

# 10-301/601: Introduction to Machine Learning

## Lecture 19 – Convolutional Neural Networks

Henry Chai

6/2/25

# Front Matter

- Announcements
  - HW5 to be released on 6/3 (tomorrow), due 6/6 at 11:59 PM
  - Schedule change: two recitations this week
    - **Recitation on 6/4 will be a PyTorch tutorial**
    - Recitation on 6/5 will be Quiz 3 preparation

# Deep Learning

- From Wikipedia's page on Deep Learning...

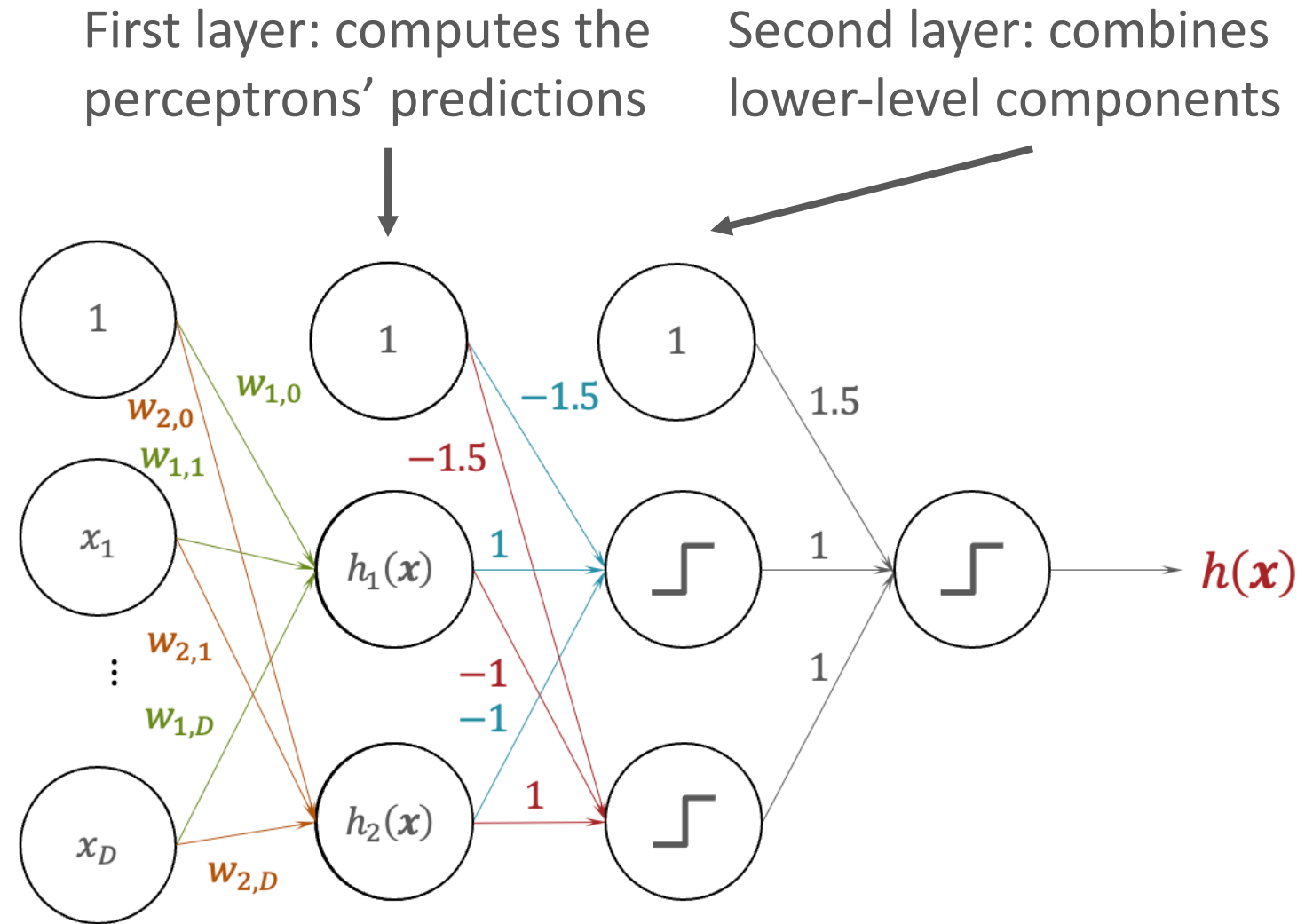
## Definition [\[ edit \]](#)

---

Deep learning is a class of [machine learning algorithms](#) that<sup>[11]</sup>(pp199–200) uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

