# 165B
# Machine Learning
# Convolutional Neural Networks

Lei Li (leili@cs)
UCSB

Acknowledgement: Slides borrowed from Bhiksha Raj's 11485 and Mu Li & Alex Smola's 157 courses on Deep Learning, with modification

# Resume in-person instruction

- starting on Jan 31, 2022

# Recap

- Generalization error: the expected error on unseen data (general population)
  - Minimizing training loss does not always lead to minimizing the generalization error
- Under-fitting: model does not have adequate capacity ==> increase model size, or choose a more complex model
- Over-fitting: validation loss does not decrease while training loss still does
- Regularization
  - L1 ==> more sparse parameters
  - L2/Weight decay ==> shrink parameters
  - Dropout, equivalent to L2, but as a network Layer
- Numerical issues in training
  - gradient explosion & gradient vanishing
  - Proper initialization of parameters
  - Gradient clipping
  - Early stoping

# Underfitting and Overfitting


Image credit: hackernoon.com

# Convolution

# Problem: Classifying Dog and Cat Images

- Use a good camera
- RGB image has 36M elements
- What is the size of a FFN with a single hidden layer (100 hidden units)?
- How to reduce parameter size?


Where is Waldo?

# Two Principles

- Translation Invariance
- Locality

# Full Projection in Tensor Form

- Input image: a matrix with size (h, w)
- Projection weights: a 4-D tensors (h,w) by (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 $v_{i,j,a,b} = w_{i,j,i+a,j+b}$

Tensor is a generalization of matrix

# 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

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

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

# 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 $v_{a,b} = 0$

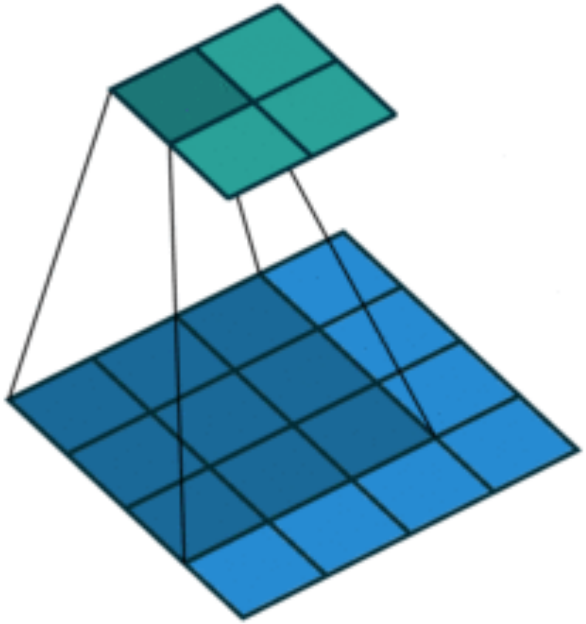
$$h_{i,j} = \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} v_{a,b} x_{i+a,j+b}$$

# 2-D Convolution Layer

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

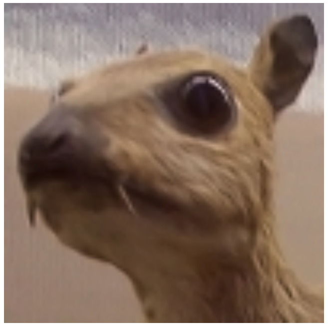
$$y_{i,j} = \sum_{a=1}^{h} \sum_{b=1}^{w} w_{a,b} x_{i+a,j+b}$$

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



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

# Examples

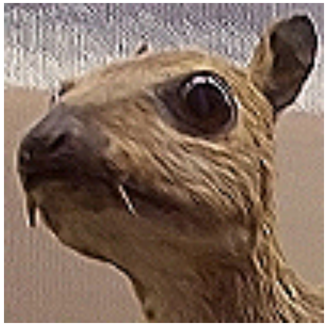


(wikipedia)

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

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

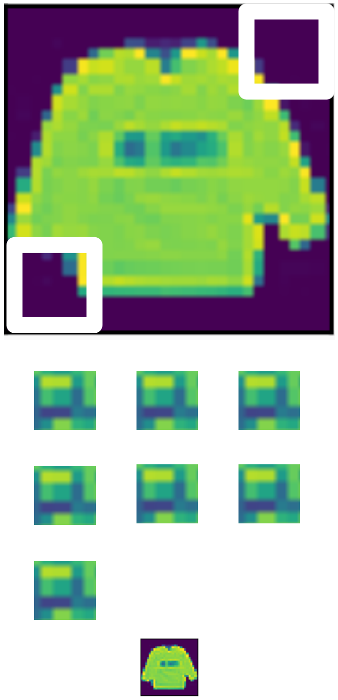
# Examples

(Rob Fergus)


