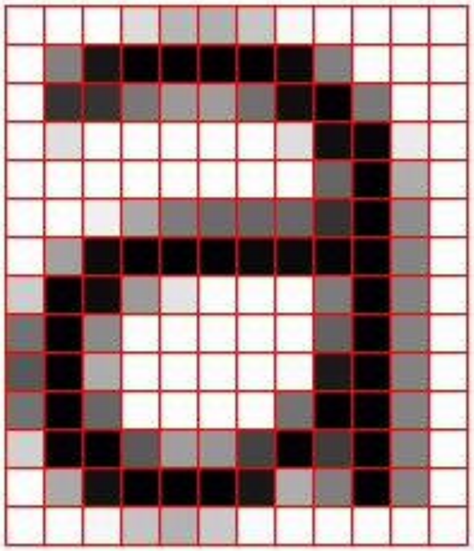
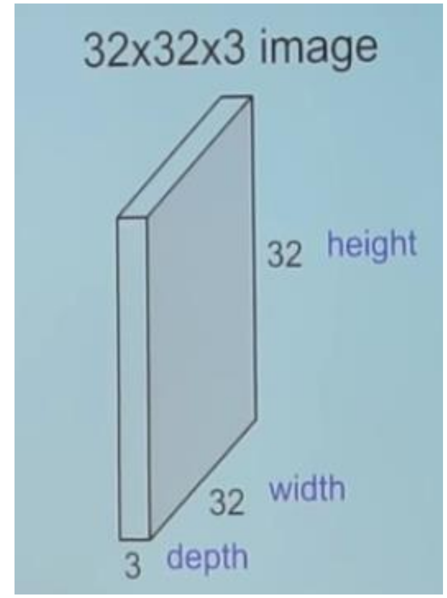
Convolutional Neural Network

Matrix representation of a picture

a



1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0



What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What Computers See

Image Classification

Training …

$$\mathbf{h} = \sigma(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{o} = \mathbf{W}_2^T\mathbf{h} + \mathbf{b}_2$$
$$\mathbf{y} = \mathrm{softmax}(o)$$

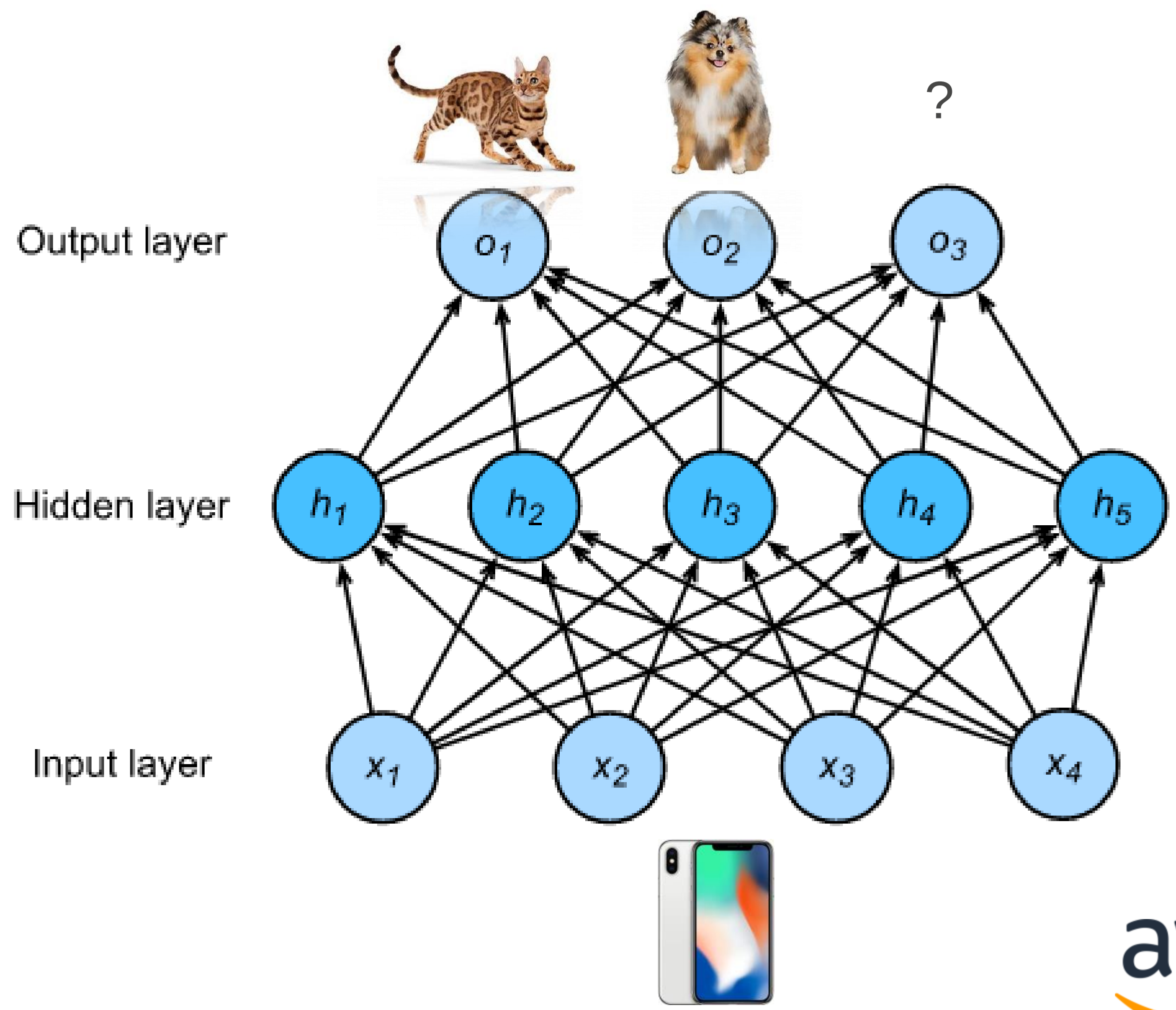
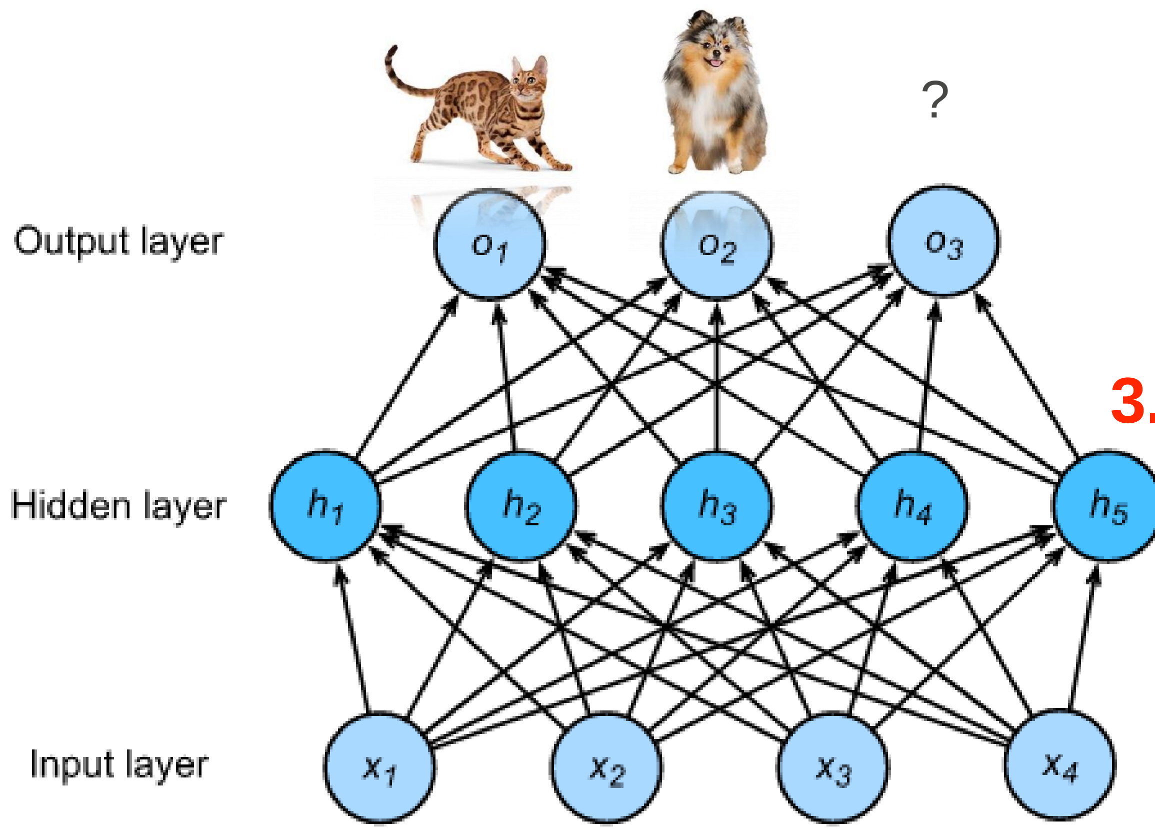