- Deep learning = more than one layer

# Deep Learning



# Convolutional Neural Networks

- Neural networks are frequently applied to inputs with some inherent spatial structure, e.g., images
- Idea: use the first few layers to identify relevant macro-features, e.g., edges
- Insight: for spatially-structured inputs, many useful macro-features are shift or location-invariant, e.g., an edge in the upper left corner of a picture looks like an edge in the center
- Strategy: learn a filter for macro-feature detection in a small window and apply it over the entire image

# Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 \* 

0	1	0
1	-4	1
0	1	0

# Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1	0
1	-4	1
0	1	0

 $=$ 

0			

$$(0 * 0) + (0 * 1) + (0 * 0) + (0 * 1) + (1 * -4) + (2 * 1) + (0 * 0) + (2 * 1) + (4 * 0) = 0$$

# Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1	0
1	-4	1
0	1	0

 $=$ 

0	-1		

$$(0 * 0) + (0 * 1) + (0 * 0) + (1 * 1) + (2 * -4) \\ + (2 * 1) + (2 * 0) + (4 * 1) + (4 * 0) = -1$$



# Convolutional Filters

- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0





 $*$ 

0	1	0
1	-4	1
0	1	0

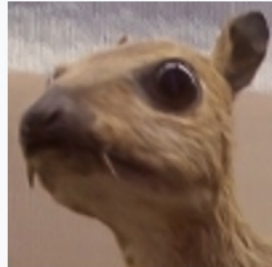
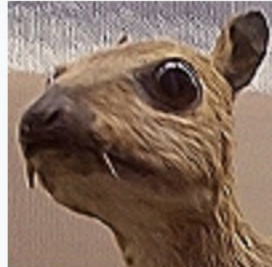
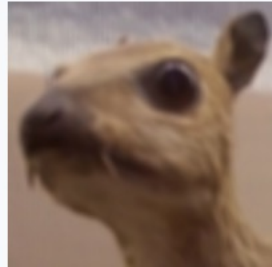
 $=$ 

0	-1	-1	0
-2	-5	-5	-2
2	-2	-1	3
-1	0	-5	0

# Convolutional Filters

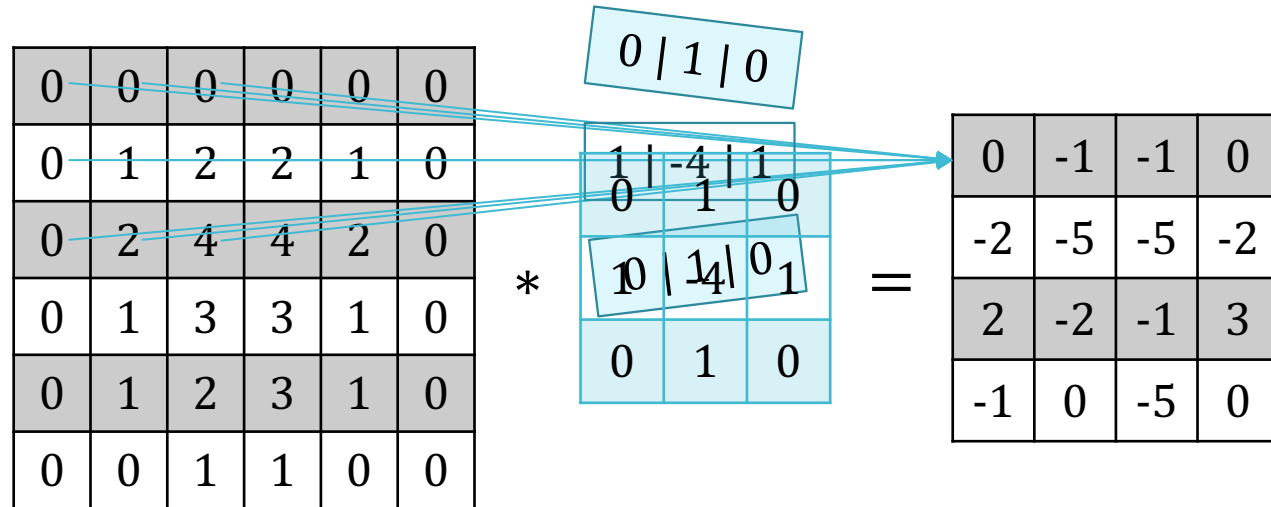
Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

# More Filters

Operation	Kernel $\omega$	Image result $g(x,y)$
<b>Identity</b>	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
<b>Sharpen</b>	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
<b>Box blur</b> (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	

