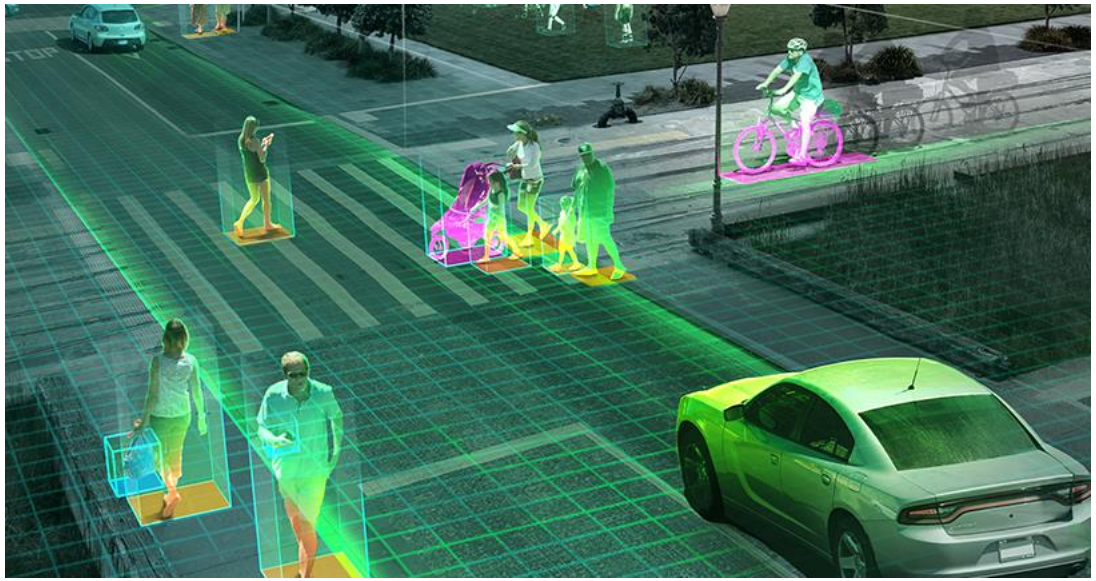
TeIAS | Tehran Institute for Advanced Studies

بنام خداوند جان و خرد



Deep Learning
Computer Vision

Mohammad Taher Pilehvar

Machine Learning 01

https://teias-courses.github.io/ml01/

Image courtesy Nvidia

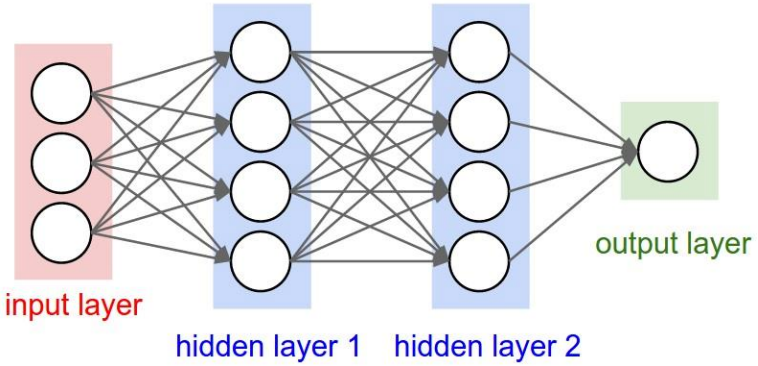
CNN: Convolutional Neural Network

- Designed specifically for images
- Still, a sequence of layers, each with many neurons and learnable weights
- Still, we have a loss function, gradient descent optimization, etc.

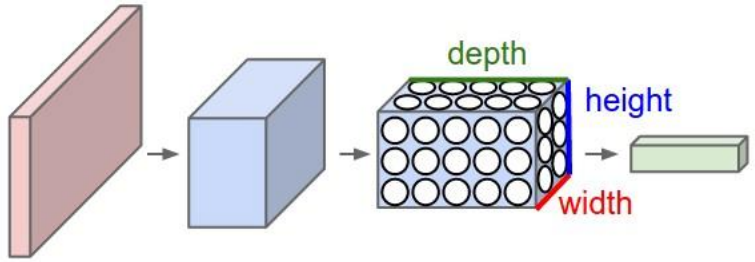
But: a different architecture!

Dense for images?

- They don't scale well to large images!
  - Image size 32x32x3 (32 wide, 32 high, 3 color channels)
    - 32*32*3 = 3072 weights
    - A more realistic image:
      - 200*200*3 = 120,000 weights

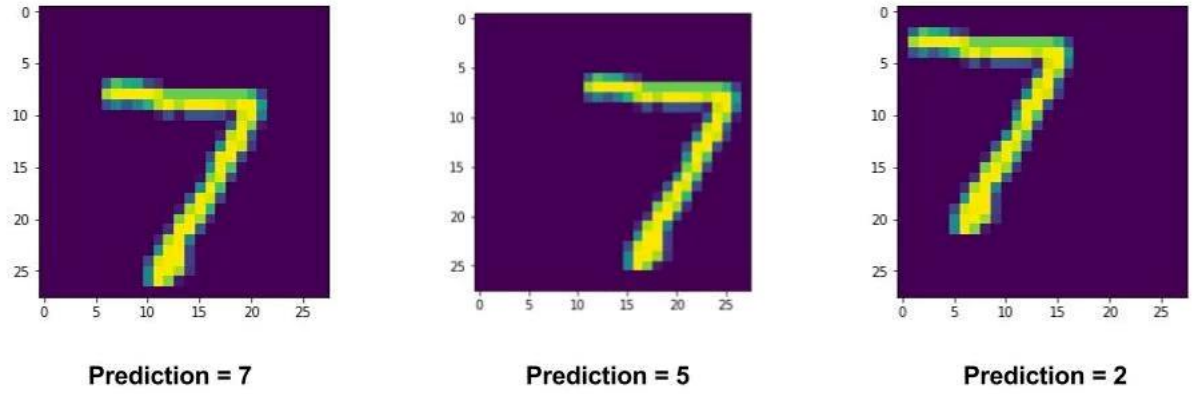


- Not easy to have *channels* for them
  - Instead, ConvNets have neurons arranged in 3 dimensions: **width, height, depth**



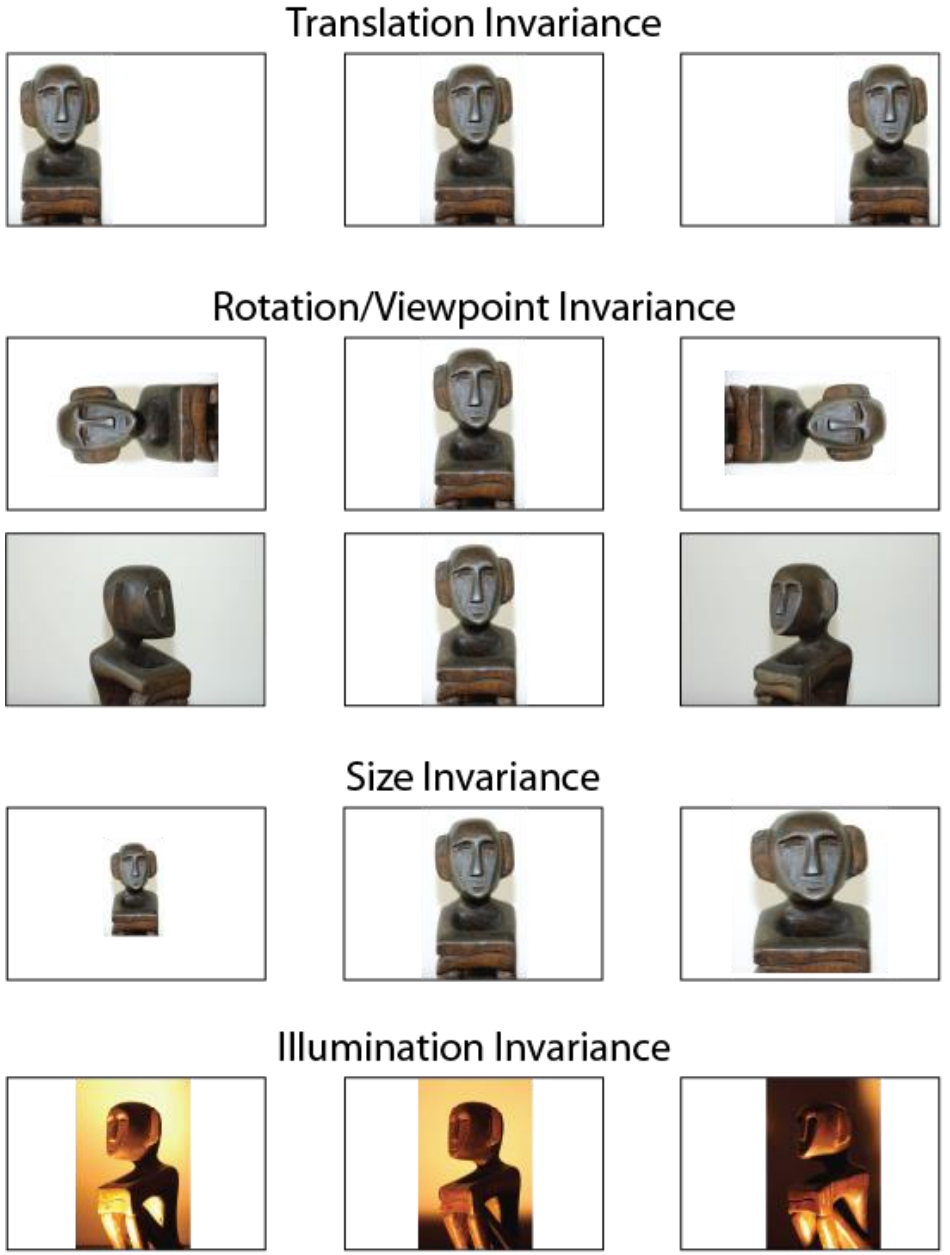
Dense for images?

- Spatial sensitiveness
  - MLPs are not translation invariant

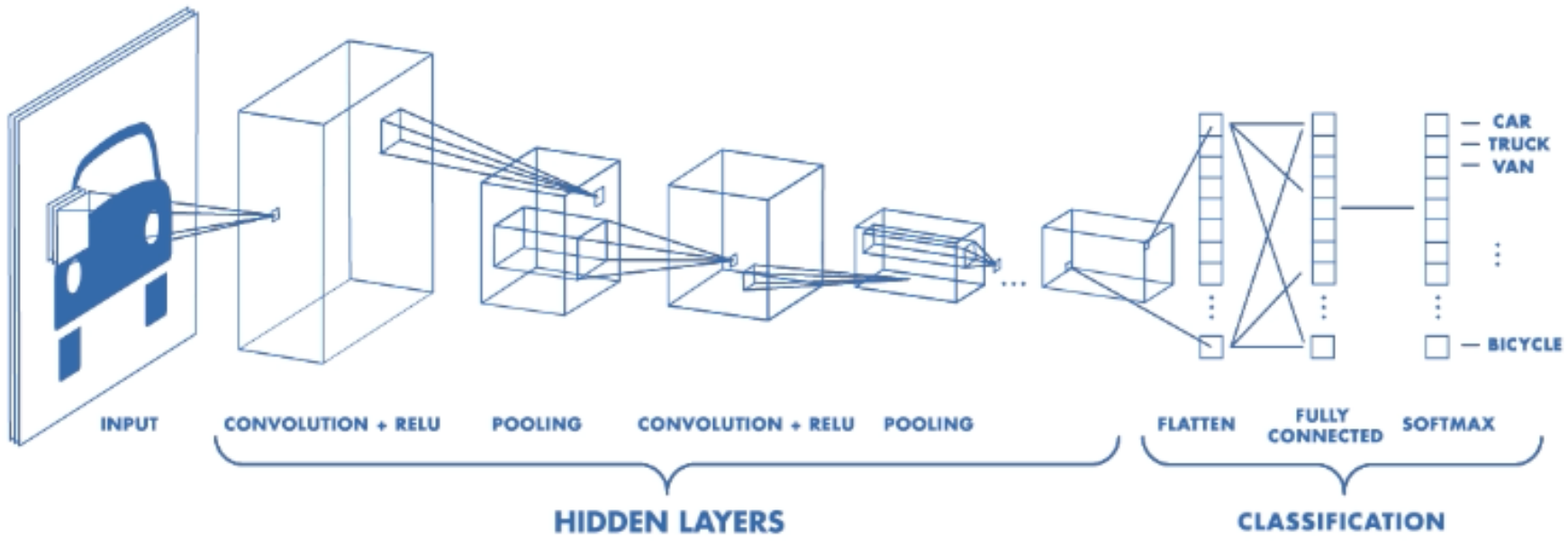


Dense for images?

