor event.

From fully connected
to convolutions

Image Classification

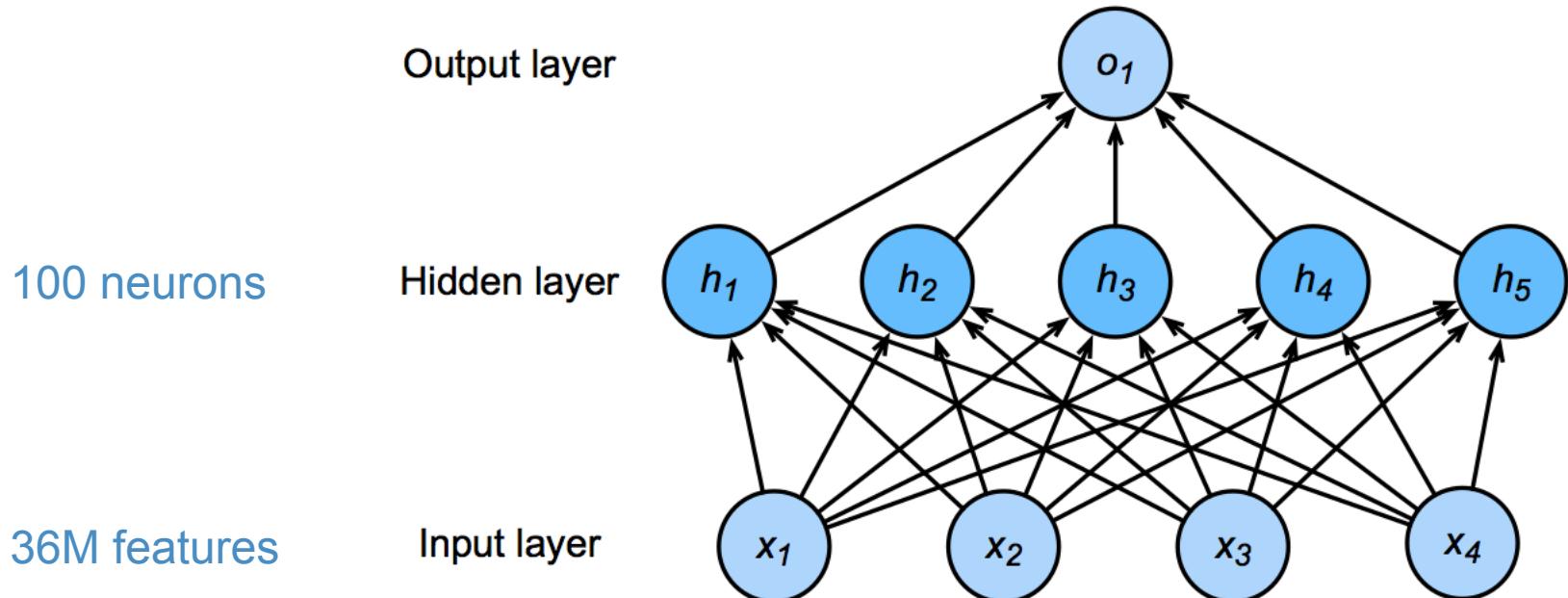
- Use a good camera
- RGB image has **36M** elements
- The model size of a single hidden layer MLP with a 100 neurons hidden size is **3.6B** parameters



Dual
12MP
wide-angle and
telephoto cameras



Flashback - Network with one hidden layer



36M features

Input layer

Output layer

Hidden layer

100 neurons

$$\mathbf{h} = \sigma(\mathbf{Wx} + \mathbf{b})$$

Flashback - Network with one hidden layer

100 neurons

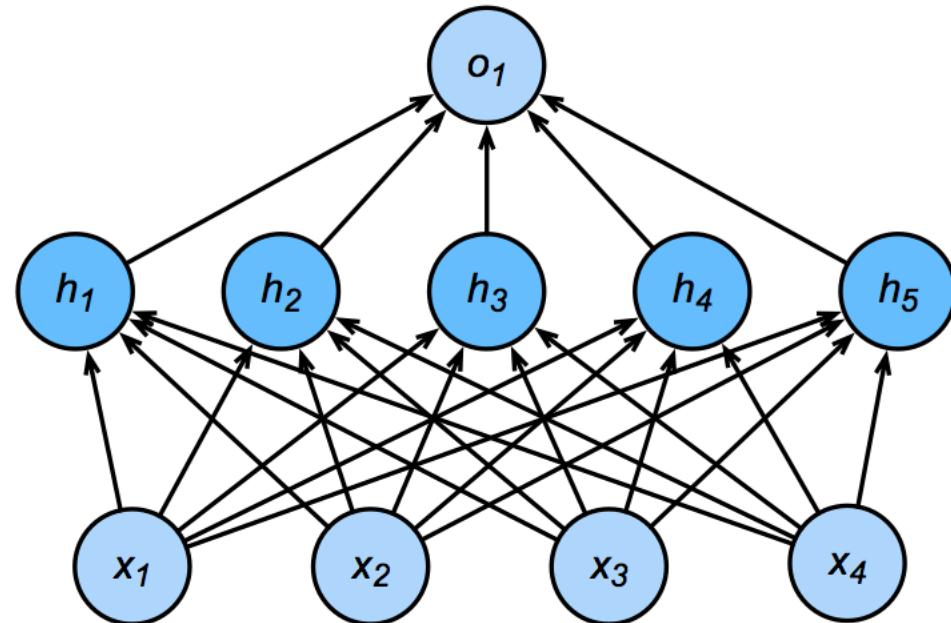
Output layer

Hidden layer

3.6B parameters = 14GB

36M features

Input layer



$$\mathbf{h} = \sigma(\mathbf{Wx} + \mathbf{b})$$

A large, dense crowd of people is shown from the waist up or full body. They are all dressed identically in red and white horizontally striped hoodies and matching red and white striped hats. Many of them are wearing black-rimmed glasses. Some individuals have their hands raised, suggesting cheering or participation in a group activity. The background is slightly blurred, emphasizing the crowd.

Can we reduce the number
of the weights a bit?

Where is
Waldo?



Two Principles



Two Principles

1. Translation Invariance:

Our vision systems
should, in some sense,
**respond similarly to the
same object regardless
of where it appears in
the image.**



Two Principles

2. Locality:

Our vision systems should, in some sense, **focus on local regions**, without regard for what else is happening in the image at greater distances.



Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w into 4-D tensors (h,w) to (h',w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l}$$

Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w into 4-D tensors (h, w) to (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l}$$



V is re-indexes W such as that

$$v_{i,j,a,b} = w_{i,j,i+a,j+b}$$

Rethinking Dense Layers

- Reshape the input, x and the hidden output, h into matrix (width, height)
- Reshape weights, w into 4-D tensors (h, w) to (h', w')

$$h_{i,j} = \sum_{k,l} w_{i,j,k,l} x_{k,l}$$
$$= \sum_{a,b} v_{i,j,a,b} x_{i+a,j+b}$$

V is re-indexes W such as that

$$v_{i,j,a,b} = w_{i,j,i+a,j+b}$$

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- A shift in x should simply lead to a shift in h

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- A shift in x should simply lead to a shift in h
- However, in the Waldo case, we did not see a lot change in the output results. Hence, v should not depend on (i, j) .

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- A shift in x should simply lead to a shift in h
- However, in the Waldo case, we did not see a lot change in the output results. Hence, v should not depend on (i, j) .

$$v_{i,j,a,b} = v_{a,b}$$

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- $v_{i,j,a,b} = v_{a,b}$

Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- $v_{i,j,a,b} = v_{a,b}$

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$



Idea #1 - Translation Invariance

$$h_{i,j} = \sum_{a,b} v_{i,j,a,b} x_{i+a, j+b}$$

- $v_{i,j,a,b} = v_{a,b}$ Reduced the dim by 36 million times smaller

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$



Idea #2 - Locality

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

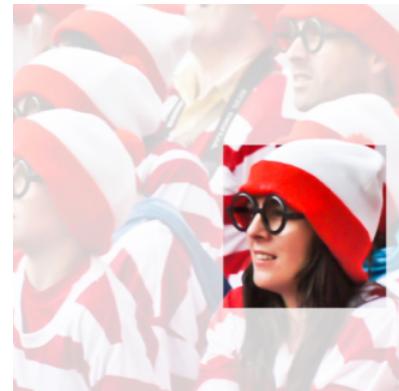
Idea #2 - Locality

$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

- We shouldn't look very far from $x(i, j)$ in order to assess what's going on at $h(i, j)$



Idea #2 - Locality

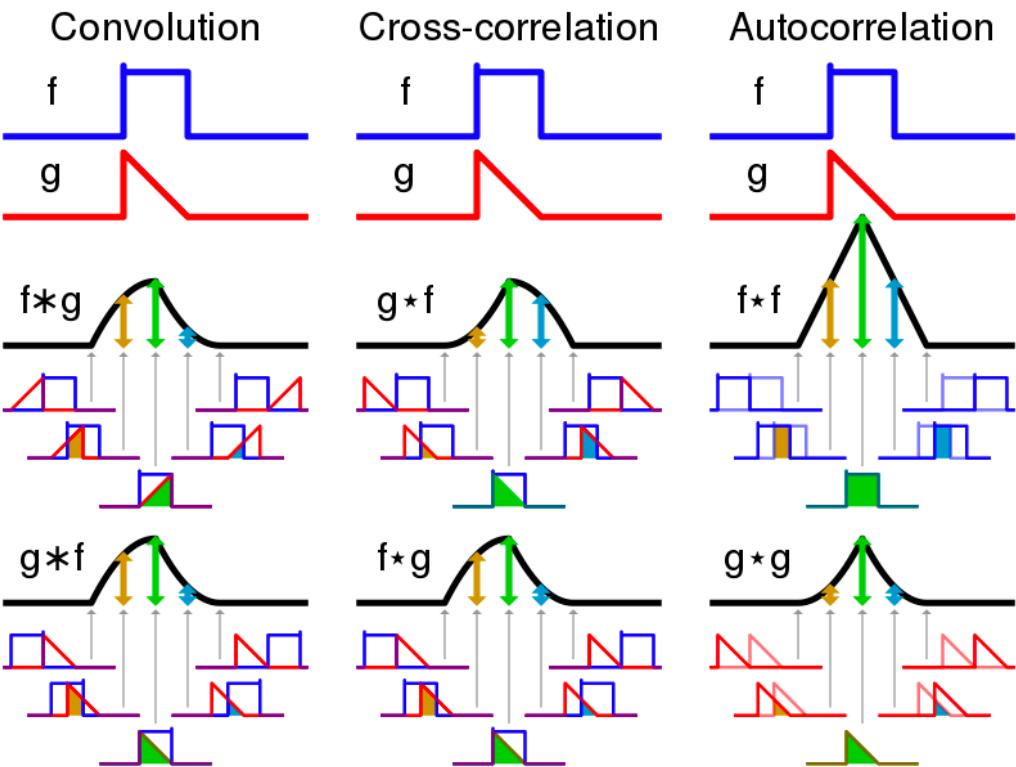


$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a, j+b}$$

- We shouldn't look very far from $x(i, j)$ in order to assess what's going on at $h(i, j)$
- Outside range $|a|, |b| > \Delta$ parameters vanish $v_{a,b} = 0$

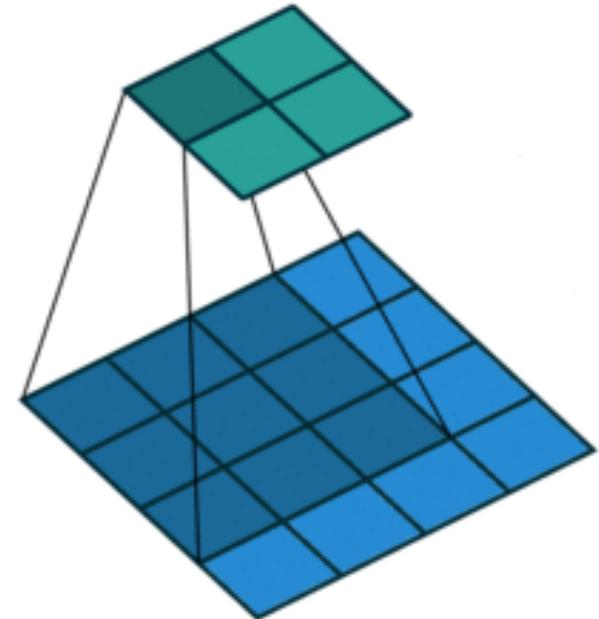
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$

Convolution



2-D Cross Correlation

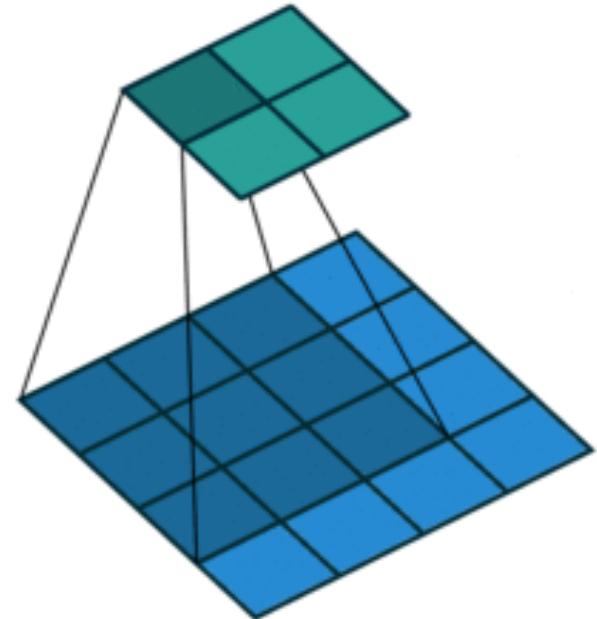
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$



(vdumoulin@ Github)

2-D Cross Correlation

$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a, j+b}$$



(vdumoulin@ Github)

2-D Cross Correlation

Input	Kernel	Output																	
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	<table border="1"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43
0	1	2																	
3	4	5																	
6	7	8																	
0	1																		
2	3																		
19	25																		
37	43																		

*

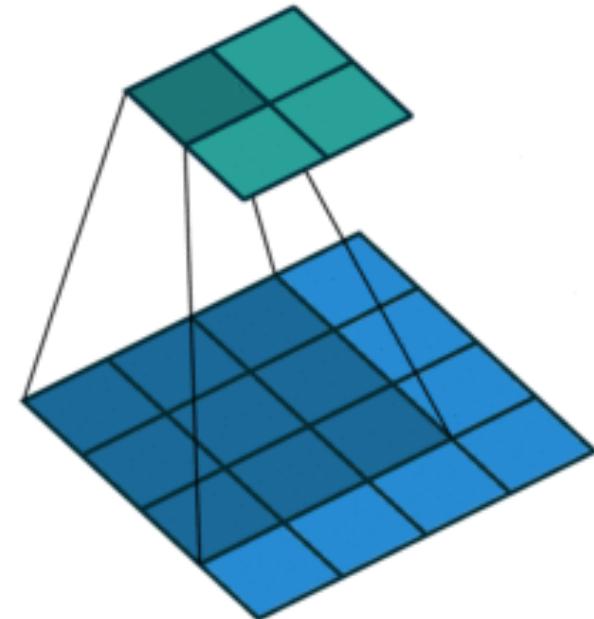
=

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)



