CONVOLUTIONAL NEURAL NETWORKS

COMP4660/8420

Bio-inspired Computing: Applications and Interfaces

Adapted from lecture notes by Christopher Chow and Josephine Plested

OVERVIEW

Biological inspirations

Structures and layers

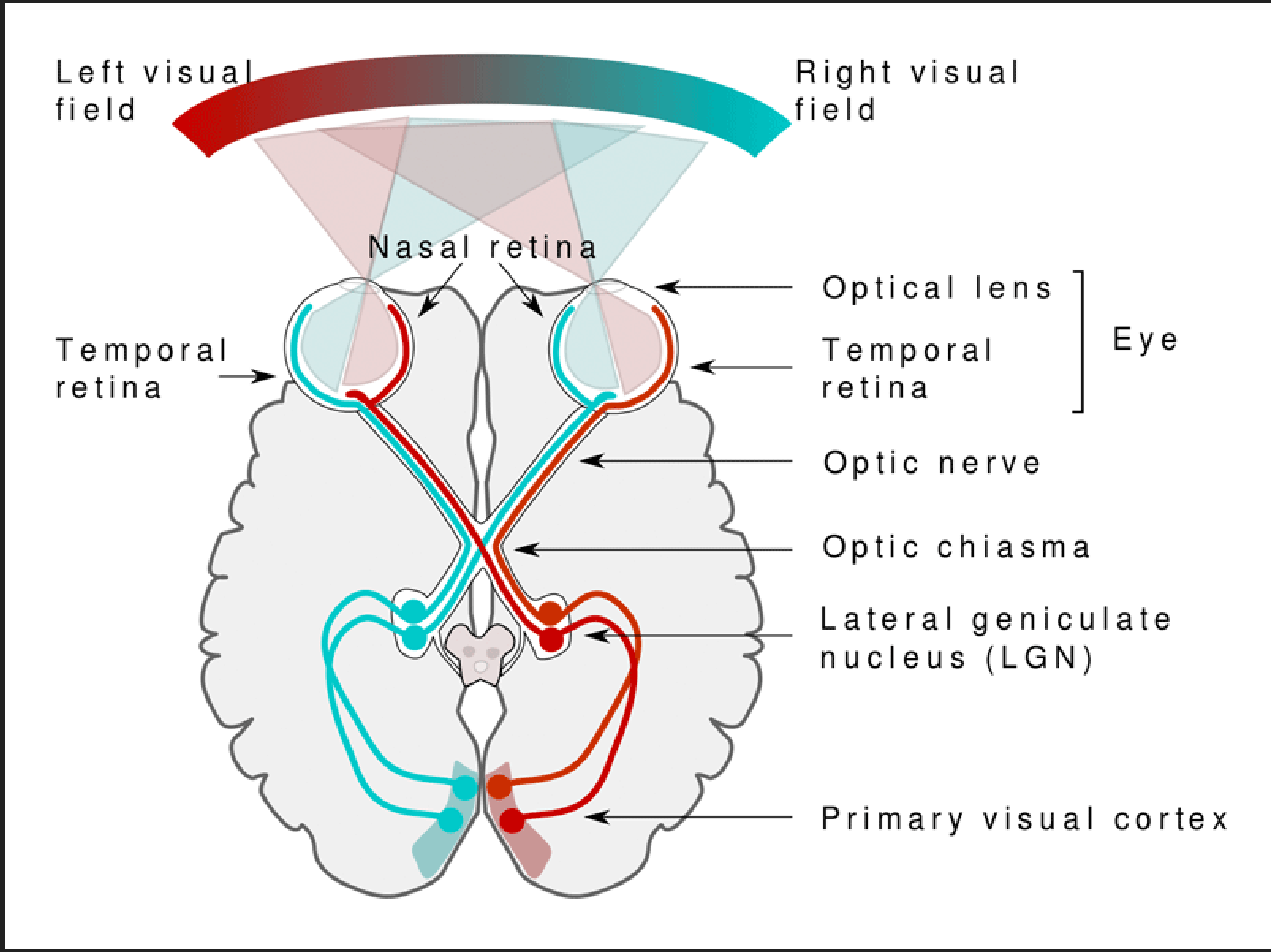
Implementation and examples

PREMISE AND BACKGROUND

CONVOLUTIONAL NEURAL NETWORKS

- CNN or ConvNet
- Successful in spatial problem domains in particular, but increasingly in temporal domains too
- Prominent component of deep learning history
- Common and popular form of deep learning

INSPIRED BY THE VISUAL PATHWAY

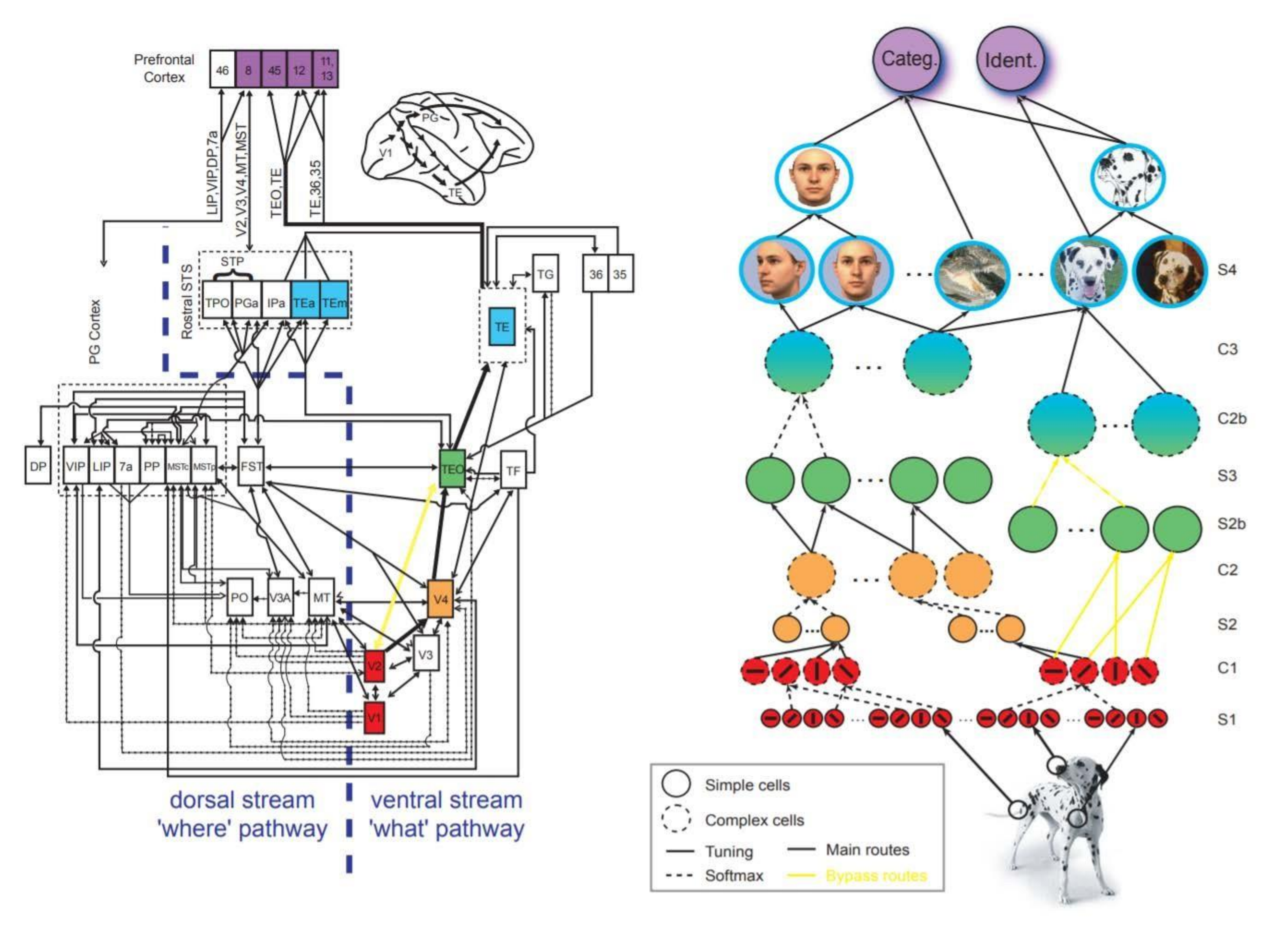


VISUAL CORTEX VENTRAL PATHWAY

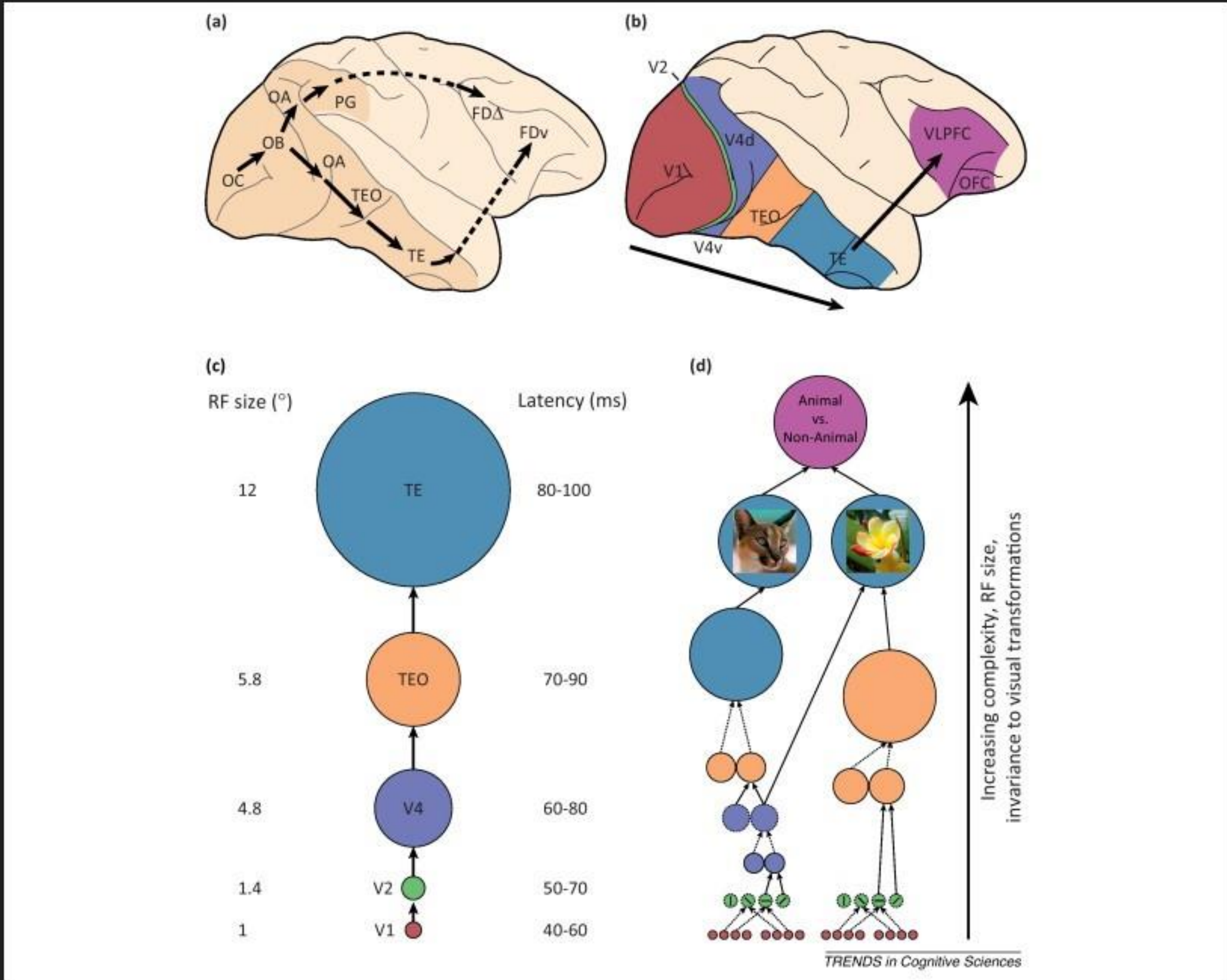
LGN-V1-V2-V4-IT hierarchy

- Lateral geniculate nucleus (LGN)
  - Provides sensory input from retina to brain (visual cortex)
  - Calculates spatial dimensions
- Visual area one of visual cortex (V1)
  - Neurons with similar tuning properties cluster together
  - Edge detection
- Visual area two of visual cortex (V2)
  - Complex properties such as contours and orientation
- Visual area four of visual cortex (V4)
  - More complex properties such as shapes, spatial frequency,
    colour
- Inferior temporal gyrus (IT)
  - Tuned for very complex properties such as faces, objects, patterns

HIERARCHICAL VISUAL PROCESSING

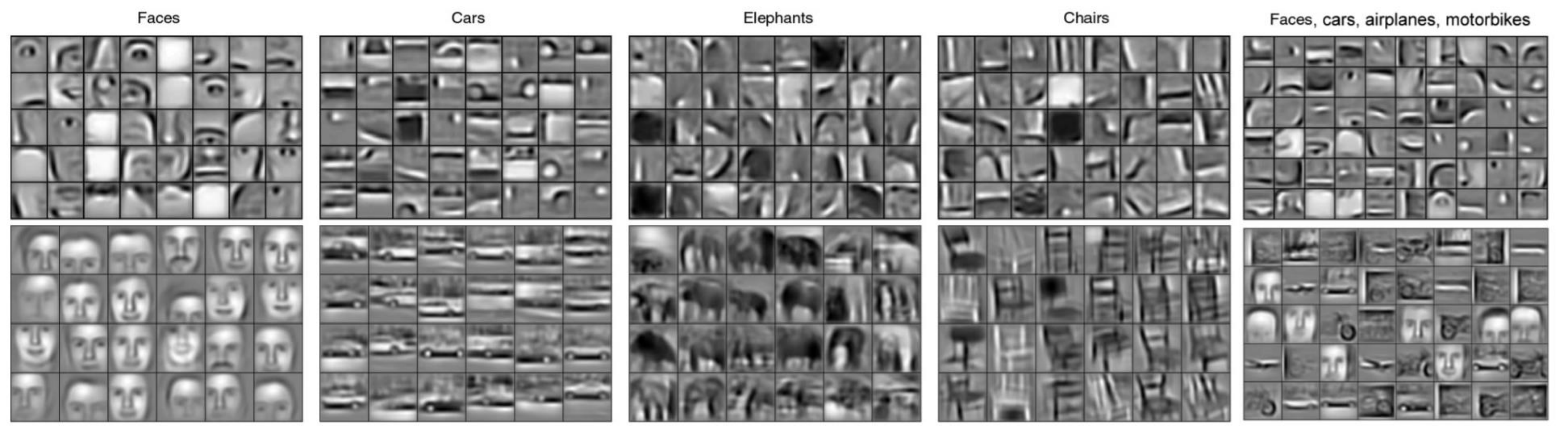


INCREASING RECEPTIVE FIELD SIZE



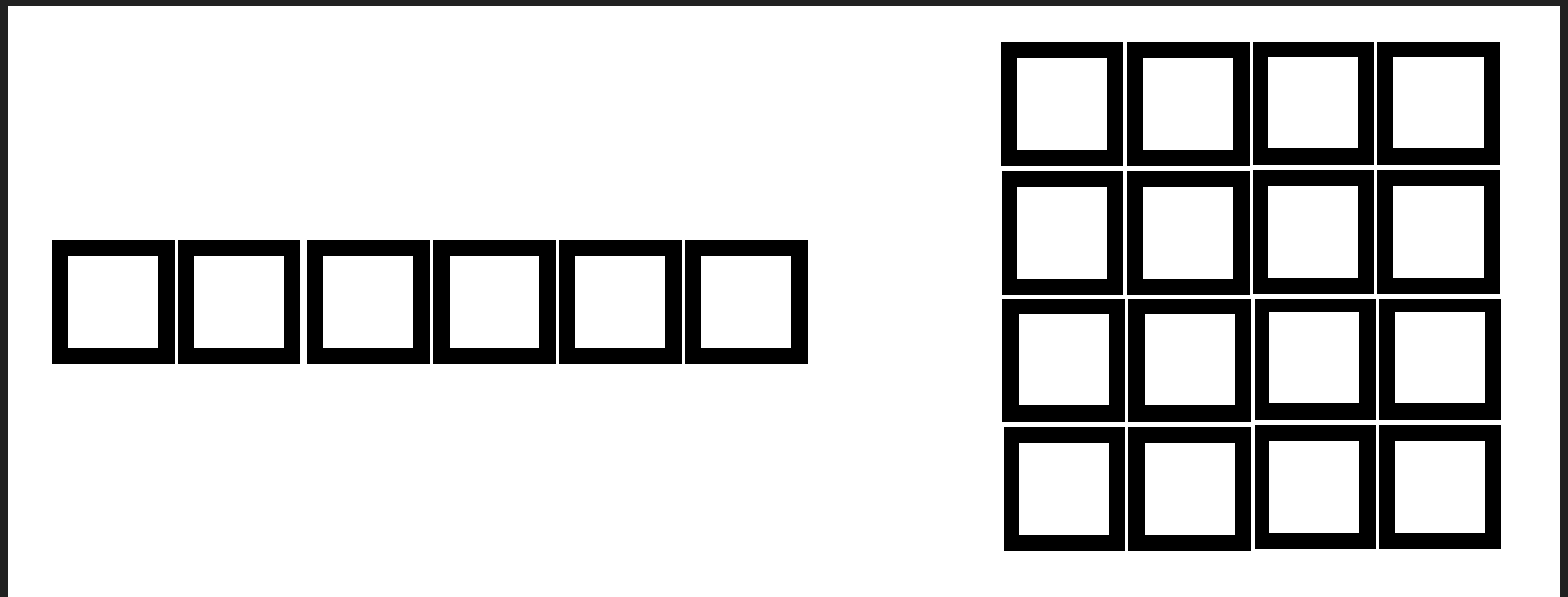
GENERAL INTUITION

- Transforms raw input, layer-by-layer, to final class scores
- More complex features are learnt as you go deeper into the network by combining simpler ones
- Higher level abstracted representations are combinations of several lower level features



