Introduction to Deep Learning

Chapter 7: Convolutional and Pooling Layers

MGMT735 Deep Learning

Slides From Alex Smola and Mu Li

courses.d2l.ai/berkeley-stat-157

Logistics


Convolutions

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 (36M $\times$ 100) Billion parameters
- Exceeds the population of dogs and cats on earth (900M dogs + 600M cats)

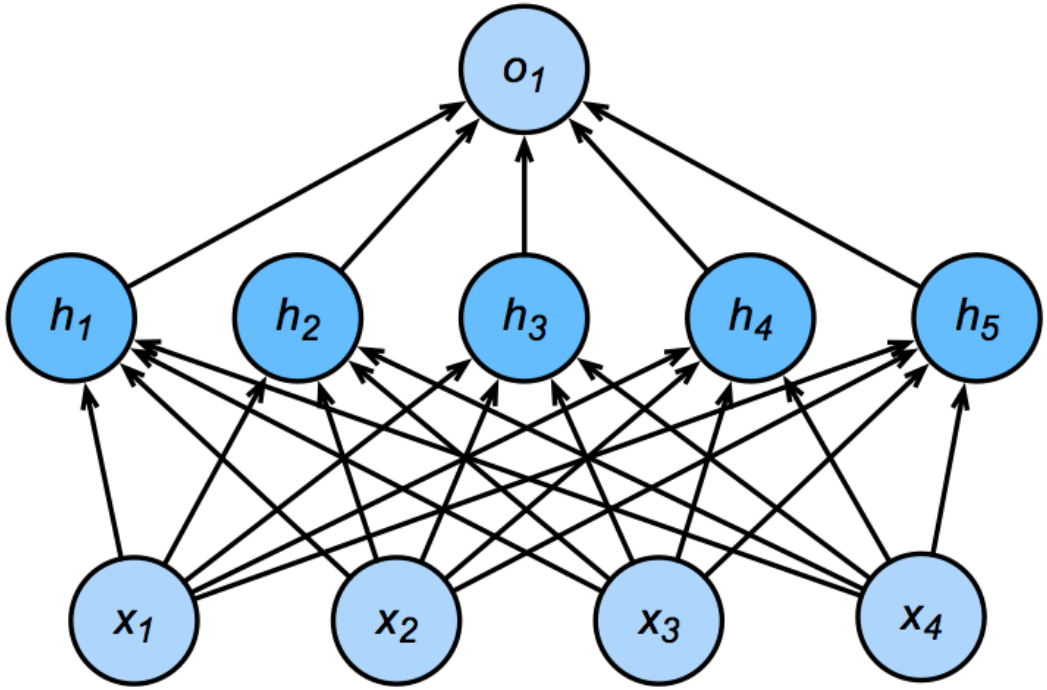
Flashback - Network with one hidden layer

Output layer

100 neurons Hidden layer

3.6B parameters = 14GB

36M features Input layer



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

Where is Waldo?

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

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

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

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

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

cross-correlation

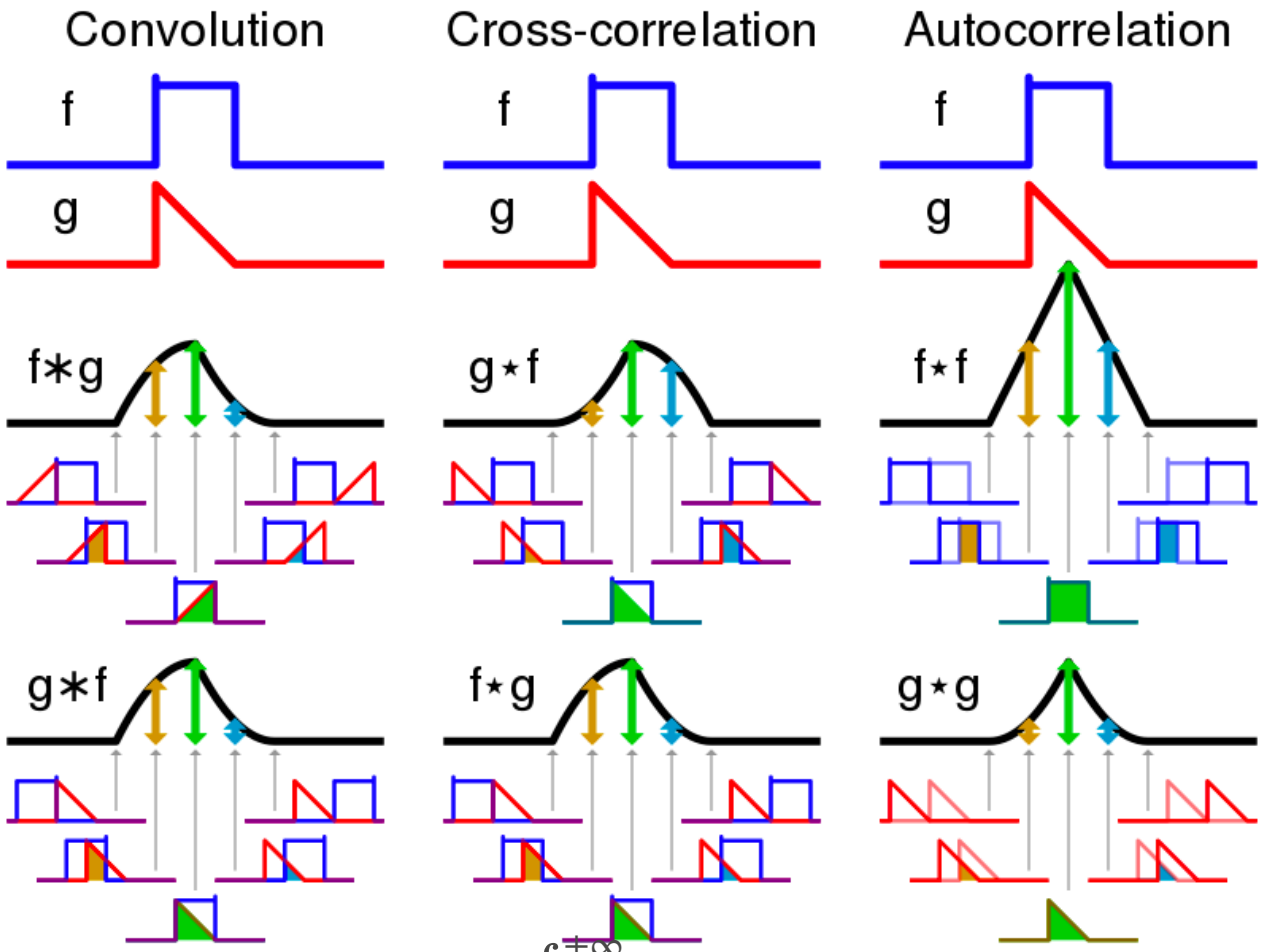
Idea #2 - Locality

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

- 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



$$f \star g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t-\tau)d\tau \quad f \times g(t) = \int_{-\infty}^{+\infty} f(\tau)g(t+\tau)d\tau = f(t) \star g(-t)$$