- Spatial sensitiveness
  - MLPs are not translation invariant

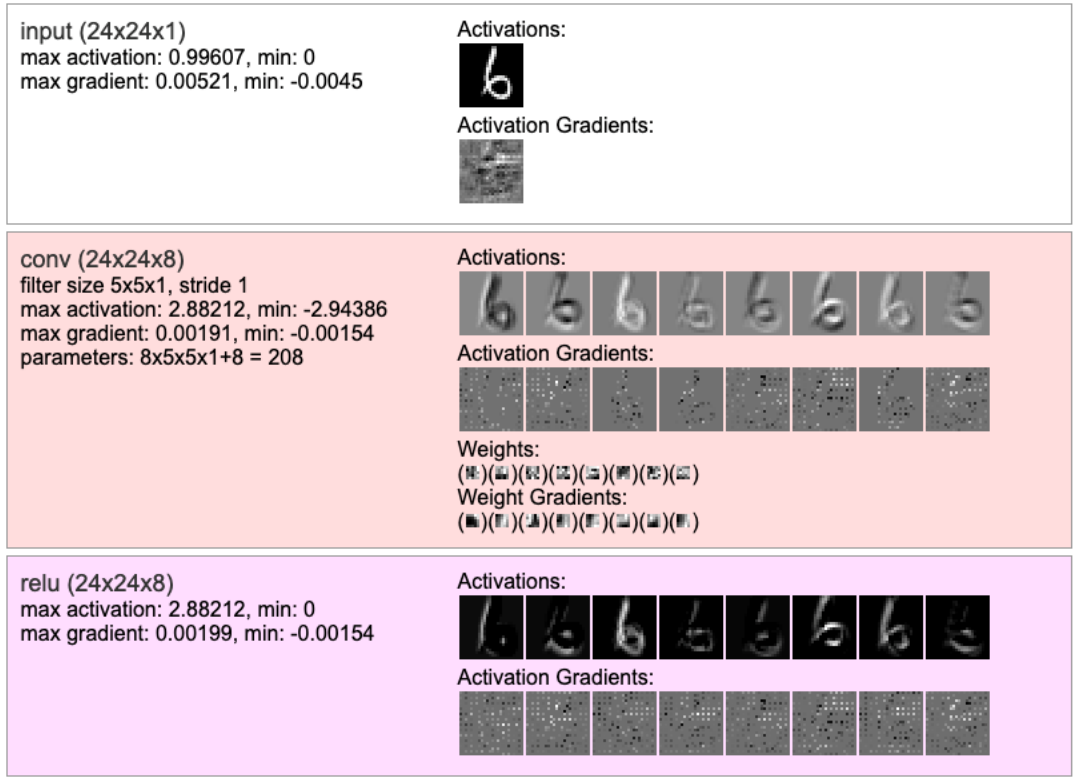


ConvNets: Overall architecture



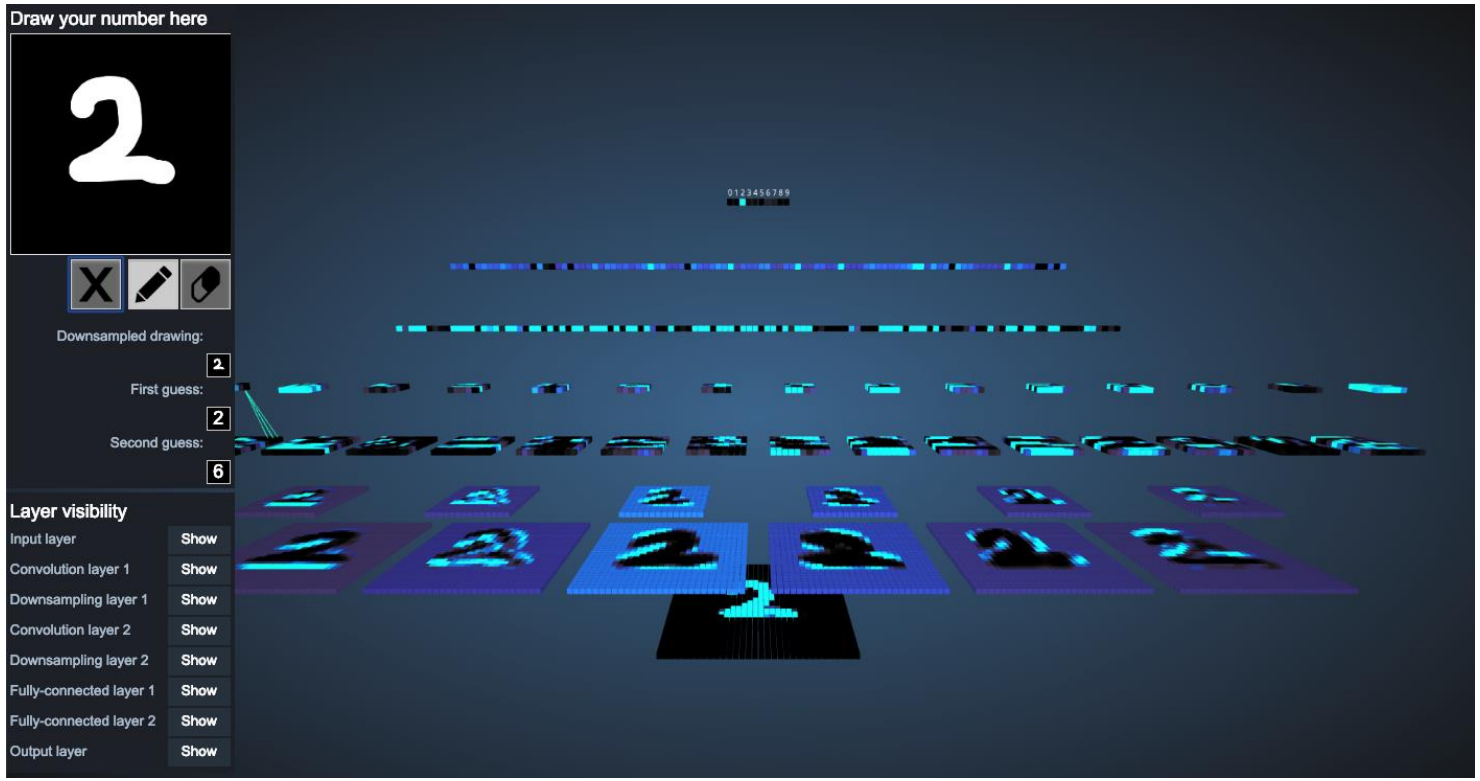
ConvNets in practice!

- https://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html



ConvNets in practice!

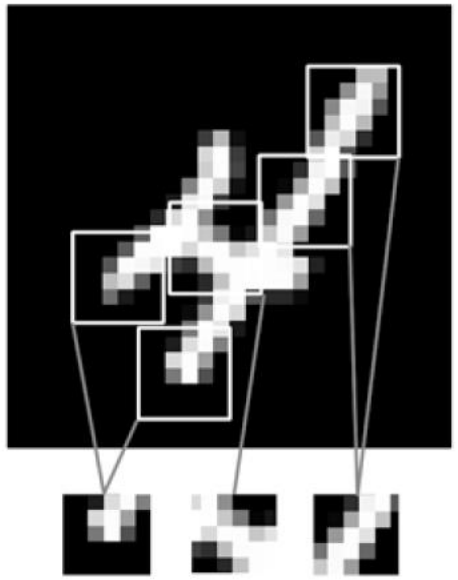
- https://adamharley.com/nn_vis/



Convolution operation

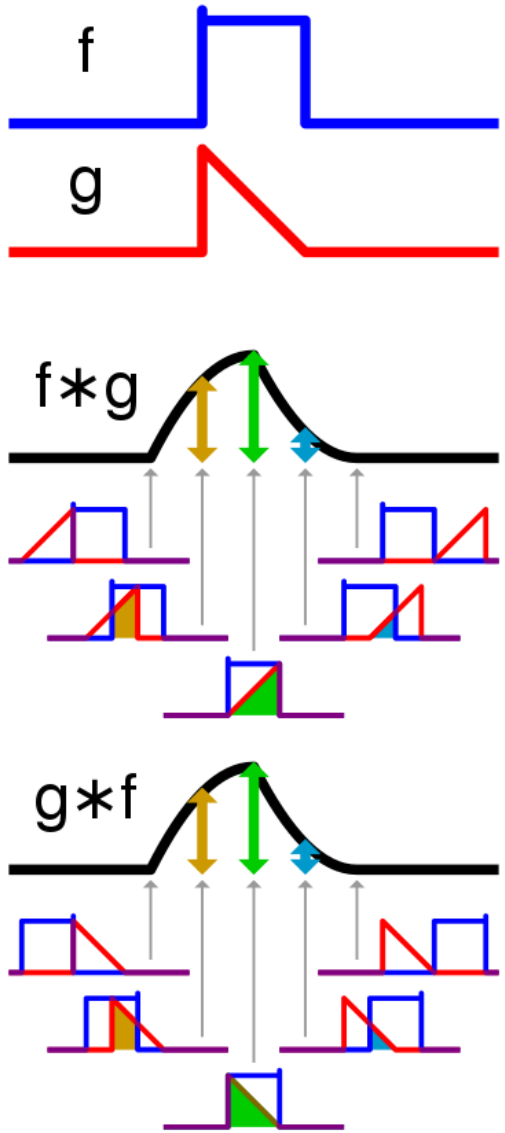
The fundamental difference between Dense and CNN:

- Dense layers learn global patterns in their input feature space
  - For example, for an MNIST digit patterns involving all pixels
- CNNs learn local patterns: in the case of images, patterns found in small 2D windows of the inputs.



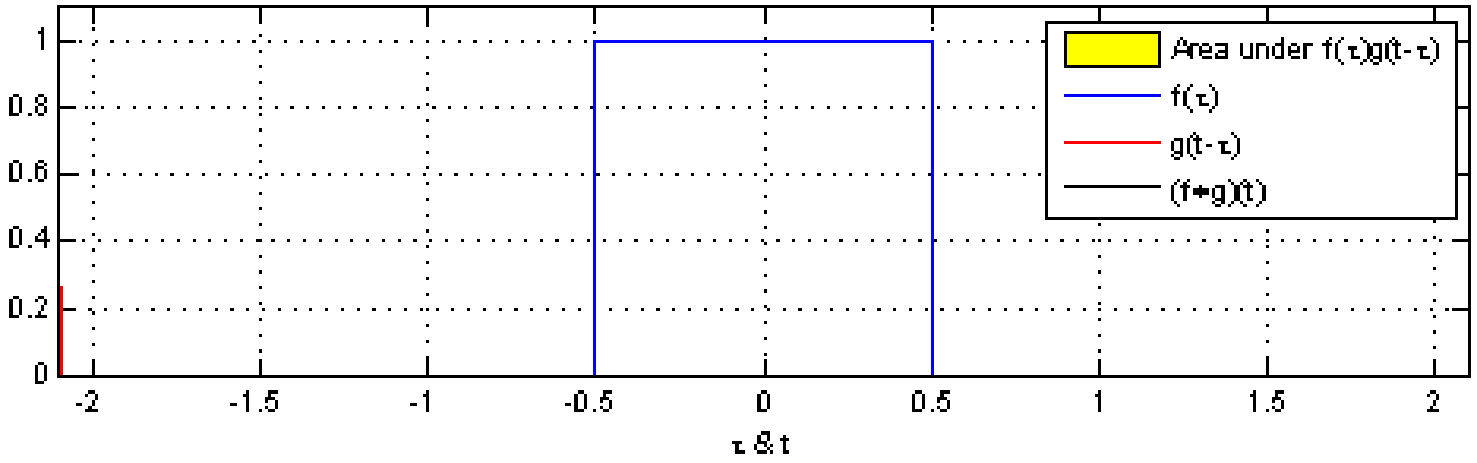
Convolution operation

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau) g(t - \tau)\, d\tau$$



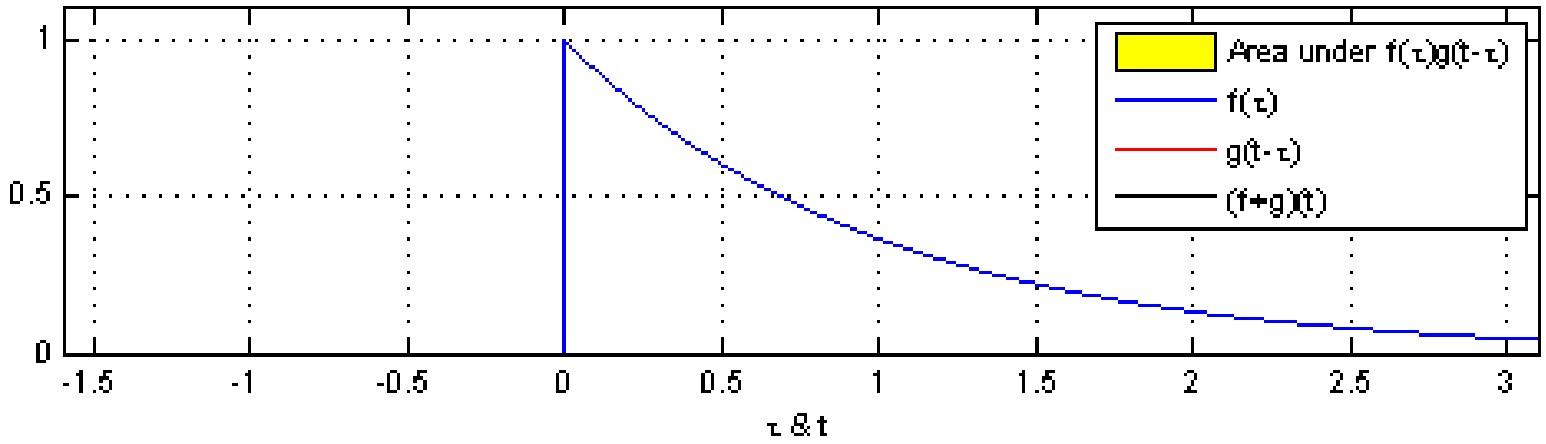
https://en.wikipedia.org/wiki/Convolution