INTUITION AND INSPIRATION

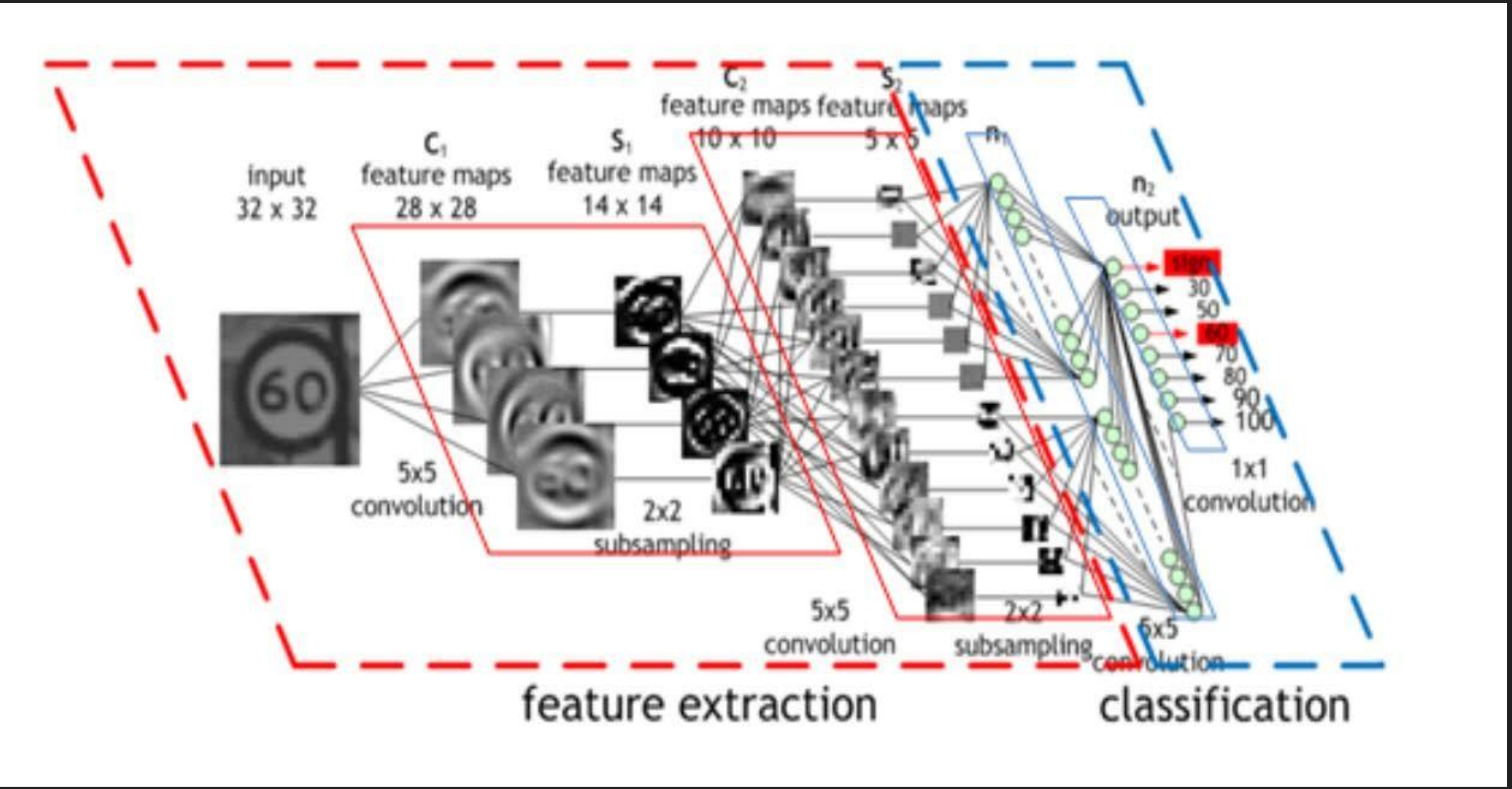
Specialises in processing grid-like data



Uses sparse interactions and parameter sharing to achieve equivariance

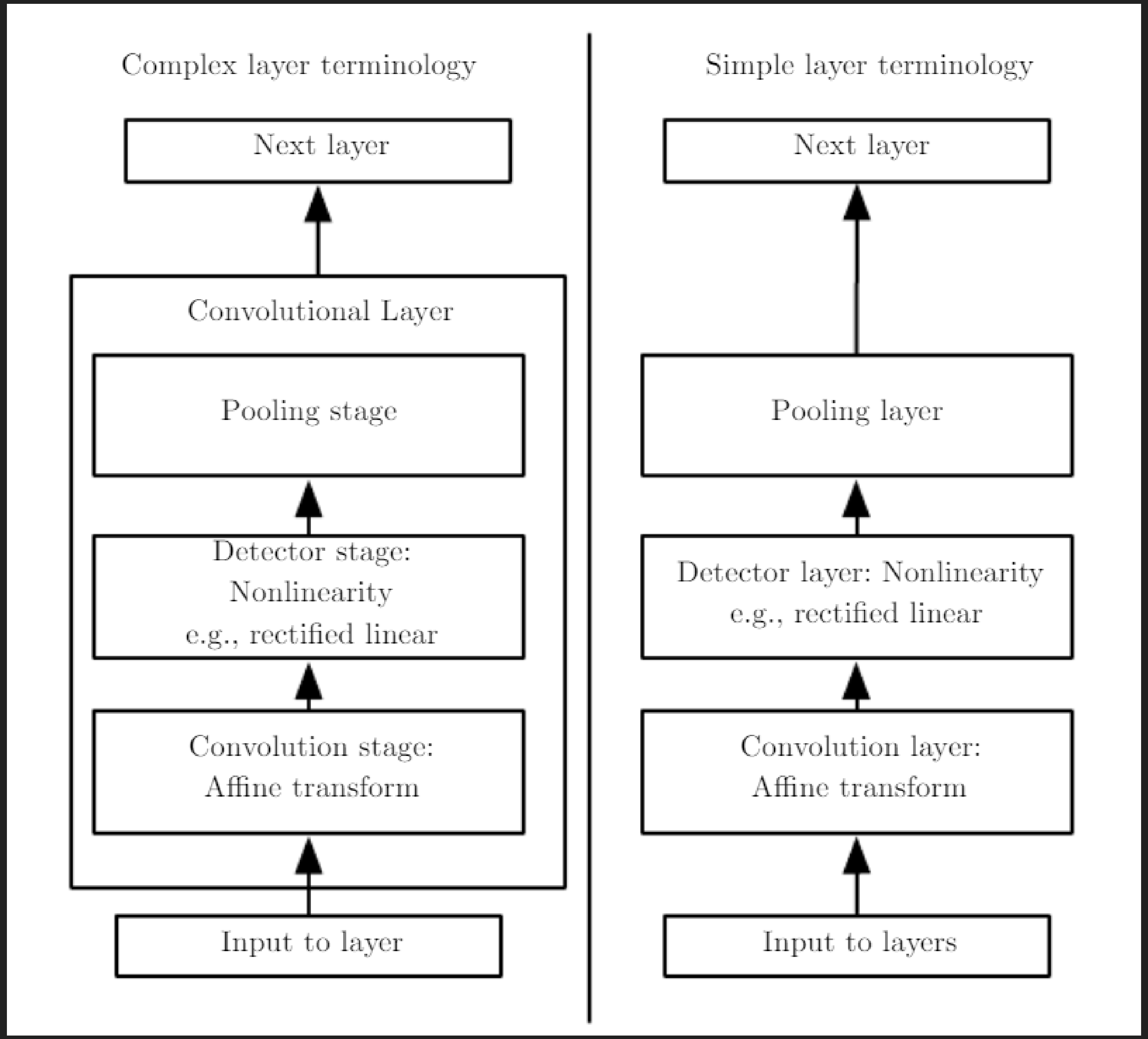
Inspired by visual processing system of the brain

CNNS AS FEATURE EXTRACTORS FOR DNNS



LAYERS AND STRUCTURES

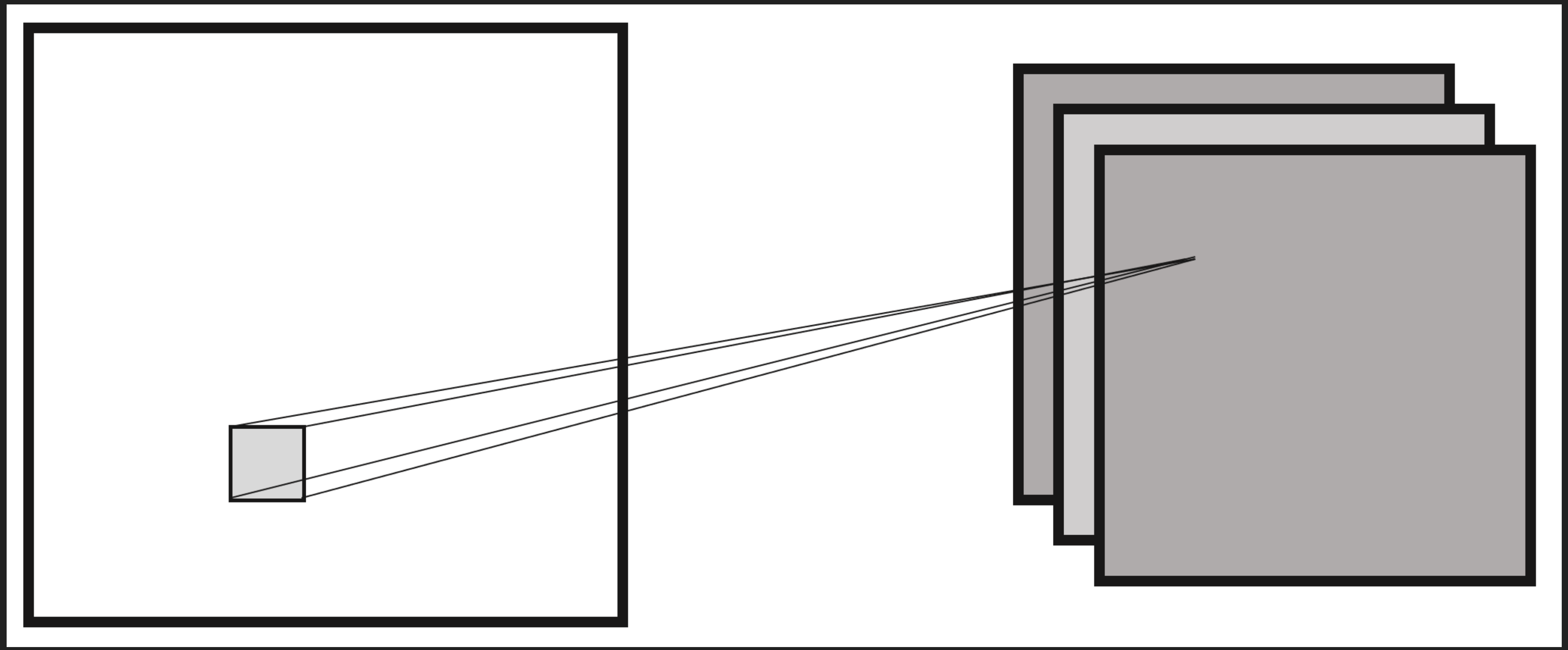
- Convolutional layer
- Pooling / subsampling layer



CONVOLUTIONAL LAYERS

OVERALL PREMISE

- Core building block and namesake of a CNN
- Number of customisable parameters in this type of layer that control a set of learnable filters
- Each filter has a set of units (neurons), which each apply the filter kernel to a specific area of the input data (its local receptive field)
- Filter are convolved with the input data to produce a feature map
- Each filter learns to activate when it sees a specific feature



CONVOLUTIONS

An operation which describes the mixing of two functions or pieces of information.

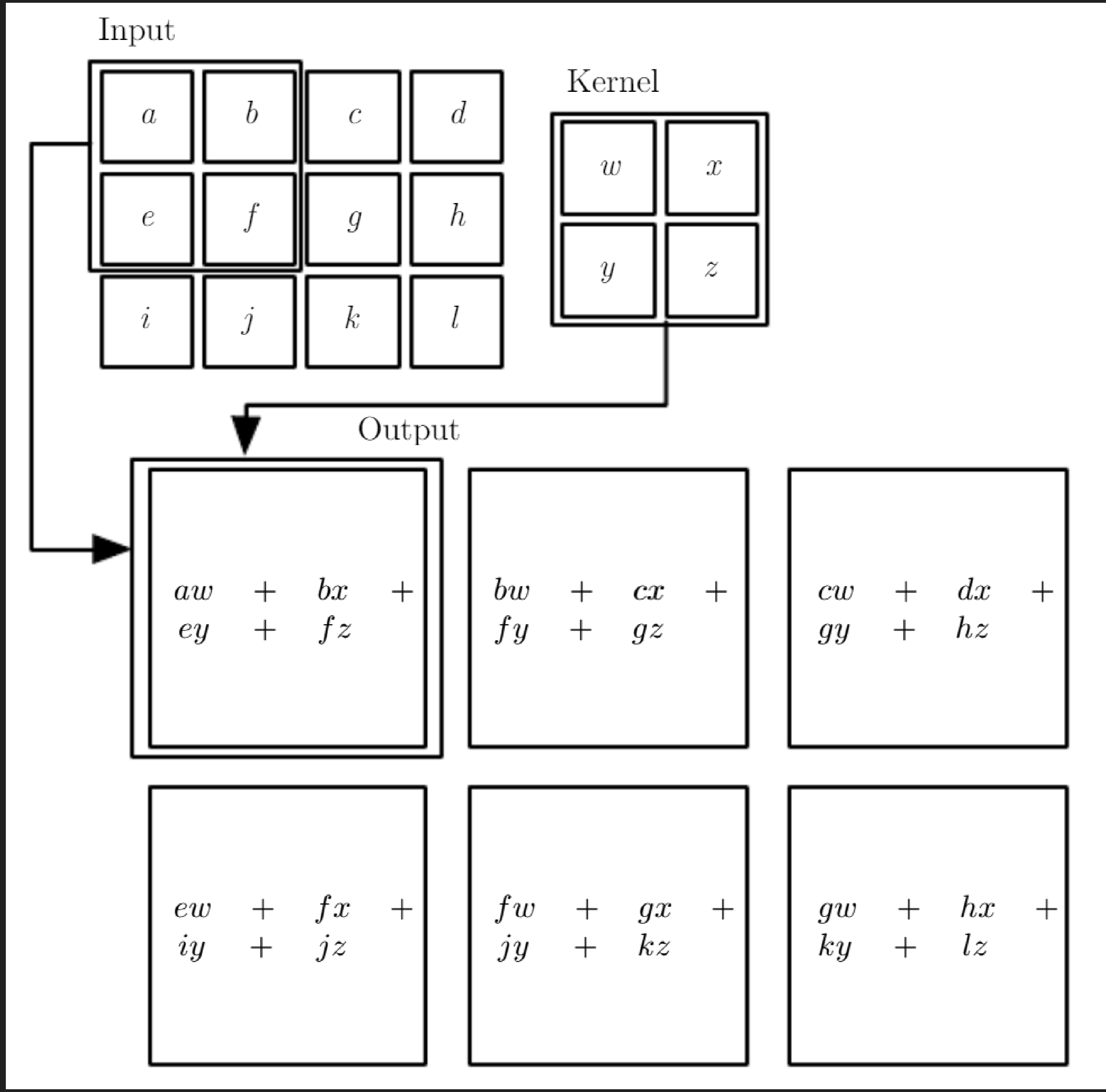
$$h(x) = f \otimes g = \int_{-\infty}^{\infty} f(x-u)g(u)\,du = \mathcal{F}^{-1}\left(\sqrt{2\pi}\mathcal{F}[f]\mathcal{F}[g]\right)$$

$$\text{feature map} = \text{input} \otimes \text{kernel} = \sum_{y=0}^{\text{columns}} \left(\sum_{x=0}^{\text{rows}} \text{input}(x-a, y-b)\,\text{kernel}(x,y) \right) = \mathcal{F}^{-1}\left(\sqrt{2\pi}\mathcal{F}[\text{input}]\mathcal{F}[\text{kernel}]\right)$$