Padding and Stride

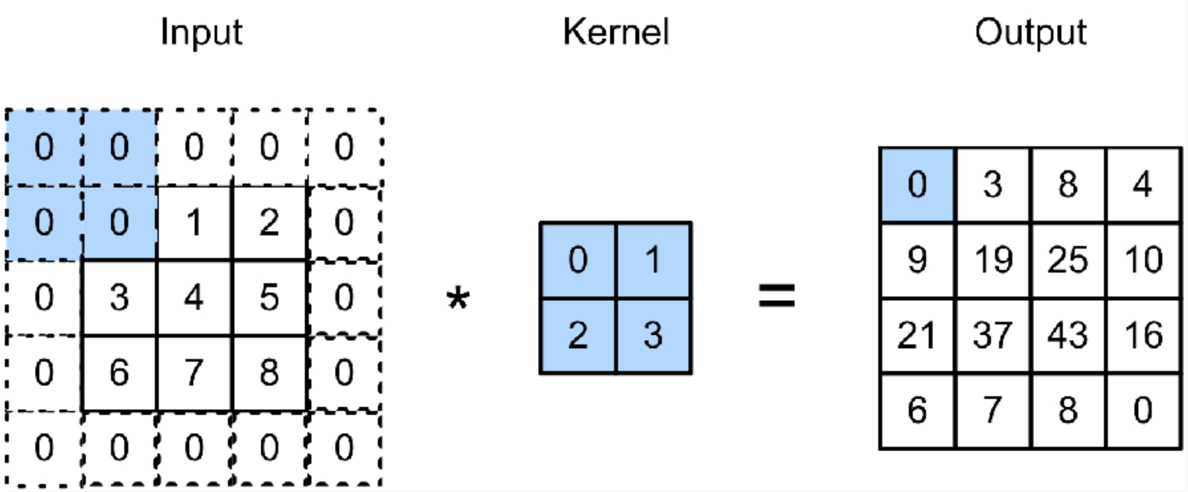
# Padding

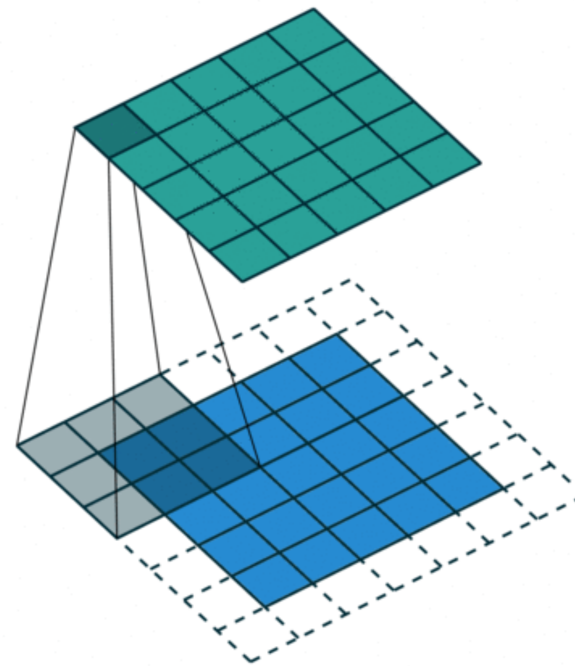
- Given a 32 x 32 input image
- Apply convolutional layer with 5 x 5 kernel
  - 28 x 28 output with 1 layer
  - 4 x 4 output with 7 layers
- Shape decreases faster with larger kernels
  - Shape reduces from $n_h \times n_w$ to $(n_h - k_h + 1) \times (n_w - k_w + 1)$