2-D Cross Correlation

Input	Kernel	Output																	
<table border="1"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td><td>5</td></tr><tr><td>6</td><td>7</td><td>8</td></tr></table>	0	1	2	3	4	5	6	7	8	<table border="1"><tr><td>0</td><td>1</td></tr><tr><td>2</td><td>3</td></tr></table>	0	1	2	3	<table border="1"><tr><td>19</td><td>25</td></tr><tr><td>37</td><td>43</td></tr></table>	19	25	37	43
0	1	2																	
3	4	5																	
6	7	8																	
0	1																		
2	3																		
19	25																		
37	43																		

*

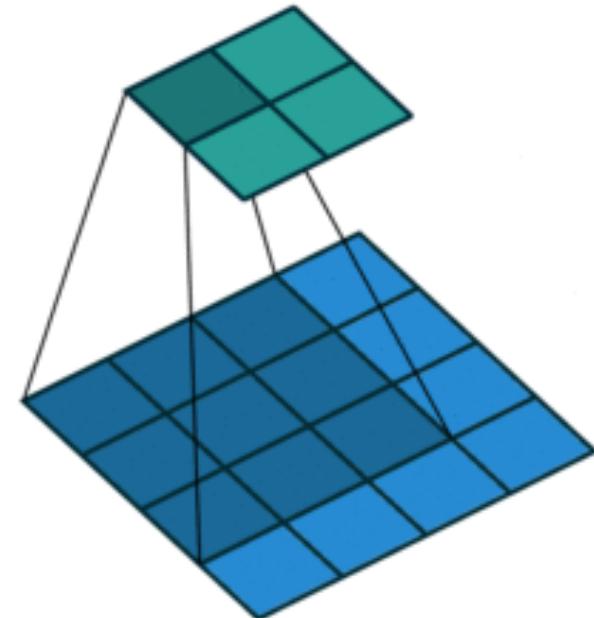
=

$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

$$1 \times 0 + 2 \times 1 + 4 \times 2 + 5 \times 3 = 25,$$

$$3 \times 0 + 4 \times 1 + 6 \times 2 + 7 \times 3 = 37,$$

$$4 \times 0 + 5 \times 1 + 7 \times 2 + 8 \times 3 = 43.$$



(vdumoulin@ Github)

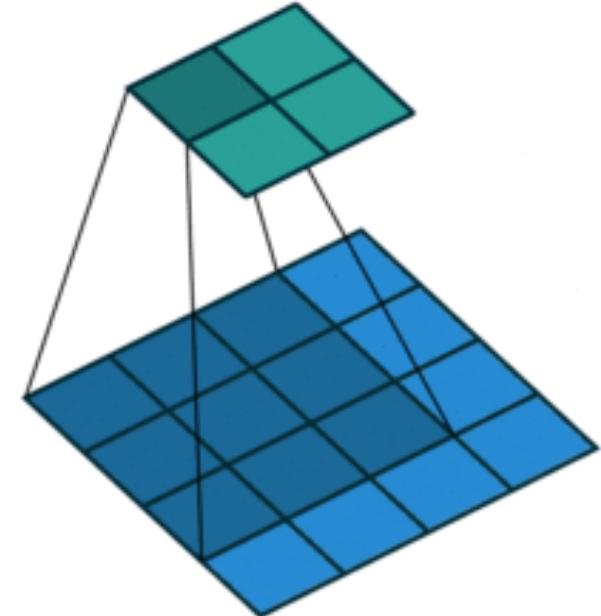


2-D Convolution Layer

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b : scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output r

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



(vdumoulin@ Github)

- \mathbf{W} and b are learnable parameters

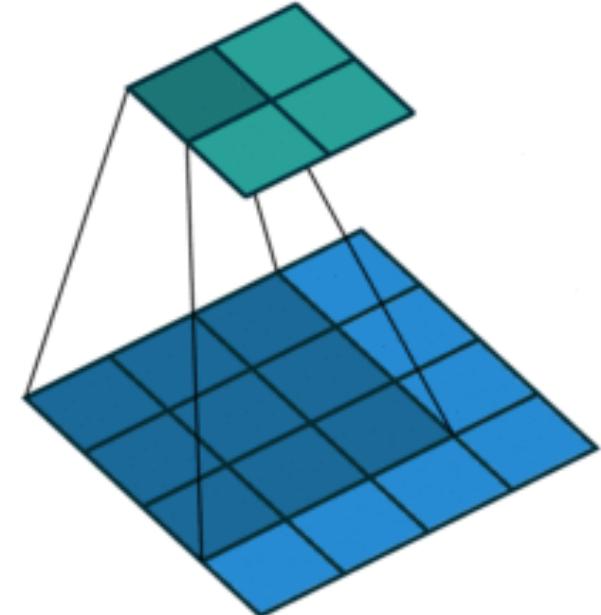


2-D Convolution Layer

$$\begin{array}{|c|c|c|} \hline 0 & 1 & 2 \\ \hline 3 & 4 & 5 \\ \hline 6 & 7 & 8 \\ \hline \end{array} * \begin{array}{|c|c|} \hline 0 & 1 \\ \hline 2 & 3 \\ \hline \end{array} = \begin{array}{|c|c|} \hline 19 & 25 \\ \hline 37 & 43 \\ \hline \end{array}$$

- $\mathbf{X} : n_h \times n_w$ input matrix
- $\mathbf{W} : k_h \times k_w$ kernel matrix
- b : scalar bias
- $\mathbf{Y} : (n_h - k_h + 1) \times (n_w - k_w + 1)$ output r

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



(vdumoulin@ Github)

- \mathbf{W} and b are learnable parameters



Examples

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



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

(wikipedia)

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

Examples



(wikipedia)

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$



Edge Detection

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



Sharpen

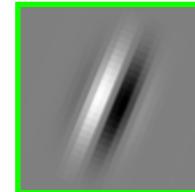
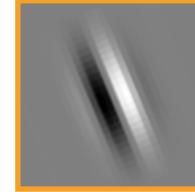
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$



Gaussian Blur

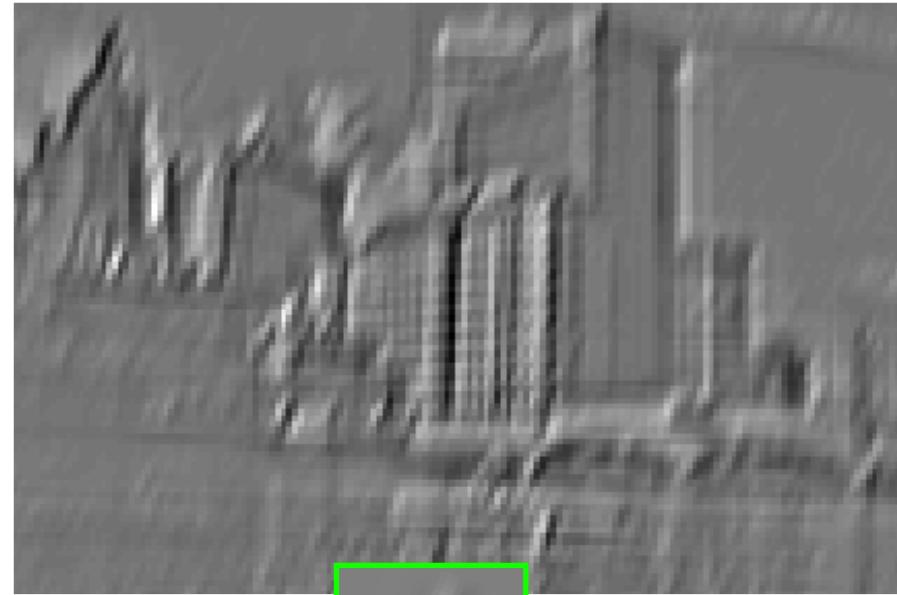


Examples



(Rob Fergus)

Examples



(Rob Fergus)

aws

Convolutions Notebook

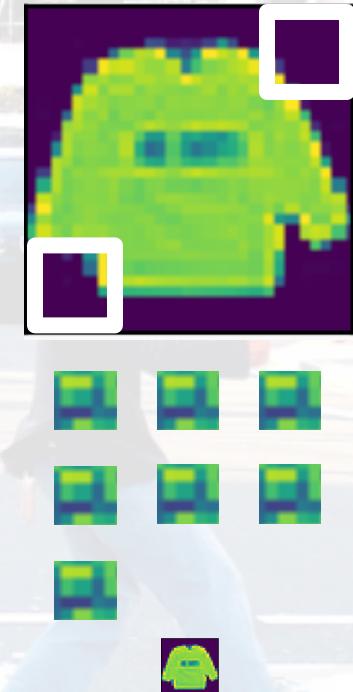


A composite image showing a man in a dark jacket and blue jeans crossing a city street four times in sequence. He is in mid-stride, with his arms outstretched, in each frame. The background shows a typical urban street with parked cars, buildings, and other people. The overall effect is a dynamic, repetitive movement.

Padding and Stride

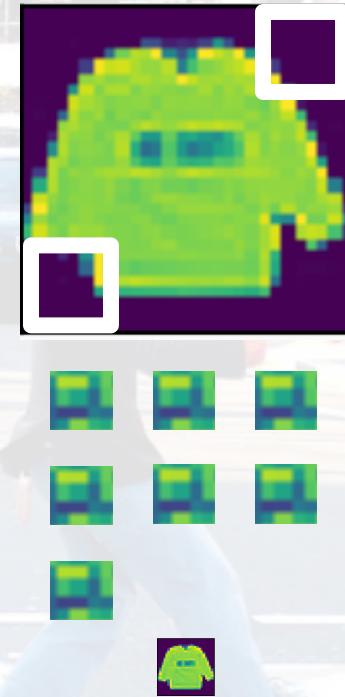
The Issue

- Given a 32×32 input image
- Apply convolutional layer with 5×5 kernel
 - 28×28 output with 1 layer
 - 4×4 output with 7 layers



The Issue

- Given a 32×32 input image
- Apply convolutional layer with 5×5 kernel
 - 28×28 output with 1 layer
 - 4×4 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

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

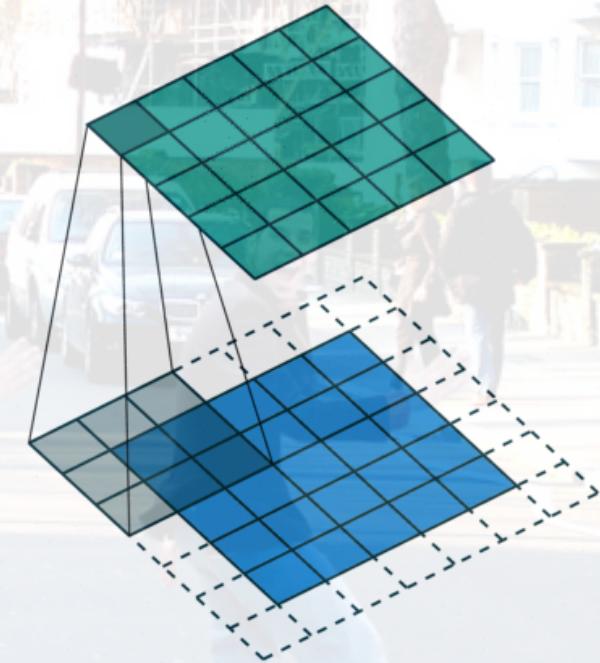
*

Kernel	
0	1
2	3

=

Output			
0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

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



Padding

Padding adds rows/columns around input

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

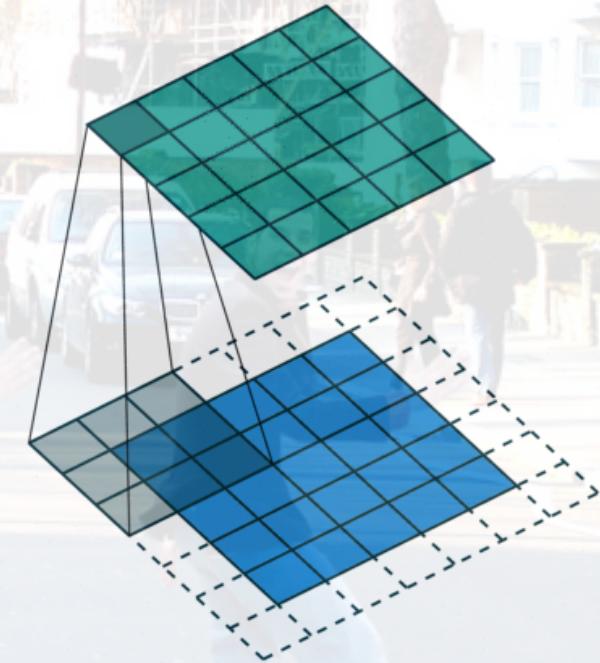
*

Kernel	
0	1
2	3

=

Output			
0	3	8	4
9	19	25	10
21	37	43	16
6	7	8	0

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

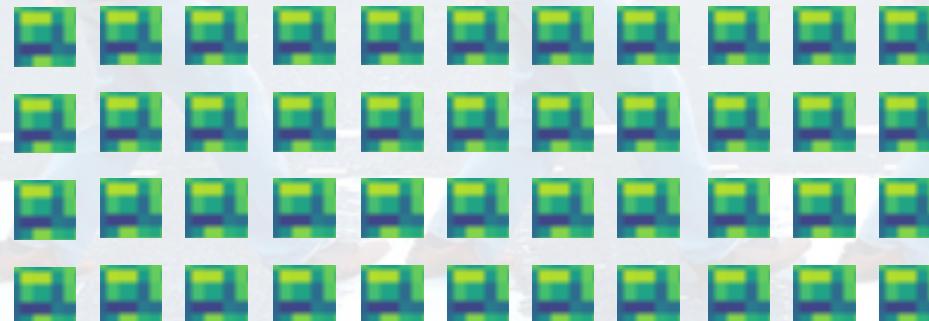


Padding

- 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

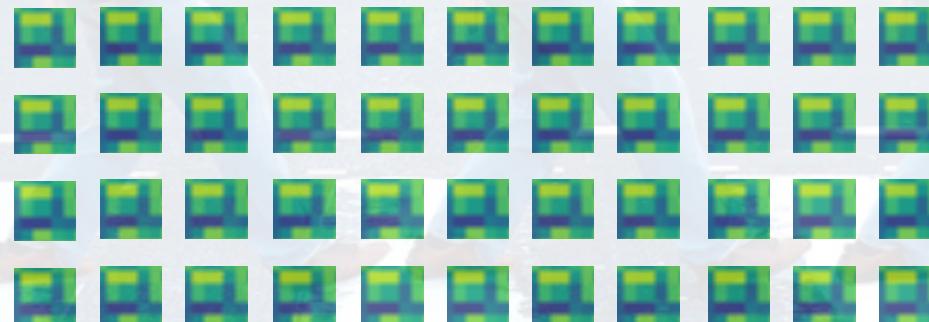
The Issue

Too SLOW !



