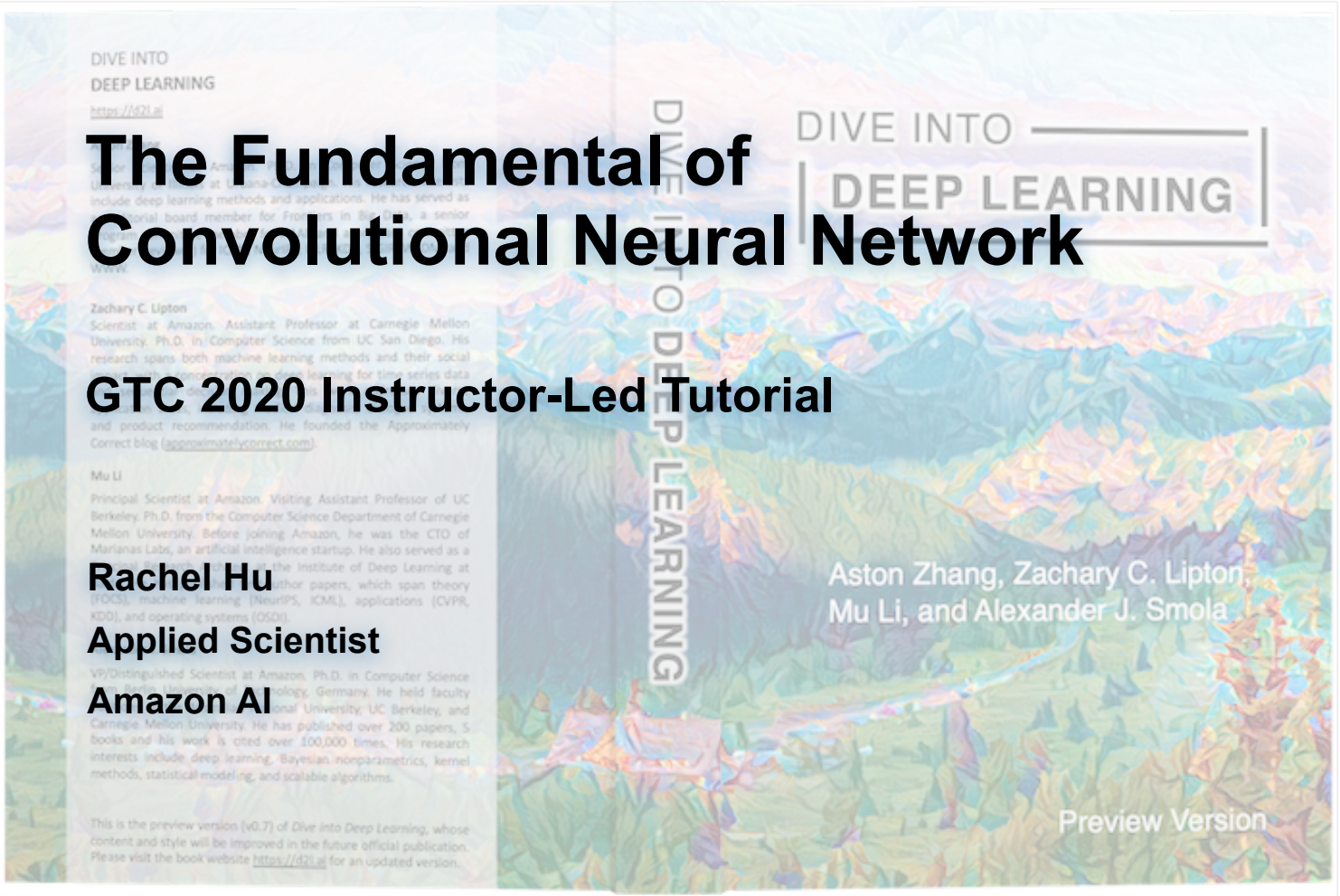


The Fundamental of Convolutional Neural Network

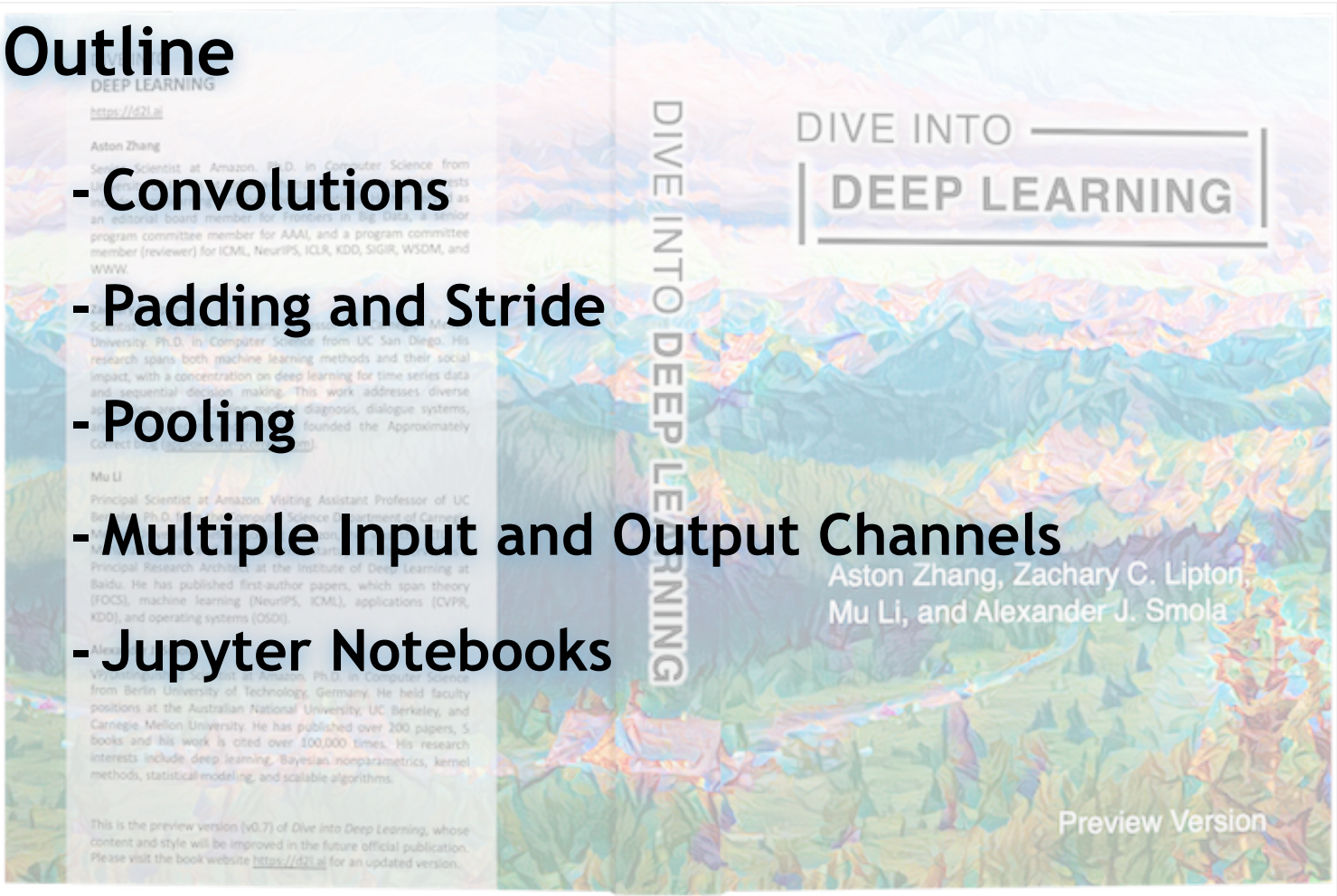
GTC 2020 Instructor-Led Tutorial

Rachel Hu

Applied Scientist

Amazon AI

Outline



- Convolutions
- Padding and Stride
- Pooling
- Multiple Input and Output Channels
- Jupyter Notebooks

Image Classification

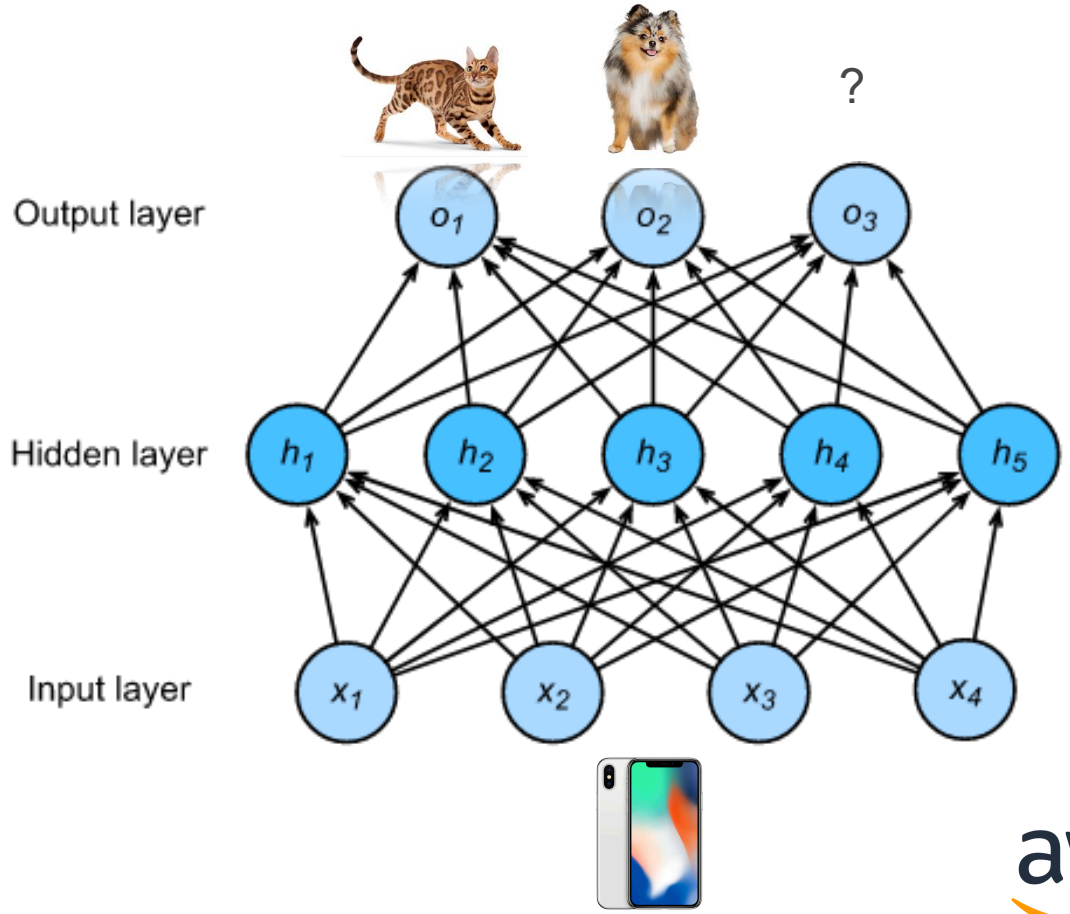


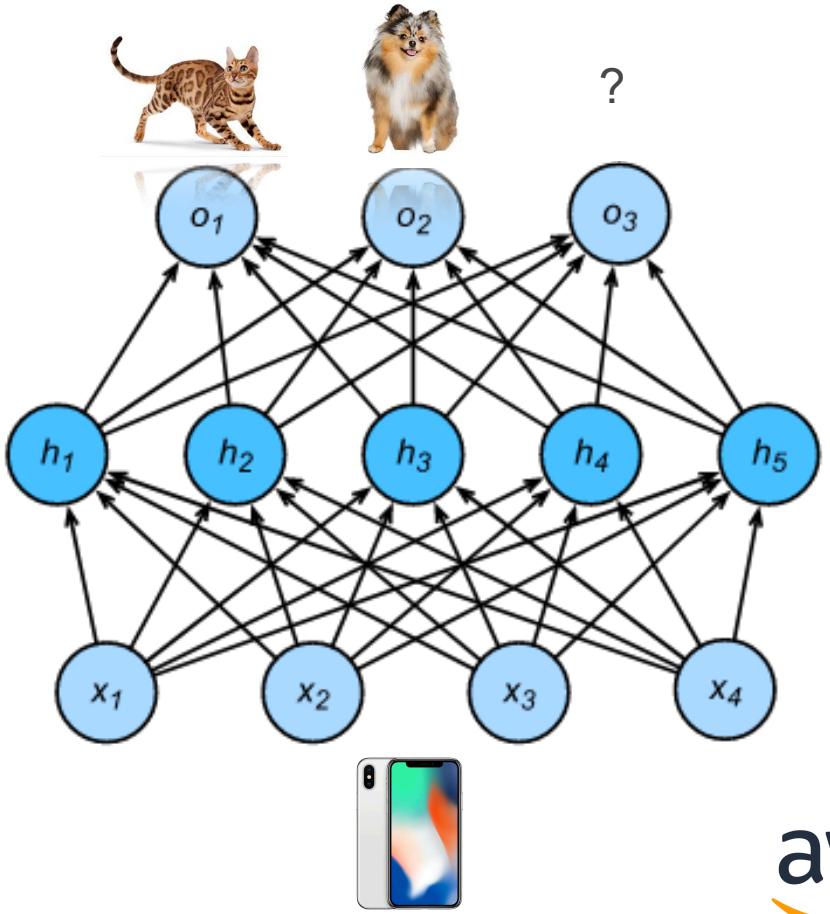
Image Classification

Training …

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

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

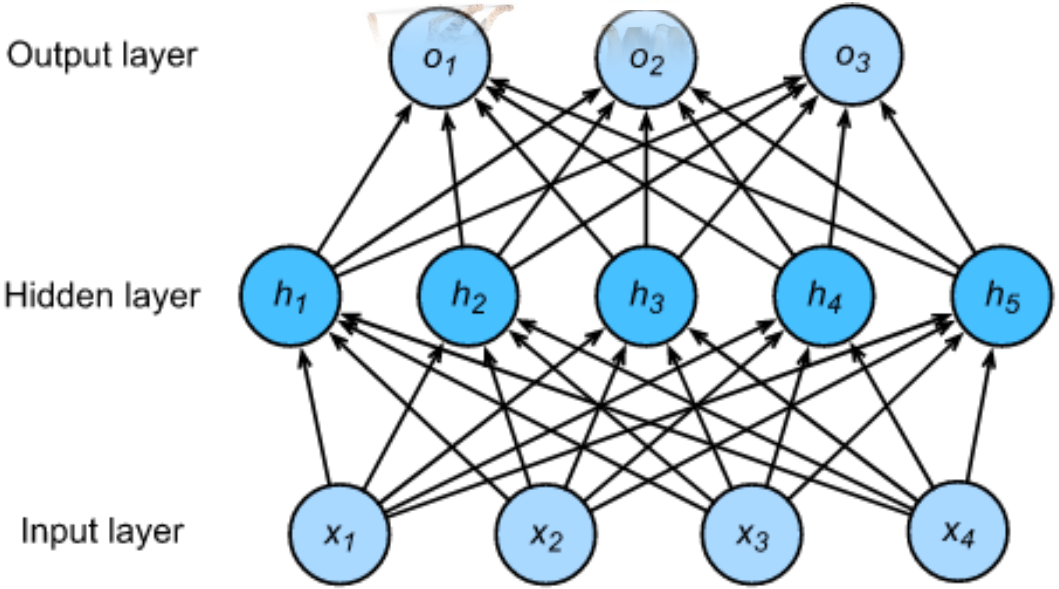
$$\mathbf{y} = \text{softmax}(o)$$



aws

Image Classification

?