# Convolutional Filters

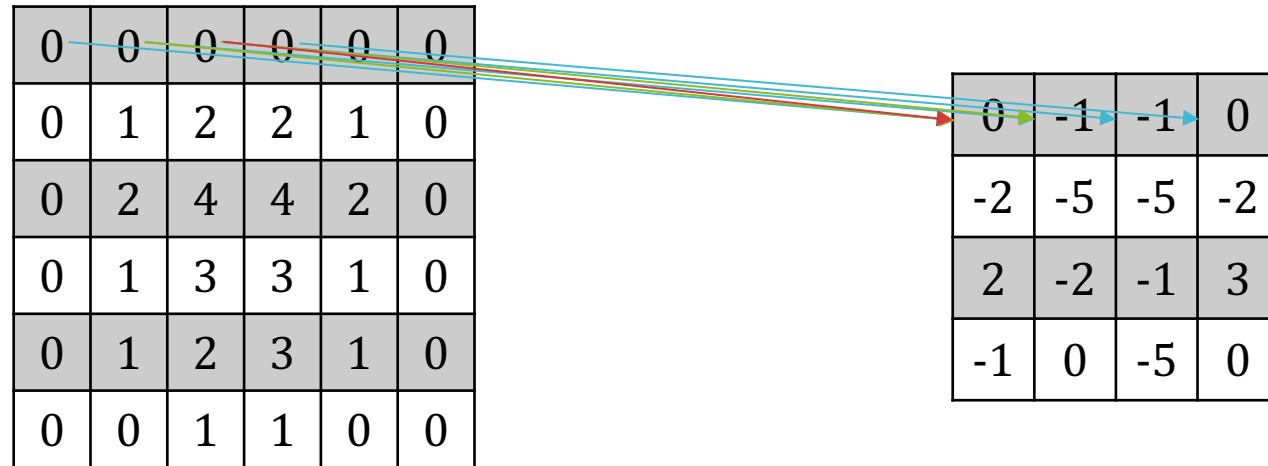
- Images can be represented as matrices, where each element corresponds to a pixel
- A filter is just a small matrix that is convolved with same-sized sections of the image matrix



# Convolutional Filters

- Convolutions can be represented by a feed forward neural network where:

1. Nodes in the input layer are only connected to some nodes in the next layer but not all nodes.
2. Many of the weights have the same value.



- Many fewer weights than a fully connected layer!
- Convolution weights are learned using gradient descent/backpropagation, not prespecified

# Convolutional Filters: Padding

- What if relevant features exist at the border of our image?
- Add zeros around the image to allow for the filter to be applied “everywhere” e.g. a *padding* of 1 with a 3x3 filter preserves image size and allows every pixel to be the center

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	1	2	2	1	0	0
0	0	2	4	4	2	0	0
0	0	1	3	3	1	0	0
0	0	1	2	3	1	0	0
0	0	0	1	1	0	0	0
0	0	0	0	0	0	0	0

 $*$ 

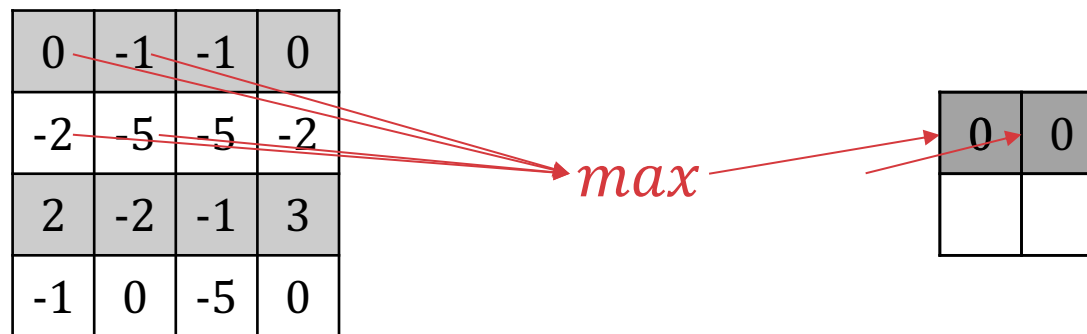
0	1	0
1	-4	1
0	1	0

 $=$ 

0	1	2	2	1	0
1	0	-1	-1	0	1
2	-2	-5	-5	-2	2
1	2	-2	-1	3	1
1	-1	0	-5	0	1
0	2	-1	0	2	0