Most libraries implement cross-correlation*

* other interpretations of convolutions can apply in other fields

COMPUTING FEATURE MAPS

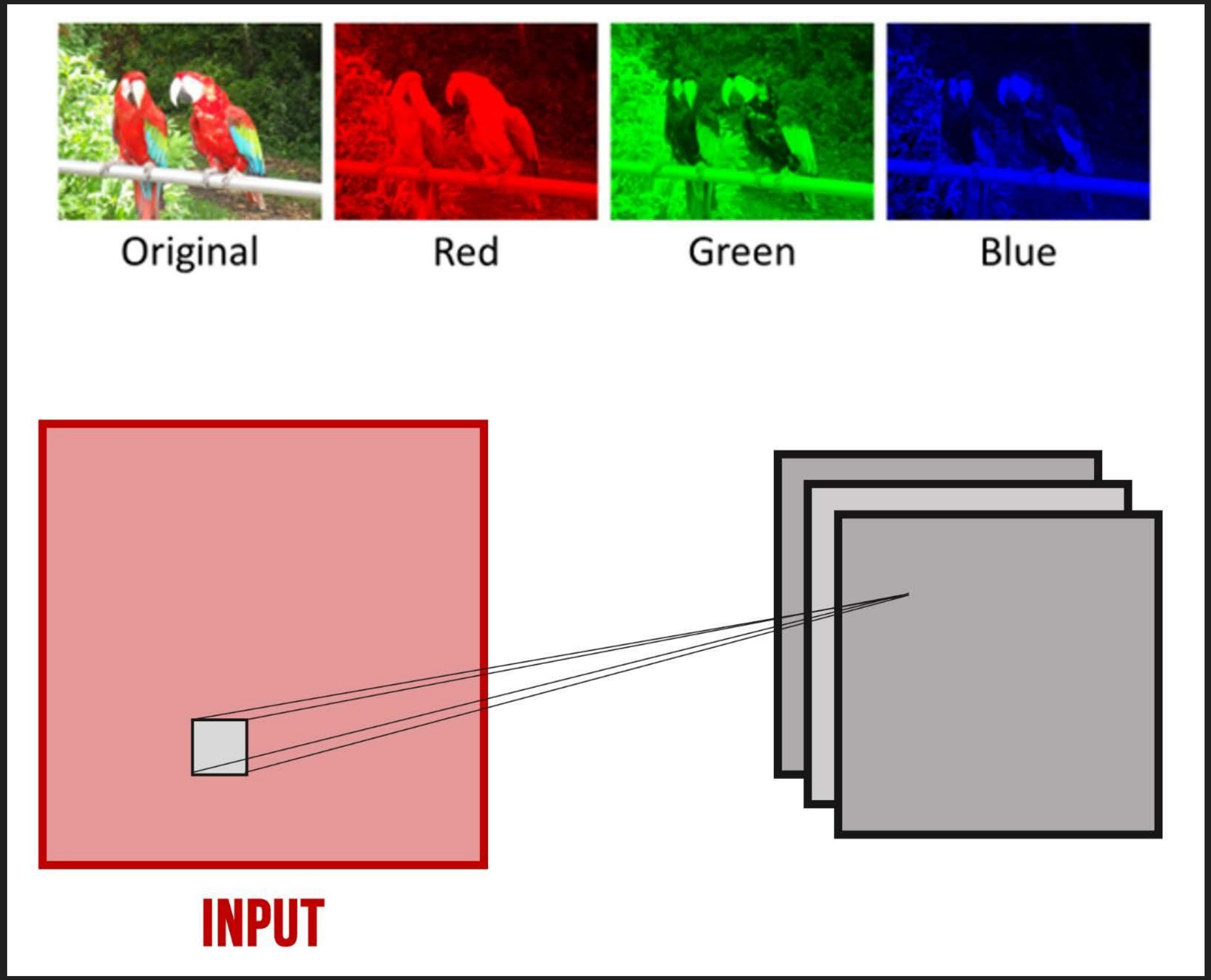


INPUT

Tensors

- 1D = sequences
- 2D = images
- 3D = video

Inputs can be multi-channel



FILTERS

- Number of filters equal to the number of features you want the network to detect, you define how many you want
- Features are not defined; the CNN will learn them by learning the weights and biases of the filters

NEURONS

- Filters convolve with the input to produce a series of neurons in an activation map that each detect a specific feature in a specific region of the input data
- Each neuron behaves the same by applying the filter's kernel, but on a different part of the input

LOCAL RECEPTIVE FIELD

- The current part of the input that is being convolved with the filter

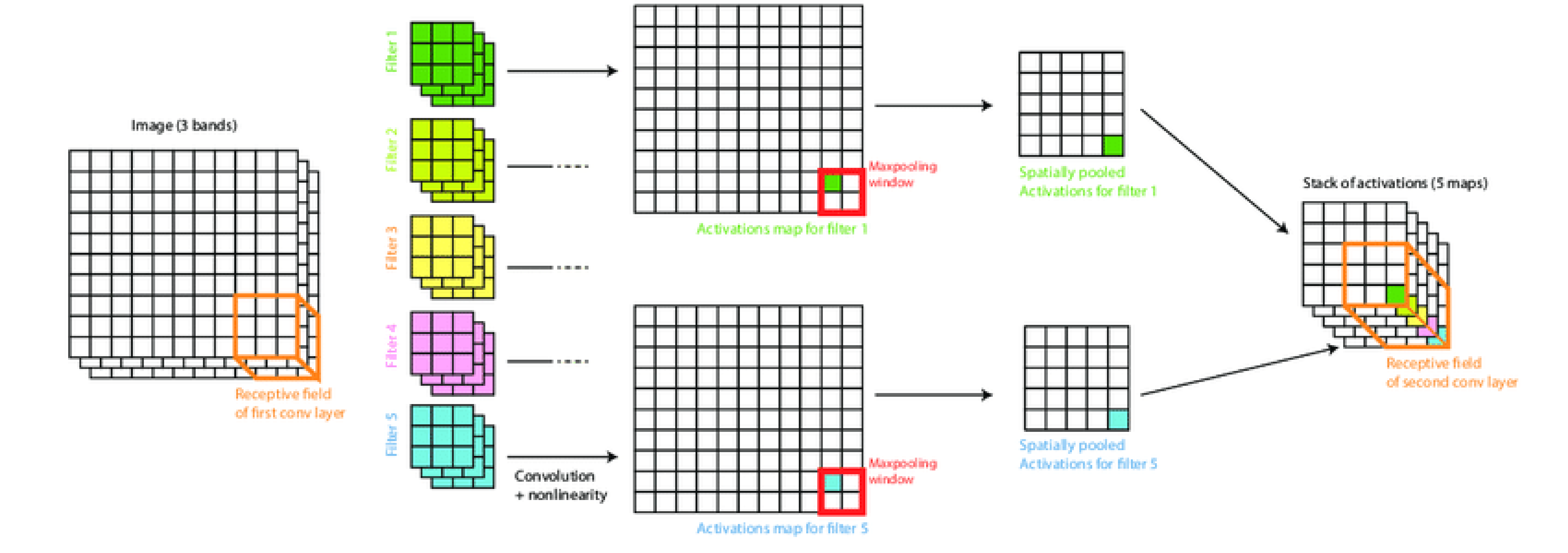
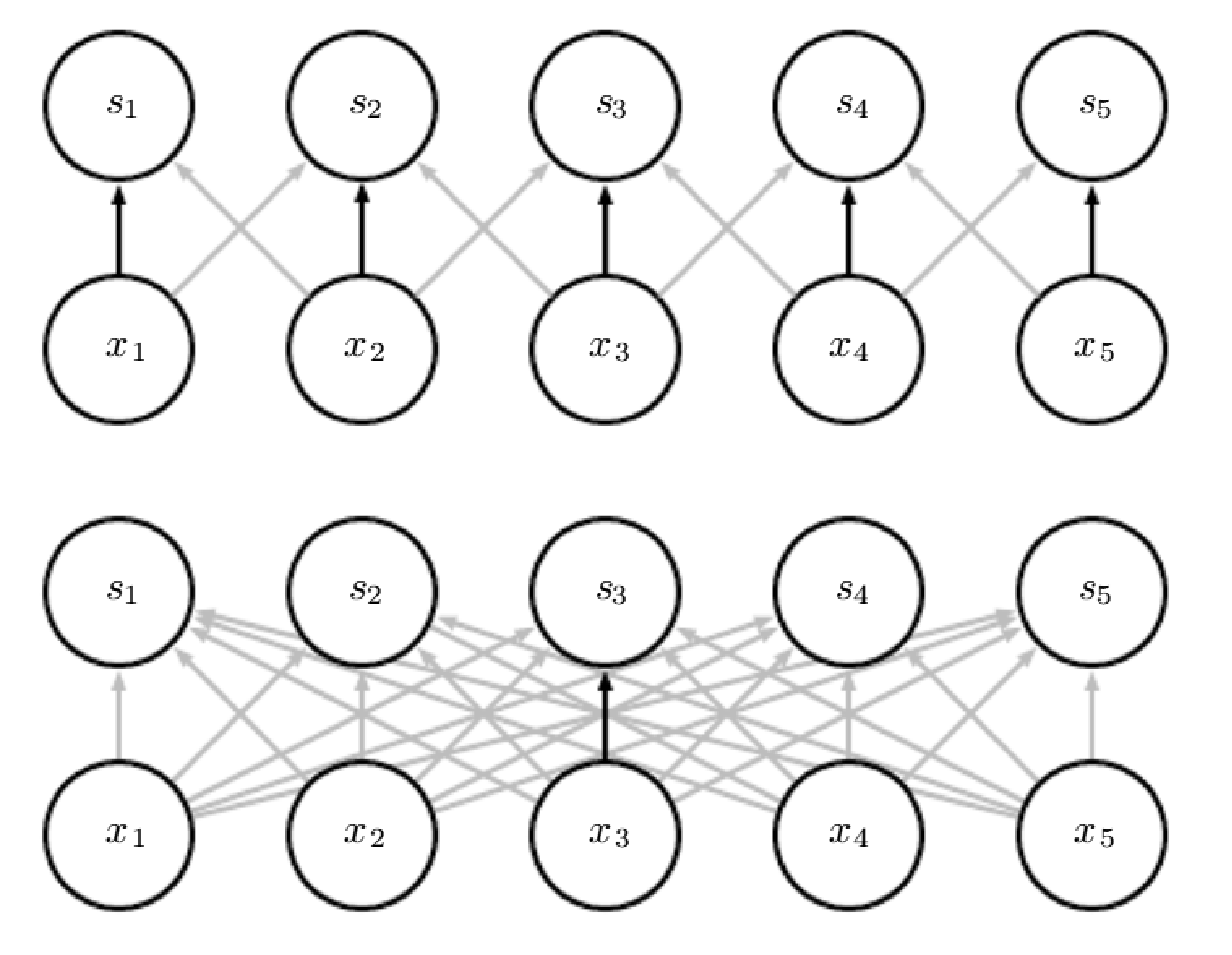


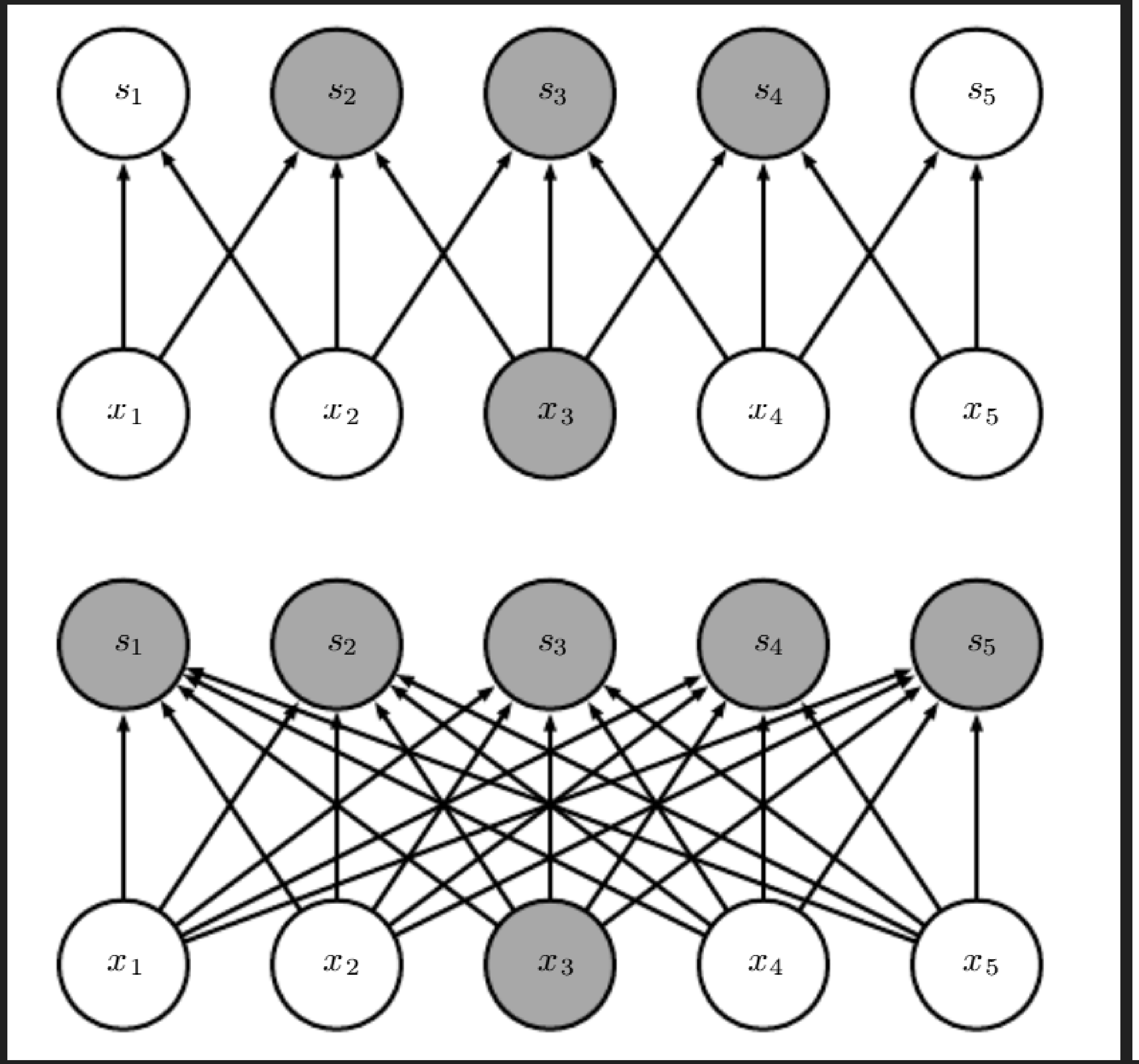
Image (3 bands)
Receptive field
of first conv layer
Filter 1
Filter 2
Filter 3
Filter 4
Filter 5
Convolution
+ nonlinearity
Maxpooling
window
Activations map for filter 1
Maxpooling
window
Activations map for filter 5
Spatially pooled
Activations for filter 1
Spatially pooled
Activations for filter 5
Stack of activations (5 maps)
Receptive field
of second conv layer

PARAMETER SHARING

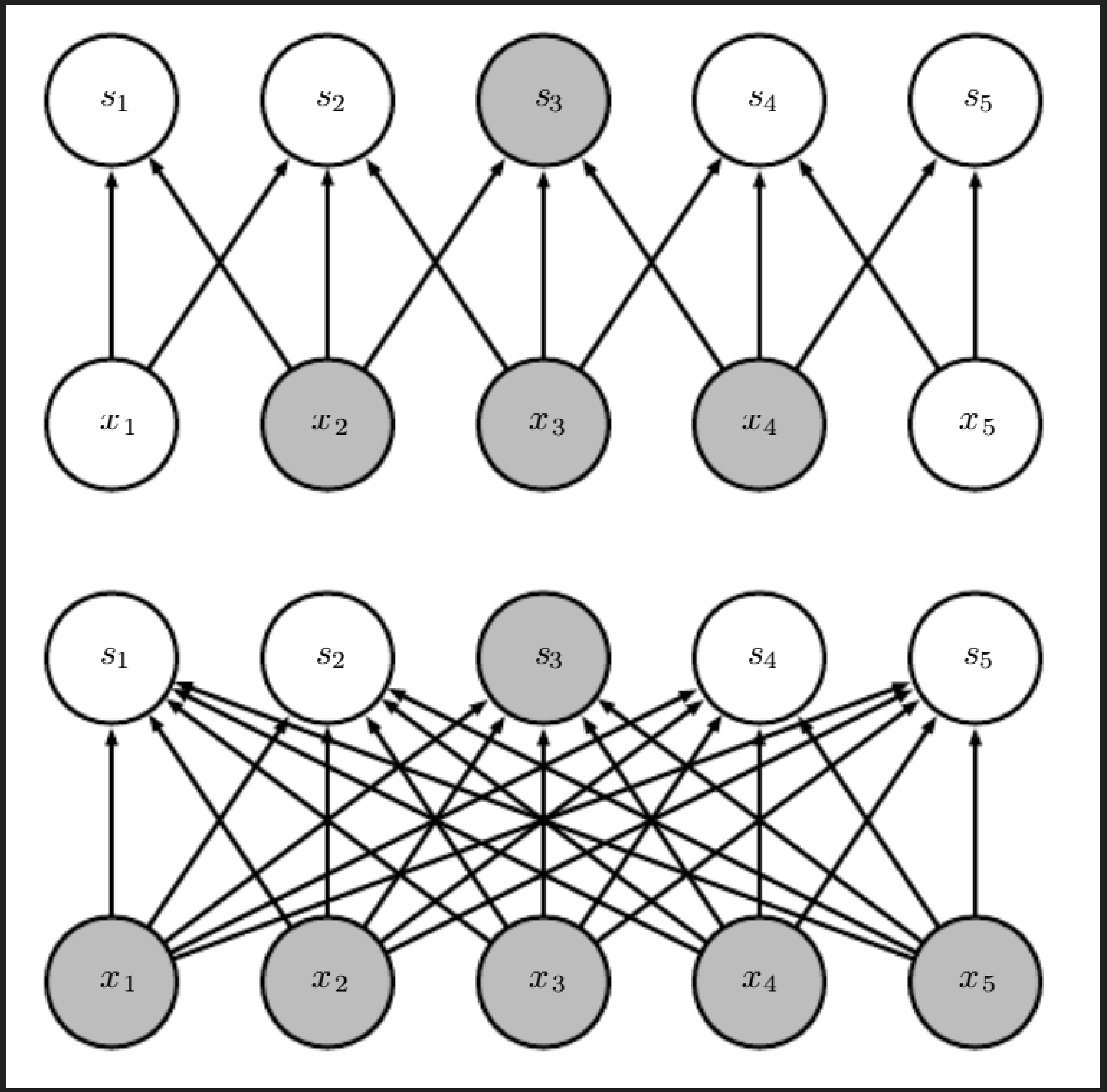
- Every neuron in a feature map has a unique local receptive field
- But, the weights and biases (filter) applied to these fields are shared
- This means that every unit in the filter detects exactly the same feature, just at different locations in the input