A RGB image has **36M** elements
(36M features)

Dual
12MP
wide-angle and
telephoto cameras

aws

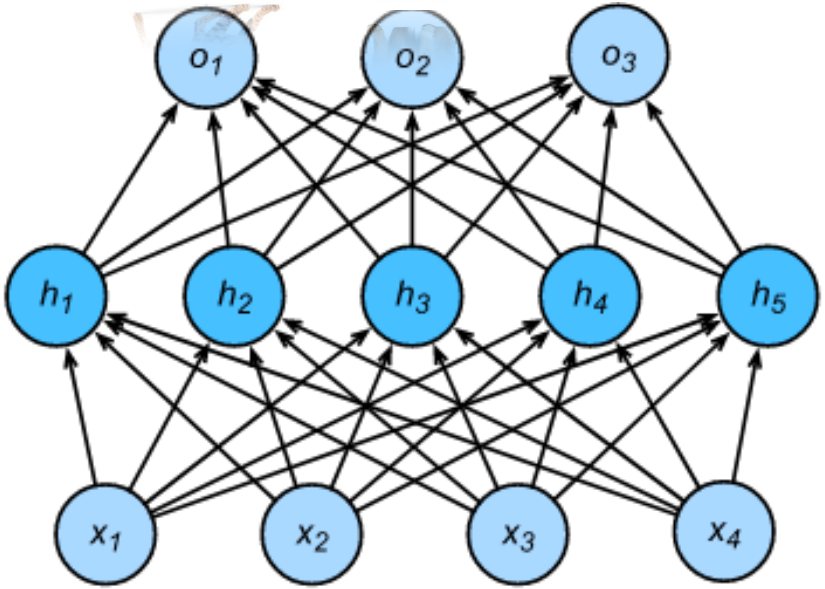
Image Classification

?

Output layer

Hidden layer

Input layer



100 neurons

A RGB image has **36M** elements
(36M features)

Dual
12MP
wide-angle and
telephoto cameras

aws

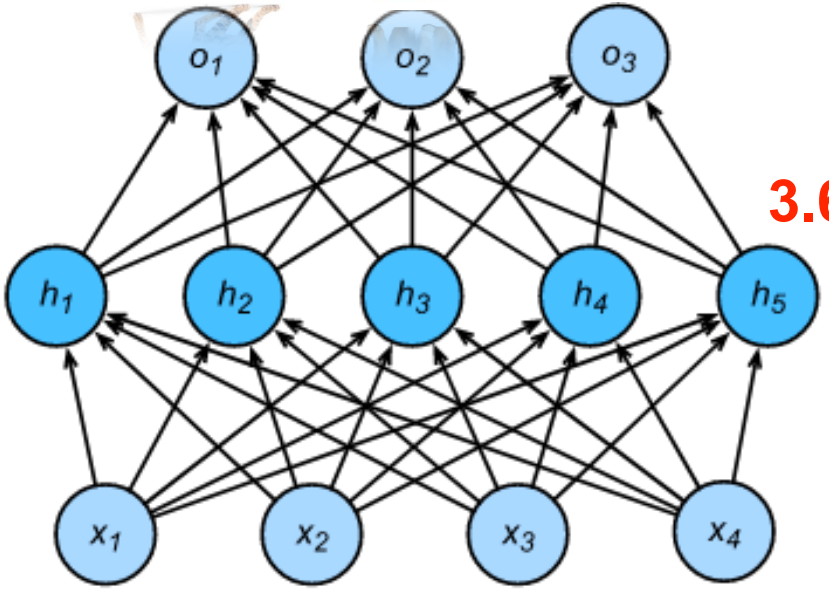
Image Classification

?

Output layer

Hidden layer

Input layer



100 neurons

3.6B parameters
=
14GB

A RGB image has **36M** elements
(36M features)

Dual
12MP
wide-angle and
telephoto cameras

aws


Can we reduce the number of parameters a bit?
aws


Can we reduce the number of parameters a bit?
Yes!
Convolutions!
aws

Where is Waldo?

Where is Waldo?

Two Principles

1. Translation Invariance:

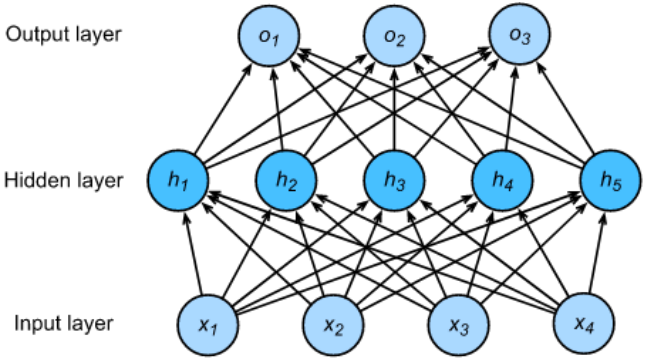
Our vision systems should, in some sense, **respond similarly to the same object regardless of where it appears** on the image.

Two Principles

2. Locality:

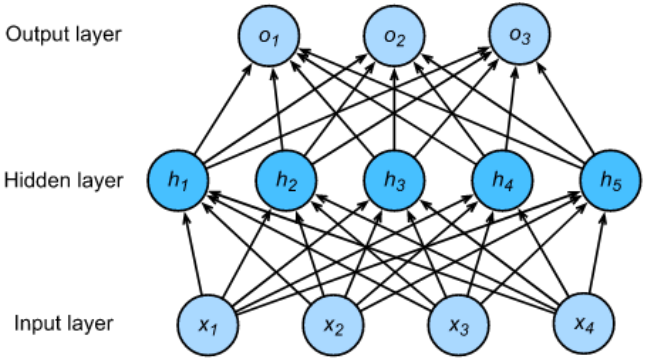
Our visions systems should, in some sense, **focus on somewhat local regions**, without regard for what else is happening on the image at greater distances.

2-D Convolution Idea



14GB

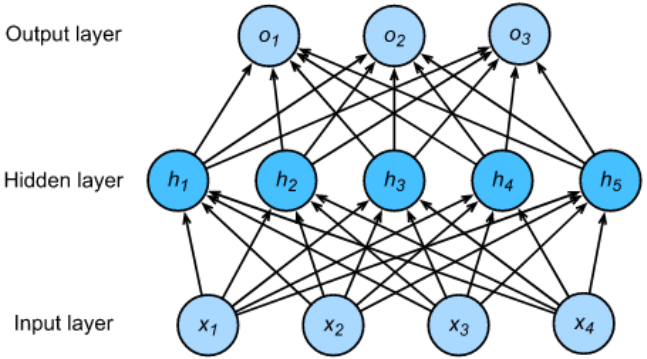
2-D Convolution Idea



- Split the input into windows
- Apply *the same* dense neural network for every window

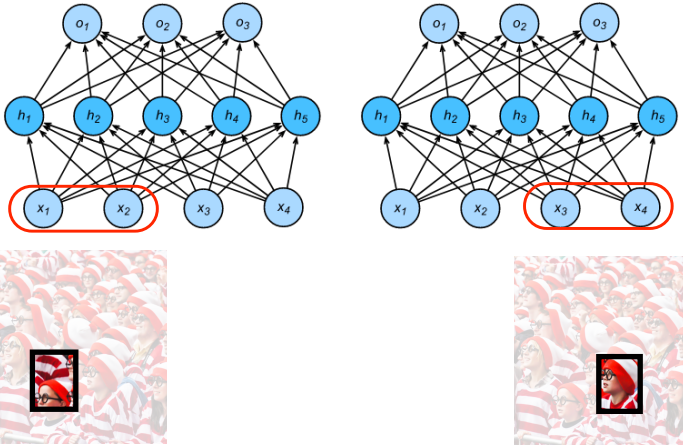
14GB

2-D Convolution Idea



14GB

- Split the input into windows
- Apply *the same* dense neural network for every window



2-D Convolution Layer

- Each pixel on the image can be realized as $x_{i,j}$
- Each “weight” on the filter window can be defined by $v_{a,b}$

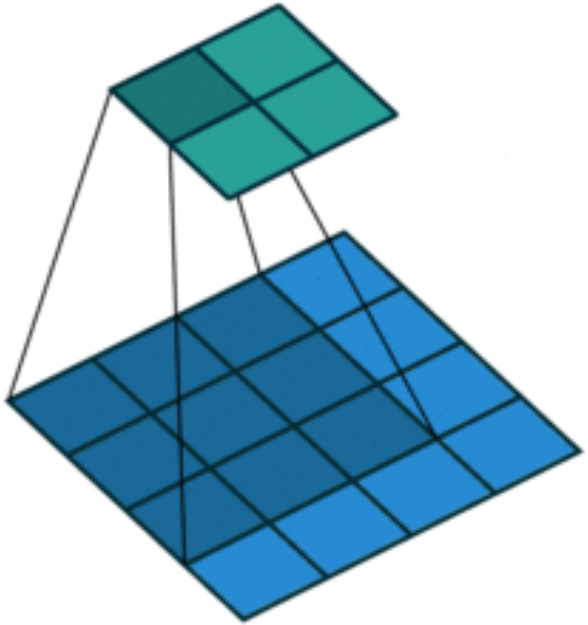
2-D Convolution Layer

- Each pixel on the image can be realized as $x_{i,j}$
- Each “weight” on the filter window can be defined by $v_{a,b}$

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

2-D Convolution Layer

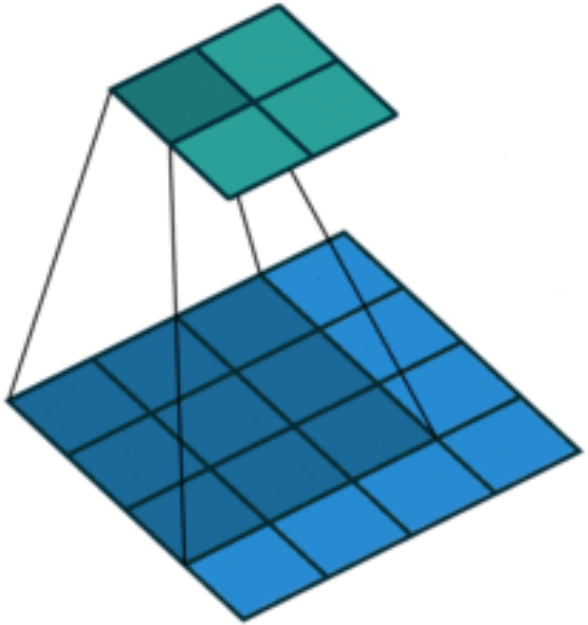
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$



(vdumoulin@ Github)

2-D Convolution Layer

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

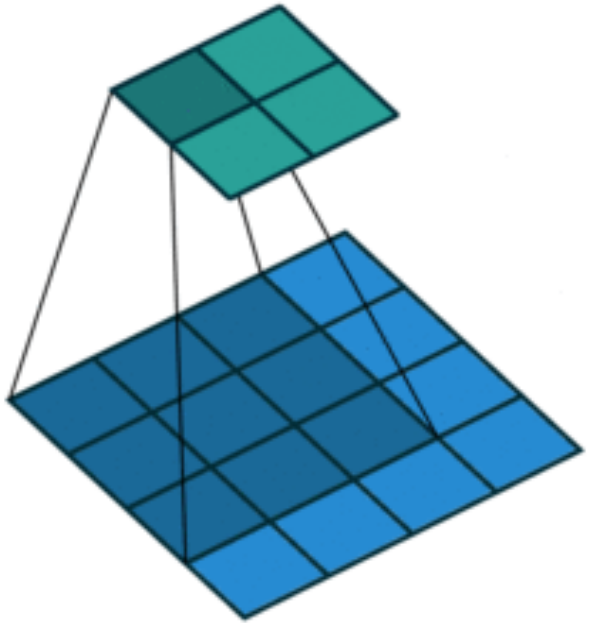


(vdumoulin@ Github)

2-D Convolution Layer

Input				Kernel			Output	
0	1	2	*	0	1	=	19	25
3	4	5		2	3		37	43
6	7	8						

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

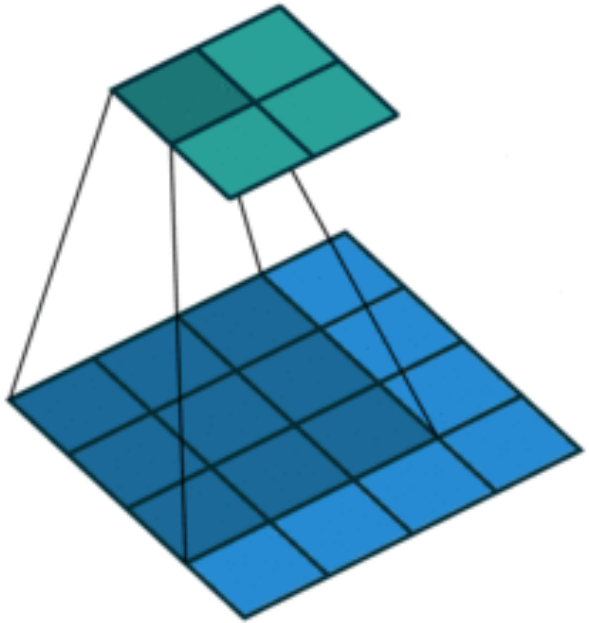


(vdumoulin@ Github)

2-D Convolution Layer

Input				Kernel			Output	
0	1	2	*	0	1	=	19	25
3	4	5		2	3		37	43
6	7	8						

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



(vdumoulin@ Github)

2-D Convolution Layer

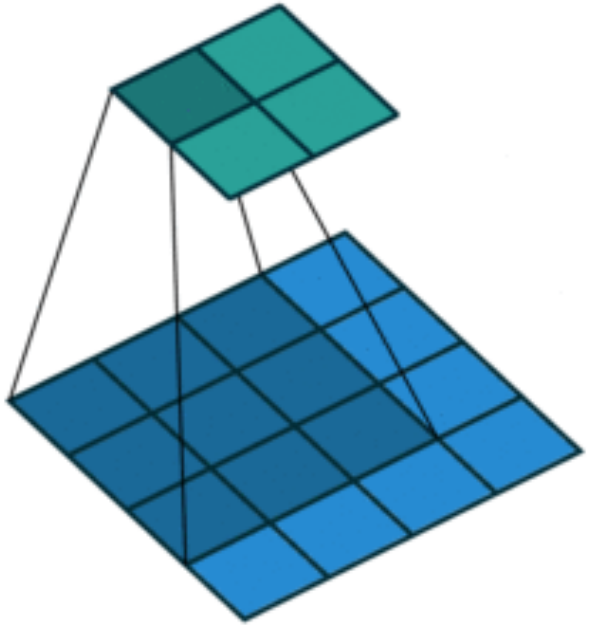
0	1	2
3	4	5
6	7	8

$*$

0	1
2	3

$=$

19	25
37	43



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

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

(vdumoulin@ Github)

2-D Convolution Layer

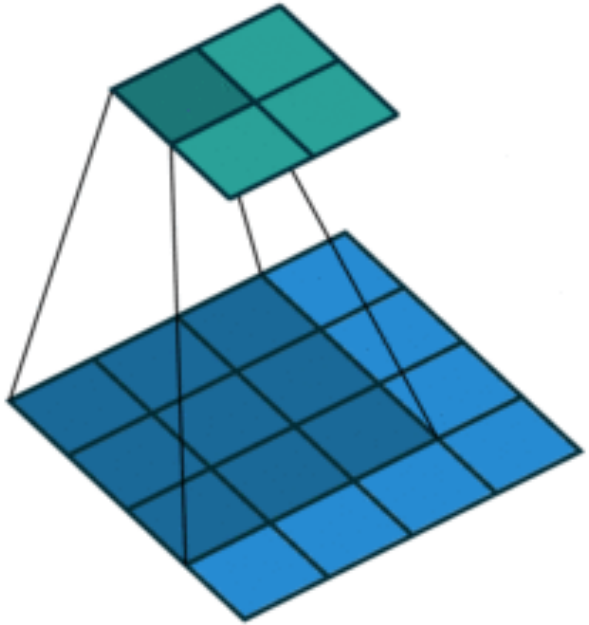
0	1	2
3	4	5
6	7	8

$*$

0	1
2	3

$=$

19	25
37	43



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

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

(vdumoulin@ Github)

