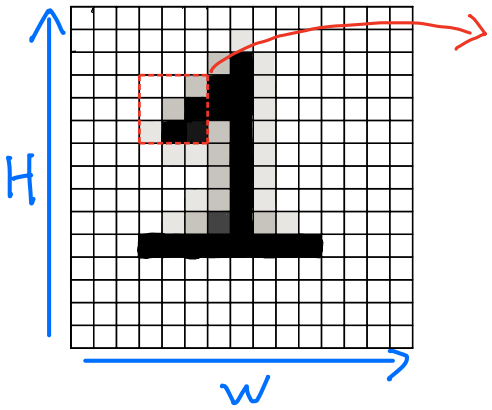
Representing Images



255	255	127
255	127	0
192	0	62

- Images composed of pixel features

- Typically 8-bit values 0 – 255 (int)

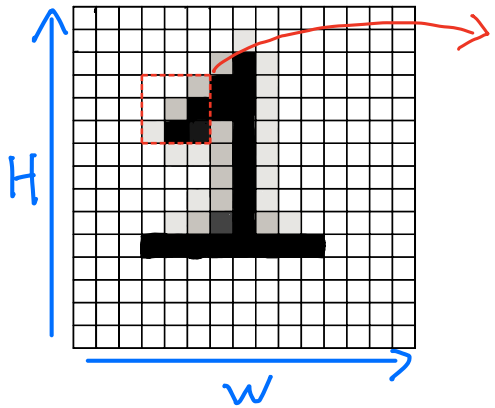


0 → Black 255 → White

Single obs x is a matrix: $H \times W$

Batch $\underline{X}$ is a 3-D Tensor: $N \times H \times W$

Representing Images



255	255	127
255	127	0
192	0	62

0 to 1

good

1.	1.	0.5
1.	0.5	0.
0.7	0.	0.2

-1 to 1

better

1.	1.	0.
1.	0.	-1.
1.2	-1.	-.5

For Neural networks Want to convert to float and rescale

Norm

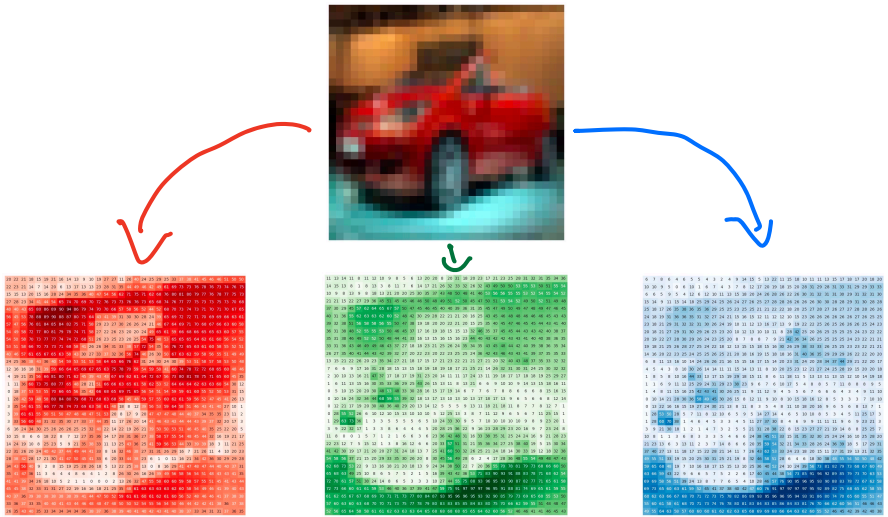
best!

[But data set Specific :]

3.1	3.1	0.1
3.1	0.1	-3.
1.4	-3.	-1.1

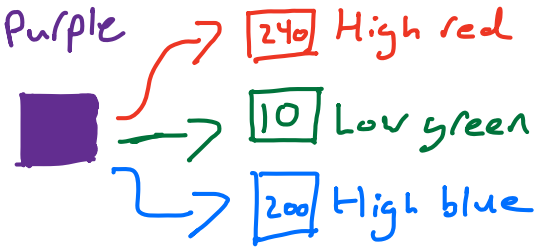
RGB Images

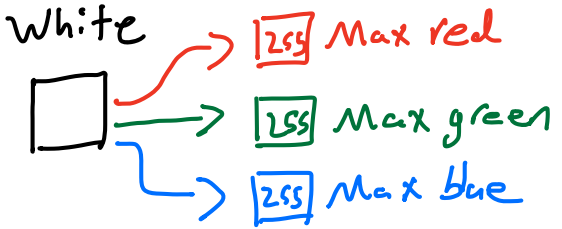
Represent color pixels w/ 3-Channels
[3-Values per pixel]

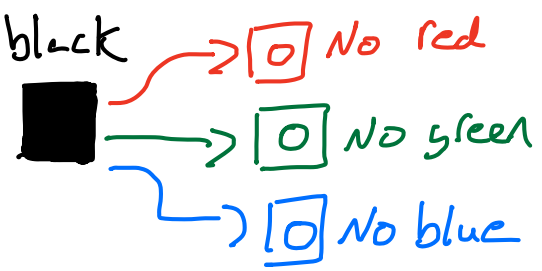


Single obs. x is 3-D tensor: $H \times W \times 3$

Batch X is 4-D tensor: $N \times H \times W \times 3$

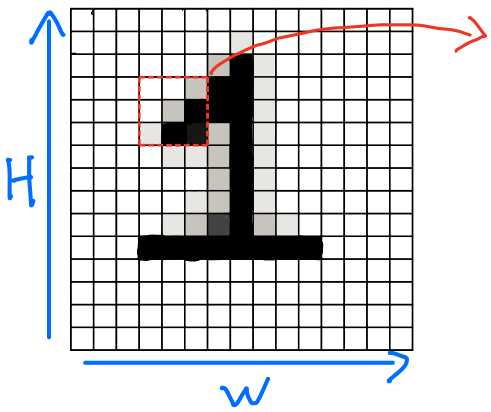






Representing Images (so far)

For standard neural network Reshape into vector



255	255	127
255	127	0
192	0	62

Single obs. Pixels

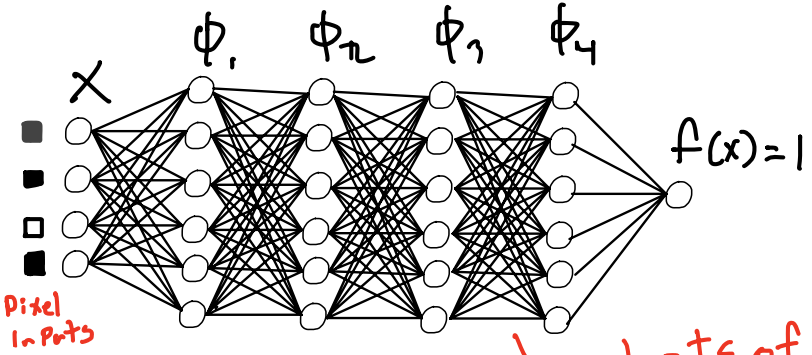
x : HW

255	255	127	255	127	0	192	0	64

Pixels

Batch Pixels

X : N x HW



Lots of parameters!