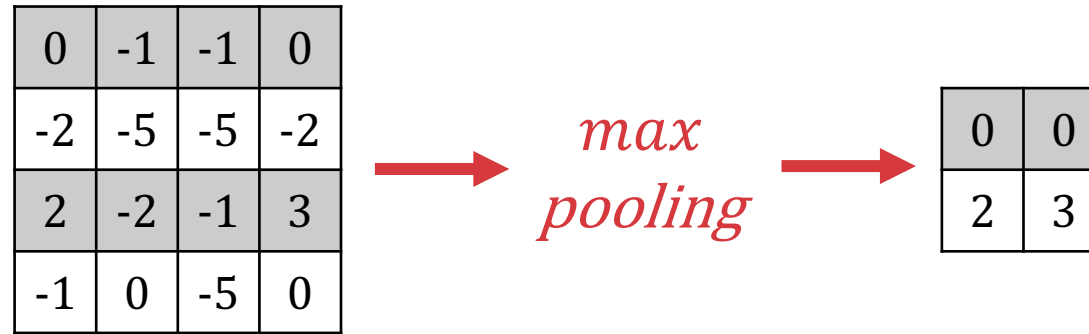
# Downsampling: Pooling

- Combine multiple adjacent nodes into a single node



# Downsampling: Pooling

- Combine multiple adjacent nodes into a single node



- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
  - Protects the network from (slightly) noisy inputs



# Downsampling: Stride

- Only apply the convolution to some subset of the image  
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1
1	-2

 $=$ 

-2		

# Downsampling: Stride

- Only apply the convolution to some subset of the image  
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1
1	-2

 $=$ 

-2	-2	

# Downsampling: Stride

- Only apply the convolution to some subset of the image  
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1
1	-2

 $=$ 

-2	-2	1

# Downsampling: Stride

- Only apply the convolution to some subset of the image  
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

 $*$ 

0	1
1	-2

 $=$ 

-2	-2	1
0		

# Downsampling: Stride

- Only apply the convolution to some subset of the image  
e.g., every other column and row = a *stride* of 2