Convolution operation



https://en.wikipedia.org/wiki/Convolution

Convolution operation



Convolution of vectors

Definition: Convolution of Vectors

If the functions f and g are represented as vectors $\mathbf{a} = \begin{bmatrix} a_1 & a_2 & \dots & a_m \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} b_1 & b_2 & \dots & b_n \end{bmatrix}$, then $f * g$ is a vector $\mathbf{c} = \begin{bmatrix} c_1 & c_2 & \dots & c_{m+n-1} \end{bmatrix}$ as follows:

$$c_x = \sum_u a_u b_{x-u+1}$$

where u ranges over all legal subscripts for a_u and b_{x-u+1}, specifically $u = \max(1, x-n+1) \dots \min(x, m)$.

From Frank Keller's slides (University of Edinburgh)

Convolution of vectors

Example

Assume $\mathbf{a} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} 0 \\ 1 \\ 2 \end{bmatrix}$.

Then

$$\mathbf{a} * \mathbf{b} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 + a_2 b_1 \\ a_1 b_3 + a_2 b_2 + a_3 b_1 \\ a_2 b_3 + a_3 b_2 \\ a_3 b_3 \end{bmatrix} = \begin{bmatrix} 1 \cdot 0 \\ 1 \cdot 1 + 0 \cdot 0 \\ 1 \cdot 2 + 0 \cdot 1 + (-1)0 \\ 0 \cdot 2 + (-1)1 \\ (-1)2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ -1 \\ -2 \end{bmatrix}.$$

From Frank Keller's slides (University of Edinburgh)

Convolution operation

Definition: Convolution of Matrices

If the functions f and g are represented as the $n \times m$ matrix A and the $k \times l$ matrix B, then $f * g$ is an $(n + k - 1) \times (m + l - 1)$ matrix C:

$$c_{xy} = \sum_u \sum_v a_{uv} b_{x-u+1,y-v+1}$$

where u and v range over all legal subscripts for a_{uv} and $b_{x-u+1,y-v+1}$.

From Frank Keller's slides (University of Edinburgh)

Convolution operation

Example

Let $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ and $B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \end{bmatrix}$.

Then for $C = A * B$, the entry $c_{33} = a_{11}b_{33} + a_{12}b_{32} + a_{13}b_{31} + a_{21}b_{23} + a_{22}b_{22} + a_{23}b_{21} + a_{31}b_{13} + a_{32}b_{12} + a_{33}b_{11}$.

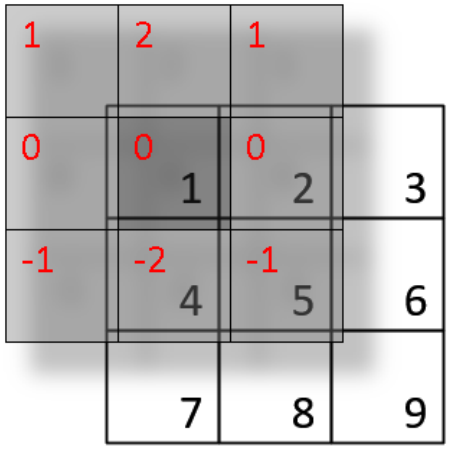
Here, B could represent an image, and A could represent a kernel performing an image operation, for instance.

From Frank Keller's slides (University of Edinburgh)

Convolution operation

1	2	3
4	5	6
7	8	9

n \ m	-1	0	1
-1	-1	-2	-1
0	0	0	0
1	1	2	1

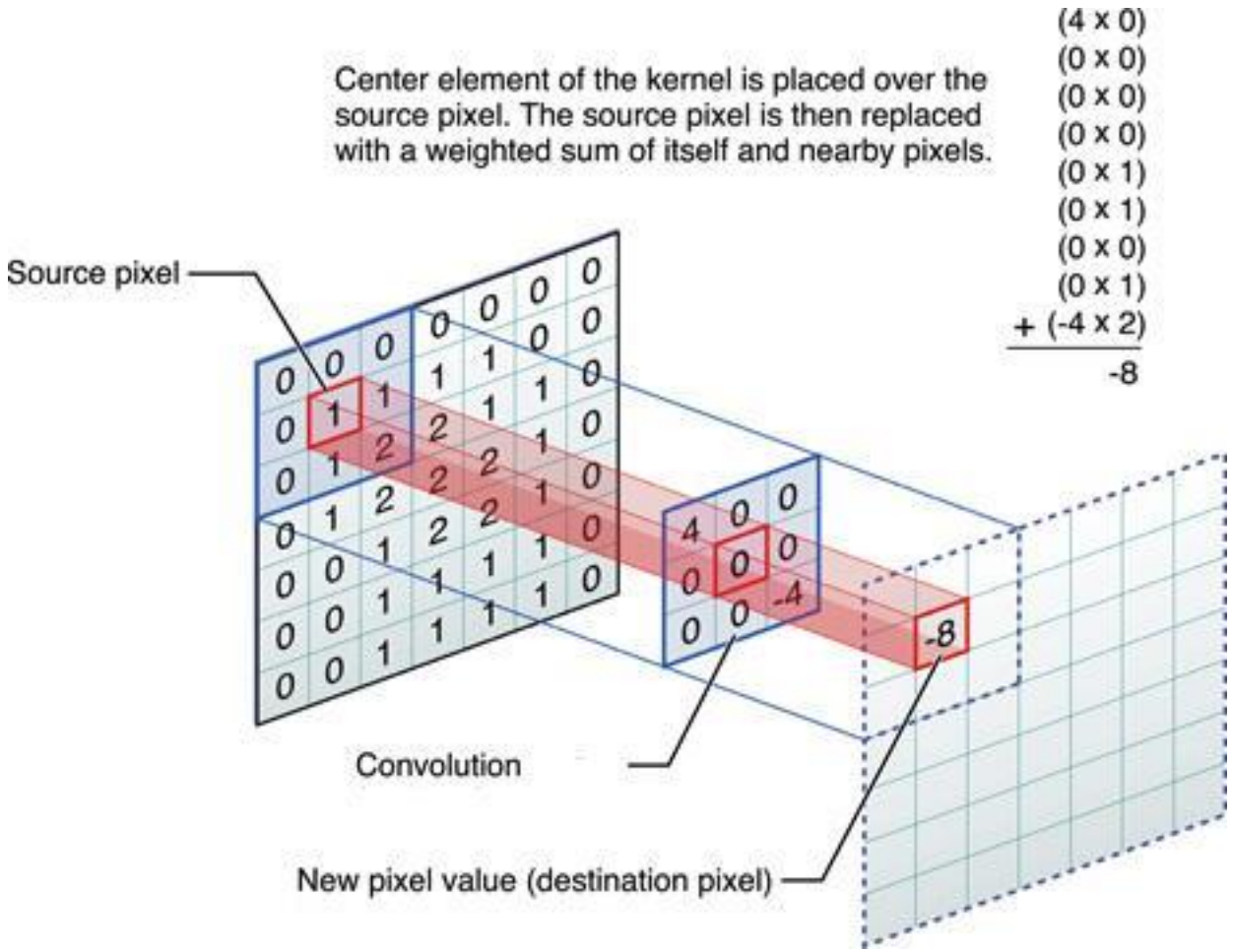


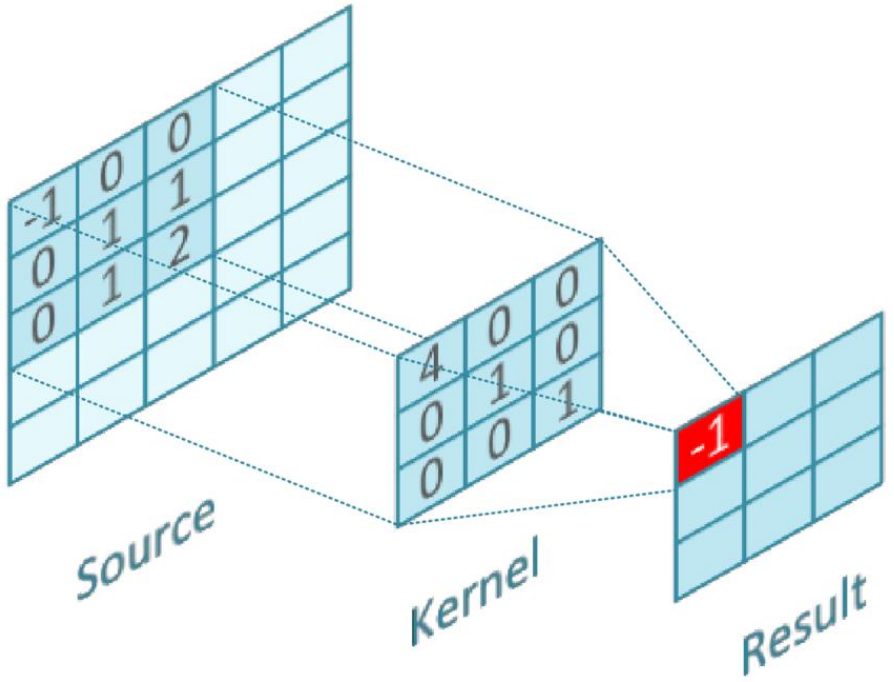
$$
\begin{aligned}
y[0,0] &= \sum_j \sum_i x[i,j] \cdot h[0-i, 0-j] \\
&= \quad x[-1,-1] \cdot h[1,1] + x[0,-1] \cdot h[0,1] + x[1,-1] \cdot h[-1,1] \\
&\quad + x[-1,0] \cdot h[1,0] \quad + x[0,0] \cdot h[0,0] \quad + x[1,0] \cdot h[-1,0] \\
&\quad + x[-1,1] \cdot h[1,-1] + x[0,1] \cdot h[0,-1] + x[1,1] \cdot h[-1,-1] \\
&= \quad 0 \cdot 1 + 0 \cdot 2 + 0 \cdot 1 \\
&\quad + 0 \cdot 0 + 1 \cdot 0 + 2 \cdot 0 \\
&\quad + 0 \cdot (-1) + 4 \cdot (-2) + 5 \cdot (-1) \\
&= -13
\end{aligned}
$$