2-D Cross Correlation

Input

0	1	2
3	4	5
6	7	8

*

Kernel

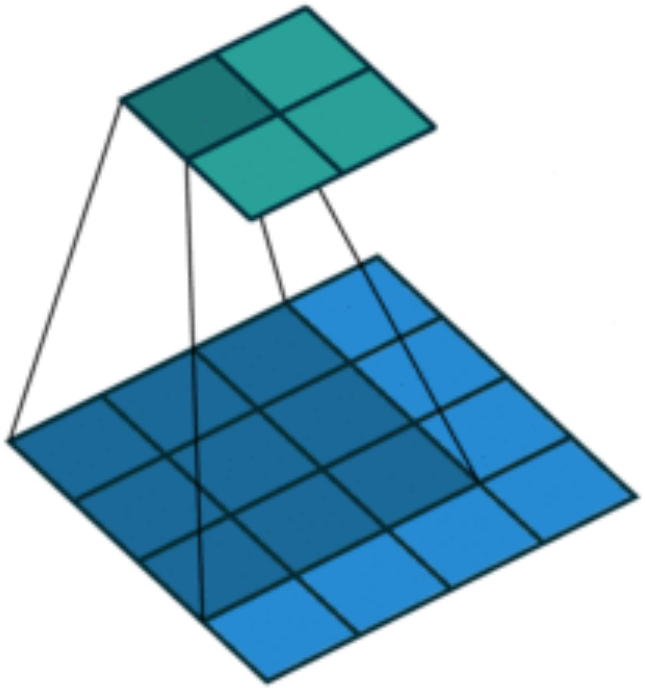
0	1
2	3

=

Output

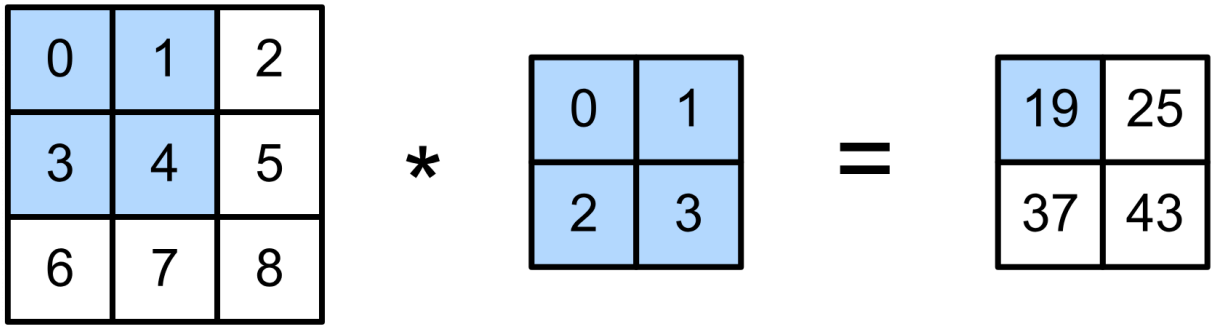
19	25
37	43

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



(vdumoulin@ Github)

2-D Convolution Layer

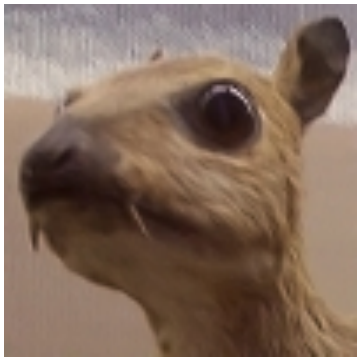


- $\mathbf{X}$: $n_h \times n_w$ input matrix
- $\mathbf{W}$: $k_h \times k_w$ kernel matrix
- b: scalar bias
- $\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

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

- **W** and *b* are learnable parameters

Examples

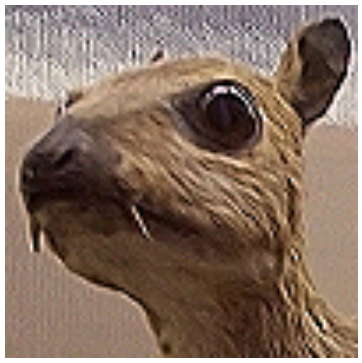


(wikipedia)

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

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)

Cross Correlation vs Convolution

- 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 (translation variant)
  - Voice
  - Time series

- 3-D

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

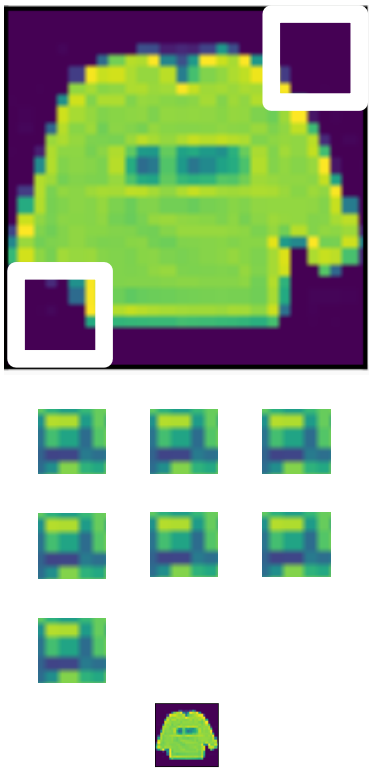
  - Video
  - Medical images


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

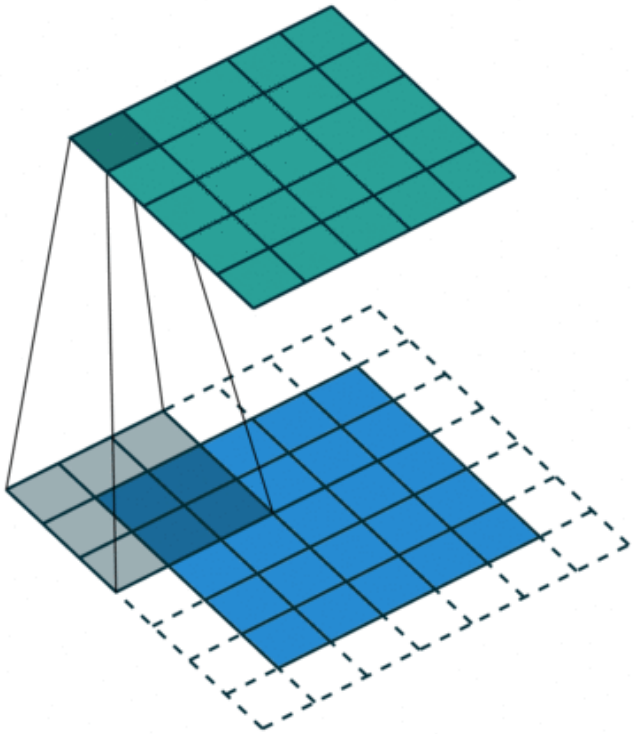
$$(n_h - k_h + 1) \times (n_w - k_w + 1)$$



