

Introduction to Deep Learning

11. Convolutional and Pooling Layers

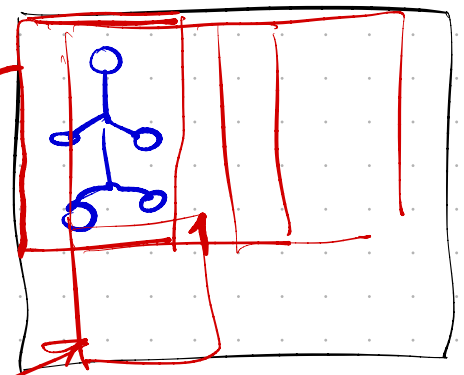
STAT 157, Spring 2019, UC Berkeley

Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

Key ideas for NN-for-images $\Rightarrow$ CNN

- recognize an object/shape/pattern (distinguish) $\Rightarrow$ computation on that frame
 $\Rightarrow$ Frame $\times$ kernel window K



Concept

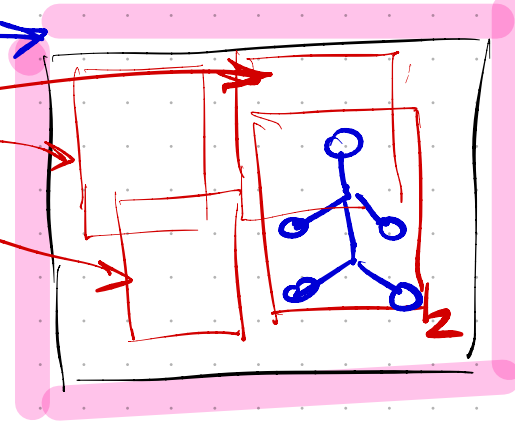
Convolution

$$\text{Frame}_1 \times K \approx \text{Frame}_2 \times K$$

- Mechanics: padding & stride

- edges/filling/size

- movement/sliding of the kernel window



- pooling: simplification of data/computation

- take avg $\begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix} = \boxed{3}$

$$\text{dim } 2 \times 2 \rightarrow \text{dim } 1 \times 1$$

- take max $\begin{bmatrix} 1 & 2 \\ 3 & 6 \end{bmatrix} = \boxed{6}$

- Mechanics: channels

- multichannel: R, G, B input pixels

- output: d channels output = input for next layer

overall design = multilayer-NNet

Classifying Dogs and Cats in Images

- Use a good camera
- RGB image has 36M elements
- The model size of a single hidden layer MLP with a 100 hidden size is 3.6 Billion parameters
- Exceeds the population of dogs and cats on earth (900M dogs + 600M cats)



Dual
12MP
wide-angle and
telephoto cameras



Flashback - Network with one hidden layer

RECAP MLP = multilayer perceptrons
- weights updated by BACKPROP
- activation $f(\text{net})$ sigmoid, ReLU, tanh
- error at top

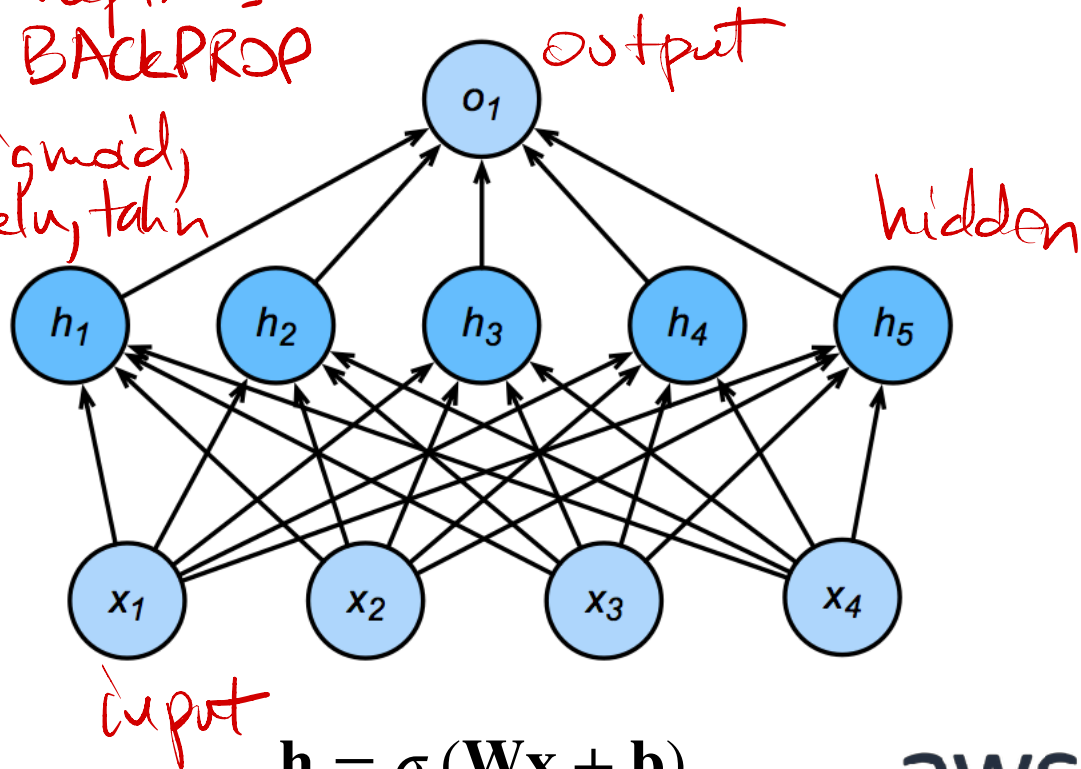
100 neurons

Hidden layer

3.6B parameters = 14GB

36M features

Input layer



Where is
Waldo?



~~Issues~~ Two Principles

- Translation Invariance
- Locality



Rethinking Dense Layers

- Reshape inputs and output into matrix (width, height)
- Reshape weights 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

cell
[i,j]
in hidden layer

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

a, b = indices in the frame

$V = W$ window (kernel)

X = frame [a,b indices]

V = window, not depend on cell [i,j]

- A shift in x also leads to a shift in h
- v should not depend on (i,j) . Fix via $v_{i,j,a,b} = v_{a,b}$