# Padding

Padding adds rows/columns around input



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

# 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

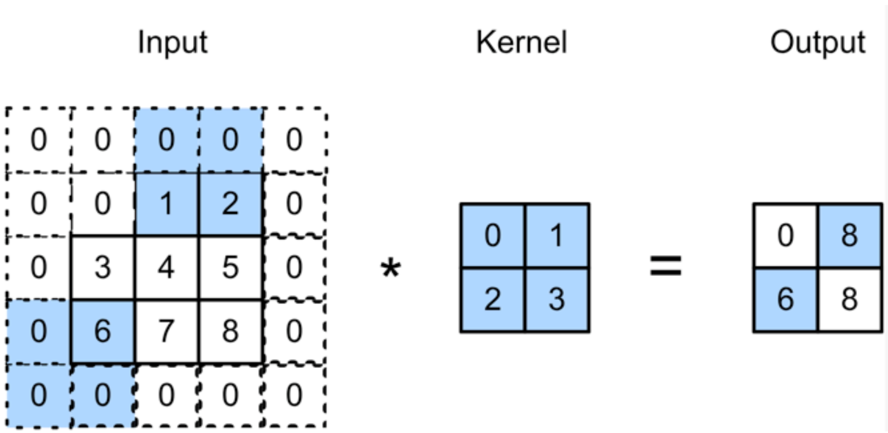
# Stride

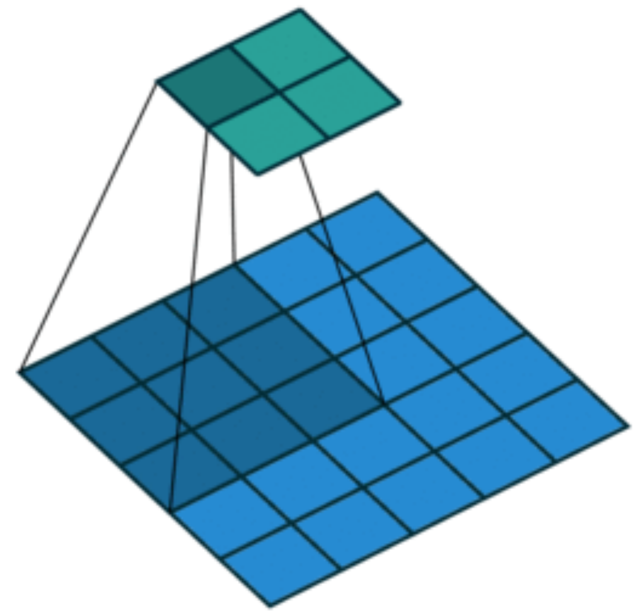
- Padding reduces shape linearly with #layers
  - Given a 224 x 224 input with a 5 x 5 kernel, needs 44 layers to reduce the shape to 4 x 4
  - Requires a large amount of computation

# Stride

- Stride is the #rows/#column

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 Channels

# Multiple Input Channels

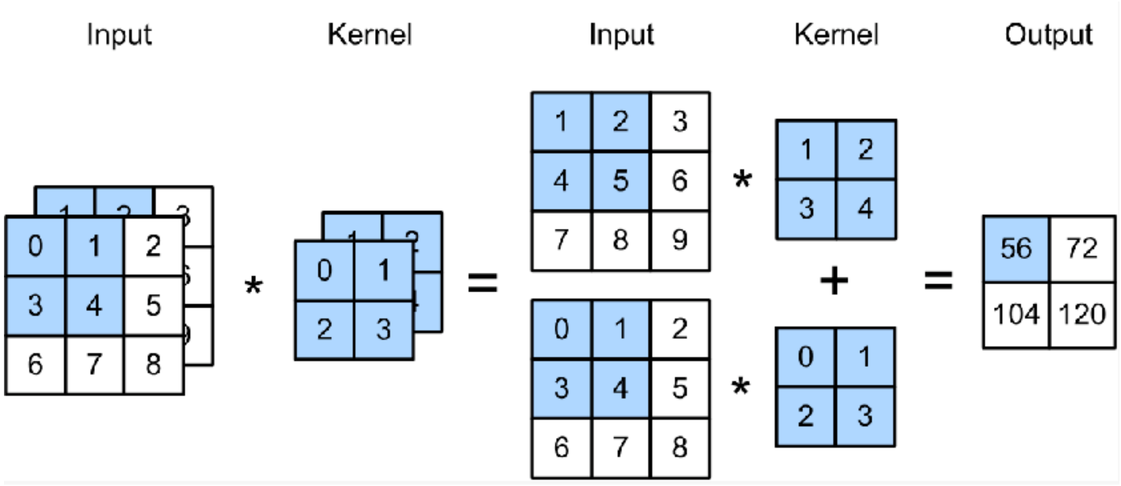
- Color image may have three RGB channels
- Converting to grayscale loses information

# Multiple Input Channels

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

# Multiple Input Channels

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




$$(1\times1+2\times2+4\times3+5\times4)$$
$$+(0\times0+1\times1+3\times2+4\times3)$$
$$=56$$

# Multiple Input Channels

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

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

# Multiple Output Channels

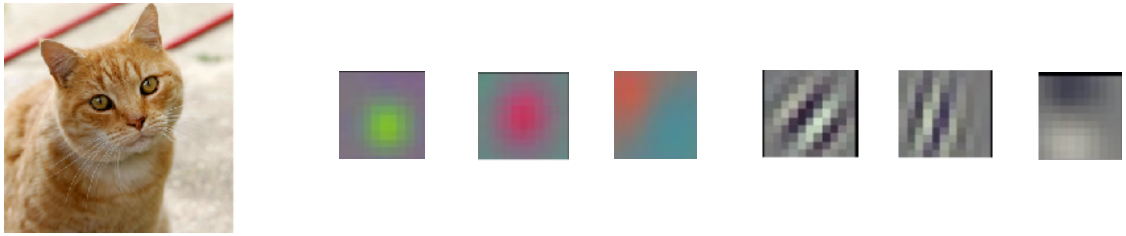
- No matter how many inputs channels, so far we always get single output channel
- We can have multiple 3-D kernels, each one generates a output channel
- Input $\mathbf{X} : c_i \times n_h \times n_w$
- Kernel $\mathbf{W} : c_o \times c_i \times k_h \times k_w$
- Output $\mathbf{Y} : c_o \times m_h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

