UNIVERSITÀ degli STUDI di CATANIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

Convolutional Neural Net

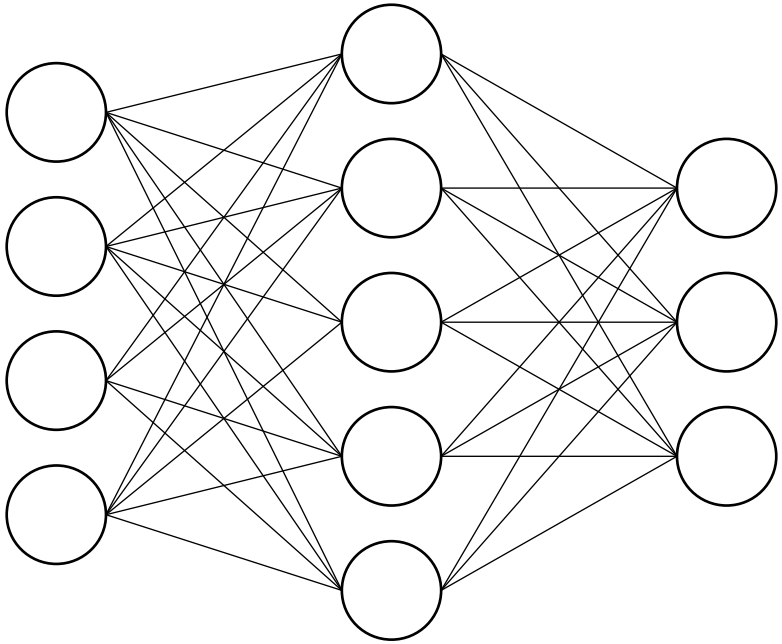
Further reading

Chapter 9 - Deep Learning

I. Goodfellow, Y. Bengio, A. Courville

IMAGE PROCESSING LABORATORY
http//iplab.dmi.unict.it

MLP



input layer hidden layer output layer

the network has 4 inputs and 3 outputs: $\mathrm{F}_{\boldsymbol{\theta}}(\boldsymbol{x}) = \theta_h f\left(\theta_f \boldsymbol{x} + b_f\right) + b_h$

Neural Networks: Representation Learning

The MLP performs two transformations:

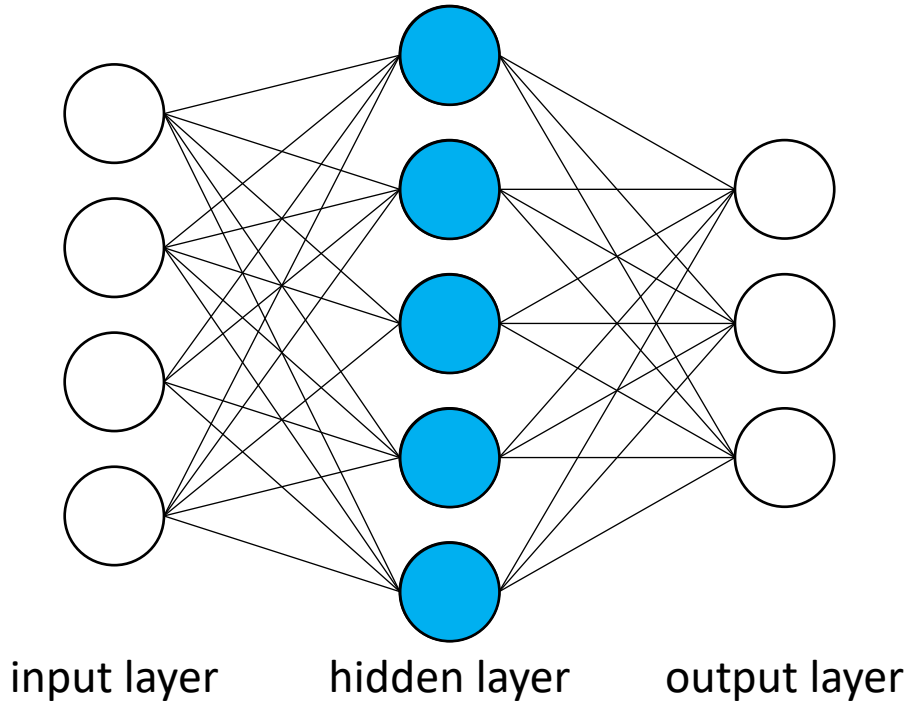
1. Transformation of input $\boldsymbol{x}$ into $\mathrm{a}(\boldsymbol{x}) = f(\boldsymbol{\theta}_f \boldsymbol{x} + b_f)$;
2. Transformation of intermediate result $\mathrm{a}(\boldsymbol{x})$ into output $\mathrm{h}(\mathrm{x}) = \boldsymbol{\theta}_h f(\boldsymbol{x}) + \boldsymbol{\theta}_h$.

The ''intermediate result'' $\mathrm{a}(\boldsymbol{x})$ is a **representation of $\boldsymbol{x}$**.

Therefore, the MLP is performing representation learning.

Neural Networks: Representation Learning

learned representation



Artificial Neural Networks: training

In general, a neural network implements a function which maps input data $\boldsymbol{x}$ to output data $\boldsymbol{y}$, given parameters $\boldsymbol{\theta}$:

$$\boldsymbol{y} = h(\boldsymbol{x}; \boldsymbol{\theta}) = \mathrm{F}_{\boldsymbol{\theta}}(\boldsymbol{x})$$

If we have a sufficient number of example pairs (dataset) including input data and desired output $< \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)} >$, we can try to find the optimal $\boldsymbol{\theta}$ such that:

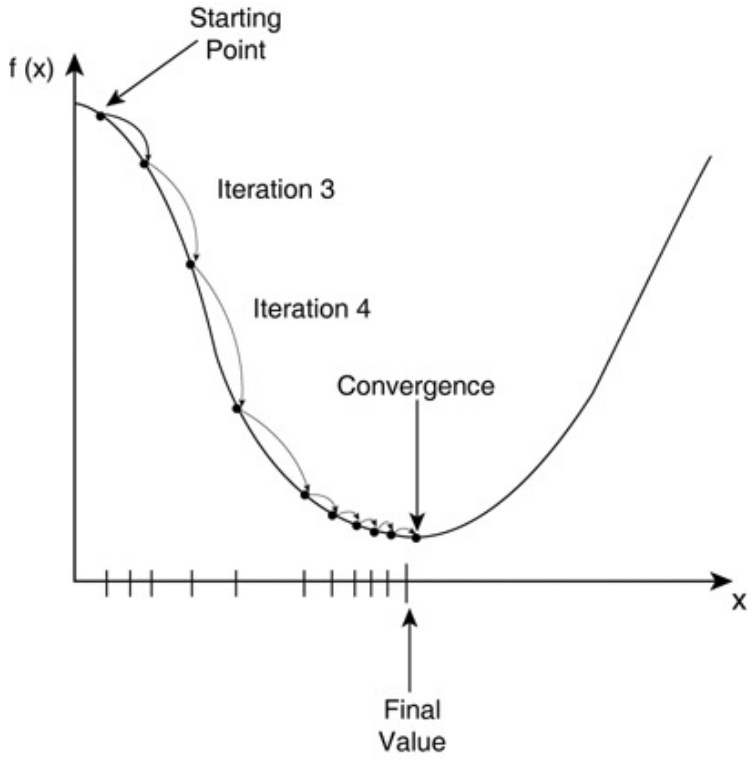
$$\sum loss\left(\boldsymbol{y}^{(i)}, h\left(\boldsymbol{x}^{(i)}, \boldsymbol{\theta}\right)\right)$$

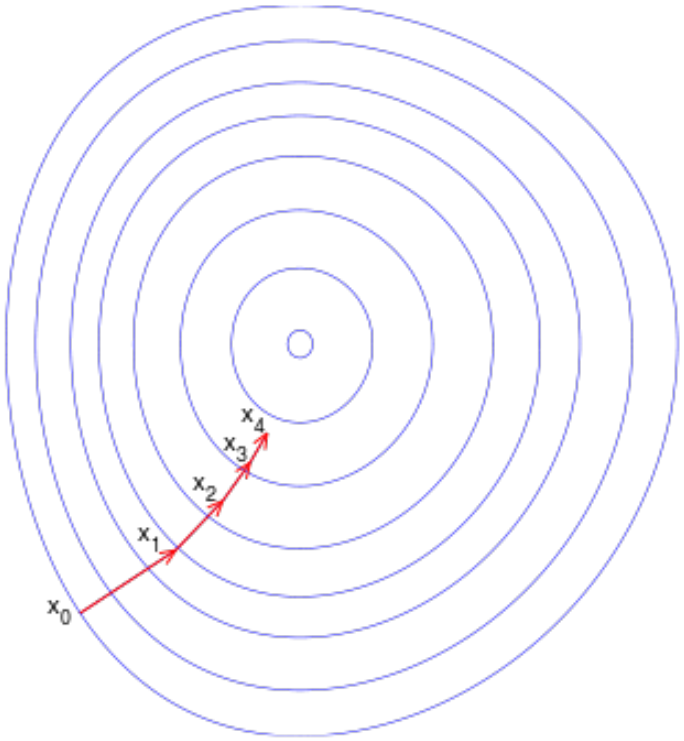
is minimal, where $loss$ function measures the error commited by function h when predicting $\boldsymbol{y}$.

The procedure of finding optimal $\boldsymbol{\theta}$ is called training procedure. The most common training algorithm for training is the backgpropagation.