SPARSITY

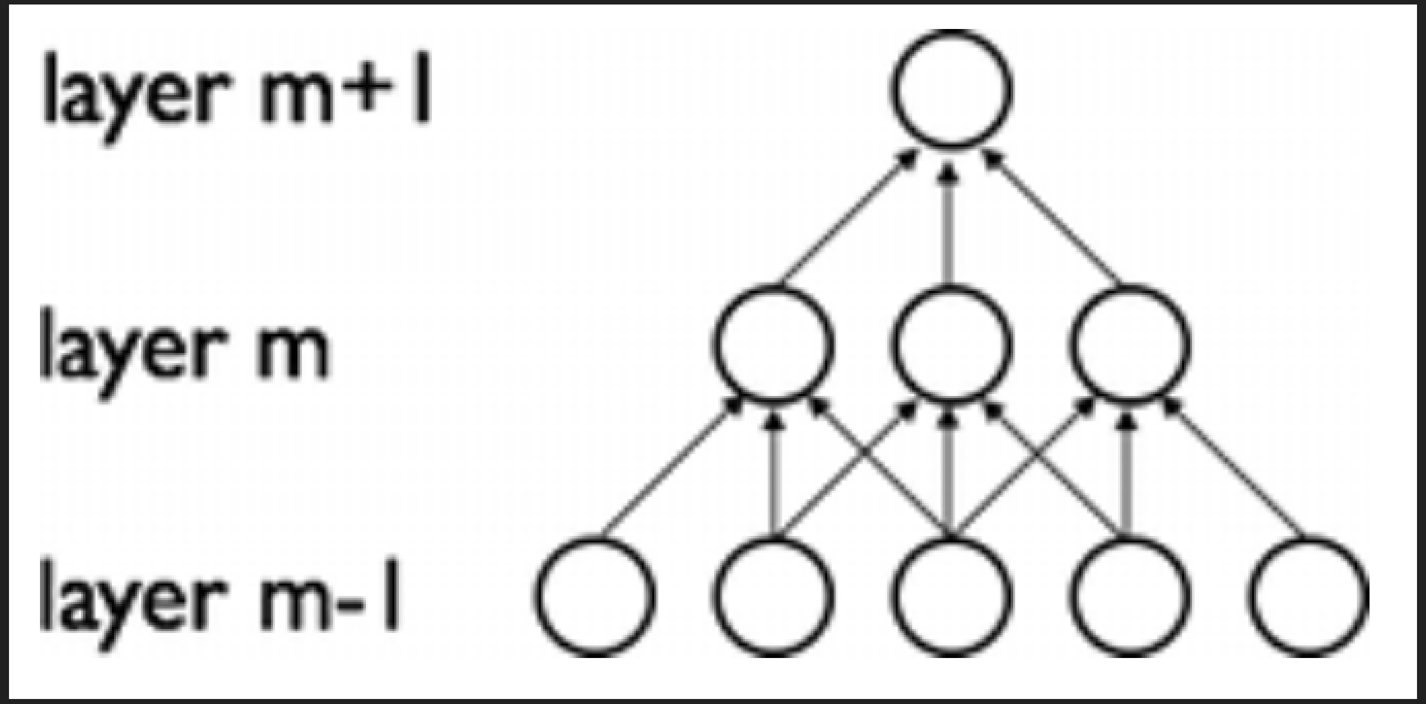


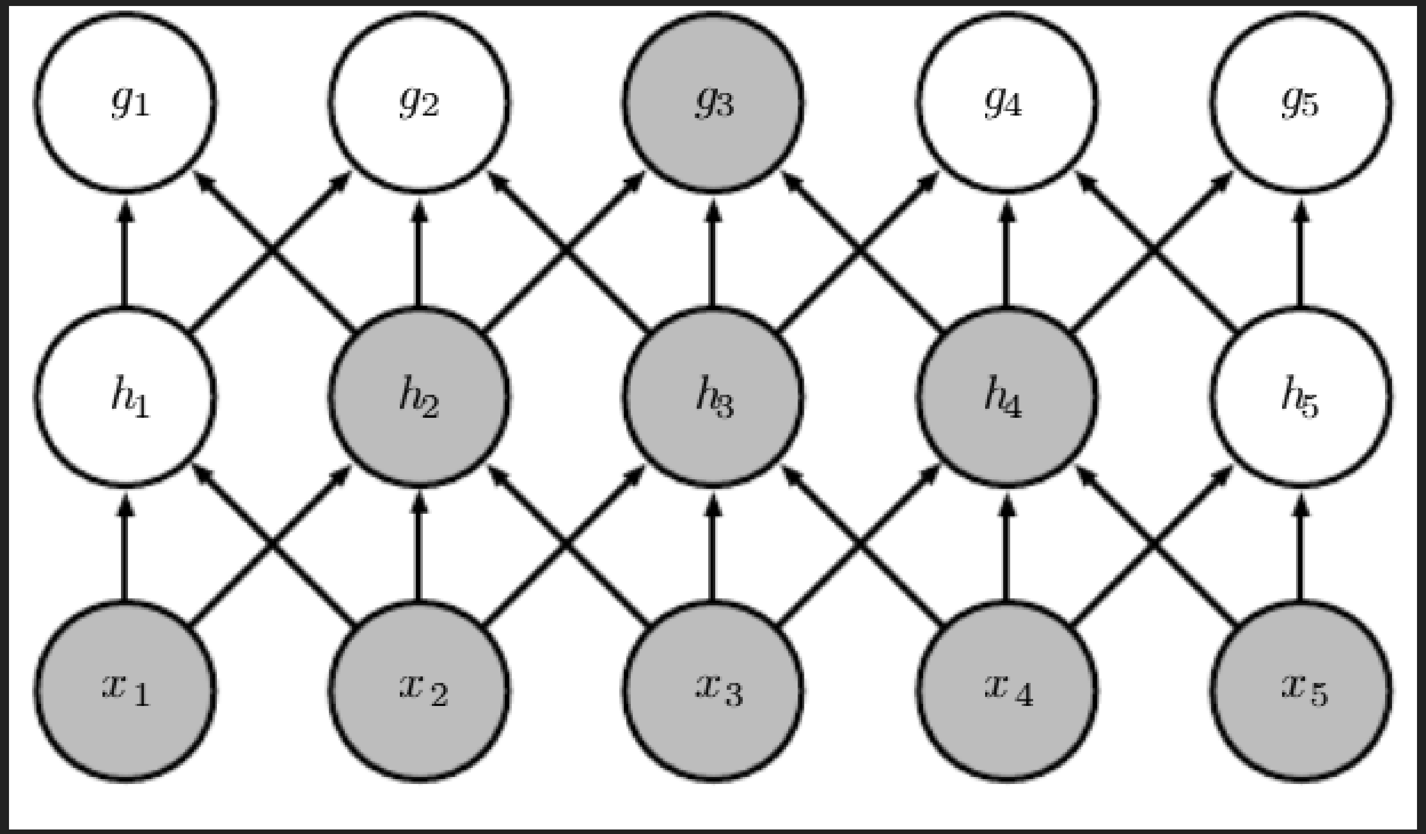
s_1 s_2 s_3 s_4 s_5
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
x_1 x_2 x_3 x_4 x_5



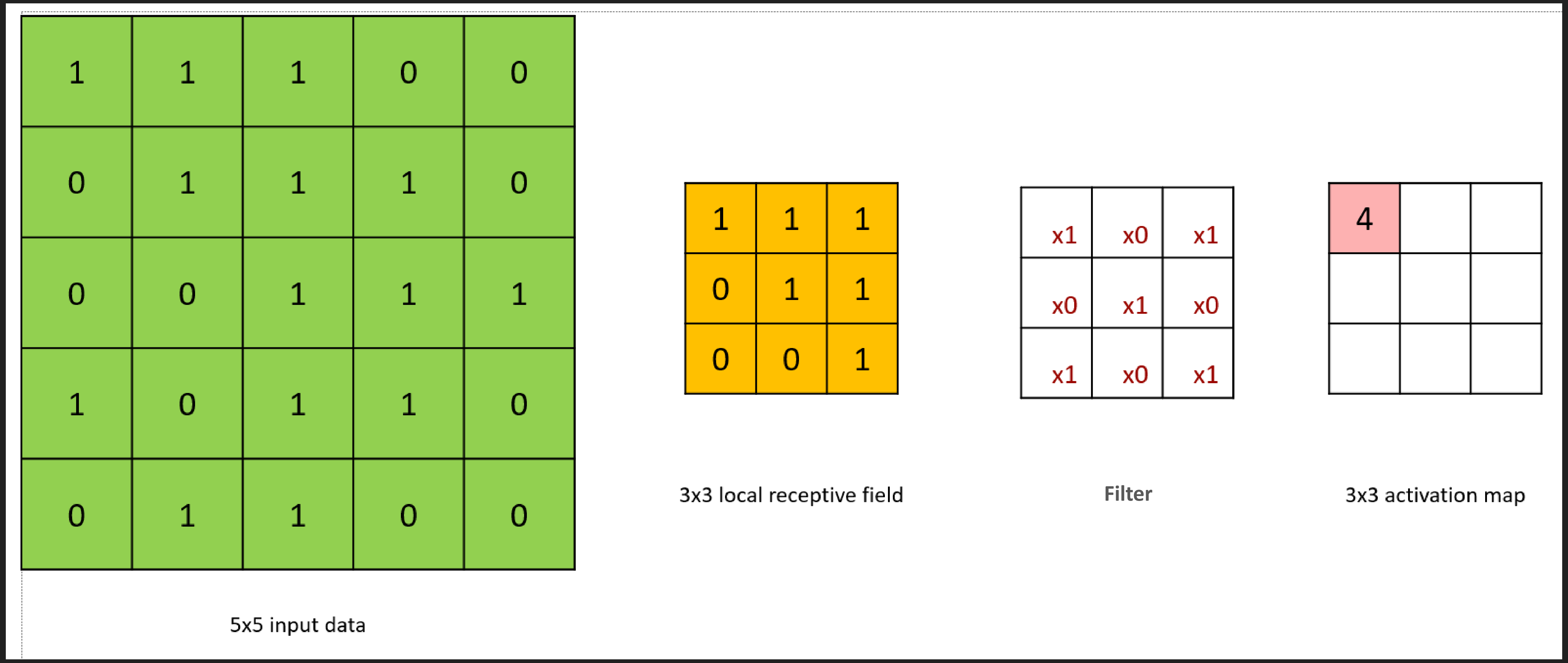
s_1 s_2 s_3 s_4 s_5
x_1 x_2 x_3 x_4 x_5
s_1 s_2 s_3 s_4 s_5
x_1 x_2 x_3 x_4 x_5

