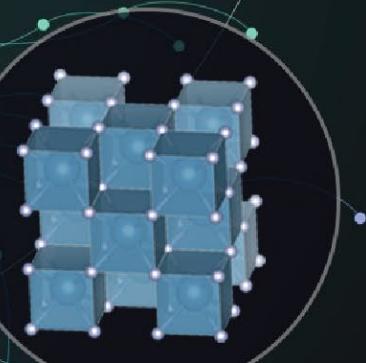
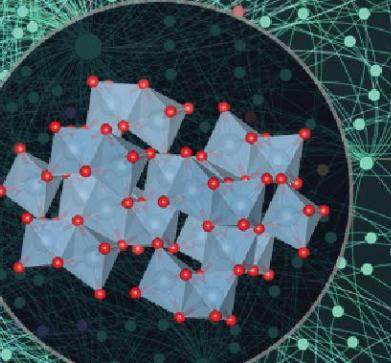
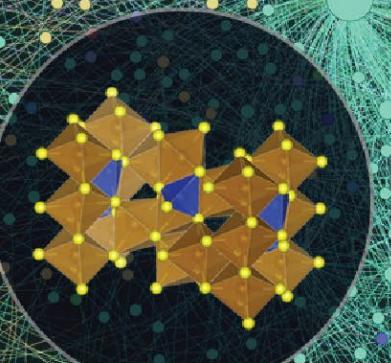
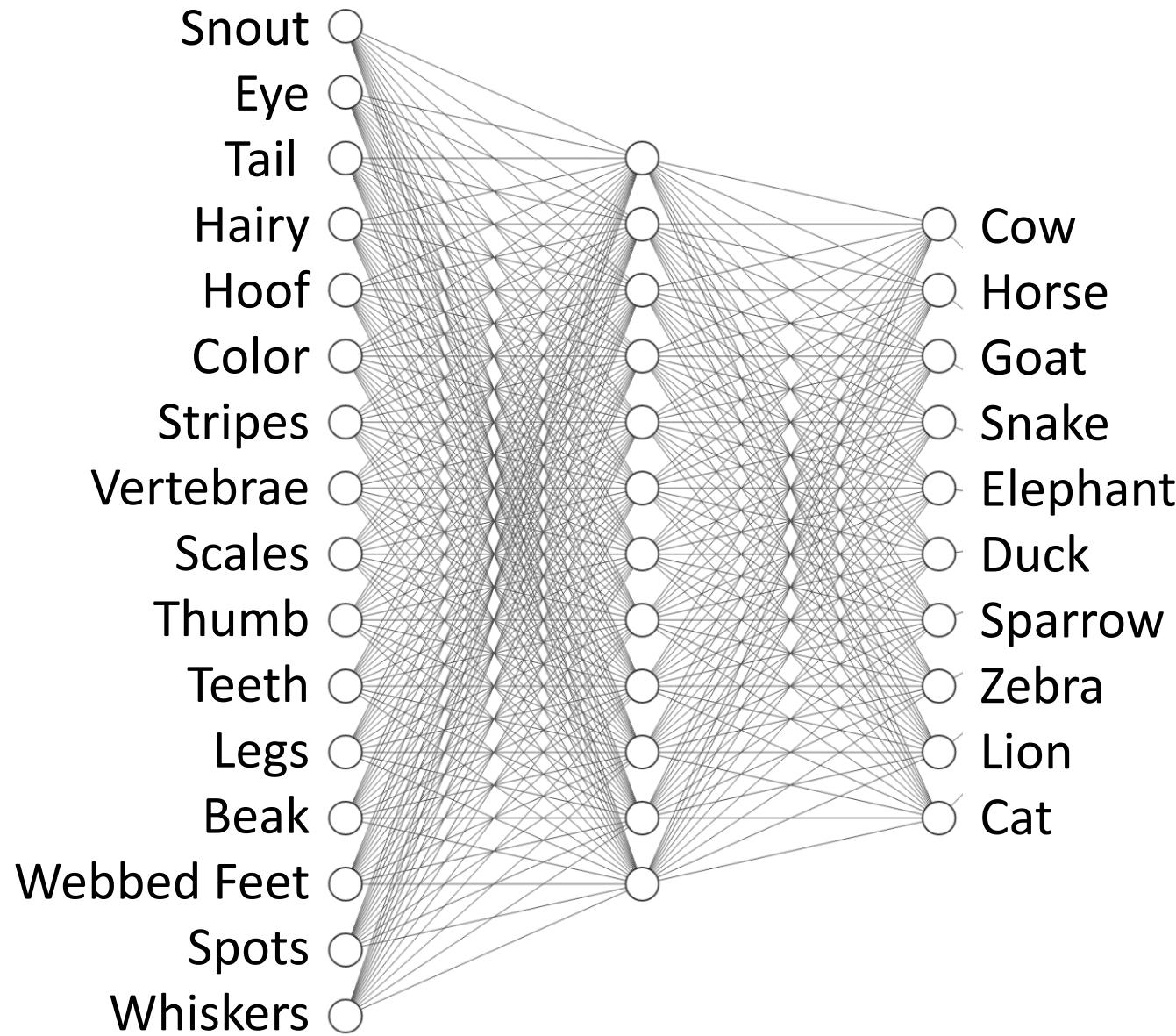


convolutional neural networks



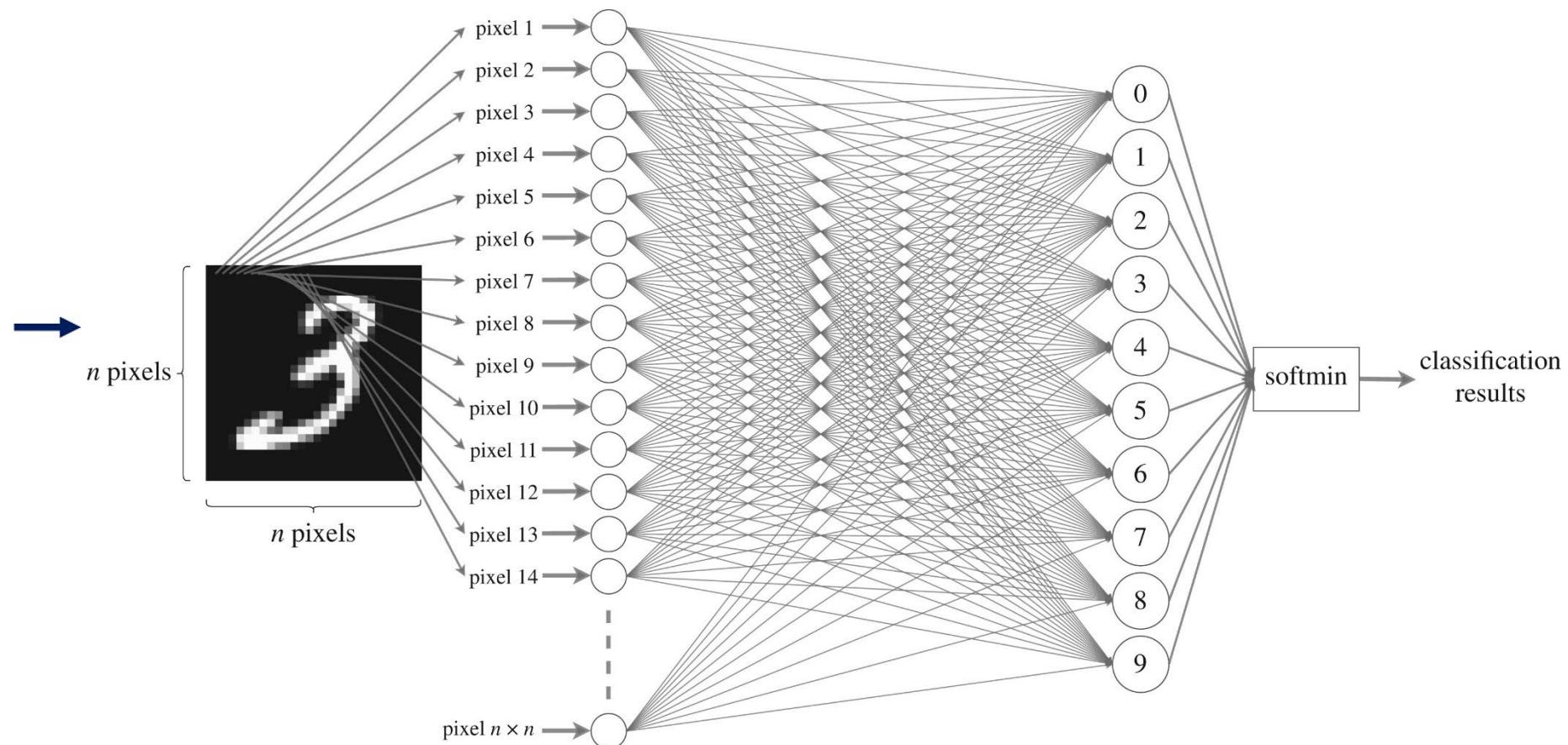
Vanilla neural networks do well to find patterns in the features provided