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

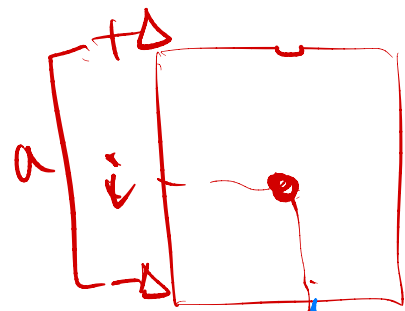
dot product
 $V \cdot X$

That's a ~~2-D convolution~~
cross-correlation

Idea #2 - Locality

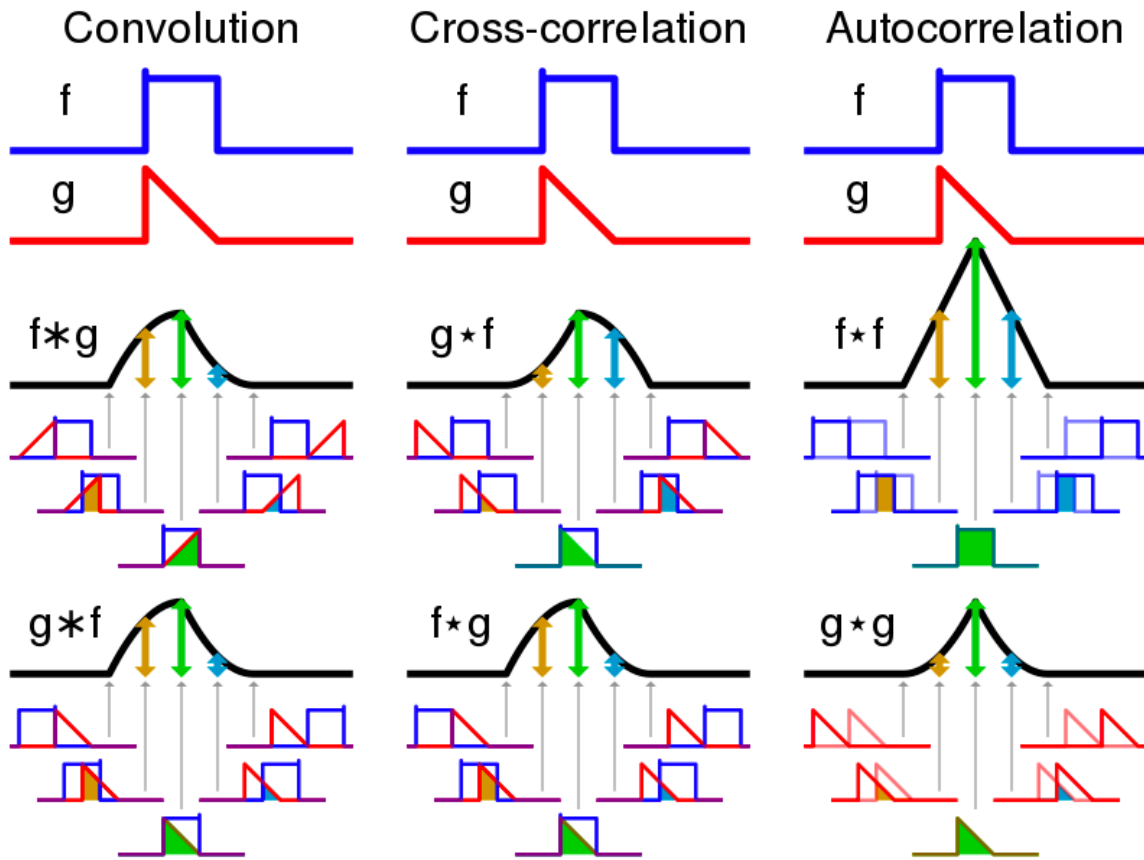
$$h_{i,j} = \sum_{a,b} v_{a,b} x_{i+a,j+b}$$

- 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 $v_{a,b} = 0$

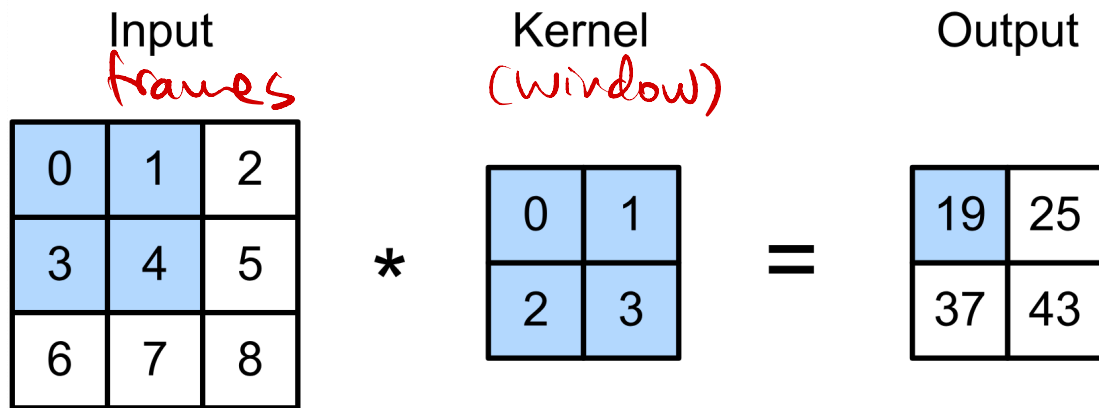


$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

Convolution



2-D Cross Correlation

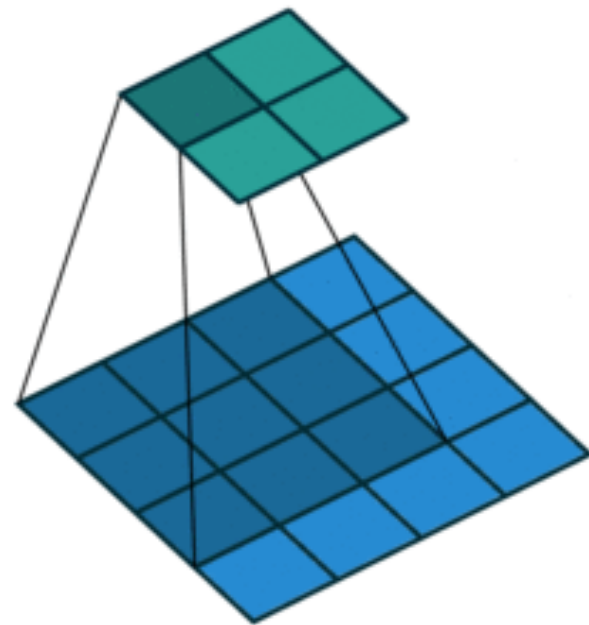


$$0 \times 0 + 1 \times 1 + 3 \times 2 + 4 \times 3 = 19,$$

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

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

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



(vdumoulin@ Github)

2-D Cross Correlation

Input

0	1	2
3	4	5
6	7	8

Kernel

0	1
2	3

*

=

Output

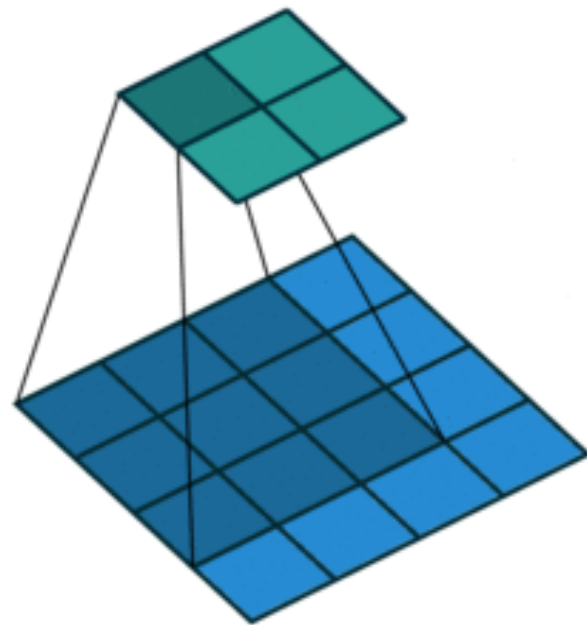
19	25
37	43

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

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

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

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



(vdumoulin@ Github)