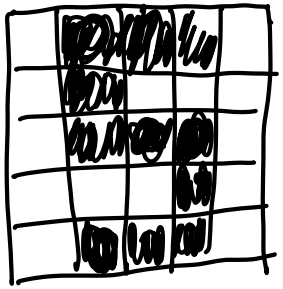
Image 1	255	255	127	255	127	0	192	0	64
Image 2	255	255	127	255	127	0	192	0	64
Image 3	255	255	127	255	127	0	192	0	64
⋮	255	255	127	255	127	0	192	0	64

N

Pixels

Convolutional Layers — Neural nets for Images

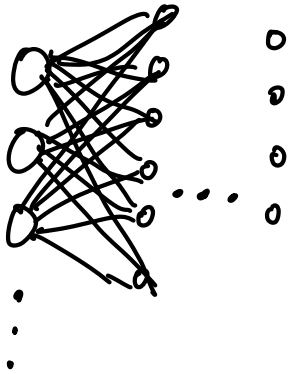
Basic approach for image inputs:



Image

Each pixel becomes an input





"Flatten the image"

Problems with this approach

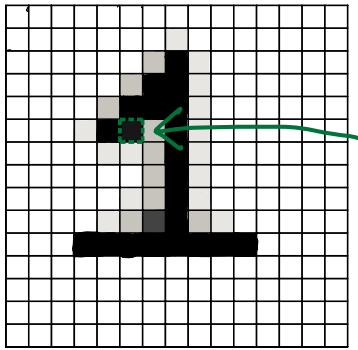
1) Scale
   - Every neuron is a weighted sum over all W x h pixels (could be millions for each neuron!)

2) Structure
   - Removes the spatial structure of images

3) Translation sensitive
   - Conversly, can't assume pixel locations are fixed!

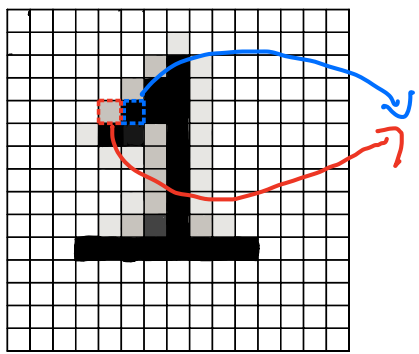
Image Structure

- Class determined by relationships between features

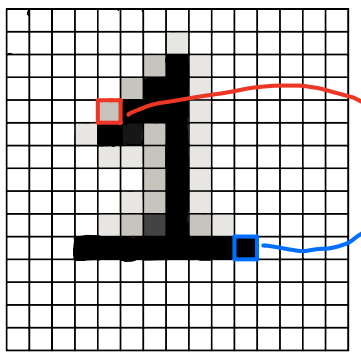


One pixel has little info by itself!

- Relationships between nearby pixels are more important than far pixels



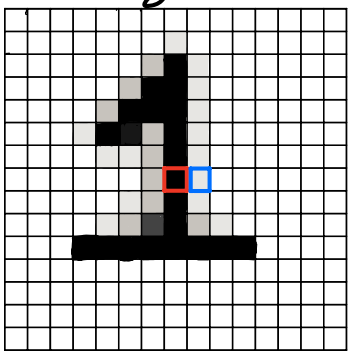
There is an edge here



?

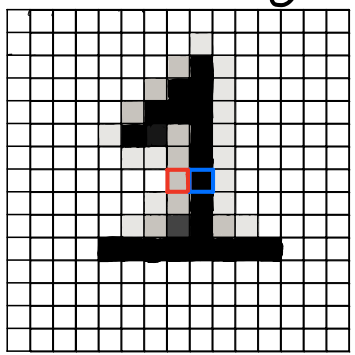
Image Structure

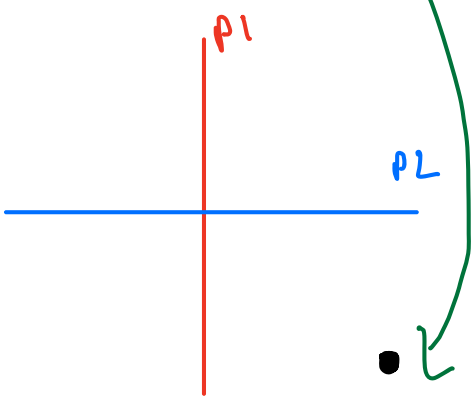
original

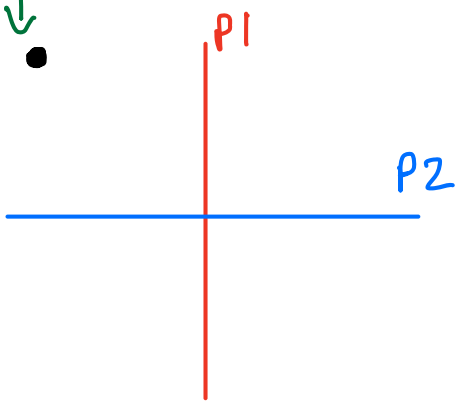


Translation should't change prediction

shifted right







But features can change a lot!

How does our brain handle Vision?

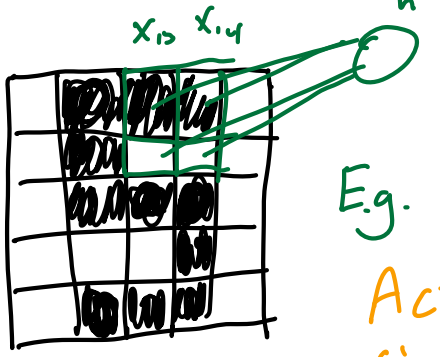
Visual system has neurons that respond to specific patterns:

E.g.



Edges points colors

We can replicate this idea with perceptrons



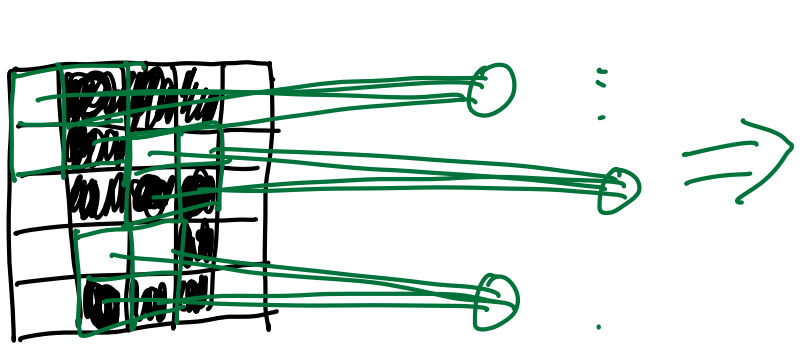
1 neuron is only connected to a small window of the input

E.g. $h = \sigma(W_1 x_{13} + W_2 x_{14} + W_3 x_{23} + W_4 x_{24} + b)$

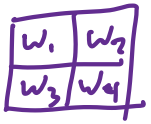
Activates if weights an input have similar pattern!