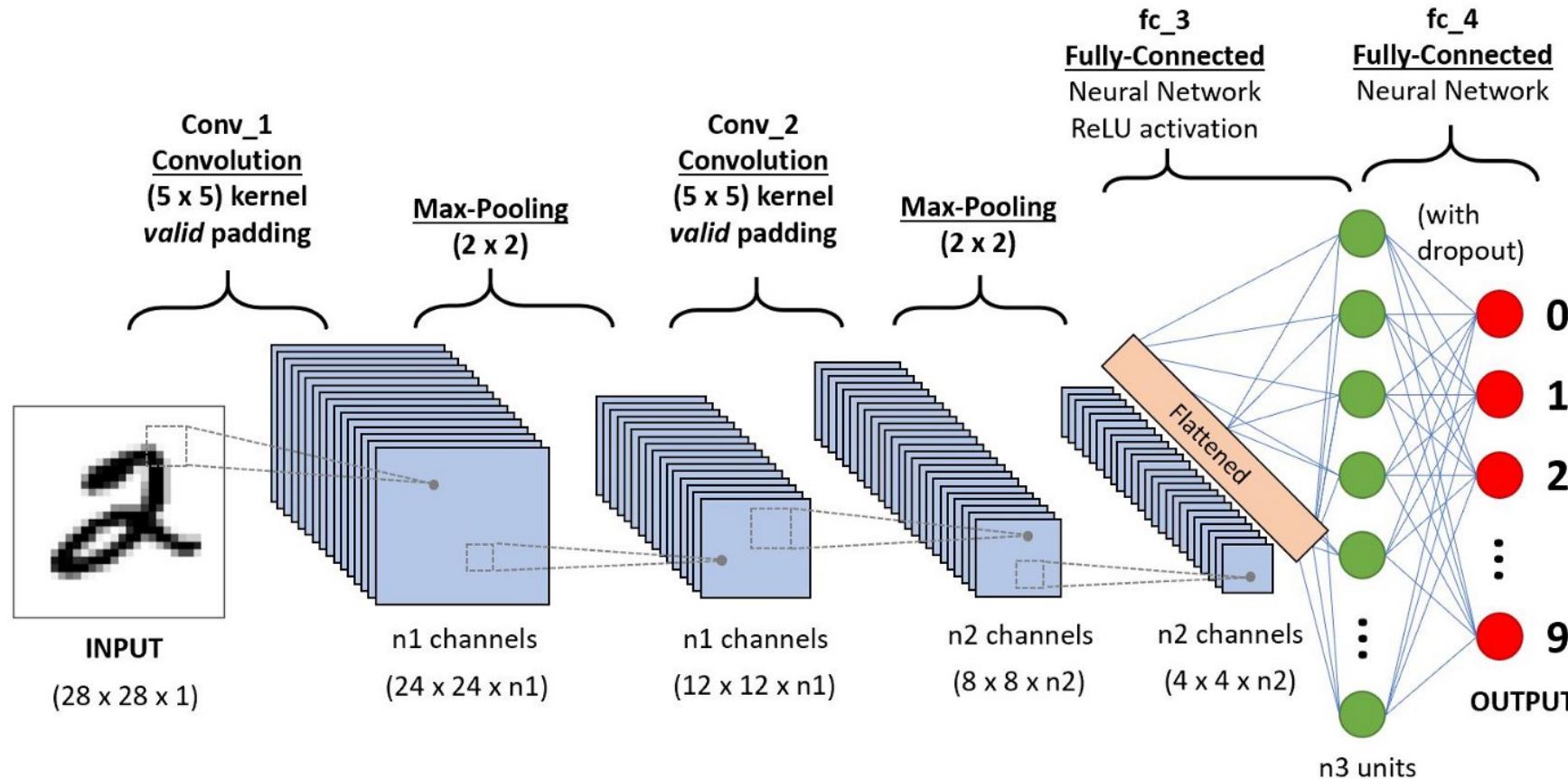
We can go beyond “vanilla” neural networks to do more advanced tasks