2-D Convolution Layer

input

0	1	2
3	4	5
6	7	8

X

*

0	1
2	3

W

=

output

19	25
37	43

Y

$k_h = k_w = w$

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

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

- $\mathbf{W}$ and b are learnable parameters

Examples

sliding kernels

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

step=1

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

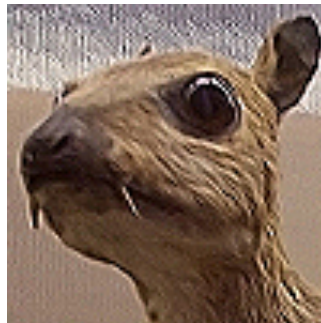
$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

output

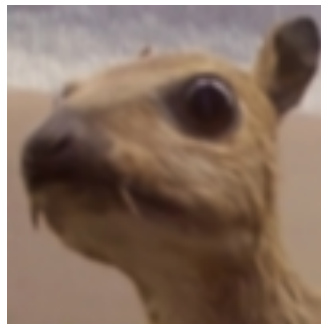
Edge Detection



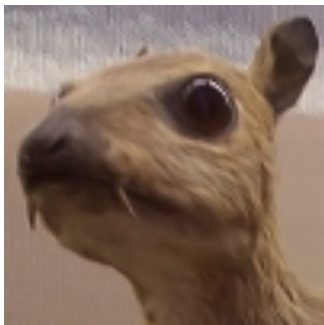
Sharpen



Gaussian Blur



X



(wikipedia)

Examples



(Rob Fergus)



Cross Correlation vs Convolution

linear algebra *math functions*

- 2-D Cross Correlation

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{a,b} x_{i+a,j+b}$$

- 2-D Convolution

$$y_{i,j} = \sum_{a=1}^h \sum_{b=1}^w w_{-a,-b} x_{i+a,j+b}$$

- No difference in practice during to symmetry

1-D and 3-D Cross Correlations

- 1-D

$$y_i = \sum_{a=1}^h w_a x_{i+a}$$

- Text
- Voice
- Time series

- 3-D

$$y_{i,j,k} = \sum_{a=1}^h \sum_{b=1}^w \sum_{c=1}^d w_{a,b,c} x_{i+a,j+b,k+c}$$

- Video
- Medical images

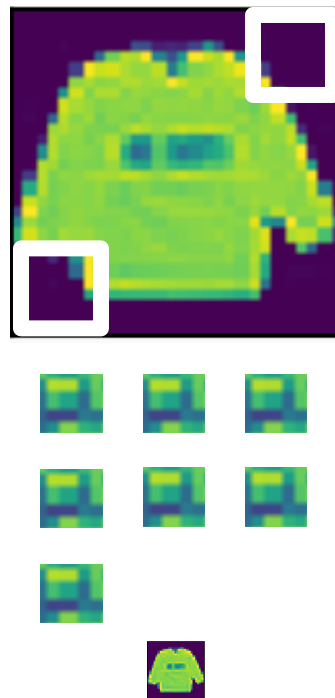
A photograph of a city street with four identical men in the foreground, each in a different pose as if captured at different times. They are wearing dark jackets, blue jeans, and brown shoes, and have their arms outstretched and mouths open in a surprised expression. The background shows a busy street with cars, buildings, and trees. The text "Padding and Stride" is overlaid in the center.

Padding and Stride

Padding

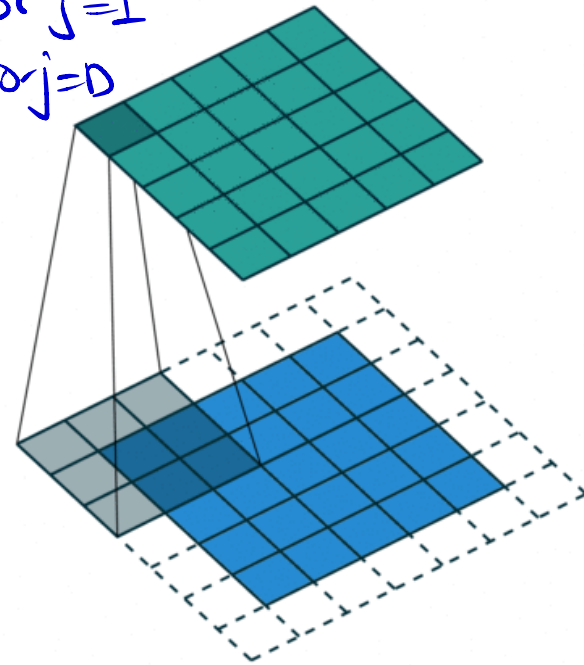
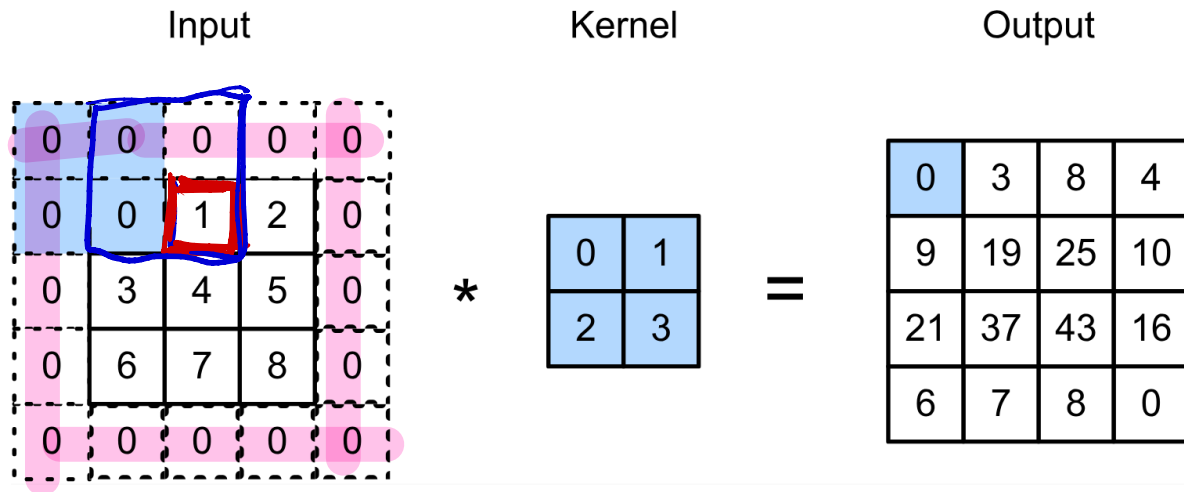
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
 - 28 x 28 output with 1 layer
 - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
 - Shape reduces from $n_h \times n_w$ to

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



Padding *to be able to compute h_{ij} $i=1$, or $j=1$
 — image size standardization or $i=D$, or $j=D$*

Padding adds rows/columns around input



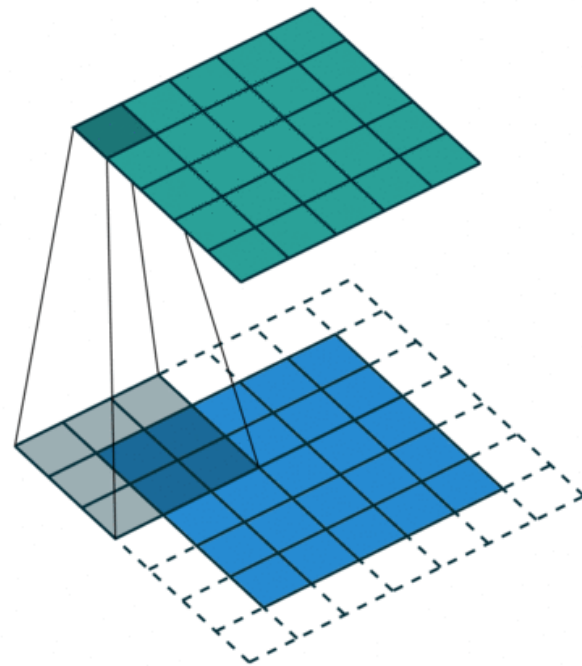
$$0 \times 0 + 0 \times 1 + 0 \times 2 + 0 \times 3 = 0$$