2-D Convolution Layer

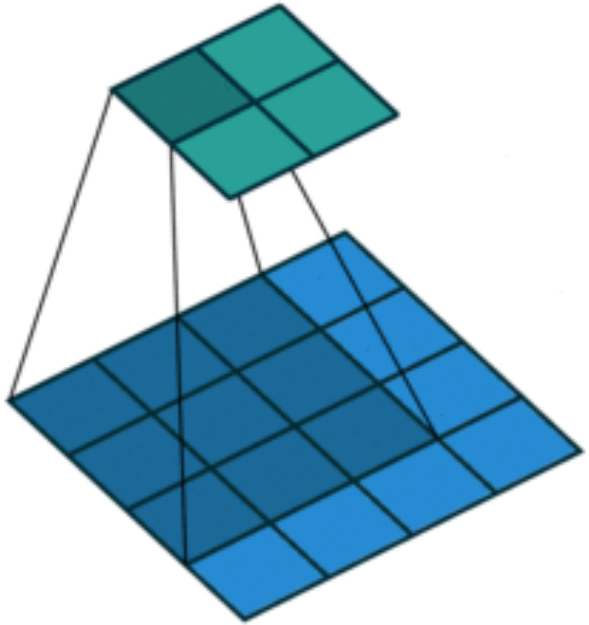
0	1	2
3	4	5
6	7	8

$*$

0	1
2	3

$=$

19	25
37	43



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

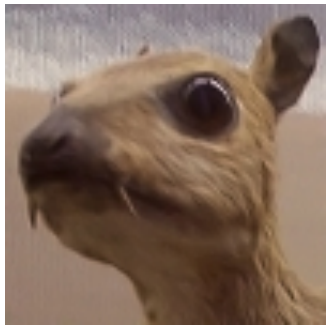
$$\mathbf{Y} = \mathbf{X} \star \mathbf{W} + b$$

(vdumoulin@ Github)

- W and b are the trainable parameters

Fliters

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



(wikipedia)

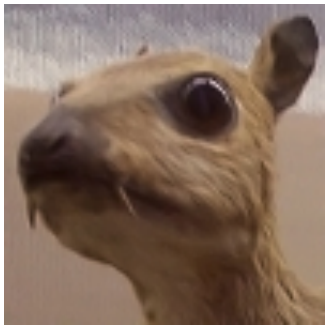
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

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

Fliters

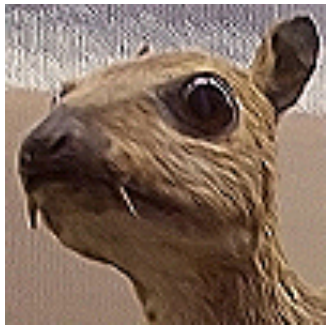
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Edge Detection



(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

What do we do near the boundary?

The original convolution window may ignore this Waldo at the boundary …

What do we do near the boundary?

The original convolution window may ignore this Waldo at the boundary …

Padding can help!

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

*

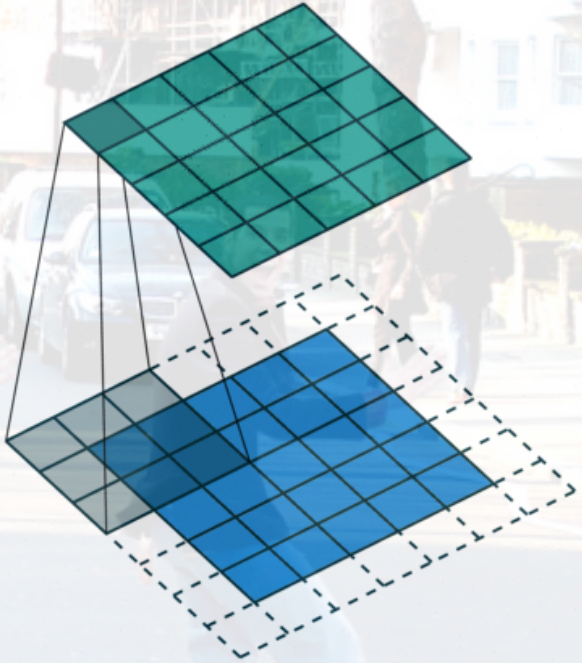
Kernel

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

*

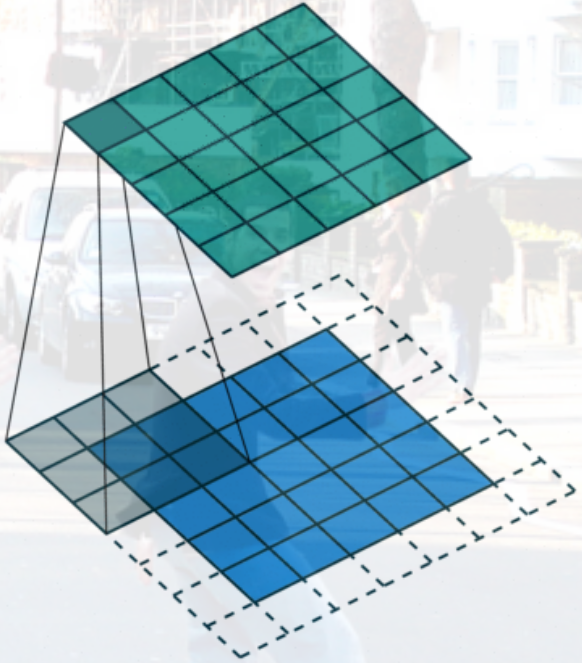
Kernel

0	1
2	3

=

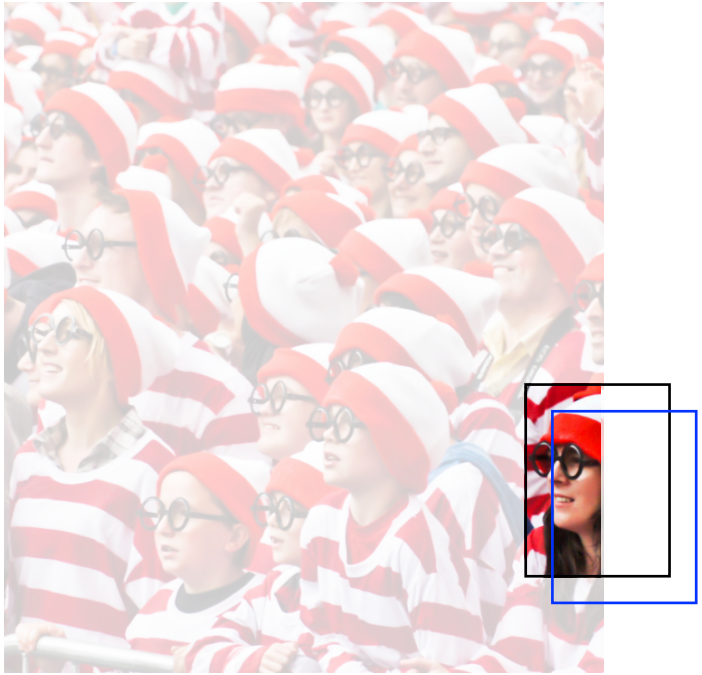
Output

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

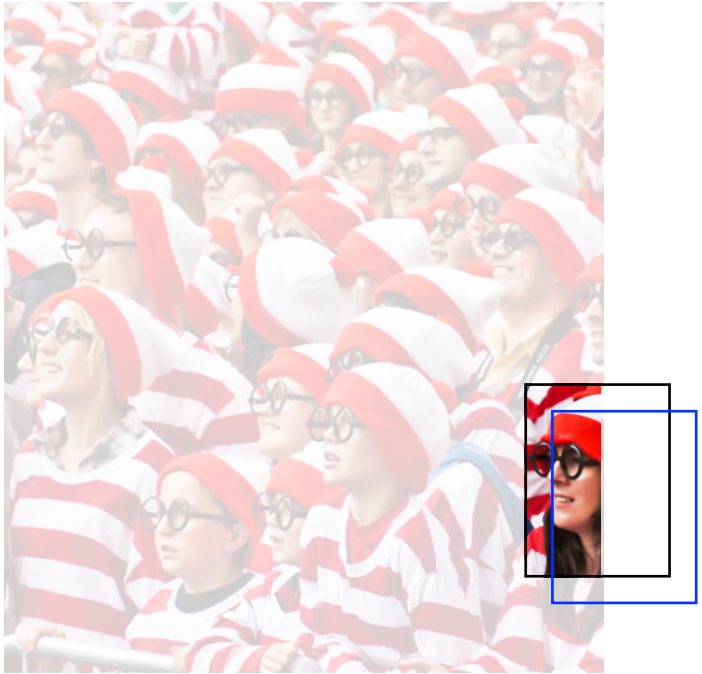


$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

How about two nearly identical windows?

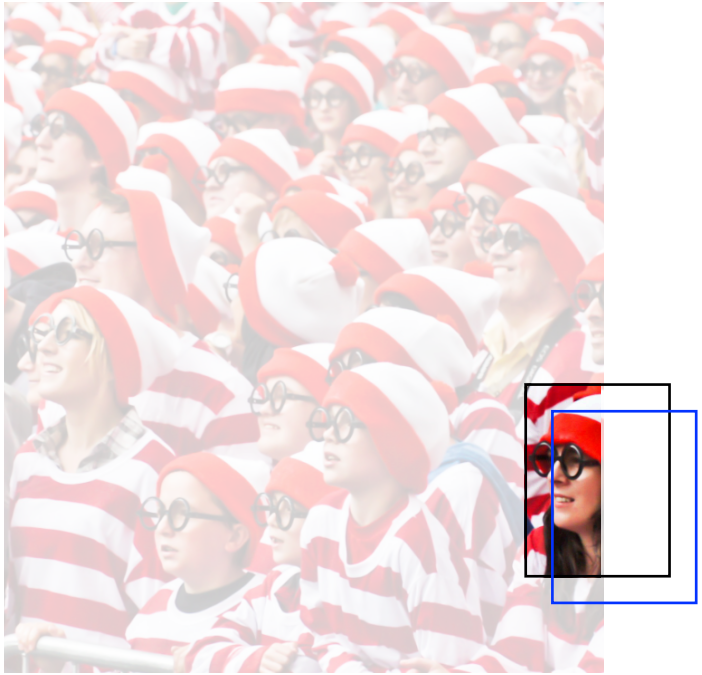


How about two nearly identical windows?



The original convolution window may be too computational expensive to slide one pixel at a time…

How about two nearly identical windows?

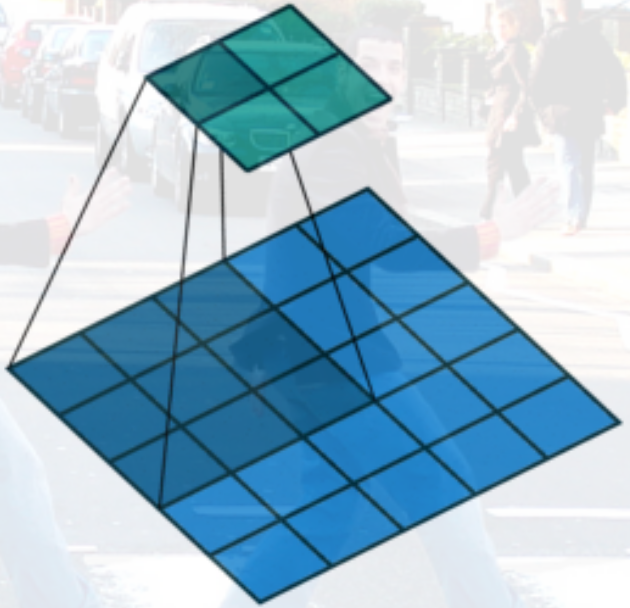


The original convolution window may be too computational expensive to slide one pixel at a time…

Stride can help!

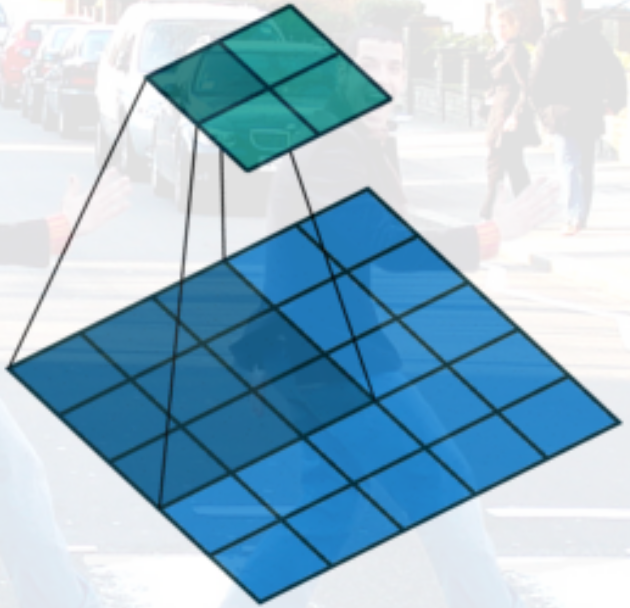
Stride

- Stride is the number of “unit” the kernel shifted per slide over rows/ columns



Stride

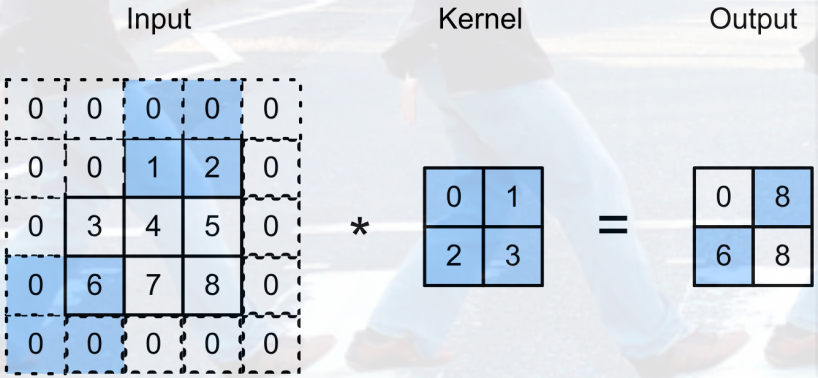
- Stride is the number of “unit” the kernel shifted per slide over rows/ columns