Replicate the same neuron across the image!

- That is, share the same weights for every neuron
- Activate on the same pattern anywhere



We call this replicated pattern of weights a kernel



Does this help our issues?

Scale: for 5x5 image, 16 neurons ✓

- Before: 25 inputs x 16 neurons = 400 weights & connections!
- Now: 4 inputs x 16 neurons = 4 weights & 64 connections!

Spatial structure ✓

- output retains spatial structure! we didn't flatten

Translation sensitivity ✓

- A neuron will fire for a pattern anywhere in the image!

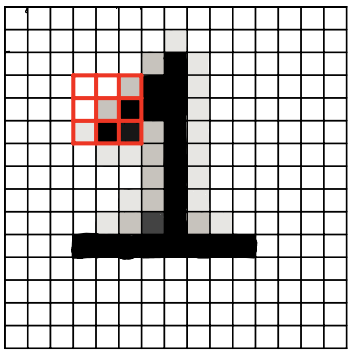
Convolutional Networks (CNNs)

1) Maintain Image structure
   (Don't flatten)

2) Shift weights to find best alignment

3) Make network sparse
   (Remove weights)

Convolutions in 2-D



Align kernel in every 2-d location

$$\text{Conv2d}(x, k)_{ij} = \sum_{a=1}^{s} \sum_{b=1}^{s} x_{i+a, j+b} \cdot k_{ab}$$

[For padding = 'valid']

5	-1	3	8	4	6	-2
-3	4	9	1	1	7	-4
5	-6	3	2	1	-2	0
0	-8	5	5	4	-2	3
4	7	2	-7	3	1	4
4	-3	1	2	6	1	-1
0	0	3	-2	3	-1	0

-1	-2	-1
0	0	0
1	2	1

			1	

$$= 1 \cdot (-1) + 1 \cdot (-2) + 7 (-1) + 2 \cdot 0 + 1 \cdot 0 +$$
$$(-2) \cdot 0 + 5 \cdot 1 + 4 \cdot 2 + (-2) \cdot 1$$

Convolutions in 2-D (Blur)

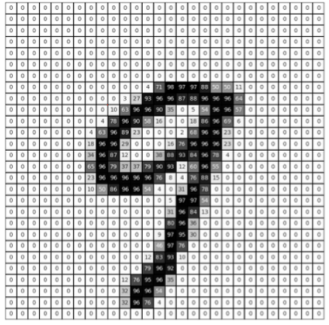
Small Blur

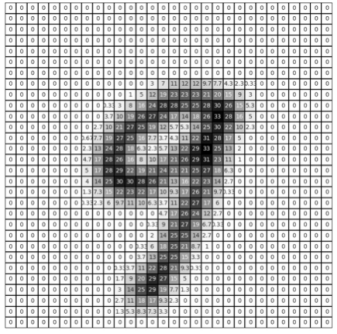
0.12	0.12	0.12
0.12	0.12	0.12
0.12	0.12	0.12

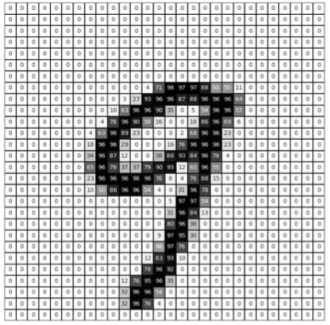
.

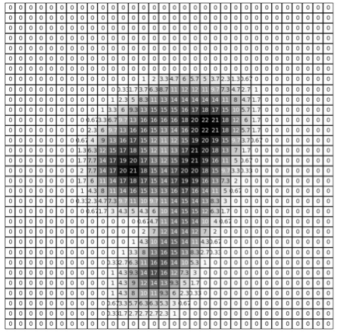
Large Blur

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04



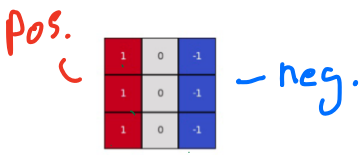


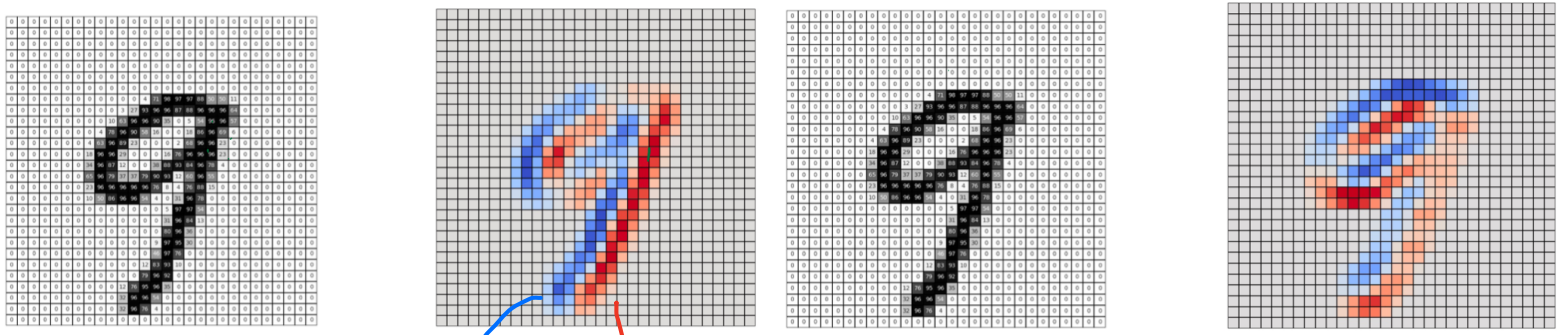




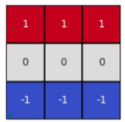
Convolutions in 2-D (Edge detect)

Vertical edges





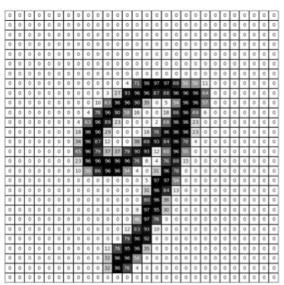
Horizontal edges

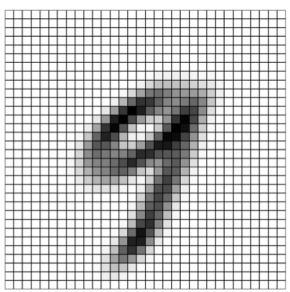


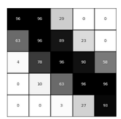
Convolutions in 2-D (Edge detect)

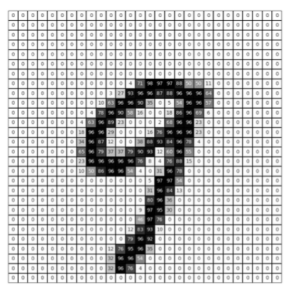
Diagonal edges

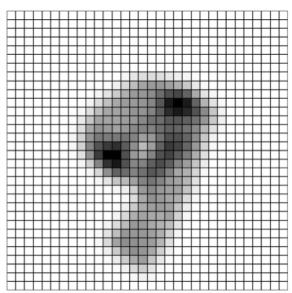












Convolutions for color images

Kernel has 3 channels like Image!