Padding

Padding adds rows/columns around input

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



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

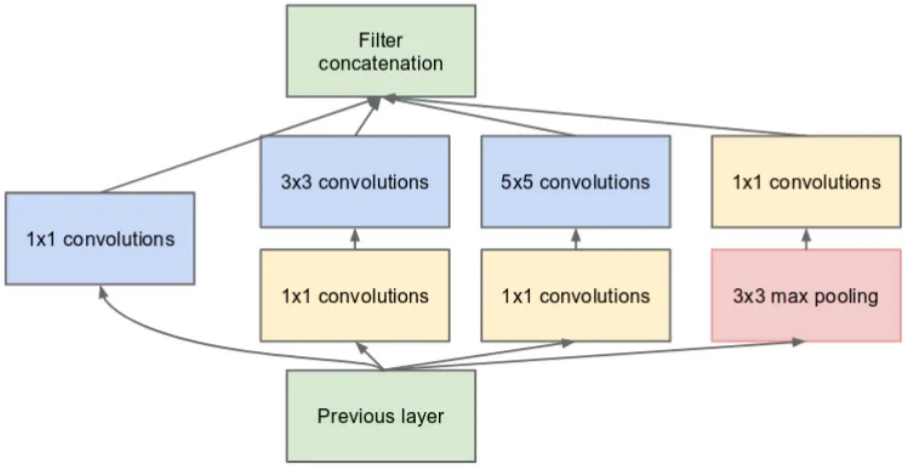
Padding

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

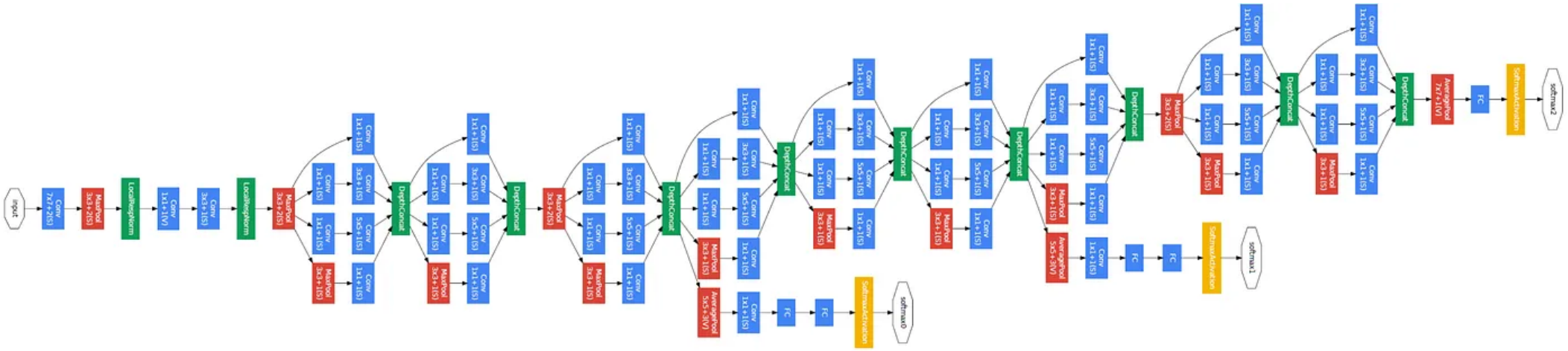
$$(n_h - k_h + p_h + 1) \times (n_w - k_w + p_w + 1)$$