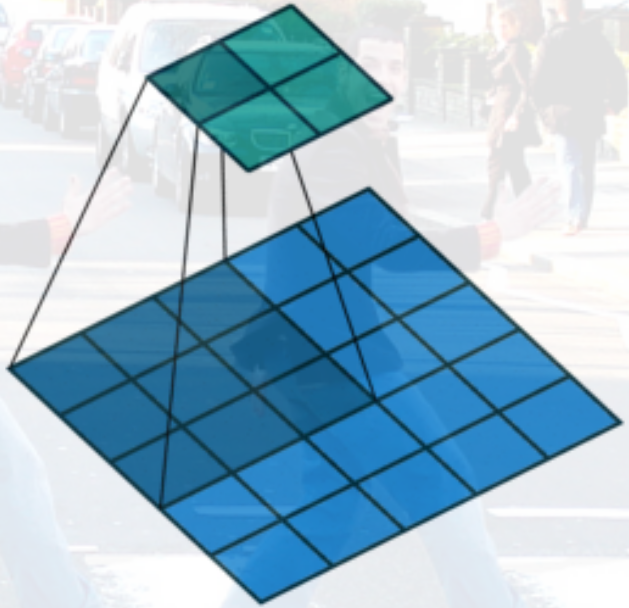
Stride

- Stride is the number of “unit” the kernel shifted per slide over rows/ columns

Strides of 3 for height and 2 for width



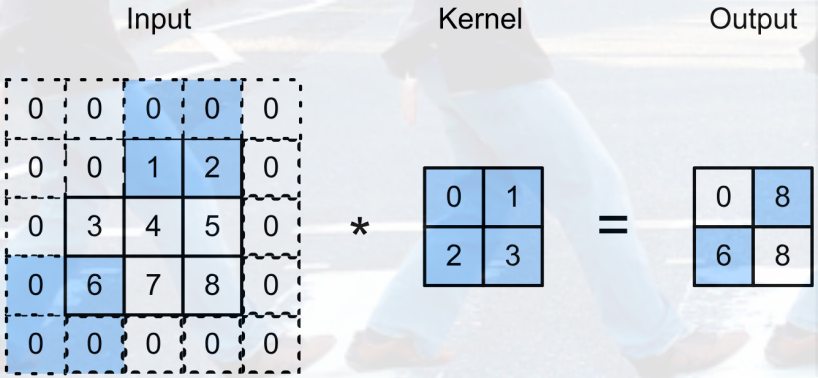
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



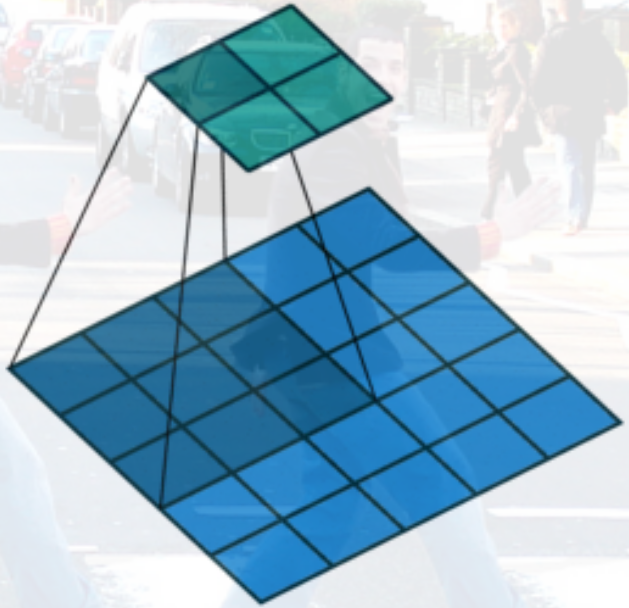
Stride

- Stride is the number of “unit” the kernel shifted per slide over rows/ columns

Strides of 3 for height and 2 for width



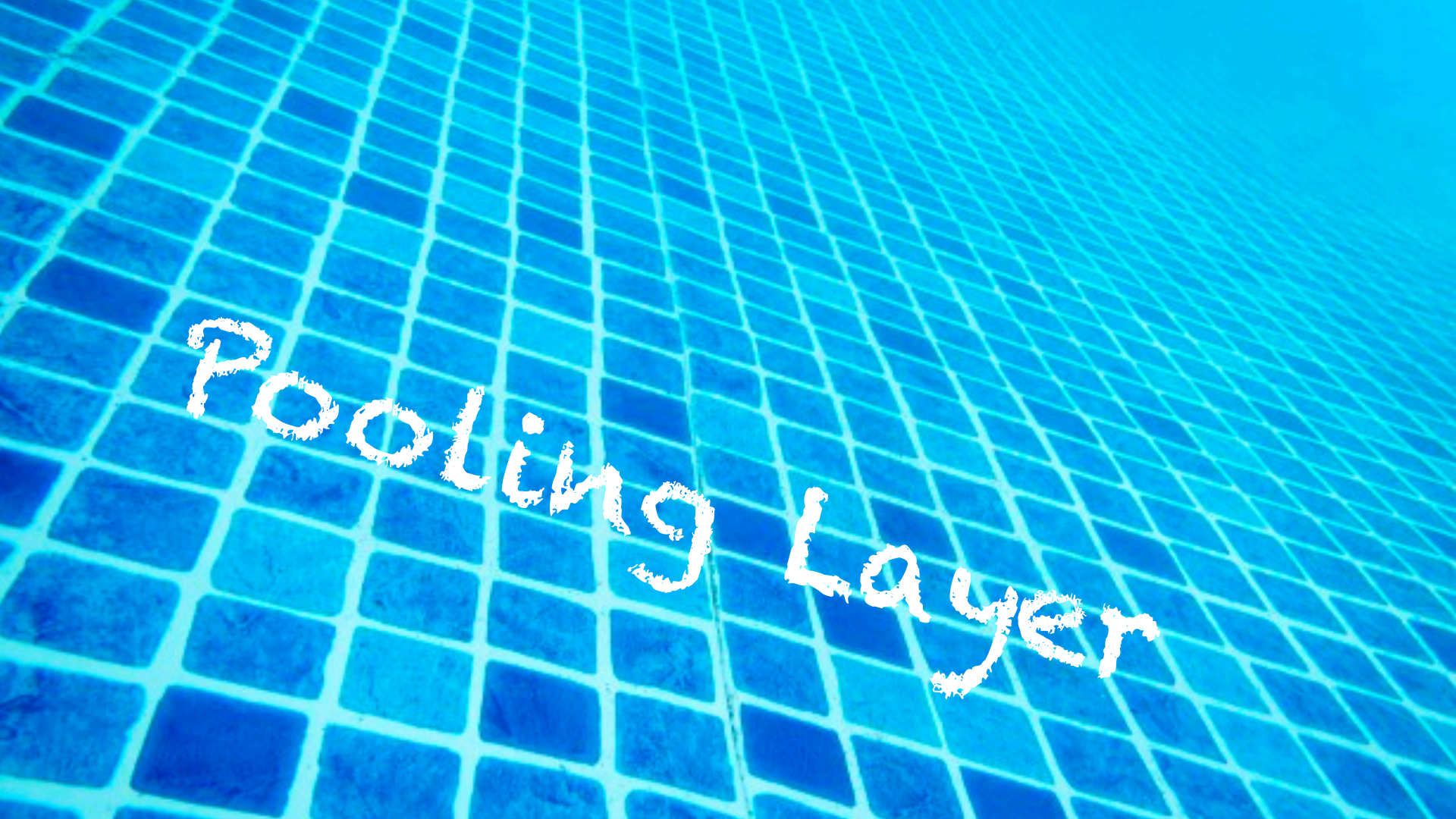
$$0 \times 0 + 0 \times 1 + 1 \times 2 + 2 \times 3 = 8$$
$$0 \times 0 + 6 \times 1 + 0 \times 2 + 0 \times 3 = 6$$



How to calculate the shape of the output?

- Given:
  - Input shape: (n_h, n_w)
  - Kernel size: (k_h, k_w)
  - Padding size: (p_h, p_w)
  - Stride size: (s_h, s_w)
- The output shape is

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


Pooling Layer

Max Pooling

- Returns the maximal value in the pooling window

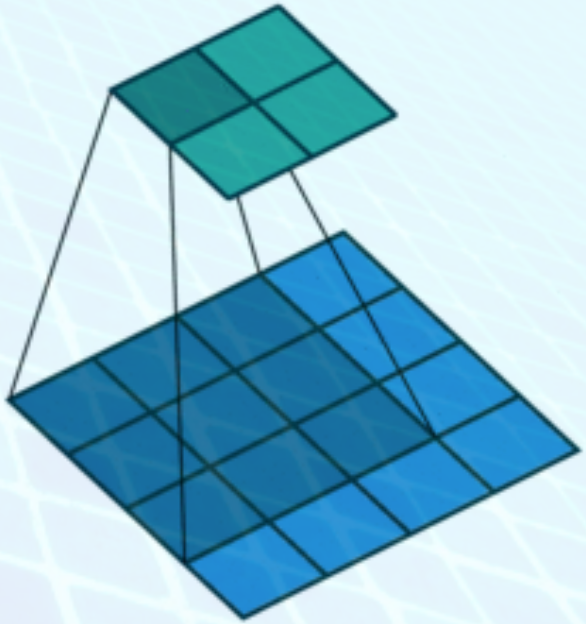
Input

0	1	2
3	4	5
6	7	8

2 x 2 Max Pooling

Output

4	5
7	8



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

Max Pooling

- Returns the maximal value in the pooling window

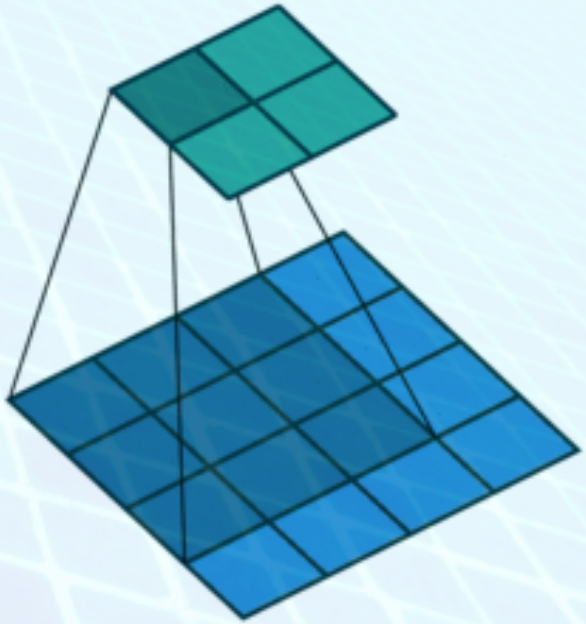
Input

0	1	2
3	4	5
6	7	8

2 x 2 Max Pooling

Output

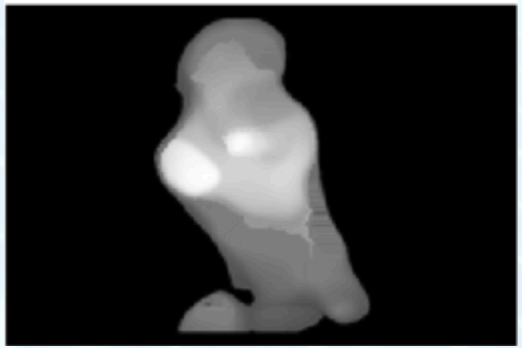
4	5
7	8



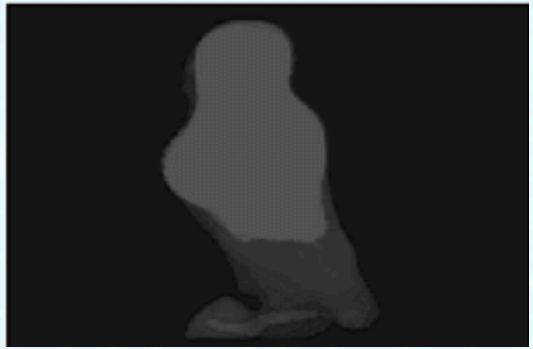
$$\max(0,1,3,4) = 4$$

Average Pooling

Max pooling



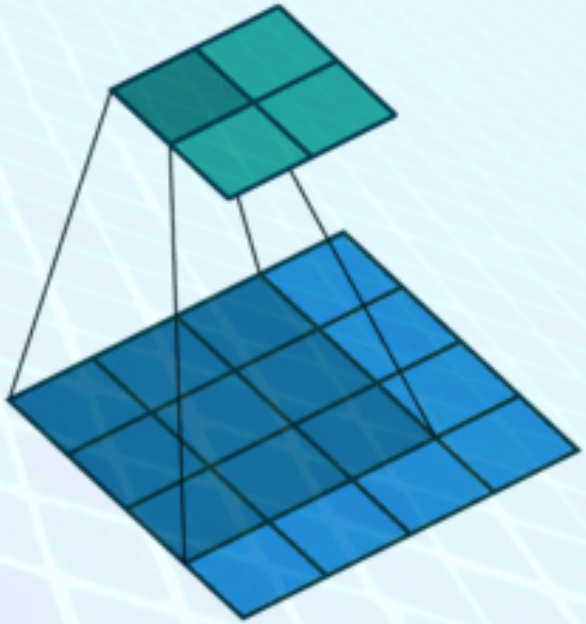
Average pooling



- Max pooling: the **strongest** pattern signal in a window
- Average pooling: the **average** signal strength in a window

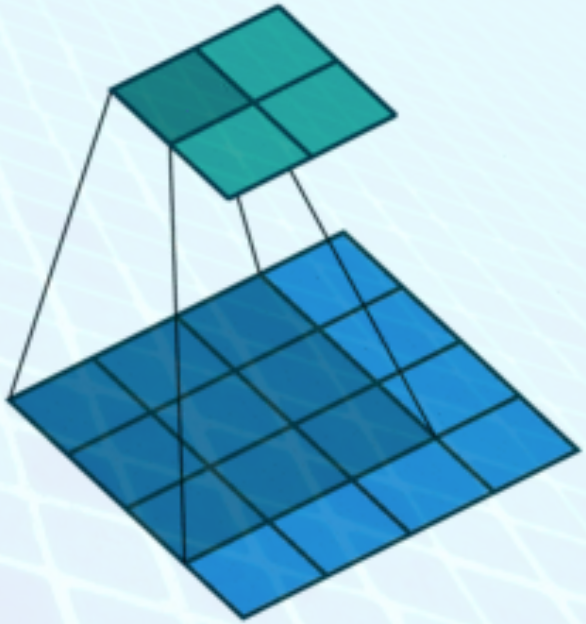
Pooling VS Convolution

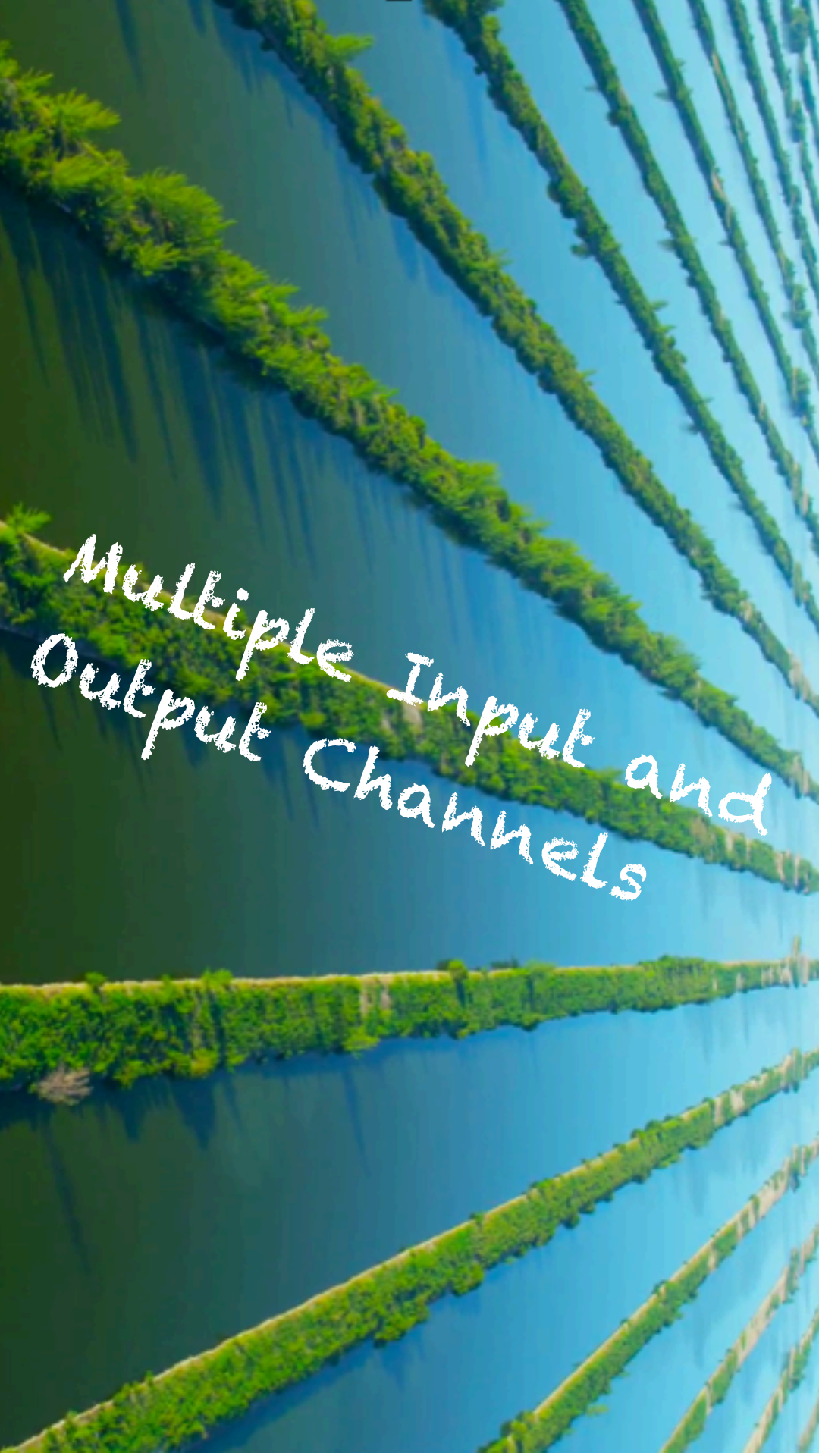
- Pooling layers can apply similar padding and stride as convolutional layers
- Pooling has no kernel (weights) to train