0	0	0	0	0	0
0	1	2	2	1	0
0	2	4	4	2	0
0	1	3	3	1	0
0	1	2	3	1	0
0	0	1	1	0	0

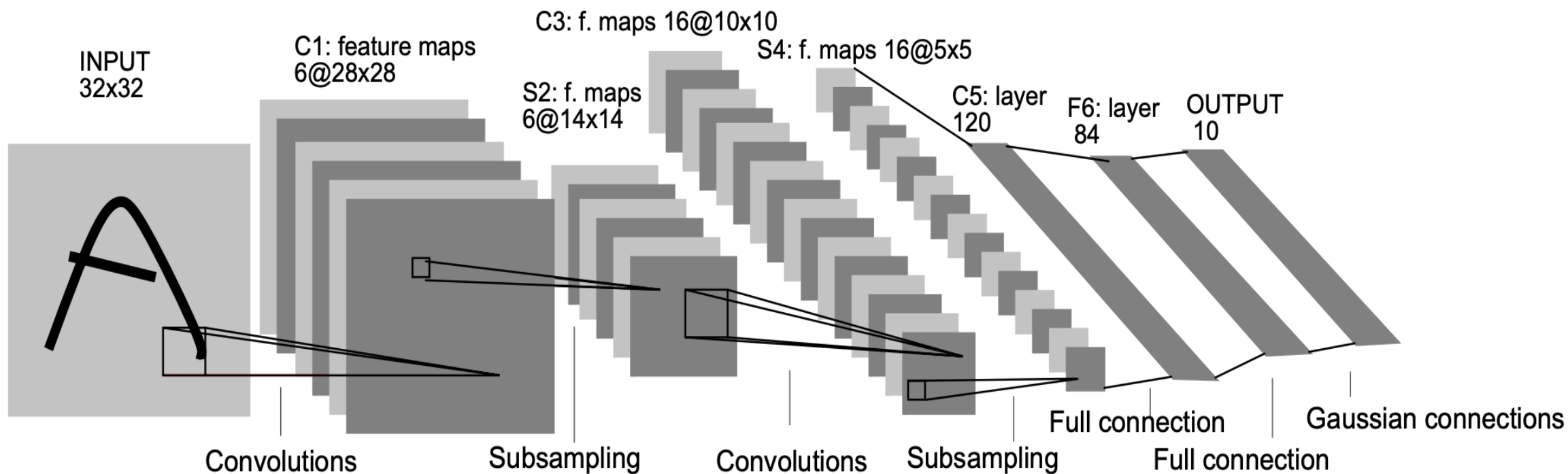
 $*$ 

0	1
1	-2

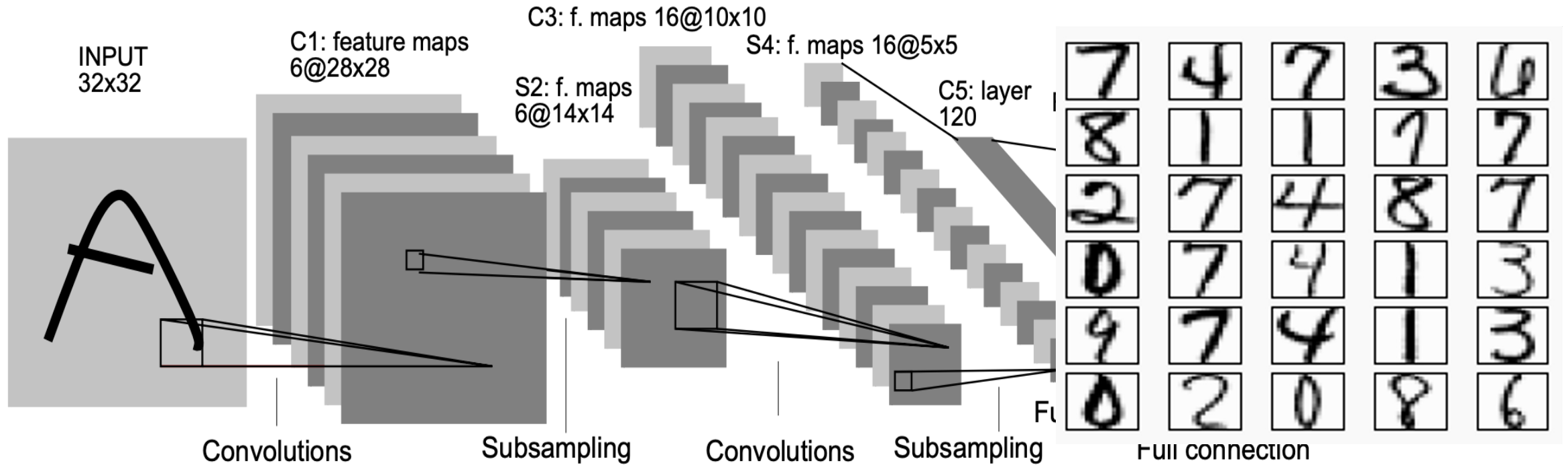
 $=$ 

-2	-2	1
0	1	1
1	2	0

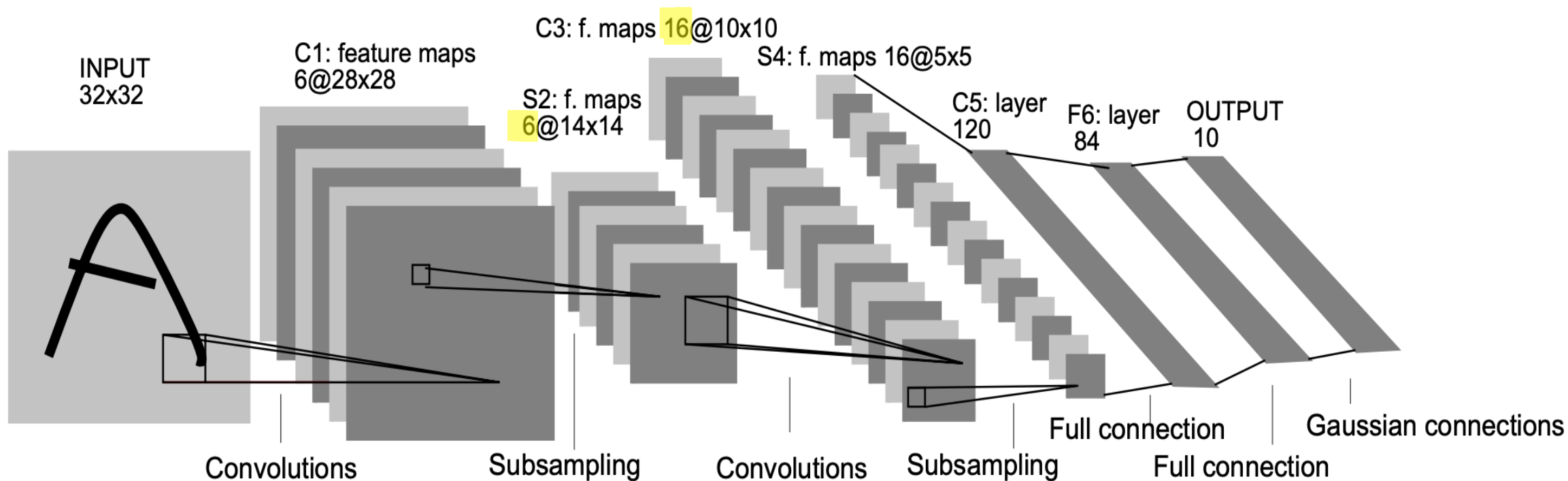
- Reduces the dimensionality of the input to subsequent layers and thus, the number of weights to be learned
- Many relevant macro-features will tend to span large portions of the image, so taking strides with the convolution tends not to miss out on too much