- 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

  - This is particular useful when you have different kernel sizes

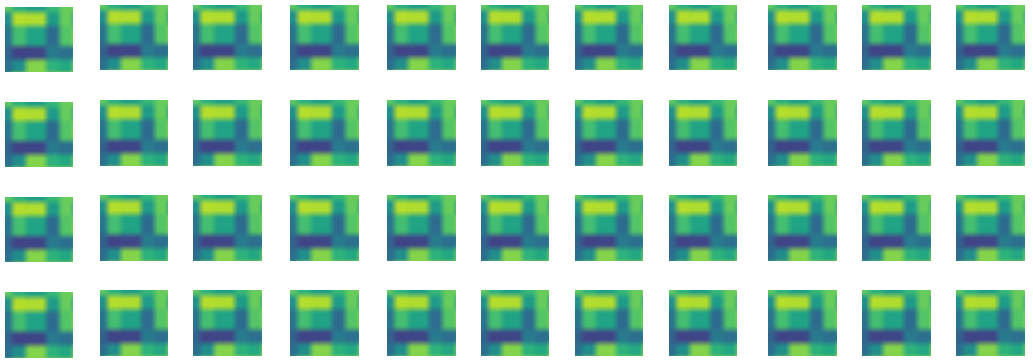


(b) Inception module with dimension reductions



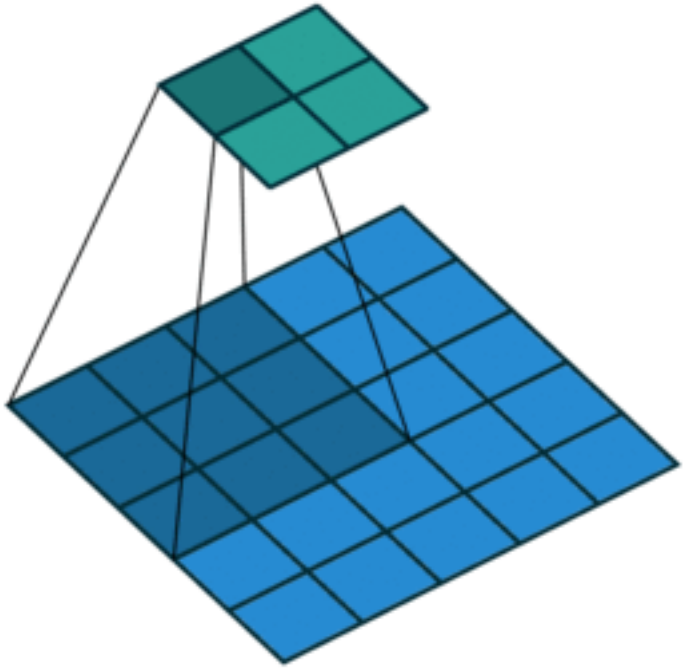
Stride

- 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

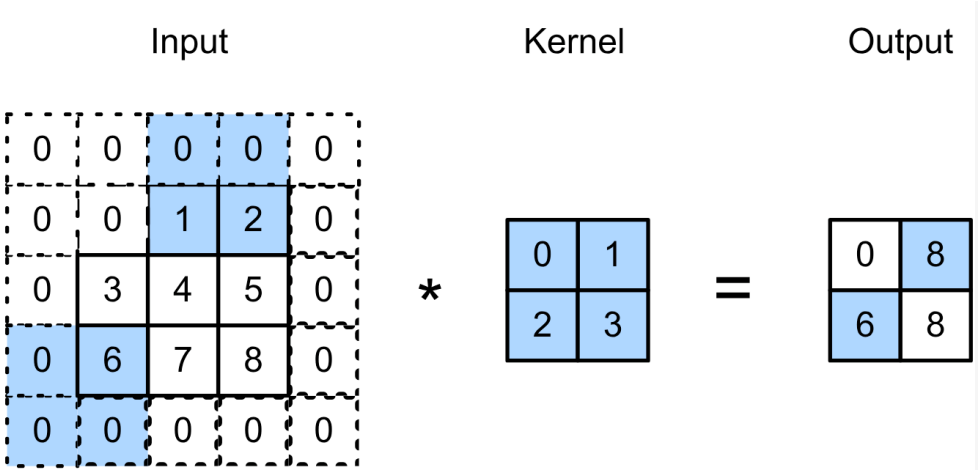


Stride

- Stride is the #rows/#columns per slide



Strides of 3 and 2 for height and width



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

Stride

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

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

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

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

- If input height/width are divisible by strides

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


Multiple Input and
Output Channels

Multiple Input Channels

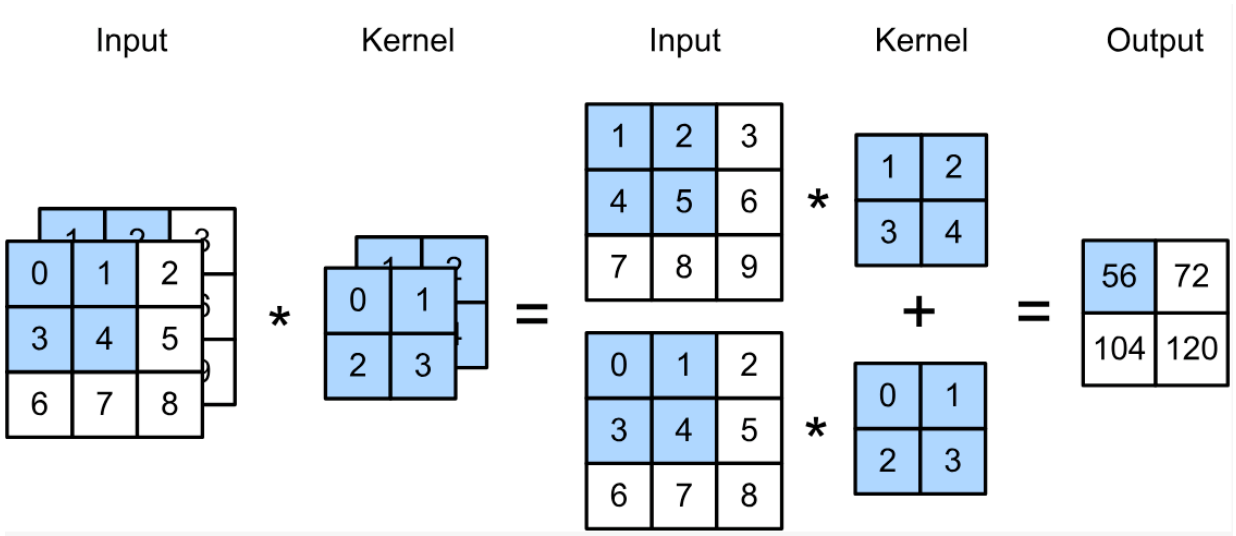
- Color image may have three RGB channels
- Converting to grayscale loses information

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



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

Multiple 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

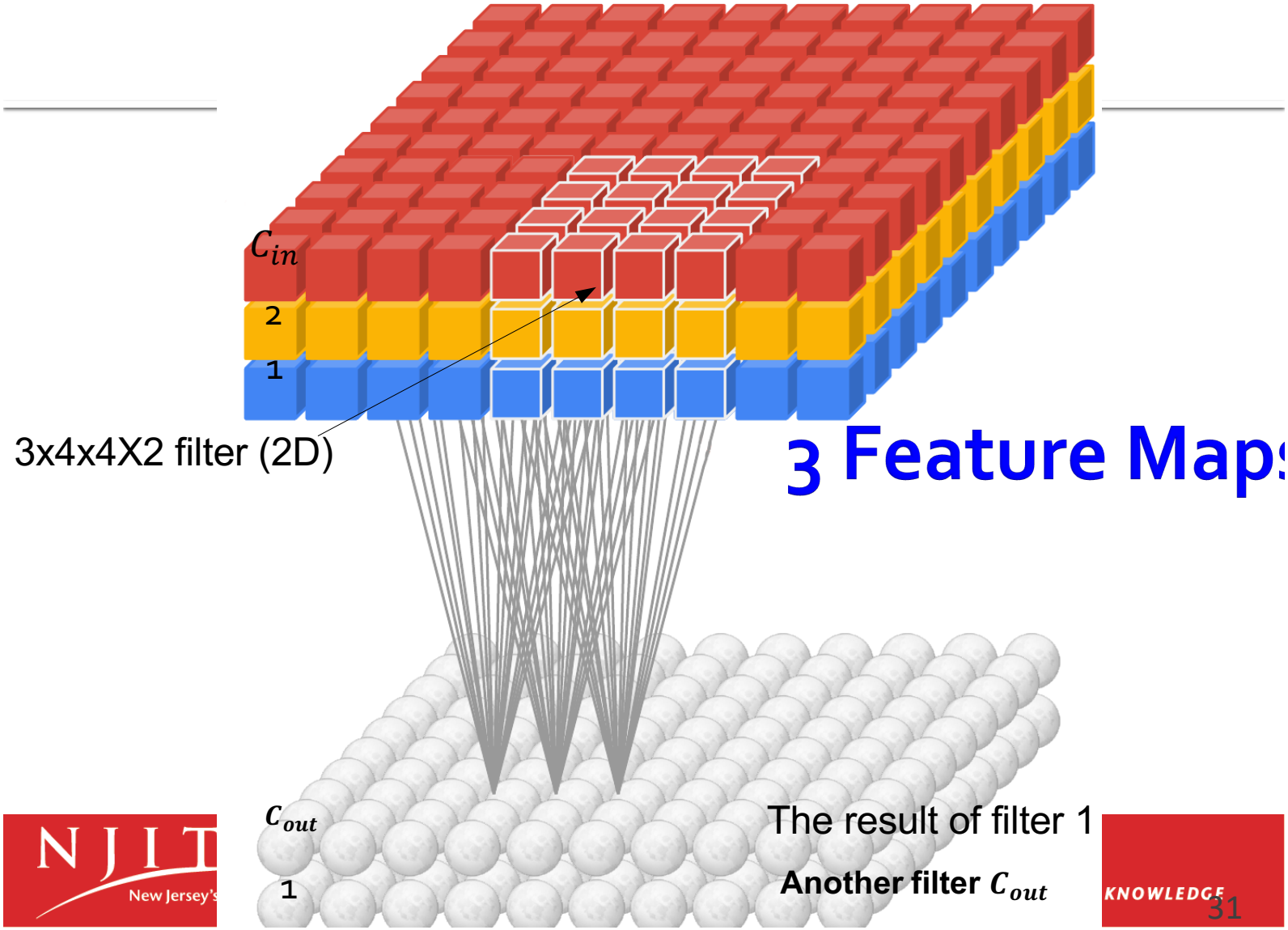
$$\mathbf{Y} = \sum_{i=0}^{c_i} \mathbf{X}_{i,:,:} \star \mathbf{W}_{i,:,:}$$

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 $\mathbf{Y} : c_o \times m_h \times m_w$