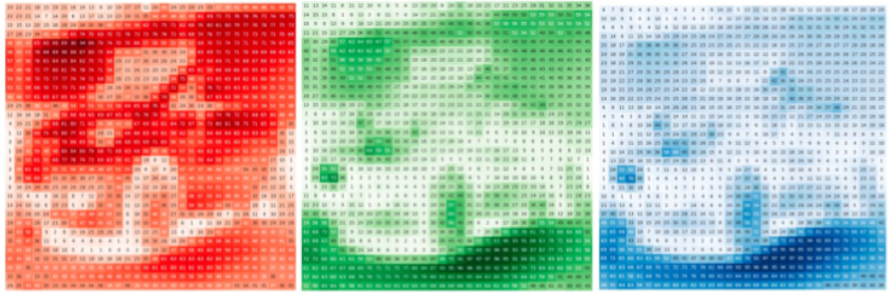
$H \times W \times 3$

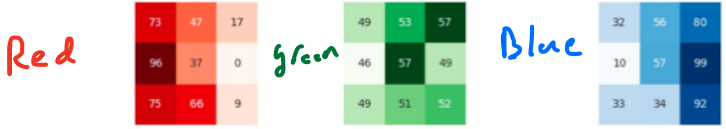
$Conv(X, K) = \underset{1}{\overset{5\times5\times3}{}}$

$Conv(X_r, K_r) +$
$Conv(X_g, K_g) +$
$Conv(X_b, K_b)$







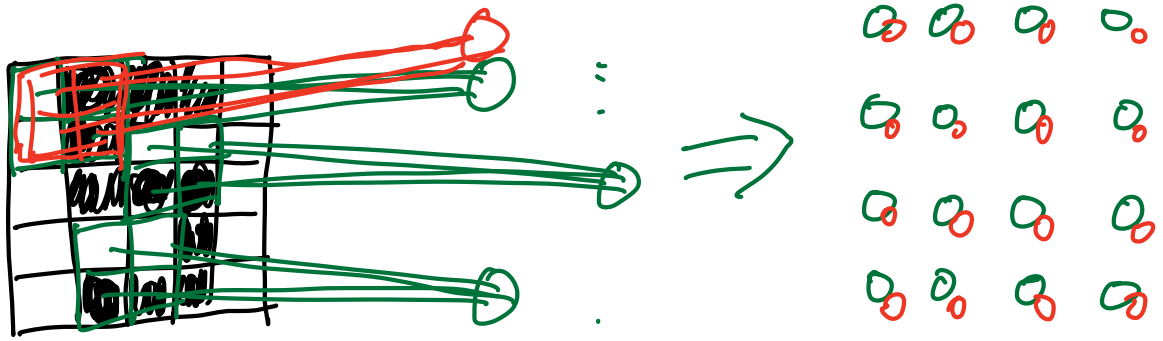




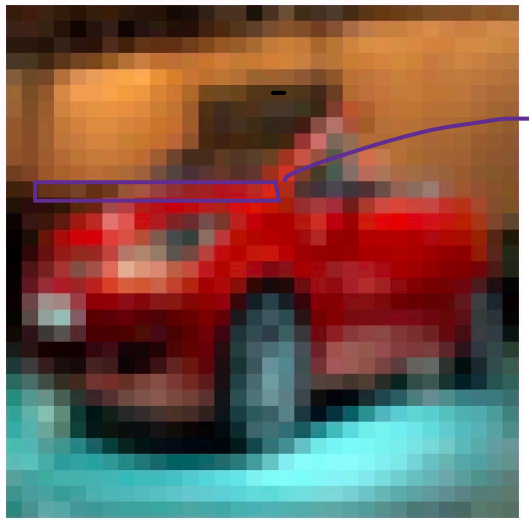
Generally not enough to look for one pattern

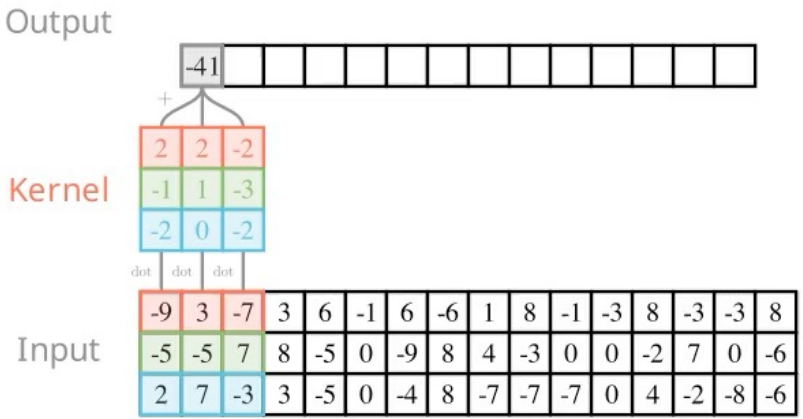
This convolutional Layer will typically have multiple kernels





Multi-Channel Convolution (1-D)





Not limited to 3-channels!

$$Conv(X, k) = \sum_{c=1}^{C} Conv(X_c, k_c)$$

Sum over channels!

X: H x W x C; k: S x S x C; X_c: H x W; k_c: S x S

General 2-D Convolutional Layer

Input (X): $N \times H \times W \times C$

- N: # Images (Batch)
- H: Height
- W: Width
- C: # Channels

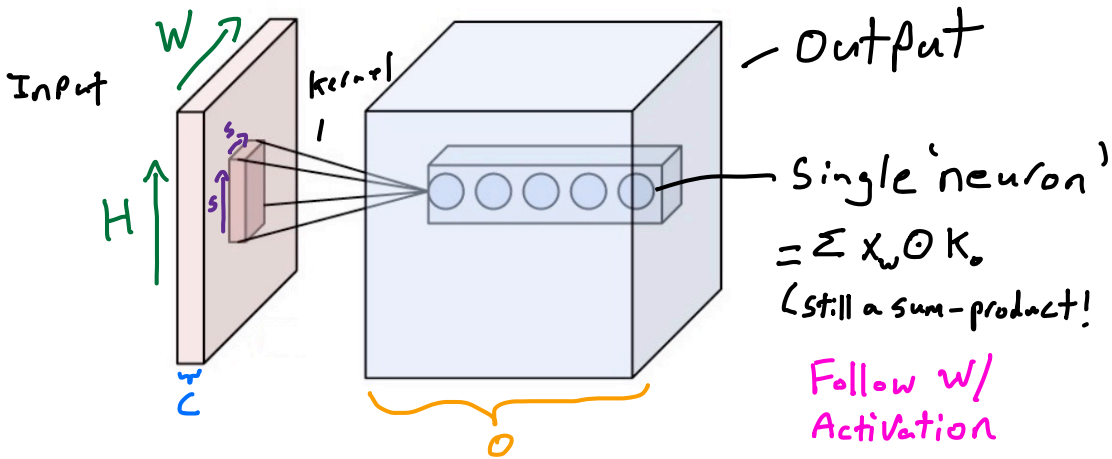
Kernel (K): $S \times S \times C \times O$