Image Classification



100 neurons

3.6B parameters
=
14GB

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

Can we reduce the number of parameters a bit?

Yes! Convolutions!

Where is Waldo?

Where is Waldo?

Two Principles

1. Translation Invariance:

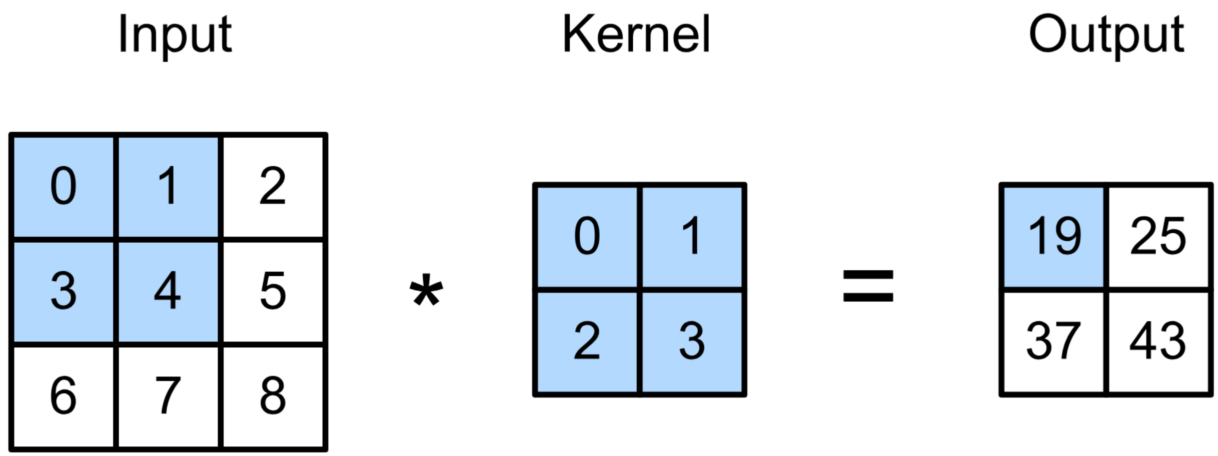
Our vision systems should, in some sense, **respond similarly to the same object regardless of where it appears** on the image.