-13	-20	-17
-18	-24	-18
13	20	17

source

Convolution operation





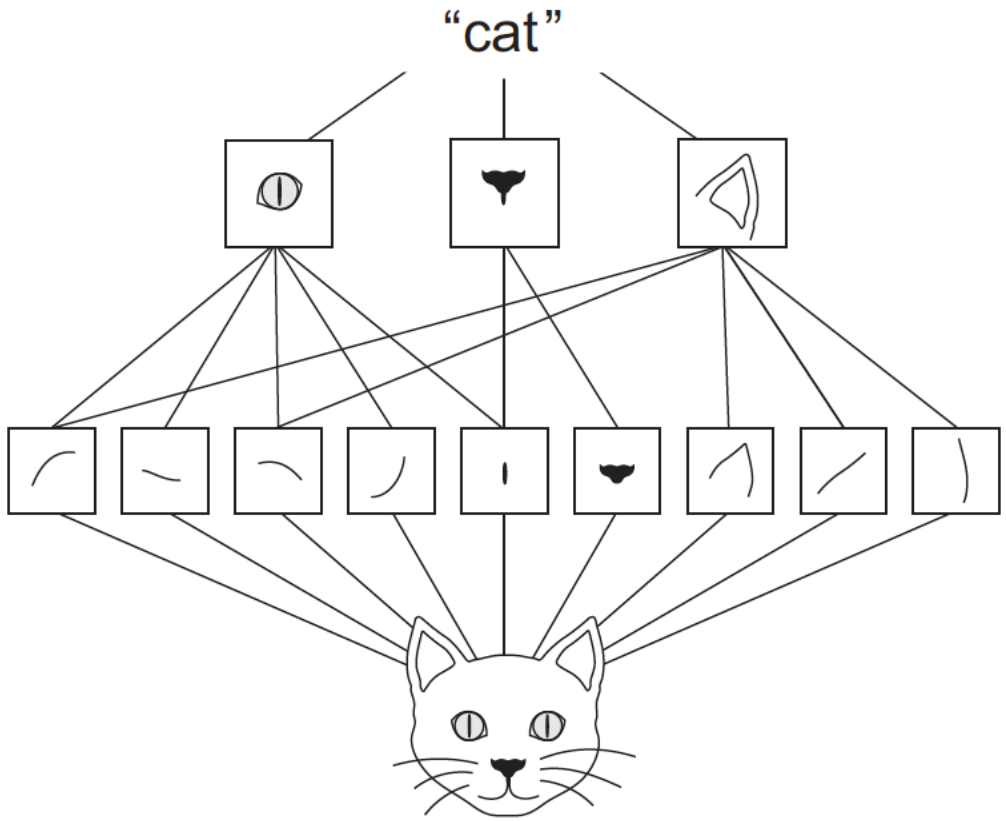
Illustrations from Madhushree Basavarajaiah and 10.13140/RG.2.2.11700.35207

CNNS: characteristics

1. The patterns they learn are translation invariant
   - Unlike Dense, a convnet can recognize a pattern anywhere in the image
   - *The visual world is fundamentally translation invariant*

CNNS: characteristics

2. They can learn spatial hierarchies of patterns
   - A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on

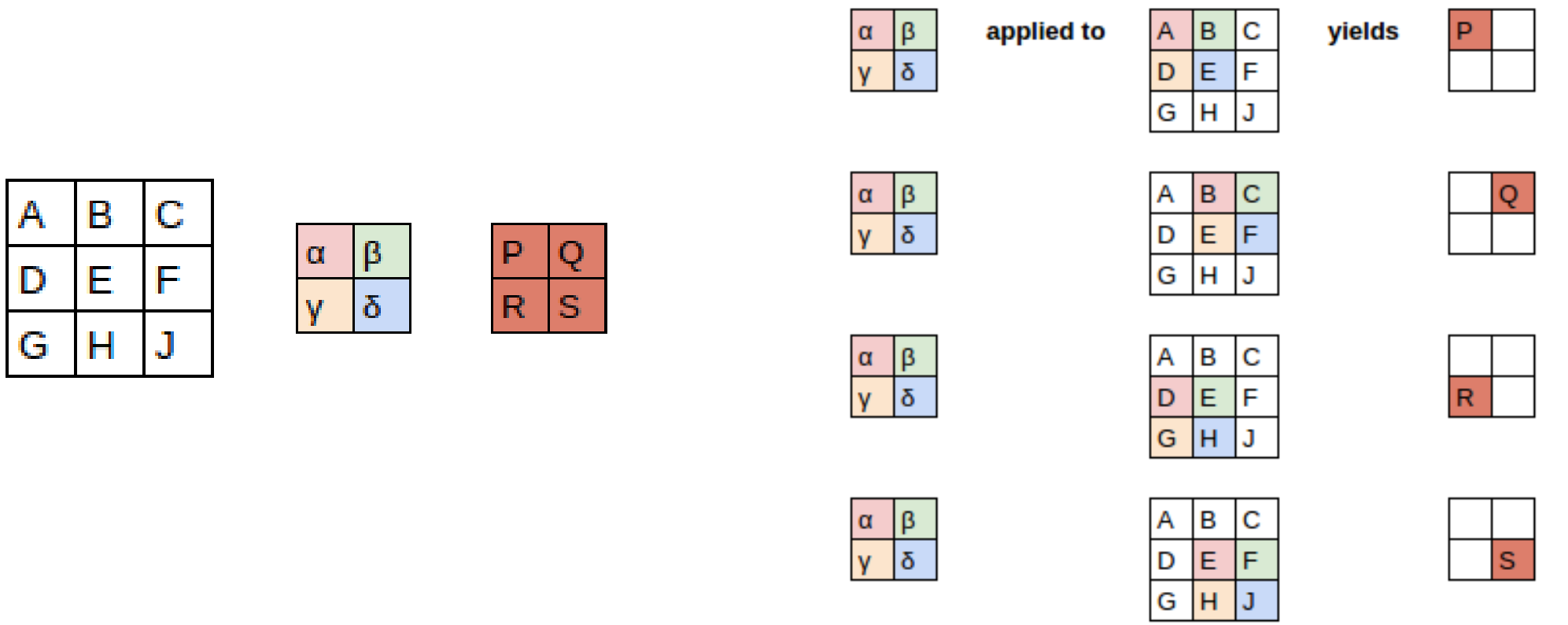


Convolutions

- Convolutions operate over 3D tensors, called *feature maps*, with two spatial axes (*height* and *width*) as well as a *depth* axis (also called the *channels* axis).

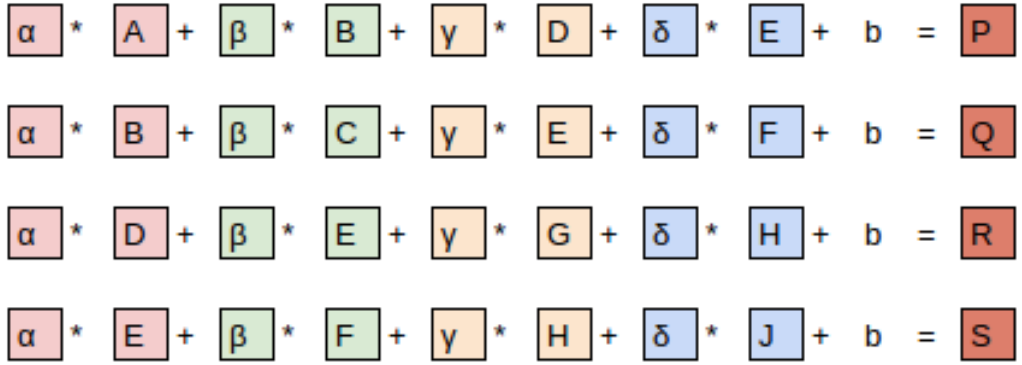
- The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an *output feature map*.

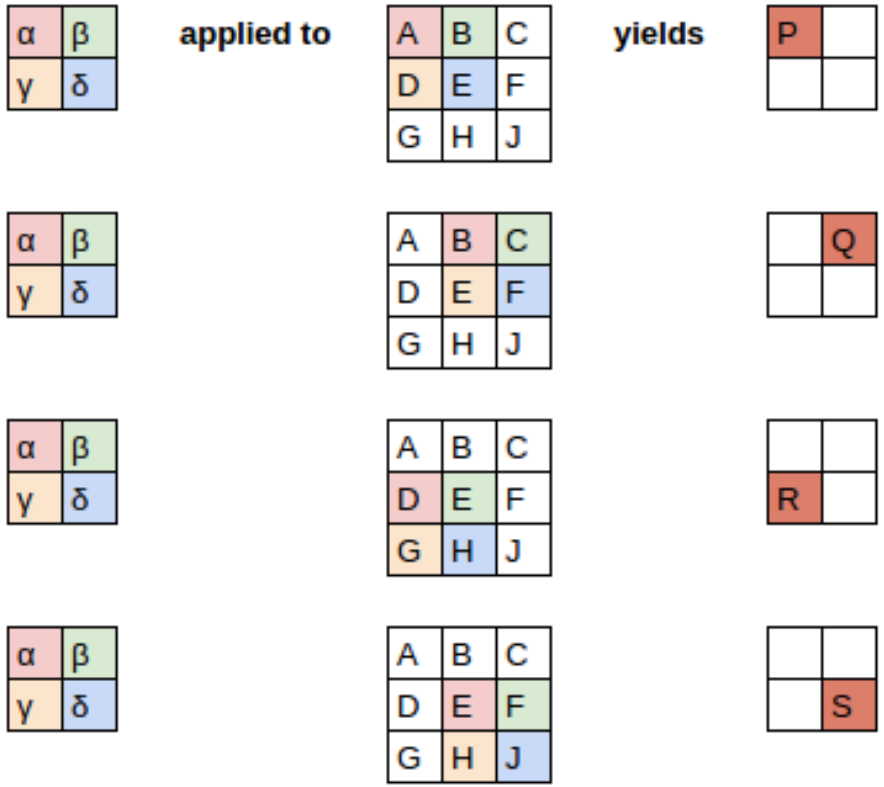
Convolution computation



By https://medium.com/@matthewkleinsmith

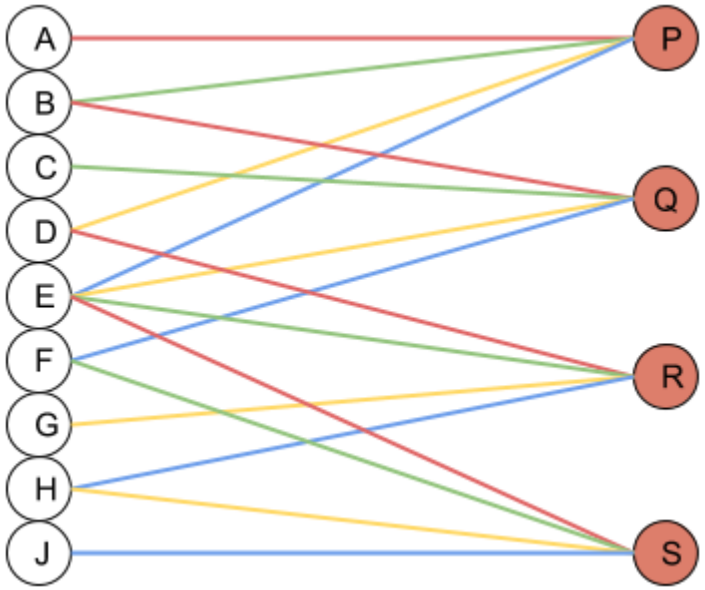
Convolution computation





By https://medium.com/@matthewkleinsmith

Convolution computation



$$\alpha A+\beta B+\gamma D+\delta E+b = P$$
$$\alpha B+\beta C+\gamma E+\delta F+b = Q$$
$$\alpha D+\beta E+\gamma G+\delta H+b = R$$
$$\alpha E+\beta F+\gamma H+\delta J+b = S$$

By https://medium.com/@matthewkleinsmith

Convolution computation

α	β	0	γ	δ	0	0	0	0
0	α	β	0	γ	δ	0	0	0
0	0	0	α	β	0	γ	δ	0
0	0	0	0	α	β	0	γ	δ

A B C D E F G H J

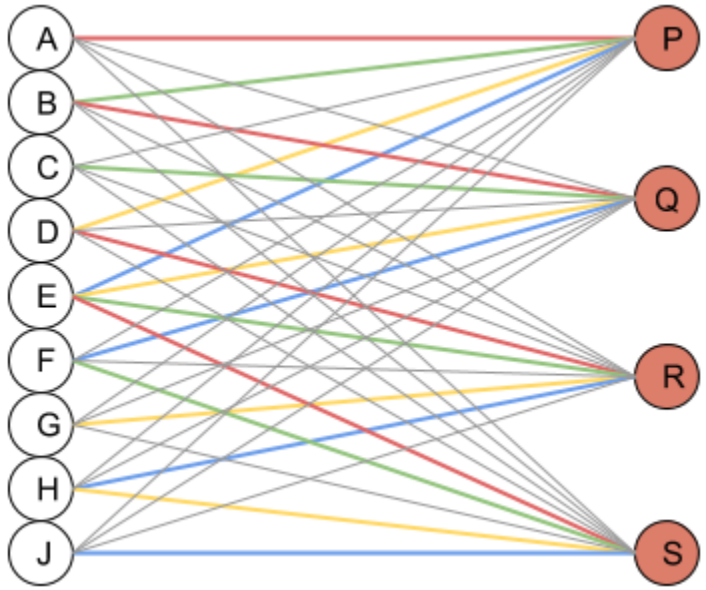
* | A | B | C | D | E | F | G | H | J |

\+ | b | b | b | b |

= | αA+βB+0C+γD+δE+0F+0G+0H+0J+b |
| 0A+αB+βC+0D+γE+δF+0G+0H+0J+b |
| 0A+0B+0C+αD+βE+0F+γG+δH+0J+b |
| 0A+0B+0C+0D+αE+βF+0G+γH+δJ+b |