The Issue

- Padding reduces shape **linearly** with #layers
 - Given a 224×224 input with a 5×5 kernel, needs 44 layers to reduce the shape to 4×4
 - Requires a large amount of computation



Stride

- Stride is the #rows/#columns per slide

Strides of 3 for height and 2 for width

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

*

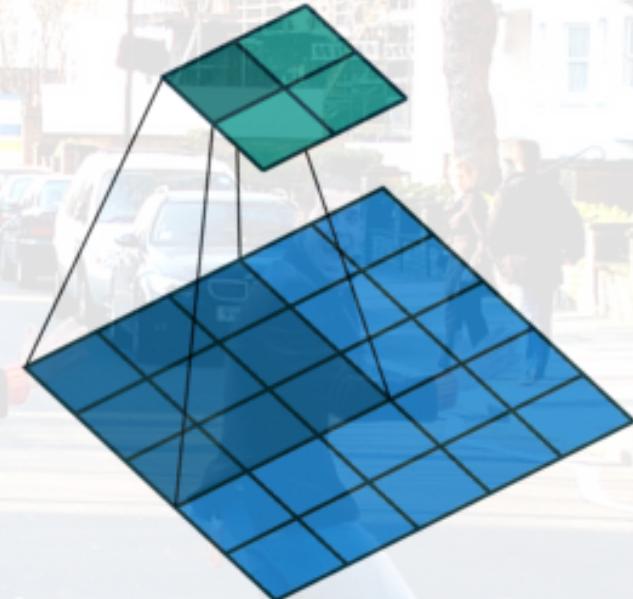
0	1
2	3

 =

0	8
6	8

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

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



Stride

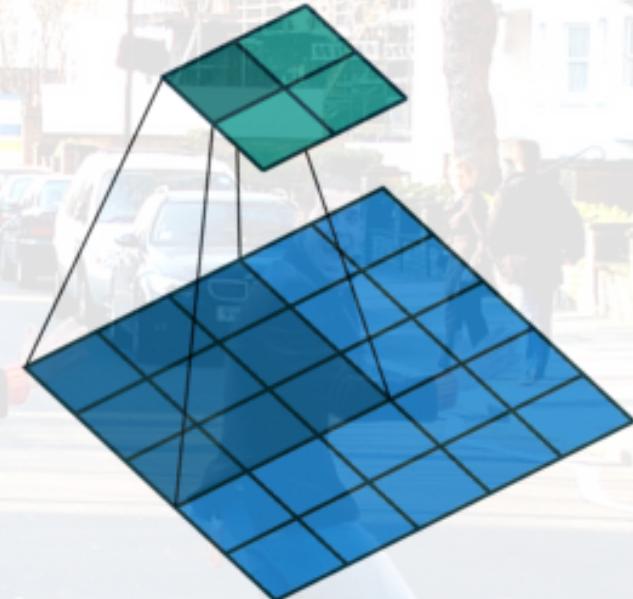
- Stride is the #rows/#columns per slide

Strides of 3 for height and 2 for width

Input					Kernel		Output			
0	0	0	0	0	*	0	1	=	0	8
0	0	1	2	0		2	3		6	8
0	3	4	5	0						
0	6	7	8	0						
0	0	0	0	0						

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

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



Stride

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

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

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

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

An aerial photograph showing a complex network of water channels. The channels are narrow and deep, filled with dark blue water. They are separated by thick, green, vegetated banks. The pattern of channels creates a series of parallel lines that converge towards the top left of the frame, illustrating a branching or deltaic system.

*Multiple Input and
Output Channels*

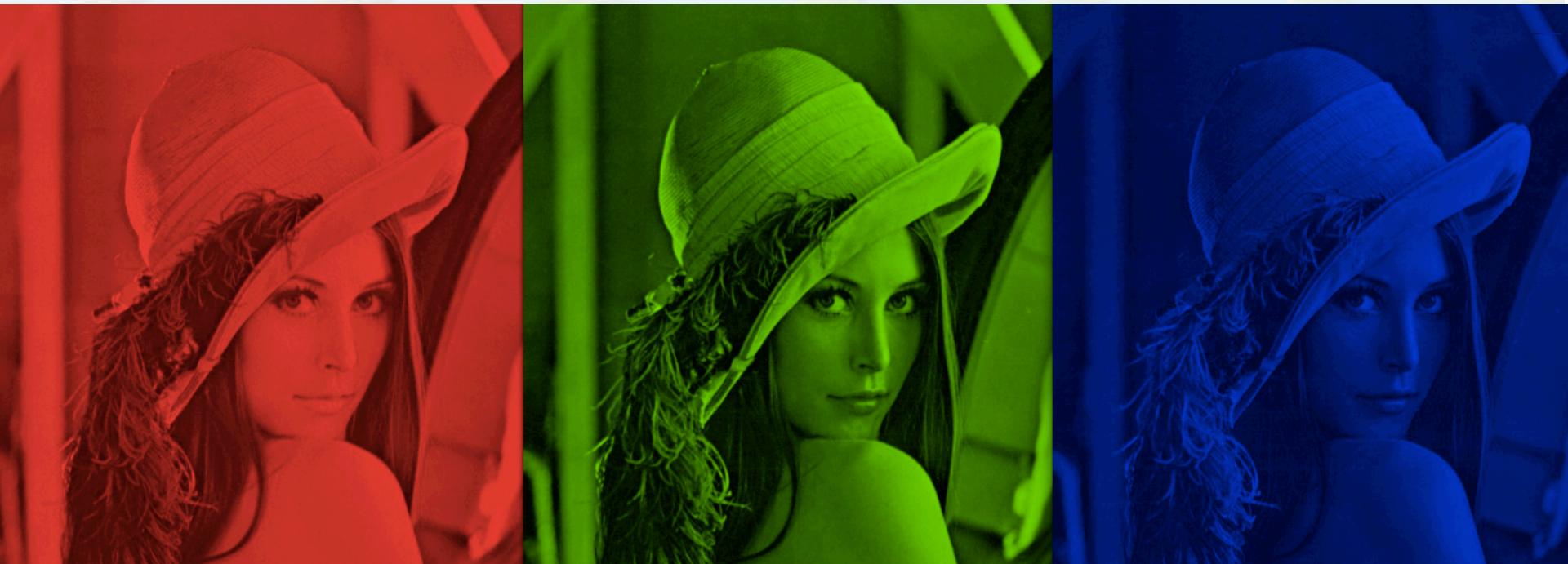
Multiple Input Channels

A lot of really interesting and challenging details back in 1970s...



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

Input	Kernel	Input	Kernel	Output																																
<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>0</td><td>1</td><td>2</td></tr><tr><td>2</td><td>3</td><td></td></tr></table>	0	1	2	2	3		<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>1</td><td>2</td><td>3</td></tr><tr><td>4</td><td>5</td><td>6</td></tr><tr><td>7</td><td>8</td><td>9</td></tr></table>	1	2	3	4	5	6	7	8	9	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>1</td><td>2</td></tr><tr><td>3</td><td>4</td></tr></table>	1	2	3	4	<table border="1" style="border-collapse: collapse; width: 100%;"><tr><td>56</td><td>72</td></tr><tr><td>104</td><td>120</td></tr></table>	56	72	104	120
1	2	3																																		
4	5	6																																		
7	8	9																																		
0	1	2																																		
2	3																																			
1	2	3																																		
4	5	6																																		
7	8	9																																		
1	2																																			
3	4																																			
56	72																																			
104	120																																			
*		+																																		

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

Multiple Input Channels

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

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

Multiple Input Channels

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

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

So far we always get single output channel, no matter how many inputs channels

Multiple Input & Output Channels

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

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

- $\mathbf{X} : c_i \times n_h \times n_w$ input
- $\mathbf{W} : c_o \times c_i \times k_h \times k_w$ kernel
- $\mathbf{Y} : c_o \times m_h \times m_w$ output

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

for $j = 1, \dots, c_o$



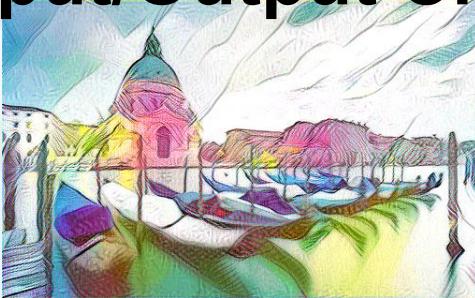
Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



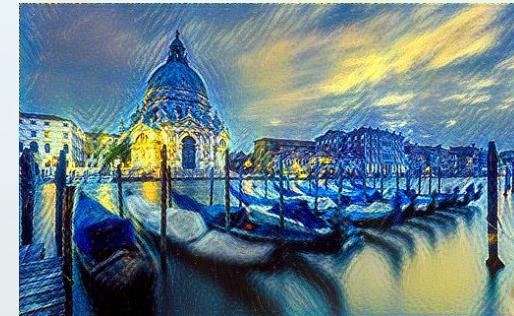
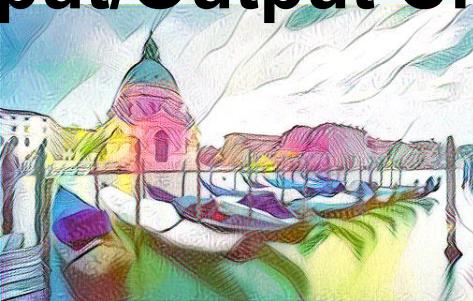
Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



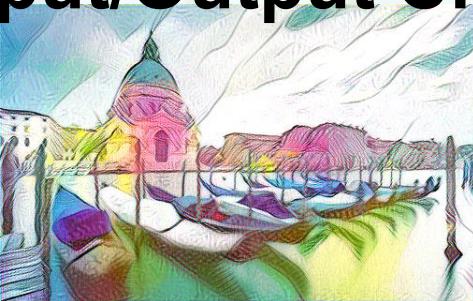
Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



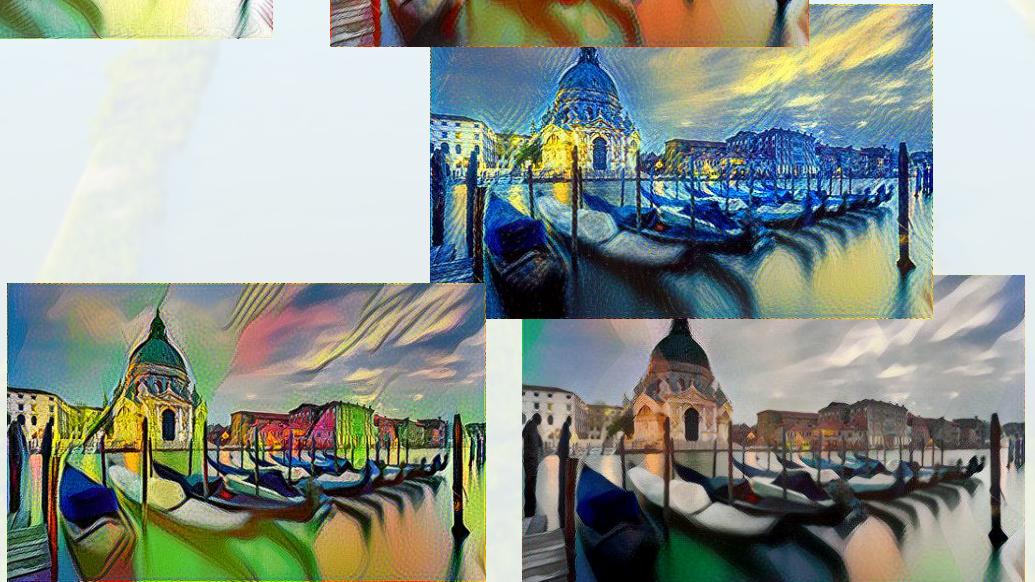
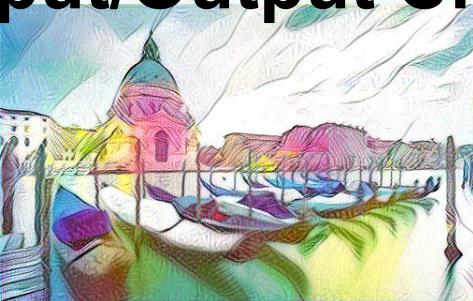
Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



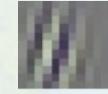
Why Multiple Input/Output Channels?



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/>



Why Multiple Input/Output Channels?



Why Multiple Input/Output Channels?

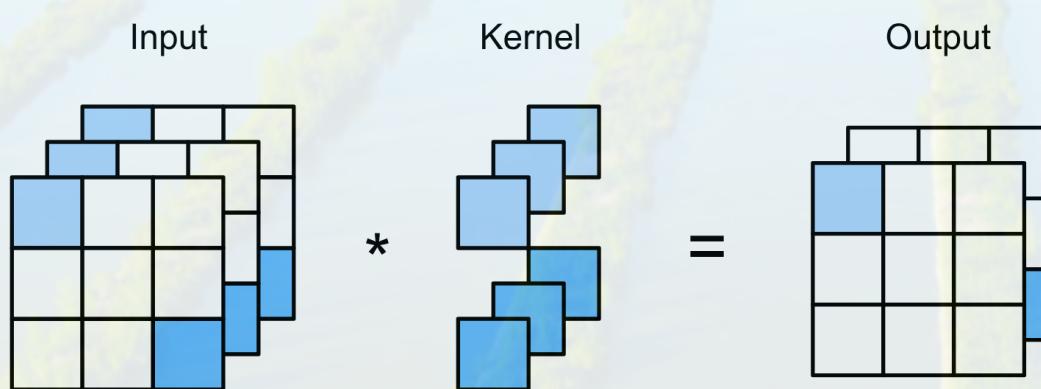
- Each input channel may recognize a particular pattern in inputs



- Each output channel recognize and combines patterns to different outputs

1 x 1 Convolutional Layer

$k_h = k_w = 1$ is a popular choice. It doesn't recognize spatial patterns, but fuse channel dimensions.

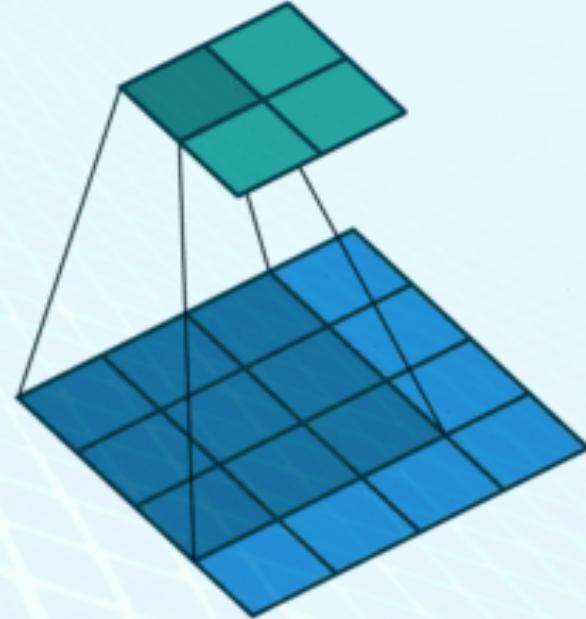


Equal to a dense layer with $n_h n_w \times c_i$ input and $c_o \times c_i$ weight.