Two Principles

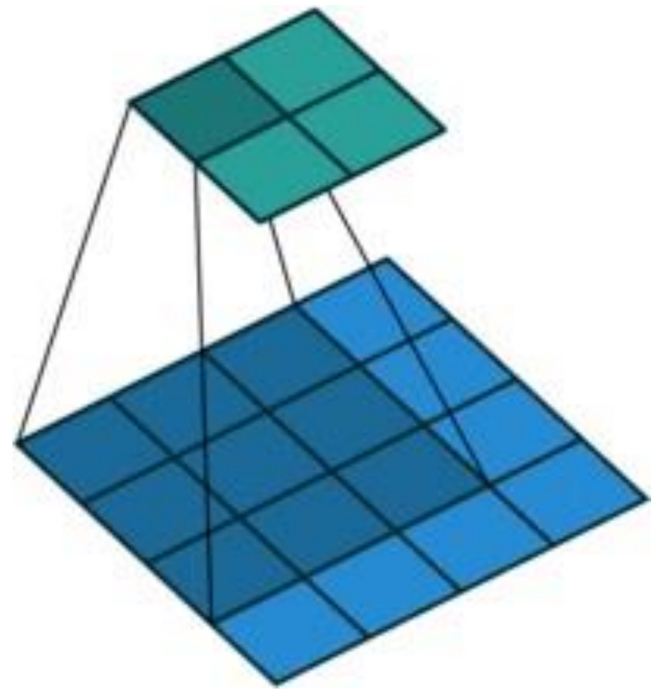
<u>2. Locality:</u>

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 Layer



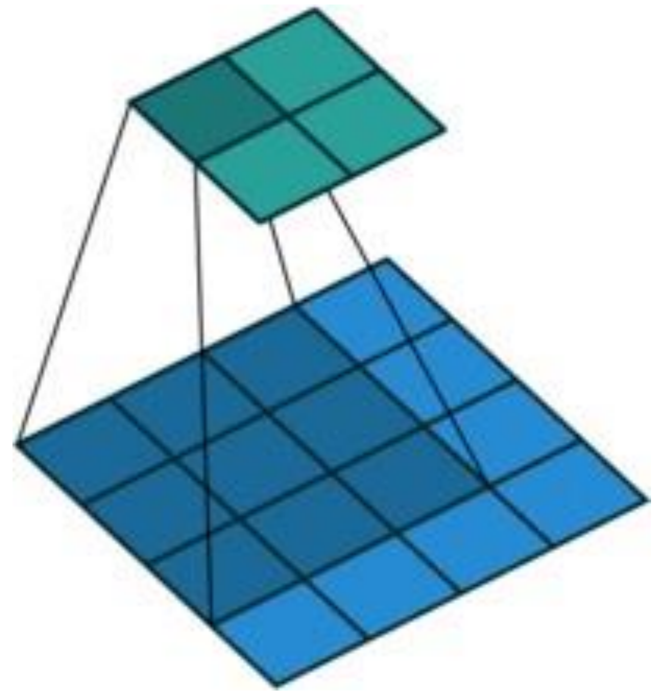
$$
\begin{aligned}
0\times0+1\times1+3\times2+4\times3&=19,\\
1\times0+2\times1+4\times2+5\times3&=25,\\
3\times0+4\times1+6\times2+7\times3&=37,\\
4\times0+5\times1+7\times2+8\times3&=43.
\end{aligned}
$$



(vdumoulin@ Github)

2-D Convolution Layer

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



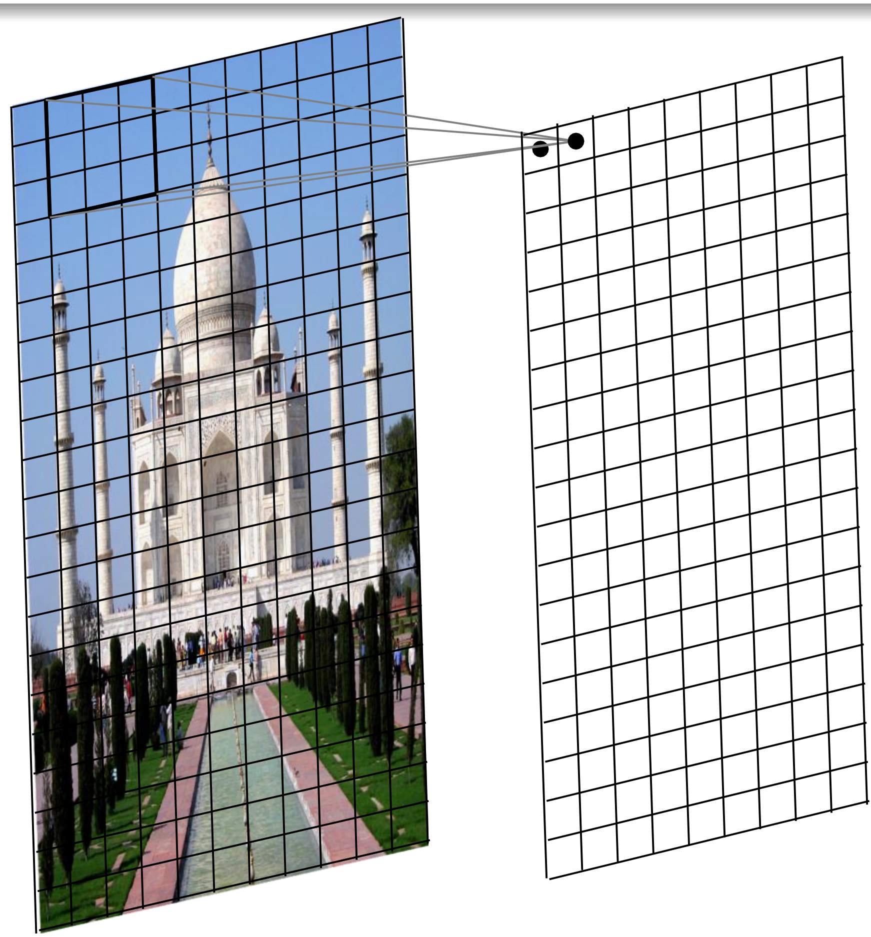
- $\mathbf{X}$: $n_h \times n_w$ input matrix
- $\mathbf{W}$: $k_h \times k_w$ kernel matrix
- : the bias scalar
- $\mathbf{Y}$: $(n_h - k_h + 1) \times (n_w - k_w + 1)$ output matrix

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

(vdumoulin@ Github)