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

for $i = 1,\ldots, c_o$

# Multiple Input/Output Channels

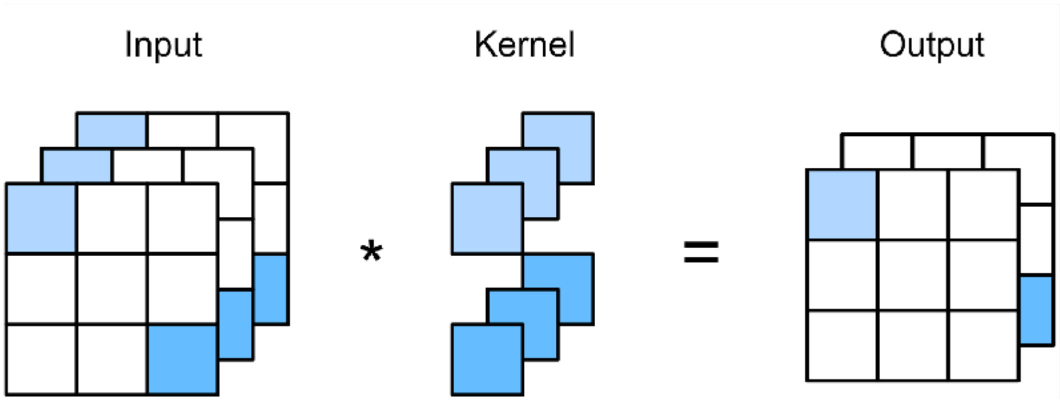
- Each output channel may recognize a particular pattern



- 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 $\mathbf{B} : c_o$
- Output $\mathbf{Y} : c_o \times m_h \times m_w$
- Complexity (number of floating point operations FLOP)

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

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

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

1GFLOP

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

# Quiz

- https://edstem.org/us/courses/16390/lessons/28985/edit/slides/166358

# Pooling Layer

# Pooling

- Convolution is sensitive to position
  - Detect vertical edges

0 output with 1

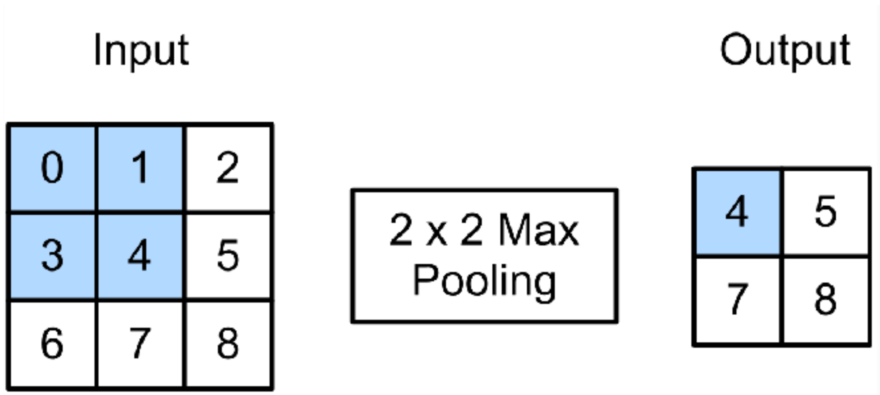
X
```
[[1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
 [1. 1. 0. 0. 0.
```

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

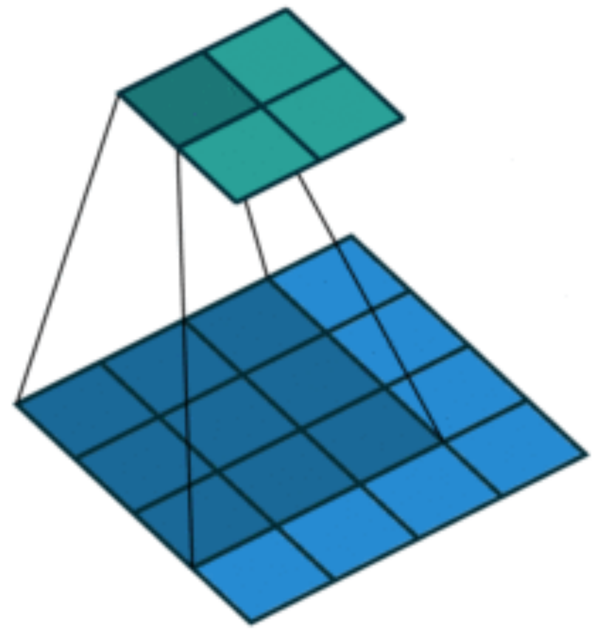
- We need some degree of invariance to translation
  - Lighting, object positions, scales, appearance vary among images