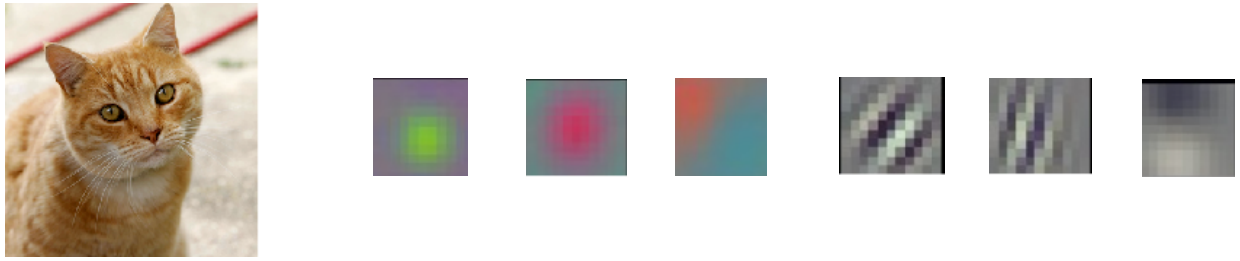
$$\mathbf{Y}_{i,:,:} = \mathbf{X} \star \mathbf{W}_{i,:,:,:}$$

for $i = 1, \ldots, c_o$


C_{in}
2
1
3x4x4X2 filter (2D)
3 Feature Maps
C_{out}
1
The result of filter 1
Another filter C_{out}

Multiple Input/Output Channels

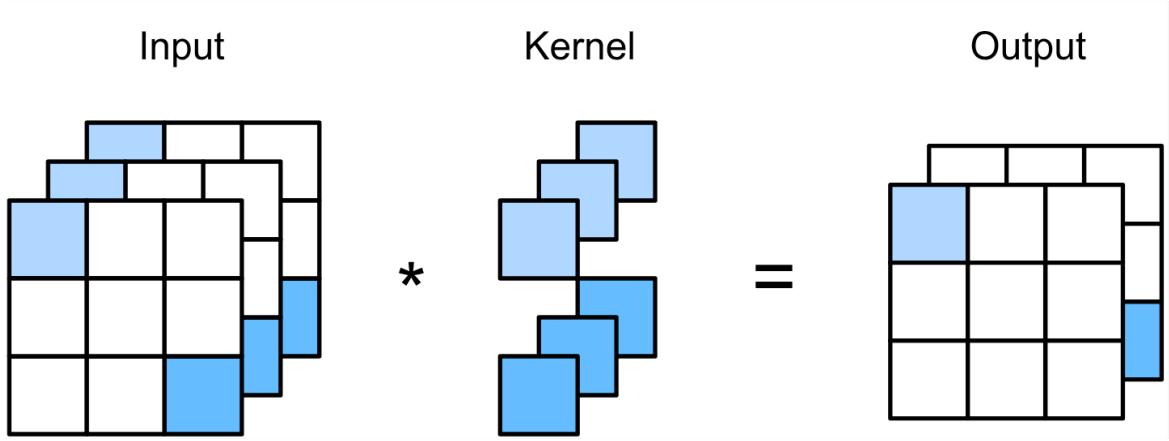
- Each output channel may recognize a particular pattern



- 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 $\mathbf{B} : c_o \times 1$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$

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

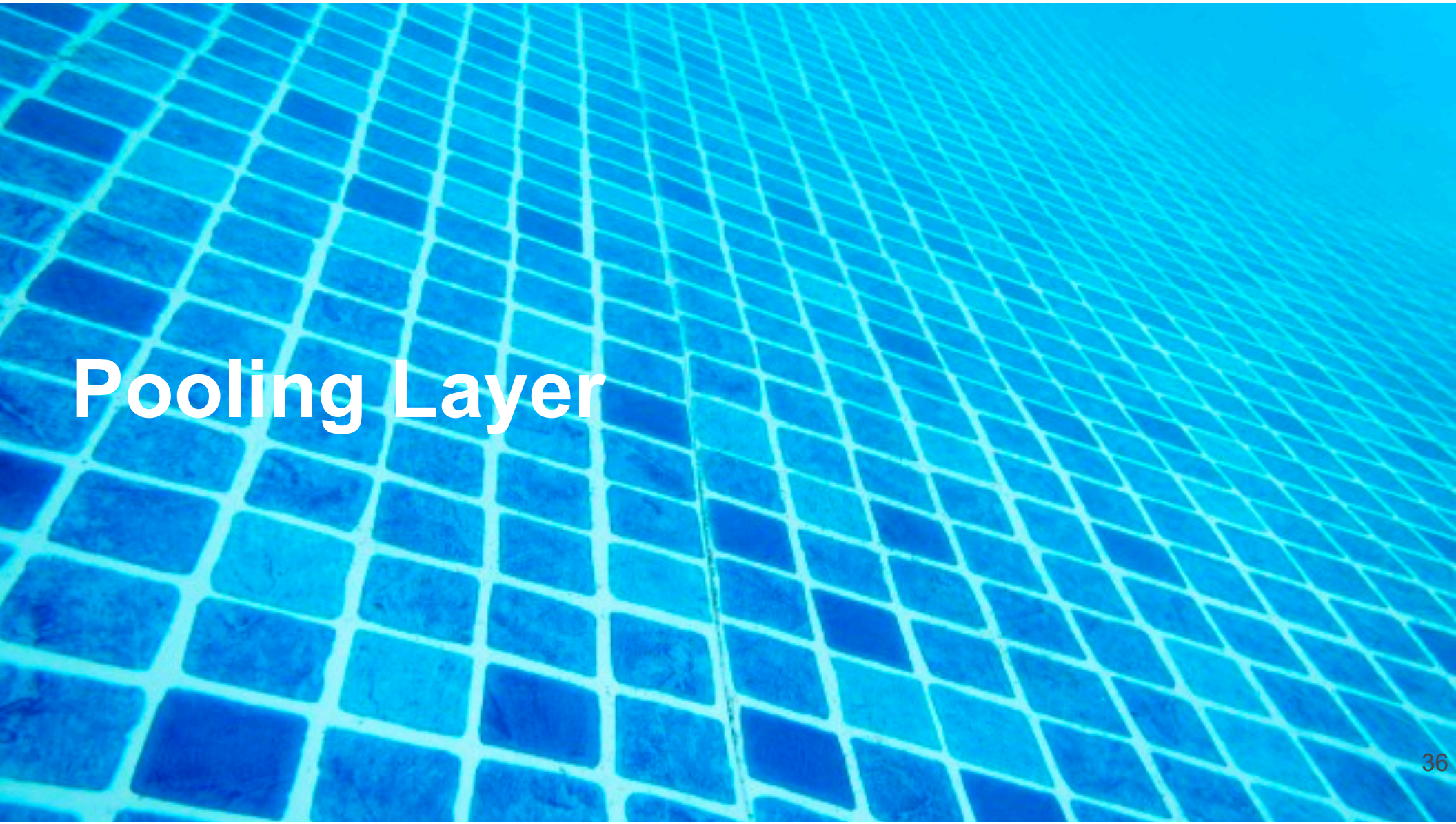
- Complexity (number of floating point operations FLOP)