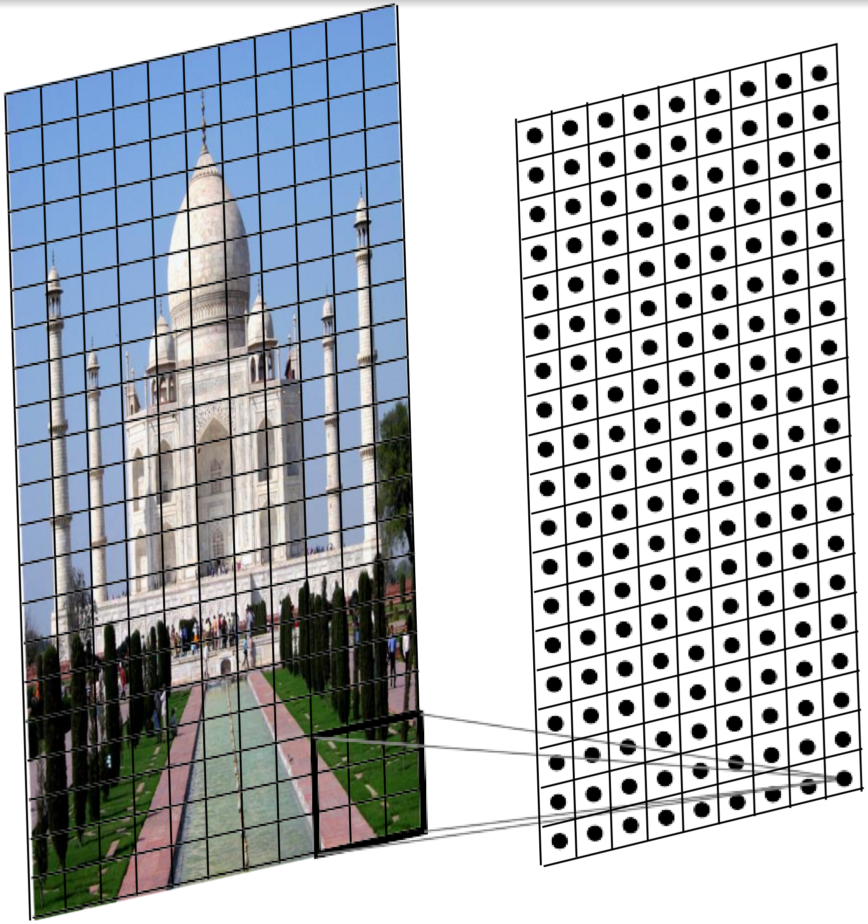
- and are the trainable parameters

Convolution



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output

2D convolutions applied to images



- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

Fliters

(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

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

*

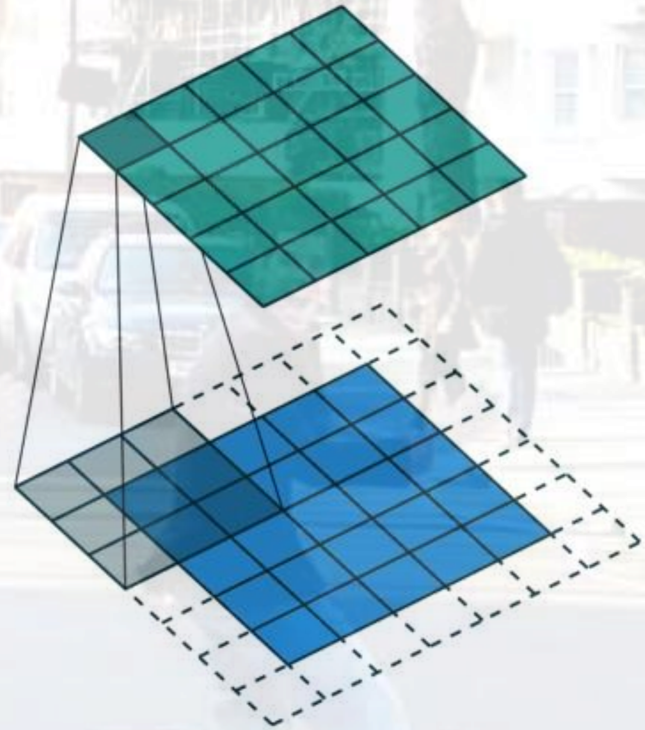
Kernel

0	1
2	3

=

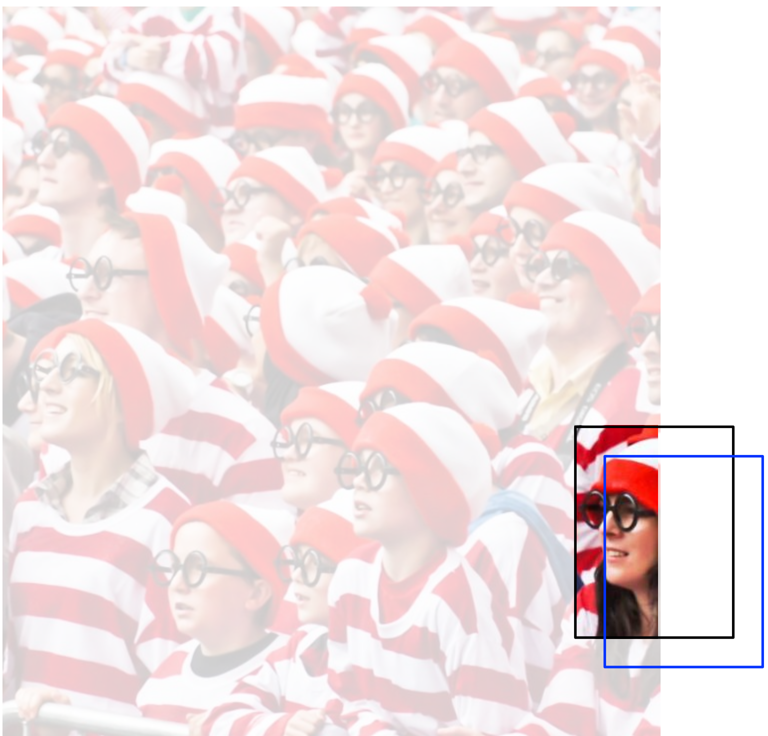
Output

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



$$0\times0+0\times1+0\times2+0\times3=0$$

How about two nearly identical windows?



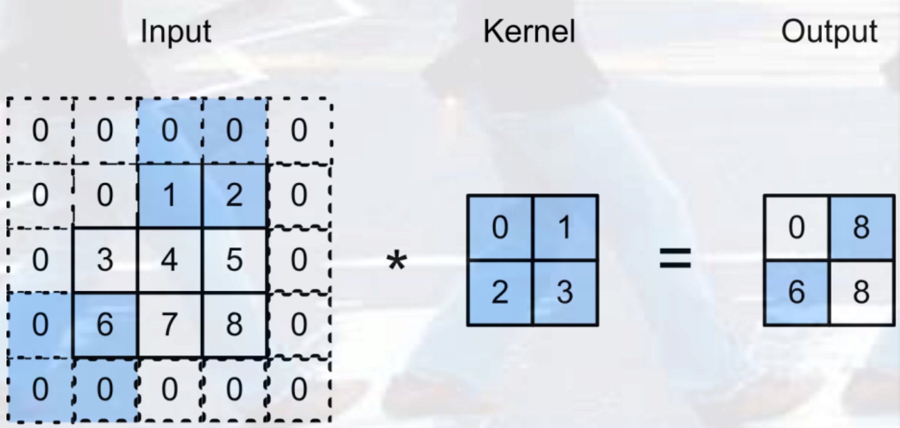
The original convolution window may be too computational expensive to slide one pixel at a time…

Stride can help!

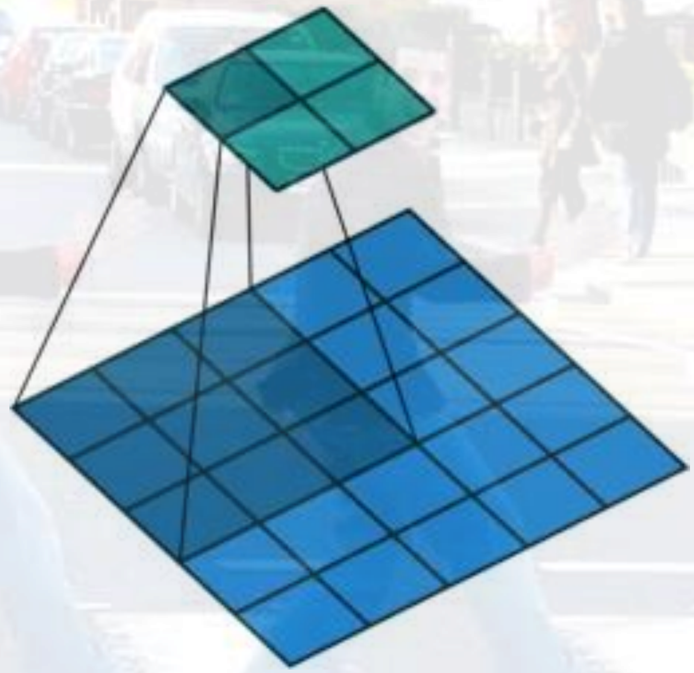
Stride

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

Strides of 3 for height and 2 for width



$$0\times0+0\times1+1\times2+2\times3=8$$
$$0\times0+6\times1+0\times2+0\times3=6$$