# 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

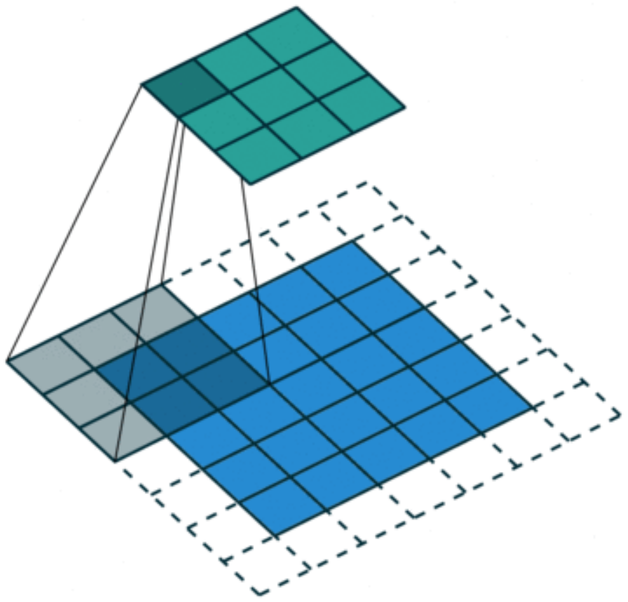
```
[[ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
 [ 1.  1.  1.  0.
```

Tolerant to 1 pixel

# 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

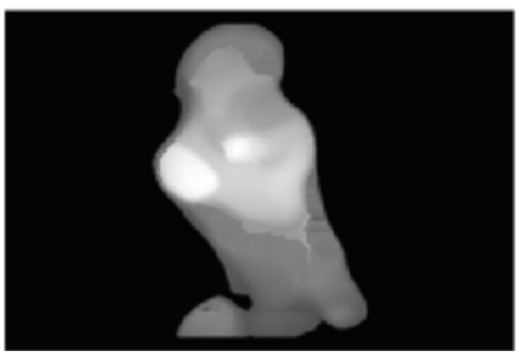
**#output channels = #input channels**



# Average Pooling

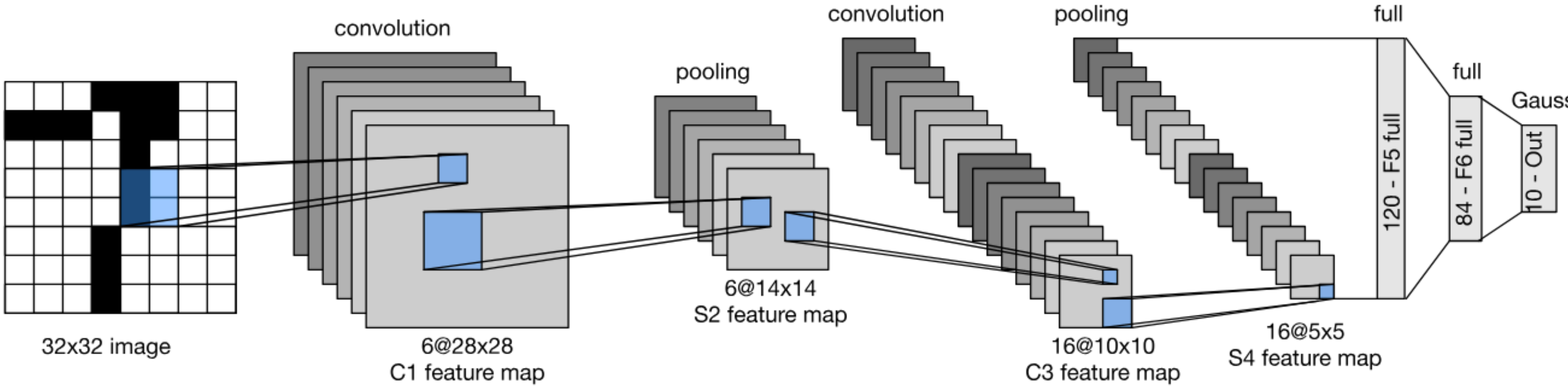
- Max pooling: the strongest pattern signal in a window
- Average pooling: replace max with mean in max pooling
  - The average signal strength in a window