Pooling Layer

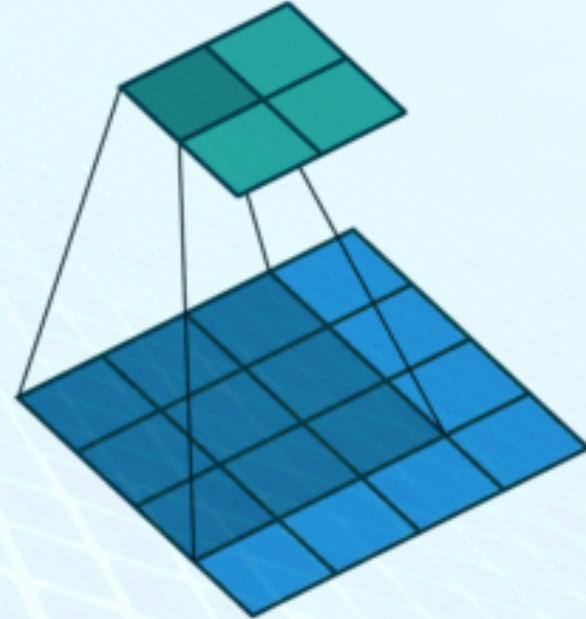
2-D Max Pooling

Like convolutional layers, pooling layers consist of a fixed-shape window that slides over all regions in the input according to its stride.



2-D Max Pooling

Like convolutional layers, pooling layers consist of a fixed-shape window that slides over all regions in the input according to its stride.

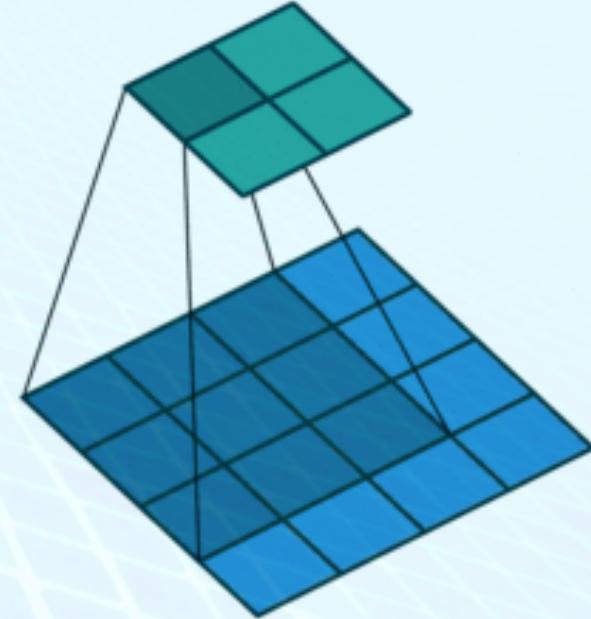


2-D Max Pooling

- Returns the maximal value in the pooling window



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

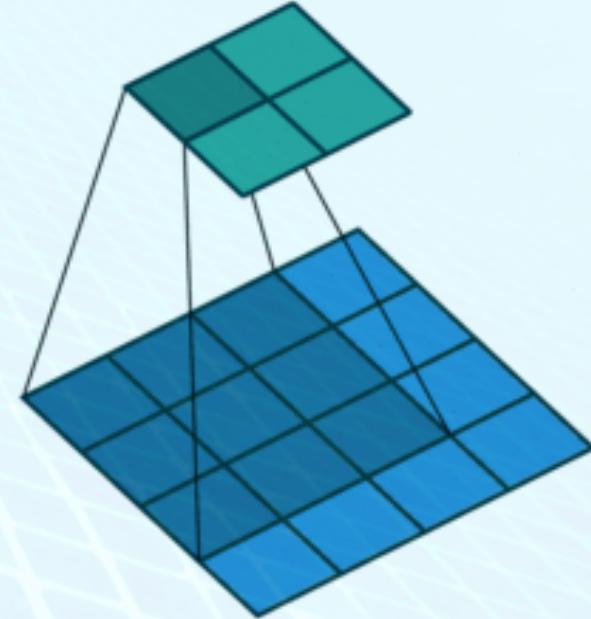


2-D Max Pooling

- Returns the maximal value in the pooling window

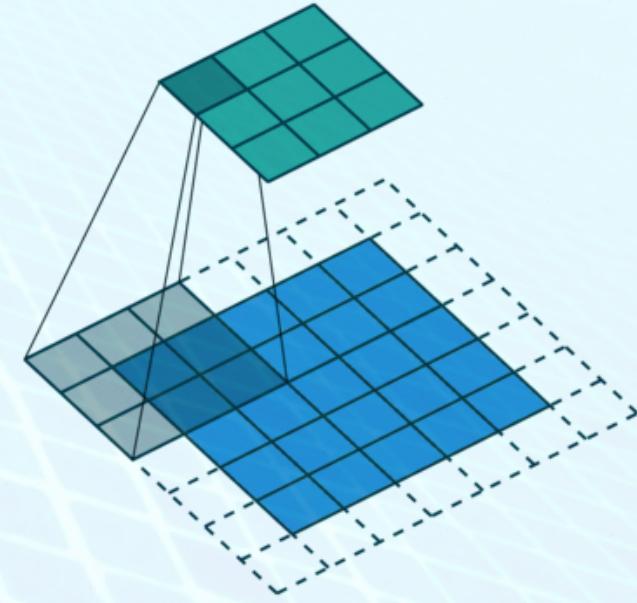


$$\max(0,1,3,4) = 4$$



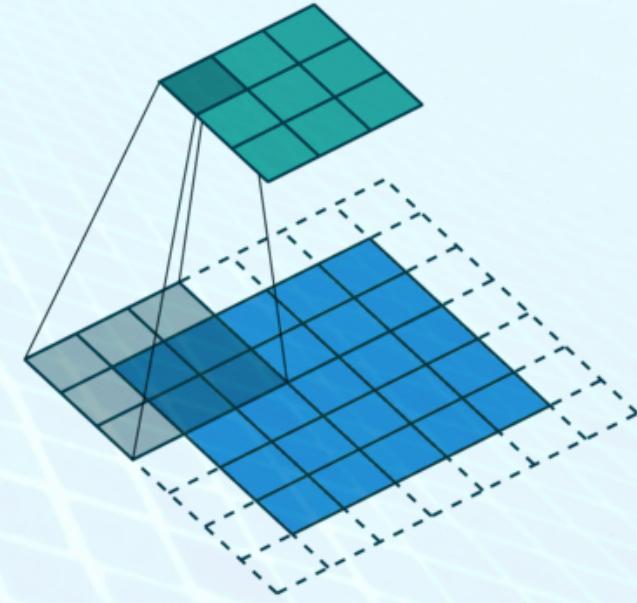
Padding, Stride, and Multiple Channels

- No learnable parameters
- Pooling layers can apply similar padding and stride as convolutional layers
- Apply pooling for each input channel to obtain the corresponding output channel
#output channels = #input channels



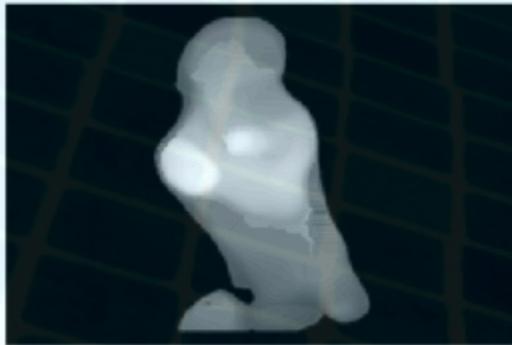
Padding, Stride, and Multiple Channels

- No learnable parameters
- Pooling layers can apply similar padding and stride as convolutional layers
- Apply pooling for each input channel to obtain the corresponding output channel
#output channels = #input channels



Average Pooling

Max pooling



Average pooling



Average Pooling

Max pooling



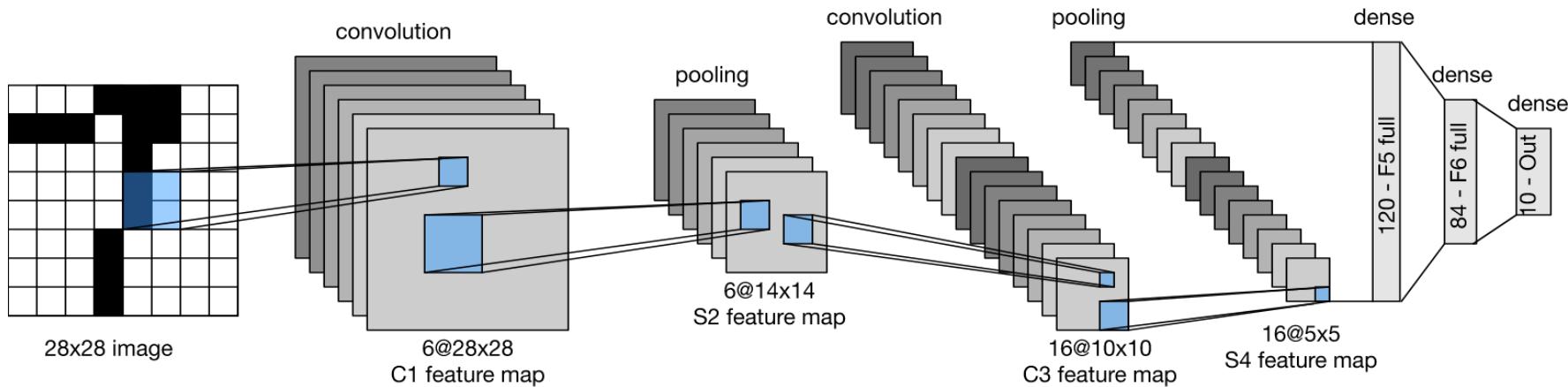
Average pooling



- 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

Pooling Notebook

LeNet

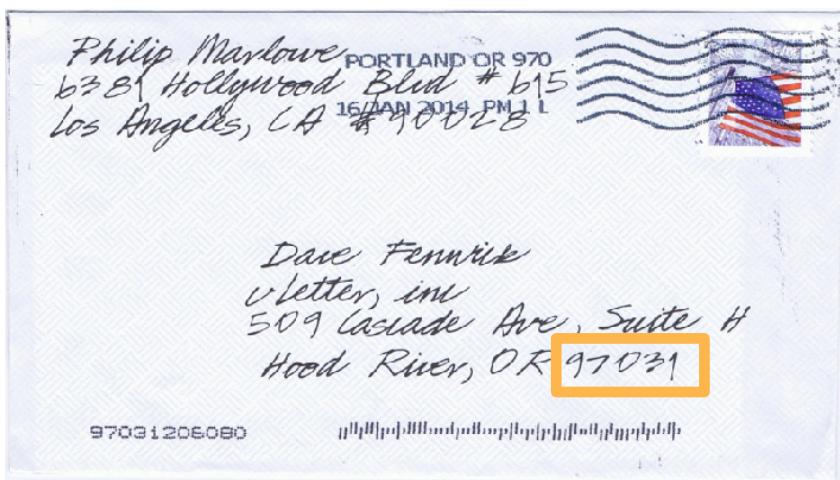


<http://yann.lecun.com/exdb/lenet/>



Why Handwritten Digit Recognition in 1990s?

AT&T had a project in order to recognize handwritten characters for postal codes on letters and also recognize the dollar amounts on checks.



MNIST

- 50,000 training data
- 10,000 test data
- 10 classes
- Grayscale
- 28 x 28 pixels
- Centered and scaled





AT&T *LeNet 5* RESEARCH

answer: 0

0
103

A 28x28 pixel grayscale image of a handwritten digit '0'. The digit is dark gray and has a slightly irregular shape. It is centered on a white background with a subtle dotted grid pattern.

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



The image displays the LeNet-5 research results visualization. It features the AT&T logo at the top left, followed by the text "LeNet 5" in red and "RESEARCH" in blue. Below this, the word "answer:" is displayed in red, followed by the number "0". The main area shows a digit recognition interface with a large digit "0" above the number "103". At the bottom, there is a small image of a handwritten digit "0" on a light gray background.

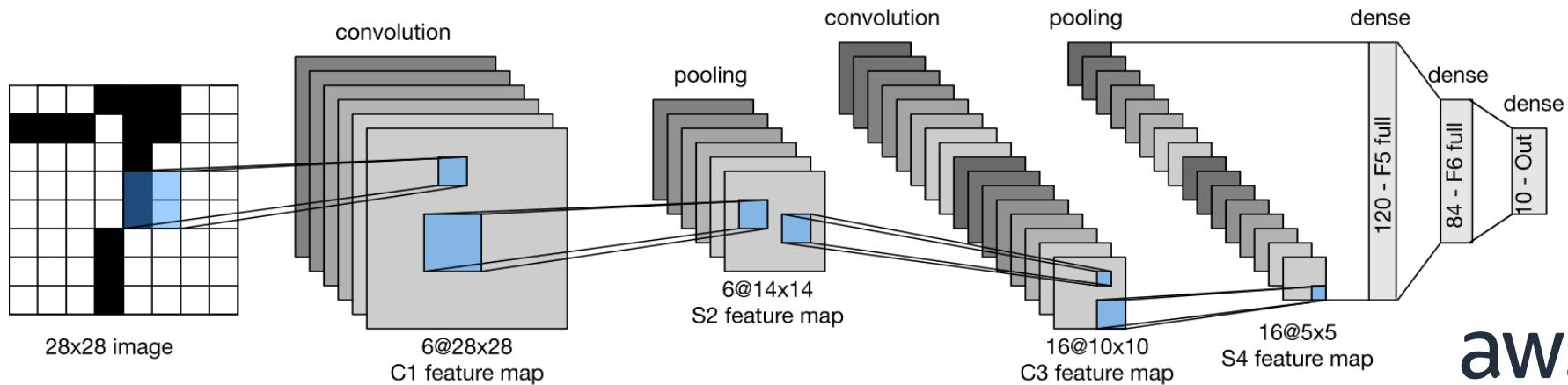
Y. LeCun, L.
Bottou, Y. Bengio,
P. Haffner, 1998
Gradient-based
learning applied to
document
recognition

LeNet

LeNet consists of two parts:

Part I. Convolution Block

- Convolution layer - To recognize the spatial patterns
 - 5×5 kernel
 - sigmoid activation function
- Average pooling layer - To reduce the dimensionality

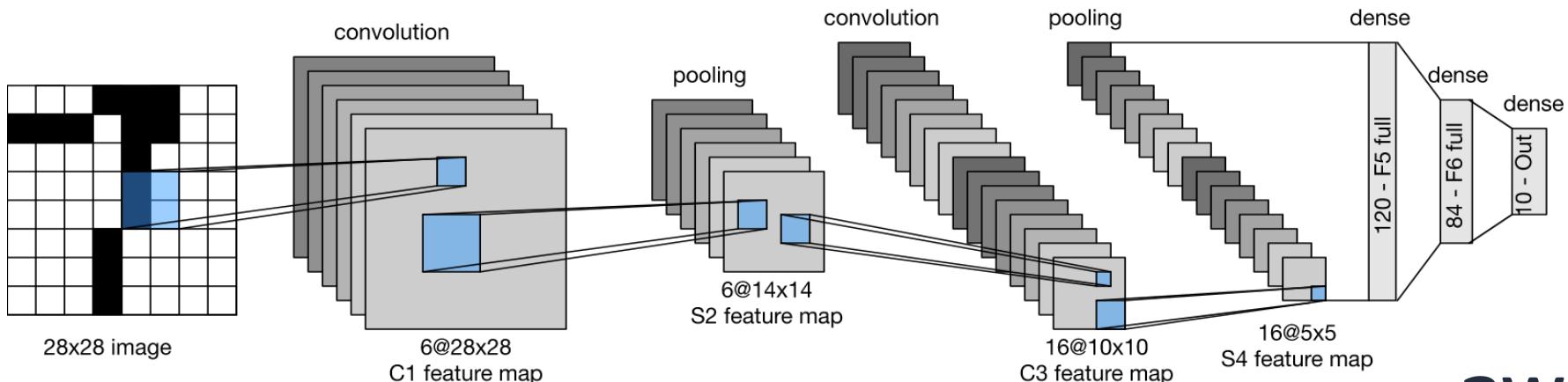


LeNet

LeNet consists of two parts:

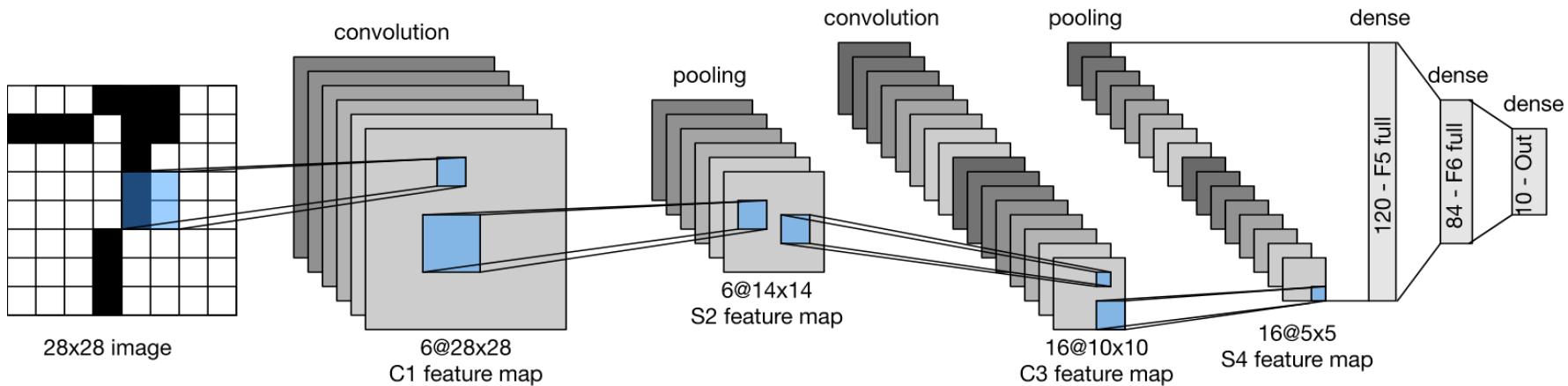
Part II. Fully-connected layers Block

- 3 fully-connected layers
 - with 120, 84, and 10 outputs, respectively



LeNet

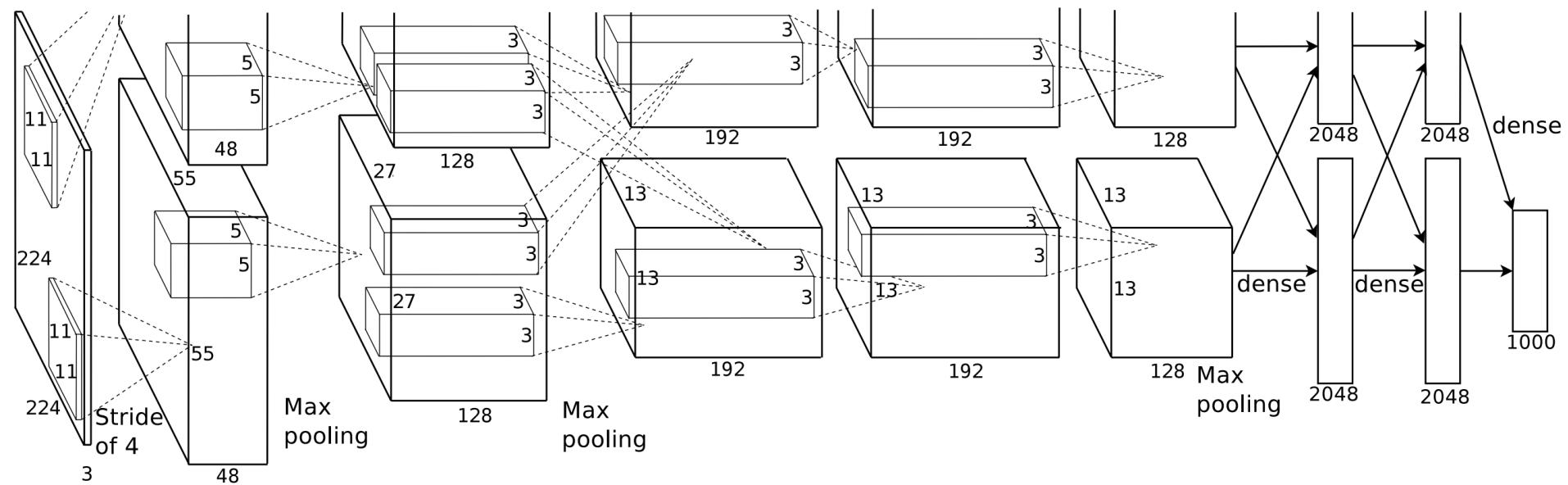
Expensive if we have many outputs



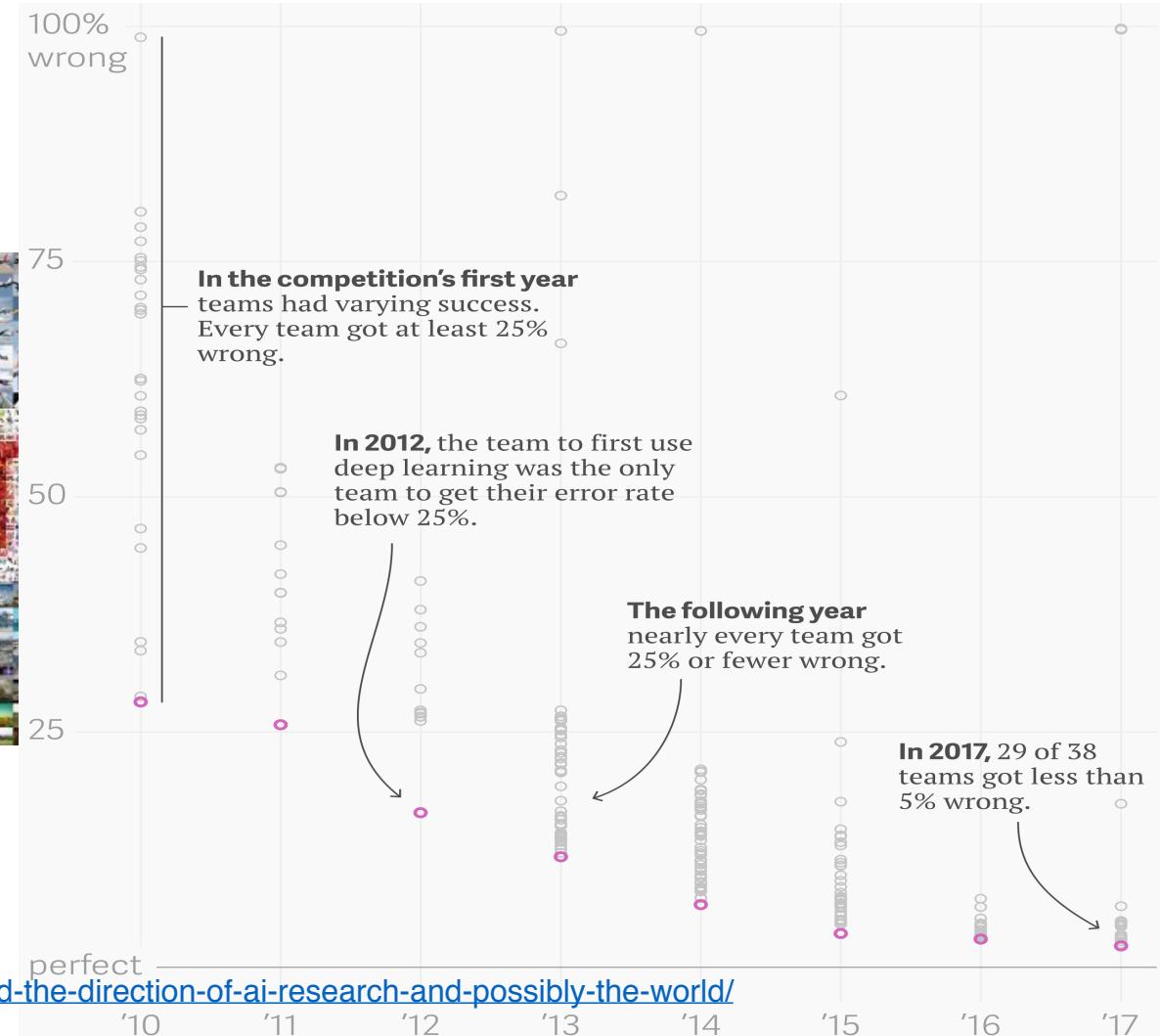
LeNet Notebook



AlexNet



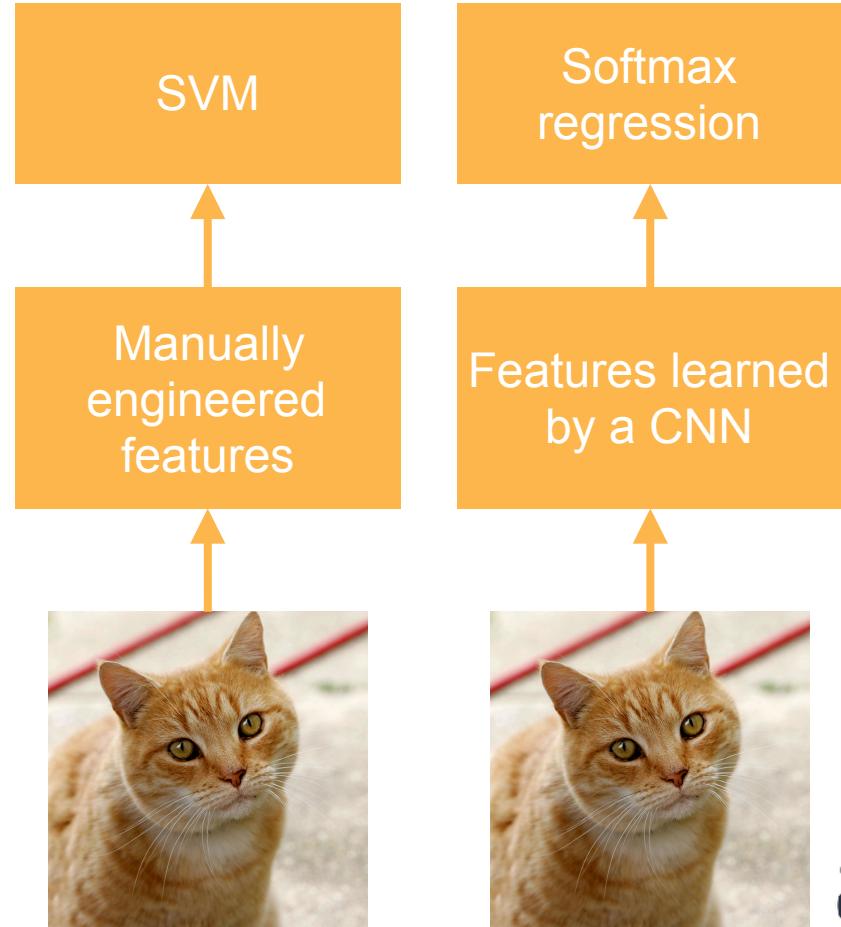
AlexNet



Yanofsky, Quartz

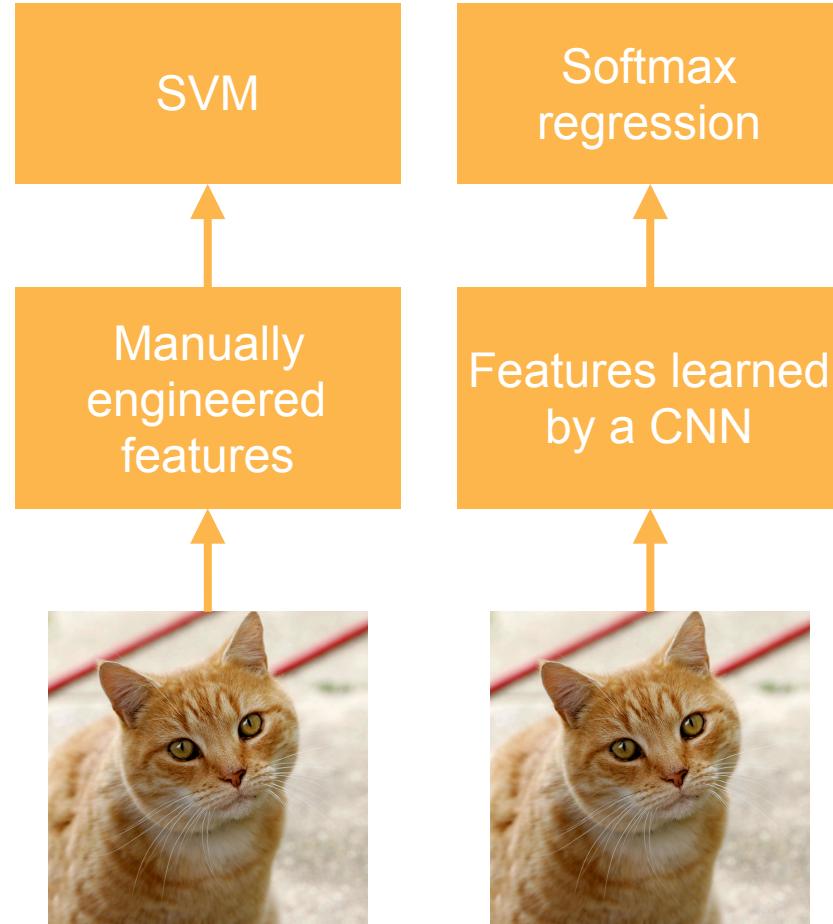
<https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>