HIERARCHY

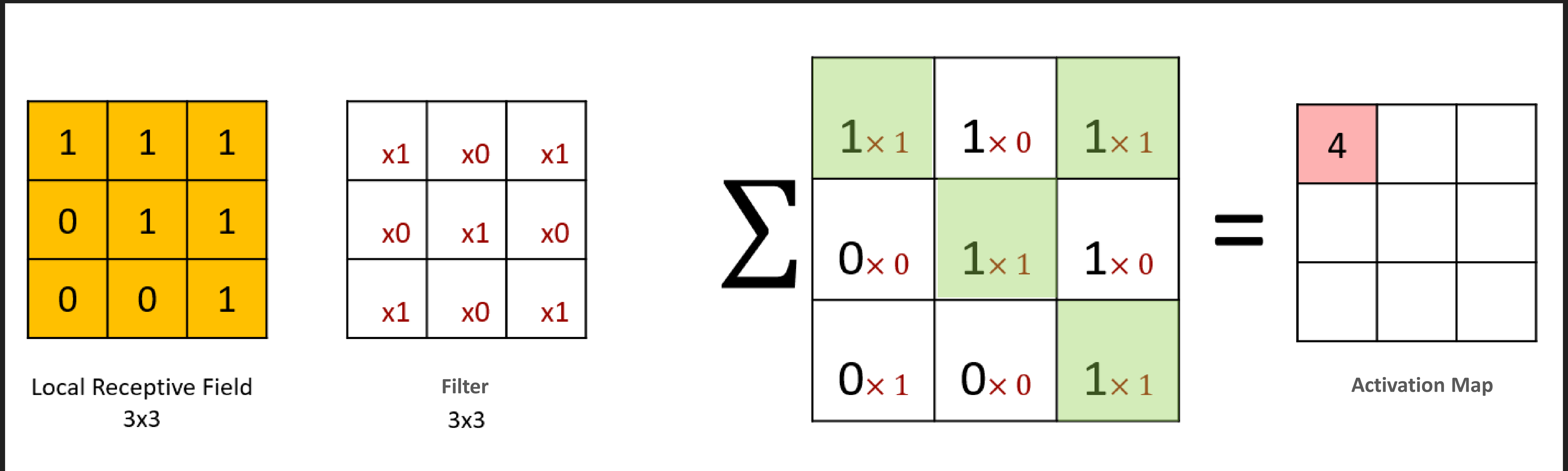

layer m+1
layer m
layer m-1


g_1
g_2
g_3
g_4
g_5
h_1
h_2
h_3
h_4
h_5
x_1
x_2
x_3
x_4
x_5

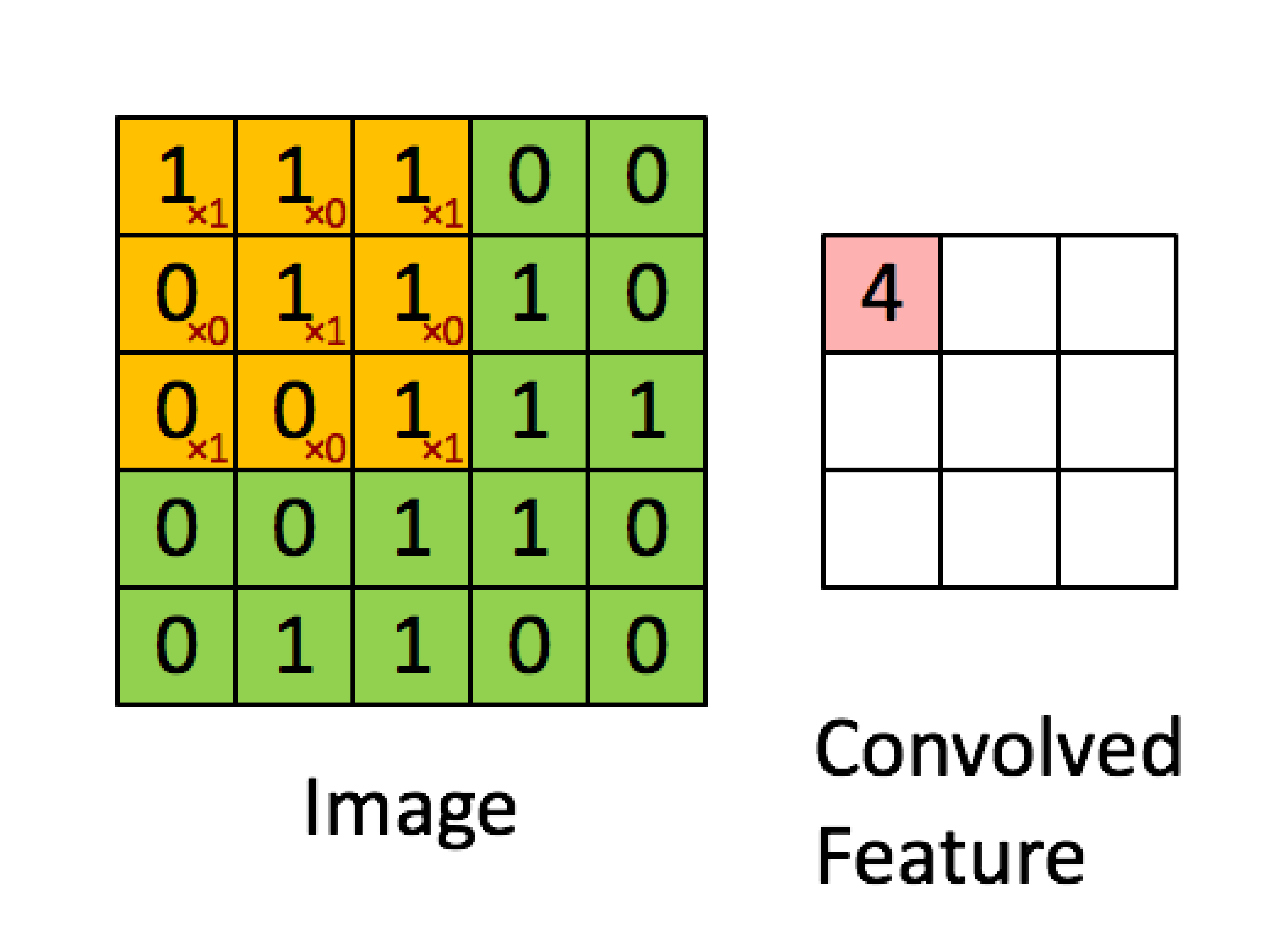
CONVOLUTIONS



COMPUTING FEATURE MAPS

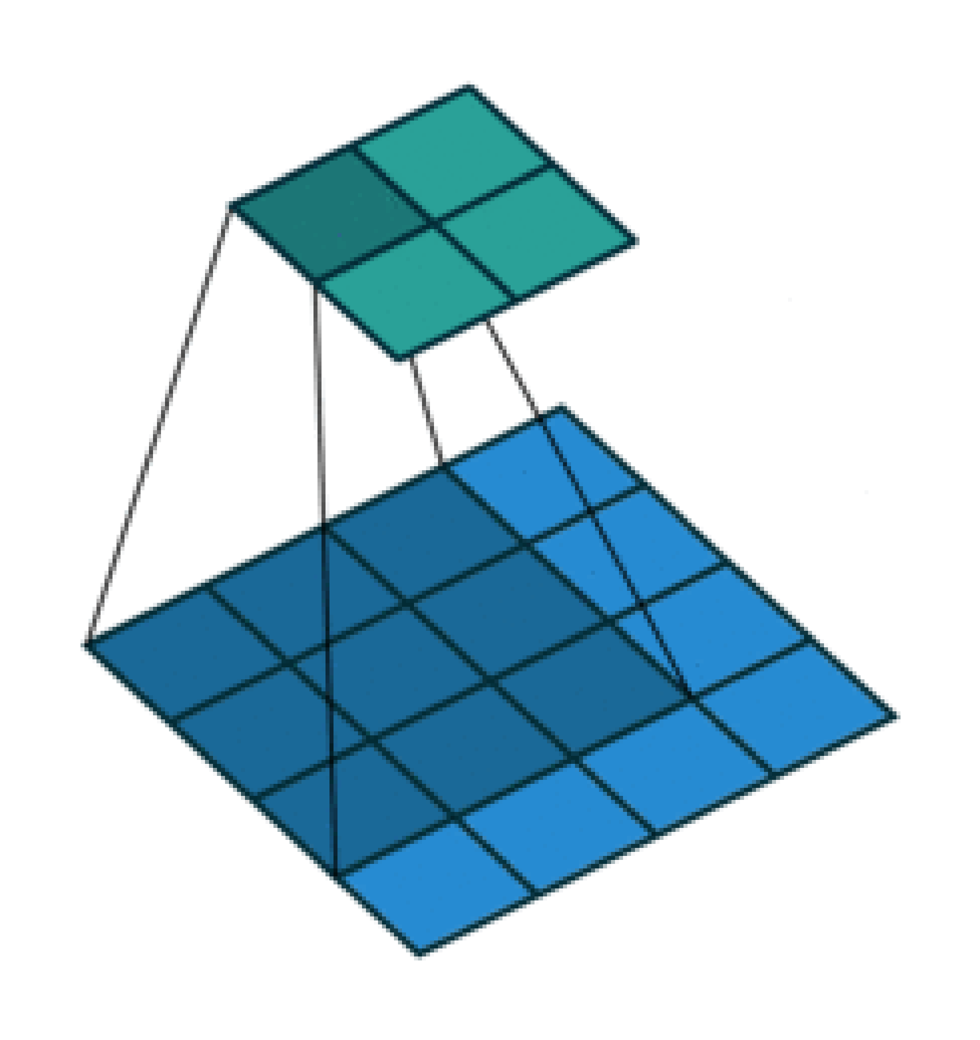


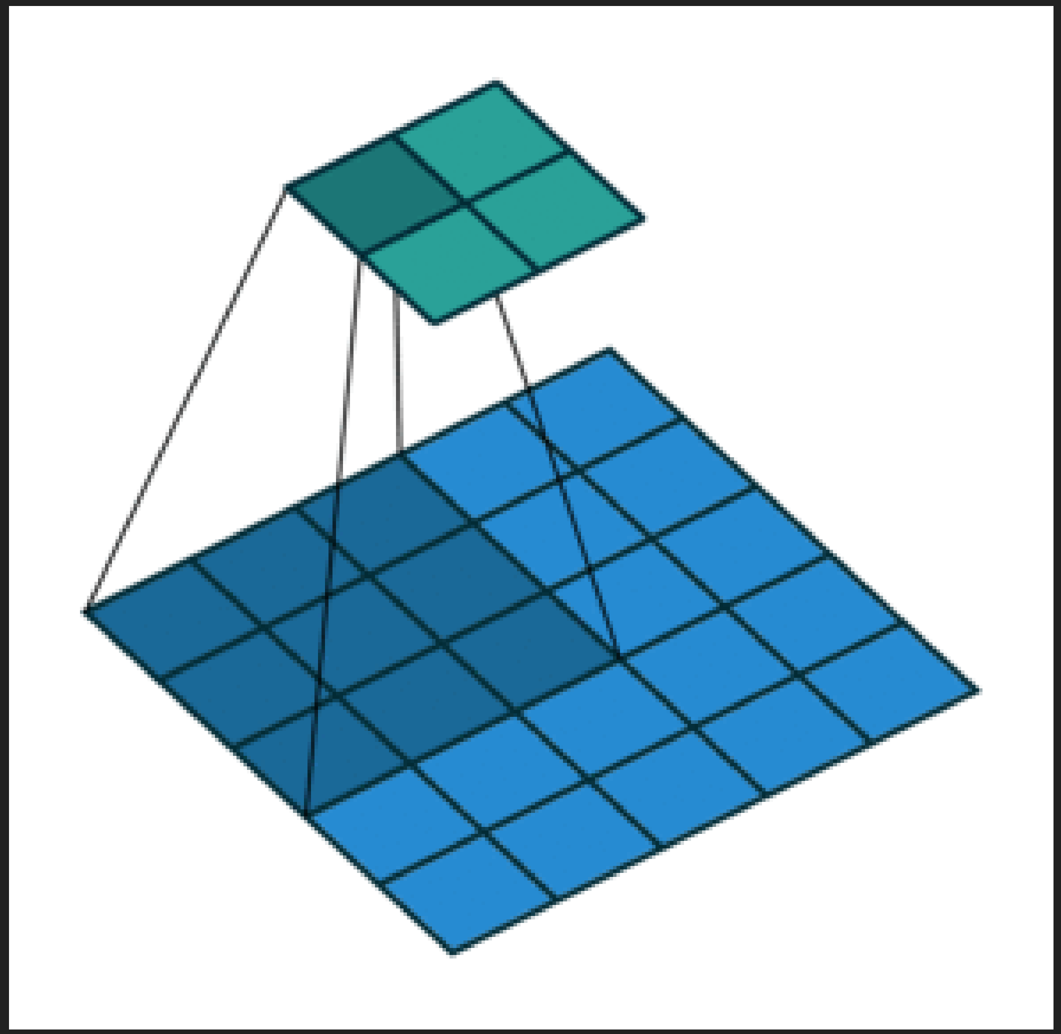
CONVOLUTIONS



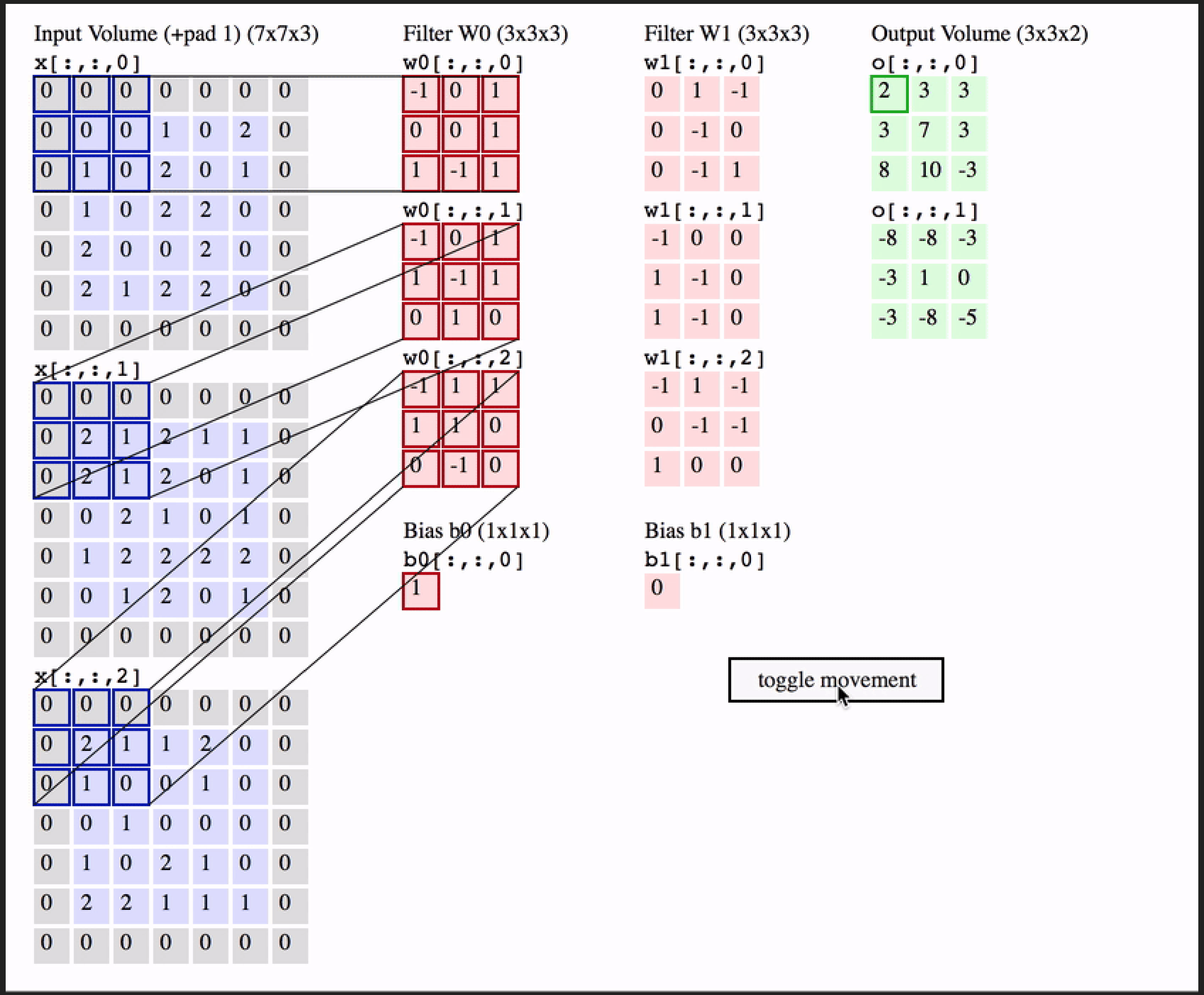
STRIDES

- Size of receptive field matters; too small and it may not pick up a feature, too big and the network becomes
fully-connected
- Width and height customisable; depth usually equal to input channels
- Stride length controls how far local receptive field slides; smaller number increases overlap between units in convolution layer



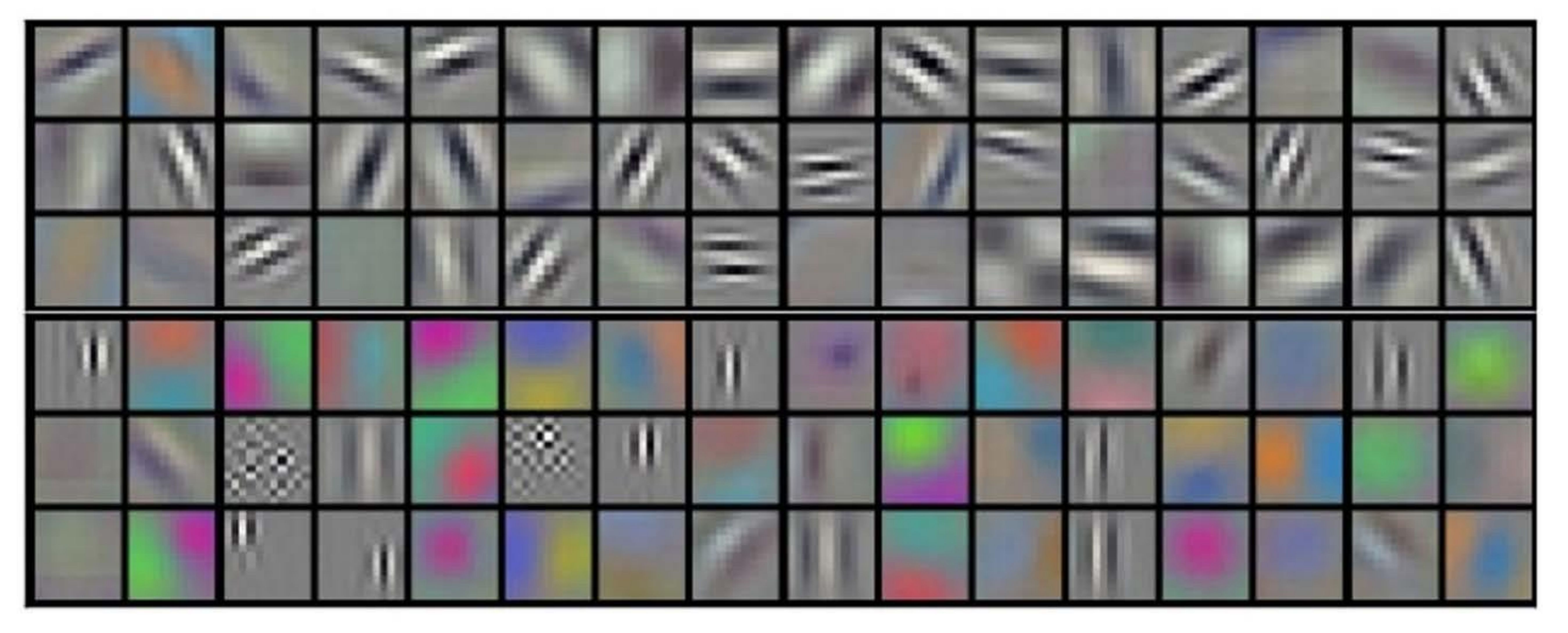


MULTI-CHANNEL CONVOLUTIONS

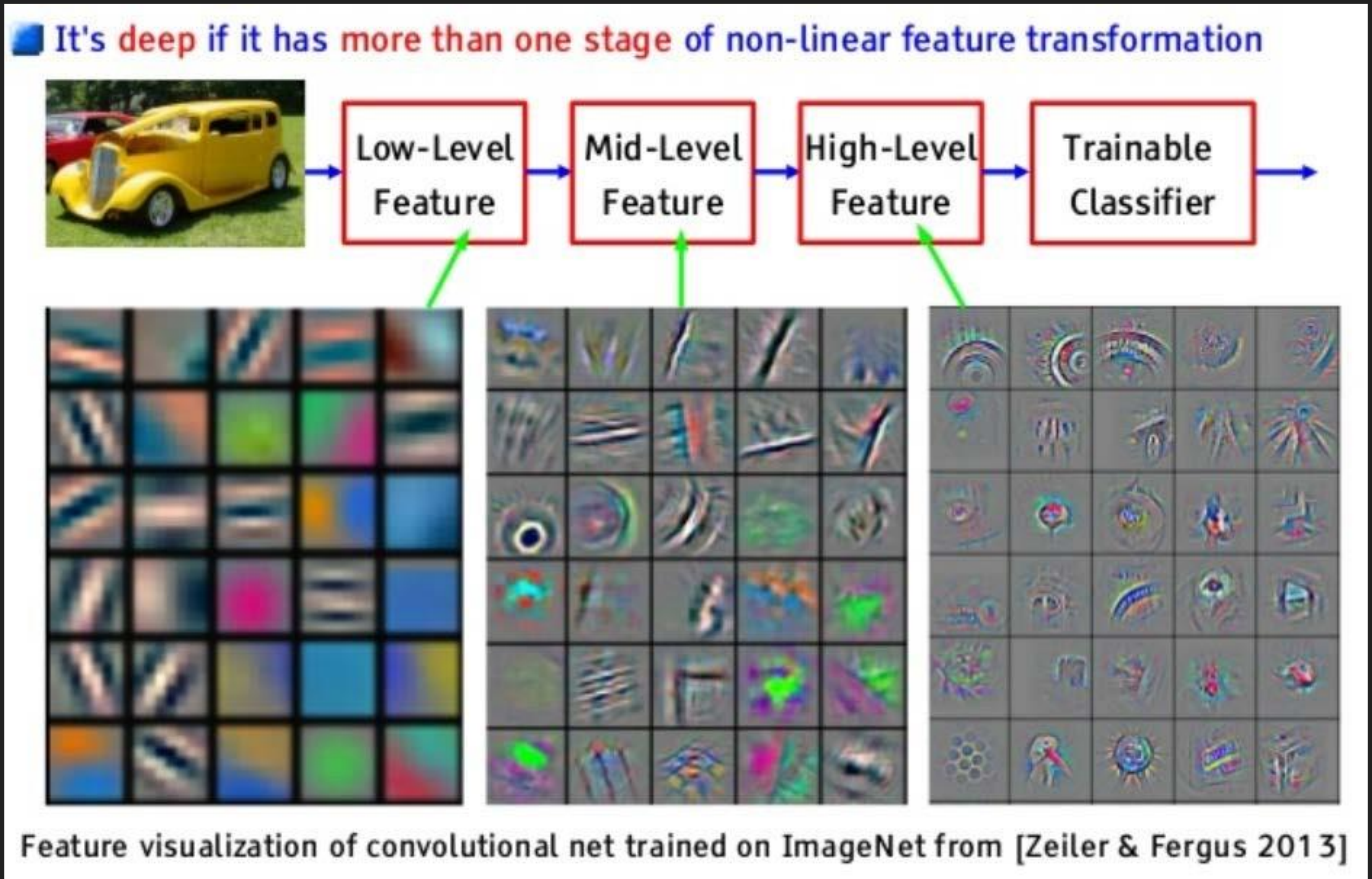


FEATURE MAPS

- Output of convolutions
- Number of feature maps produced equals number of features wanted to be detected (number of filters)
- Represents presence of feature at a given neuron
- Serves as input to the next layer (typically a pooling layer)