Padding

Padding adds rows/columns around input

Input					Kernel		Output			
0	0	0	0	0	*	=	0	3	8	4
0	0	1	2	0			9	19	25	10
0	3	4	5	0			21	37	43	16
0	6	7	8	0			6	7	8	0
0	0	0	0	0						

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



Padding

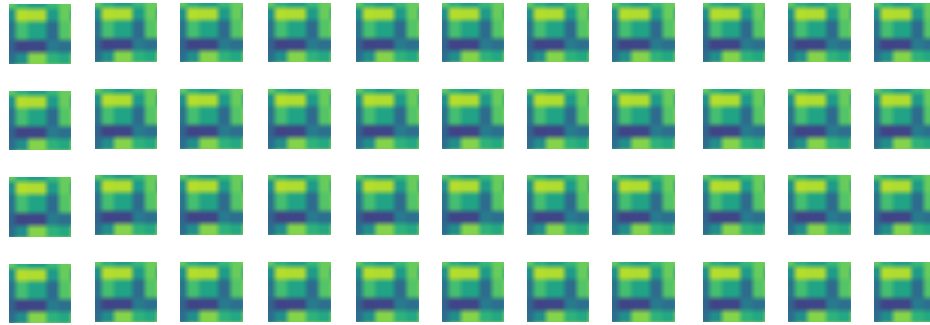
- Padding p_h rows and p_w columns, output shape will be

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

- 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

Stride : step size (default is 1)

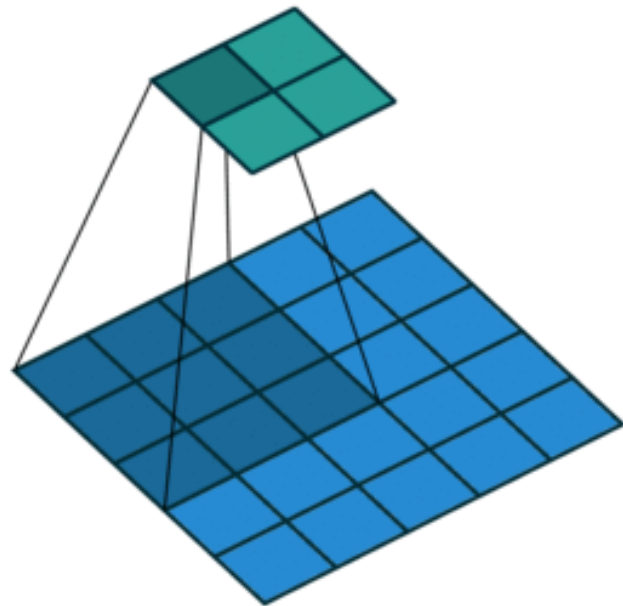
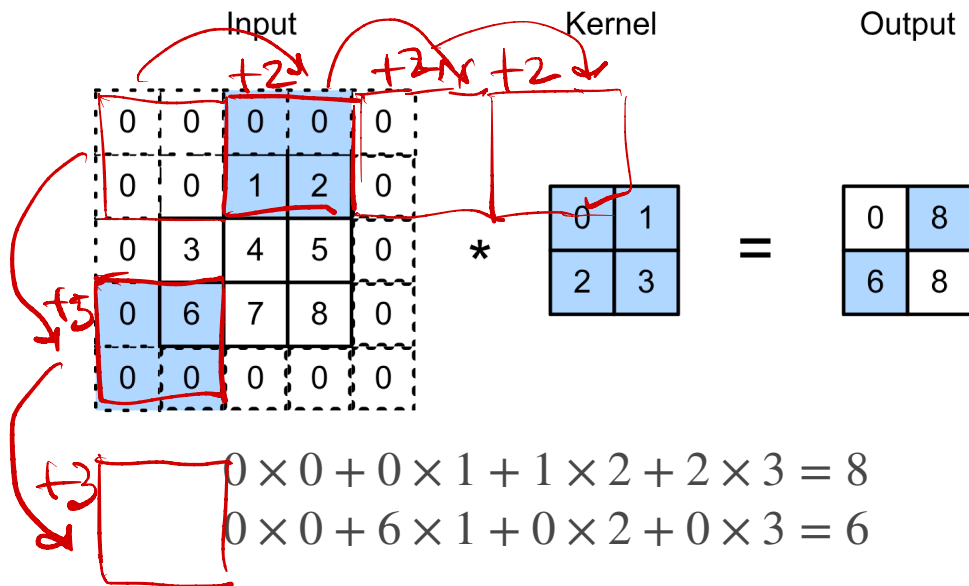
- Padding reduces shape linearly with #layers
 - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
 - Requires a large amount of computation



Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width



Stride

- Stride is the #rows/#columns per slide

Strides of 3 and 2 for height and width

Diagram illustrating a 2D convolution operation:

- Input:** A 5x5 grid with a 3x3 kernel area highlighted in blue. The input values are:

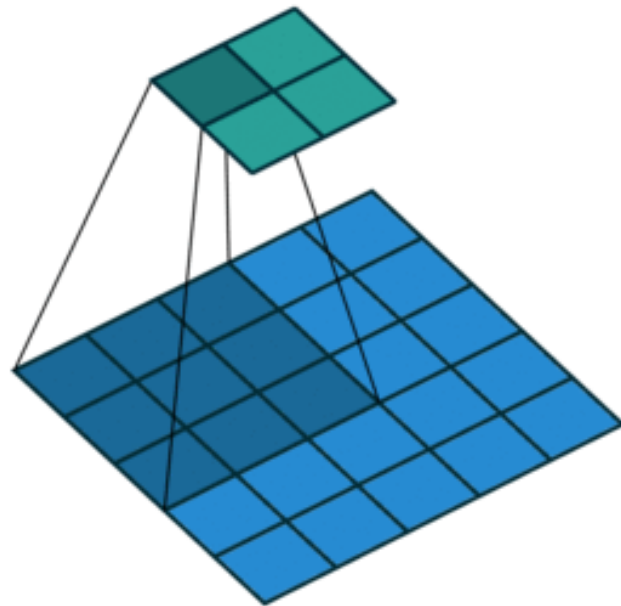
0	0	0	0	0
0	0	1	2	0
0	3	4	5	0
0	6	7	8	0
0	0	0	0	0
- Kernel:** A 2x2 grid with values:

0	1
2	3
- Output:** A 2x2 grid with values:

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

- 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

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

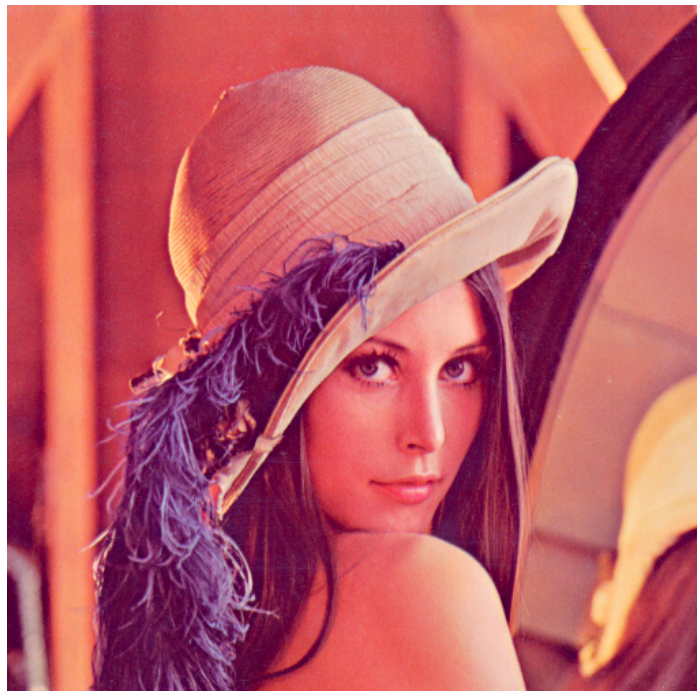
An aerial photograph showing a vast, organized agricultural system. The landscape is composed of numerous long, parallel, rectangular channels filled with vibrant green plants, likely a type of aquatic or semi-aquatic crop. These channels are separated by narrow, light-colored paths or ditches. The entire system is set within a body of water that has a deep blue hue. The perspective is from a high angle, looking down at the channels as they recede into the distance, creating a strong sense of depth and repetition. The lighting is bright, casting soft shadows and highlighting the textures of the plants and the water.