AlexNet

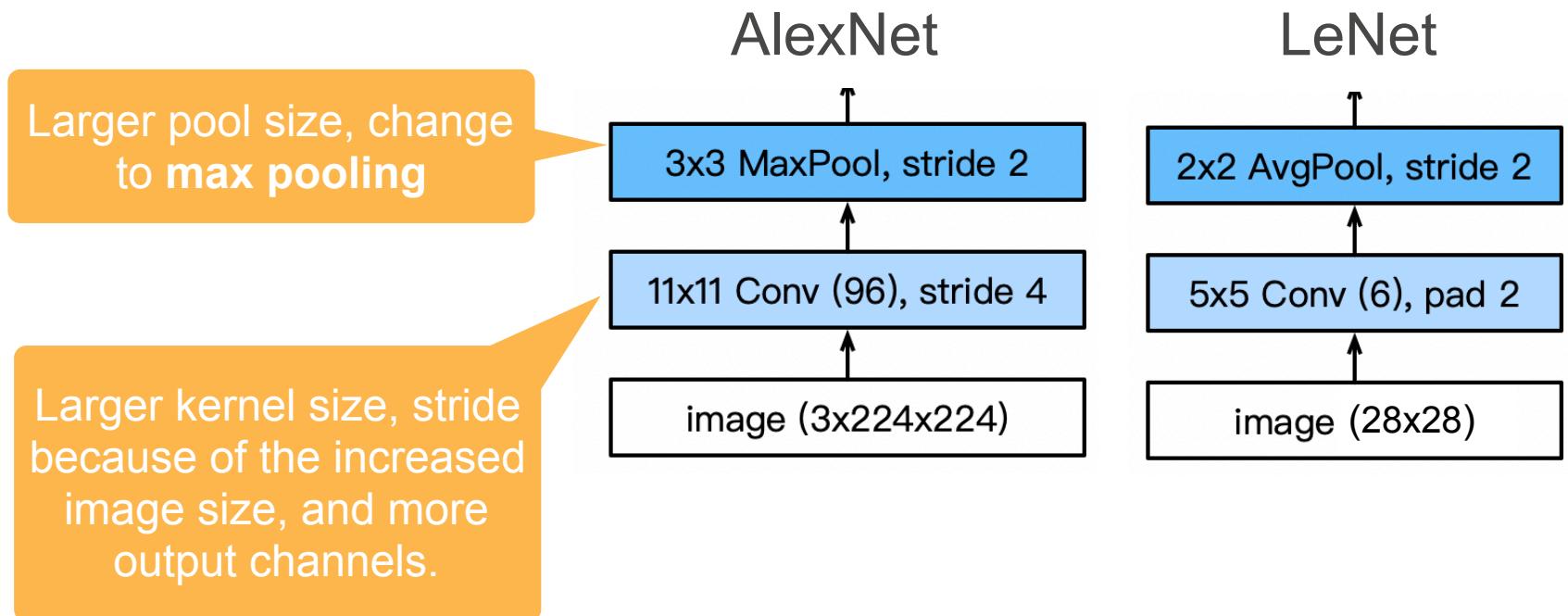


AlexNet

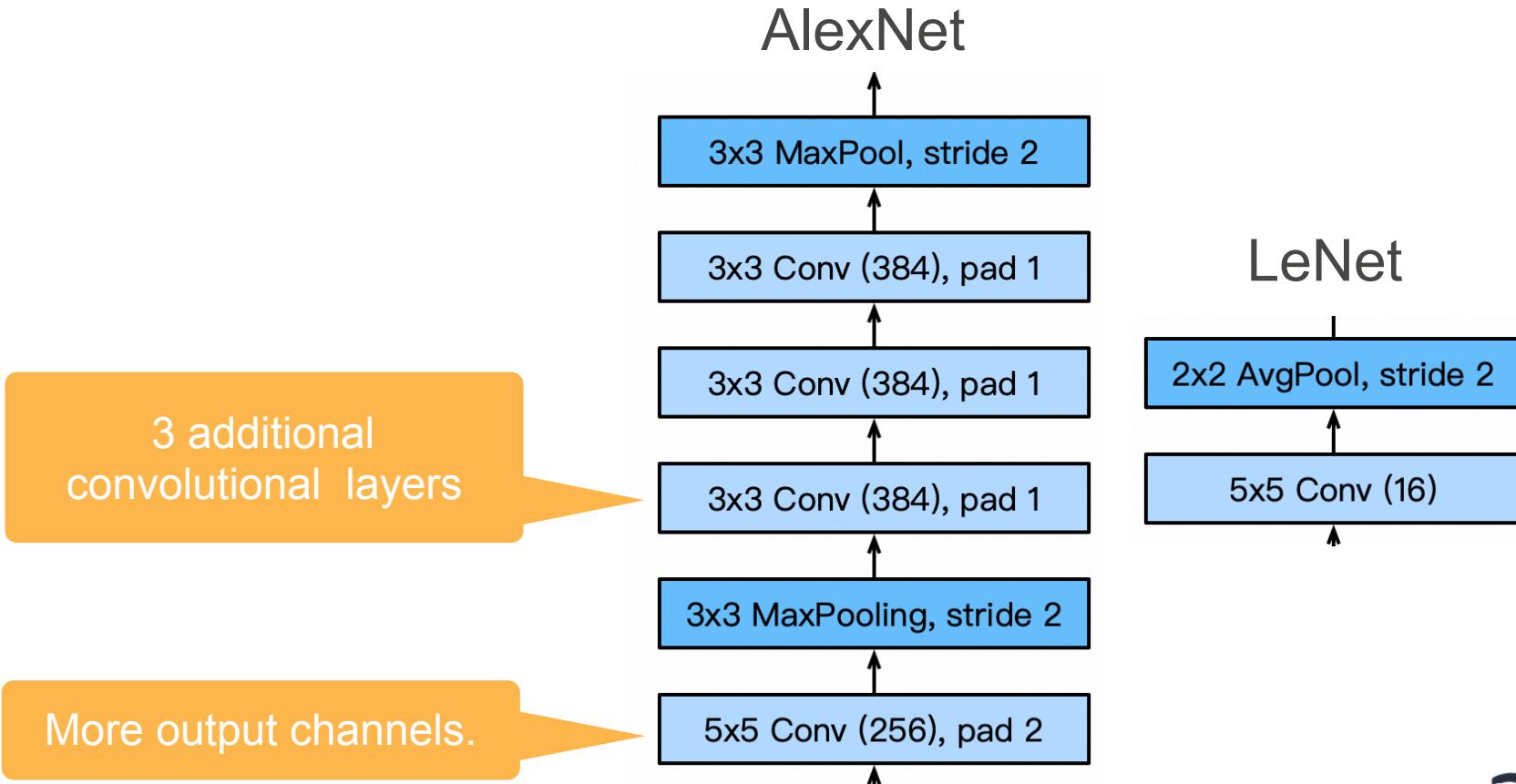
- Deeper and bigger LeNet
- Key modifications
 - Dropout (regularization)
 - ReLu (training)
 - MaxPooling
- Paradigm shift for computer vision



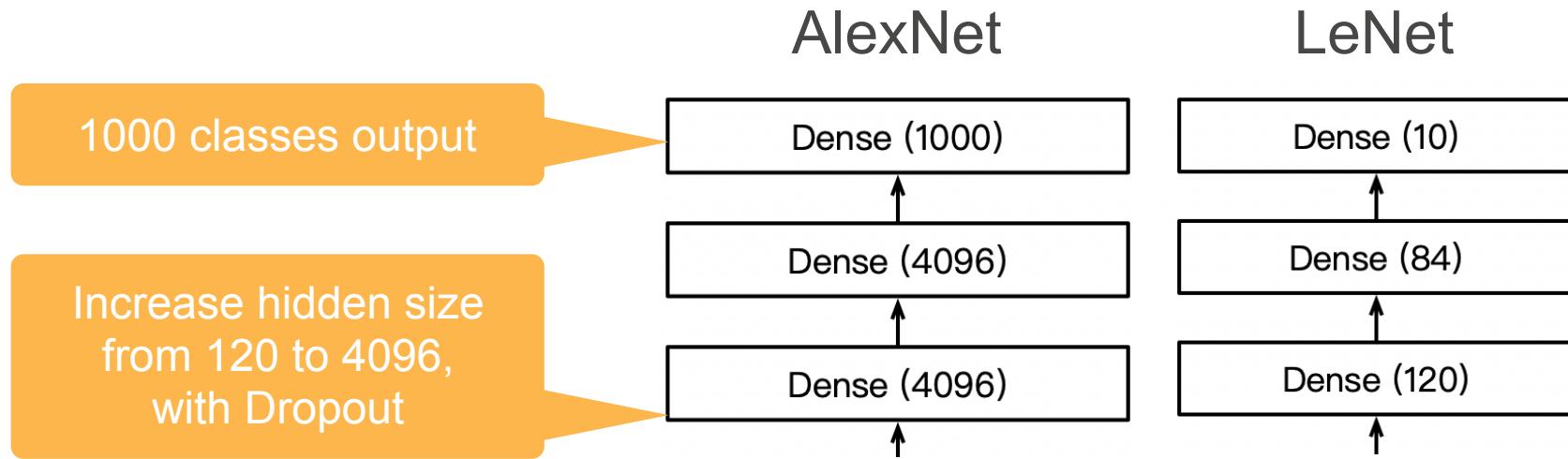
AlexNet Architecture



AlexNet Architecture



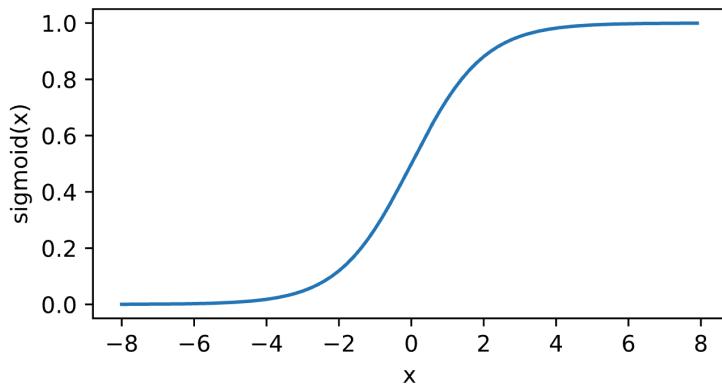
AlexNet Architecture



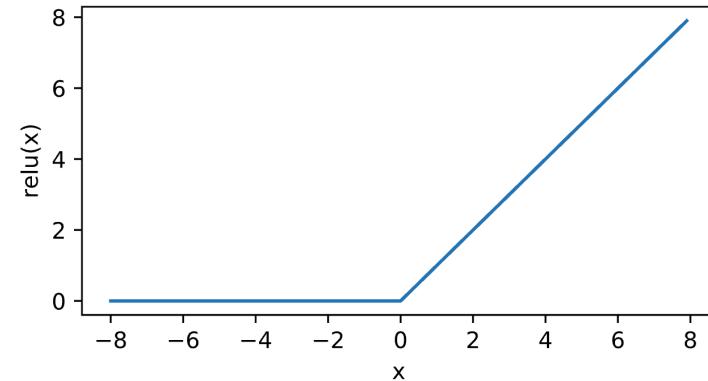
More Tricks

1. Change activation function from sigmoid to **ReLU**
(no more vanishing gradient)

$$\text{sigmoid}(x) = \frac{1}{1 + \exp(-x)}$$



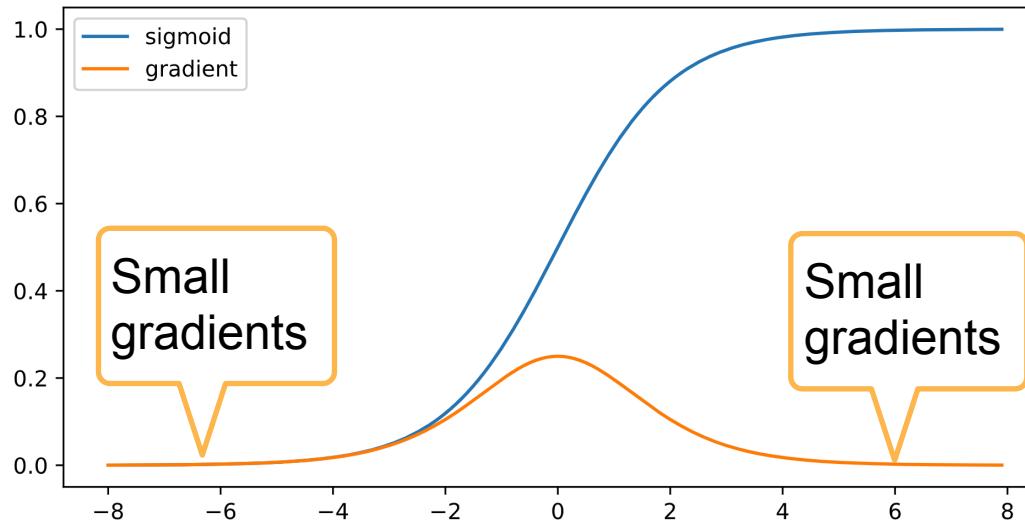
$$\text{ReLU}(x) = \max(x, 0)$$



Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$



Gradient Vanishing

- Use sigmoid as the activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \quad \sigma'(x) = \sigma(x)(1 - \sigma(x))$$

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^i} = \prod_{i=t}^{d-1} \text{diag}(\sigma'(\mathbf{W}^i \mathbf{h}^{i-1})) (\mathbf{W}^i)^T$ are products of $d-t$ small values

$$0.8^{100} \approx 2 \times 10^{-10}$$



Gradient Vanishing Issues

- Gradients with value 0
 - Severe with 16-bit floating points (Range: 6e-5 - 6e4)
- No progress in training
 - No matter how to choose learning rate
- Severe with bottom layers
 - Only top layers are well trained
 - No benefit to make networks deeper

More Tricks

1. Change activation function from sigmoid to **ReLU**
(no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers
(better robustness / regularization)

Dropout

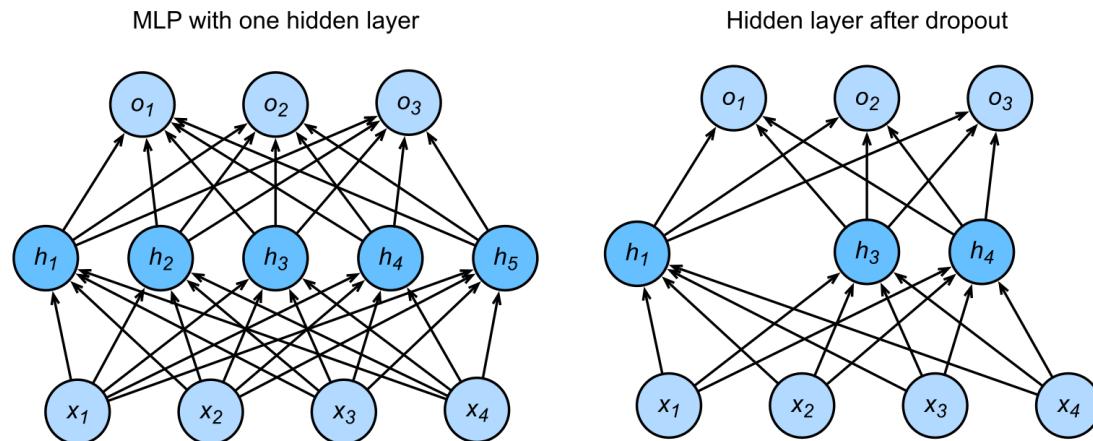
- Often apply dropout on the output of hidden fully-connected layers

$$\mathbf{h} = \sigma(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

$$\mathbf{o} = \mathbf{W}_2 \mathbf{h}' + \mathbf{b}_2$$

$$\mathbf{y} = \text{softmax}(\mathbf{o})$$



Dropout - Inference

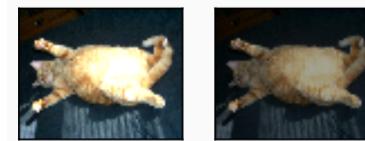
- Regularization is only used in training
- The dropout layer for inference is