- S: Kernel size
- C: # channels
- O: # output channels

Output (ϕ): $N \times H \times W \times O$

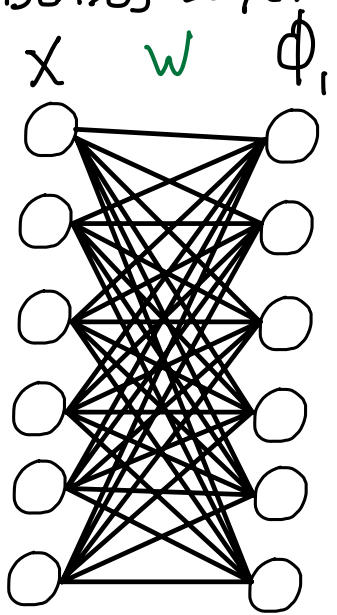
May change for padding $\neq$ 'same'

Diagram:



Convolution as a Layer

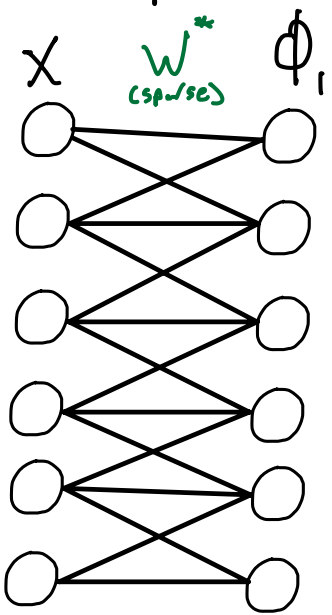
Standard ('Dense') Layer



$$\phi_1 = \sigma(x^T W + b)$$

Every output depends on every input