Multiple Input and Output Channels

Multiple Input Channels

input = 3 channels RGB index "c"
output = D outputs $y_1 \dots y_D$ index "d"

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



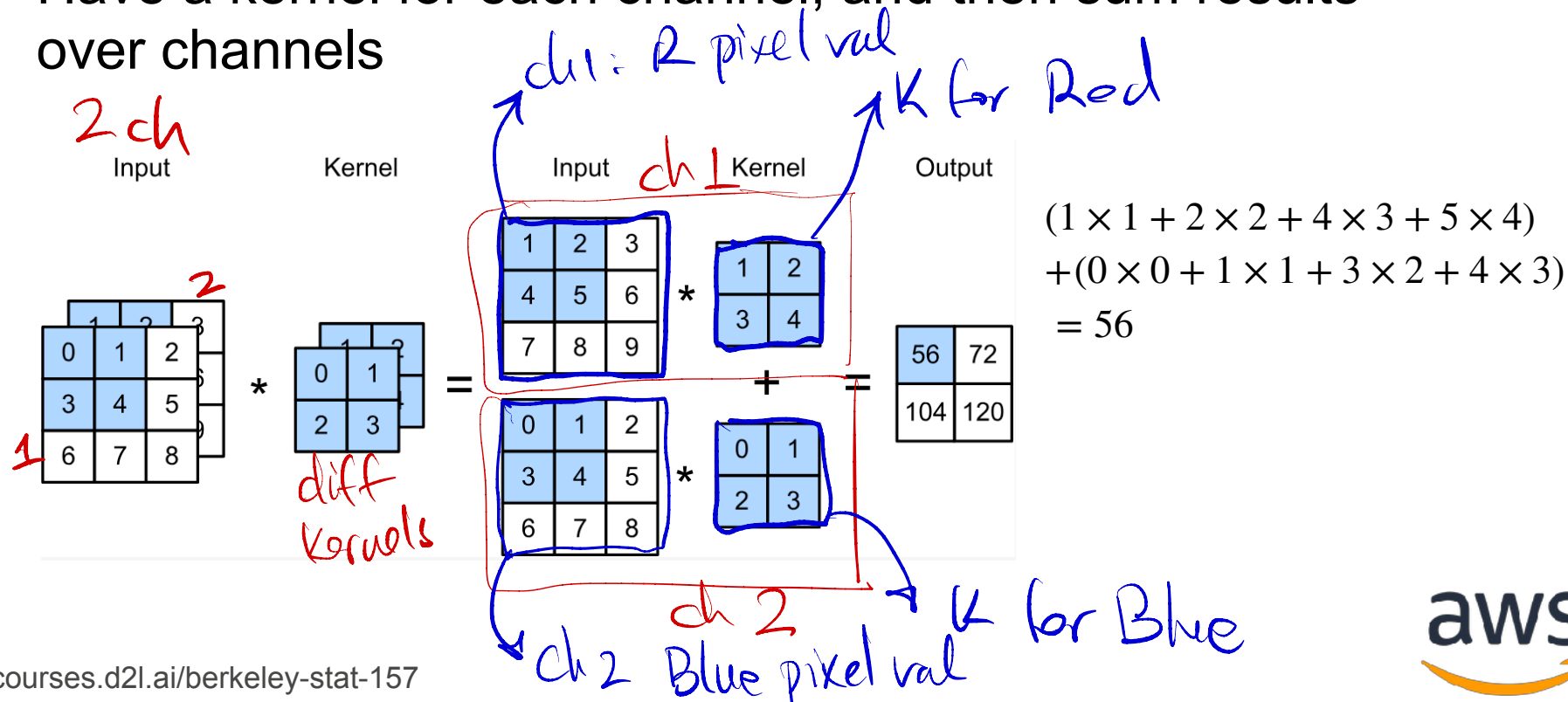
Multiple Input Channels

- 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



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 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 $i = 1, \dots, c_o$

Multiple Input/Output Channels

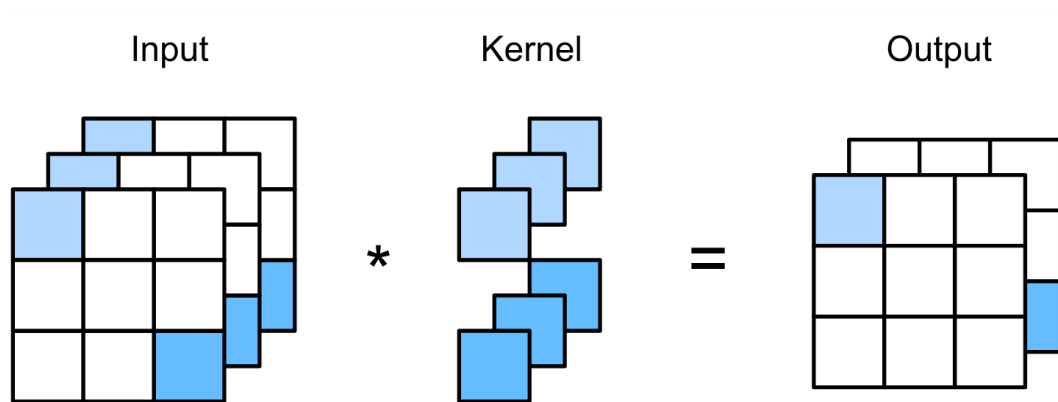
- Each output channel may recognize a particular pattern



- Input channels kernels recognize and combines patterns in inputs

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.

2-D Convolution Layer Summary

- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Bias $\mathbf{B} : c_o \times c_i$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$
- Complexity (number of floating point operations FLOP)