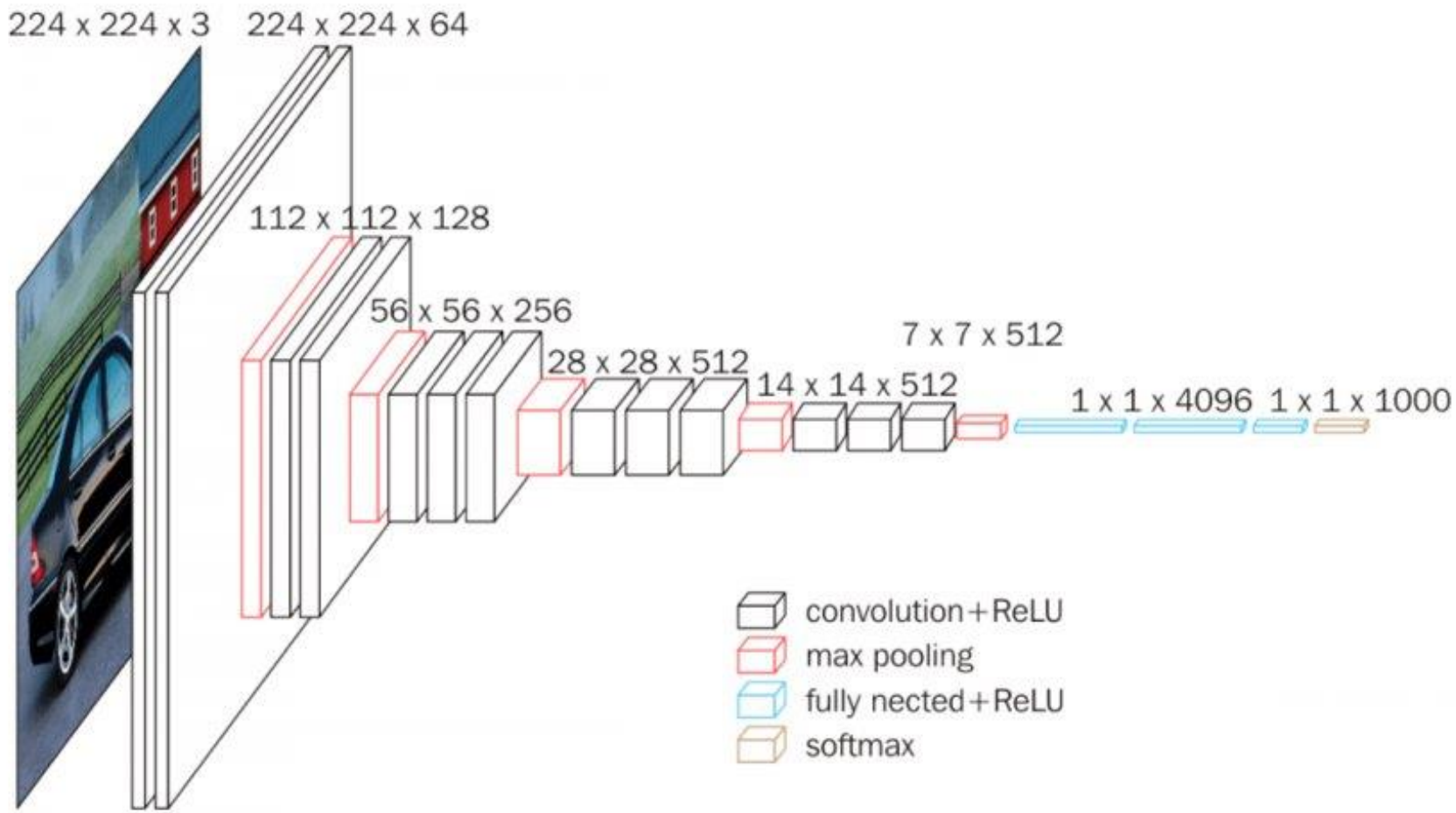
= | αA+βB+γD+δE+b |
| αB+βC+γE+δF+b |
| αD+βE+γG+δH+b |
| αE+βF+γH+δJ+b |

= | P |
| Q |
| R |
| S |



By https://medium.com/@matthewkleinsmith

CNNs


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution + ReLU
max pooling
fully nected + ReLU
softmax

Convolution layer

- It has a set of learnable filters
- Every filter is small spatially, but extends through the full depth of the input volume.
  - For instance, 5x5x3
- During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position.
- As a result, we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position.

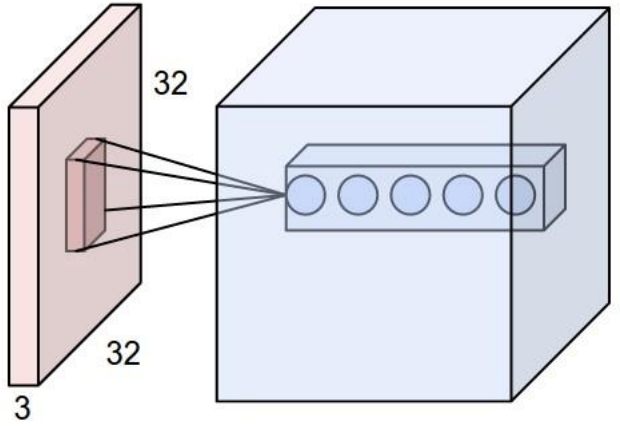
Convolution layer

- Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network.

- Each filter will produce a separate 2-dimensional activation map.

- We stack these to create the output volume.

Example 1



- **Depth**: a hyperparamter, the number of filters, each looking at a different feature in input, 5 in the example

- **Stride**: the step size for sliding the filter.

- **Zero-padding**: pad the input volume across borders with zeros.
  - We can control the spatial size of the output volumes

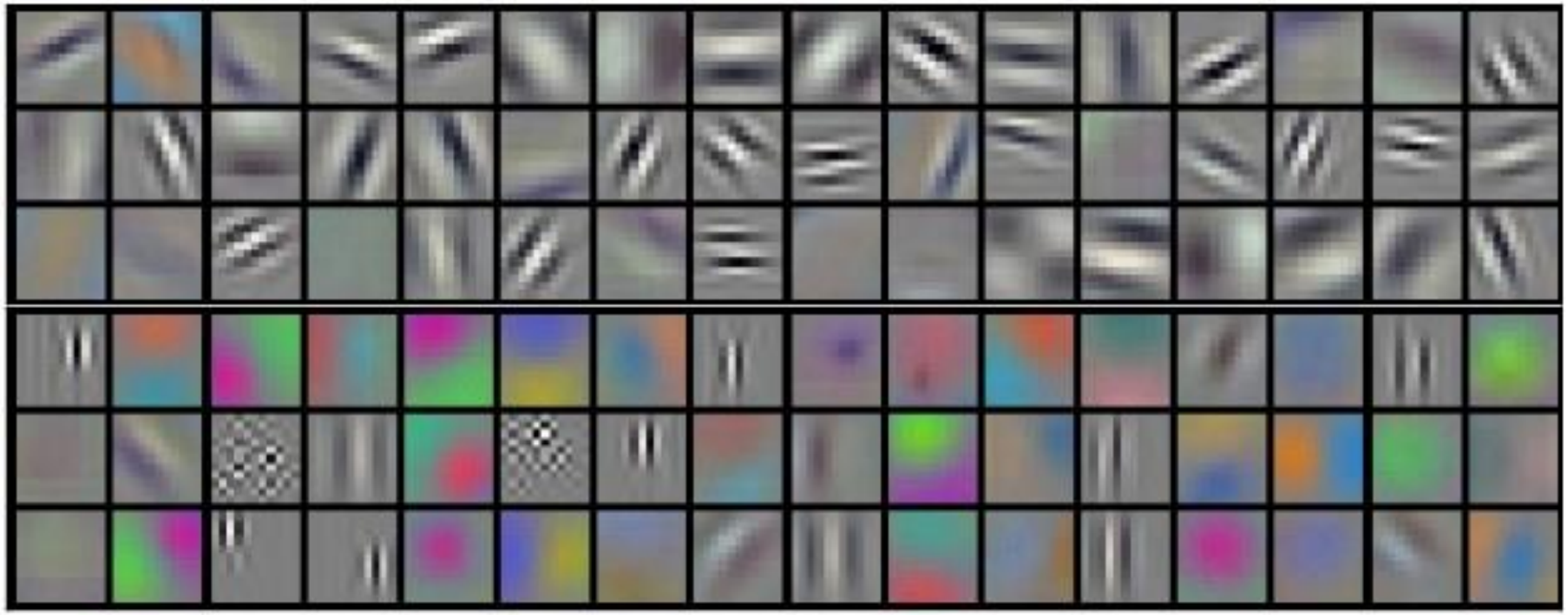
Why padding?

To avoid:

- Shrinking outputs

- Loosing information on corners of the image
  - The pixel in the corner will only get covers one time but if you take the middle pixel it will get covered more than once

Filters

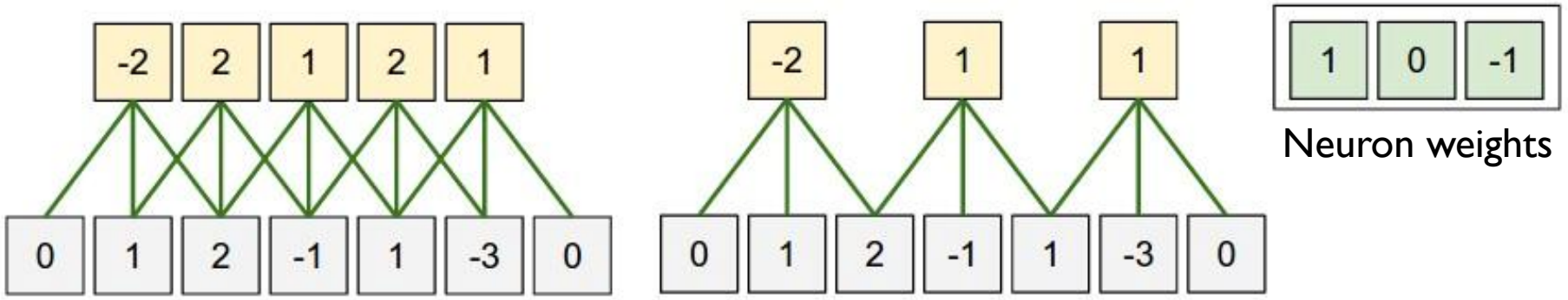


Convolution: output size

- Input volume size: W
- Receptive field (filter) size: F
- Stride: S
- Zero padding: P

$$\frac{W - F + 2P}{S} + 1$$

Quiz



- Compute output size for both cases.

- Left: receptive field size of F = 3, the input size is W = 5, and there is zero padding of P = 1: (5 - 3 + 2)/1+1 = 5
- Right: stride is 2: (5 - 3 + 2)/2+1 = 3

$$\frac{W - F + 2P}{S} + 1$$

Convolution: number of parameters

- Image size: [227x227x3]
- F=11, S=4, P=0, K=96
- Output size?
  - 55*55*96
- Total number of weights?
  - Each connected to a region of 11x11x3
  - 55*55*96 = 290,400
  - 11*11*3 = 363 and 1 for bias
  - 290400 * 364 = 105,705,600 !! Too many

CNN: parameter sharing

- We assume that: if one feature is useful to compute at some spatial position (x,y), then it should also be useful to compute at a different position (x2,y2).

- We are going to constrain the neurons in each depth slice to use the same weights and bias
  - e.g. a volume of size [55x55x96] has 96 *depth slices*, each of size [55x55]

- Therefore: our example will have 96 unique set of weights
  - 96*11*11*3 = 34,848 + 96 biases

CNN: parameter sharing

Sometimes the parameter sharing assumption may not make sense, when?

When the input images to a ConvNet have some specific centered structure, where we should expect, for example, that completely different features should be learned on one side of the image than another.