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

Data & Labels

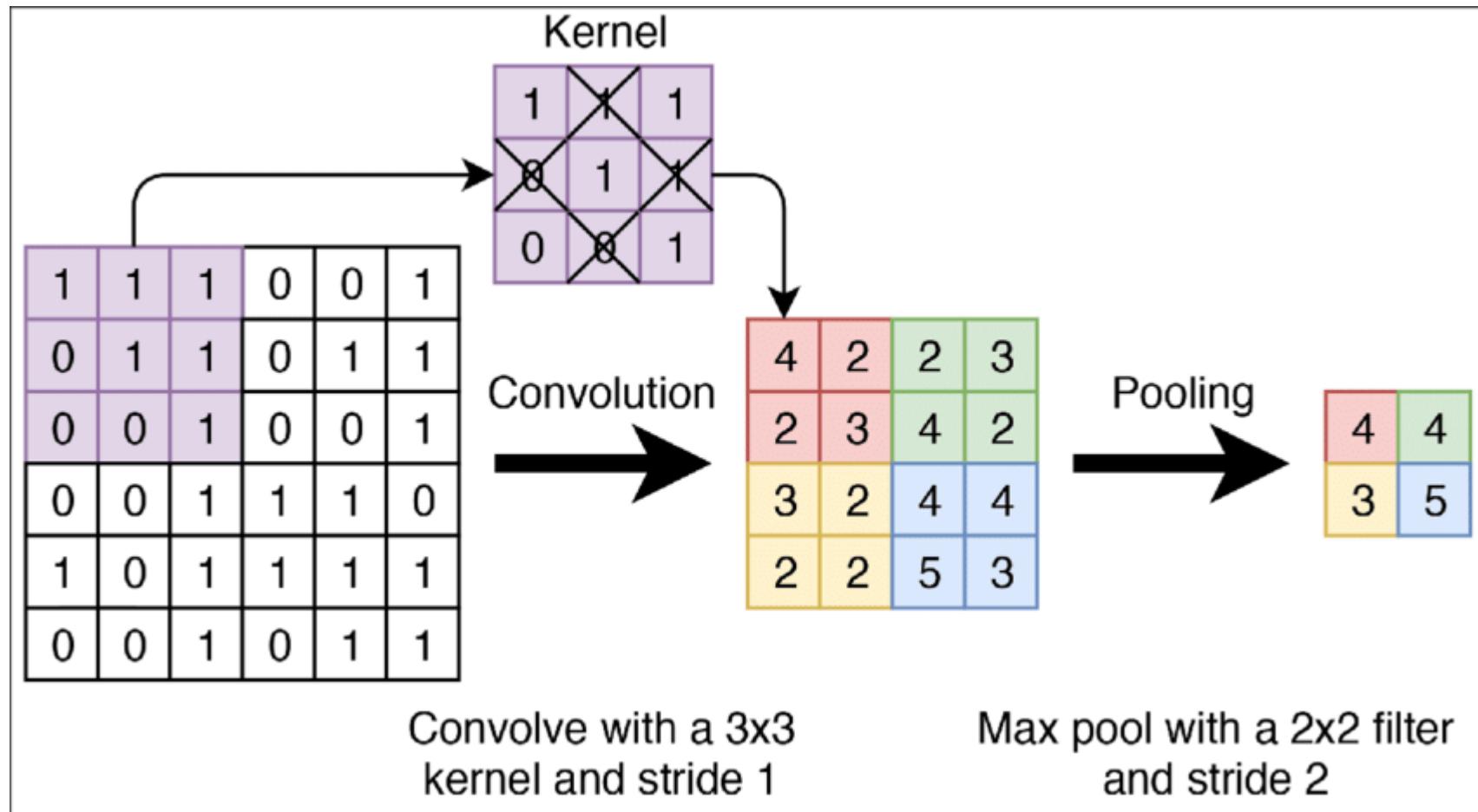


Convolutional neural networks do really well at learning from image data

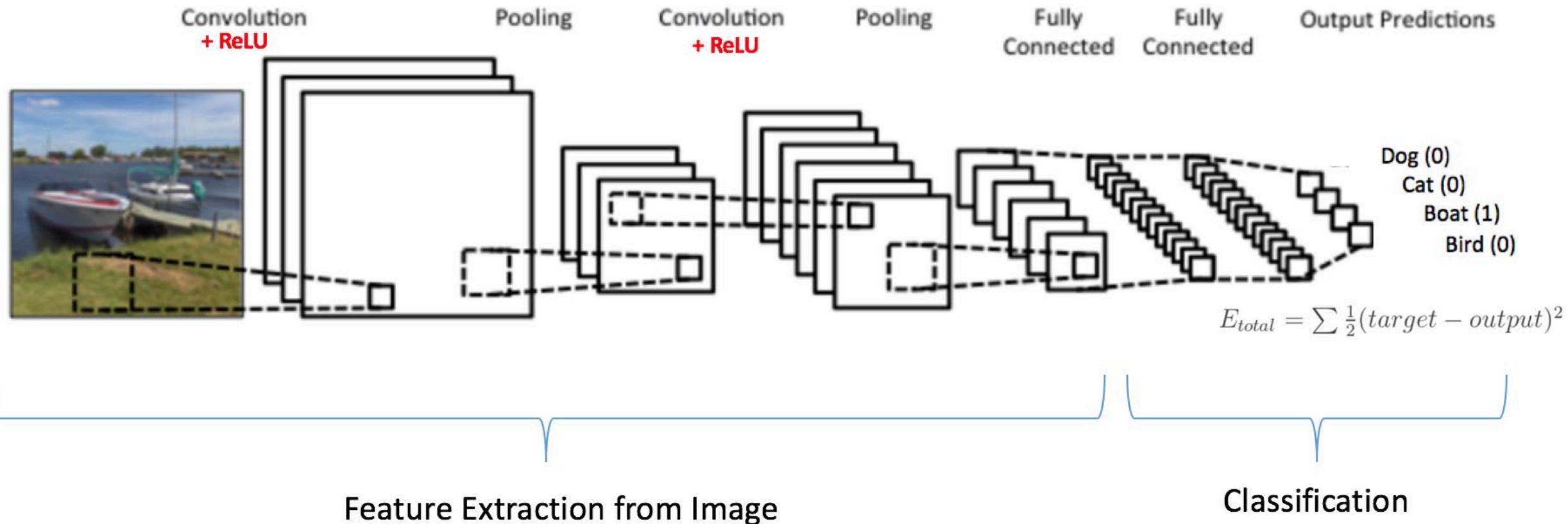


Key assumption: we shouldn't ignore how pixels are spatially located!

CNNs are made up of convolutions and pooling layers



Convolutions and pooling extract features automatically!



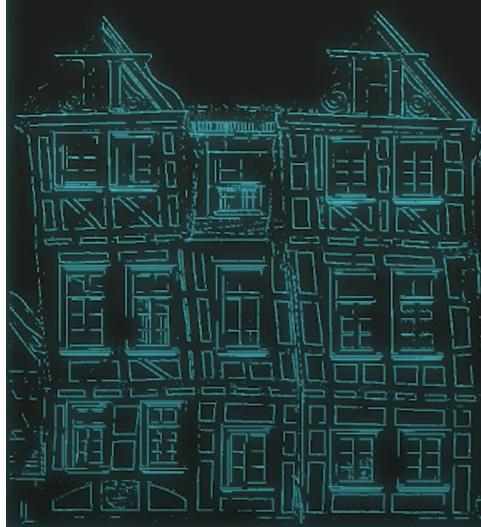
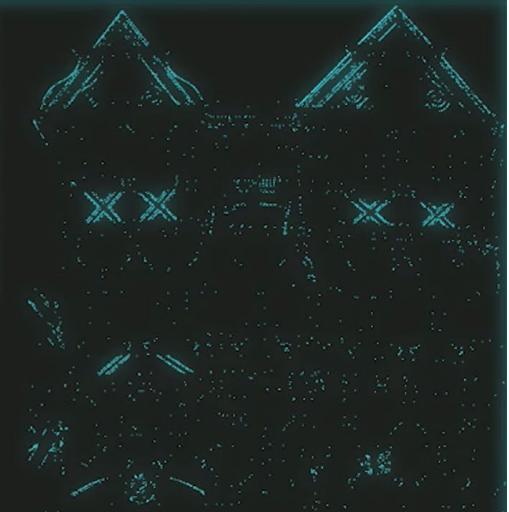
Low level filters do a good job of finding edges, corners, shapes, colors, and gradients