Backpropagation
(See previous lesson)

Since h is differentiable in $\boldsymbol{\theta}$, the backpropagation algorithm works using the gradient descent principle.





Neural Net Example: Regression

Neural nets can approximate mapping functions F. However, how can such a mapping be useful in practice?

Regression problem: predict miles per gallon (mpg) consumption of a car given:

- number of cylinders (c);
- horsepower (p);
- weight (w);

Dataset of 398 instances.
Each instance contains <mpg,c,h,w>

It looks reasonable to belive that it exists some function h such that:

$$\text{mpg} = h(c, p, w)$$

Why learning from data is useful? –> how shall we define such relationship mathematically?

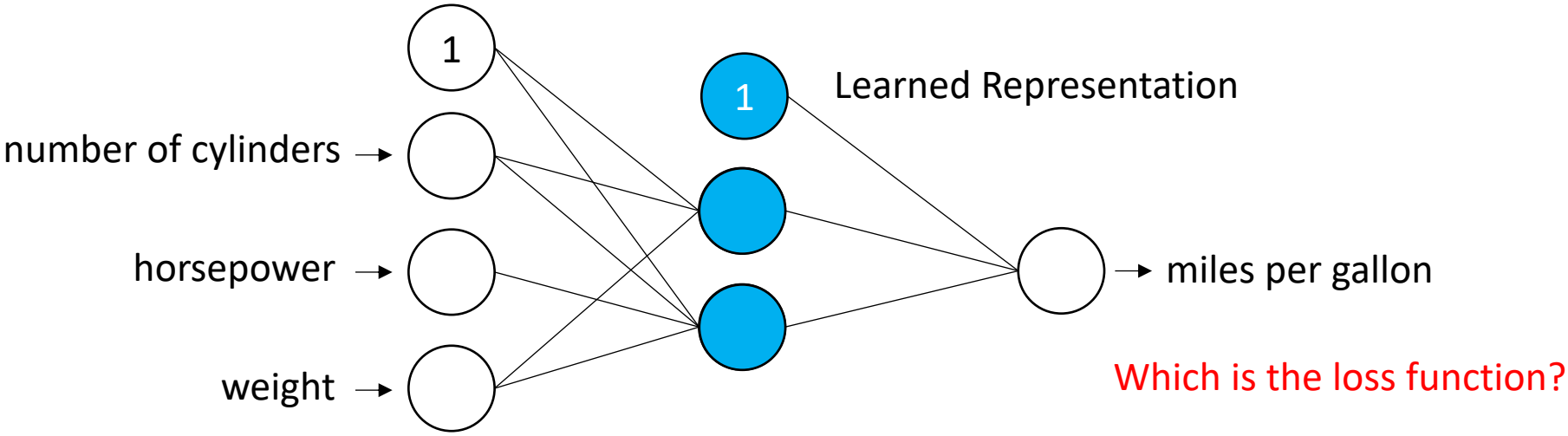
We don't. We learn it from data.

https://archive.ics.uci.edu/ml/datasets/auto+mpg

Neural Net Example: Regression

Regression problem: predict miles per gallon (mpg) consumption.

We want to find a mapping function h to compute miles per gallon from features. Let us consider the following network:



$$\mathrm{F}_{\boldsymbol{\theta}}(\boldsymbol{x}) = h(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{\theta}_h \alpha\left(\boldsymbol{\theta}_f \boldsymbol{x} + b_f\right) + b_h$$

We can use our dataset to learn good weights to approximate the desired function.

Neural Net Example: Classification

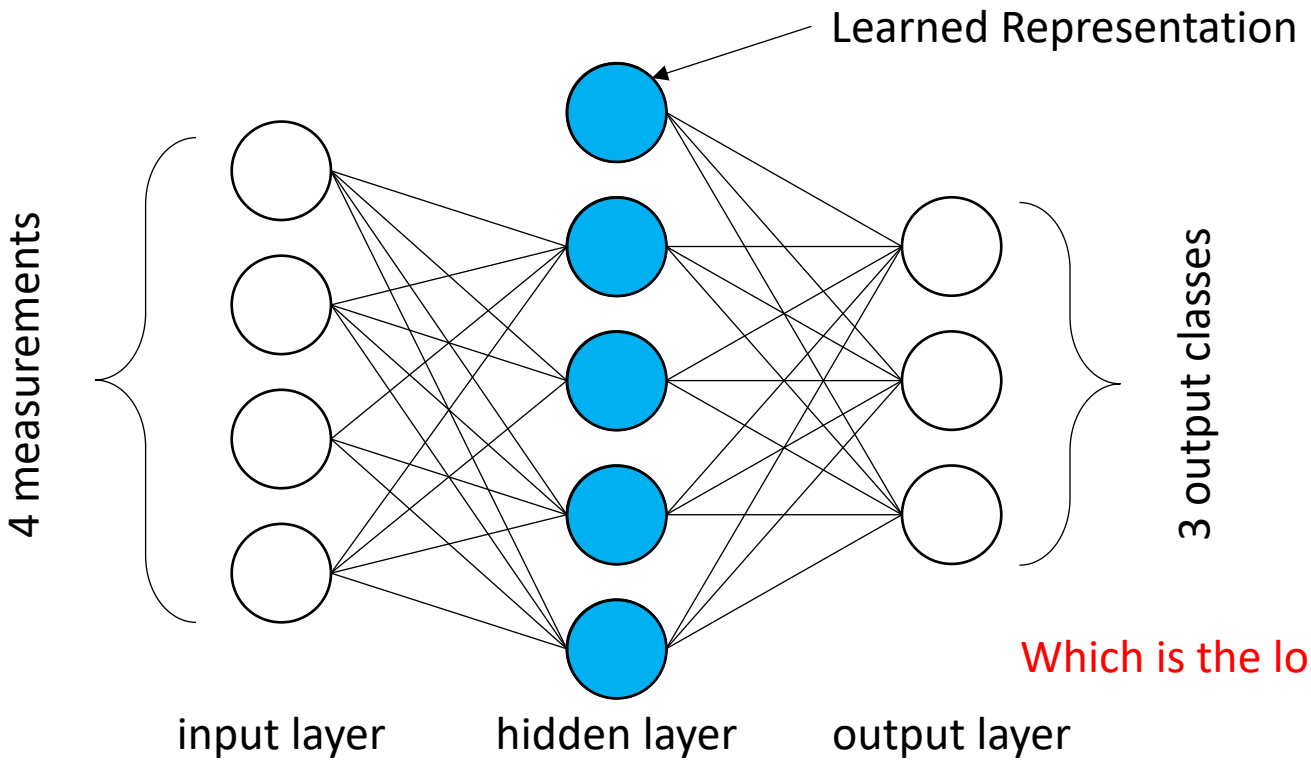
What if we want to classify the input into n classes?

Classification problem: dataset of measurements from 150 instances of Iris (4 measurements per instance), grouped into 3 species. Classify flowers given the 4 measurements.

- Inputs: 4 (the four measurements);
- Ouputs: ?? 3

A common technique is to let the function compute one score per class, independently, then pick the largest score to perform classification.

Neural Net Example: Classification

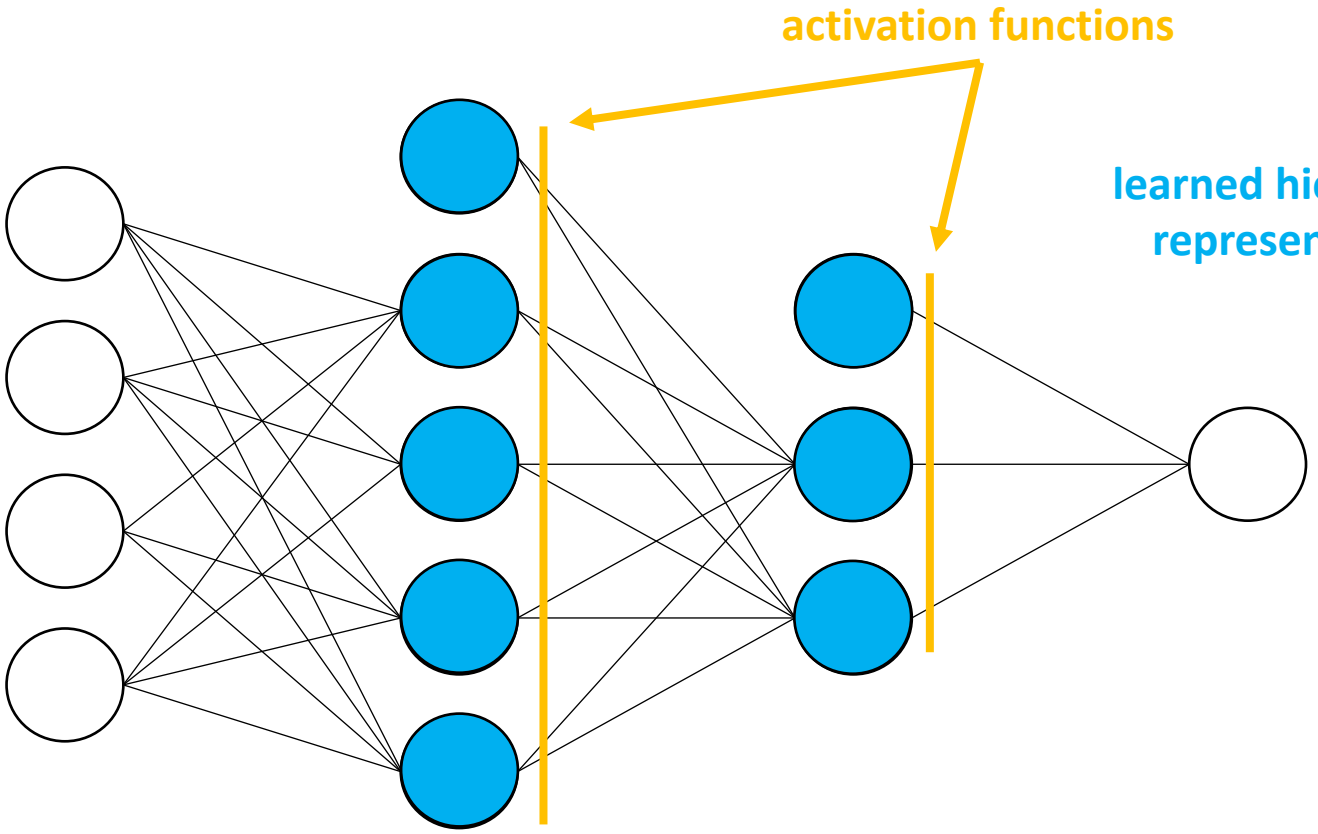


Which is the loss function?