# LeNet (LeCun et al., 1998)

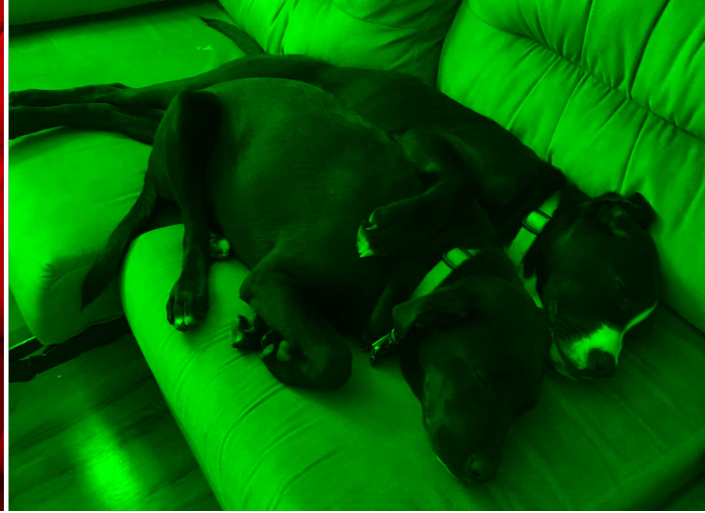
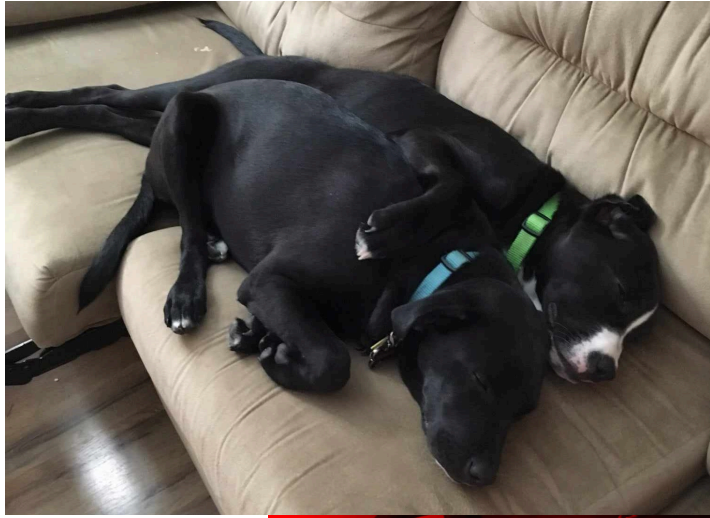


- One of the earliest, most famous deep learning models – achieved remarkable performance at handwritten digit recognition ( $< 1\%$  test error rate on MNIST)
- Used sigmoid (or logistic) activation functions between layers and mean-pooling, both of which are pretty uncommon in modern architectures