2	2	4	2	2
1	1	2	1	1
0	0	0	0	0
-1	-1	-2	-1	-1
-2	-2	-4	-2	-2



1	1	0	-1	-1
1	1	0	-1	-1
0	0	0	0	0
-1	-1	0	1	1
-1	-1	0	1	1

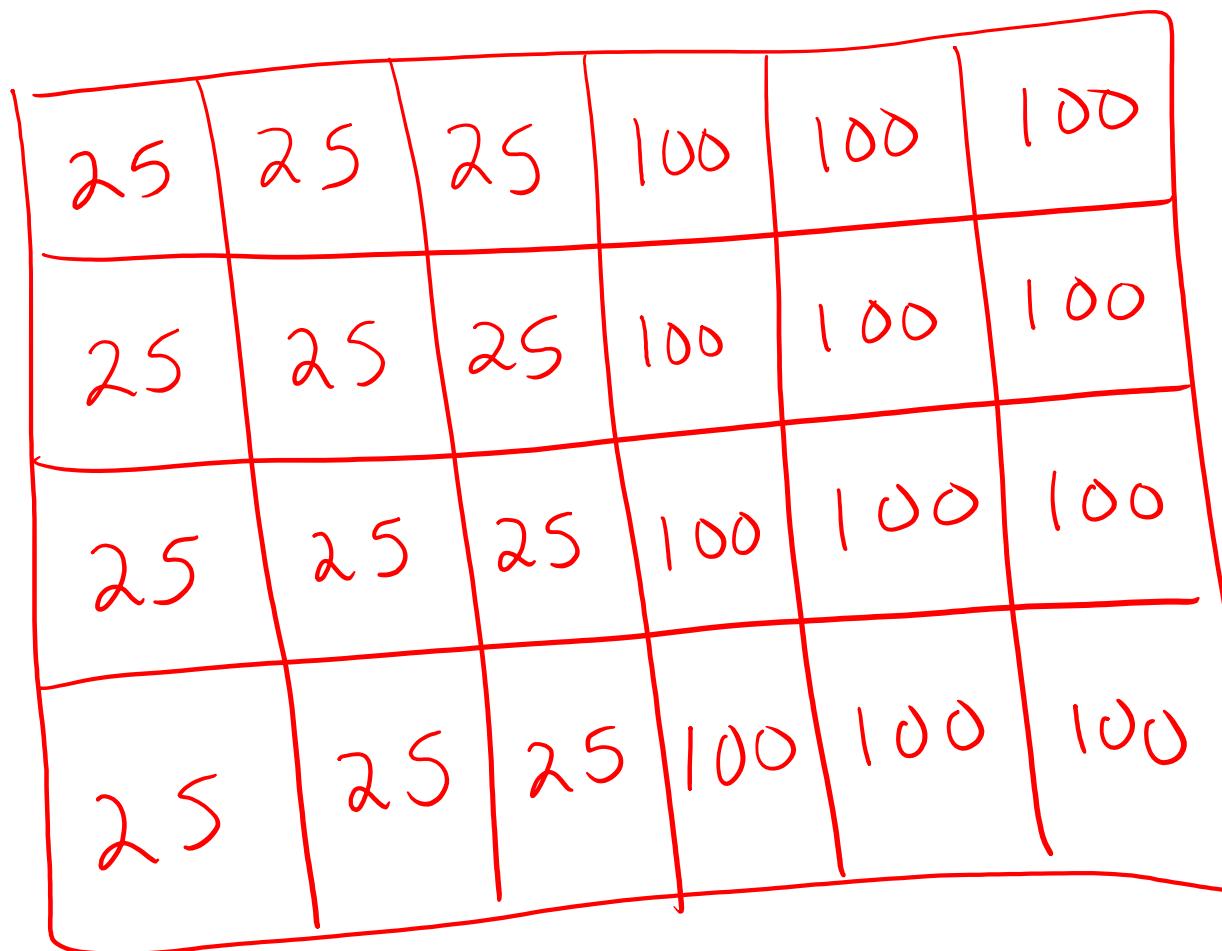


0	1	2	3	4
-1	0	1	2	3
-2	-1	0	1	2
-3	-2	-1	0	1
-4	-3	-2	-1	0

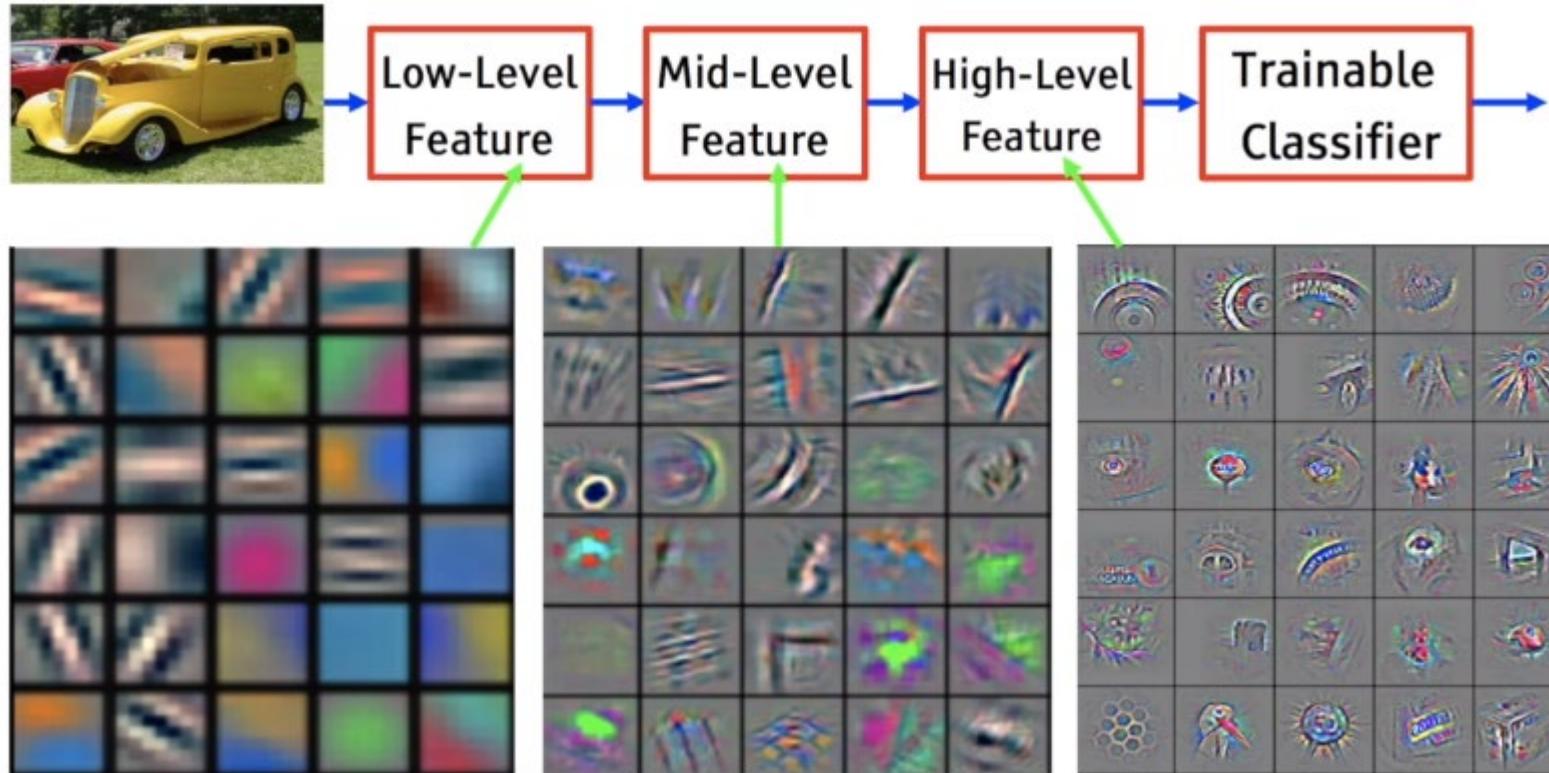
0	0	-1	0	0
0	0	-1	0	0
-1	-1	5	-1	-1
0	0	-1	0	0
0	0	-1	0	0