$$c_i = c_o = 100$$
$$k_h = h_w = 5$$
$$m_h = m_w = 64$$

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

1GFLOP

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


Pooling Layer

Pooling

- Convolution is sensitive to position
  - Detect vertical edges

X

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

Y

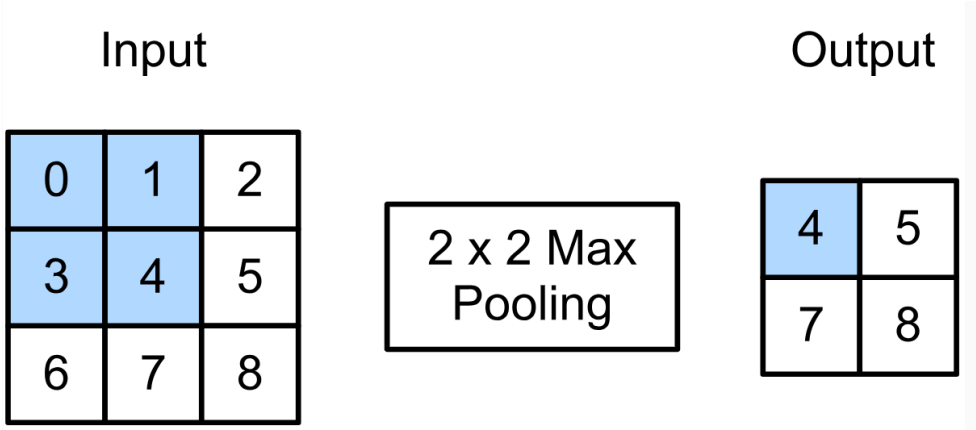
```
[[ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
 [ 0.  1.  0.  0.
```

0 output with 1 pixel shift

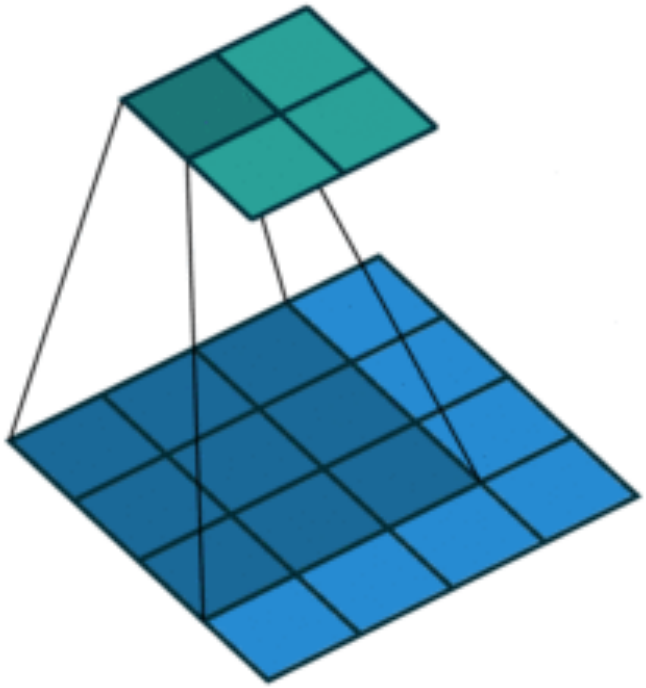
- Image shifts, causes edge feature shifts, and different representation.
- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

2-D Max Pooling

- Returns the maximal value in the sliding window



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



2-D Max Pooling

- 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.
```

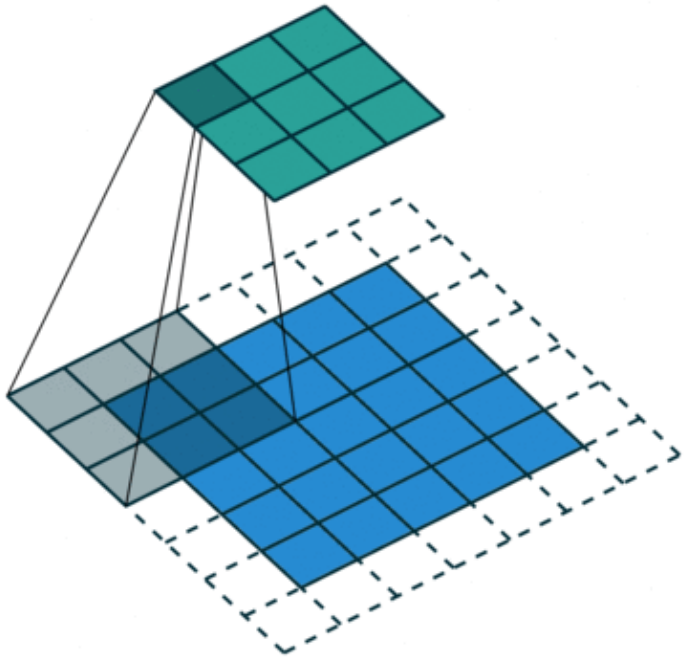
2 x 2 max pooling

```
[[ 1.  1.  0.  0.
 [ 1.  1.  0.  0.
 [ 1.  1.  0.  0.
 [ 1.  1.  0.  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
- What is the stride size?

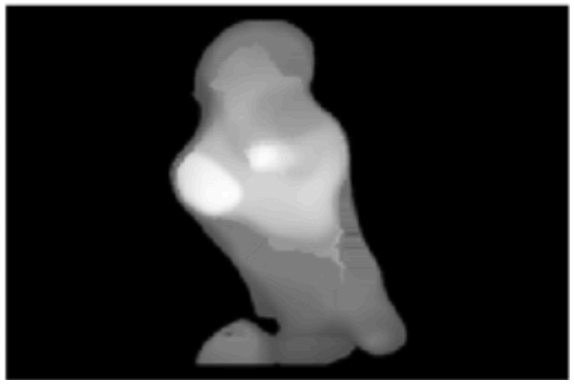


#output channels = #input channels

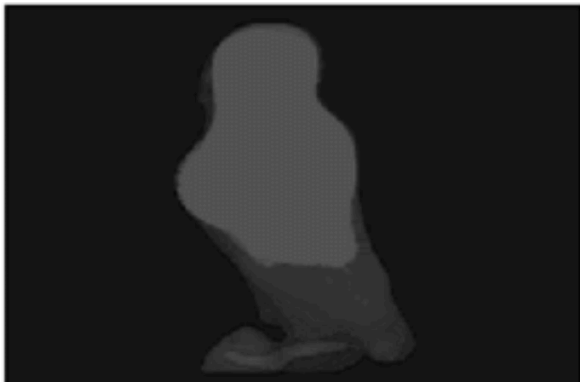
Average Pooling

- 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
  - Max pooling is better except at the last layer.