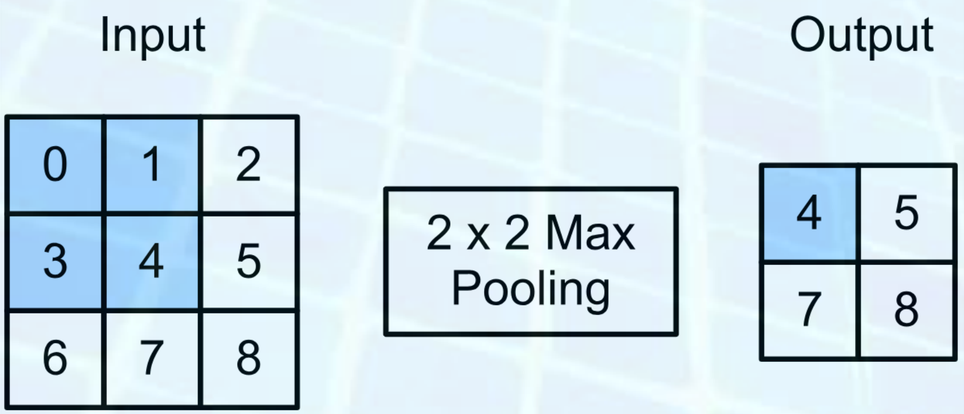
How to calculate the shape of the output?

- 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

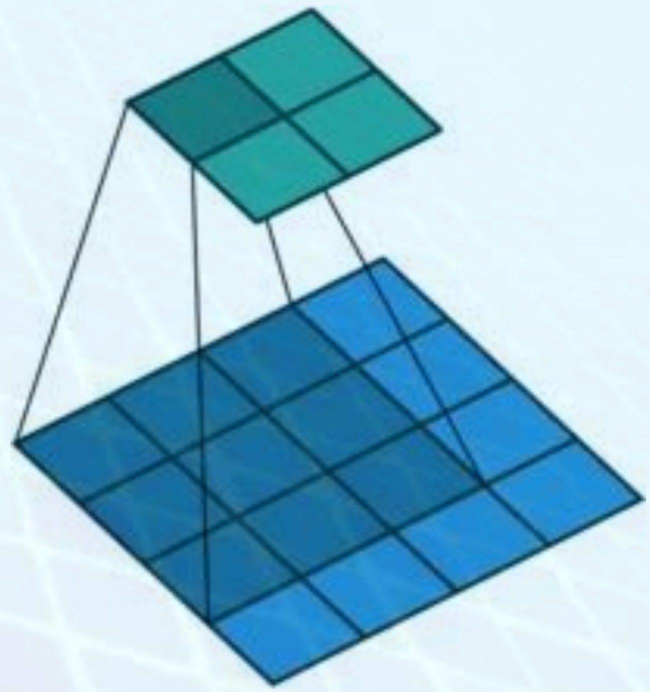
$$\lfloor (n_h - k_h + p_h + s_h)/s_h \rfloor \times \lfloor (n_w - k_w + p_w + s_w)/s_w \rfloor$$