Max pooling



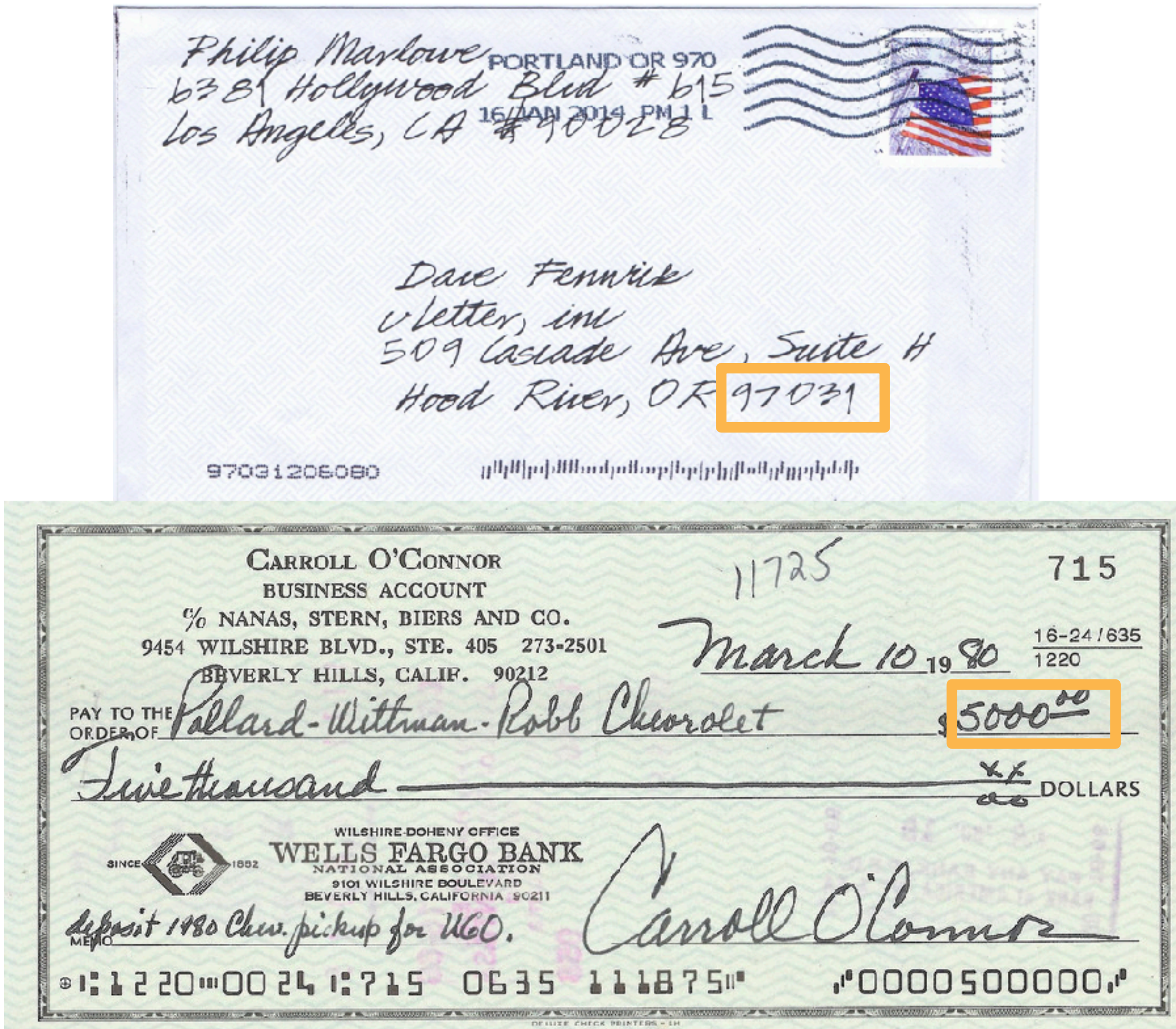
Average pooling

# LeNet Architecture



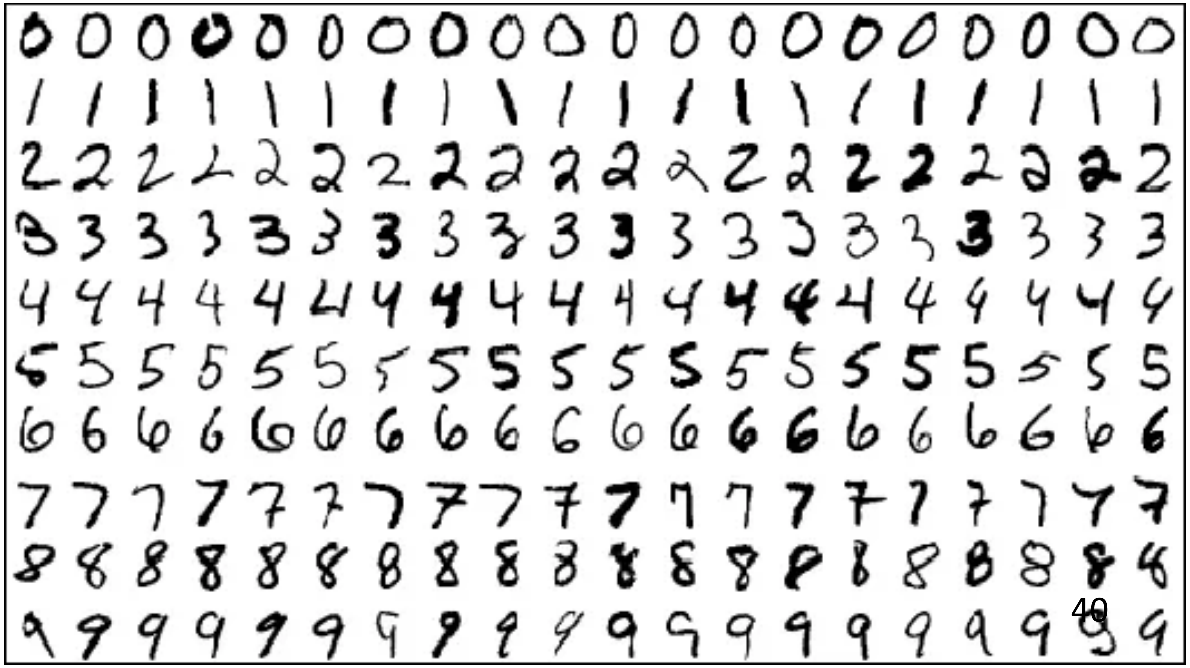