Low level filters do a good job of finding edges, corners, shapes, colors, and gradients

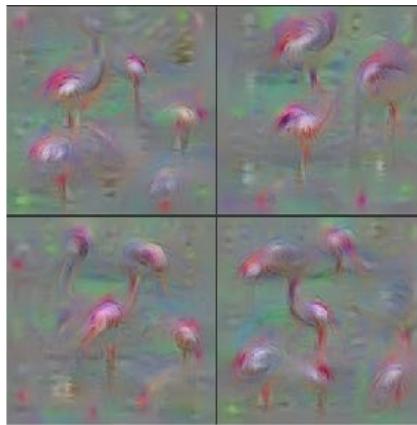


Filters are learned at different scales and change in complexity

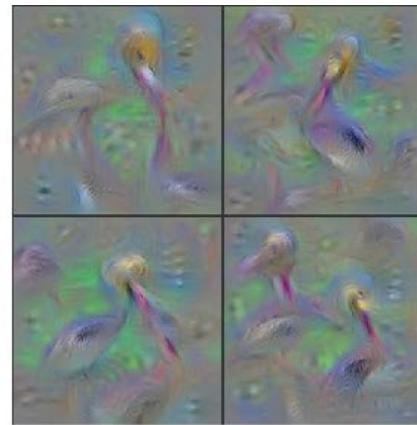


Simple features → Complex shapes similar to real world

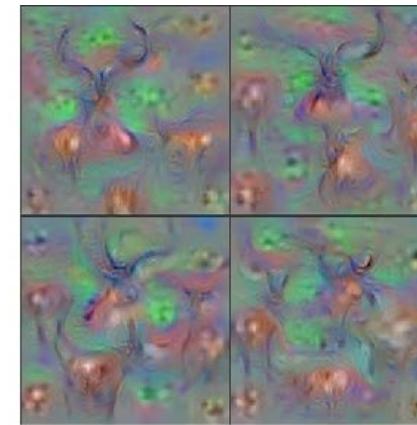
Filters help us learn the feature “essence” of an object



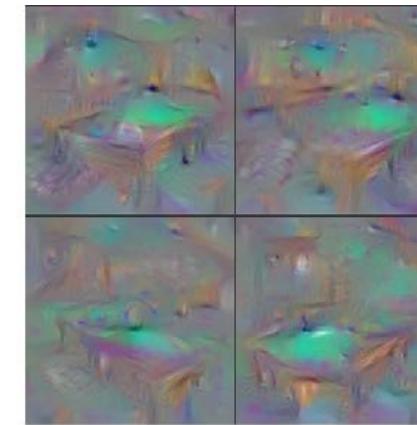
Flamingo



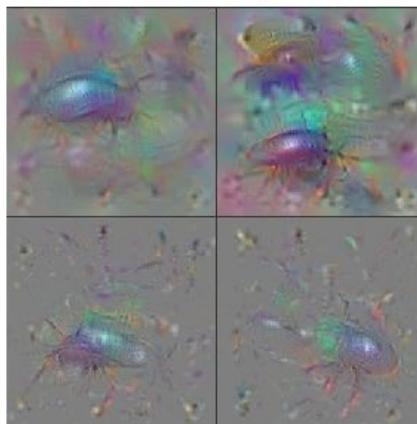
Pelican



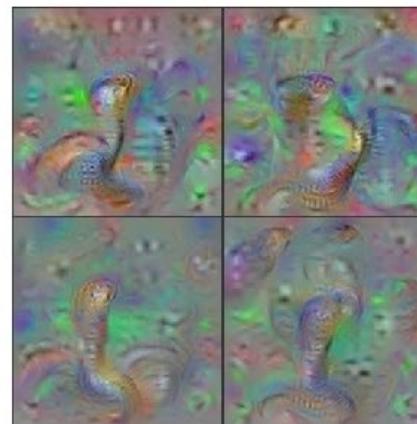
Hartebeest



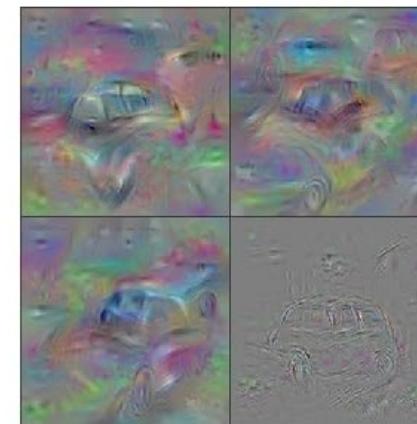
Billiard Table



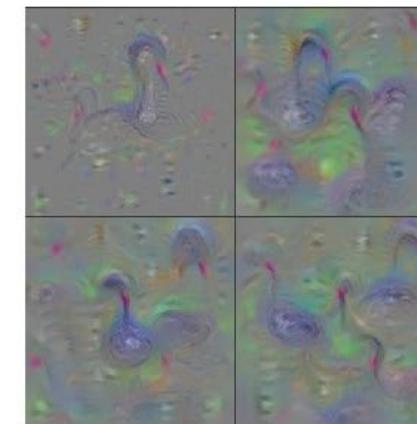
Ground Beetle



Indian Cobra



Station Wagon



Black Swan

Preferred inputs for different class categories

CNNs capture spatial dependence through the use of filters

We need to reduce image to a computationally tractable size without losing information

Original input 5X5

Special 3X3
filter/kernel used for
convolution

$$\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Image

4		

Convolved
Feature

Convolved feature is
3X3 because we had
9 kernel shifts

If our input has depth > 1, we use filter with same depth and sum to single channel output

0	0	0	0	0	0	0	...
0	156	155	156	158	158	158	...
0	153	154	157	159	159	159	...
0	149	151	155	158	159	159	...
0	146	146	149	153	158	158	...
0	145	143	143	148	158	158	...
...

Input Channel #1 (Red)

0	0	0	0	0	0	0	...
0	167	166	167	169	169	169	...
0	164	165	168	170	170	170	...
0	160	162	166	169	170	170	...
0	156	156	159	163	168	168	...
0	155	153	153	158	168	168	...
...