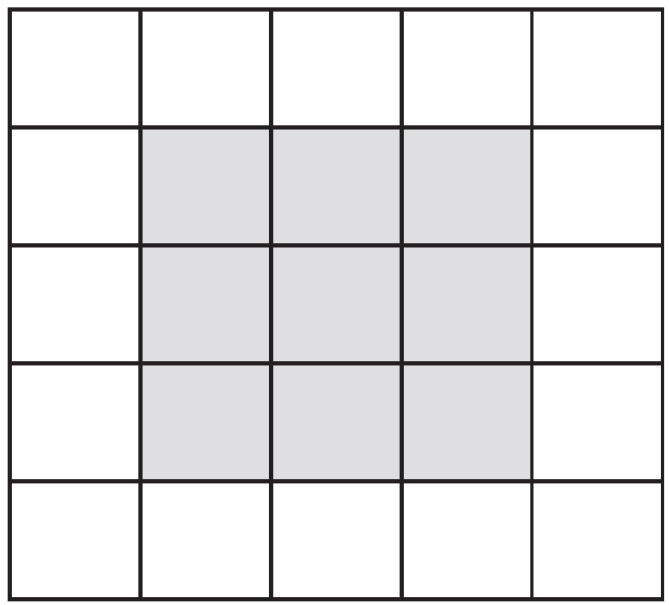
One practical example is when the input are faces that have been centered in the image. You might expect that different eye-specific or hair-specific features could (and should) be learned in different spatial locations.

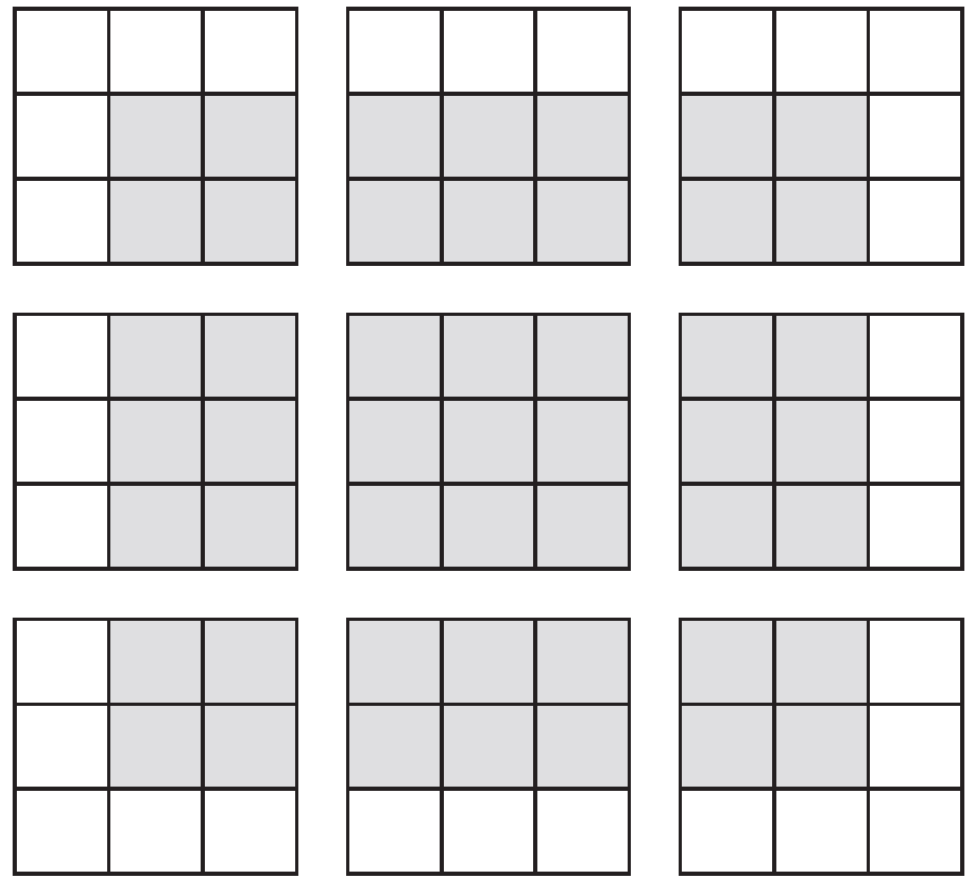
CNN: NOT parameter sharing

The `LocallyConnected2D` layer works similarly to the `Conv2D` layer, except that weights are unshared, that is, a different set of filters is applied at each different patch of the input.

CNN summary

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K, their spatial extent F, stride S, zero padding P
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d^{th} depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d^{th} filter over the input volume with a stride of S, and then offset by d^{th} bias.