Training time: find the weights which allow to correctly classify flowers (high score for target class, low score for all others.

Test time: pass a sample through the network and pick the class with highest score.

MLP: From Shallow to Deep



$$h(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}_h f_2(\boldsymbol{\theta}_{f_2} f_1(\boldsymbol{\theta}_{f_1} \boldsymbol{x} + b_{f_1}) + b_{f_2}) + b_h$$

In general, we can add any number of hidden layers.

Universal Approximation Theorem

Do we really need to go deep? **Well, theoretically no**. The universal approximation theorem states that:

a multilayer perceptron with a single hidden layer containing a finite number of units can approximate continuous functions on compact subsets of R^n

In practice however, the number of hidden units needed to approximate a given function can get very large. Hence, **in practice, it is useful to "decompose" the network introducing additional hidden layers.**

MLP and images

Multilayer perceptrons are not well suited to process images. The main motivations are the following ones:

- Images are big: a small $100\ x\ 100$ grayscale image contains 10000 pixels;
- Images are very complex and several layers may be needed to process them;
- When the number of layers and number of inputs gets very high, the number of parameters increases accordingly;

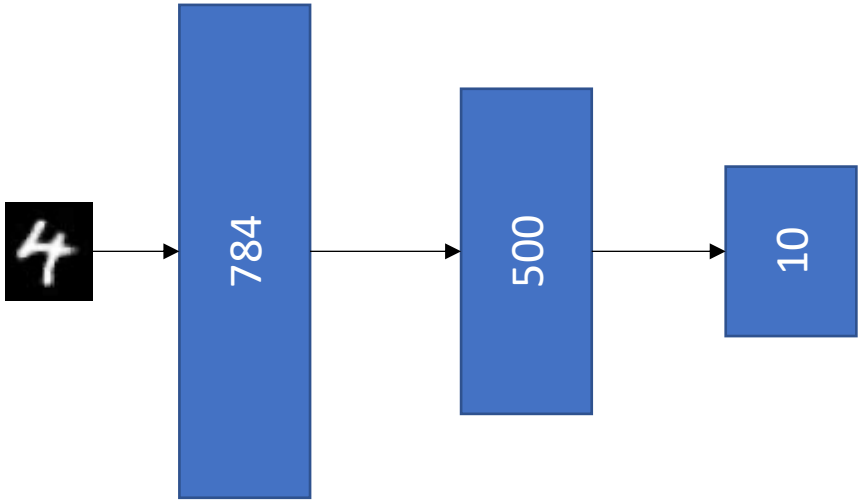
-> IDEA: reduce the number of parameters in each layer and build deeper models

MNIST Handwritten Digits Classification Problem and Dataset

- 70K images (28×28 pixels);
- 10 classes (digits from 0 to 9);

Simple Approach: MLP

- Reshape 28×28 images to 784×1 vector;
- Build a MLP with 1 hidden layer and 500 units;
- Output layer with 10 units (one per class).



$$h(\boldsymbol{x},\boldsymbol{\theta}) = \boldsymbol{\theta}_h f\left(\boldsymbol{\theta}_f \boldsymbol{x} + b_f\right) + b_h$$

How many parameters?

$\boldsymbol{\theta}_f[500\times784]$
$b_f[500\times1]$
$\boldsymbol{\theta}_h[10\times500]$
$b_h[10\times1]$

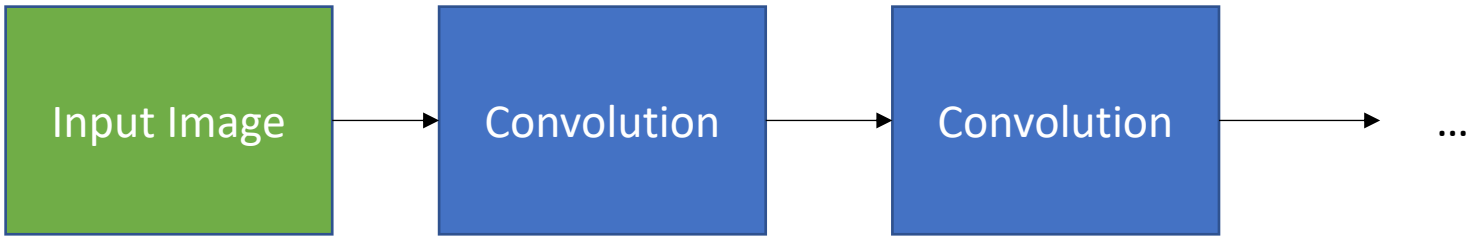
Total = $500 \cdot 784 + 500 +$
$+10 \cdot 500 + 10 =$
397 510

Convolutional Neural Networks (CNN)

Convolutional Neural Networks

Convolutional Neural Networks are designed to scale up neural networks to process very large images and video sequences.

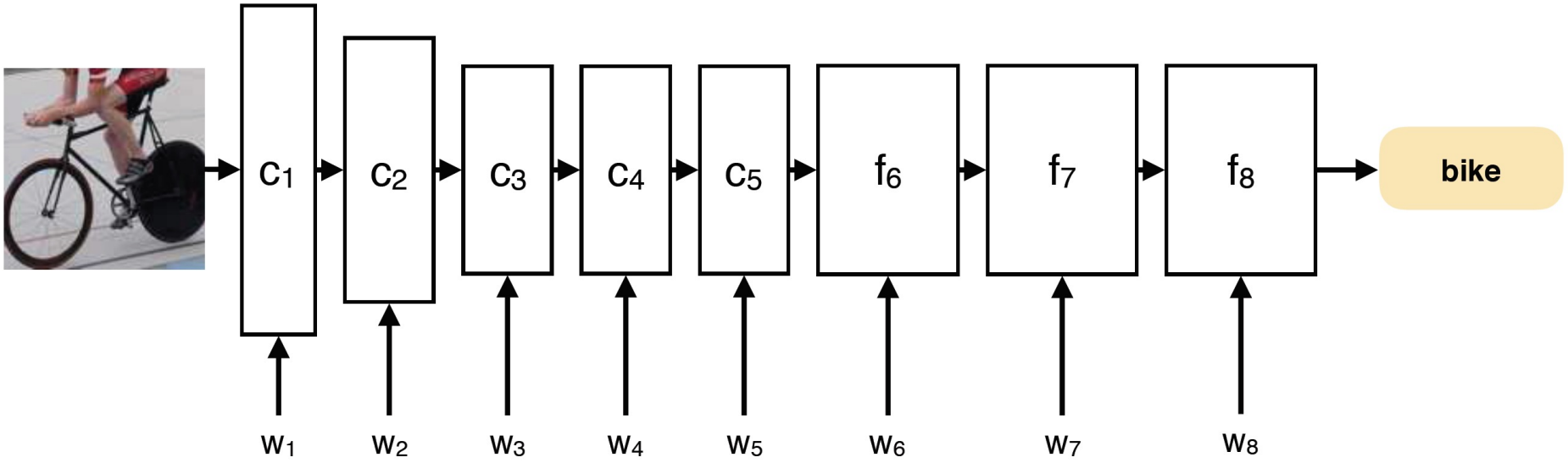
The main idea is to replace matrix multiplication with convolution.



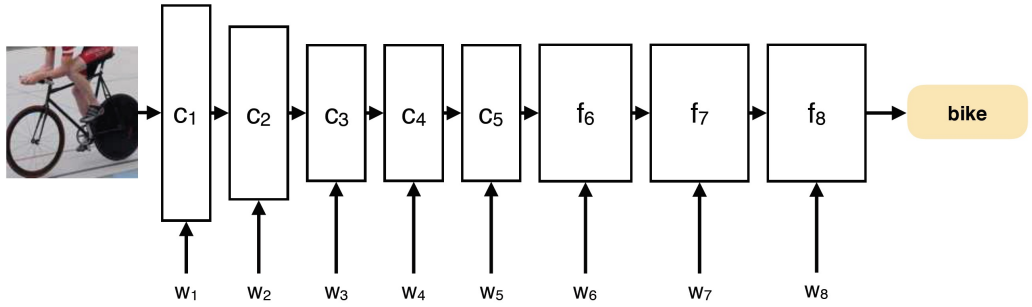
OBSERVATION: feature extraction should be **invariant to translation** along x and y axes. Regular **matrix multiplication would learn different weights at different positions**. **Convolutions** retain **invariance with respect to translation**.