Max Pooling

- Returns the maximal value in the pooling window

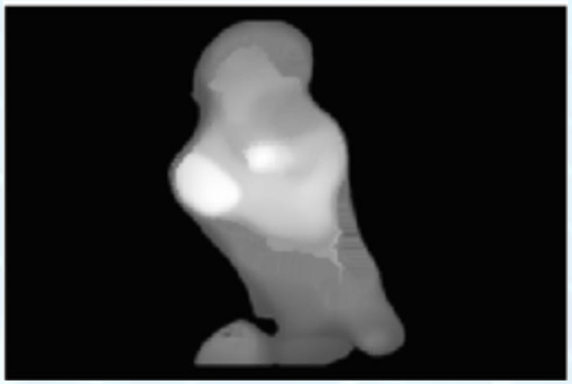


$$\max(0,1,3,4) = 4$$



Average Pooling

Max pooling

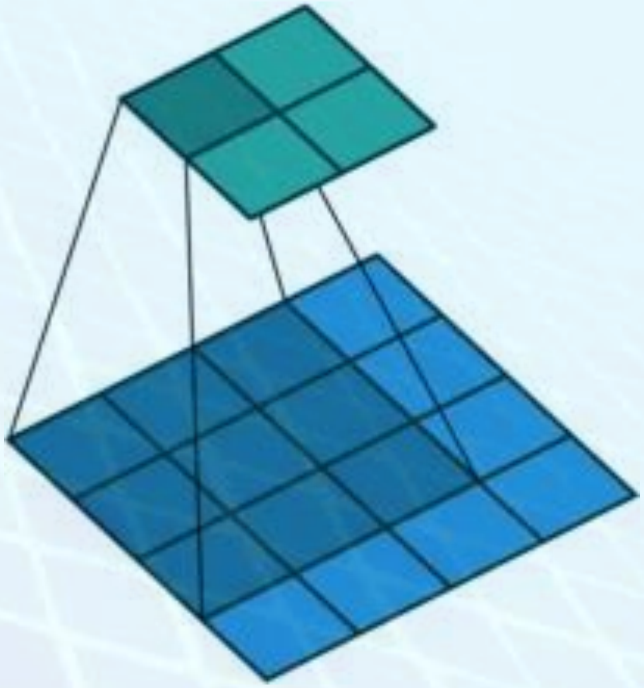


Average pooling

- 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

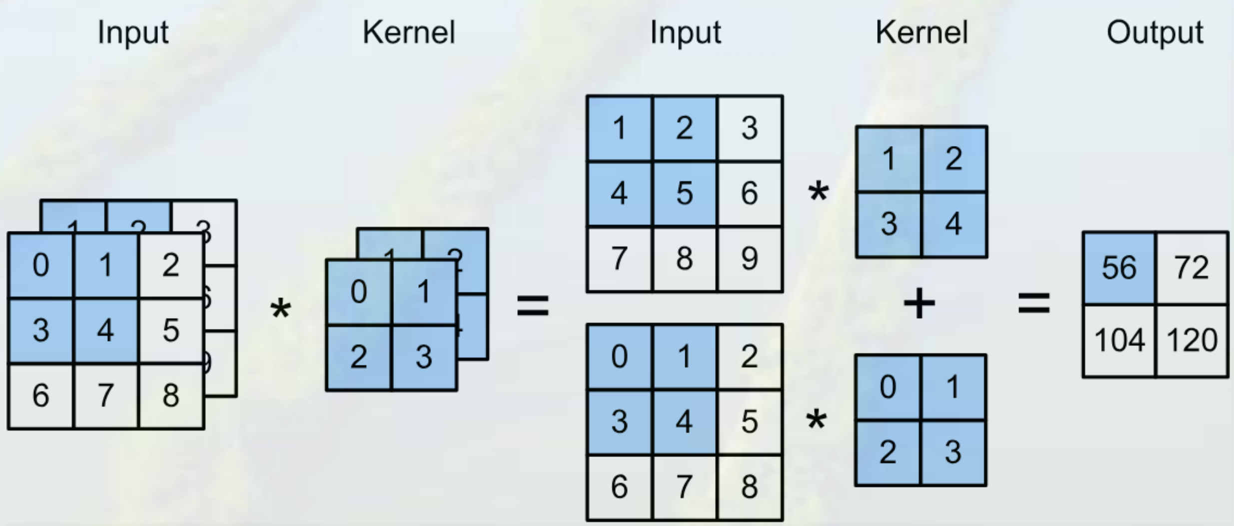


Multiple Input Channels

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

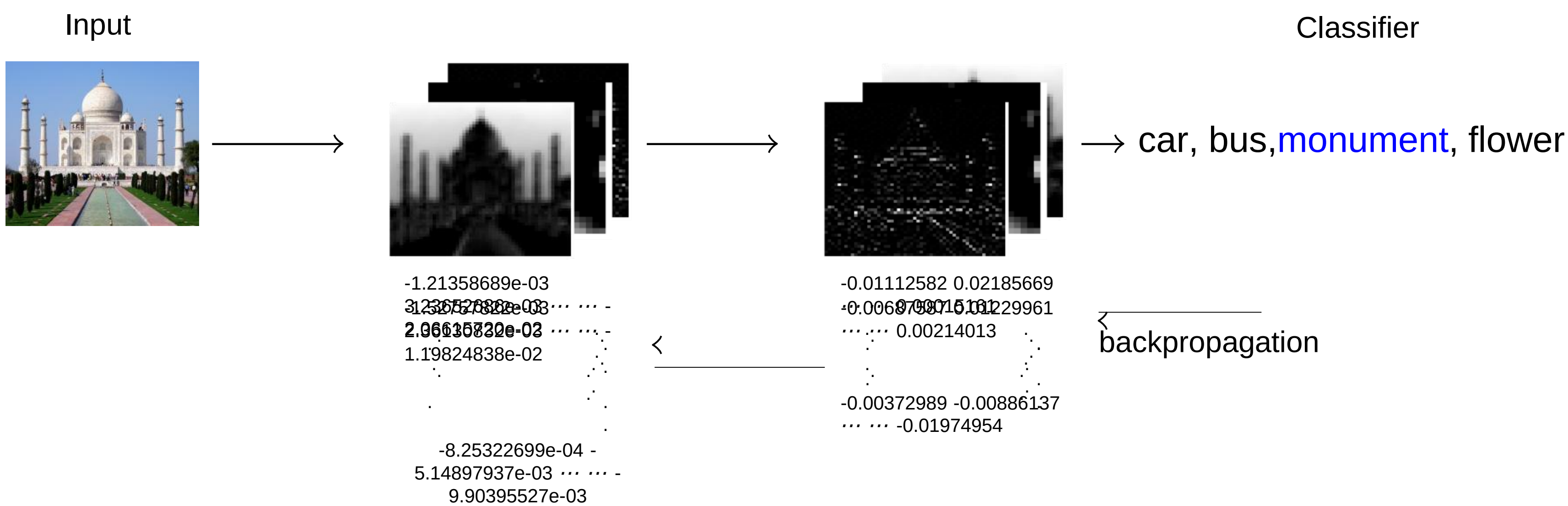
Multiple Input Channels

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



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

Convolutional Neural Network



- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

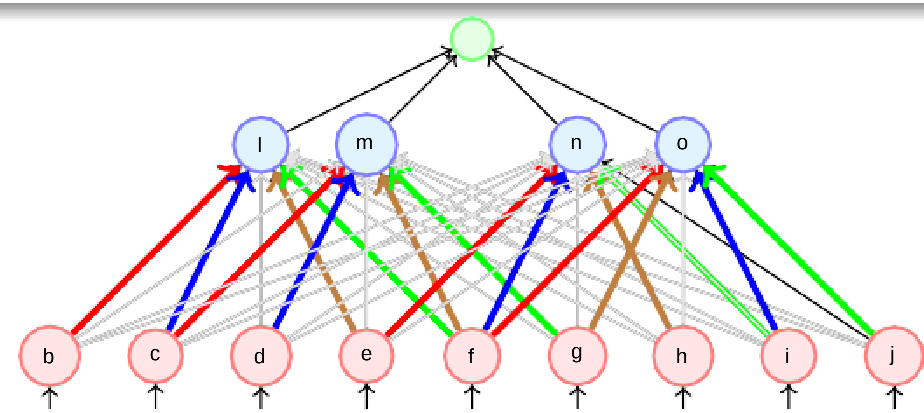
Training CNN

Input

b	c	d
e	f	g
h	i	j

Kerne
l

w	x
y	z



- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

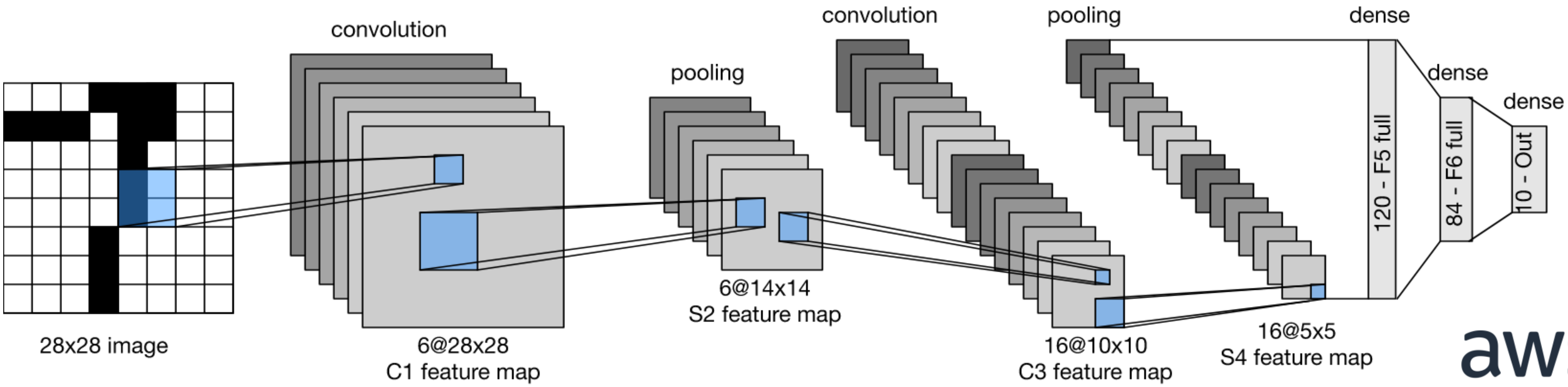
- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections

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

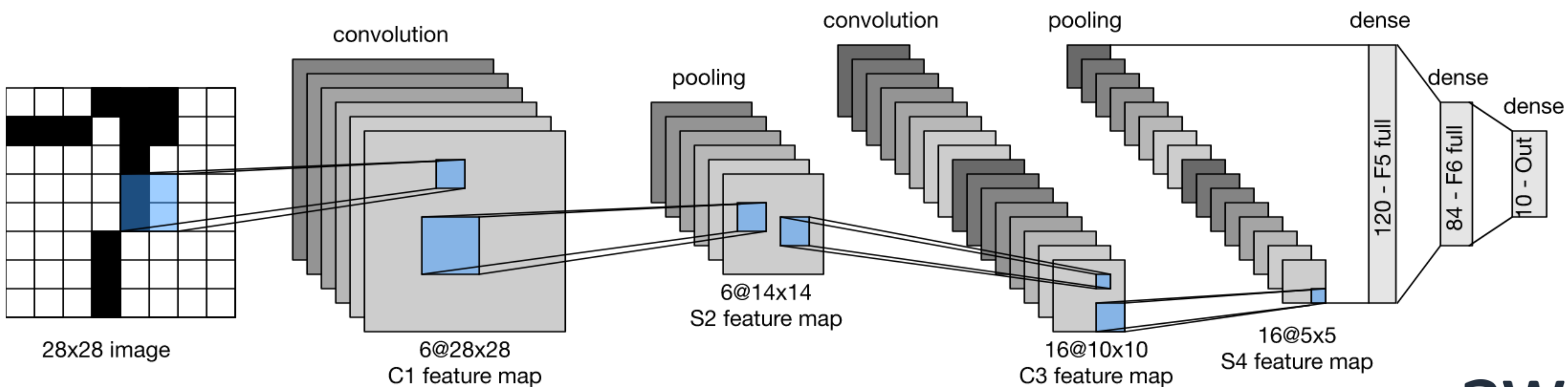


LeNet

LeNet consists of two parts:

Part II. Fully-connected layers Block

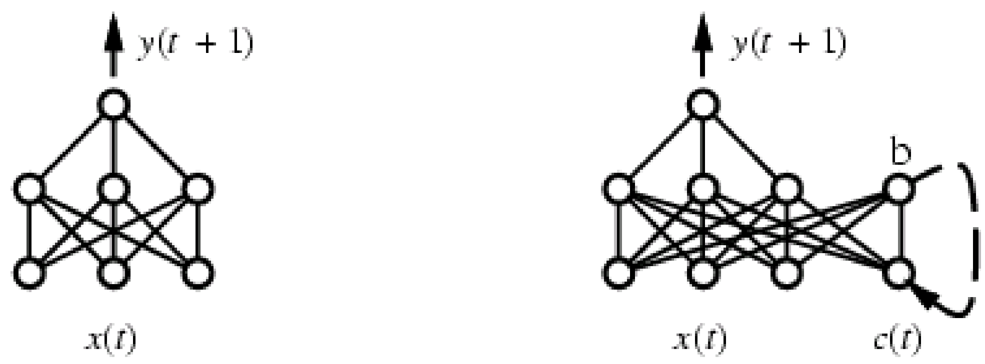
- 3 fully-connected layers
  - with 120, 84, and 10 outputs, respectively



Recurrent Networks: Time Series

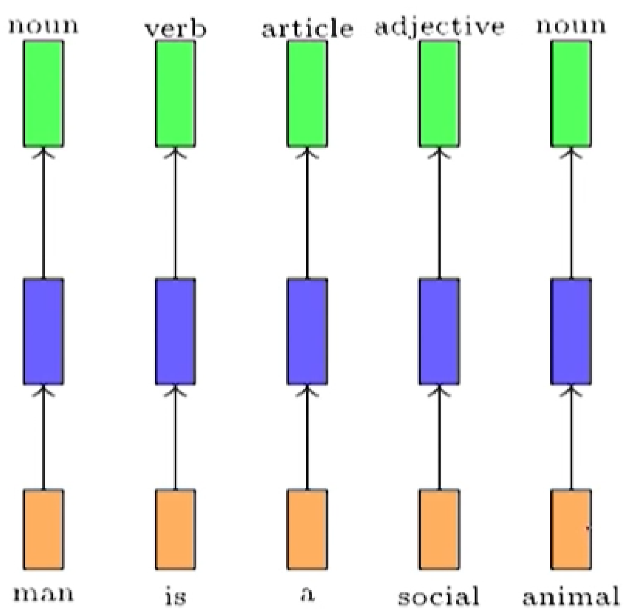
- Suppose we want to predict next state of world
  - and it depends on history of unknown length
  - e.g., robot with forward-facing sensors trying to predict next sensor reading as it moves and turns

- Idea: use hidden layer in network to capture state history



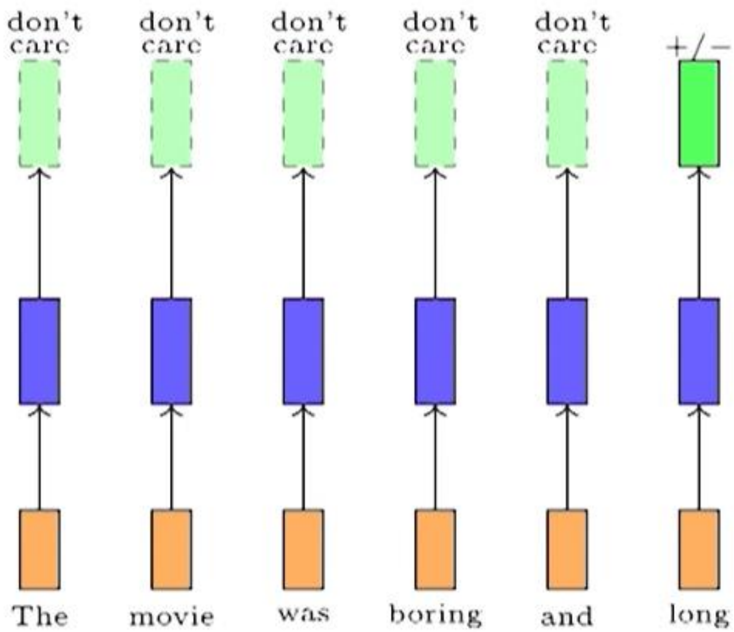
(a) Feedforward network (b) Recurrent network