Input Channel #2 (Green)

0	0	0	0	0	0	0	...
0	163	162	163	165	165	165	...
0	160	161	164	166	166	166	...
0	156	158	162	165	166	166	...
0	155	155	158	162	167	167	...
0	154	152	152	157	167	167	...
...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1



308

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2



-498

0	1	1
0	1	0
1	-1	1

Kernel Channel #3



164

$$+ 1 = -25$$

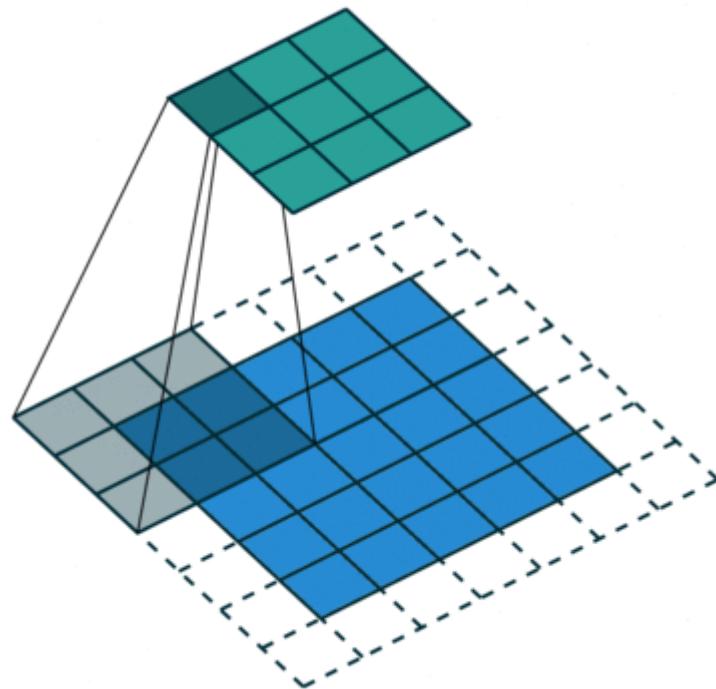
Bias = 1

-25				...
				...
				...
				...
...

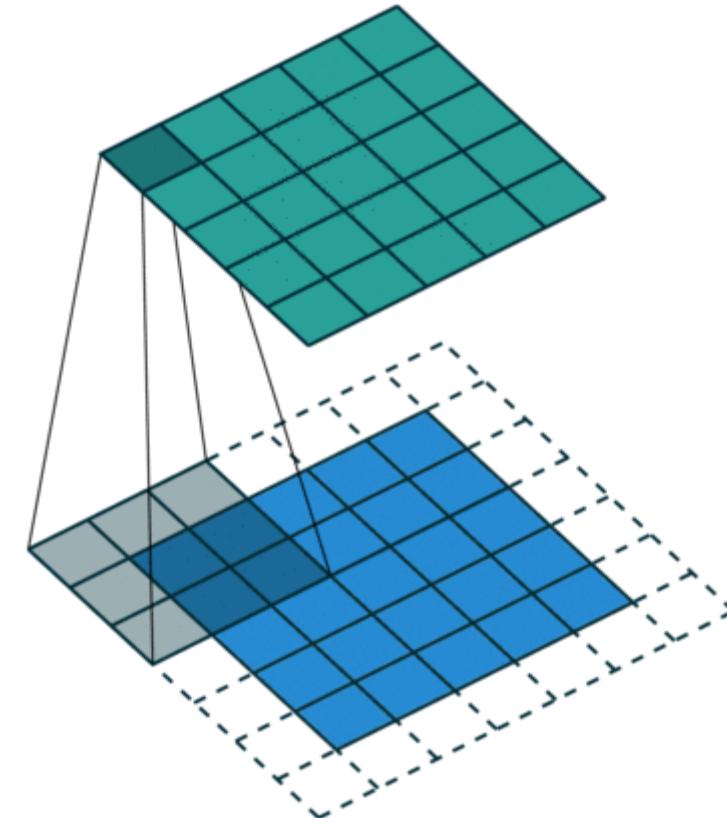
Output

Convolutions can take place over different “strides” with different padding

“Valid padding” reduces dimensionality

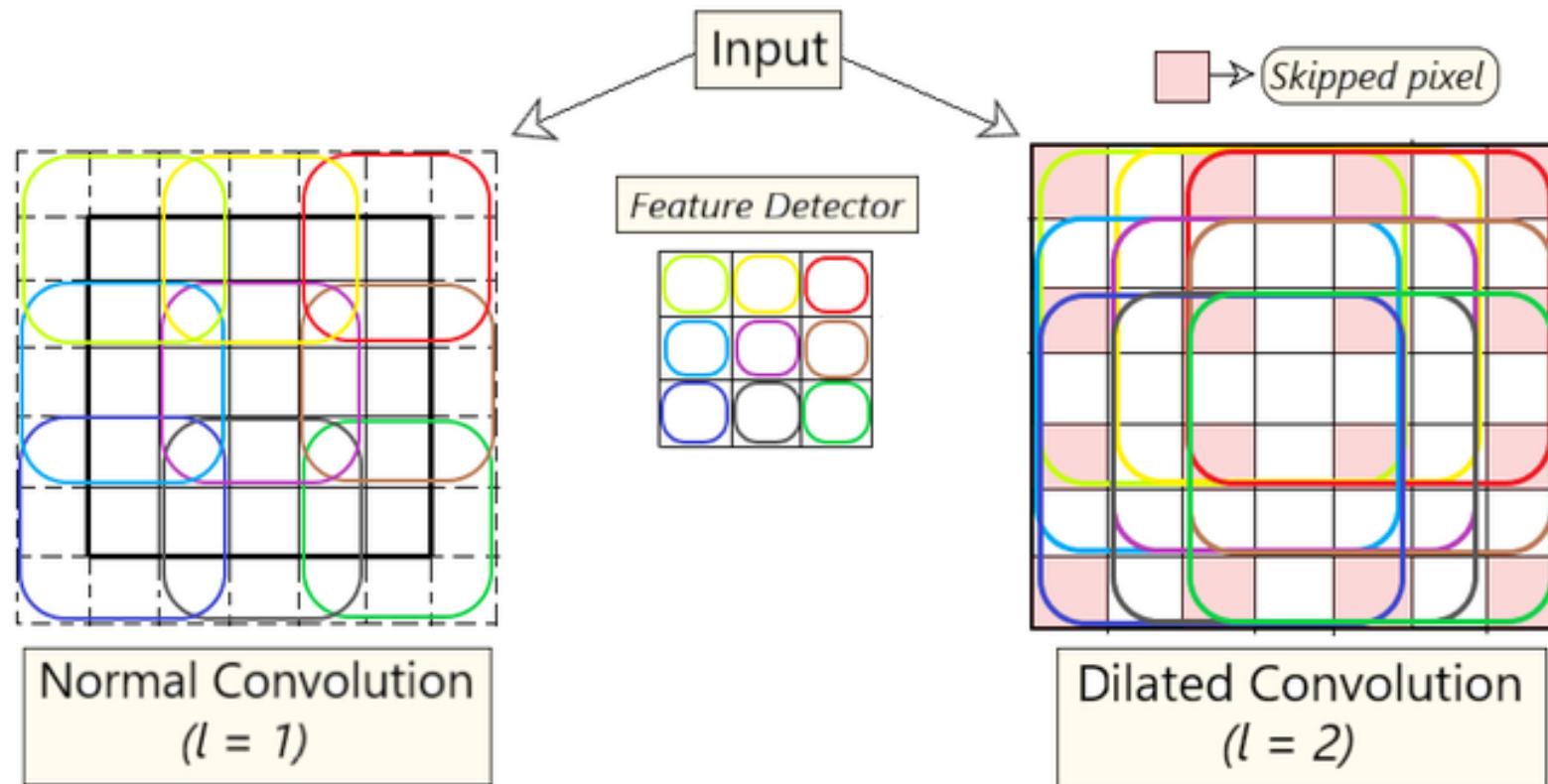


“Same padding” keeps it same or increases



"Dilated" convolution allows us to sample kernel over a larger area

An additional parameter l (dilation factor) tells how much the input is expanded. In other words, based on the value of this parameter, $(l-1)$ pixels are skipped in the kernel.