Pooling VS Convolution

- Pooling layers can apply similar padding and stride as convolutional layers
- Pooling has no kernel (weights) to train



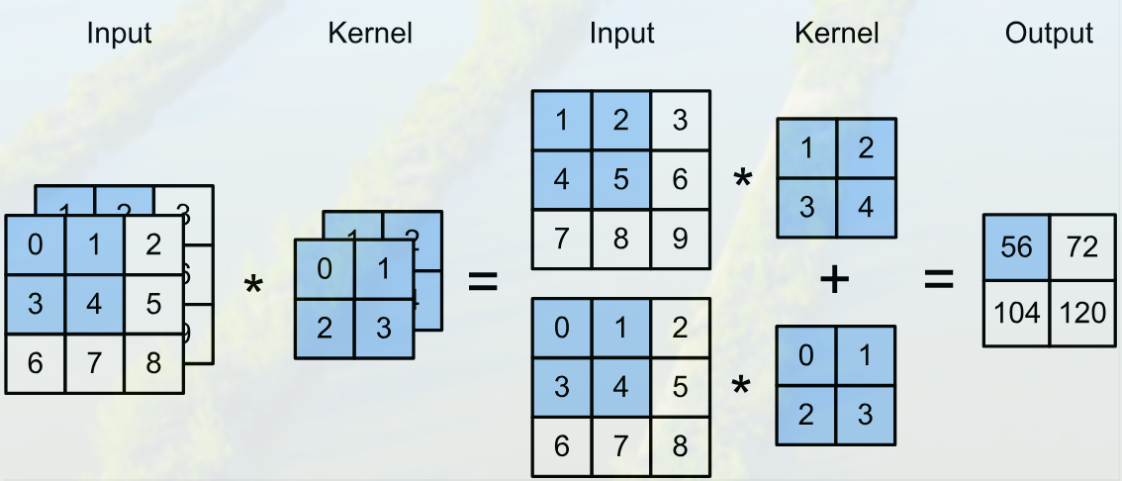

Multiple Input and
Output Channels

Multiple Input Channels

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

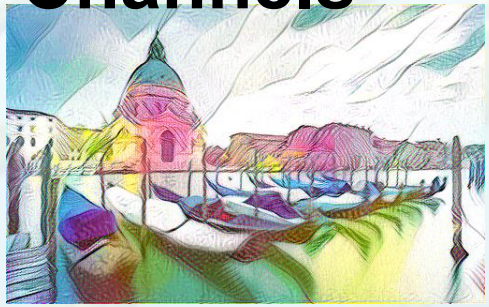
Multiple Input Channels

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



$$(1 \times 1 + 2 \times 2 + 4 \times 3 + 5 \times 4)$$
$$+(0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3)$$
$$= 56$$

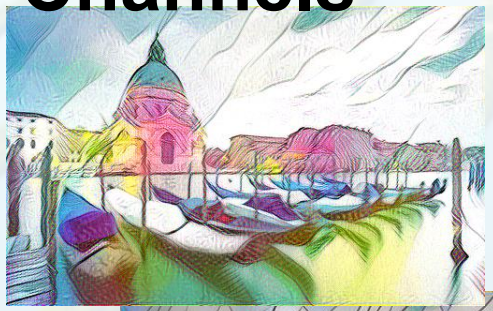
Multiple Output Channels

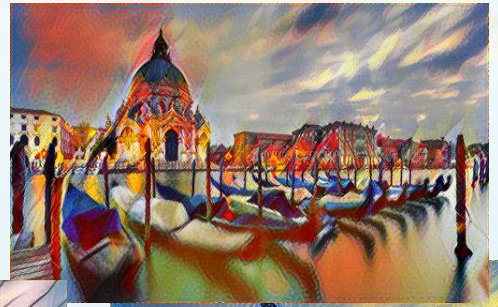


https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

aws

Multiple Output Channels



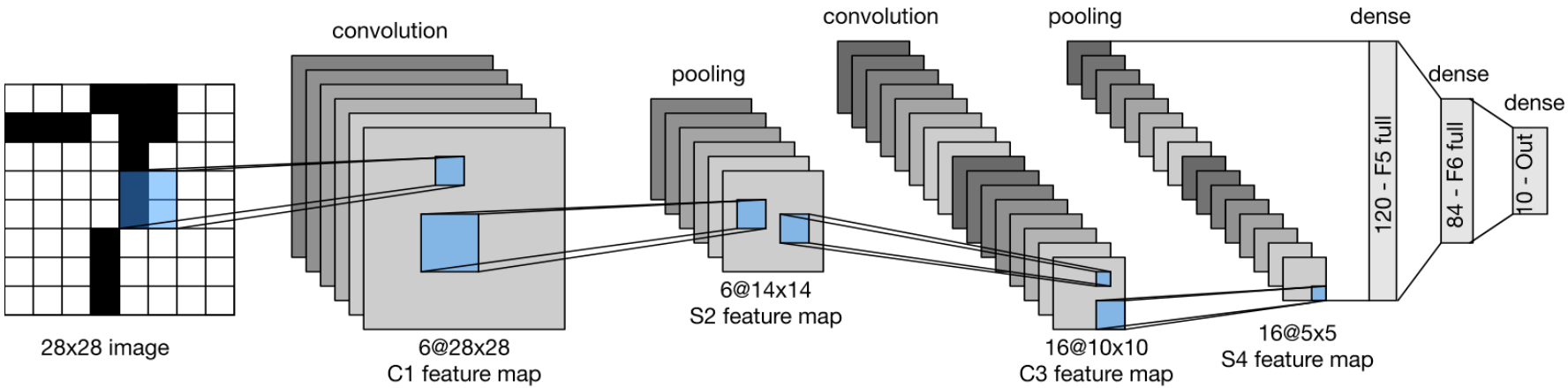


https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer/

aws

Convolutions Jupyter Notebook

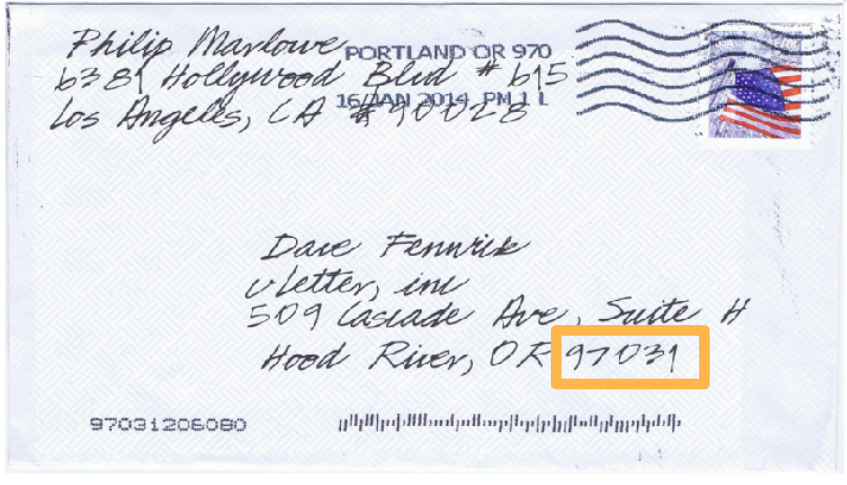
LeNet

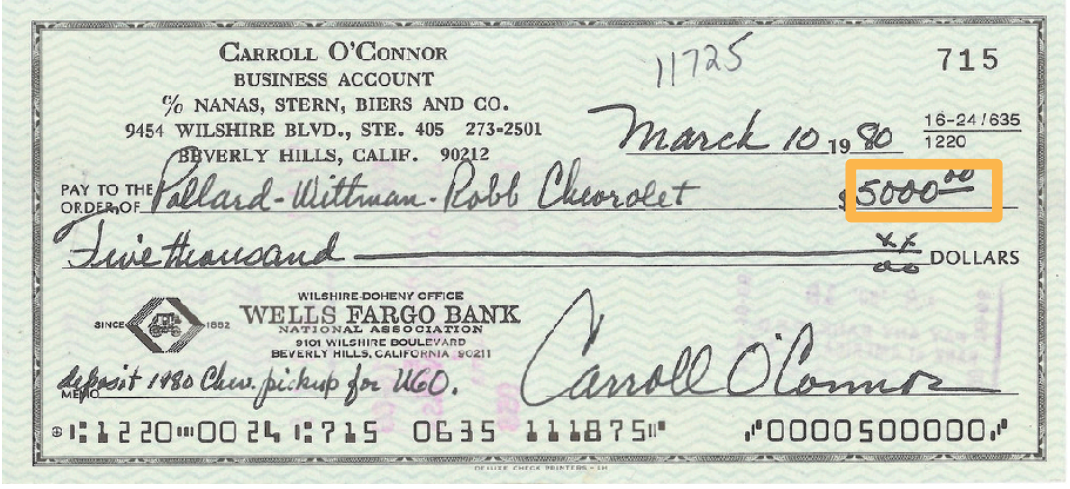


http://yann.lecun.com/exdb/lenet/

Why CNN in 1990s?

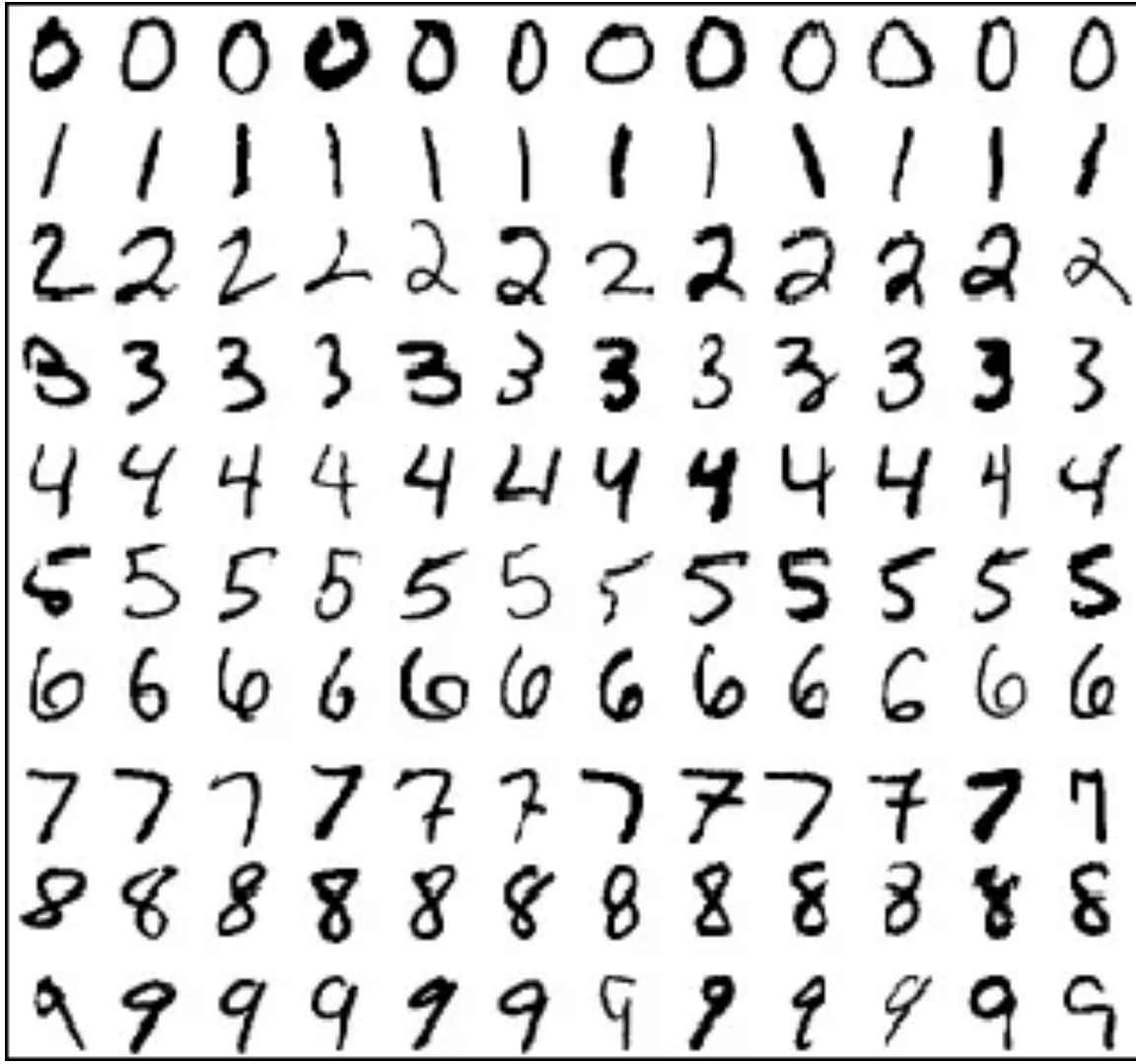
AT&T had a project in order to recognize handwritten characters for postal codes on letters and also recognize the dollar amounts on checks.