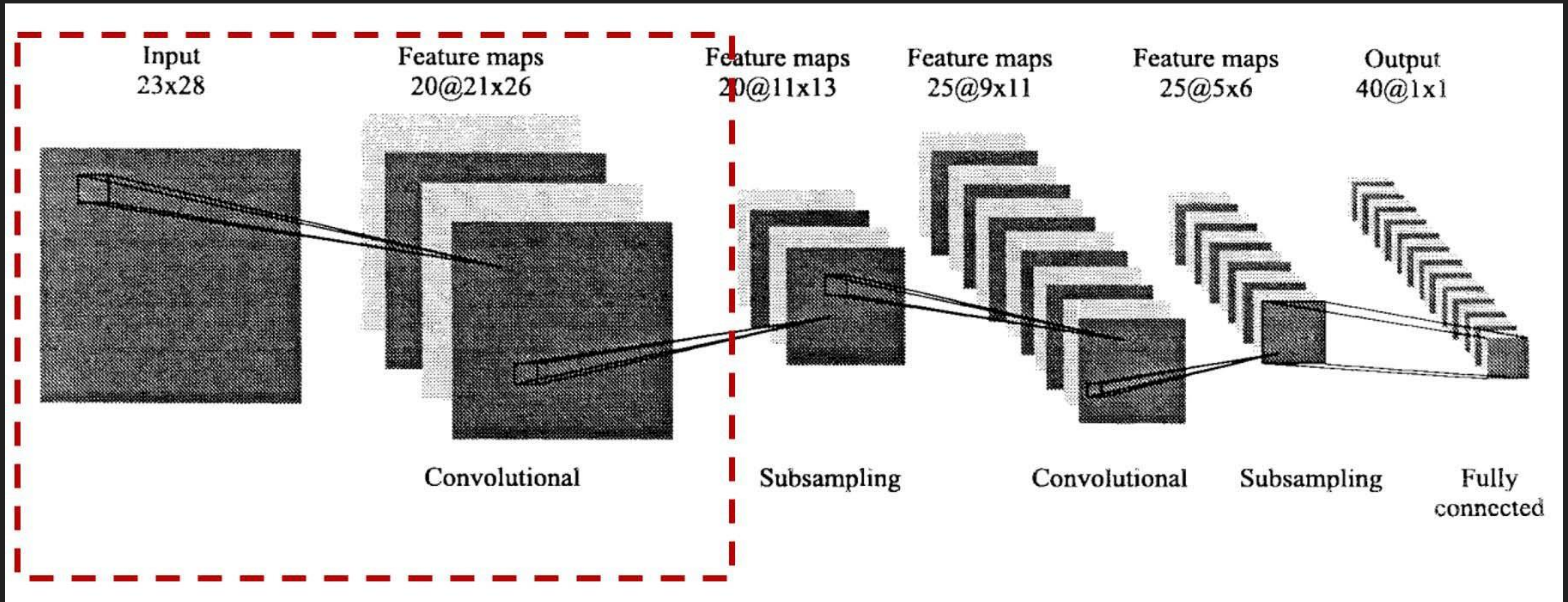
HIERARCHICAL FEATURE LEARNING



EQUIVARIANCE

- Translation equivariance
- Feature detected in one part of the data can appear in any other part of the data
- Creates a feature detector from one small sample that can be re-used elsewhere in the data
- Take the learned features and ‘convolve’ them with the larger dataset, getting different feature activation value at each location

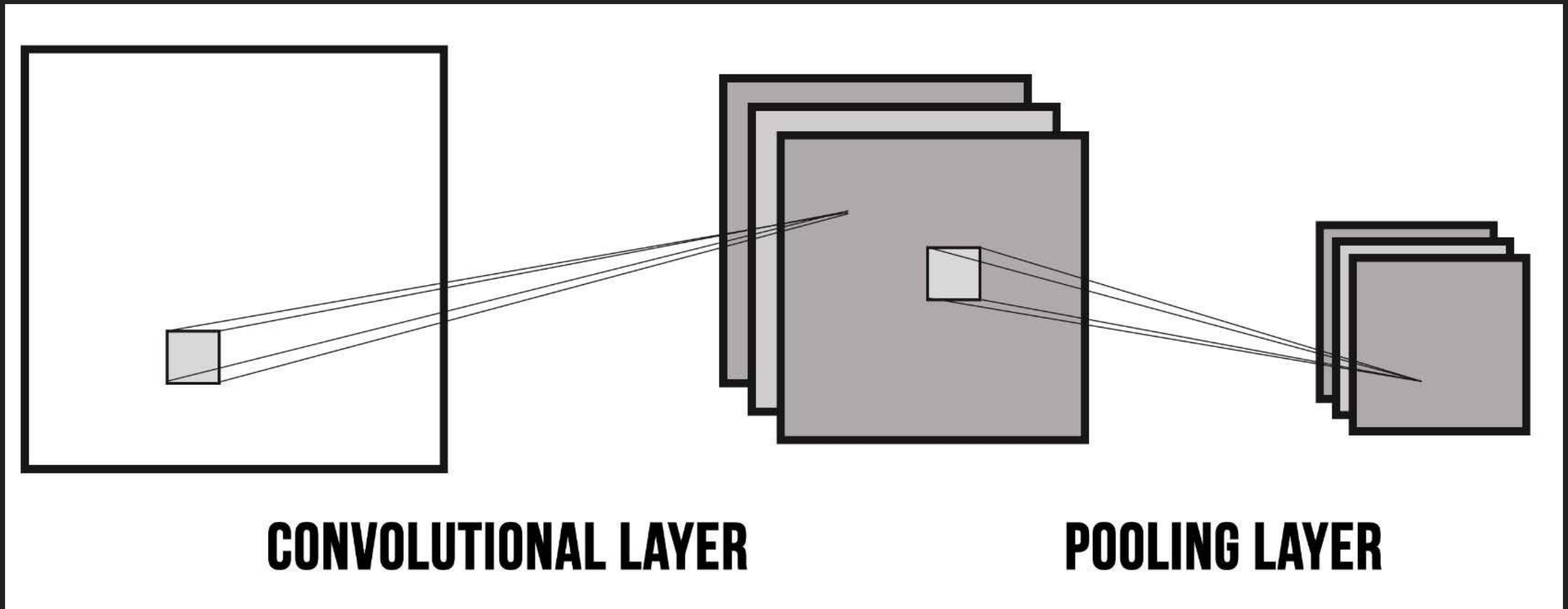
LAYER


Input
23x28
Feature maps
20@21x26
Feature maps
20@11x13
Feature maps
25@9x11
Feature maps
25@5x6
Output
40@1x1
Convolutional
Subsampling
Convolutional
Subsampling
Fully
connected

POOLING LAYERS

OVERALL PREMISE

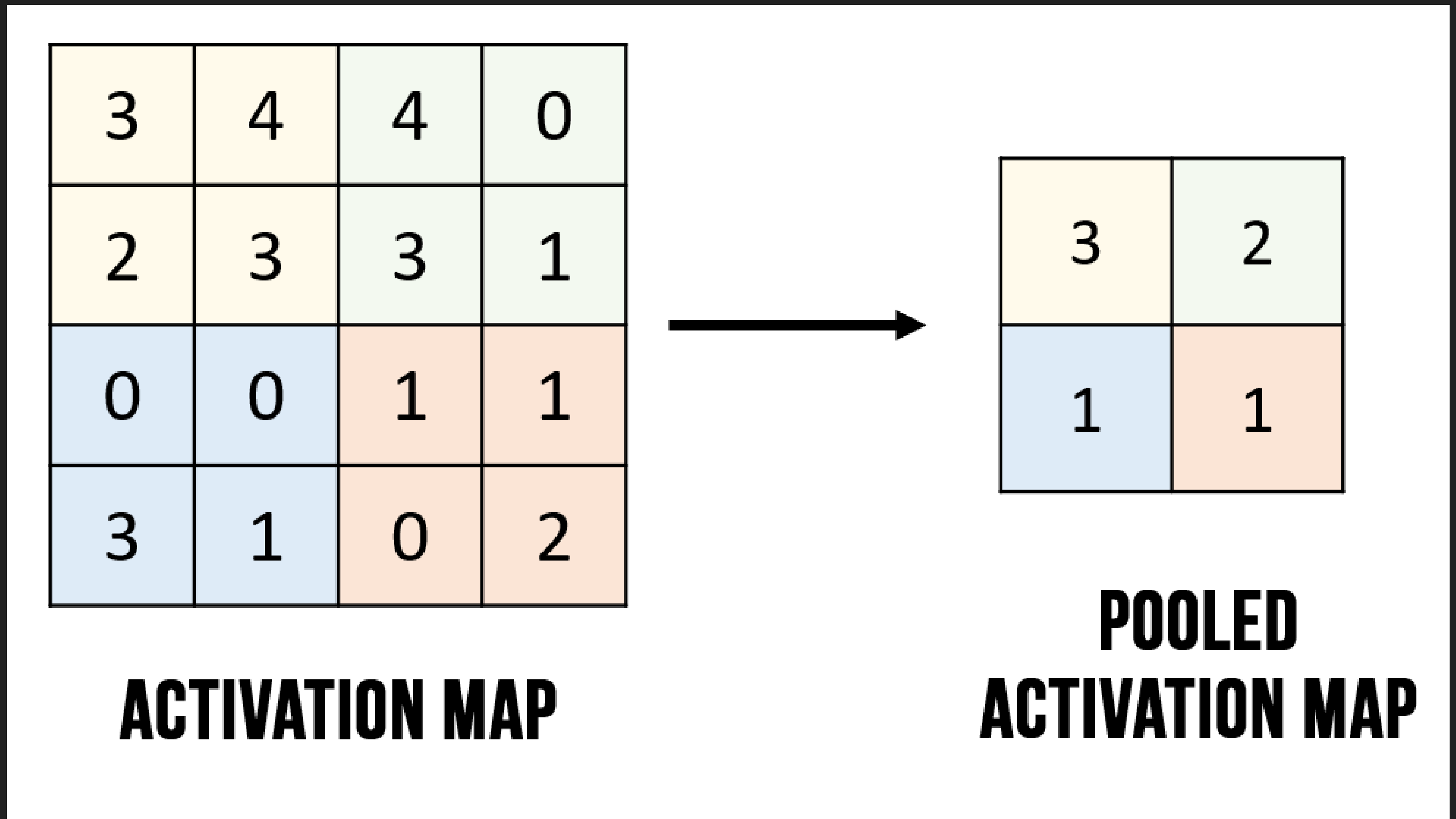
- Produces a summary statistic of the output from the previous convolution layer
- Helps learned representation to be invariant to small translations of input
- Can downsample feature map into a condensed version
- Assists in controlling overfitting; reduces network parameters, size and computational cost