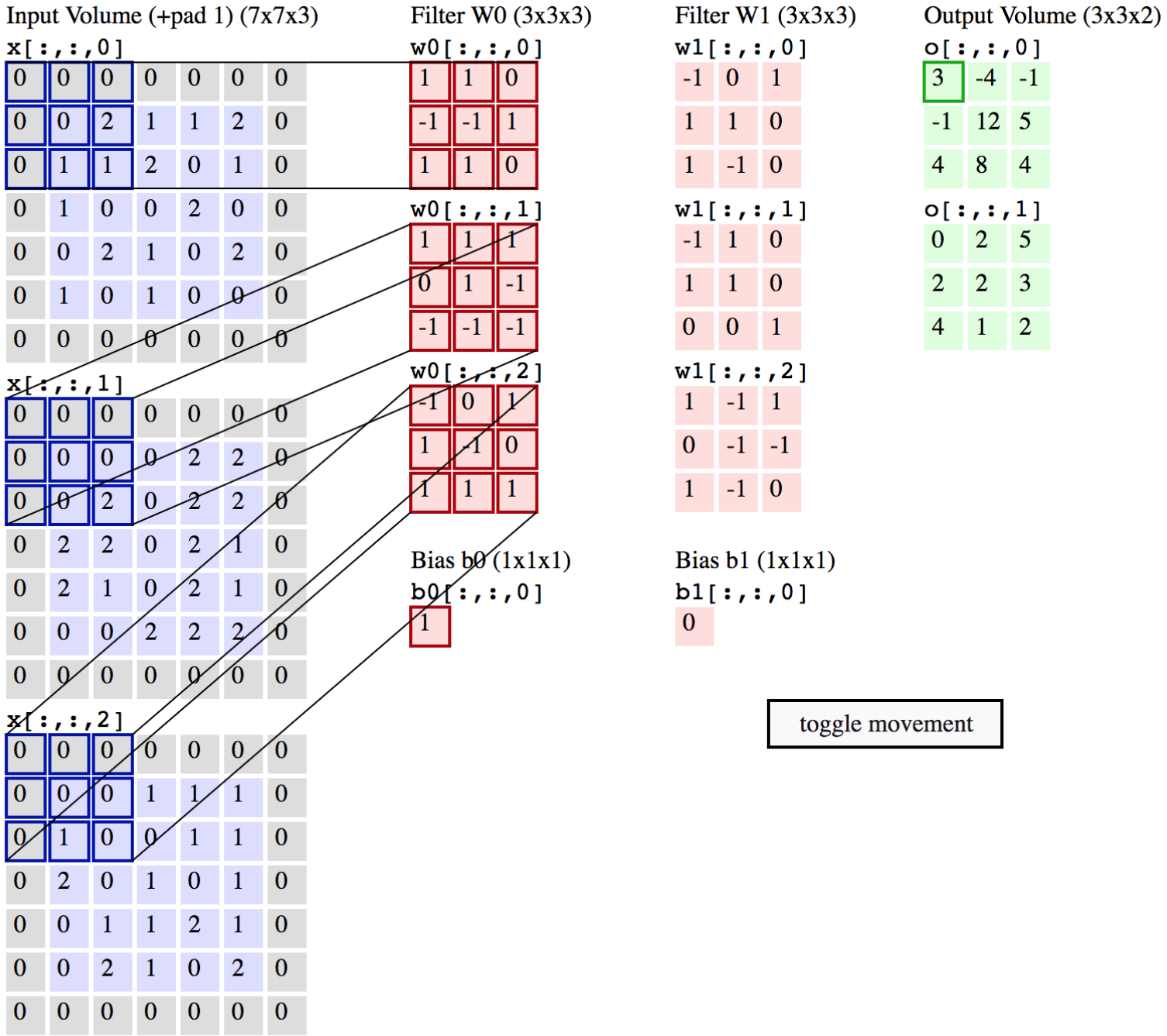




Demo

W_1=5
H_1=5
D_1=3

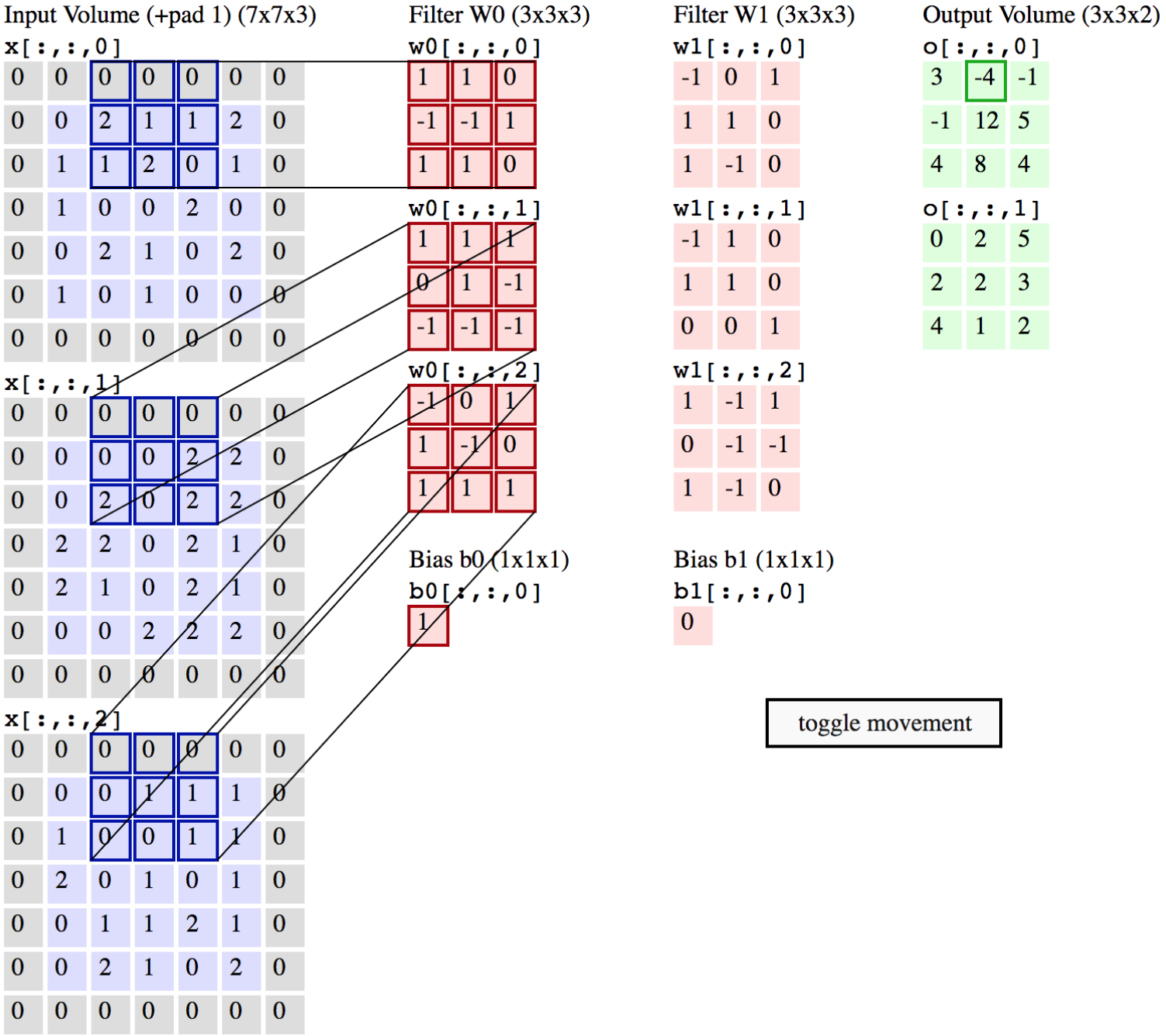
K=2
F=3
S=2
P=1



Demo

W_1=5
H_1=5
D_1=3

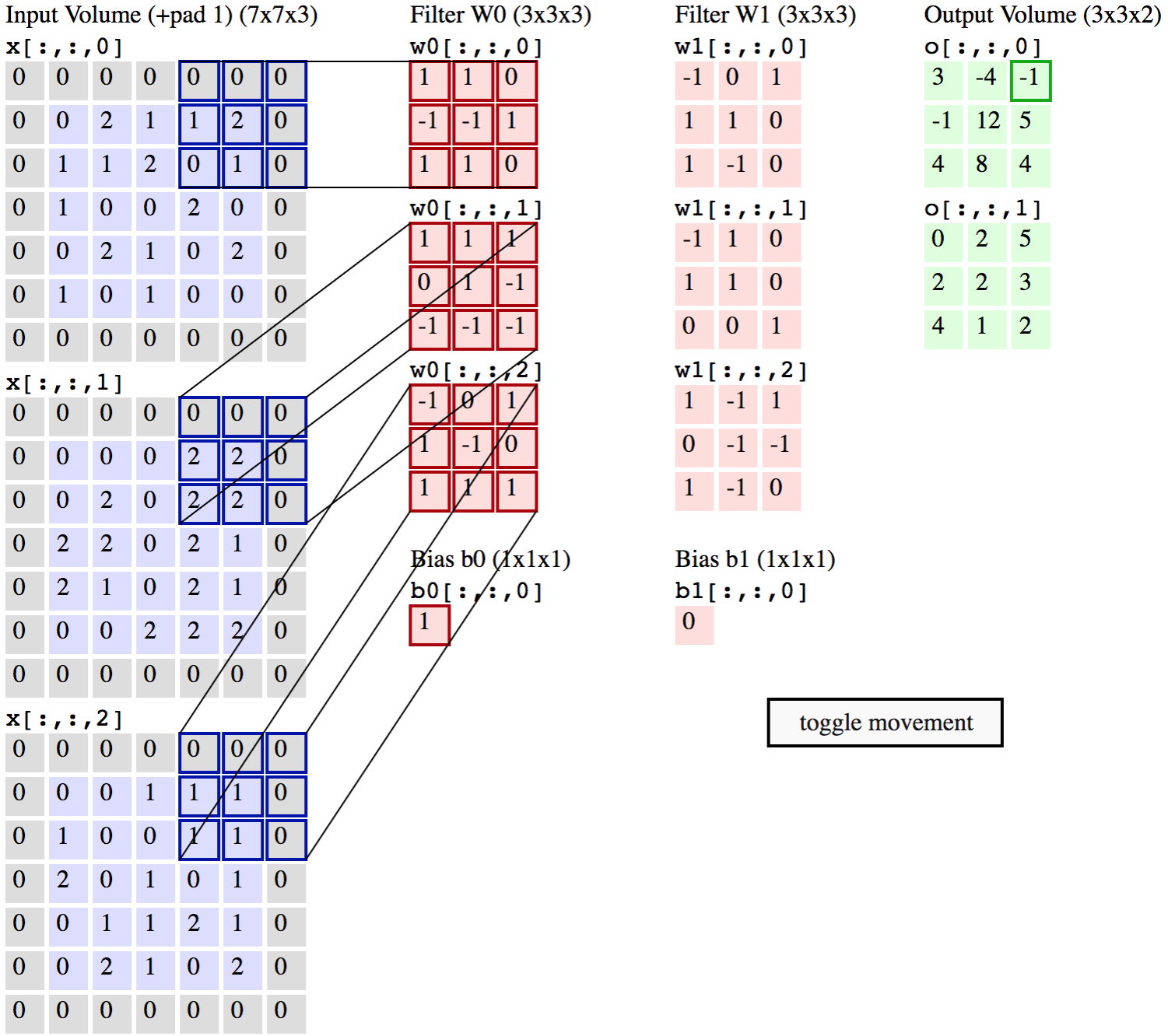
K=2
F=3
S=2
P=1



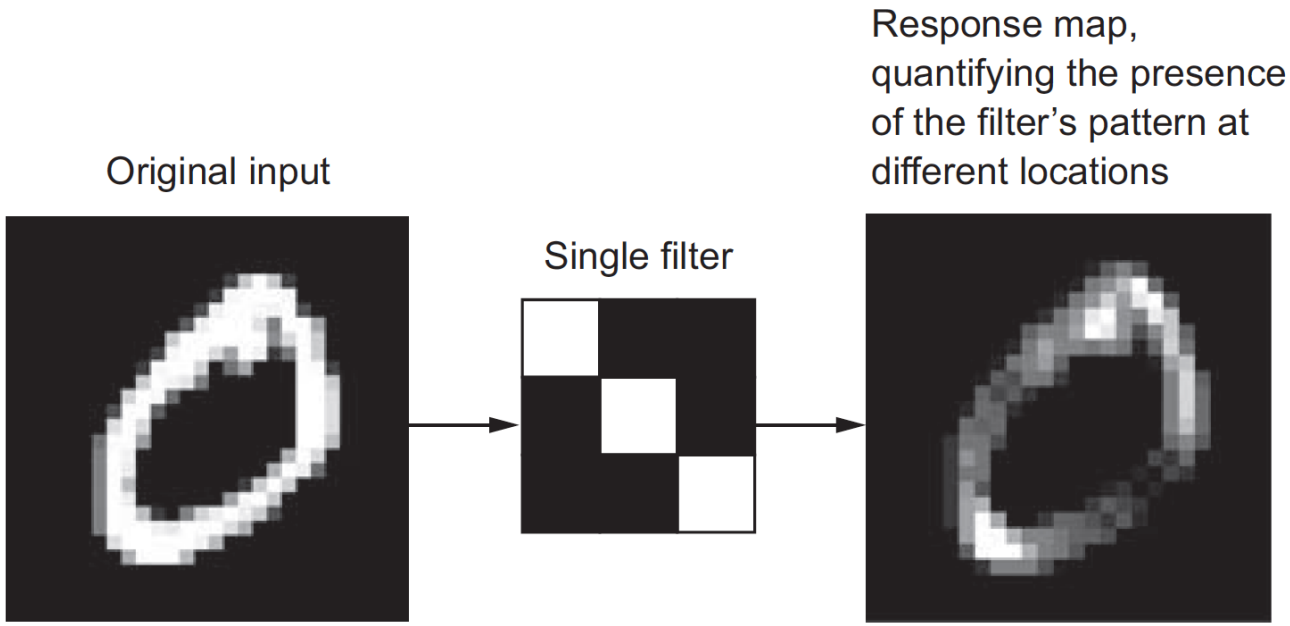
Demo

W_1=5
H_1=5
D_1=3

K=2
F=3
S=2
P=1

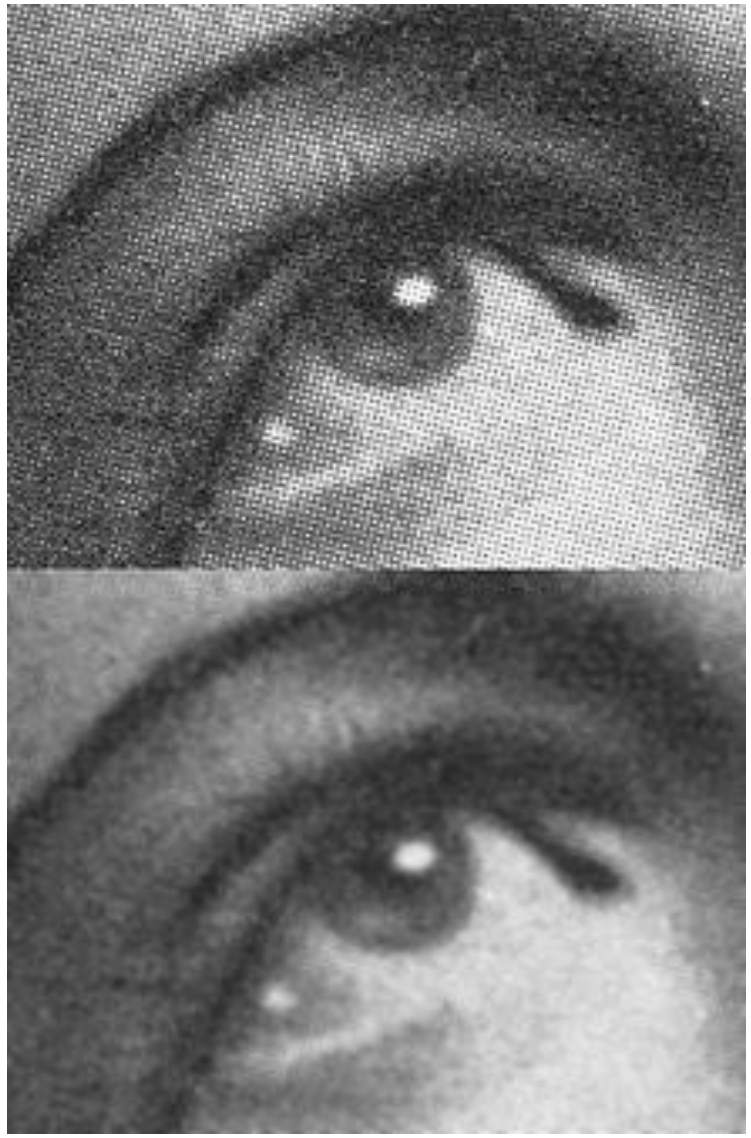


Response map

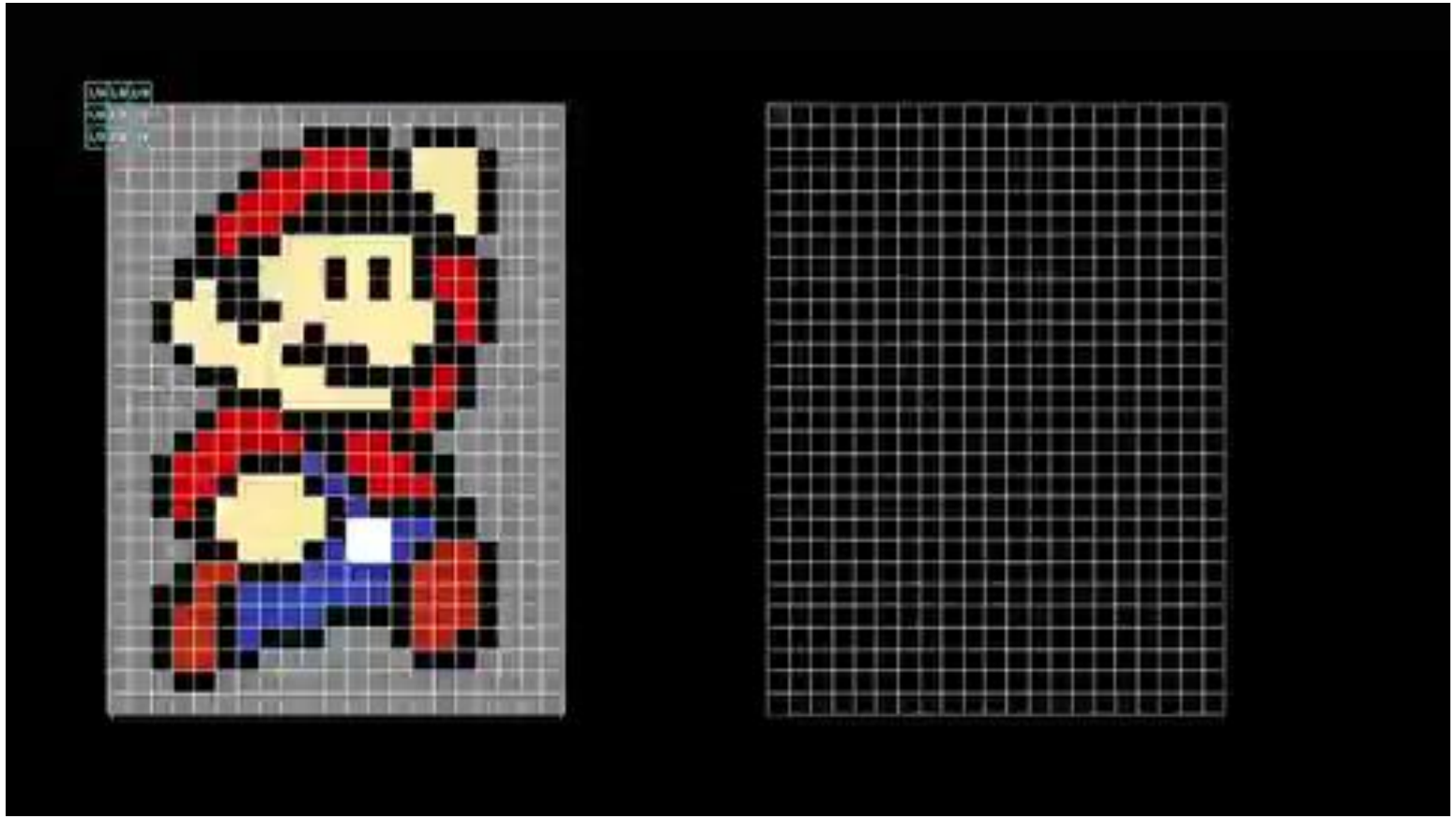


Response map

Gaussian blur and convolution



Response map

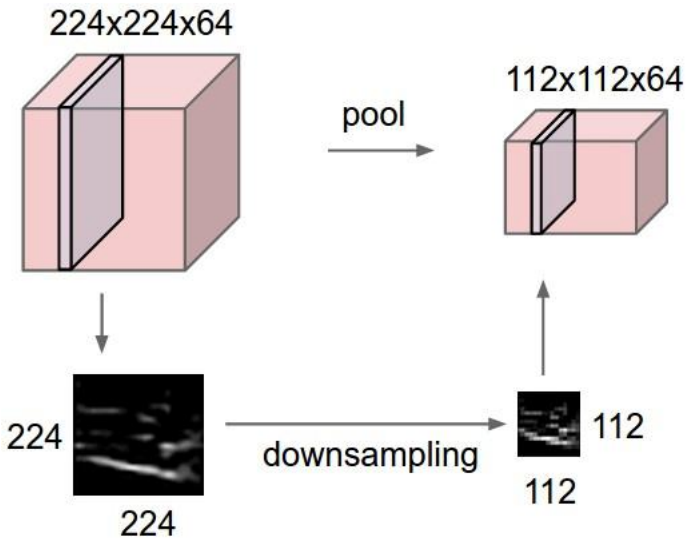


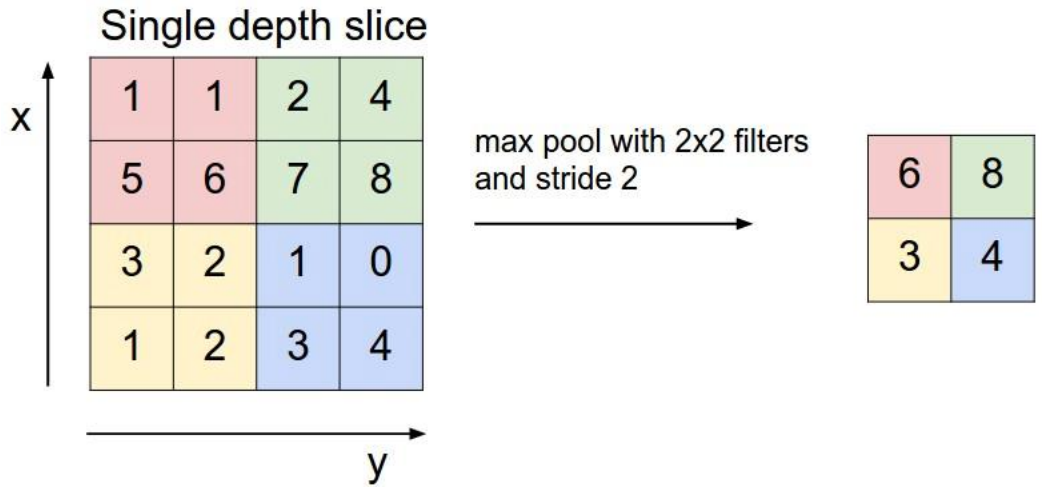
Pooling layer

- Inserted in between Conv layers
- Aim: reduce the spatial size of the representation
  - Hence, reduce the amount of parameters
    - Hence, to control overfitting

- Usually, filters of 2x2, applied with a stride of 2, downsamples every depth in both height and weight, discarding 75% of the activations
  - MAXing over four numbers

Pooling layer



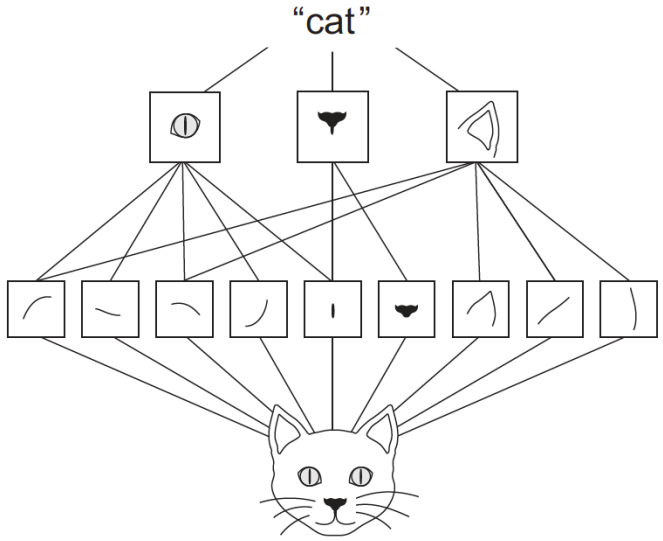


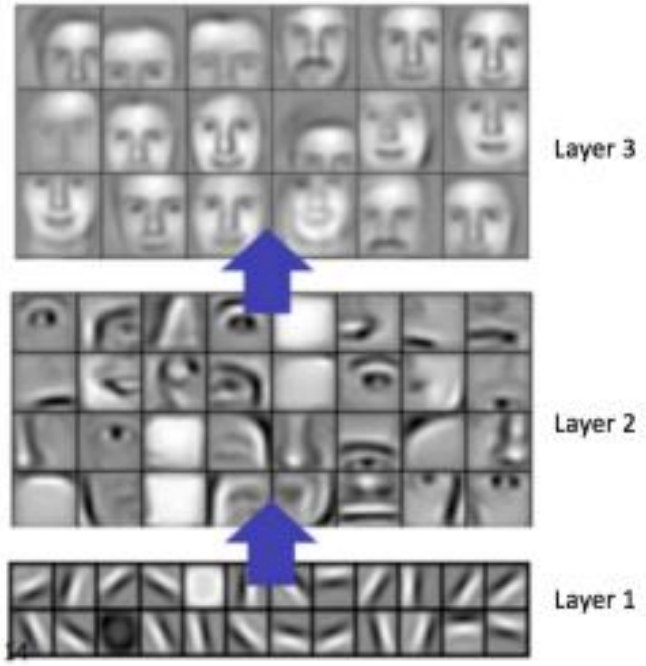
Pooling layer

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - Their spatial extent F
  - The stride S
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W2=(W1-F)/S+1$
  - $H2=(H1-F)/S+1$
  - $D2=D1$
- Introduces zero parameters since it computes a fixed function of the input
- Other forms: *average pooling* or *L2-norm pooling*

Why pooling?

- To make the window larger!



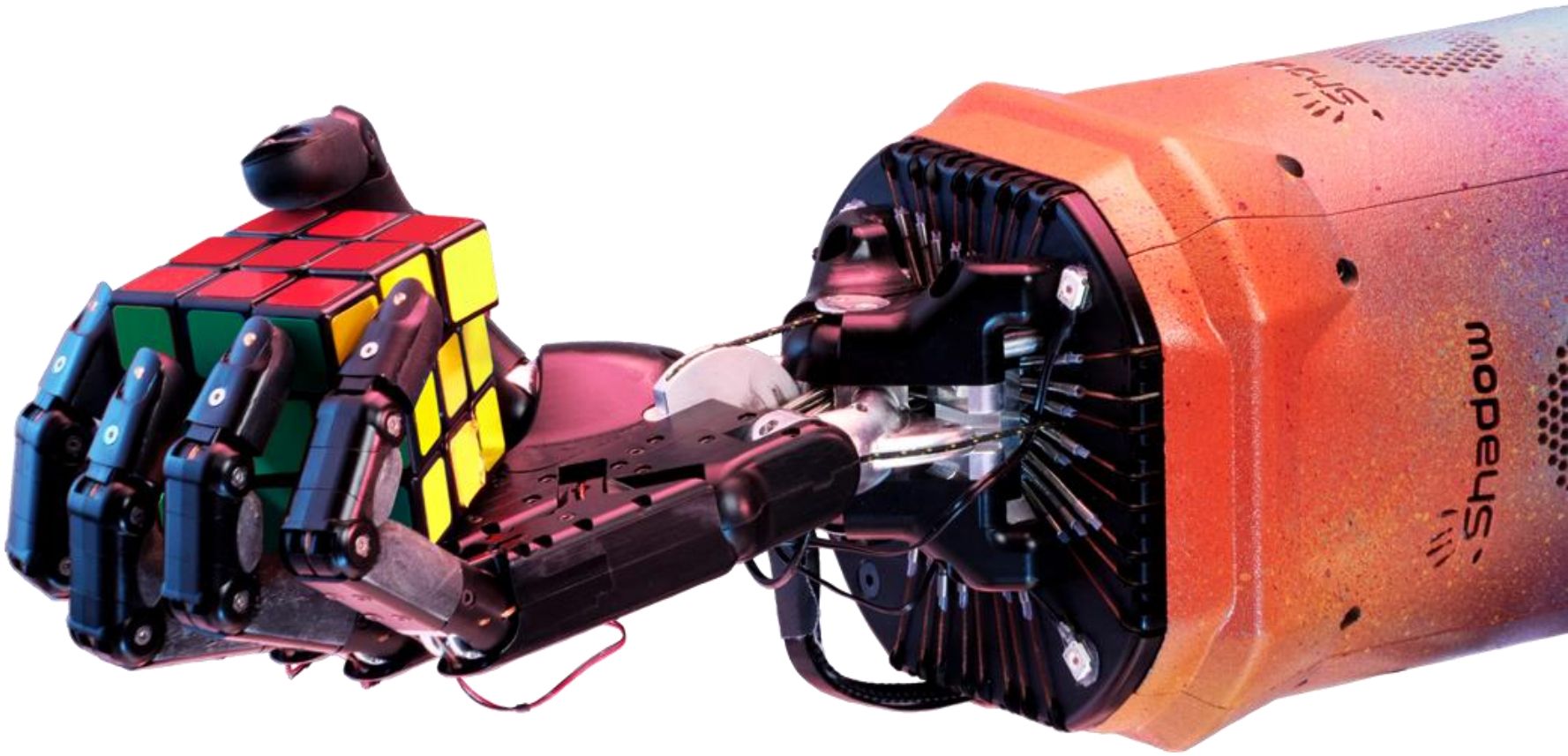


Lee et al (2009)

- To reduce the number of parameters in the Dense layer
- Transformation, translation and scaling invariance

OpenAI

- Solving Rubik's Cube with a Robot Hand



CNN EXPLAINER

https://poloclub.github.io/cnn-explainer/

Backpropagation in CNNs

- Max-pooling

$$X = \begin{bmatrix} 1 & 3 \\ 4 & 2 \end{bmatrix} \quad \rightarrow \quad M = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix}$$

- Mean-pooling

$$dZ = 1 \quad \rightarrow \quad dZ = \begin{bmatrix} 1/4 & 1/4 \\ 1/4 & 1/4 \end{bmatrix}$$

Weight initialization

- As before, Glorot (Xavier)

```
def glorot_normal(shape, name=None):
    ''' Reference: Glorot & Bengio, AISTATS 2010