# Handwritten Digit Recognition

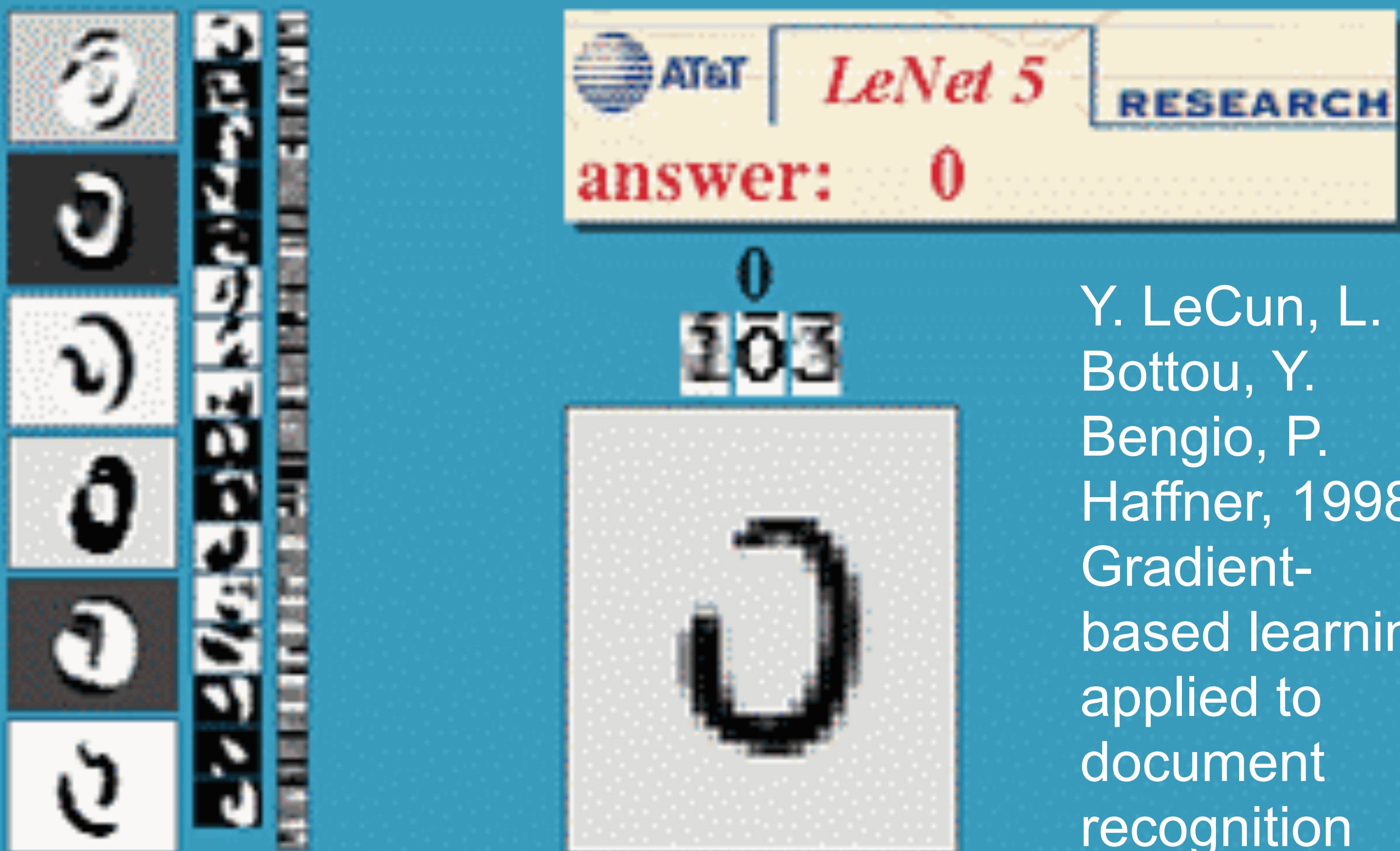
An instance of optical character recognition (OCR)