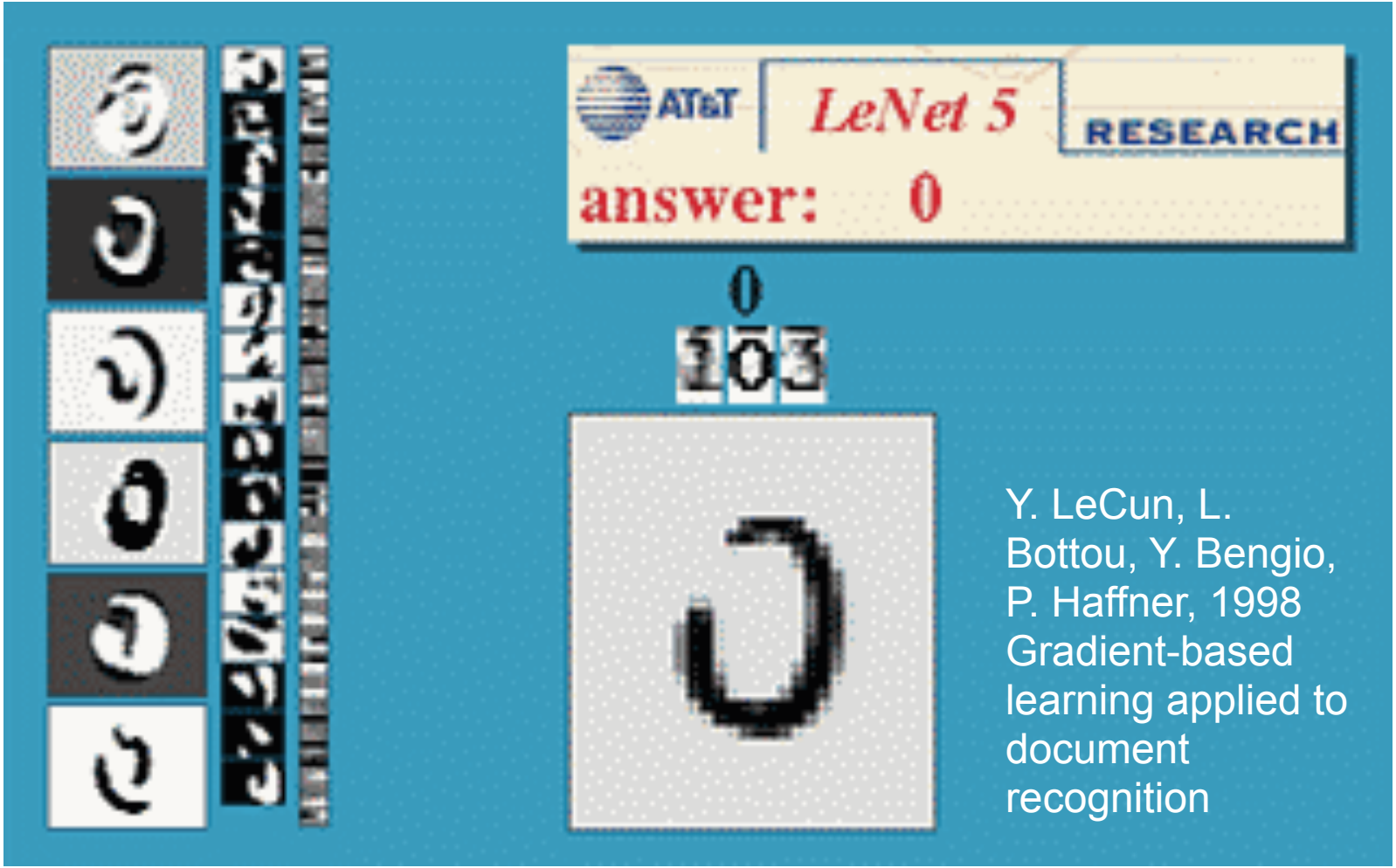




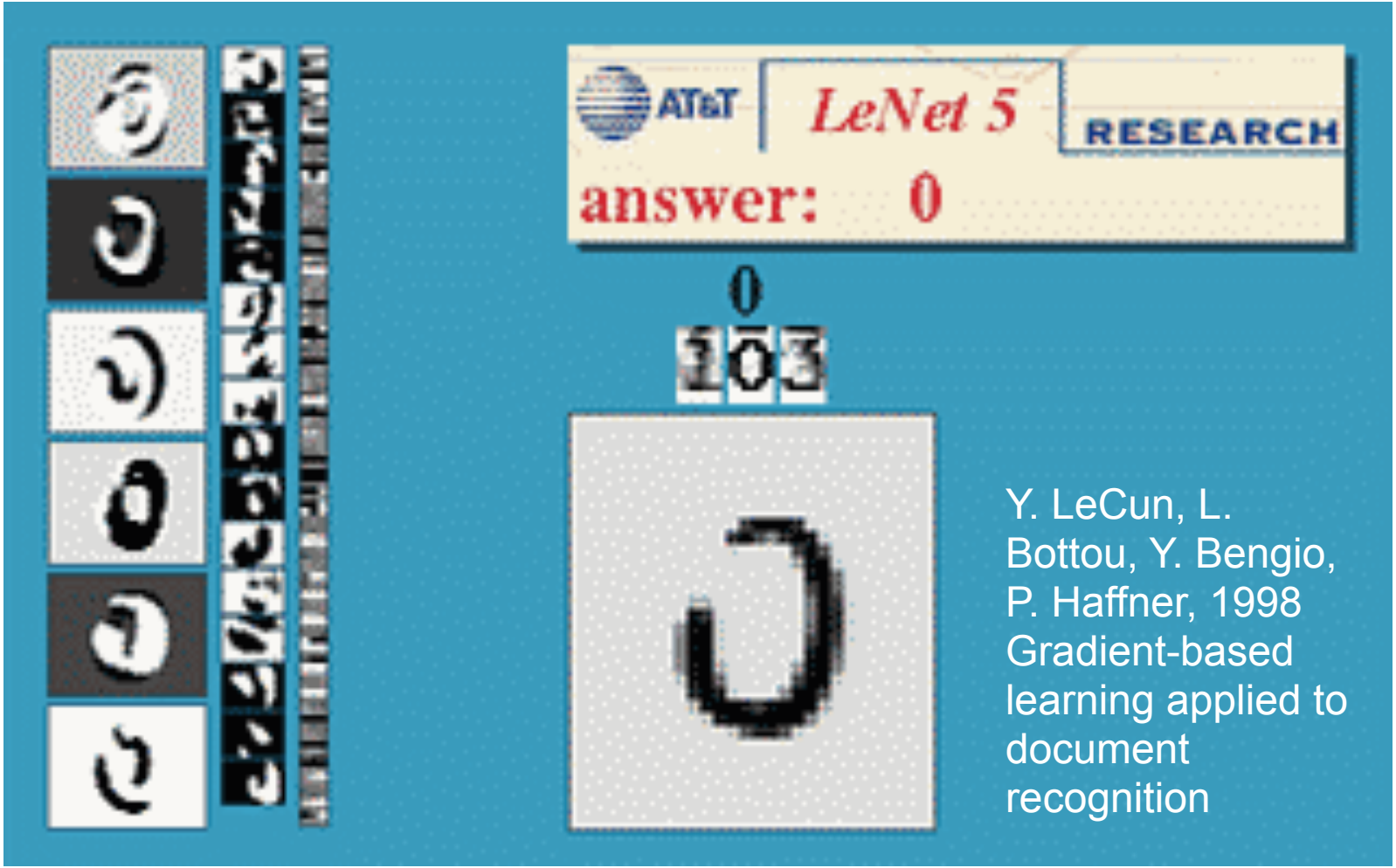
MNIST

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





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



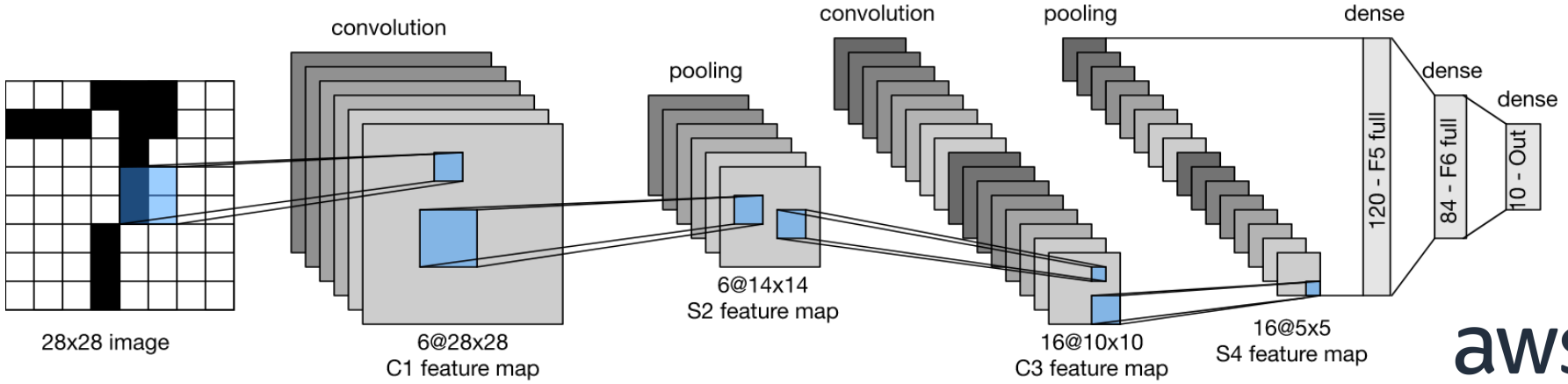
Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, 1998 Gradient-based learning applied to document recognition

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

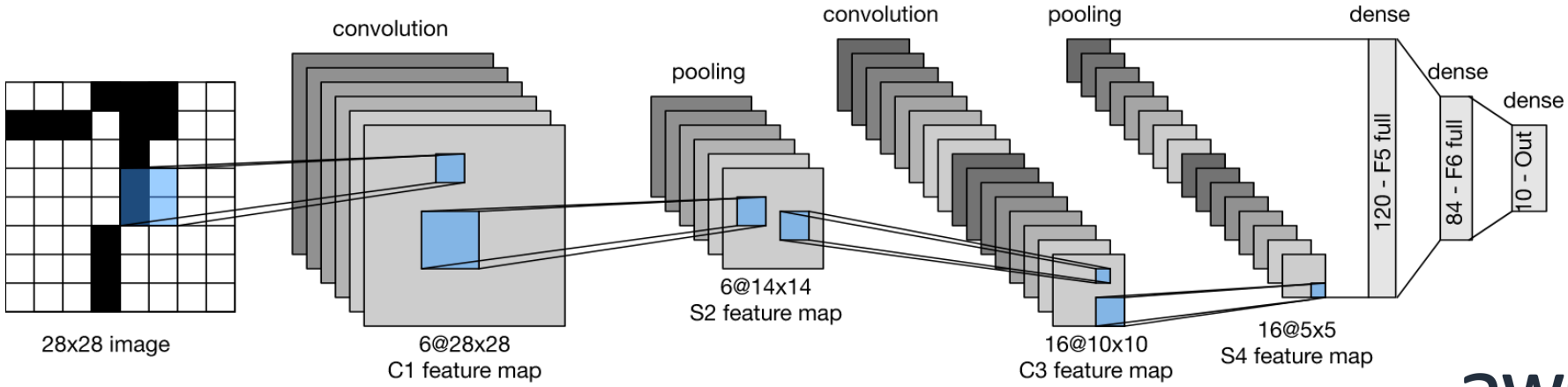


LeNet

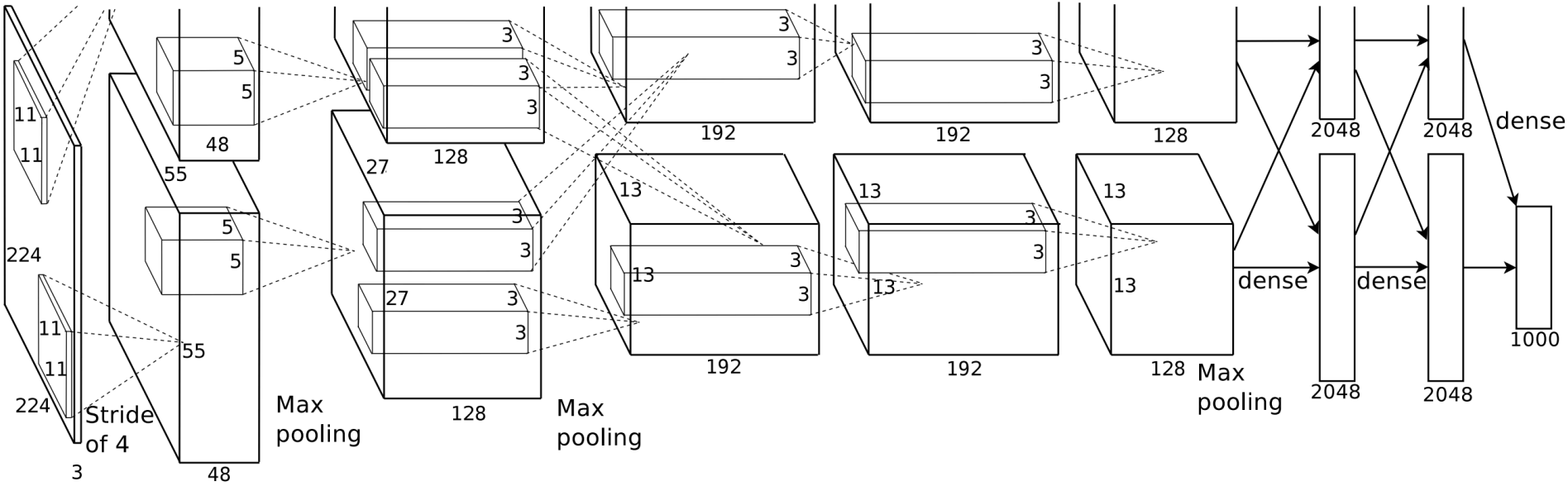
LeNet consists of two parts:

Part II. Fully-connected layers Block

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



AlexNet

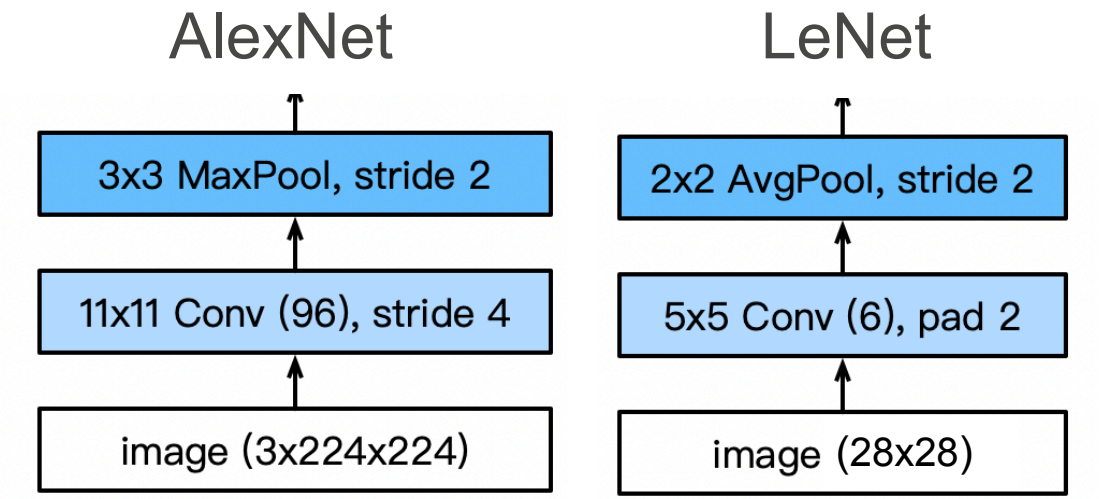

224
224
3
11
11
11
11
Stride
of 4
55
55
48
48
5
5
5
5
Max
pooling
27
27
128
128
3
3
3
3
3
3
Max
pooling
13
13
13
13
13
192
192
192
192
128
128
3
3
3
3
3
3
3
3
3
3
dense
dense
dense
2048
2048
2048
2048
1000