$$c_i = c_o = 100$$

$$k_h = k_w = 5$$

$$m_h = m_w = 64$$

$$O(c_i c_o k_h k_w m_h m_w)$$

1GFLOP

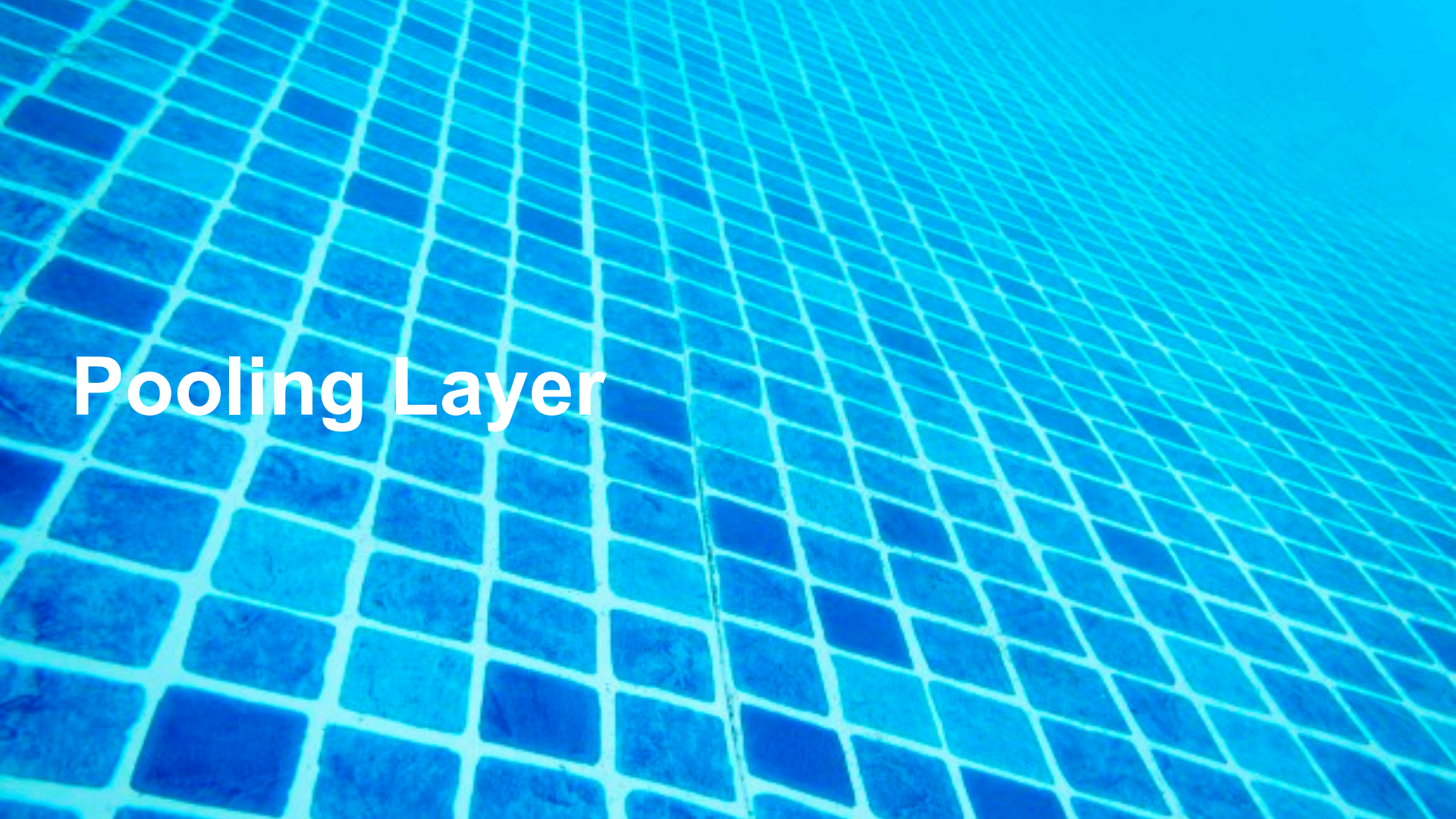
- 10 layers, 1M examples: 10PF
(CPU: 0.15 TF = 18h, GPU: 12 TF = 14min)

$$H[i,j,d] = \sum_a \sum_b \sum_c W_{abcd} X_{i+a, j+b, c}$$

$a = -\Delta, b = -\Delta$

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

c = input ch
 W_{abcd}
 $X_{i+a, j+b, c}$
 d = out ch
 a, b = window slider indices
 i, j = pixel position in image

The background of the slide is a blue grid pattern that recedes into the distance, creating a sense of depth. The grid is composed of squares that become smaller as they move towards the top right corner.

Pooling Layer

Pooling *EASY* *"layer"*

- Convolution is sensitive to position
 - Detect vertical edges

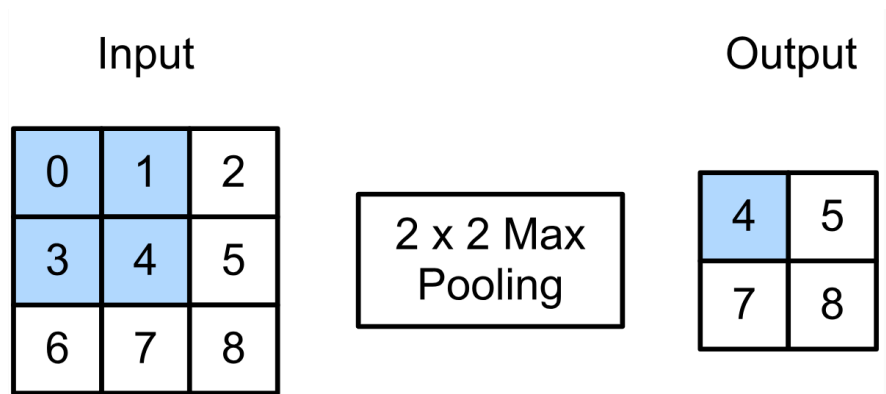
$$X \begin{bmatrix} 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \\ 1. & 1. & 0. & 0. & 0. \end{bmatrix} Y \begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 1. & 0. & 0. \end{bmatrix}$$

0 output with
1 pixel shift

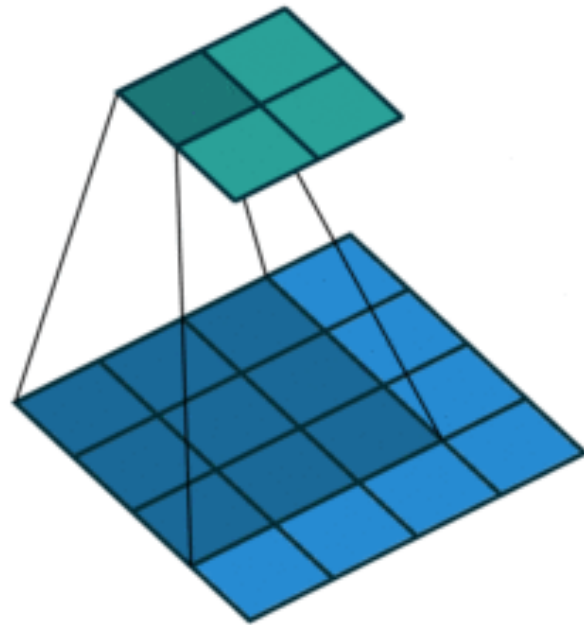
- We need some degree of invariance to translation
 - Lighting, object positions, scales, appearance vary among images