$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

- Guarantee deterministic results

More Tricks

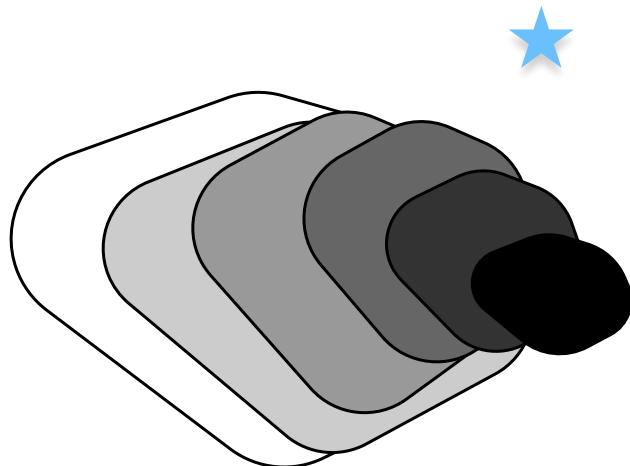
1. Change activation function from sigmoid to **ReLU**
(no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers
(better robustness / regularization)
3. Data augmentation



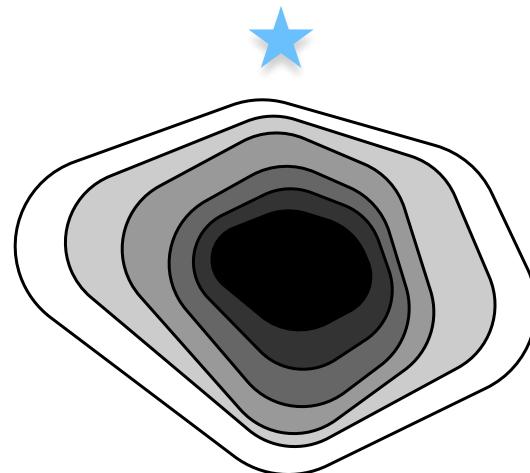
AlexNet Notebook



Residual Networks



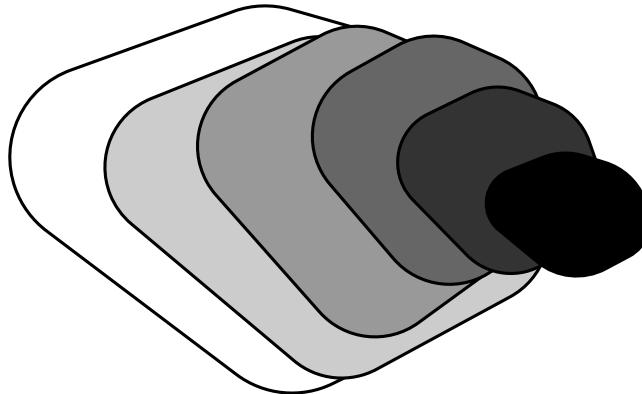
generic function classes



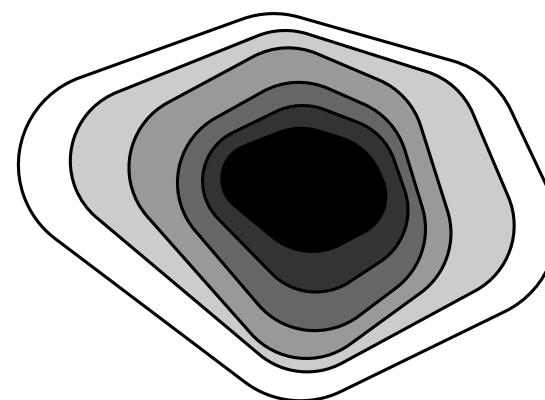
nested function classes

Does adding layers improve accuracy?

$$f^*_{\mathcal{F}} := \underset{f}{\operatorname{argmin}} L(X, Y, f) \text{ subject to } f \in \mathcal{F}$$



generic function classes



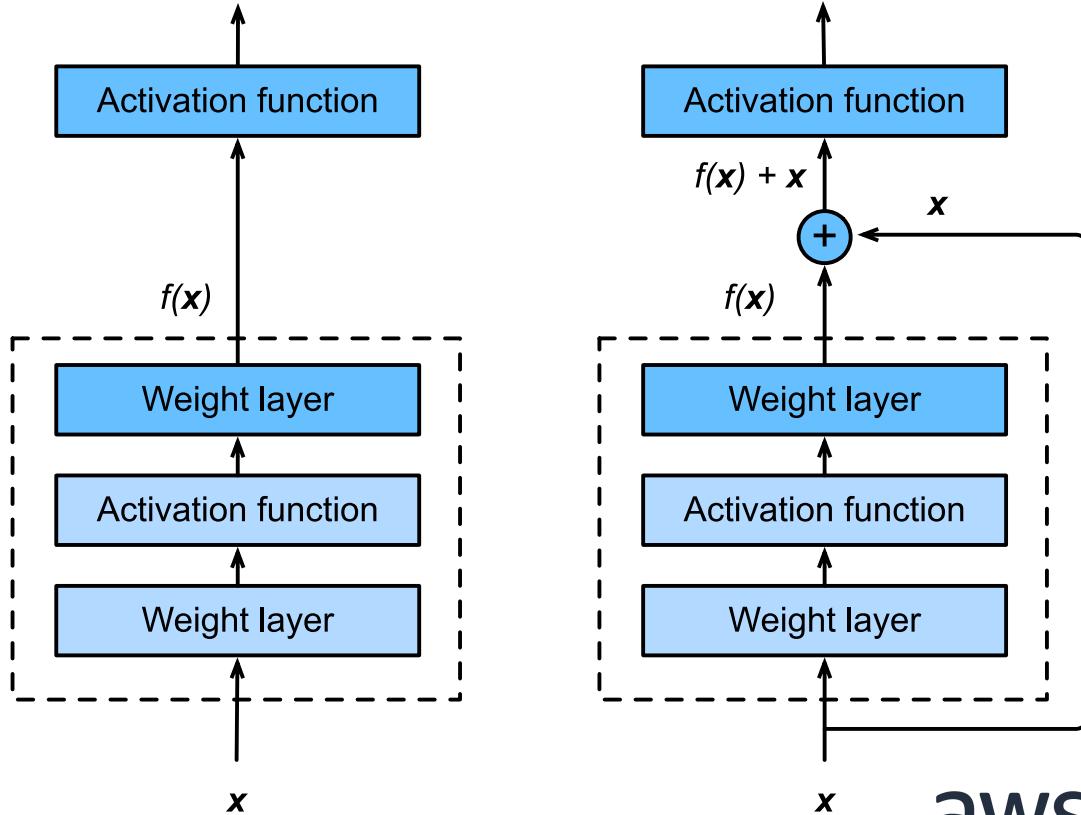
nested function classes



Residual Networks

- Adding a layer **changes** function class
- We want to **add to** the function class
- ‘Taylor expansion’ style parametrization

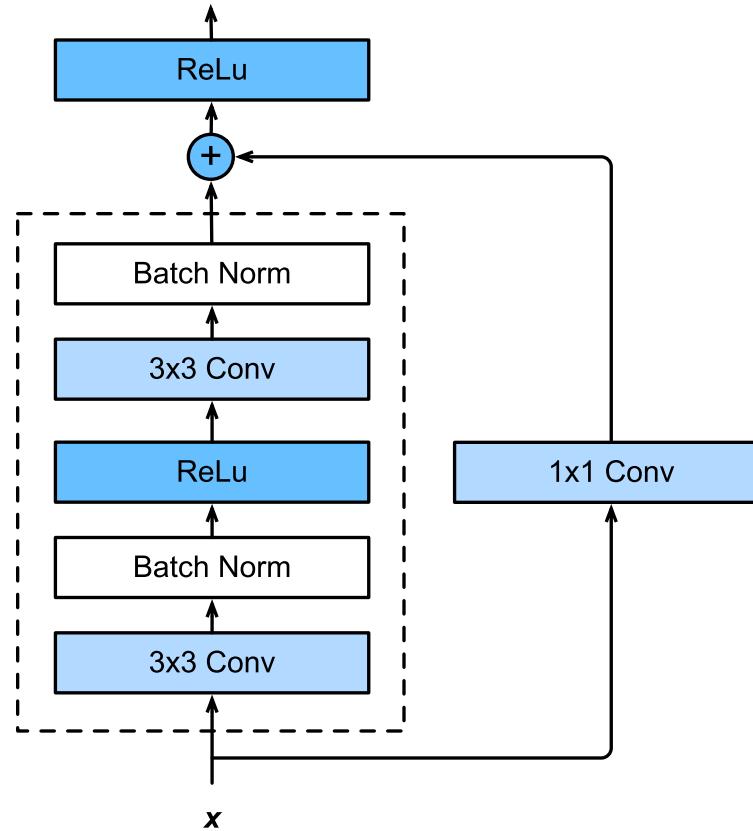
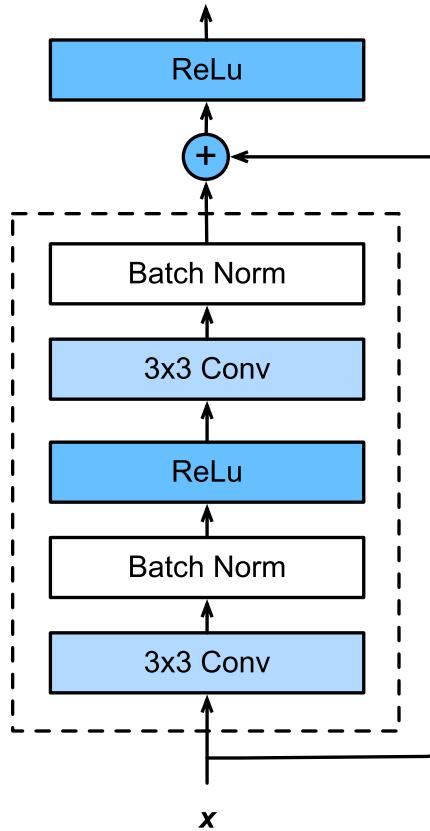
$$f(x) = x + g(x)$$