ImageNet Dataset

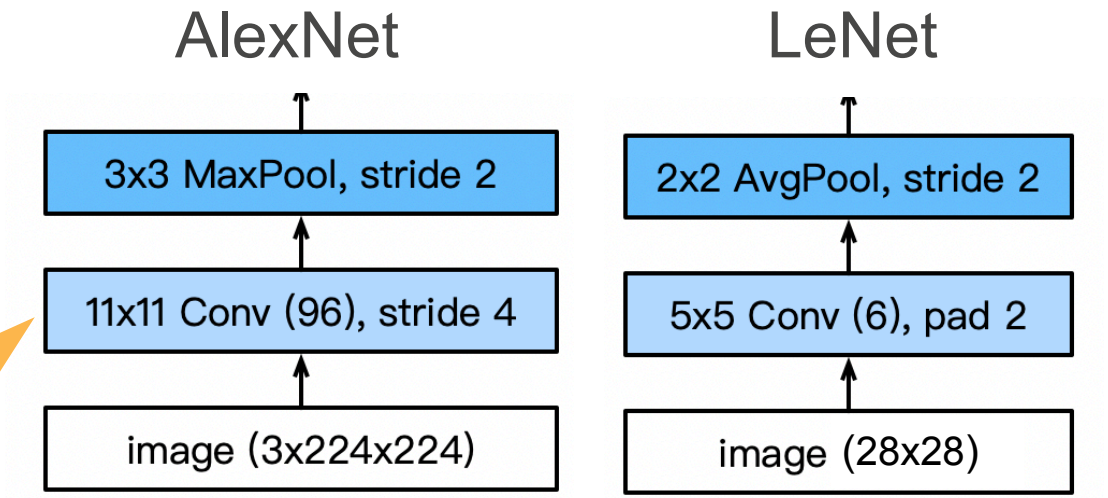
http://www.image-net.org/

- # of images: 14 M
- # of annotated images: 1.2 M
- Feature classes: 1000

AlexNet Architecture



AlexNet Architecture

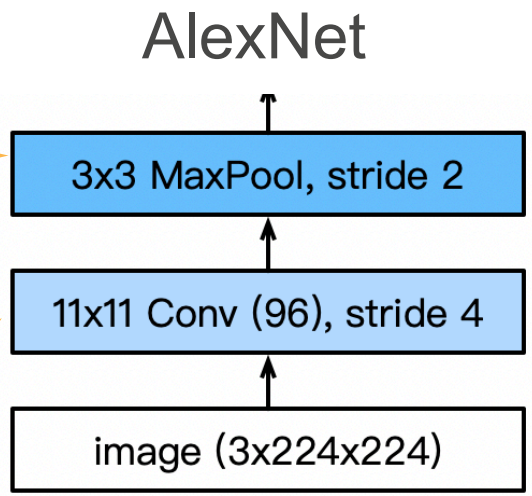


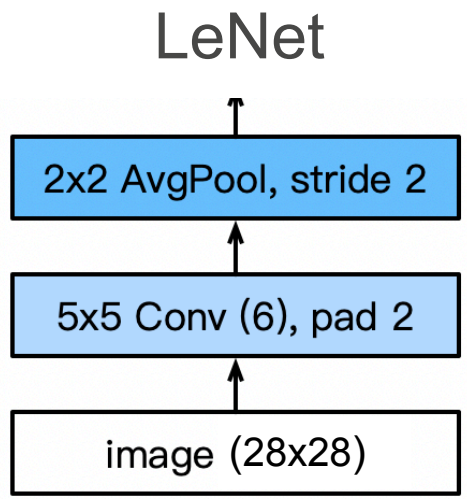
Larger kernel size, stride because of the increased image size, and more output channels

AlexNet Architecture

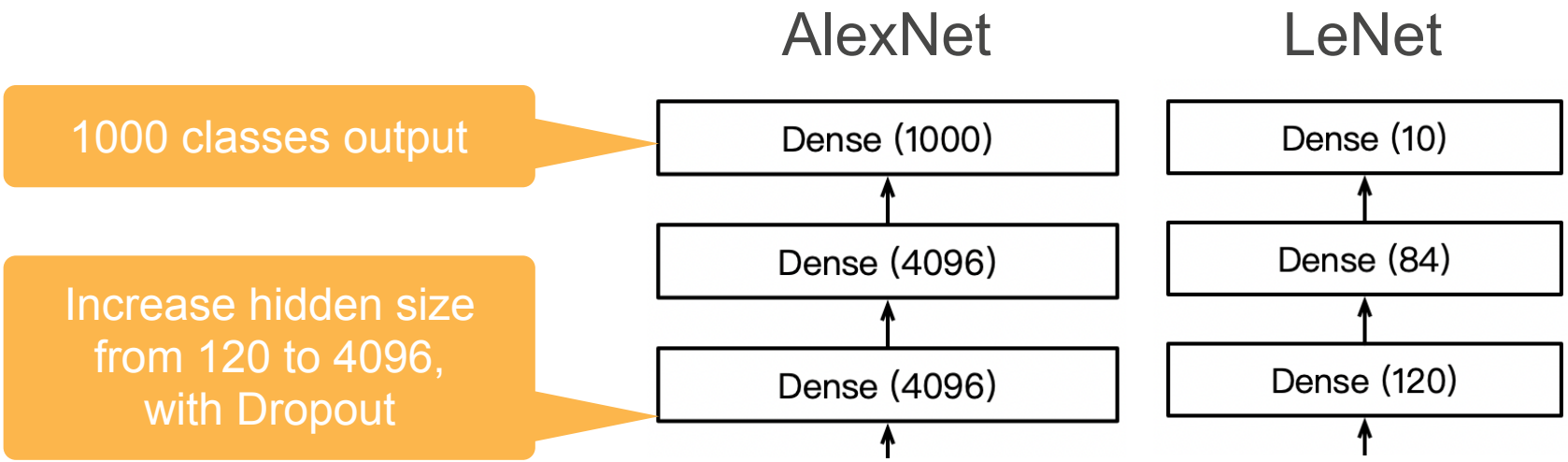
Larger pool size, change to **max pooling**

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





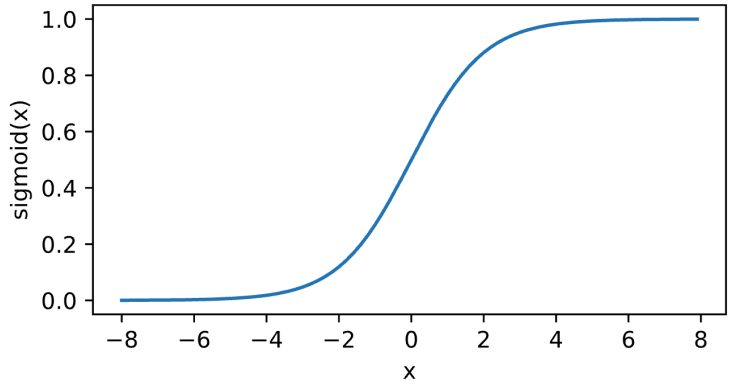
AlexNet Architecture



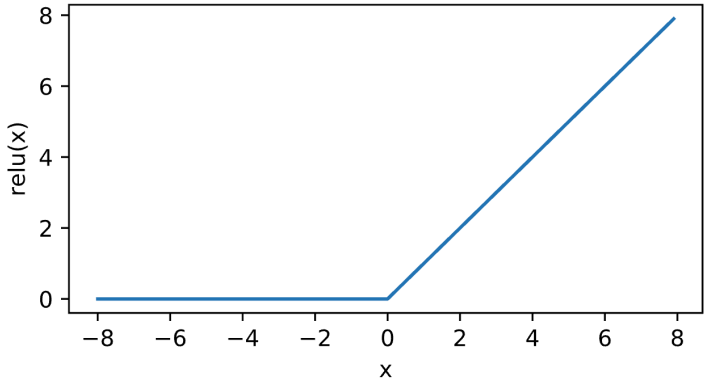
More Tricks

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

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



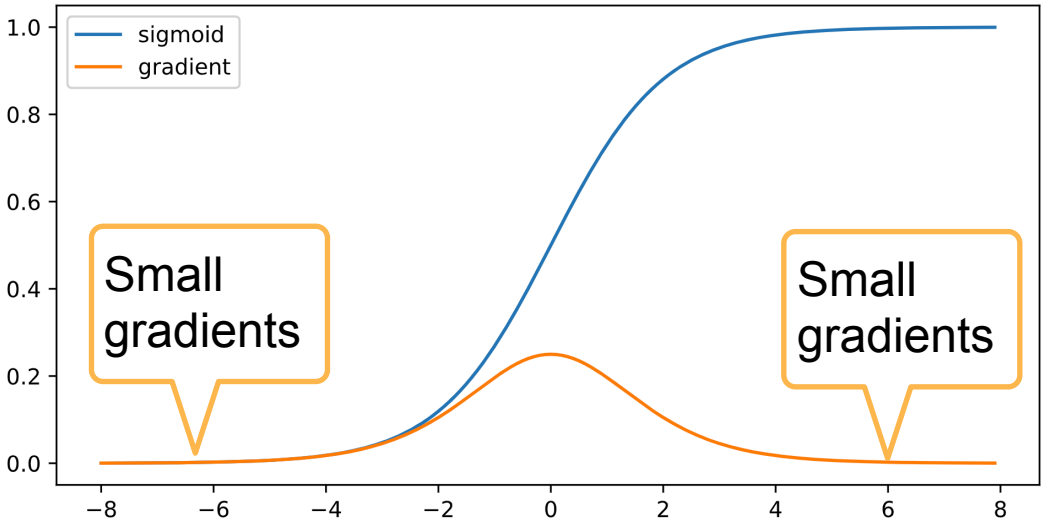
$$\text{ReLU}(x) = \max(x, 0)$$



Gradient Vanishing

- Use sigmoid as the activation function

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



Gradient Vanishing

- Use sigmoid as the activation function

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

- Elements $\prod_{i=t}^{d-1} \frac{\partial \mathbf{h}^{i+1}}{\partial \mathbf{h}^{i}} = \prod_{i=t}^{d-1} \text{diag}\left(\sigma'(\mathbf{W}^{i}\mathbf{h}^{i-1})\right)(W^{i})^{T}$ 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

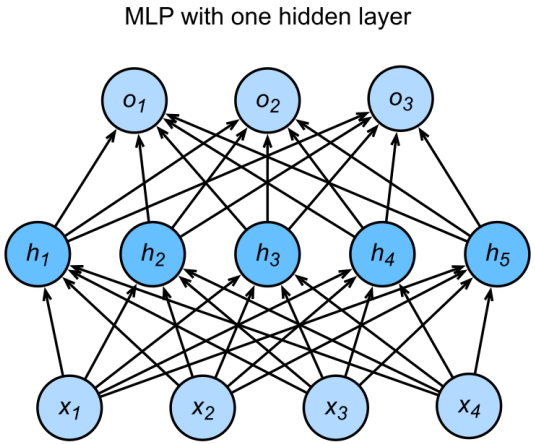
More Tricks

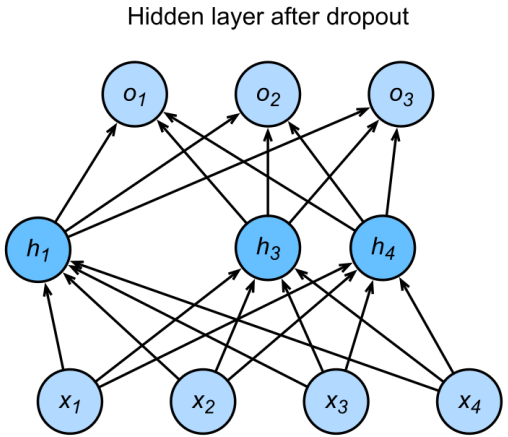
1. Change activation function from sigmoid to **ReLu** (no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers (better robustness / regularization)

Dropout

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

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$
$$\mathbf{o} = \mathbf{W}_2\mathbf{h}' + \mathbf{b}_2$$
$$\mathbf{y} = \text{softmax}(o)$$





Dropout - Inference

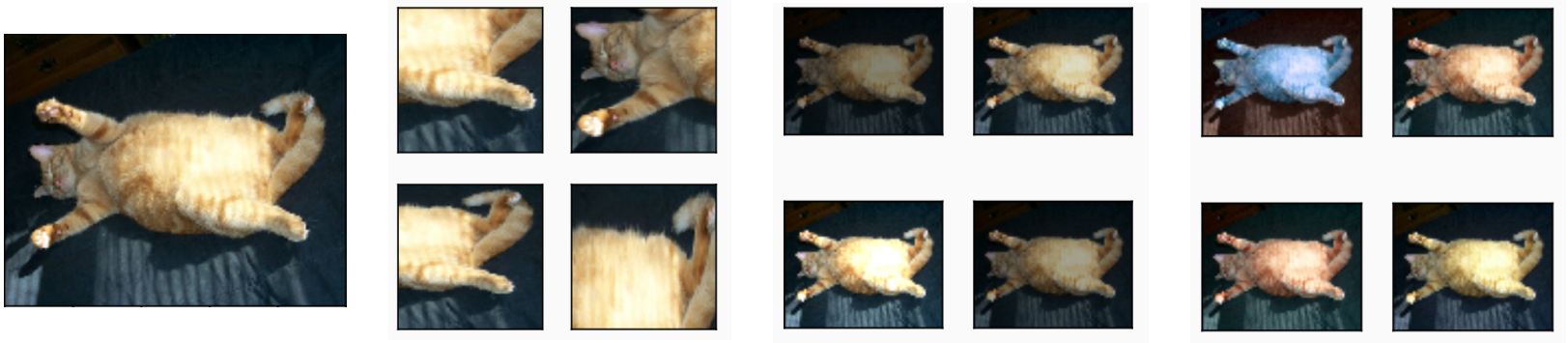
- Regularization is only used in training
- The dropout layer for inference is

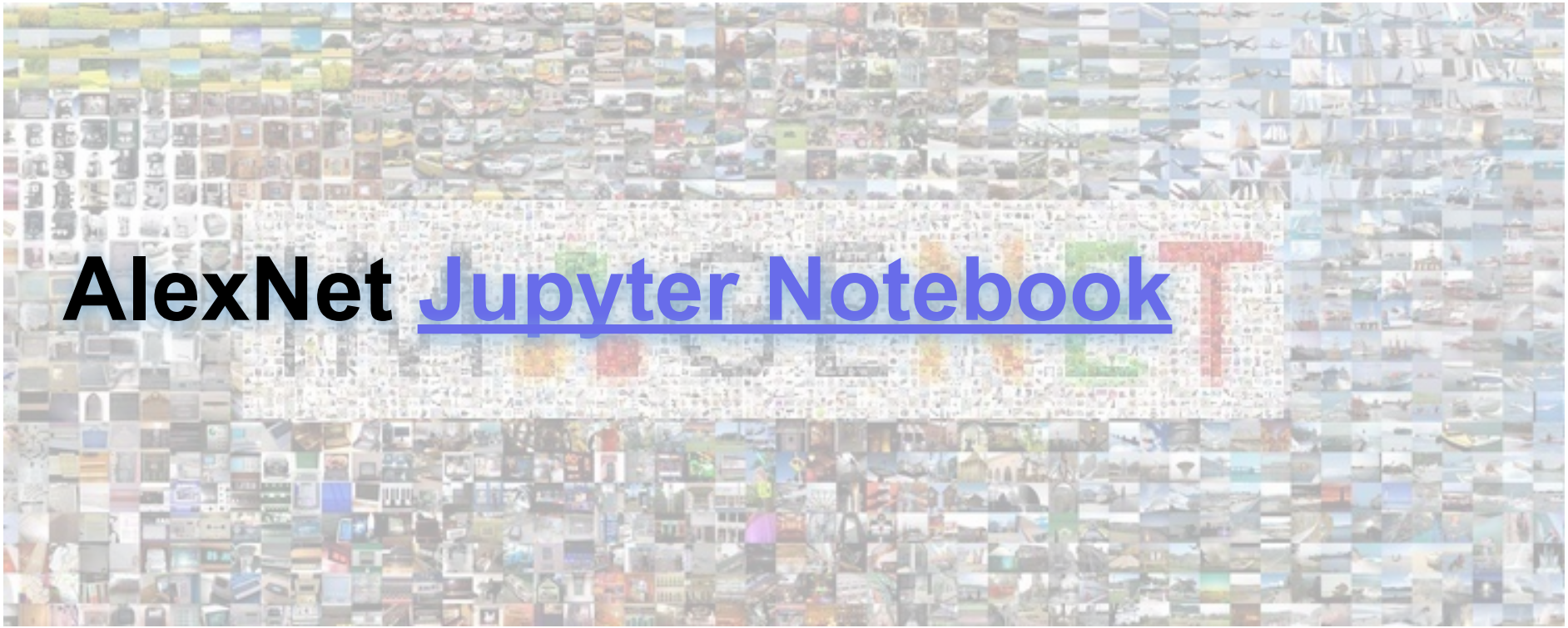
$$\mathbf{h}' = \text{dropout}(\mathbf{h})$$