Convolutional Neural Networks

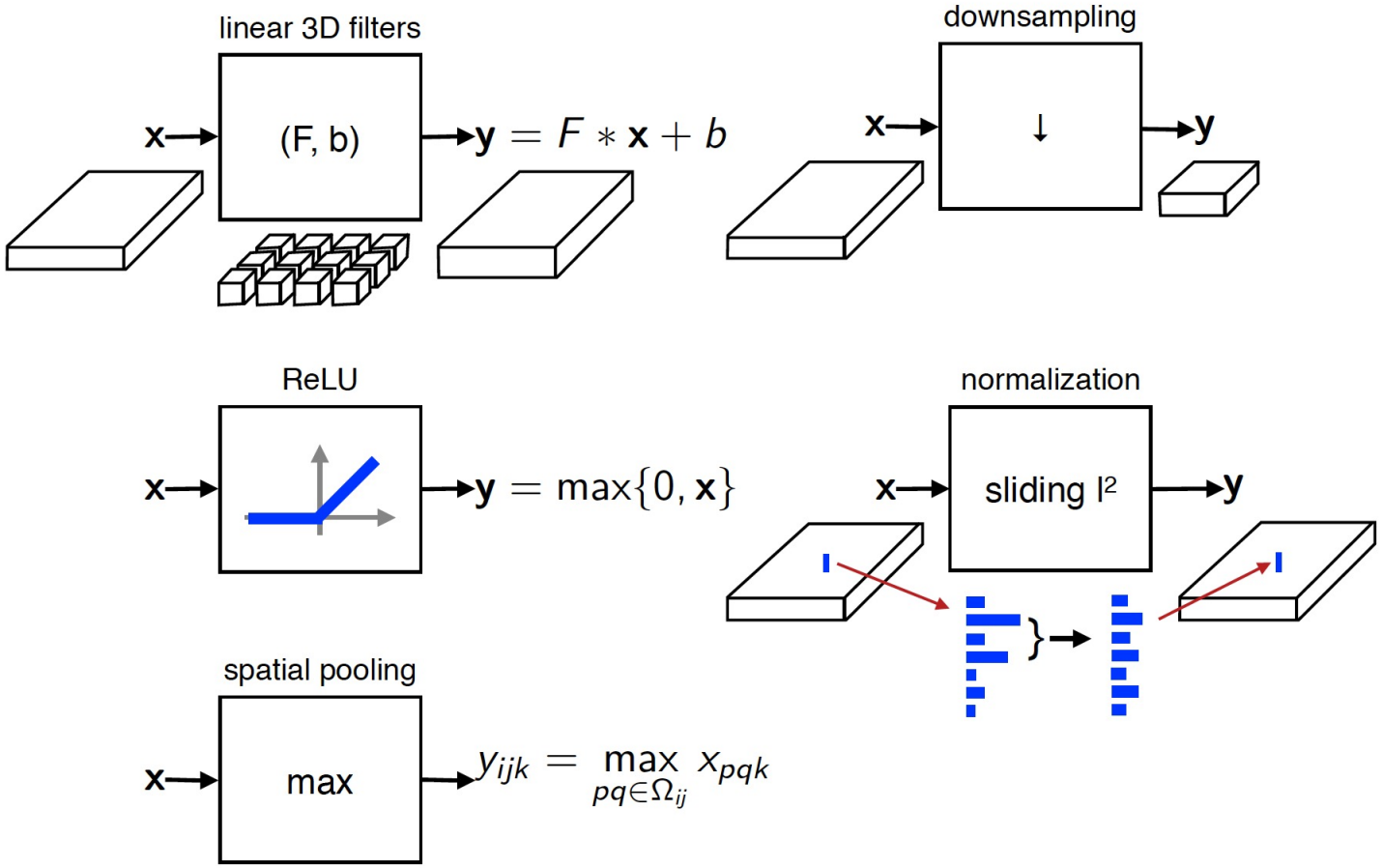
Architecture and Components



A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet classification with deep convolutional neural networks*. In Proc. NIPS, 2012.

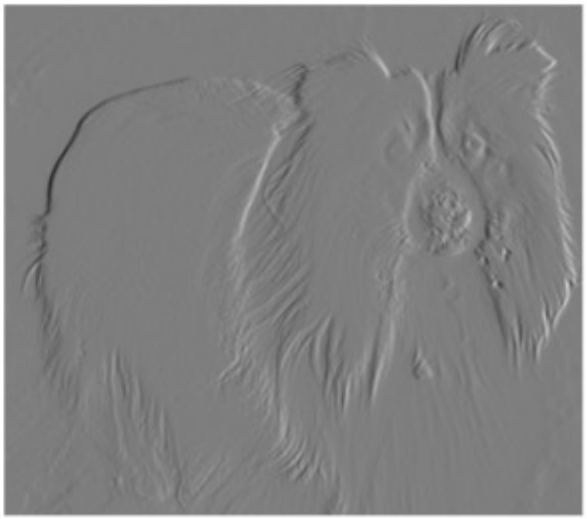


Exploitable blocks in the convolutional layers



Example of 2D Convolution

Input



Output

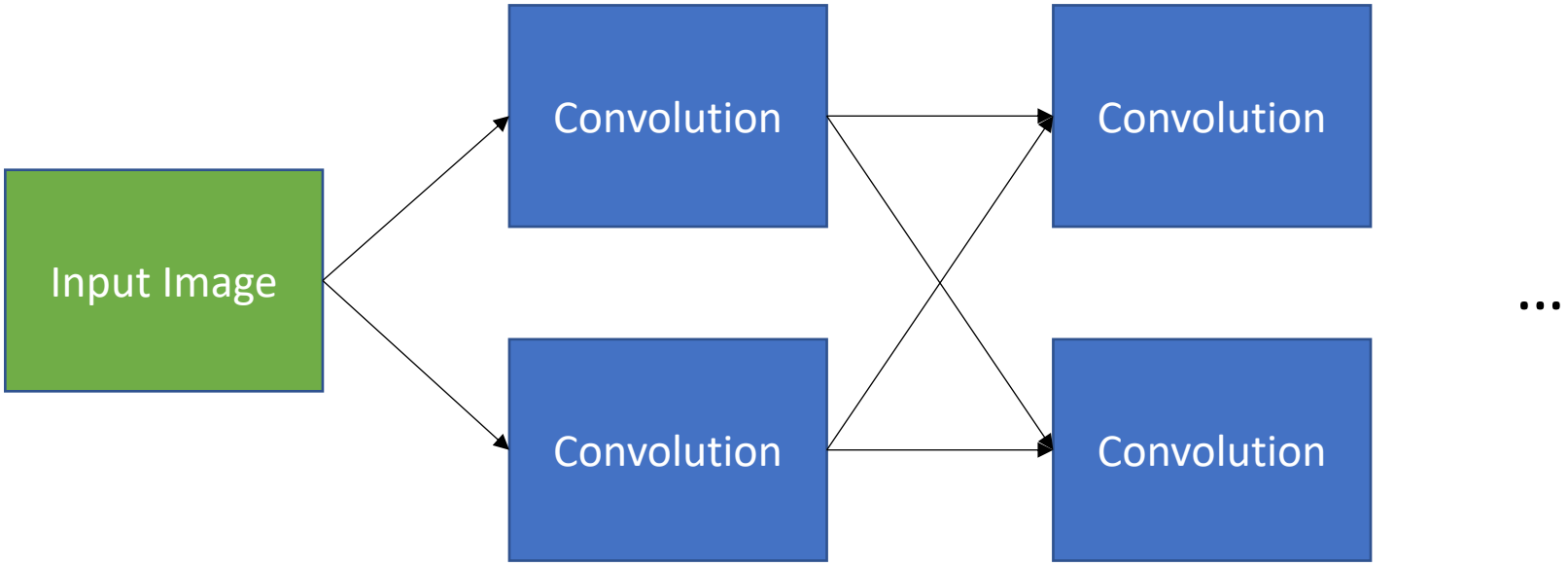
1	-1

Kernel

Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning. 2016. MIT Press.