Dilated convolutions offer some advantages

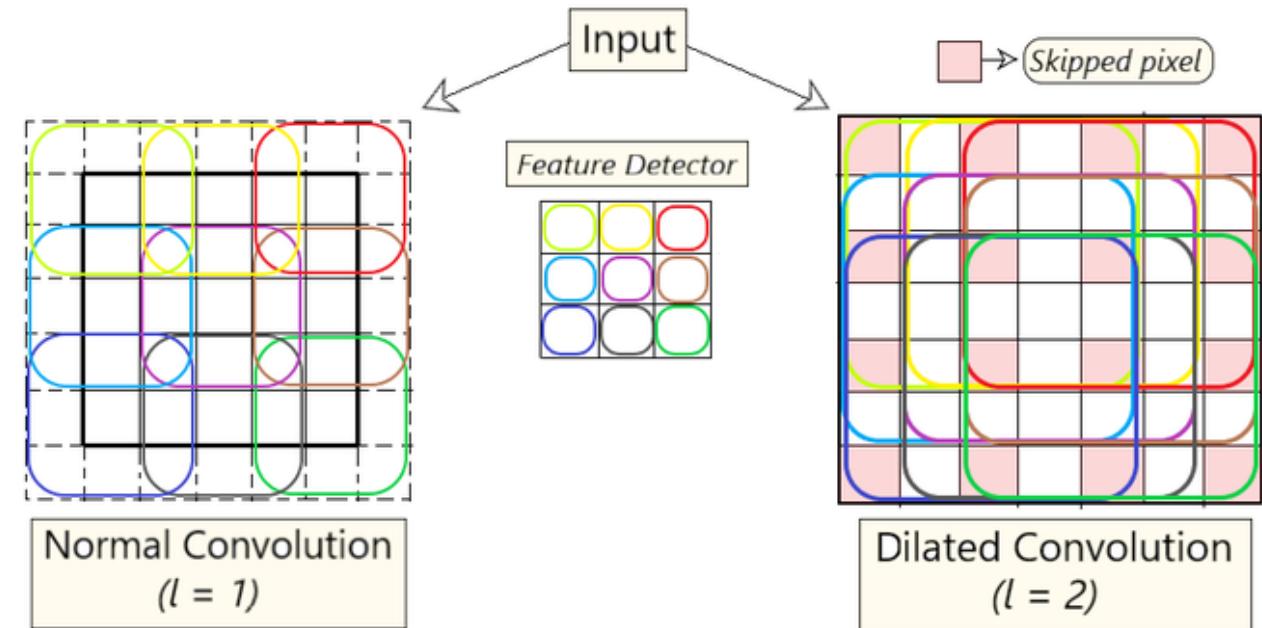
Larger receptive field (no loss of coverage)

Computationally efficient (larger coverage at same cost)

Less memory usage since it can skip the pooling step

No loss of resolution (dilation instead of pooling)

Order of data maintained



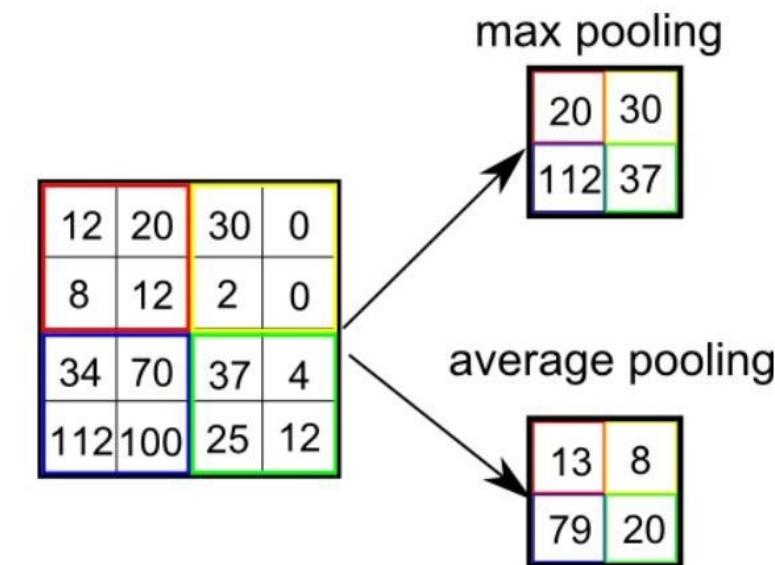
Pooling also reduces dimensionality while also extracting dominant features

Max pooling or average pooling possible

Both use a kernel, but instead of summing, we return max value or average

3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

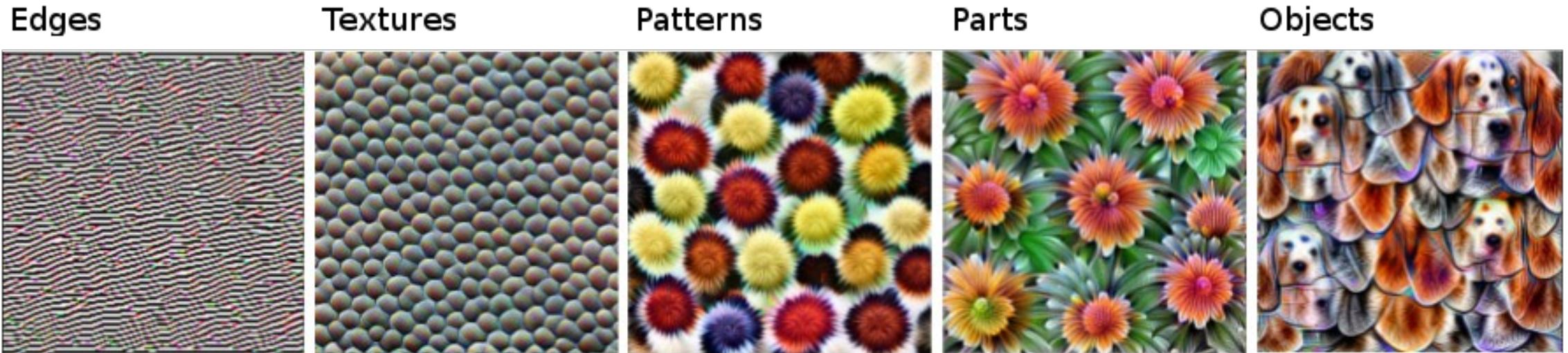
3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1