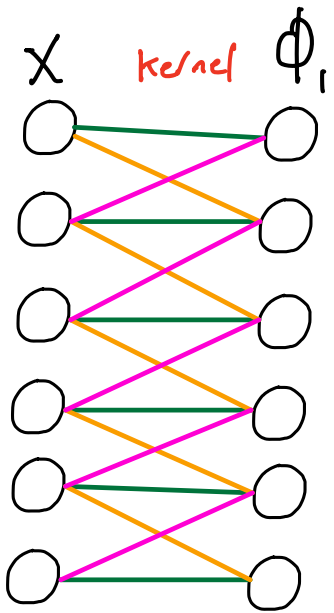
Locally-connected Layer



$$\phi_1 = \sigma(x^T W^* + b)$$

Every output depends only on Local inputs

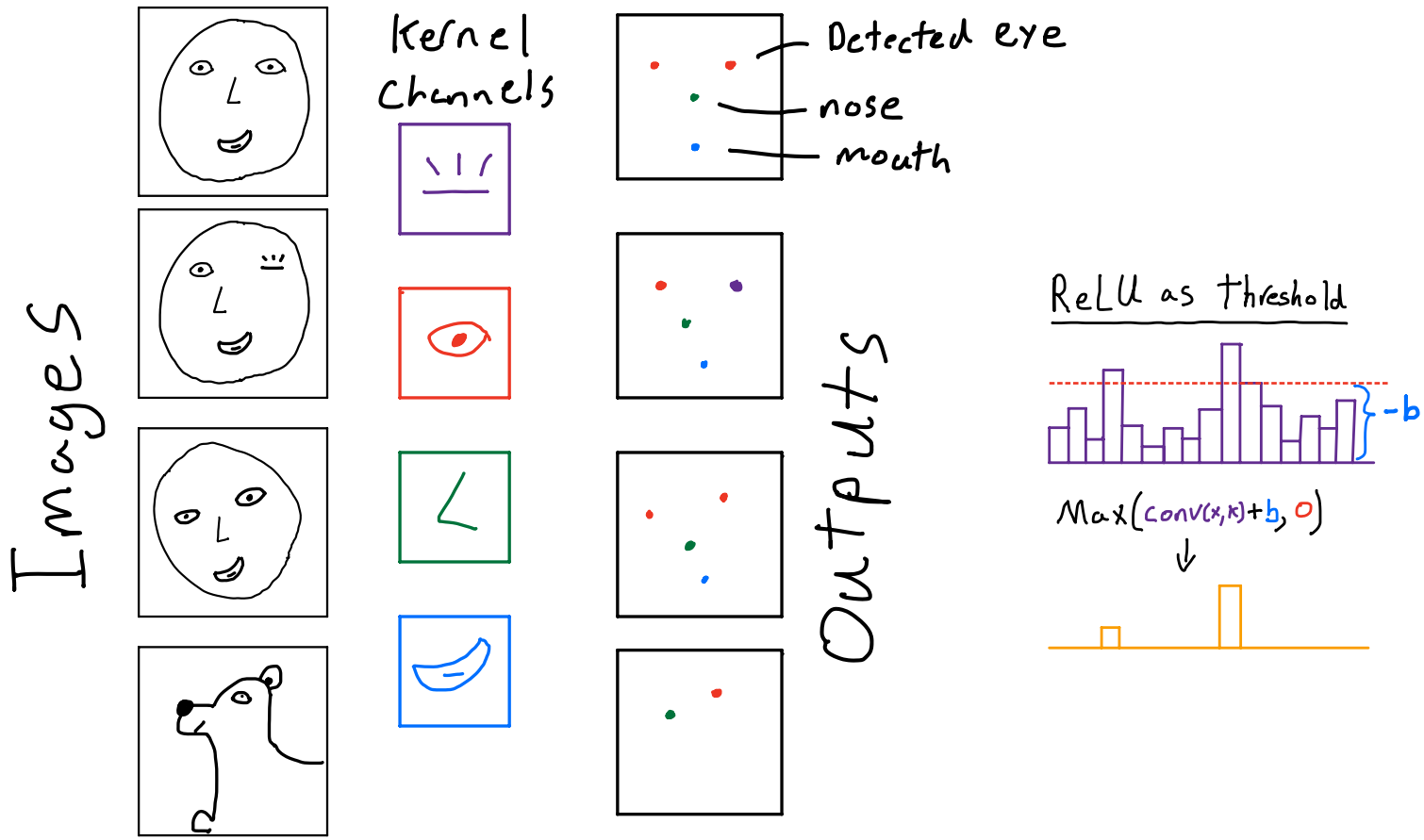
Convolutional Layer



$$\phi_1 = \sigma(\text{conv}(x, k) + b)$$

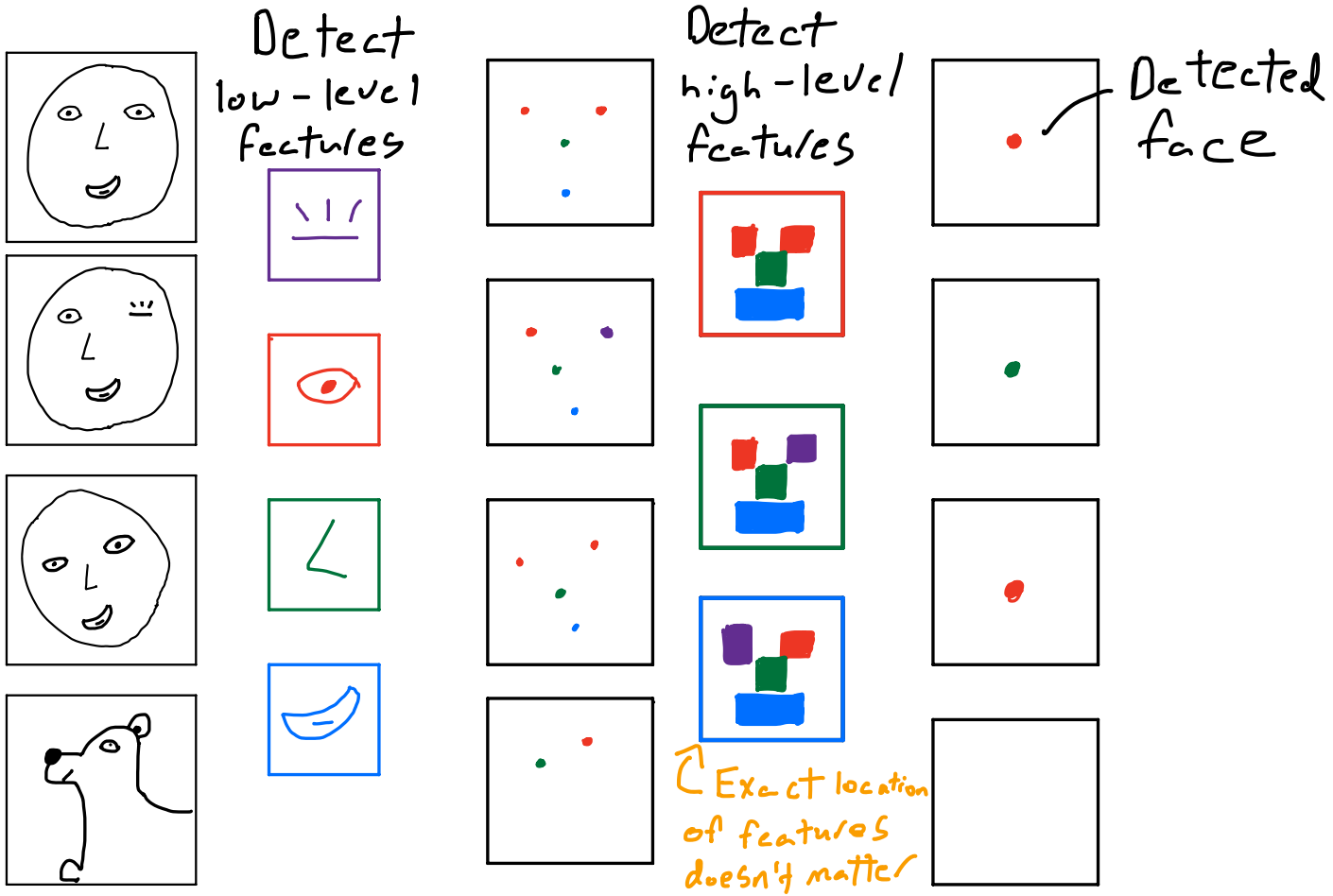
And weights are Shared for each output!