    '''
    fan_in, fan_out = get_fans(shape)
    s = np.sqrt(2. / (fan_in + fan_out))
    return normal(shape, s, name=name)
```

```
def glorot_uniform(shape, name=None):
    fan_in, fan_out = get_fans(shape)
    s = np.sqrt(6. / (fan_in + fan_out))
    return uniform(shape, s, name=name)
```

- What is the input size?
  - The size of the kernel

Data Augmentation

- Overfitting is caused by having *too few samples* to learn from, rendering you unable to train a model that can **generalize** to new data.

- Data augmentation:

  generating more training data from existing training samples …

  by augmenting the samples via …

  a number of random transformations that yield believable-looking images.

- This helps expose the model to more aspects of the data and generalize better.

Data Augmentation in Keras

```
datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')
```

- **rotation_range** is a value in degrees (0–180), a range within which to randomly rotate pictures.
- **width_shift** and **height_shift** are ranges (as a fraction of total width or height) within which to randomly translate pictures vertically or horizontally.
- **shear_range** is for randomly applying shearing transformations.
- **zoom_range** is for randomly zooming inside pictures.
- **horizontal_flip** is for randomly flipping half the images horizontally—relevant when there are no assumptions of horizontal asymmetry (for example, real-world pictures).
- **fill_mode** is the strategy used for filling in newly created pixels, which can appear after a rotation or a width/height shift.

Augmented Images (samples)

Figure 5.11 Generation of cat pictures via random data augmentation

Data Augmentation

- You can't produce new information, you can only remix existing information.
  - As such, this may not be enough to completely get rid of overfitting.

- To further fight overfitting, you'll also add a `Dropout` layer to your model, right before the densely connected classifier.

Vision – Difficulties in real world applications

An example for detecting "stop sign" in self-driving cars