2-D Max Pooling

- Returns the maximal value in the sliding window



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

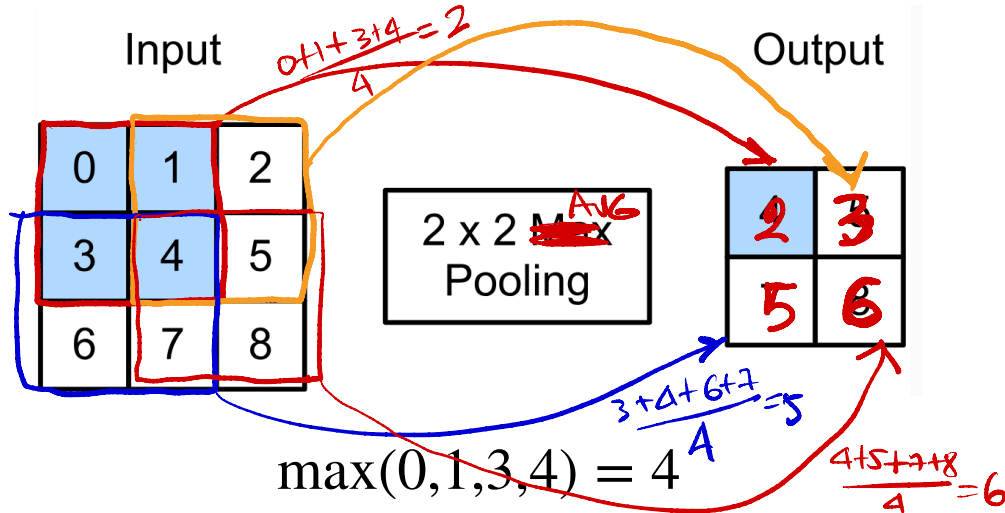
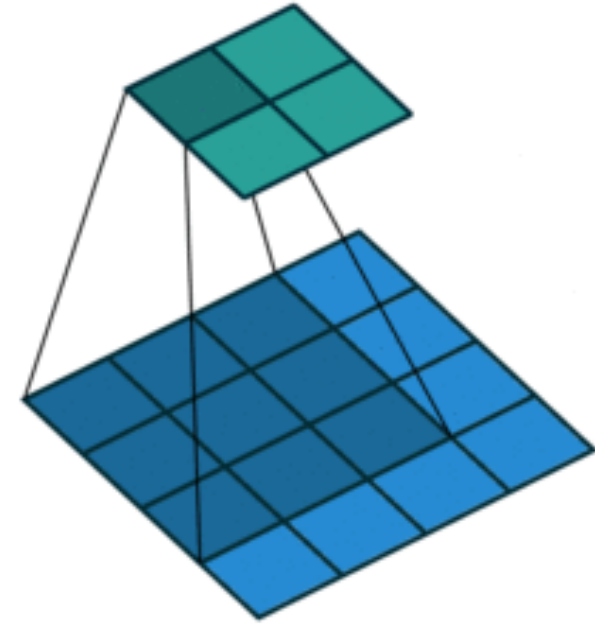


2-D Max Pooling

AVG

AVG

- Returns the ~~maximal~~ ^{max} value in the sliding window



2-D Max Pooling

- Returns the maximal value in the sliding window

Vertical edge detection

```
[ [1. 1. 0. 0. 0.
  [1. 1. 0. 0. 0.
  [1. 1. 0. 0. 0.
  [1. 1. 0. 0. 0.
```

Conv output

```
[ [ 0. 1. 0. 0.
  [ 0. 1. 0. 0.
  [ 0. 1. 0. 0.
  [ 0. 1. 0. 0.
```

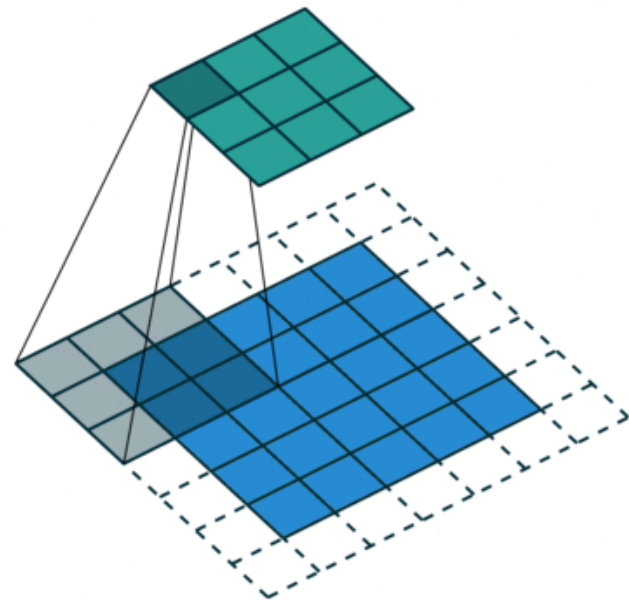
2 x 2 max pooling

```
[ [ 1. 1. 1. 0.
  [ 1. 1. 1. 0.
  [ 1. 1. 1. 0.
  [ 1. 1. 1. 0.
```

Tolerant to 1
pixel shift

Padding, Stride, and Multiple Channels

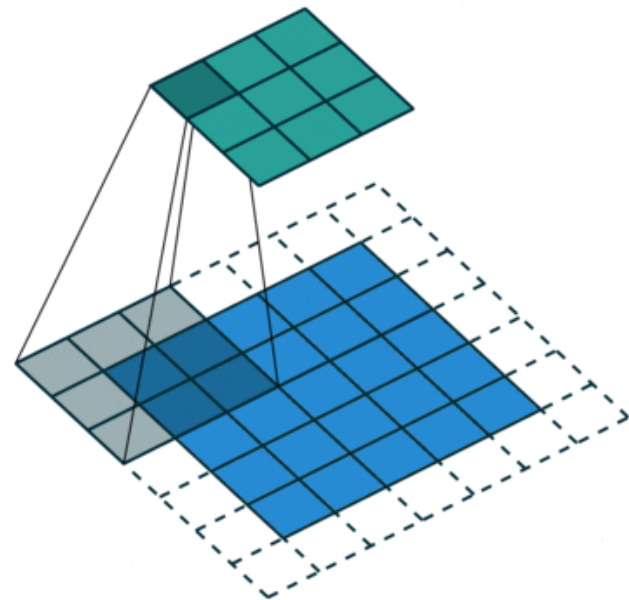
- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel



#output channels = #input channels

Padding, Stride, and Multiple Channels

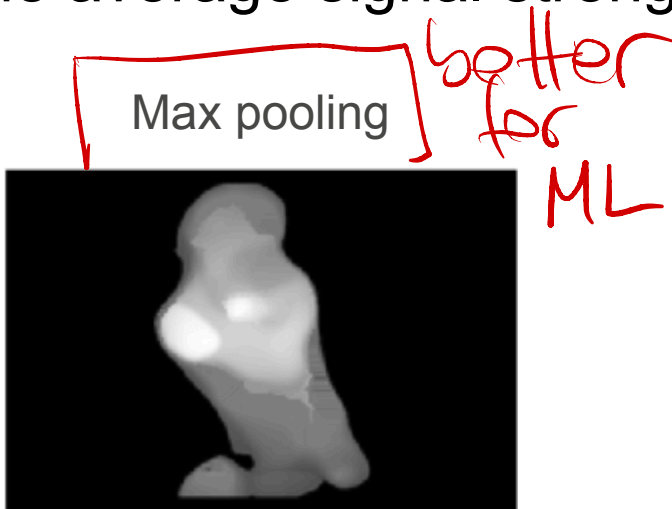
- Pooling layers have similar padding and stride as convolutional layers
- No learnable parameters
- Apply pooling for each input channel to obtain the corresponding output channel