Convolutions as feature detectors



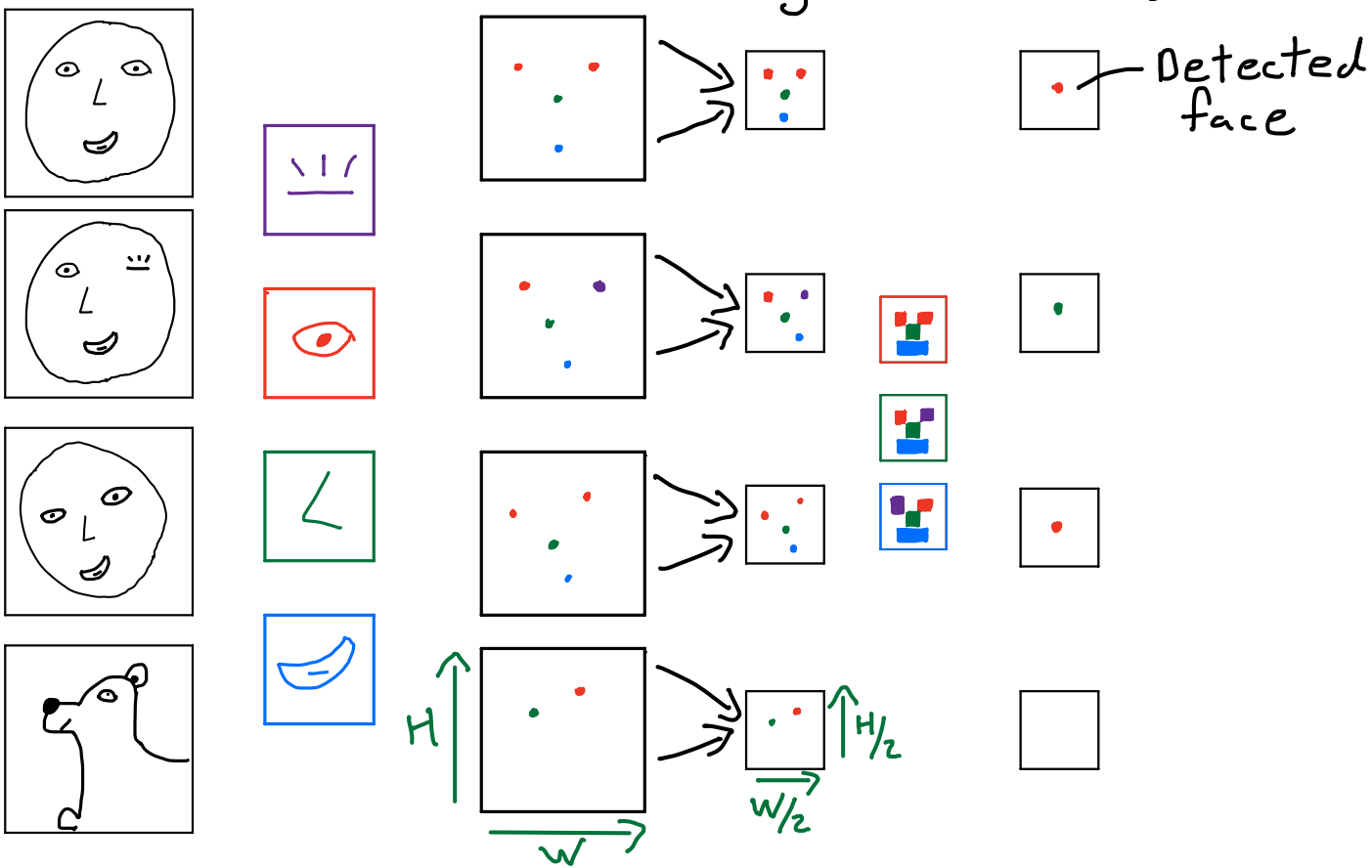
ReLU as threshold

Multi-Layer CNNs



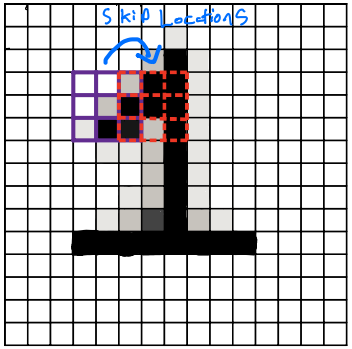
Down Sampling

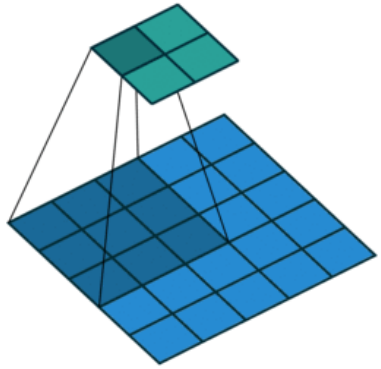
Reduce resolution → Easier and less expensive to find high-level features



Strided Convolutions (2-D)

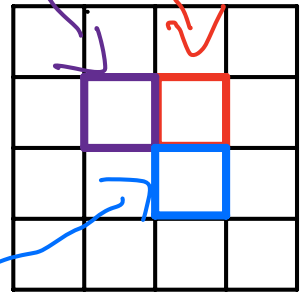
- Apply stride in each dimension





5	-1	3	8	4	6	-2
-3	4	9	1	1	7	-4
5	-6	3	2	1	-2	0
0	-8	5	5	4	-2	3
4	7	2	-7	3	1	4
4	-3	1	2	6	1	-1
0	0	3	-2	3	-1	0

-1	-2	-1
0	0	0
1	2	1

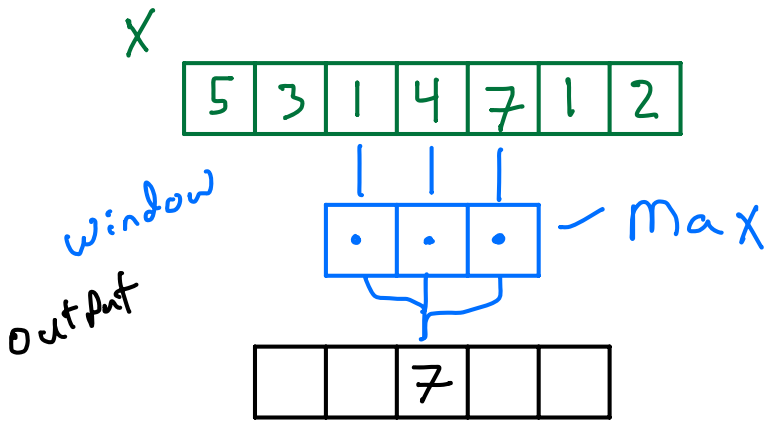


Stride = 2

Pooling operator (1-D)

Inputs: x : Array of length d

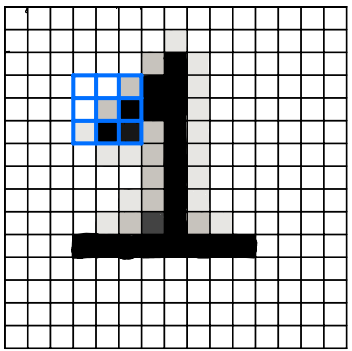
window size s



$7 = \max(1, 4, 7)$

- Take max output around each location before downsampling
- Make sure we don't miss any high activations
- Can also take Avg. (Average pooling)

Pooling in 2-D



Align window in every 2-d location

$$\text{Max-Pool}(X)_{ij} = \max_{a=1}^{s}\left(\max_{b=1}^{s}\left(X_{i+a,j+b}\right)\right)$$

$$\text{Avg.-Pool}(X)_{ij} = \frac{1}{s^2}\sum_{a=1}^{s}\sum_{b=1}^{s} X_{i+a,j+b}$$

5	-1	3	8	4	6	-2
-3	4	9	1	1	7	-4
5	-6	3	2	1	-2	0
0	-8	5	5	4	-2	3
4	7	2	-7	3	1	4
4	-3	1	2	6	1	-1
0	0	3	-2	3	-1	0