POOLING

Configurable pooling windowand stride

- max-pooling
- L2-pooling
- average-pooling
- overlapping pooling



MAX POOLING

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

$$x_1 = \max\{3,4,2,3\}$$

$$x_2 = \max\{4,0,3,1\}$$

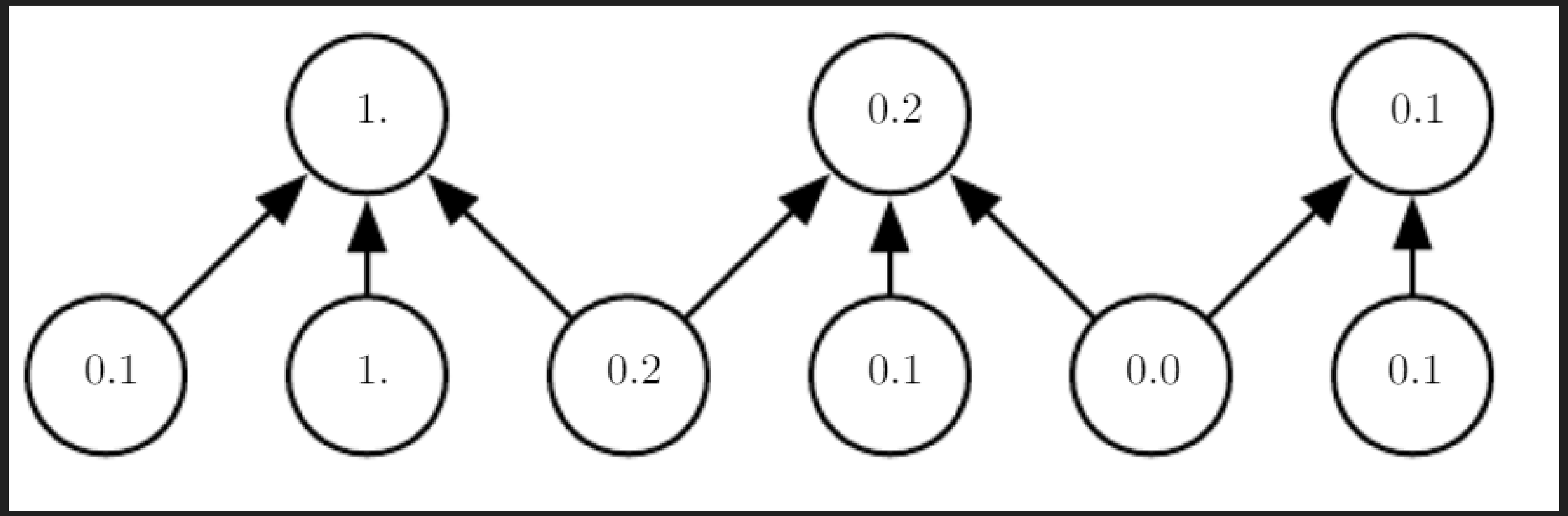
$$x_3 = \max\{0,0,3,1\}$$

$$x_4 = \max\{1,1,0,2\}$$

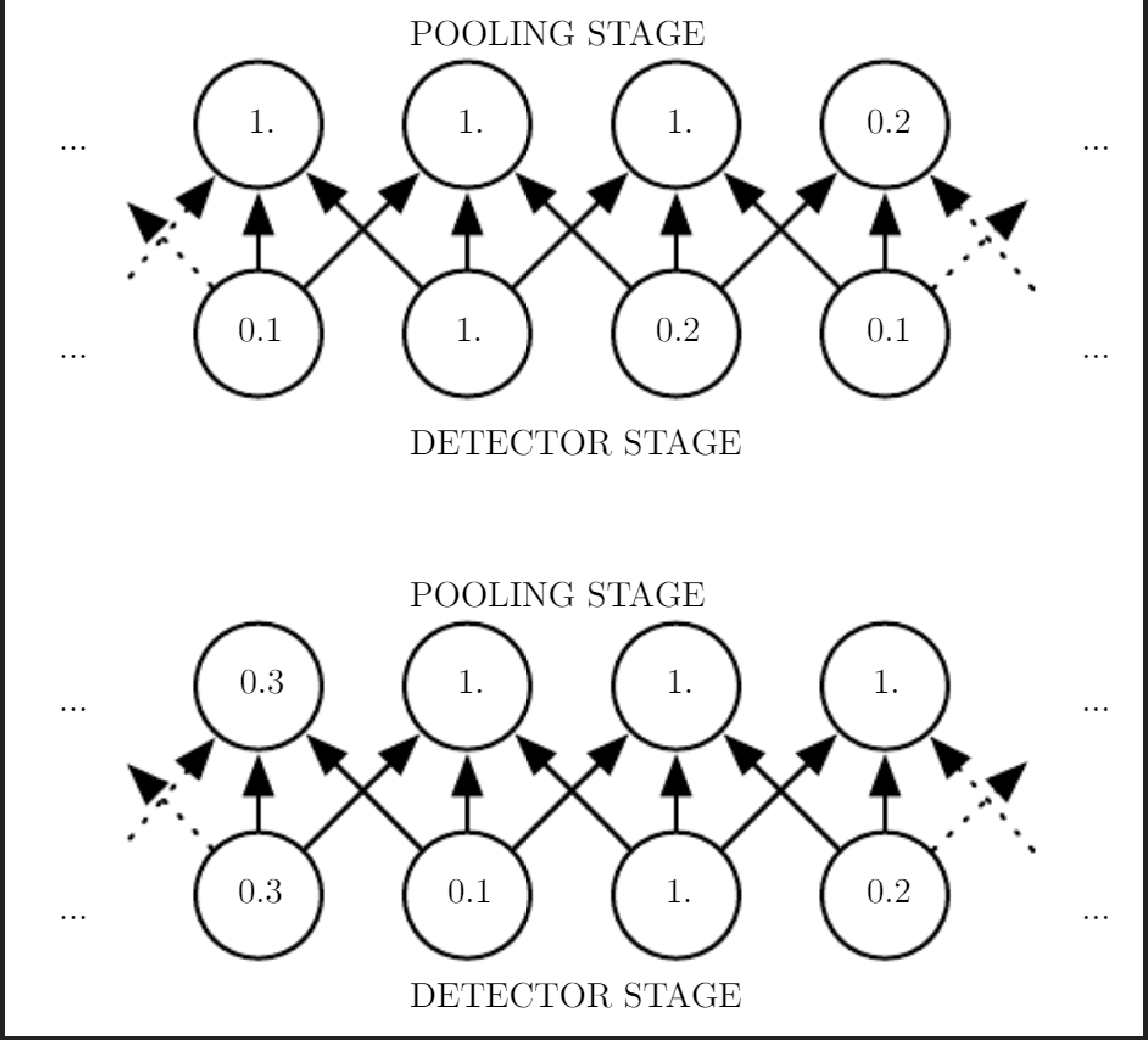
$\longrightarrow$

4	4
3	2

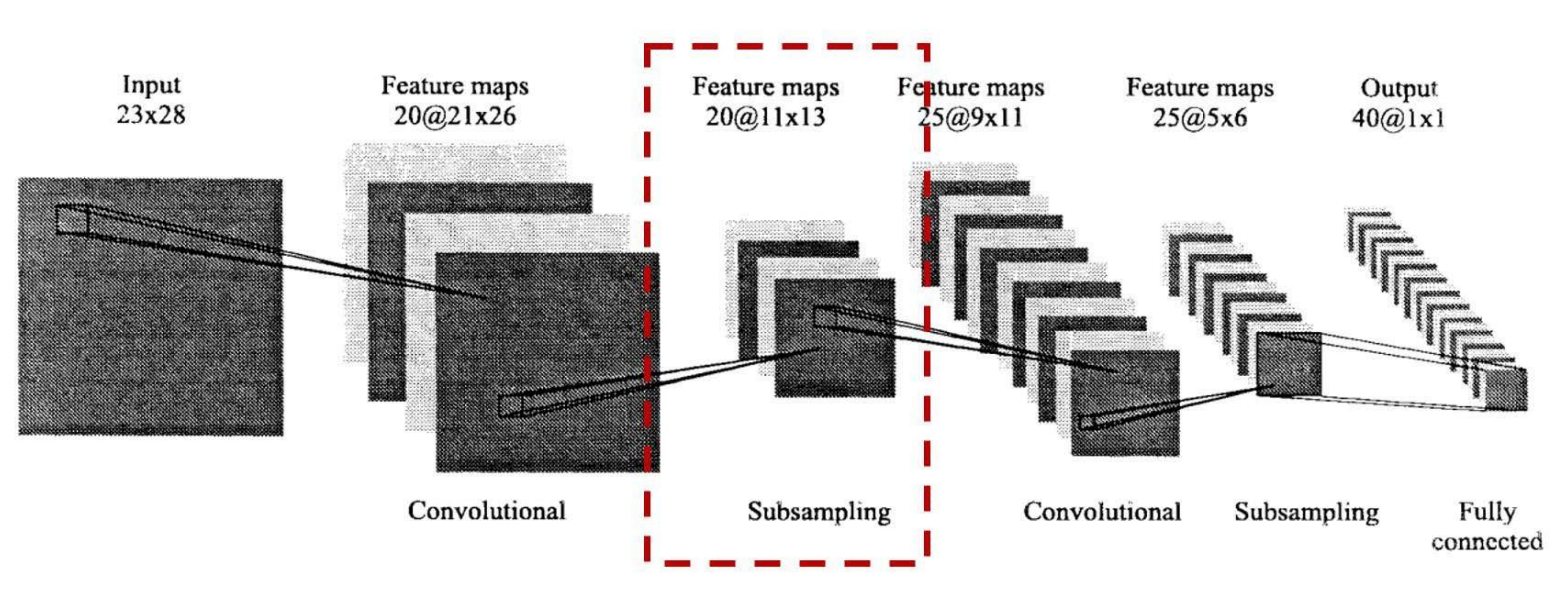
POOLING WITH DOWNSAMPLING



POOLING WITH NO DOWNSAMPLING



LAYER



Input
23x28
Feature maps
20@21x26
Feature maps
20@11x13
Feature maps
25@9x11
Feature maps
25@5x6
Output
40@1x1
Convolutional
Subsampling
Convolutional
Subsampling
Fully
connected

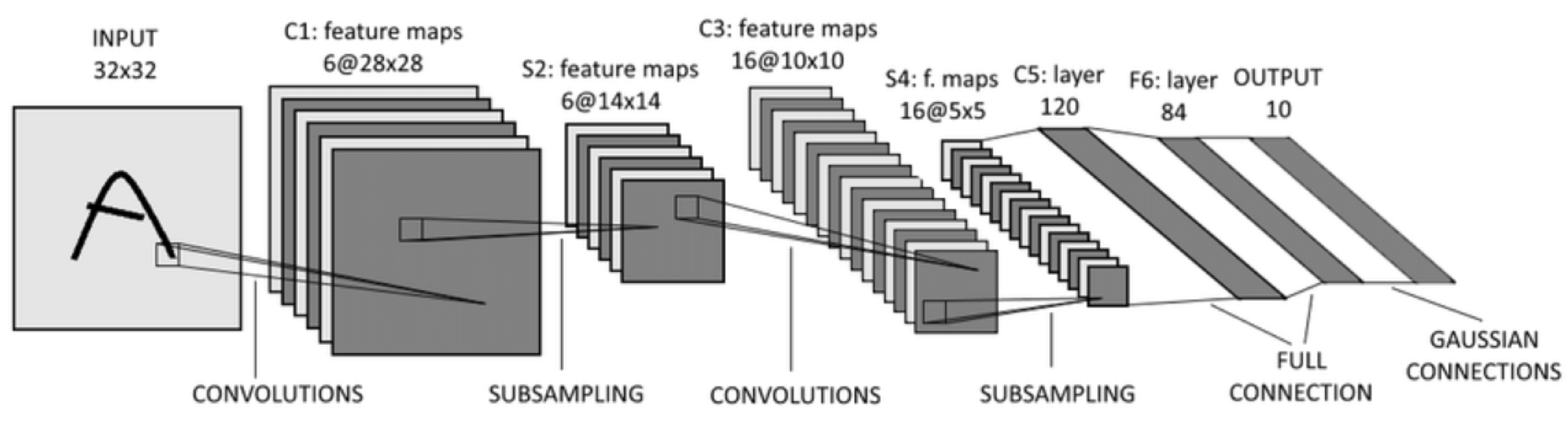
DESIGN AND IMPLEMENTATION

ARCHITECTURE

- No one-size-fits-all approach to architectures
- Carefully consider input data including dimensions, size and channels
- Consider other parameters, such as pooling and kernel receptive field sizes
- Don't necessarily have to re-invent the wheel - other existing model architectures might be useful
- Can use other evolutionary algorithms to help determine right architectures and hyperparameters