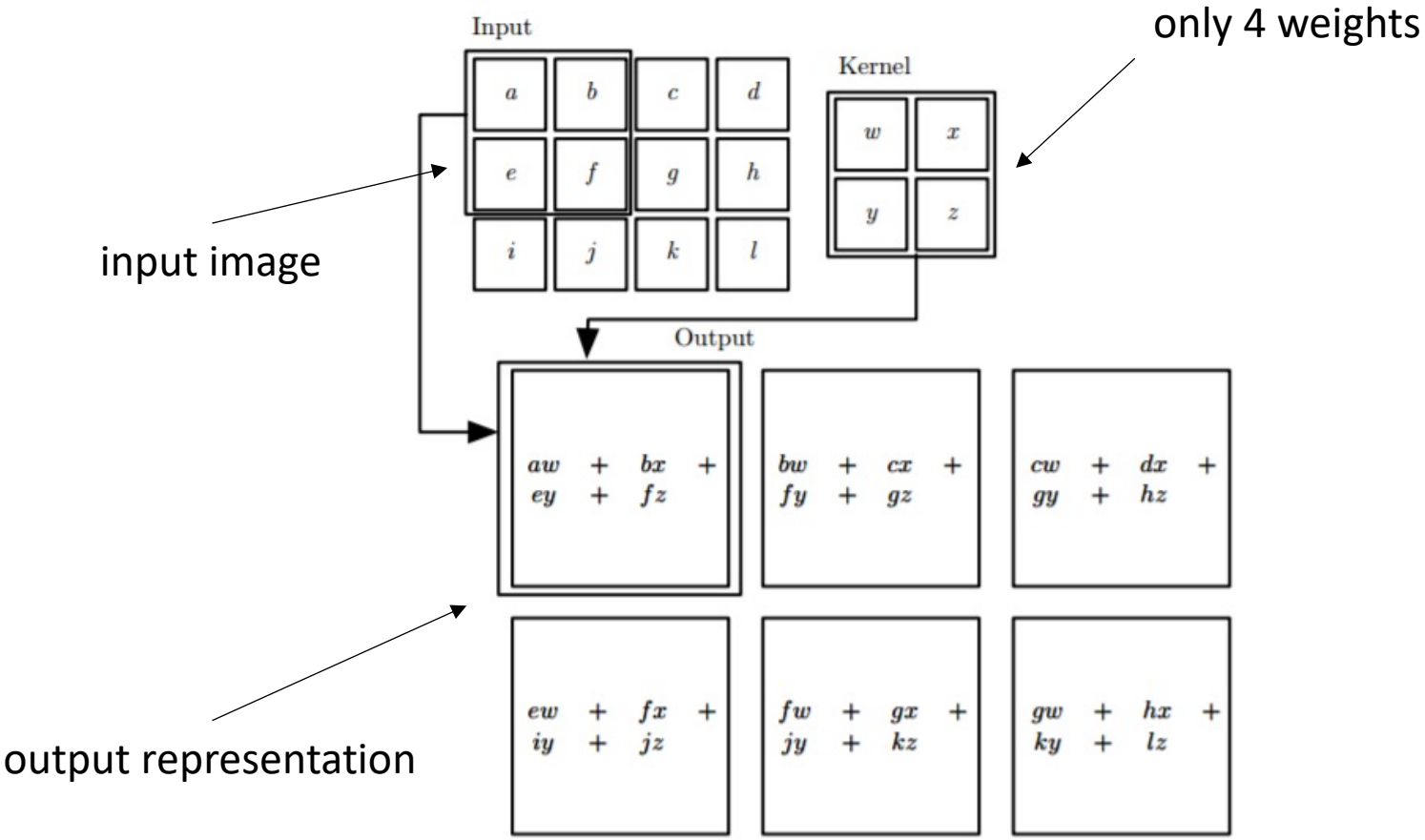
2D Convolutions

In a given layer, we can learn more than a convolutional kernel. This is similar to having multiple perceptrons in the same layer.



2D Convolution (Cross-Correlation)

In this case, the weights to be learned in a given layer are the values of the kernel.



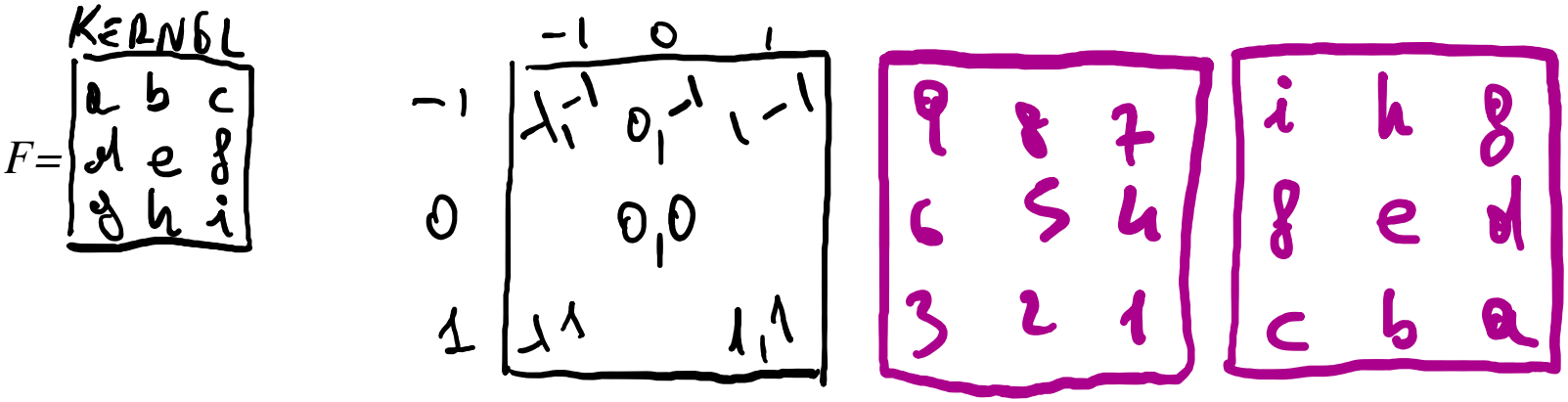
Ian Goodfellow and Yoshua Bengio and Aaron Courville. Deep Learning. 2016. MIT Press.

Cross-Correlation vs Convolution

Cross-Correlation

$$G = h \otimes F$$

$$G[i,j] = \sum_{u=-k}^{} \sum_{v=-k}^{} h[u,v]F[i+u, j+v]$$



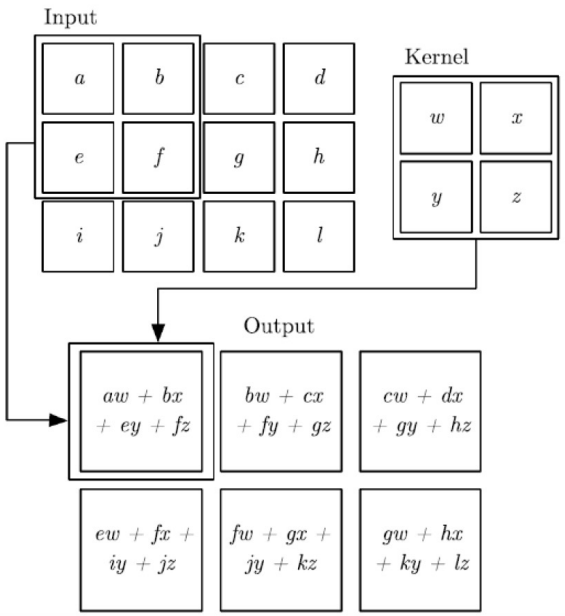
Convolution

$$G = h * F$$

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v]F[i-u, j-v]$$

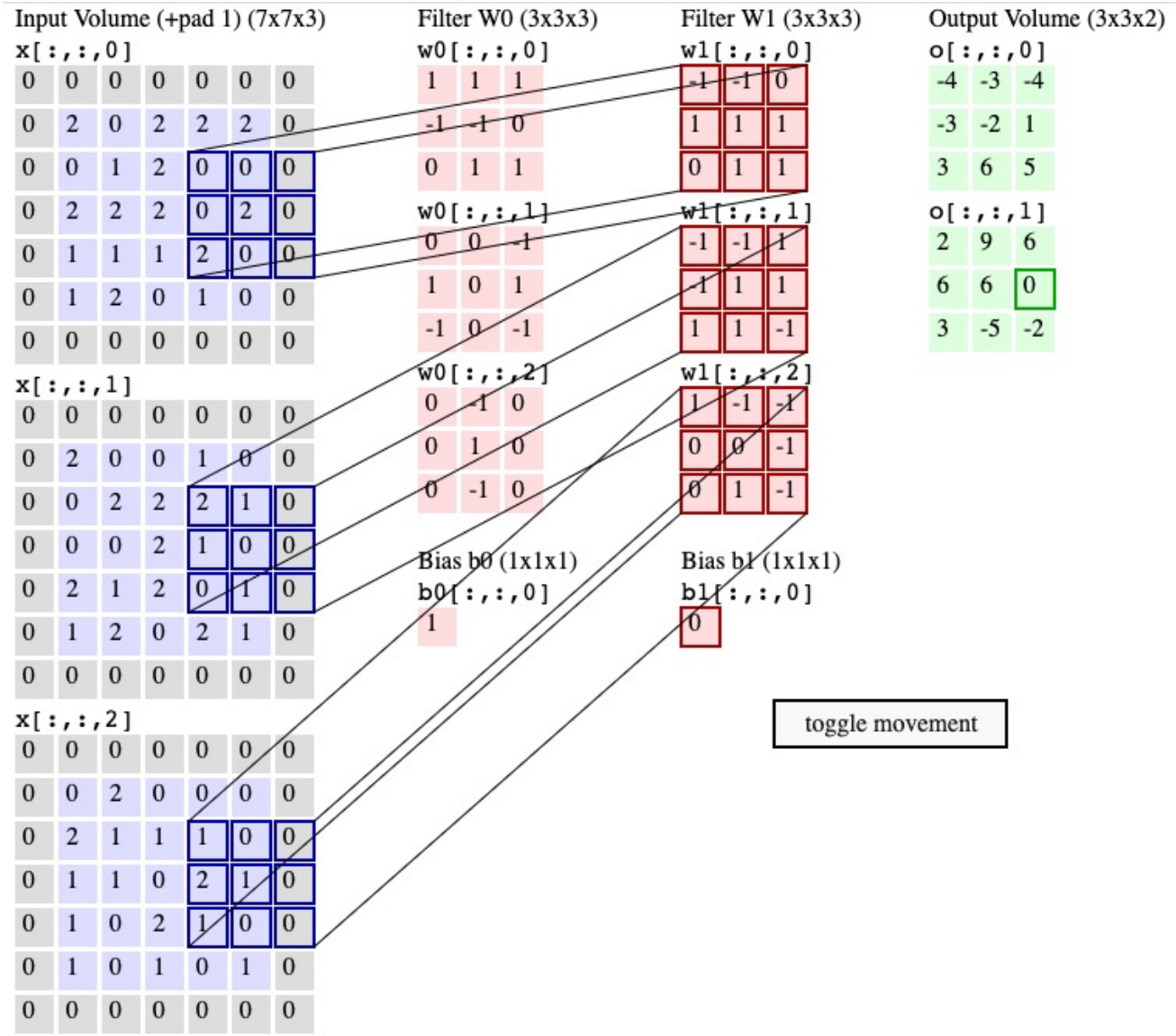
Convolutions

- A convolutional layer's **output** shape is **affected by** the **shape of its input** as well as the **choice of kernel shape, zero padding, dilatation and strides**.
- Demo 1
- Demo 2
- Demo 3
- Demo 4