max

			7	

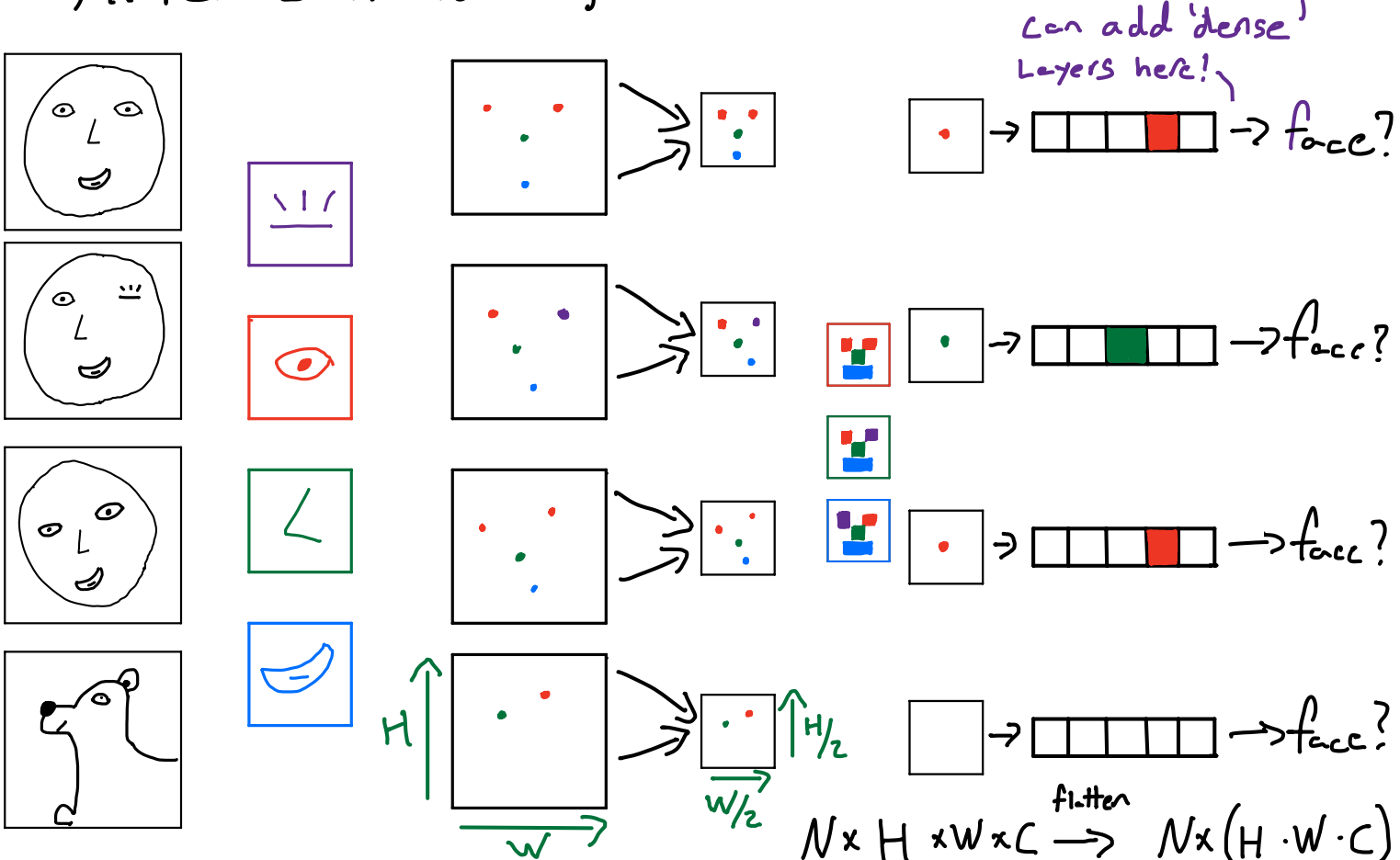
$$7 = \max(1, 1, 7, 2, 1, -2, 5, 4, -2)$$

Down Sampling Comparison



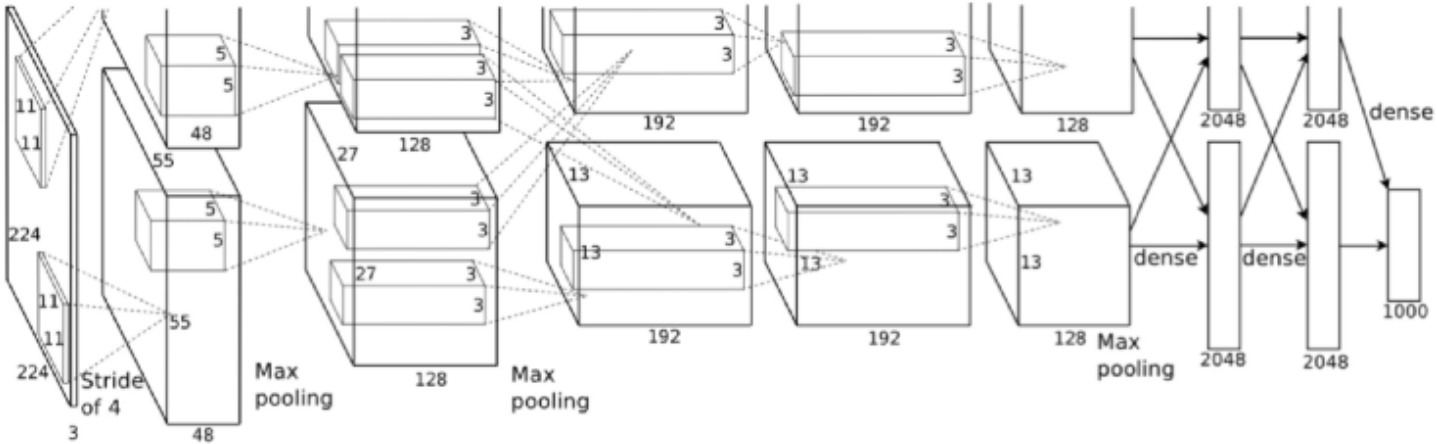
Flattening

- After convolutions, flatten as before

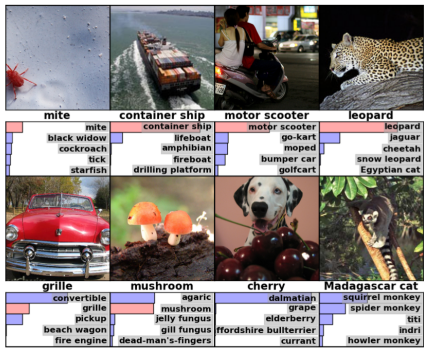


AlexNet (2012)

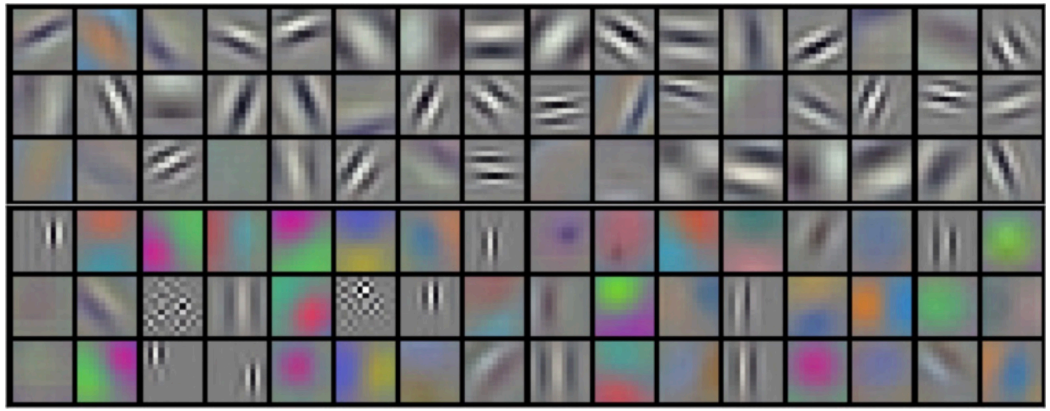
Architecture:



1000 Classes



Learned kernels



CNN Features by Layer