  - Guarantee deterministic results

More Tricks

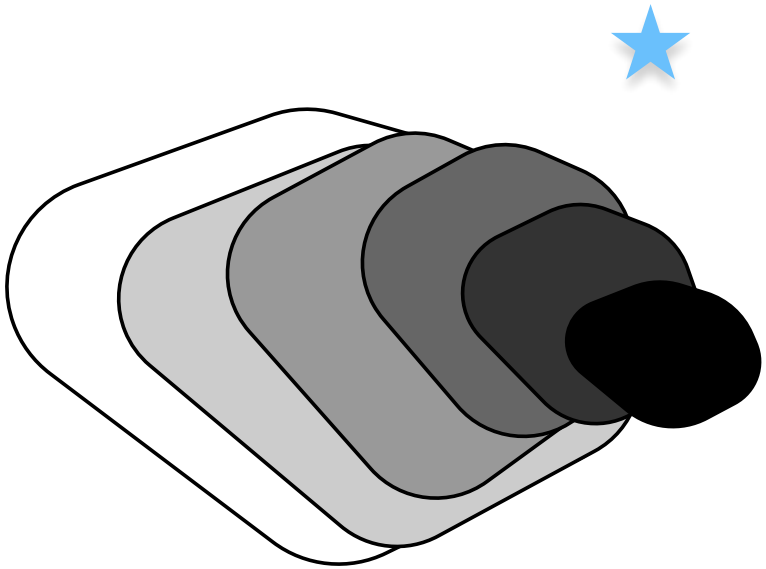
1. Change activation function from sigmoid to **ReLu** (no more vanishing gradient)
2. Add a **dropout** layer after two hidden dense layers (better robustness / regularization)
3. Data augmentation




AlexNet Jupyter Notebook

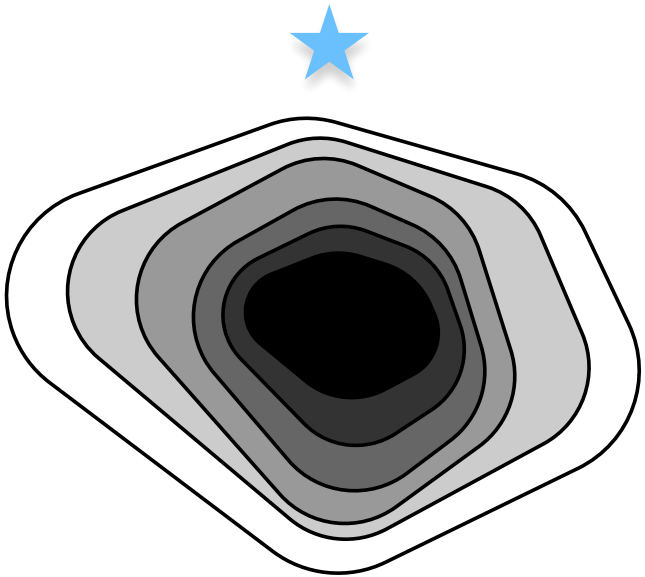
More Ideas

Does adding layers improve accuracy?



generic function classes

?



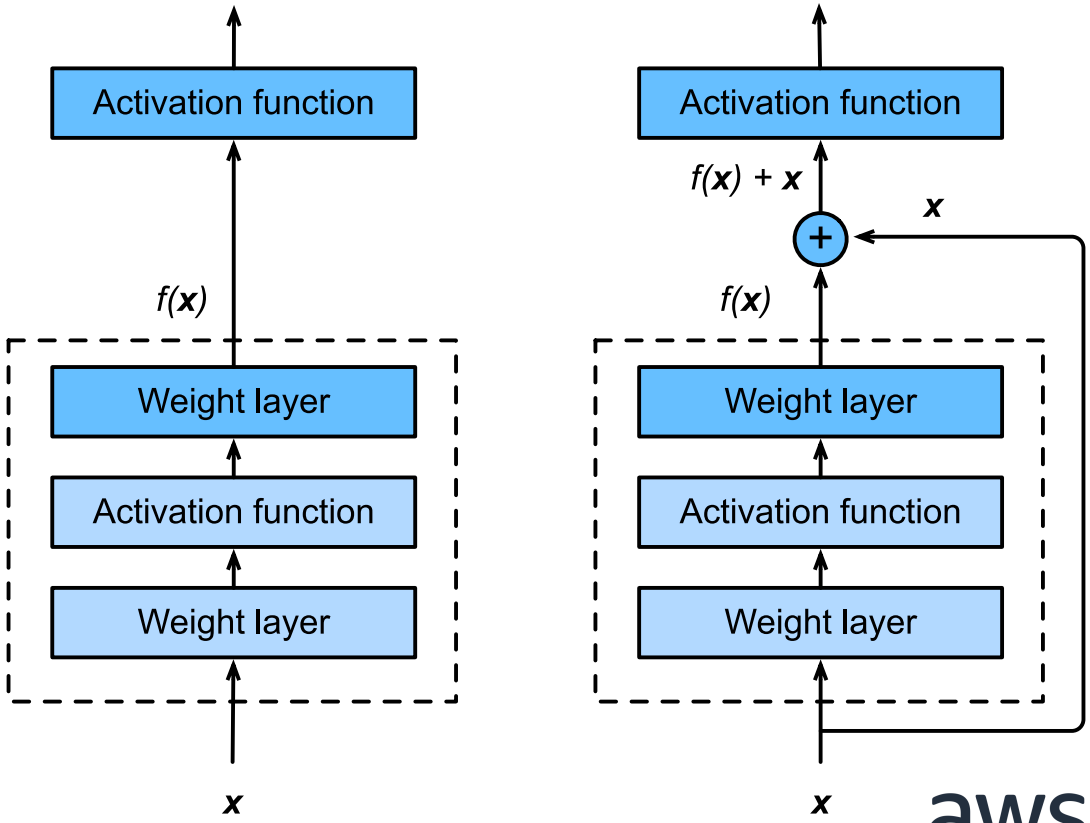
nested function classes

✔

Residual Networks

- Adding a layer **changes** function class
- ‘Taylor expansion' style parametrization

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



He et al., 2015

DenseNet (Huang et al., 2016)

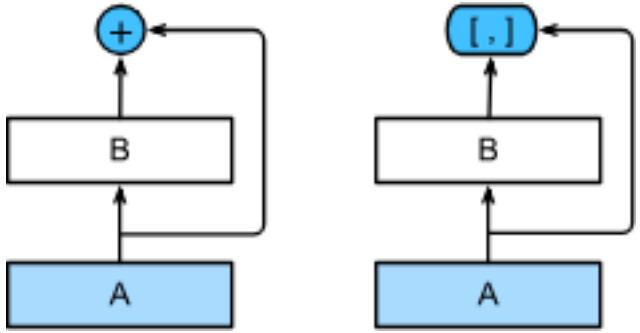
- ResNet combines x and f(x)
- DenseNet uses higher order 'Taylor series' expansion

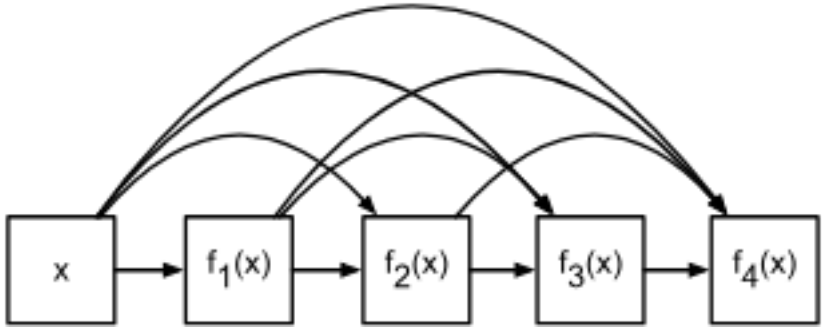
$$x_{i+1} = [x_i, f_i(x_i)]$$

$$x_1 = x$$

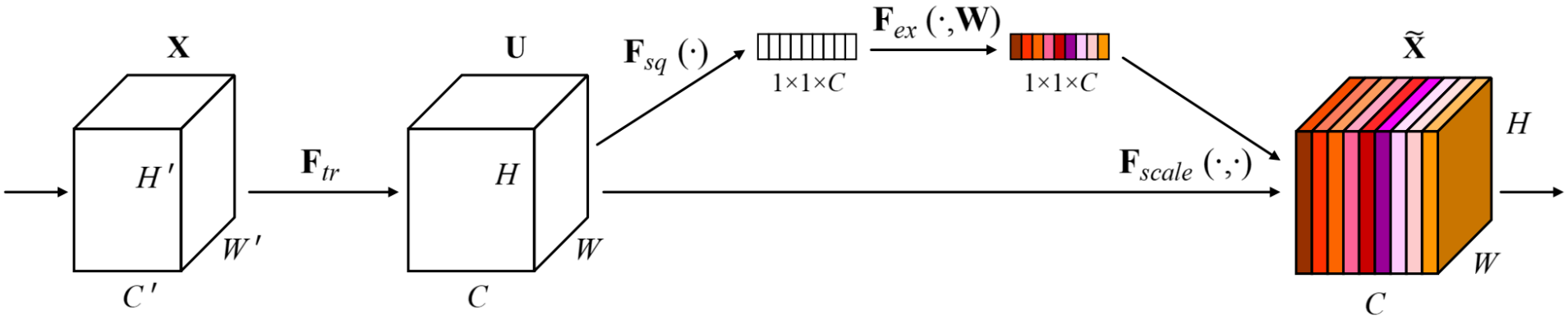
$$x_2 = [x, f_1(x)]$$

$$x_3 = [x, f_1(x), f_2([x, f_1(x)])]$$





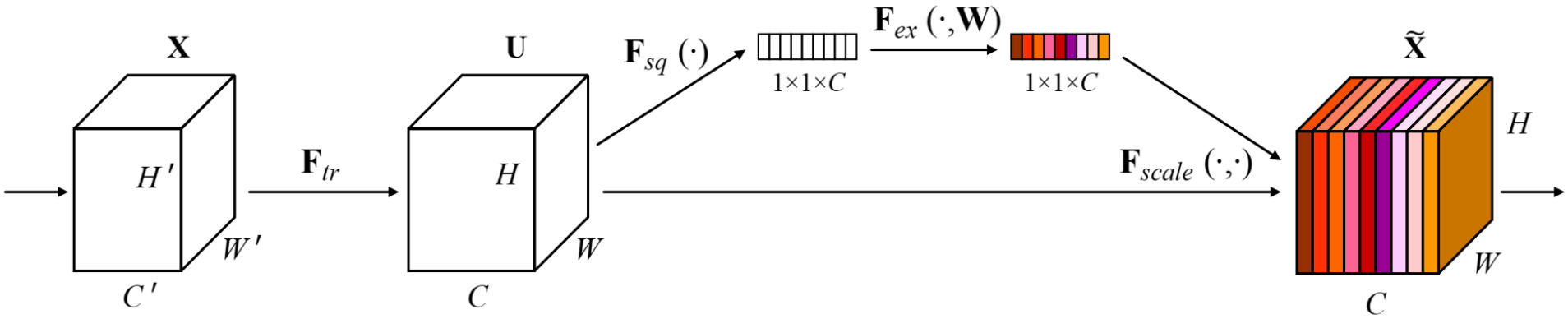
Squeeze-Excite Net (Hu et al., 2017)



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

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

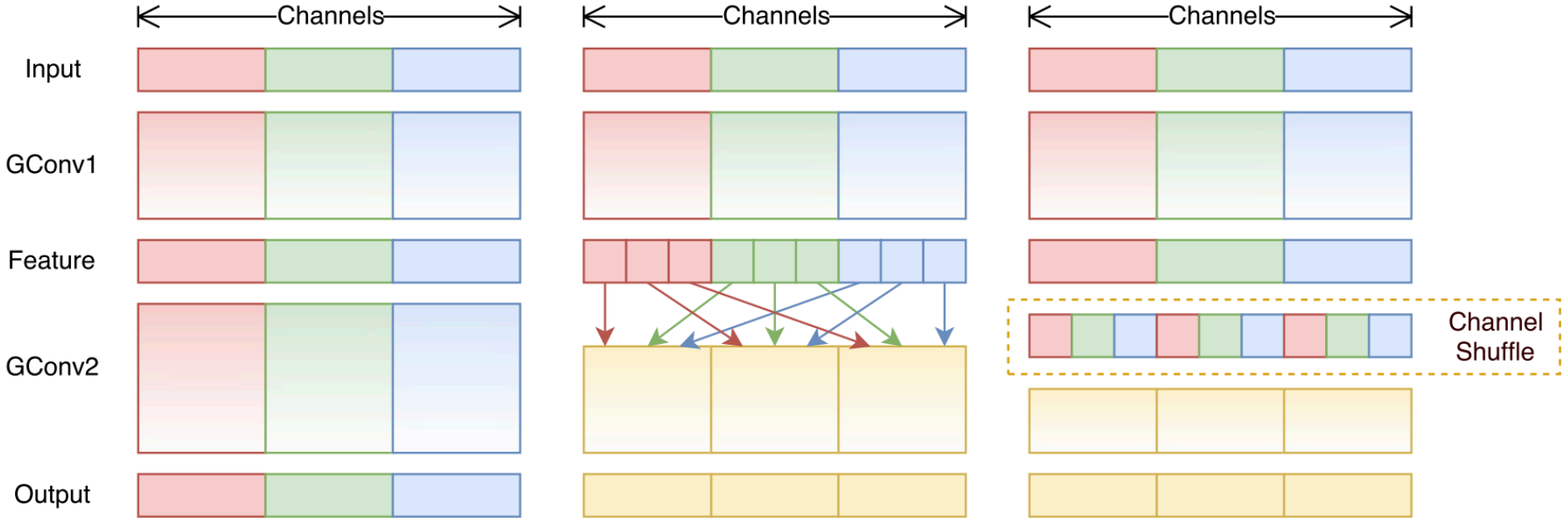
Squeeze-Excite Net (Hu et al., 2017)



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

aws

ShuffleNet (Zhang et al., 2018)



- break convolution into channels
- mix by grouping (very efficient for mobile)

More CNNs at D2L

Link