Recurrent Neural Network



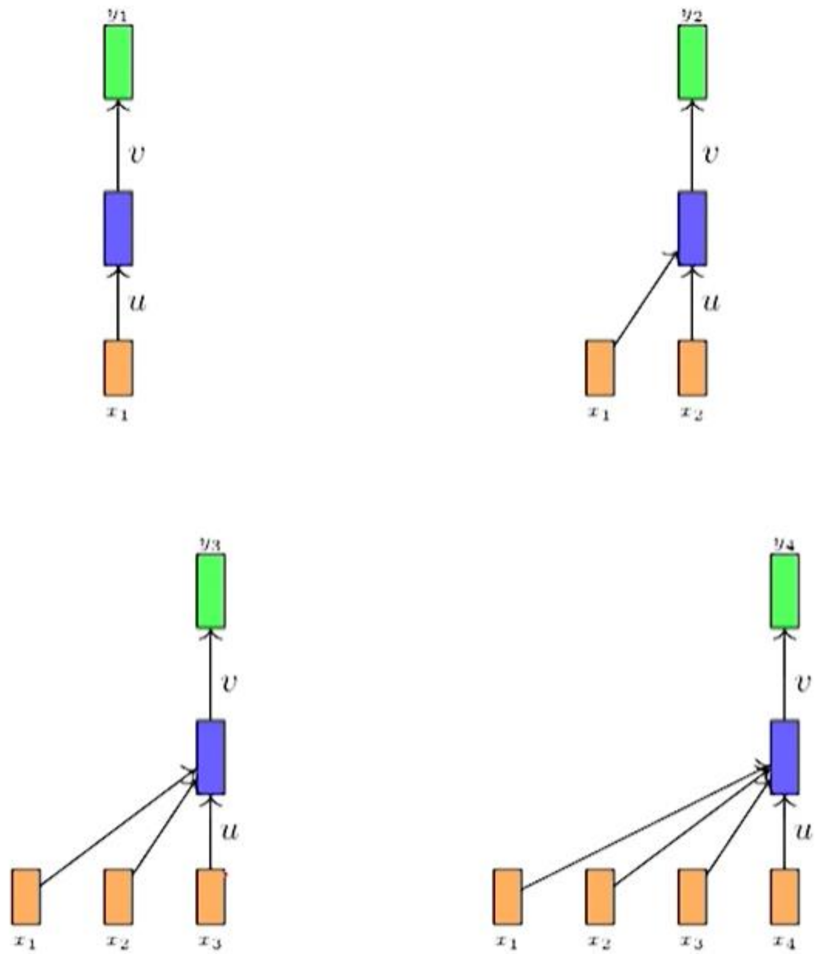
- Consider the task of predicting the part of speech tag (noun, adverb, adjective verb) of each word in a sentence
- Once we see an adjective (social) we are almost sure that the next word should be a noun (man)
- Thus the current output (noun) depends on the current input as well as the previous input
- Further the size of the input is not fixed (sentences could have arbitrary number of words)
- Notice that here we are interested in producing an output at each time step
- Each network is performing the same task (**input** : word, **output** : tag)

Recurrent Neural Network



- Sometimes we may not be interested in producing an output at every stage
- Instead we would look at the full sequence and then produce an output
- For example, consider the task of predicting the polarity of a movie review
- The prediction clearly does not depend only on the last word but also on some words which appear before
- Here again we could think that the network is performing the same task at each step (input : word, output : +/−) but it's just that we don't care about intermediate outputs

Recurrent Neural Network

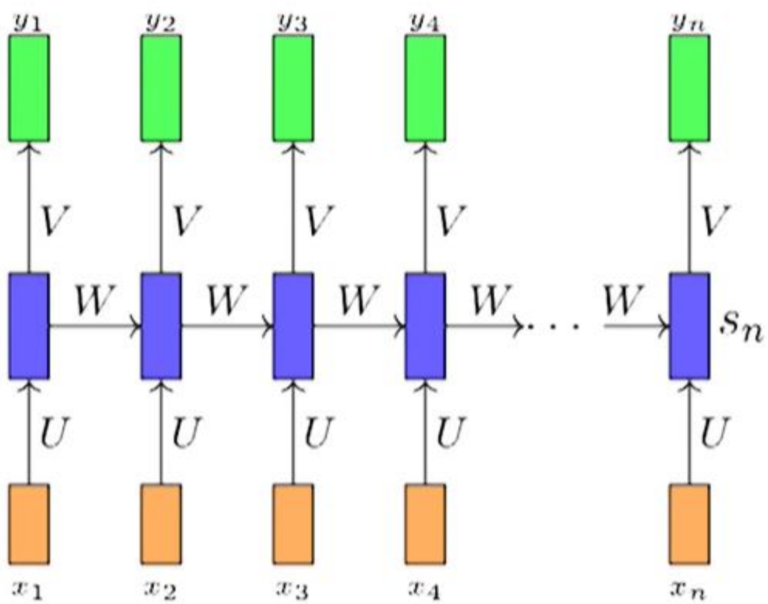


- First, the function being computed at each time-step now is different

$$
\begin{aligned}
y_1 &= f_1(x_1) \\
y_2 &= f_2(x_1, x_2) \\
y_3 &= f_3(x_1, x_2, x_3)
\end{aligned}
$$

- The network is now sensitive to the length of the sequence
- For example a sequence of length 10 will require $f_1, \ldots, f_{10}$ whereas a sequence of length 100 will require $f_1, \ldots, f_{100}$

Recurrent Neural Network



- The solution is to add a recurrent connection in the network,

$$
\begin{aligned}
s_i &= \sigma(Ux + Ws_{i-1} + b) \\
y_i &= \sigma(Vs_i + c) \\
&\quad or \\
y_i &= f(x_i, s_i, W, U, V)
\end{aligned}
$$

- s_i is the state of the network at timestep i
- The parameters are W, U, V which are shared across timesteps
- The same network (and parameters) can be used to compute $y_1, y_2, \ldots, y_{10}$ or y_{100}

Artificial Neural Networks: Summary

- Highly non-linear regression/classification
- Hidden layers learn intermediate representations
- Potentially millions of parameters to estimate

- Deep networks have produced real progress in many fields
  - computer vision
  - speech recognition
  - mapping images to text
  - recommender systems
  - …
- They learn very useful non-linear representations

References

Mitesh Khapra
https://www.youtube.com/watch?v=yw8xwS15Pf4

Visualization of CNN
https://www.youtube.com/watch?v=cNBBNAxC8l4

Back propagation
https://www.youtube.com/watch?v=G5b4jRBKNxw&list=PLZbbT5o_s2xq7LwI2y8_QtvuXZedL6tQU&index=25

Dive into deeplearning
https://c.d2l.ai/gtc2020/