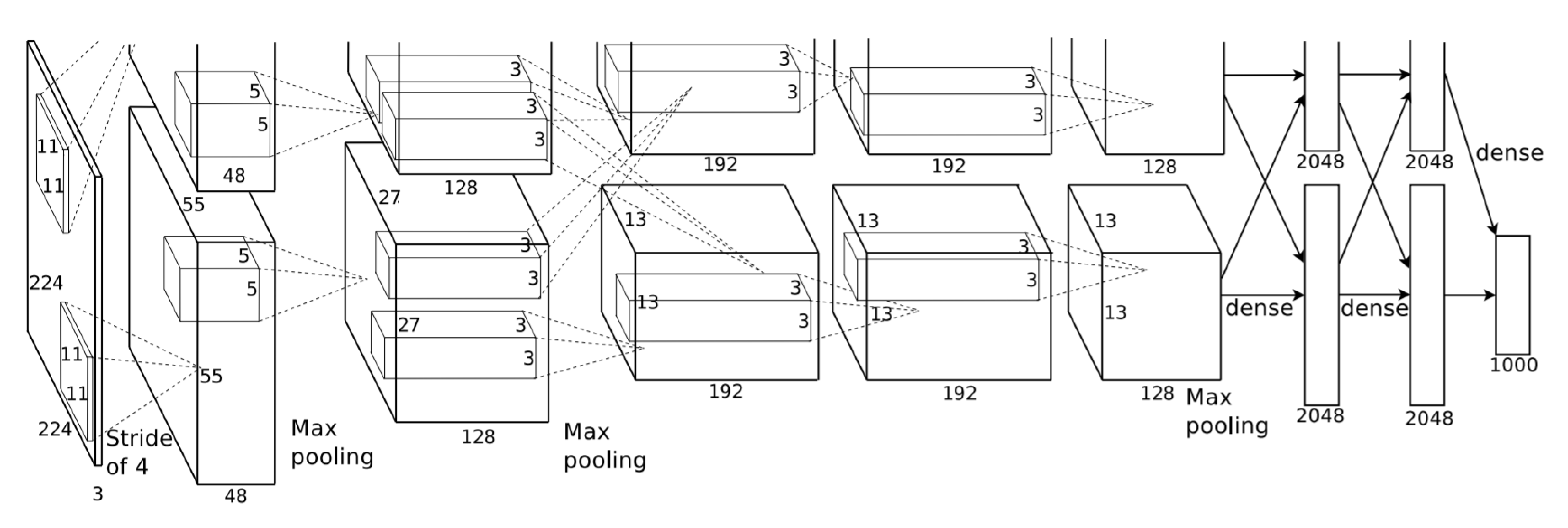
EXAMPLE ARCHITECTURES

LeNet



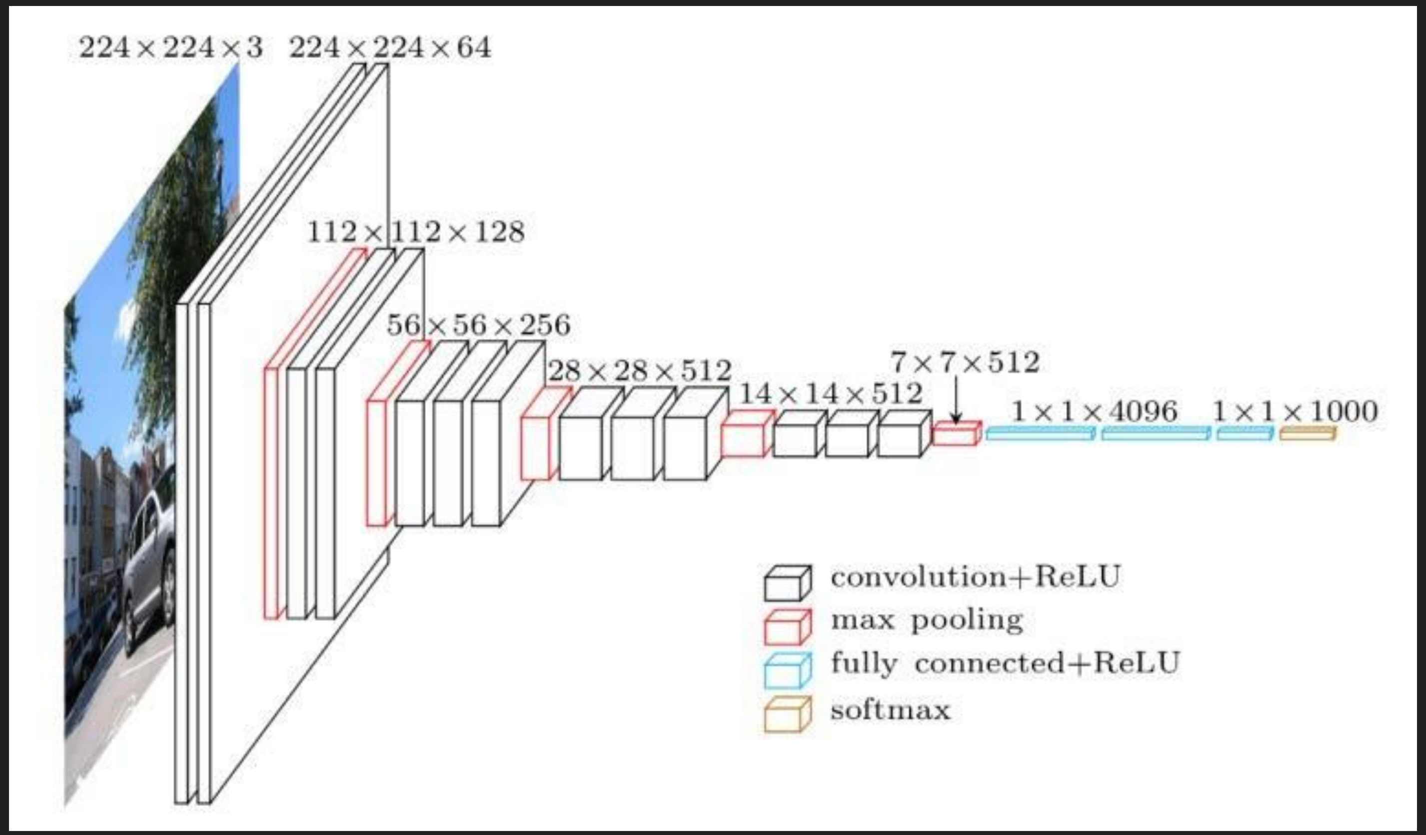
EXAMPLE ARCHITECTURES

AlexNet



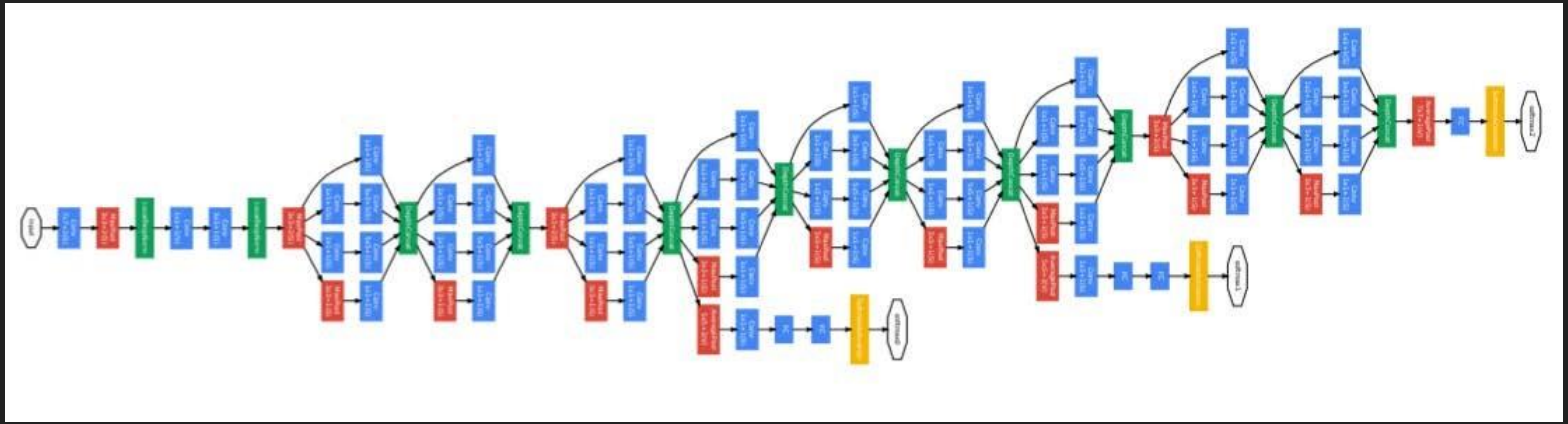
EXAMPLE ARCHITECTURES

VGG



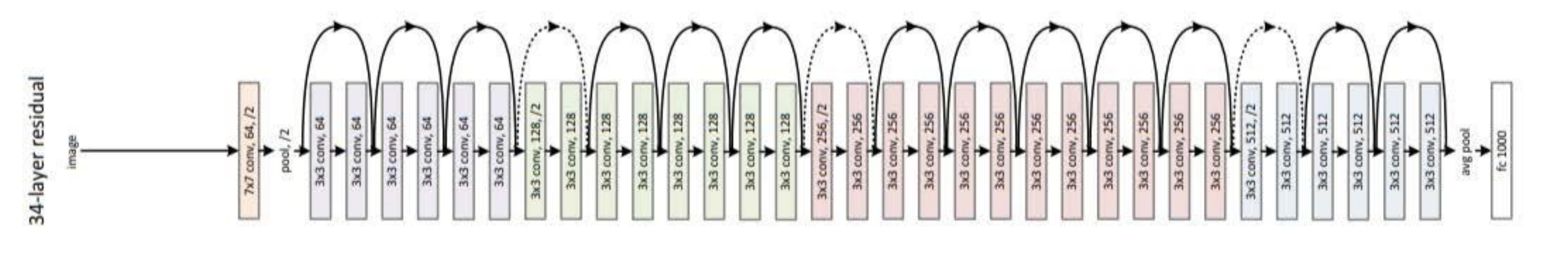
EXAMPLE ARCHITECTURES

GoogLeNet



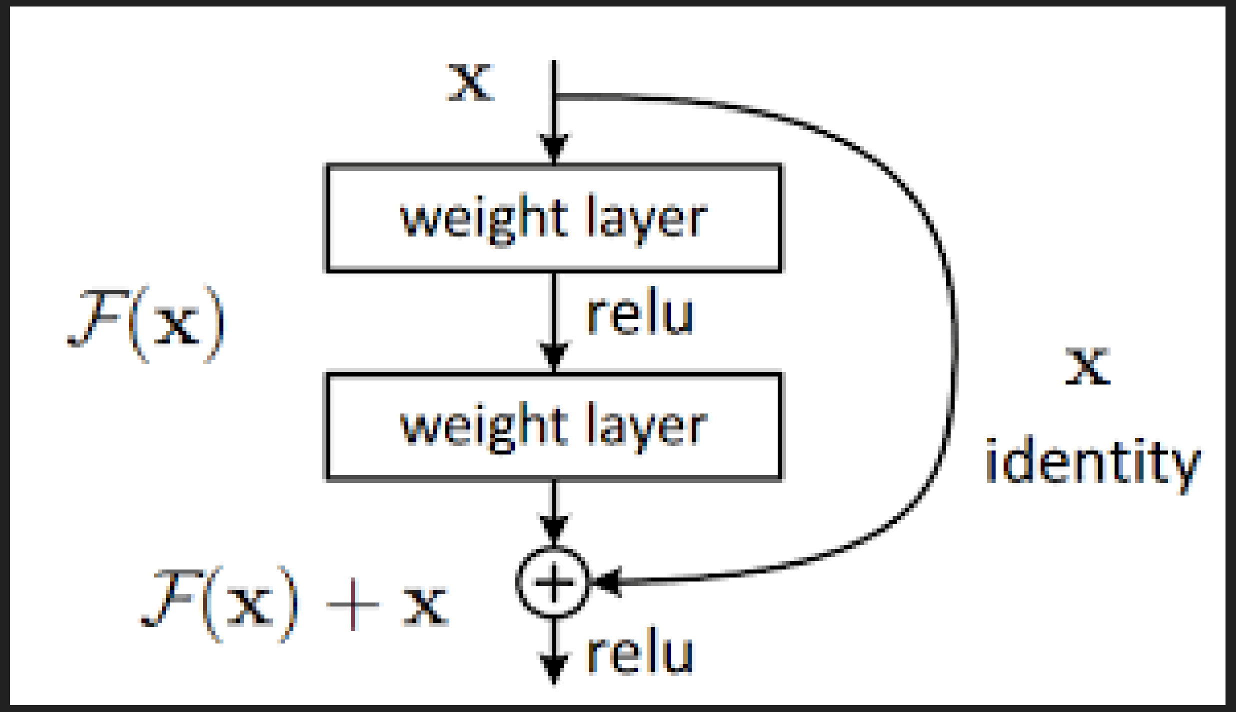
EXAMPLE ARCHITECTURES

ResNet



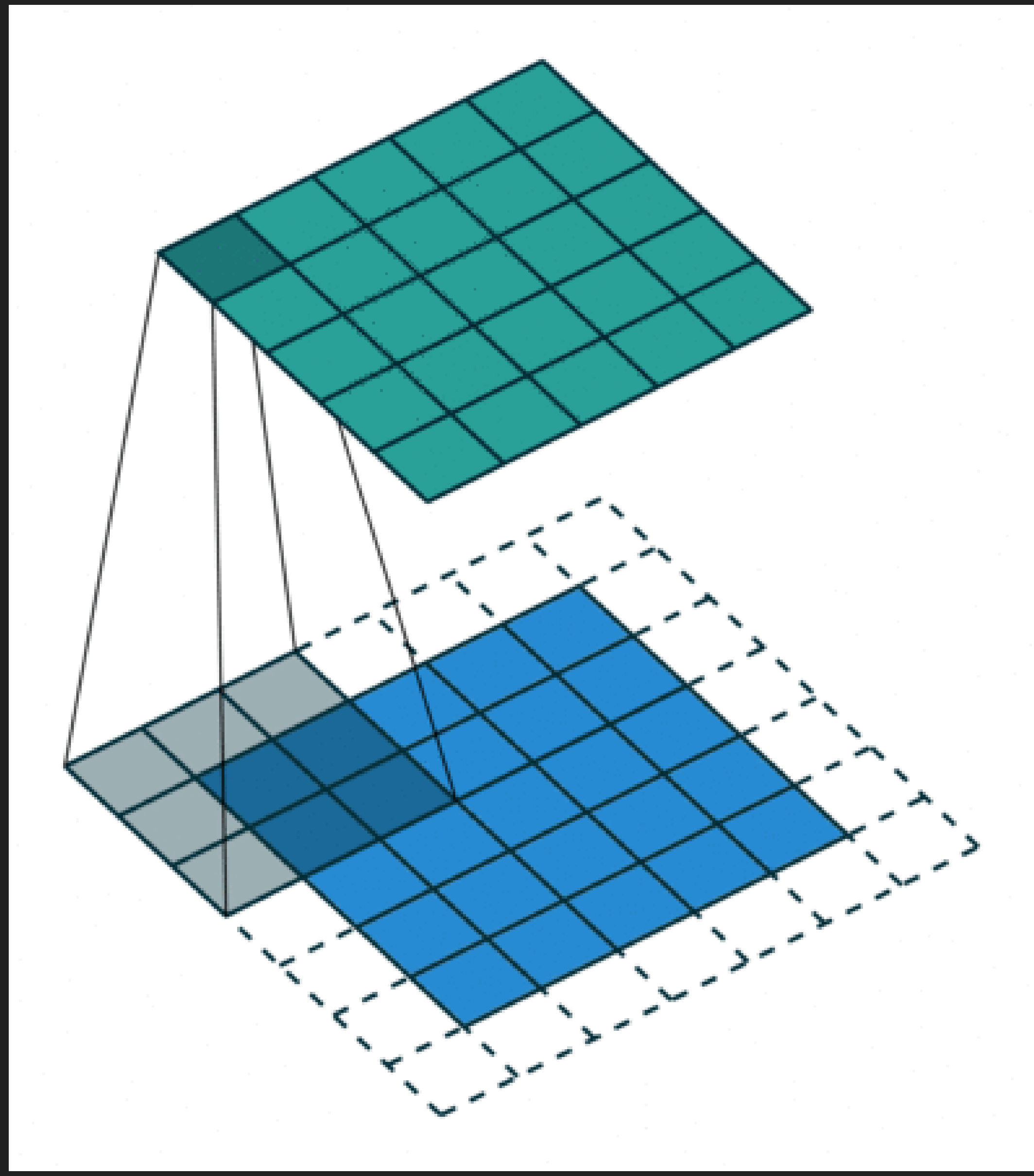
RESIDUAL CONNECTIONS

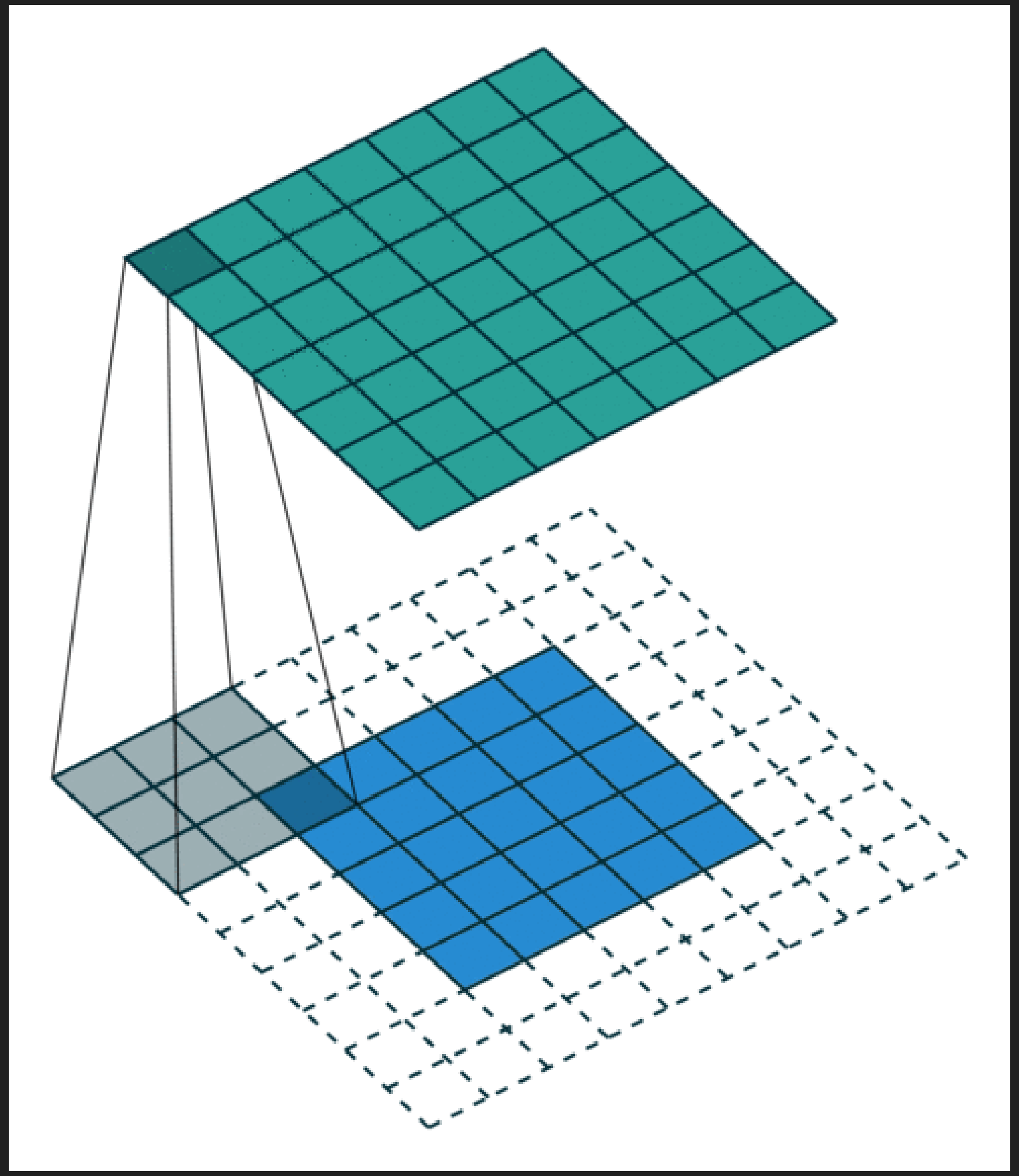
- Add input of block to output.
- Creates clear path of unmodified gradients through all layers.



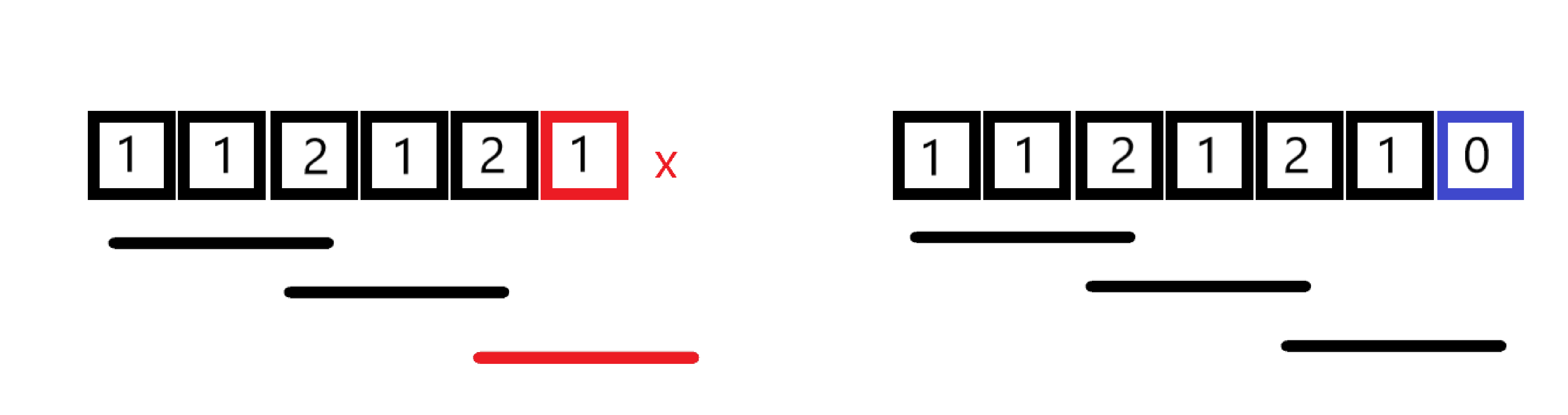
PADDING

- Adds dummy data to the input
- As filter receptive fields are applied, the spatial dimensions can decrease faster than desired
- Padding allows spatial dimensions to be maintained and for filters to be applied in situations where it would exceed the input dimensions





PADDING



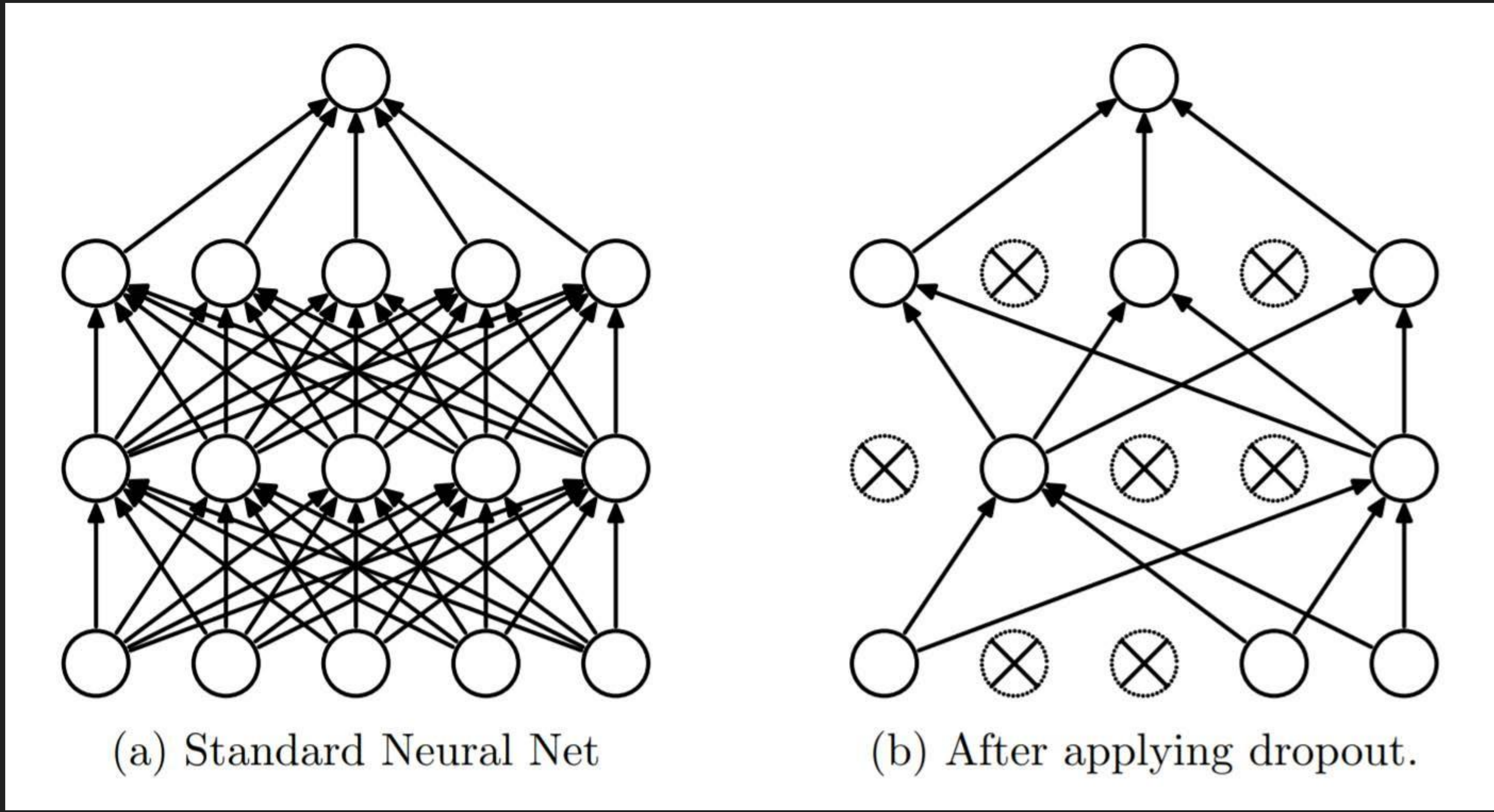
REGULARISATION

- A central goal of machine learning is making an algorithm perform well on new, unseen data points
- In ANNs and deep learning, several strategies exist for regularisation, including
  - Parameter sharing
  - Dataset augmentation
  - Dropout

DROPOUT

- During each training pass, randomly remove a fraction of neural connections
- Adds noise to hidden units, particularly in the DNN classifier
- Inspired by sexual reproduction: half genes from both parents plus a small amount of random mutation results in offspring

DROPOUT



BATCH NORMALISATION

- Batch normalisation reduces the the internal covariate shift
- Implicitly regularizes the model due to the noise in the batch estimates for mean and variance

CODE EXAMPLES

PYTORCH

```
import torch
import torch.nn as nn

class CNN(nn.Module):
        def __init
          (self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(
                in_channels=1,              # channels in input (eg: 1 could be black/white images)
                out_channels=16,            # number of filters/features we
                want kernel_size=5,              # 5x5 local receptive field
                stride=1,                   # stride step of 1 (will overlap)
                padding=2,
            ),
            nn.ReLU(),                      # activation function
            nn.MaxPool2d(kernel_size=2),     # Apply max pooling (pool size of 2 means a 2x2 pooling window)
        )
        self.conv2 = nn.Sequential(          # another series of layers stacked
            nn.Conv2d(16, 32, 5, 1, 2),      # in channels must equal out channels of previous layer (16)
            nn.ReLU(),
            nn.MaxPool2d(2)
            ,
        )
        self.out = nn.Linear(32 * 7 * 7, 10)   # a DNN with 10 output classes

    def forward(self,
        x): x =
        self.conv1(x) x
        = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output, x    # return x for

visualization cnn = CNN()
```

SUMMARY

- CNNs are inspired by the visual processing pathways of the brain
- Perform convolutions between input and weight tensors
- Pooling can downsample and provide translation invariance
- A DNN classifier maps CNN-produced representational features to outputs