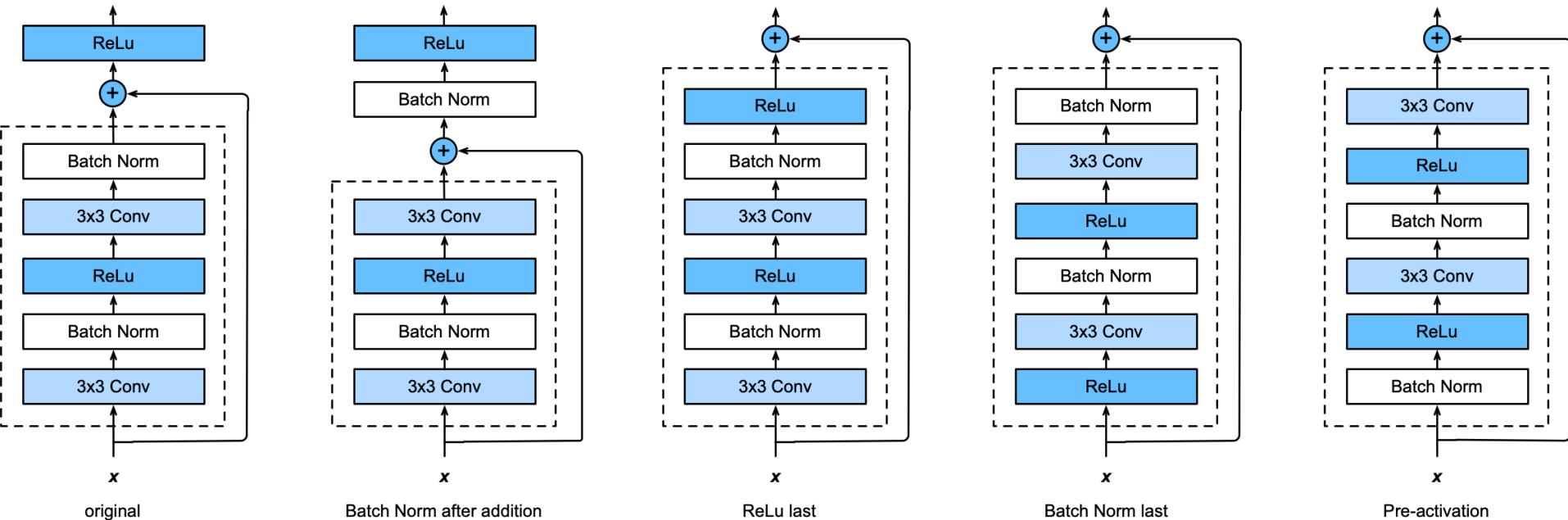
He et al., 2015



ResNet Block in detail



The many flavors of ResNet blocks



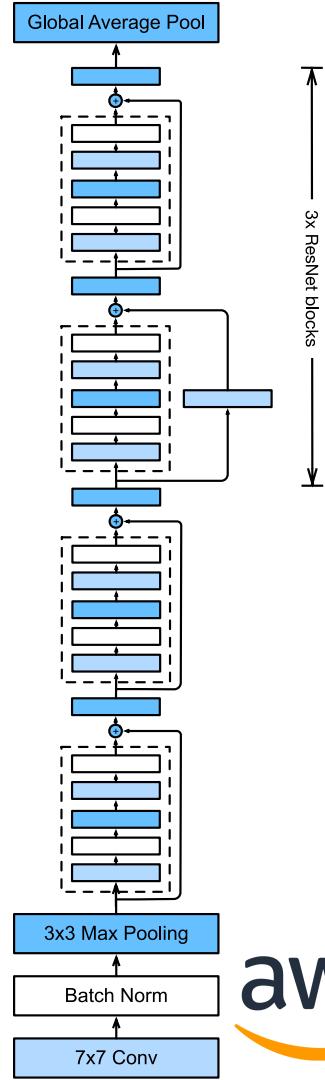
Try every permutation

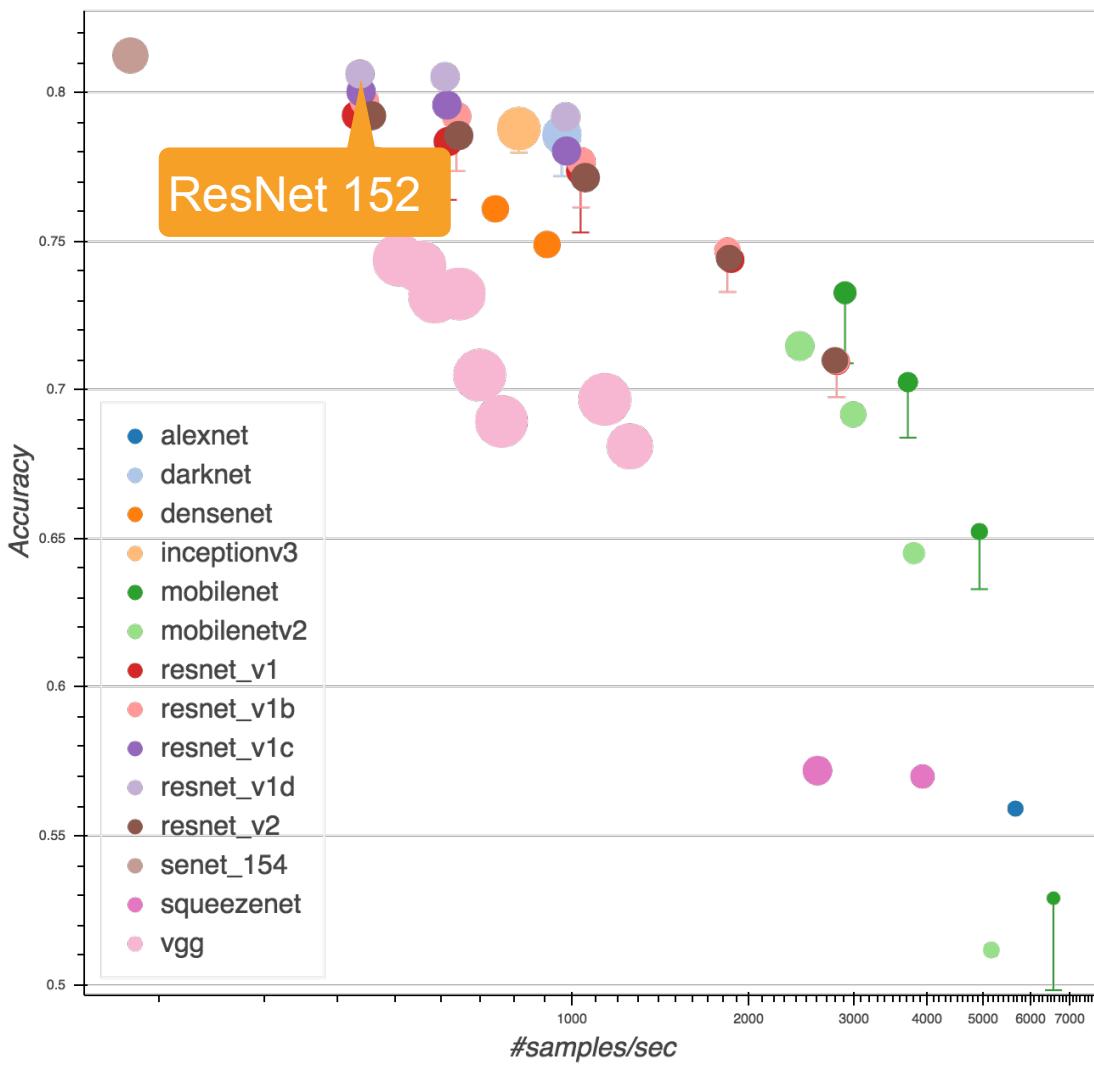


Putting it all together

- Same block structure as e.g. VGG or GoogleNet
- Residual connection to add to expressiveness
- Pooling/stride for dimensionality reduction
- Batch Normalization for capacity control

... train it at scale ...





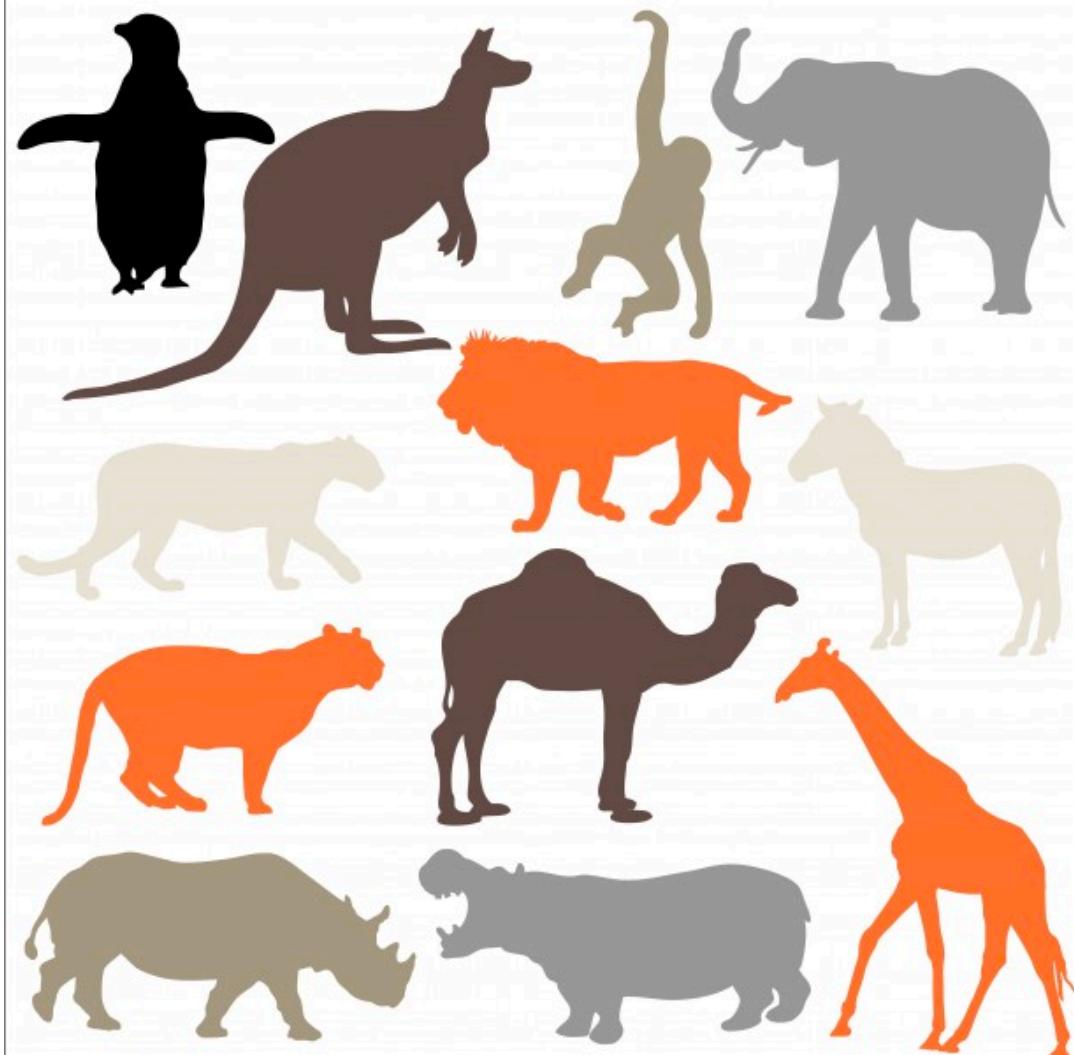
GluonCV Model Zoo
https://gluon-cv.mxnet.io/model_zoo/classification.html



Jupyter Notebook



More Ideas



DenseNet (Huang et al., 2016)

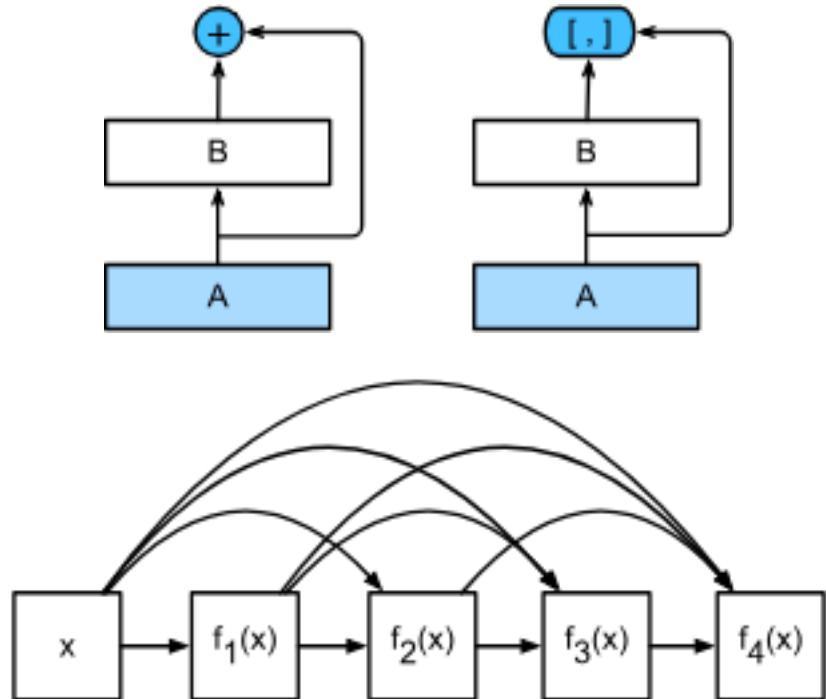
- ResNet combines x and $f(x)$
- DenseNet uses higher order ‘Taylor series’ expansion

$$x_{i+1} = [x_i, f_i(x_i)]$$

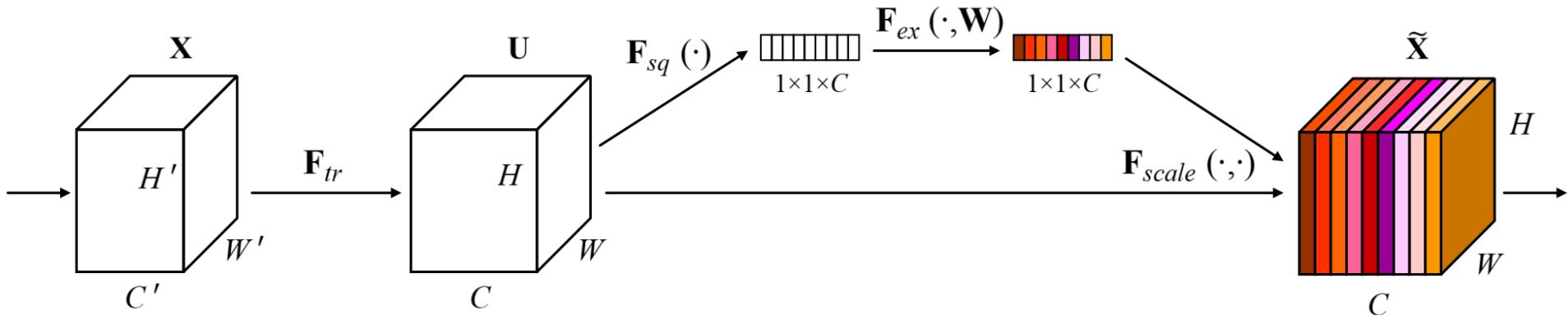
$$x_1 = x$$

$$x_2 = [x, f_1(x)]$$

$$x_2 = [x, f_1(x), f_2([x, f_1(x)])]$$



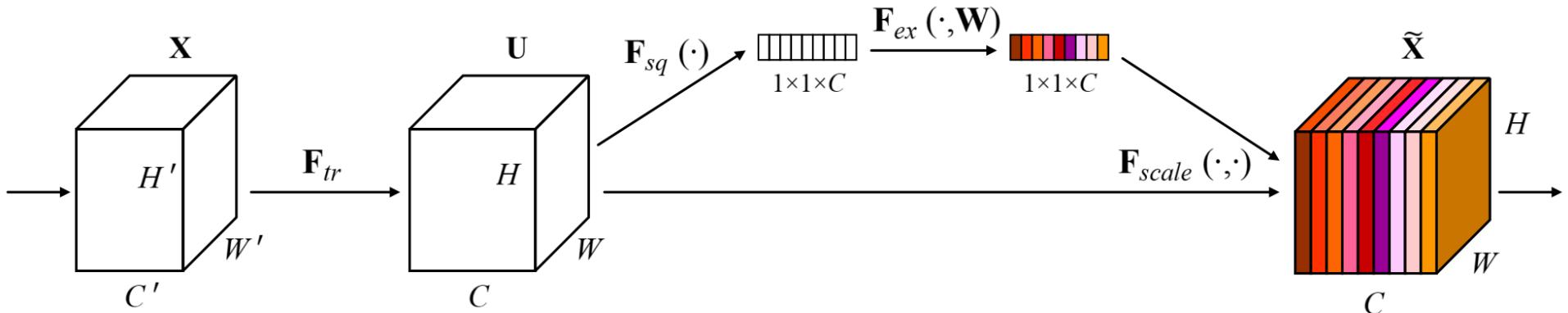
Squeeze-Excite Net (Hu et al., 2017)



For the standard neural net, the network weights each of its channels **equally** when creating the output feature maps.

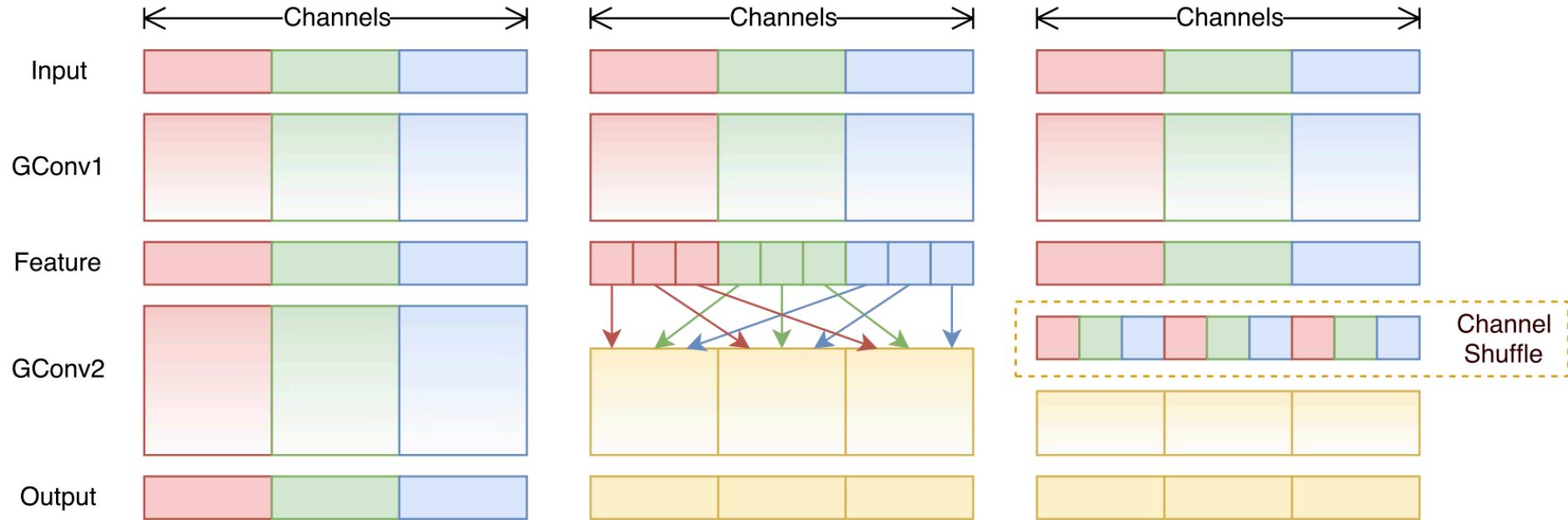
SENets are all about changing this by **adding a content aware mechanism** to weight each channel adaptively.

Squeeze-Excite Net (Hu et al., 2017)



- Learn global weighting function per channel
- Allows for fast information transfer between pixels in different locations of the image

ShuffleNet (Zhang et al., 2018)



- ResNeXt breaks convolution into channels
- ShuffleNet mixes by grouping (very efficient for mobile)