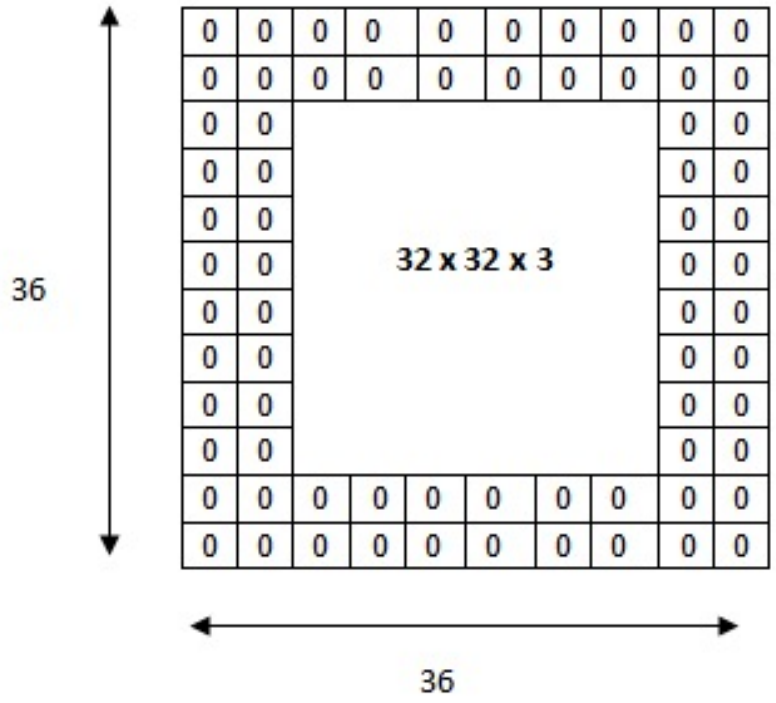


- The weights correspond to the kernel
- The weights are shared in a channel (depth slice)
- We are effectively learning filters that respond to some part/entities/visual-cues etc.

Example 3D Convolution

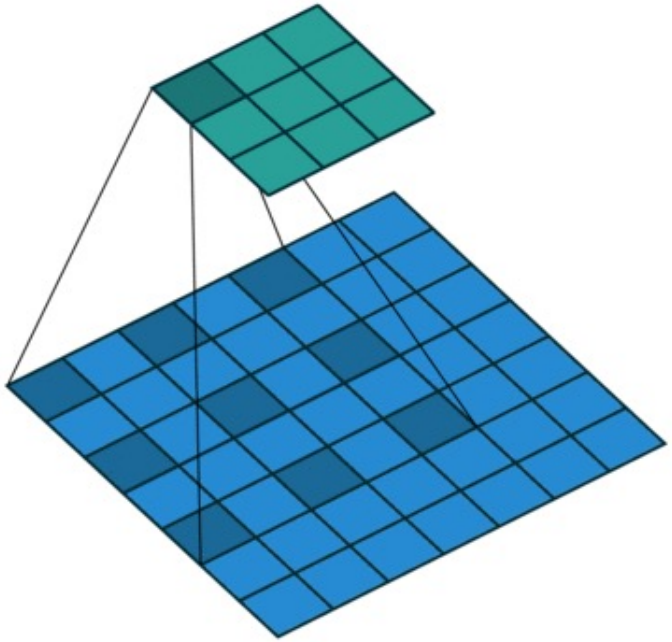


Padding



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

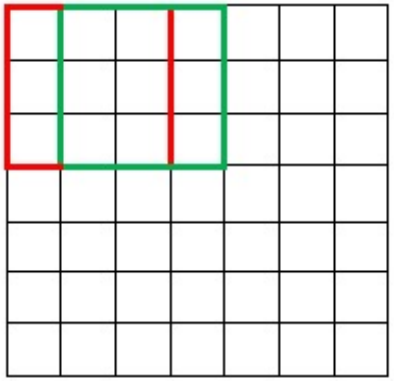
Dilatation



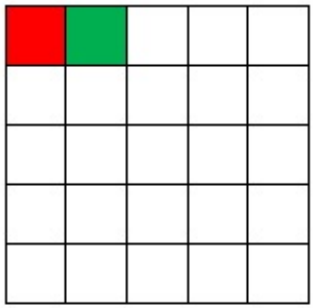
The more **important** point is that the architecture is based on the fact that **dilated convolutions** support exponential expansion of the receptive field without loss of resolution or coverage. Allows one to have larger receptive field with same computation and memory costs while also preserving resolution.