# MNIST

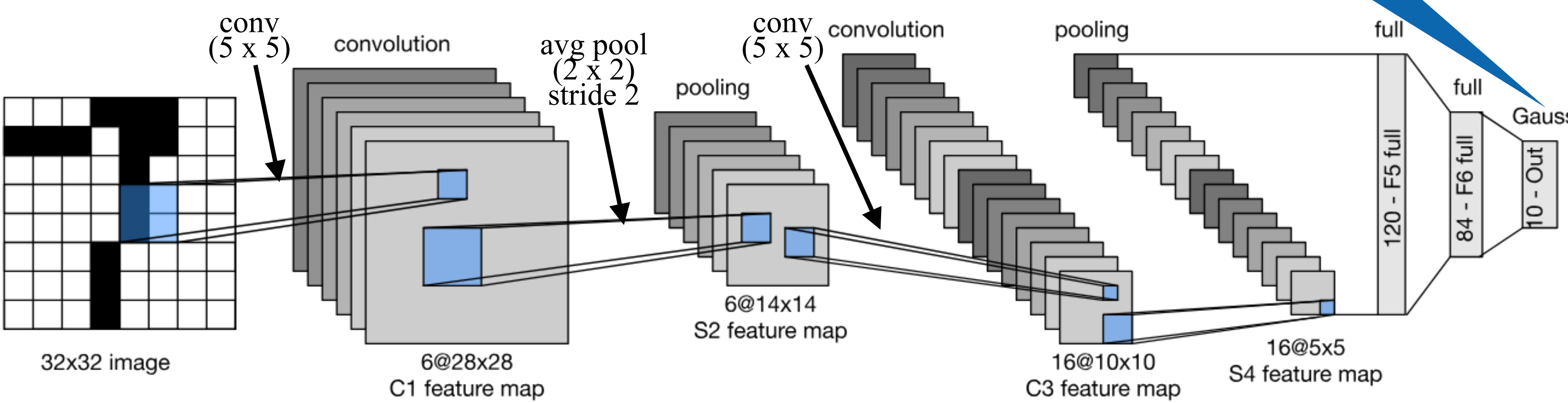
- Centered and scaled
- 50,000 training data
- 10,000 test data
- 28 x 28 images
- 10 classes






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

Expensive if we have many outputs



# LeNet-5

| Layer | #channels | kernel size | stride | activation | feature map size |
|---|---|---|---|---|---|
| Input | | | | | 32 x 32 x 1 |
| Conv 1 | 6 | 5 x 5 | 1 | tanh | 28 x 28 x 6 |
| Avg Pooling 1 | | 2 x 2 | 2 | | 14 x 14 x 6 |
| Conv 2 | 16 | 5 x 5 | 1 | tanh | 10 x 10 x 16 |
| Avg Pooling 2 | | 2 x 2 | 2 | | 5 x 5 x 16 |
| Conv 3 | 120 | 5 x 5 | 1 | tanh | 120 |
| FC 1 | | | | | 84 |
| FC 2 | | | | | 10 |

# LeNet in Pytorch

```
class LeNet(nn.Module):

  def __init__(self):
    super(LeNet, self).__init__()
    self.model = nn.Sequential(
      nn.Conv2d(in_channels = 1, out_channels = 6, kernel_size = 5, stride = 1,
padding = 0),
      nn.Tanh(),
      nn.AvgPool2d(kernel_size = 2, stride = 2),
      nn.Conv2d(in_channels = 6, out_channels = 16, kernel_size = 5, stride = 1,
padding = 0),
      nn.Tanh(),
      nn.AvgPool2d(kernel_size = 2, stride = 2),
      nn.Conv2d(in_channels = 16, out_channels = 120, kernel_size = 5, stride =
1, padding = 0),
      nn.Flatten(),
      nn.Linear(120, 84),
      nn.Tanh(),
      nn.Linear(84, 10))

  def forward(self, x):
    y = self.model(x)
    return y
```

# Recap

- 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

# Next Up

- More advanced Convolutional neural networks: ResNet