Max pooling also performs noise suppression

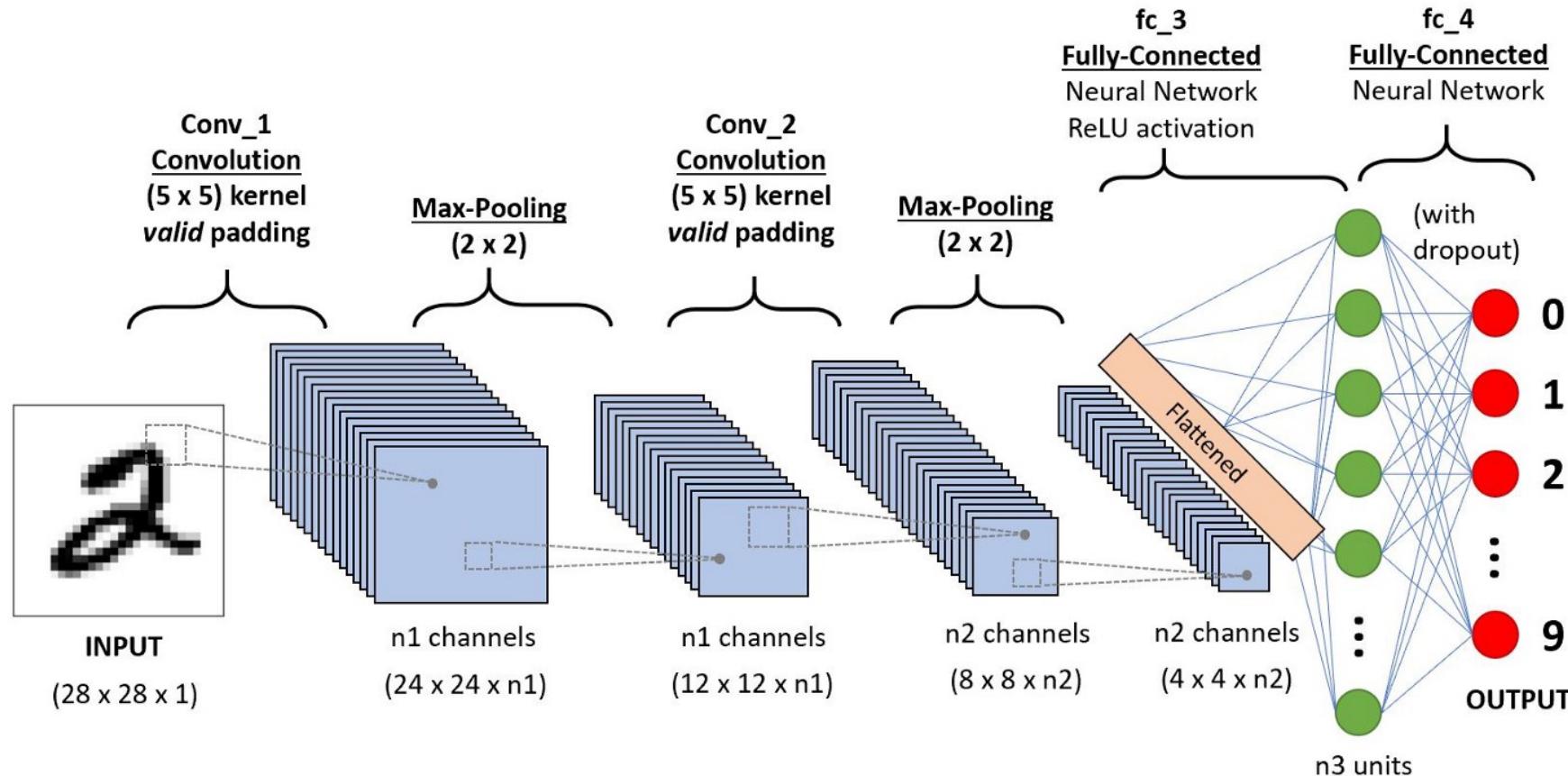
Pooling prevents overfitting and speeds up calculations

Convolution and pooling layers are repeated to build up abstraction in network



More abstract kernels are made via combination of simple filters

Convolutions/Pooling learn features, but we need to correlate these with target



CNNs can be broken down into steps

Step 1: We initialize all filters and parameters / weights with random values

Step 2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

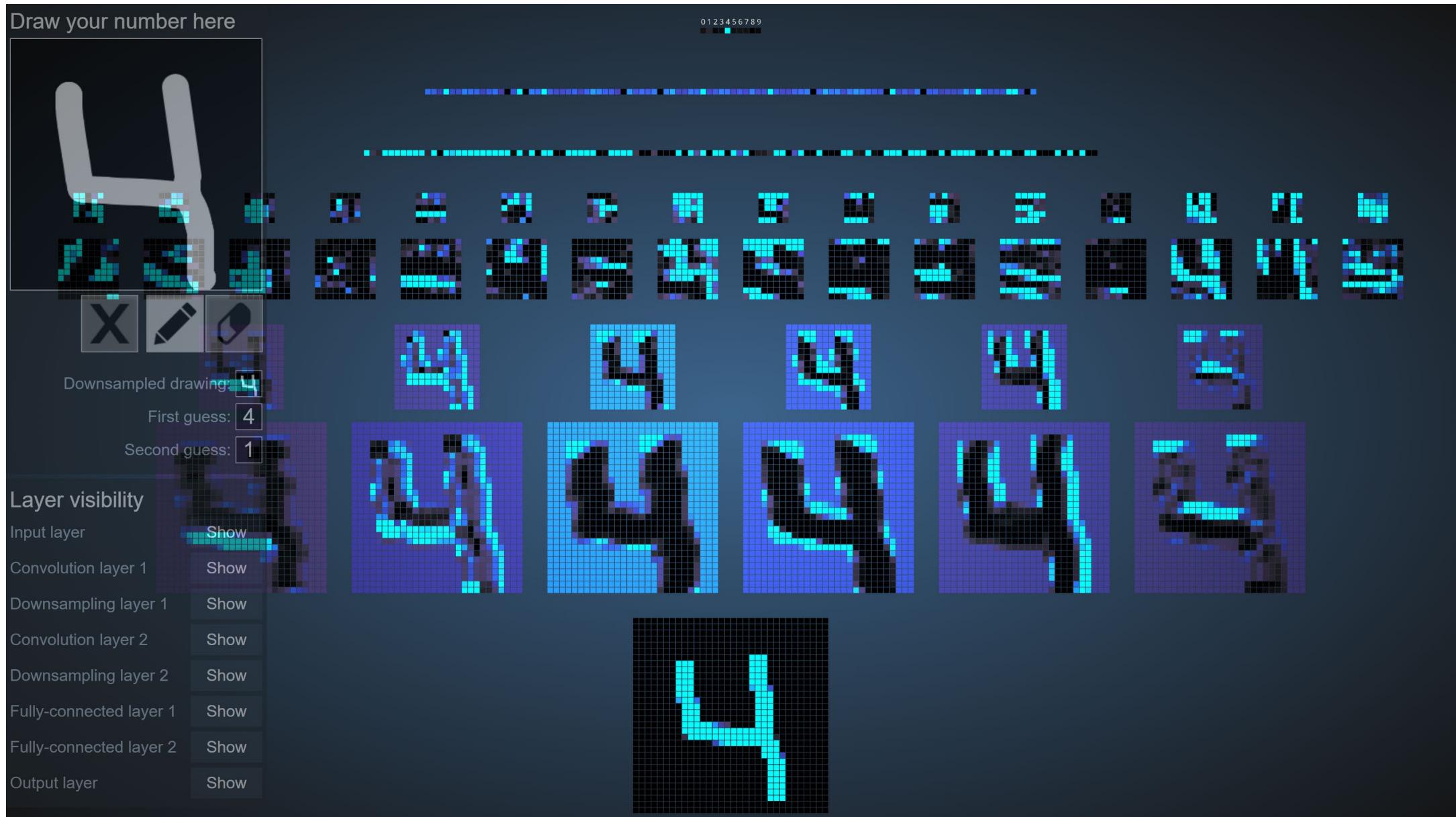
Step 3: Calculate the total error at the output layer

Step 4: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.*

Step 5: Repeat steps 2-4 with all images in the training set.

*Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.

We can visualize CNNs in action!



<https://www.cs.ryerson.ca/~aharley/vis/>

recurrent neural networks