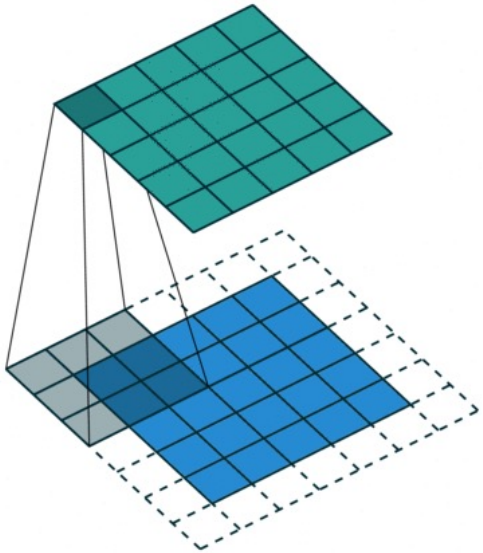
Stride

7 x 7 Input Volume

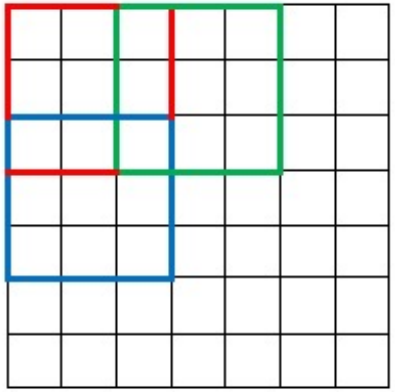


5 x 5 Output Volume

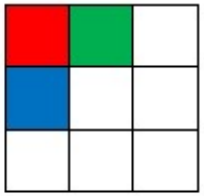


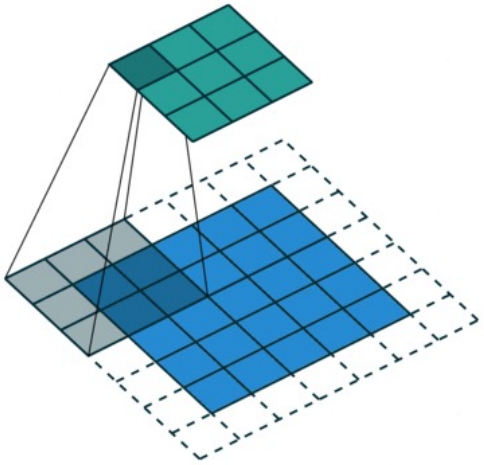


7 x 7 Input Volume

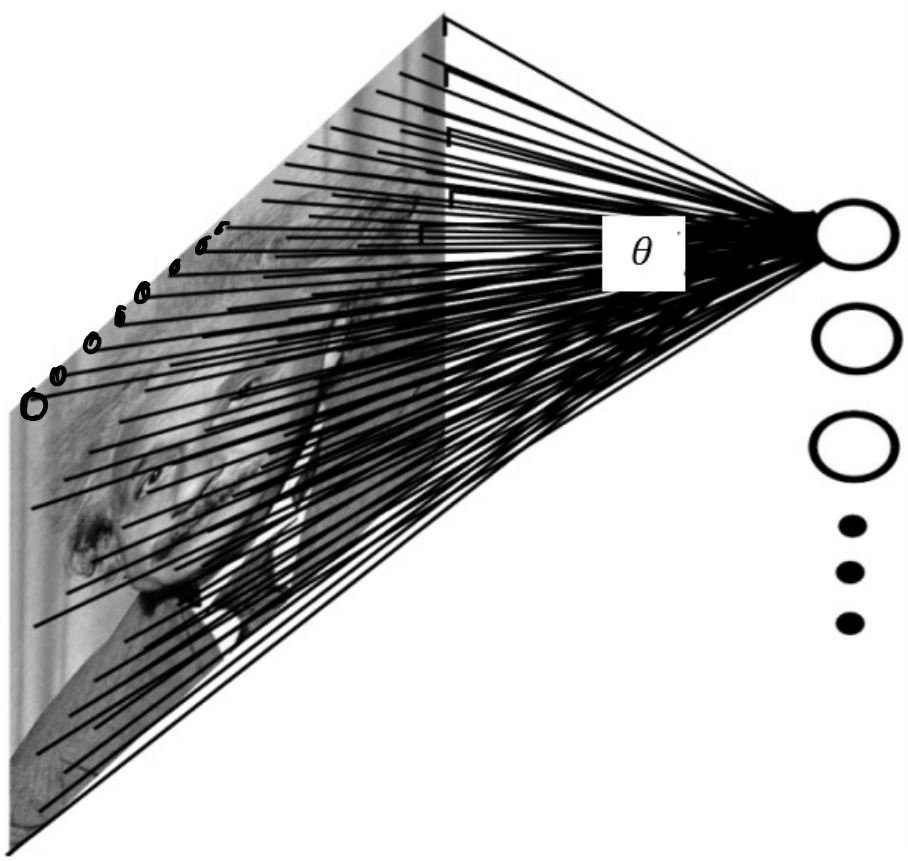


3 x 3 Output Volume





Convolutional Neural Networks
Convolutions

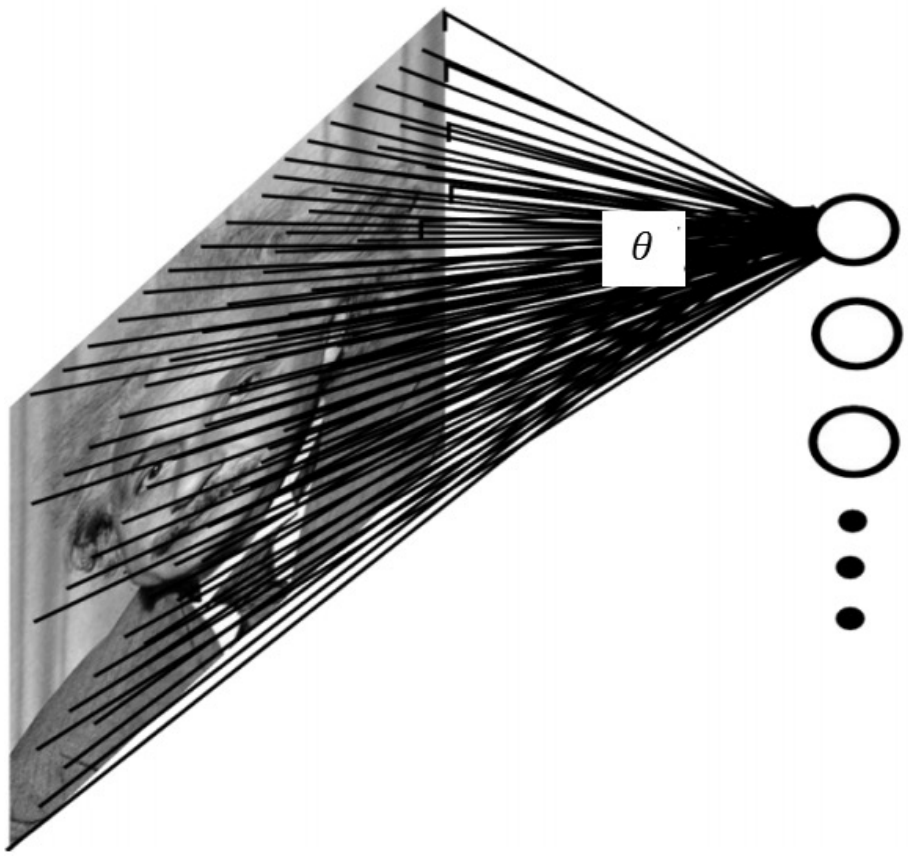


In classic Neural Network every node is fully connected to all the adjacent layer.

Example:

- 1000x1000 image
- 1M parameters per neuron

Convolutional Neural Networks
Convolutions



In classic Neural Network every node is fully connected to all the adjacent layer.

Example:

1000x1000 image

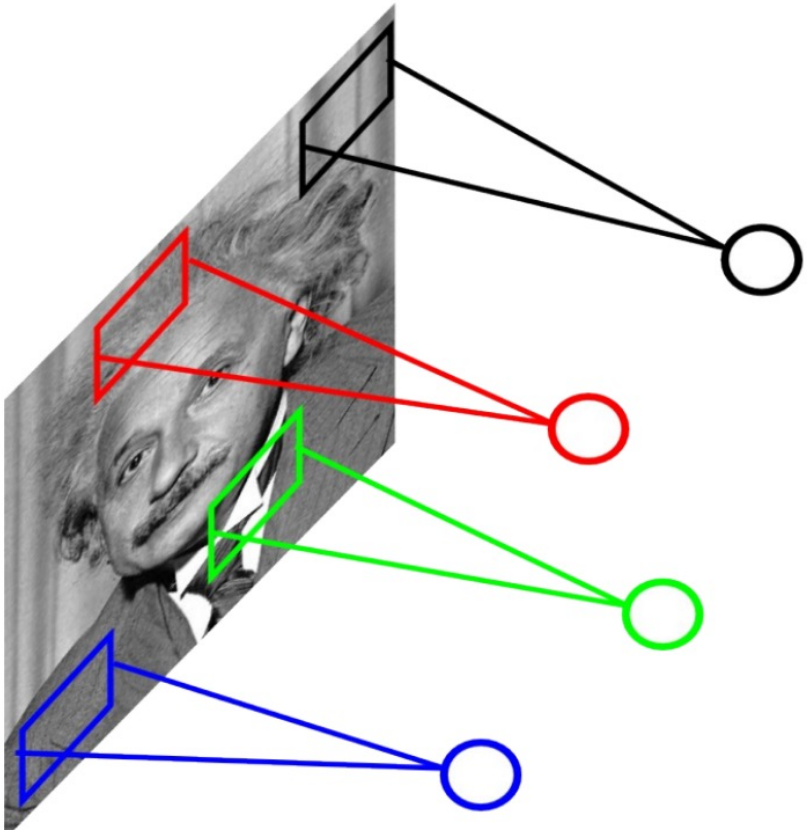
1M parameters per neuron

CNN Solution:

- Local receptive field
- Shared weights
- Pooling Layers (to reduce dimensionality going deeper)

Convolutional Neural Networks

Local Receptive Field

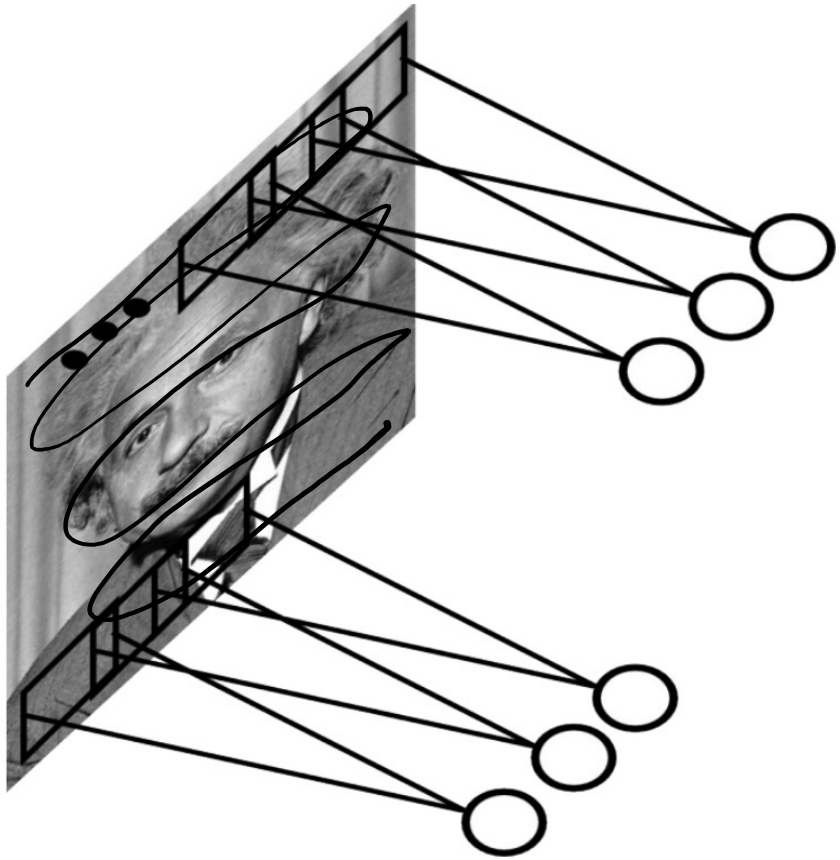


Local receptive field: every neuron of a hidden layer is fully connected only with a region of the input (called receptive field). Every connession is used to learn a weigh (i.e., a filter for every node)

E.g. with a receptive field 5x5 the every neuron has 25 connessions.