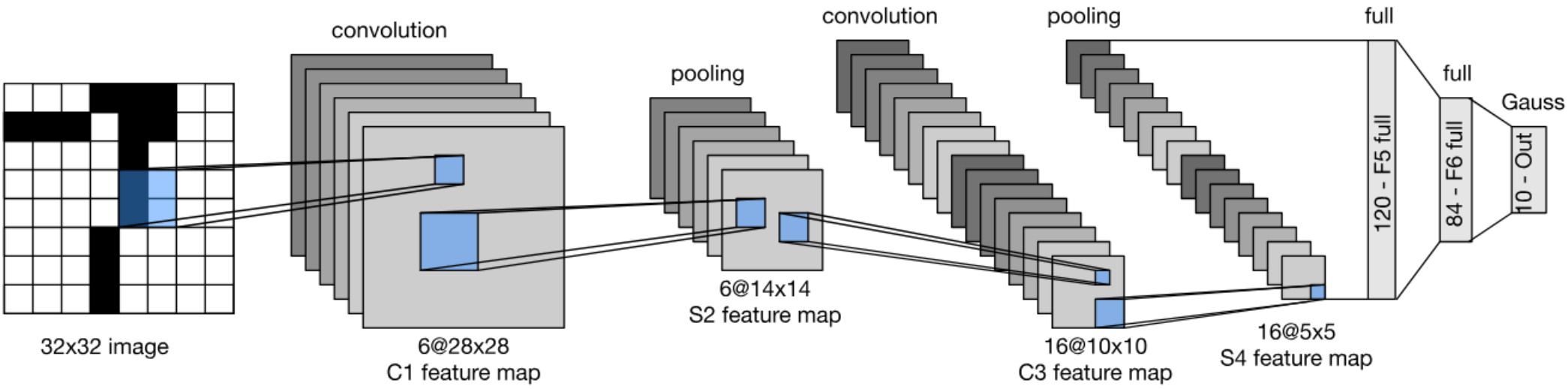
Max pooling



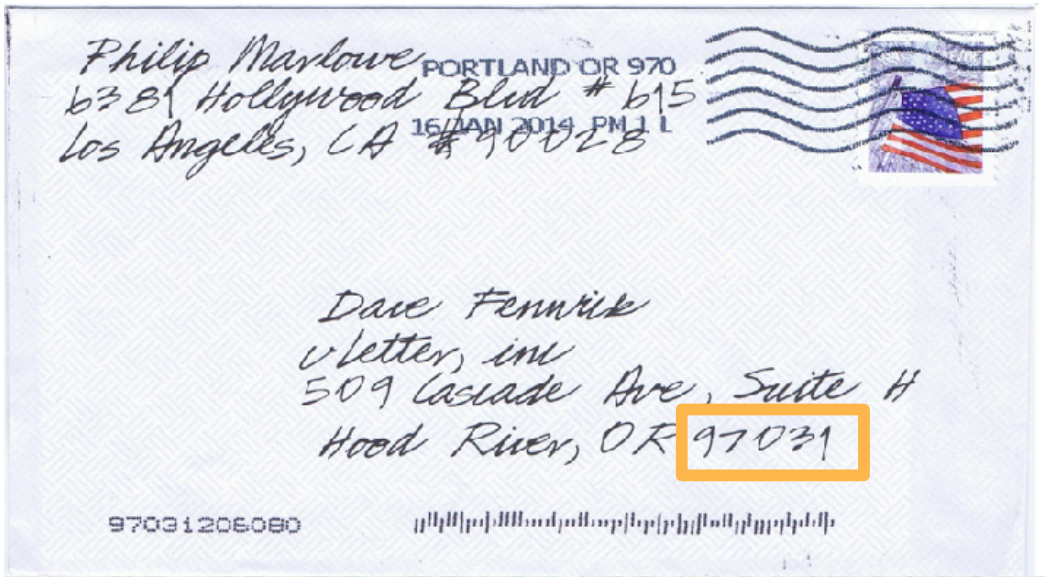
Average pooling

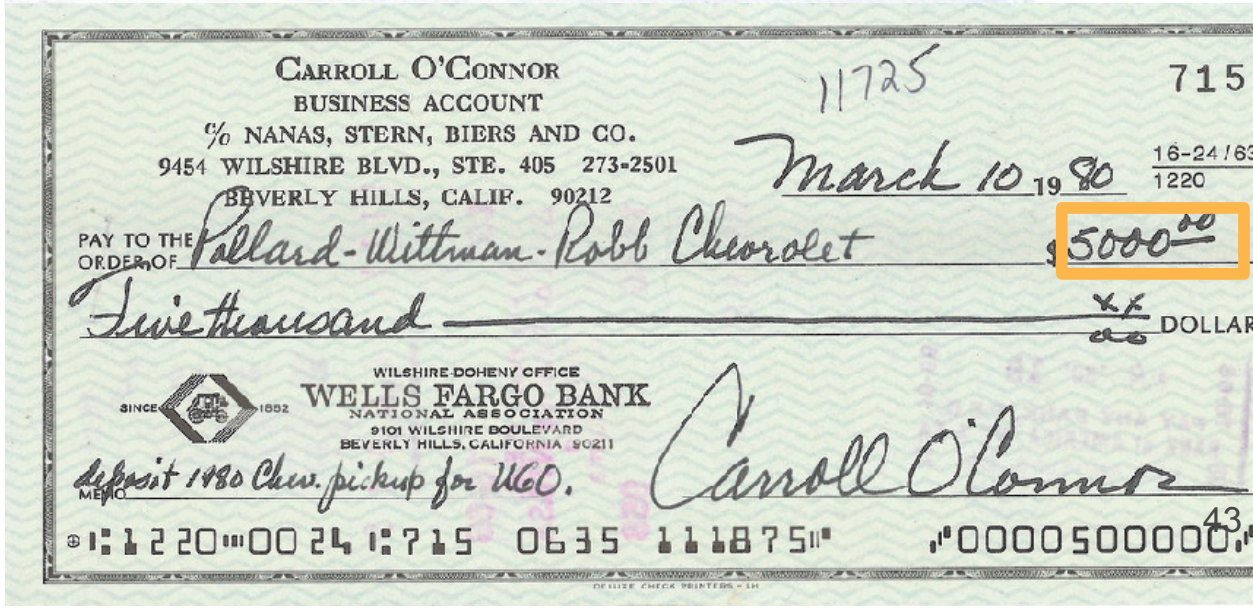


LeNet Architecture



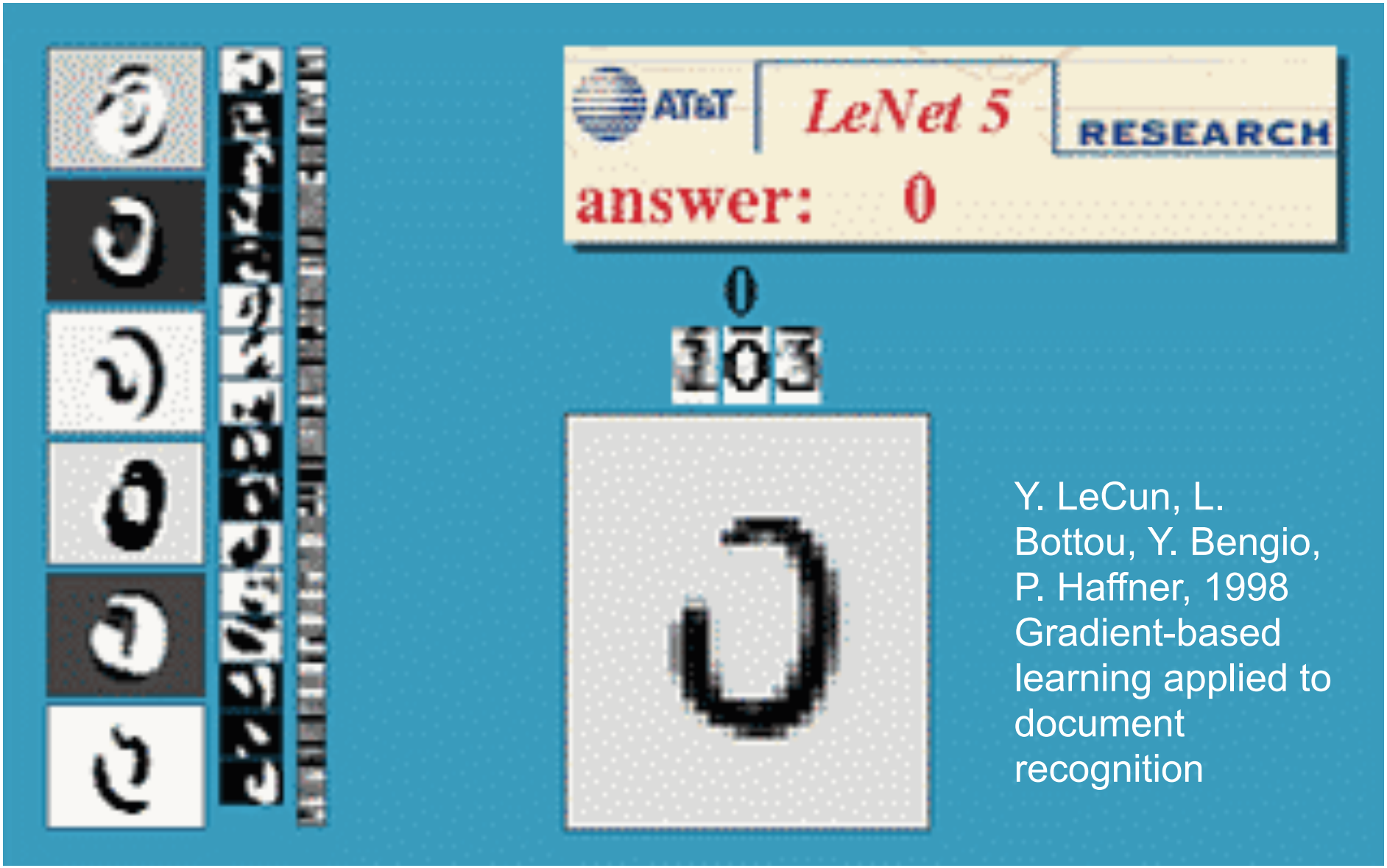
Handwritten Digit Recognition



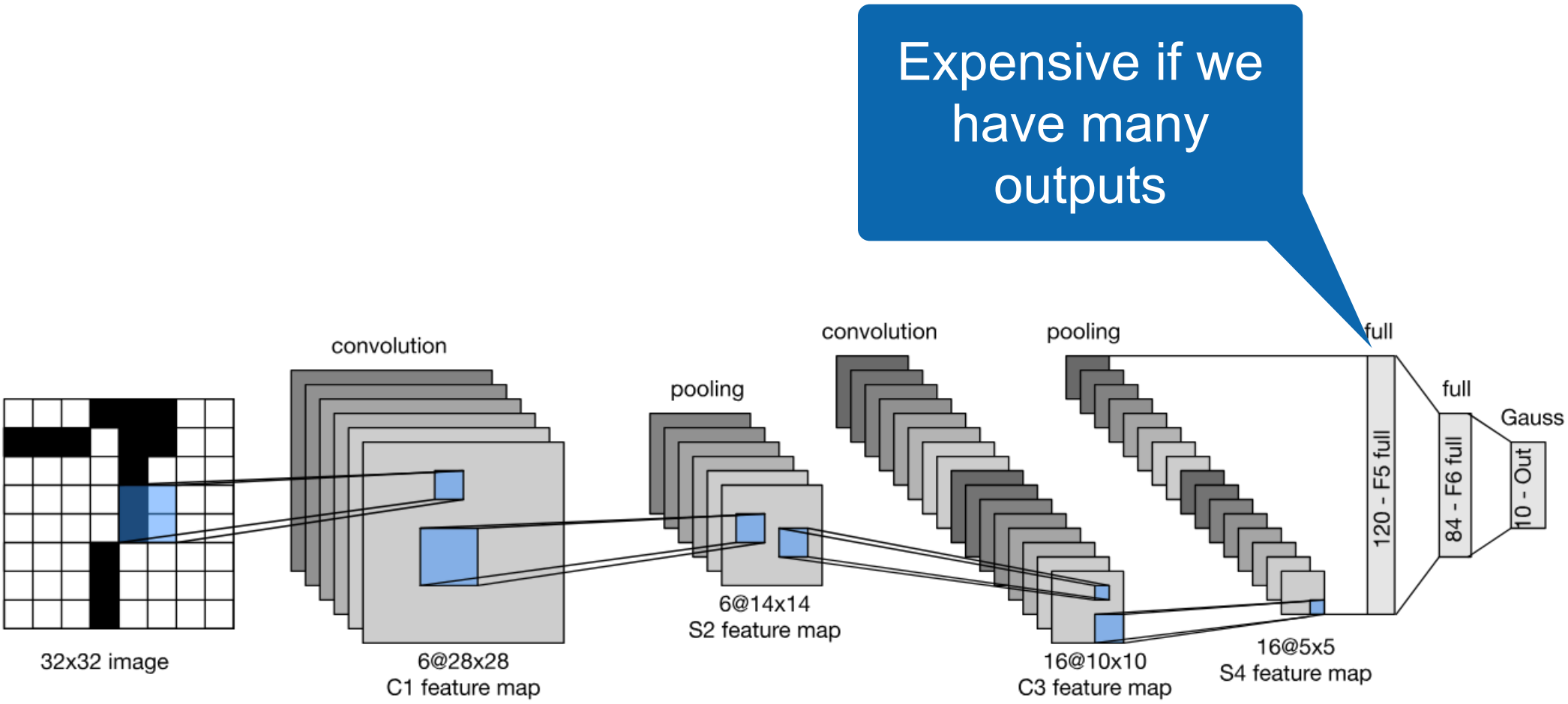


MNIST

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



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


Expensive if we have many outputs
convolution
pooling
convolution
pooling
full
full
Gauss
120 - F5 full
84 - F6 full
10 - Out
32x32 image
6@28x28
C1 feature map
6@14x14
S2 feature map
16@10x10
C3 feature map
16@5x5
S4 feature map

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 pattens
  - High computation complexity
  - Control output shape via padding, strides and channels
- Max/Average Pooling layer
  - Provides some degree of invariance to translation