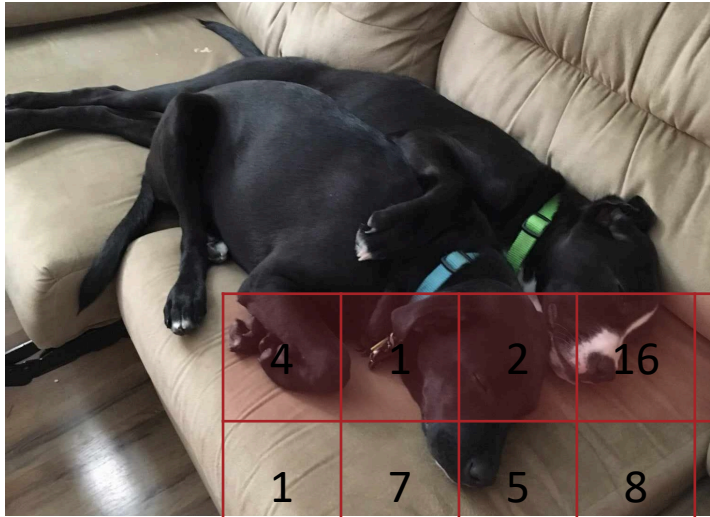


# Wait how did we go from 6 to 16?





# Channels



4	1	2	16	3	6
1	7	5	8	19	27
5	2	5	12	17	8
0	4	9	9	6	11

5	2	6	14	15	8
26	3	6	8	4	9
0	15	24	6	1	8
7	4	9	5	24	17

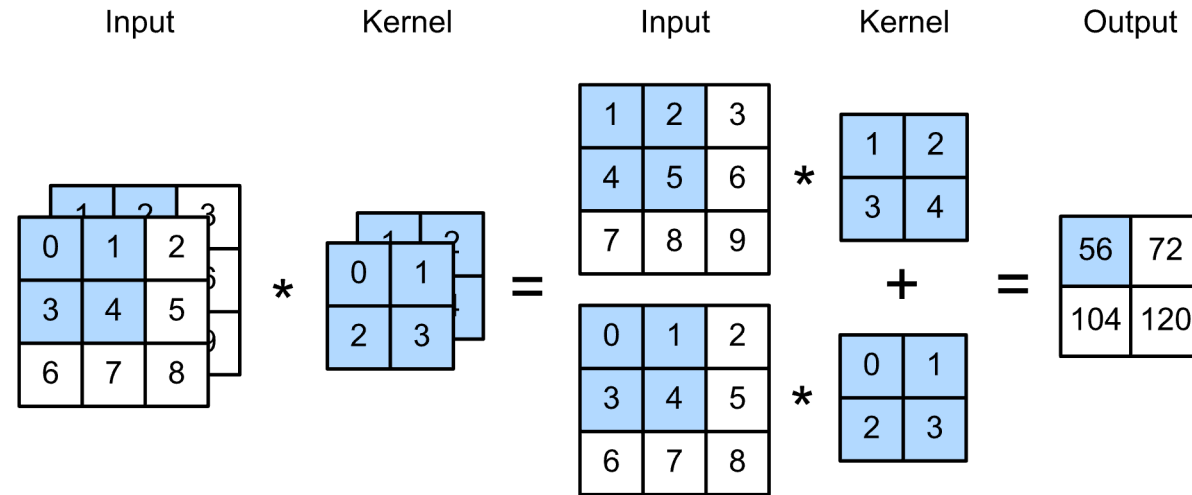
4	6	8	9	5	3
16	5	2	8	2	1
5	2	14	11	7	8
15	2	5	0	9	8

- An image can be represented as the sum of red, green and blue pixel intensities
- Each color corresponds to a *channel*



# Convolutions on Multiple Input Channels

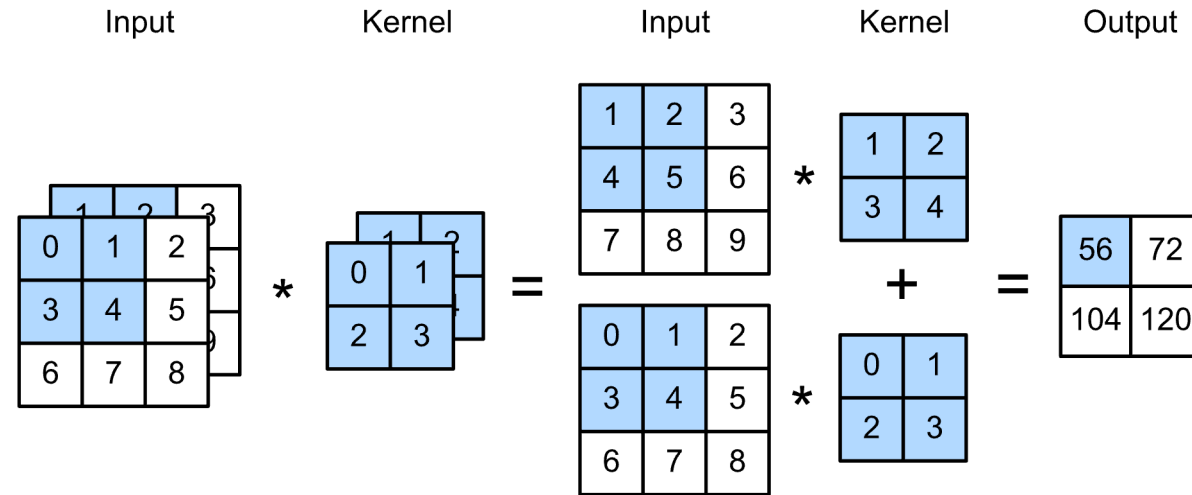
- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor



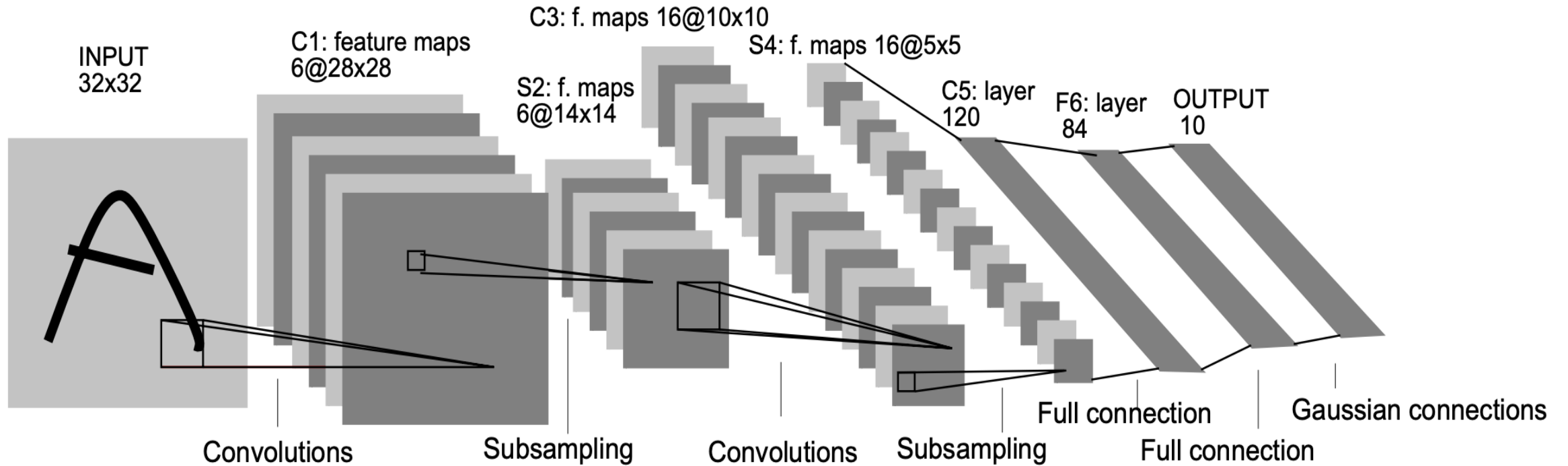
- For  $c$  channels and  $h \times w$  filters, we have  $chw + c$  learnable parameters (each filter has a bias term)

# Convolutions on Multiple Input Channels

- Given multiple input channels, we can specify a filter for each one and sum the results to get a 2-D output tensor



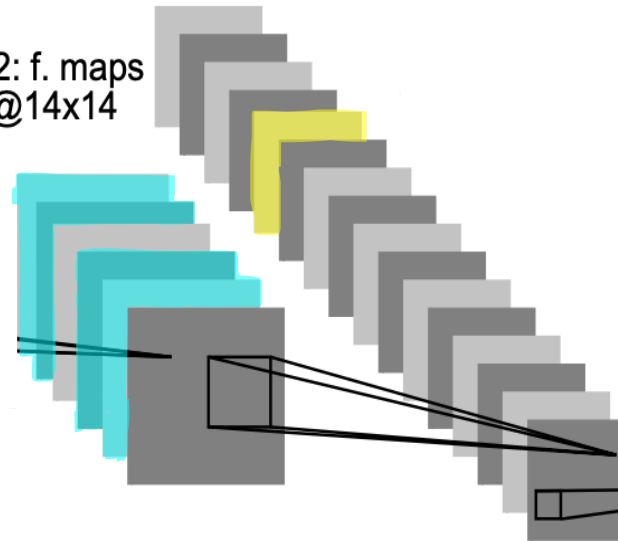
- Questions:
  - Why might we want a different filter for each input?
  - Why do we combine them together into a single output channel?



- Channels in hidden layers correspond to different macro-features, which we might want to manipulate differently → one filter per channel

C3: f. maps 16@10x10

S2: f. maps  
6@14x14



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

TABLE I

EACH COLUMN INDICATES WHICH FEATURE MAP IN S2 ARE COMBINED BY THE UNITS IN A PARTICULAR FEATURE MAP OF C3.

- We can combine these macro-features into a new, interesting, “higher-level” feature
  - But we don’t always need to combine all of them!
  - Different combinations → multiple output channels
  - Common architecture: more output channels and smaller outputs in deeper layers

# Key Takeaways

- Convolutional neural networks use convolutions to learn macro-features
  - Can be thought of as slight modifications to the fully-connected feed-forward neural network
  - Can still be learned using SGD
  - Padding is used to preserve spatial dimensions while pooling, stride and  $1 \times 1$  convolutions are used to downsample intermediate representations
  - Channels are used to represent color; different filters are learned on different (combinations) of channels