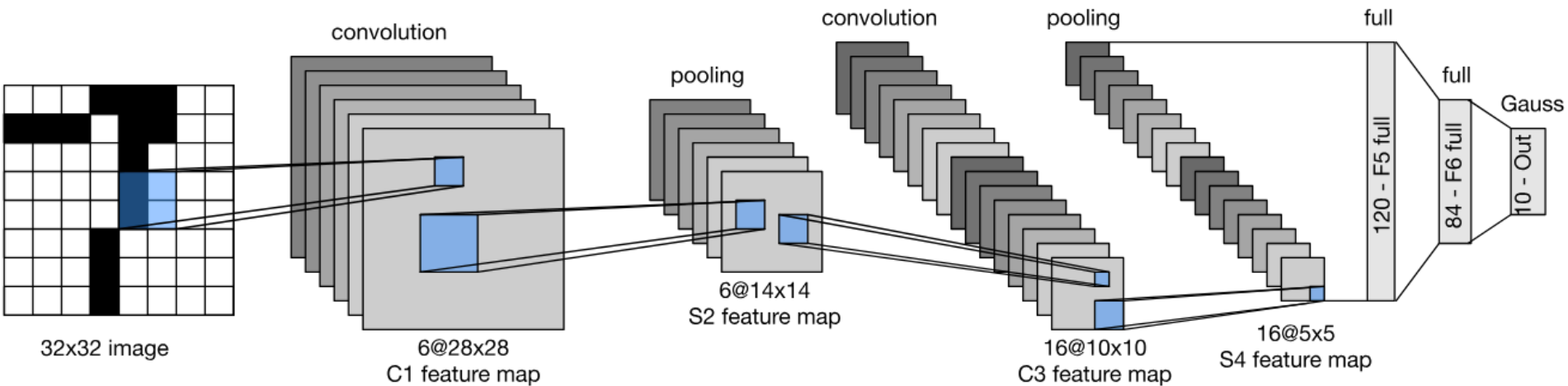
#output channels = #input channels

Average Pooling

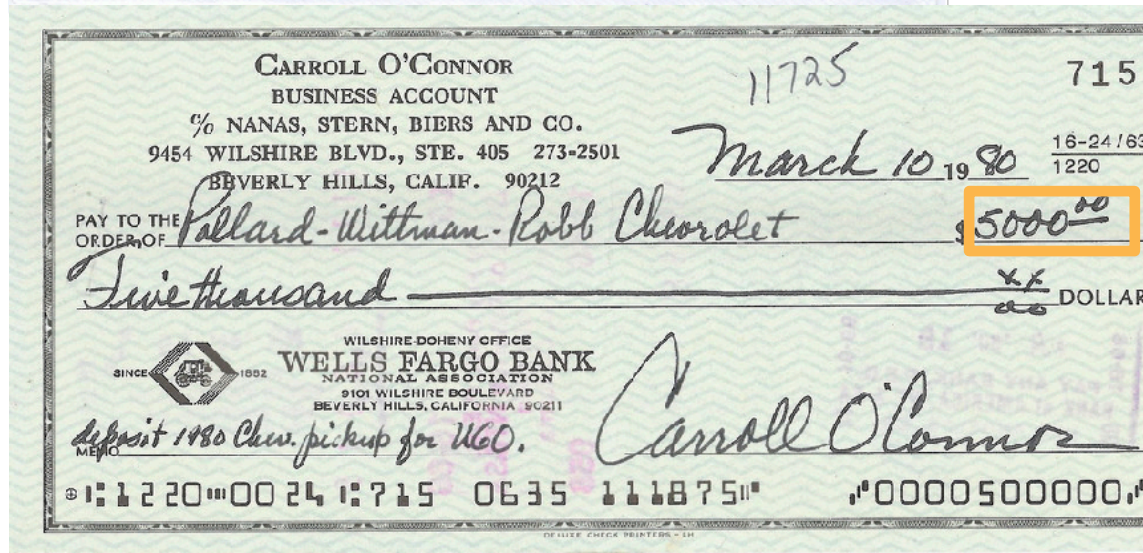
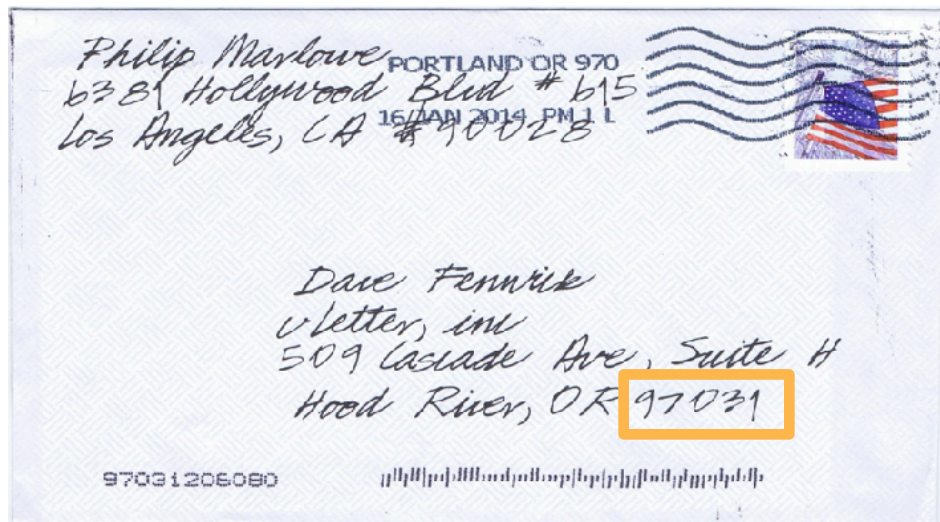
- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
 - The average signal strength in a window



LeNet Architecture



Handwritten Digit Recognition



MNIST

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





0
103



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

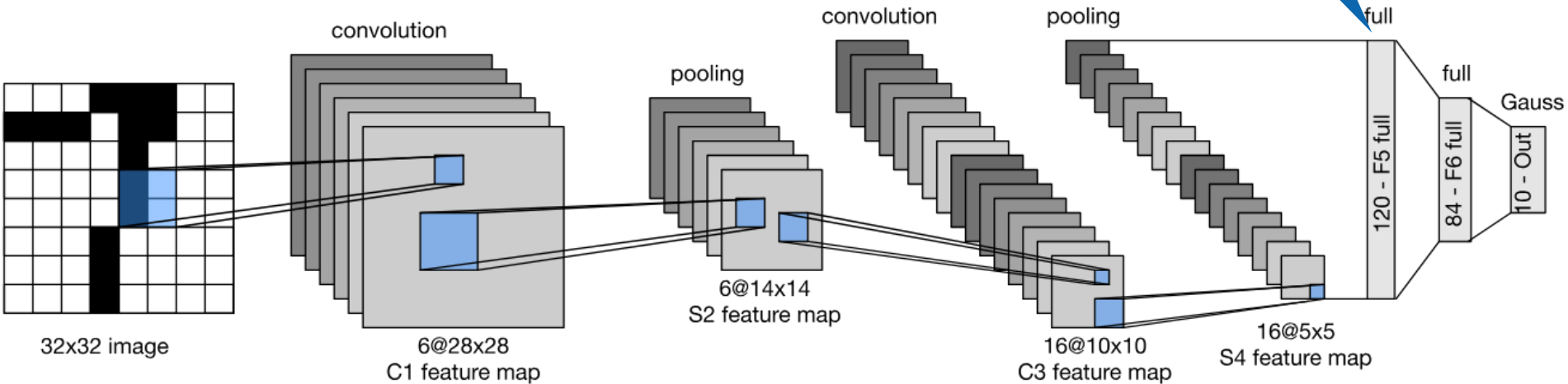


0
103



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

Expensive if we
have many
outputs



LeNet in MXNet

```
net = gluon.nn.Sequential()
with net.name_scope():
    net.add(gluon.nn.Conv2D(channels=20, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Conv2D(channels=50, kernel_size=5, activation='tanh'))
    net.add(gluon.nn.AvgPool2D(pool_size=2))
    net.add(gluon.nn.Flatten())
    net.add(gluon.nn.Dense(500, activation='tanh'))
    net.add(gluon.nn.Dense(10))
```

```
loss = gluon.loss.SoftmaxCrossEntropyLoss()
```

(size and shape inference is automatic)

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
 - Provides some degree of invariance to translation