Convolutional Neural Networks

Shared Weights

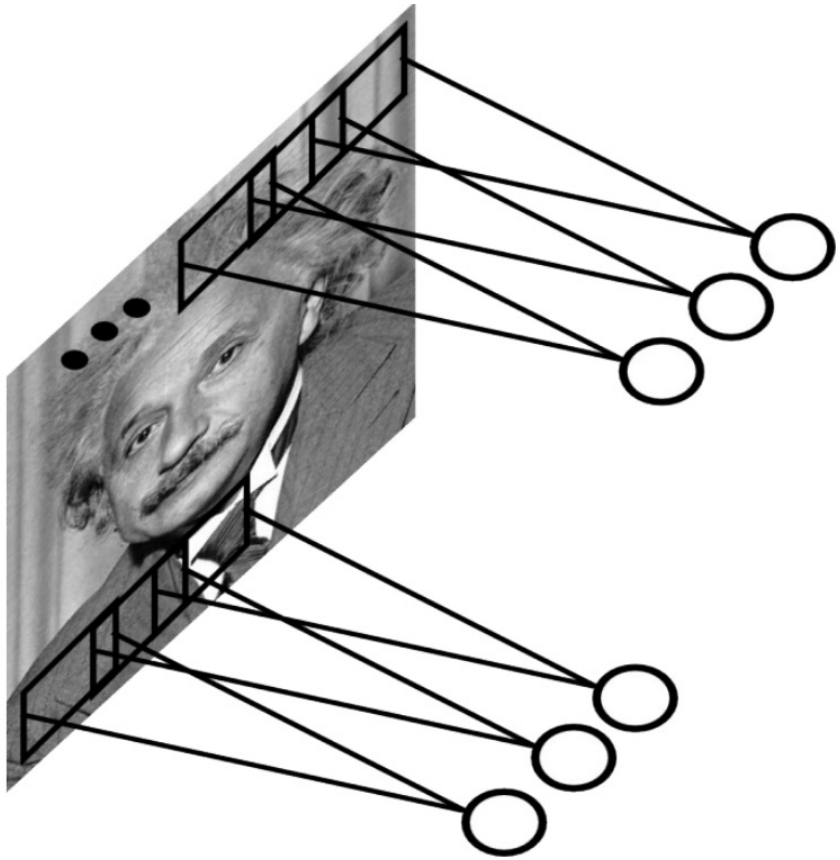


Shared weights: since features could be anywhere in the image, neurons of the same layer share weights.

This means that all the neurons of a layer are sensible to the same feature (filter response), which is computed in different spatial position of the input.

Convolutional Neural Networks

Shared Weights



Shared weights: since features could be anywhere in the image, neurons of the same layer share weights.

This means that all the neurons of a layer are sensible to the same feature (filter response), which is computed in different spatial position of the input.

→ Map of weights applied to different positions → **convolution**

Size of Output of Convolutional Layer

- Input size is WxH
- Kernel size is F_wxF_h
- Pading size is P
- Strides with respect width is S_w
- Strides with respect height is S_h

$$\text{output width} = \frac{W - F_w + 2P}{S_w} + 1$$

$$\text{output height} = \frac{H - F_h + 2P}{S_h} + 1$$

IF THE NUMBER IS NOT AN INTEGER, THEN VALUE OF STRIDE IS INCORRECT

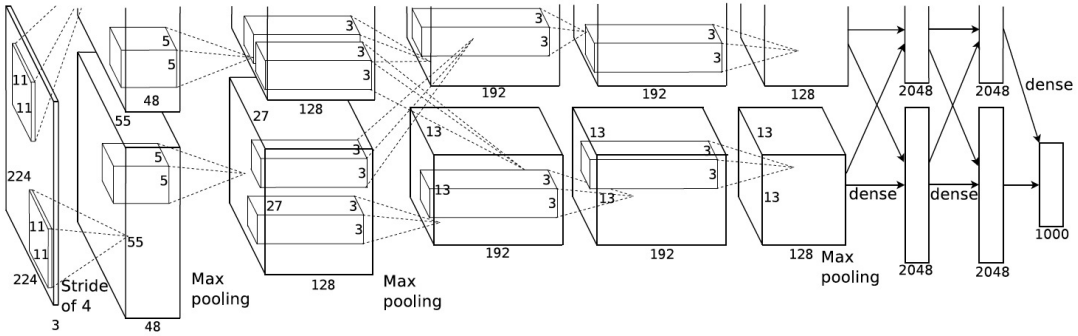
$W=10 \quad P=0 \quad F=3$

$$\frac{W-F+2P}{S}+1 = \frac{10-3+0}{S}+1 = \frac{7}{S}+1$$

$\Rightarrow S$ CANNOT BE AN INTEGER OTHER THAN 7 OR 1.

ZERO PADDING IS YOUR FRIEND HERE

Real Example – AlexNet



IMAGES : 224x224x3

H=W=227

F=11

S=4

P=0

$\frac{224-11+1}{4} = 55$ SIZE OF OUTPUT MAP

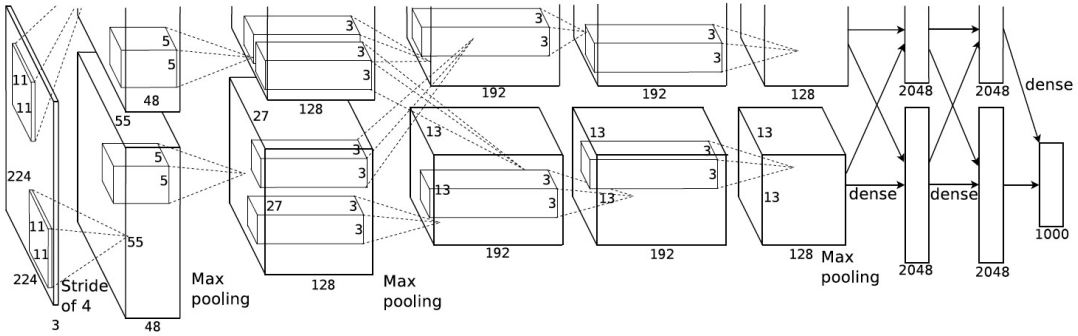
1° CONVOLUTIONAL LAYER 96 NEURONS 11x11x3 ⇒ 34848 PARAMETERS

+ 96 bias parameters

OUTPUT UNITS 55x55x96 = 290400 AT THE FIRST LAYER

1000 classes, input 224x224x3,
>60Million of parameters in total.

AlexNet Architecture



Layer	Units	Weights	Connections
L_1 (Conv)	290,400	34,848	105,415,200
L_2 (Conv)	186,624	307,200	111,974,400
L_3 (Conv)	64,896	884,736	149,520,384
L_4 (Conv)	64,869	663,552	112,140,288
L_5 (Conv)	43,264	442,368	74,760,192
L_6 (Dense)	4096	37,748,736	37,748,736
L_7 (Dense)	4096	16,777,216	16,777,216
L_8 (Dense)	1000	4,096,000	4,096,000
Conv Subtotal	650,080	2,332,704	553,810,464
Dense Subtotal	9192	58,621,952	58,621,952
Total	659,272	60,954,656	612,432,416

Convolutions: sparse connections

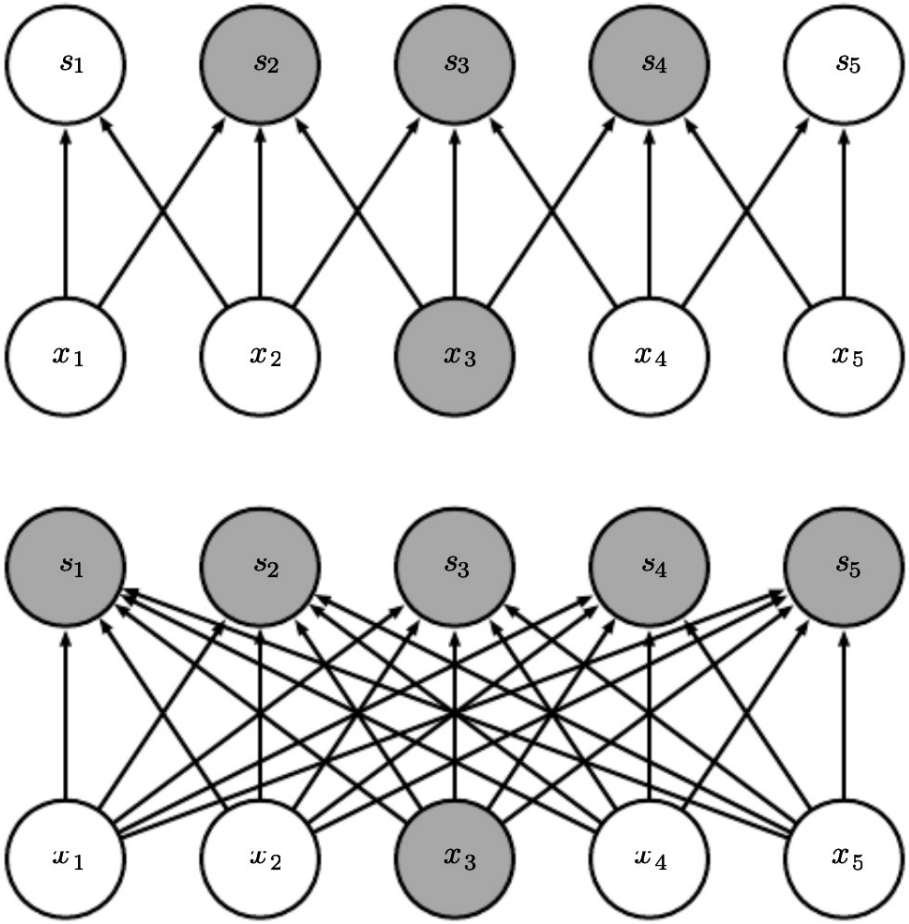


Figure 9.2: Sparse connectivity, viewed from below. We highlight one input unit, x_3, and highlight the output units in $\boldsymbol{s}$ that are affected by this unit. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three outputs are affected by $\boldsymbol{x}$. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the outputs are affected by x_3.

Chapter 9 – Deep Learning (www.deeplearningbook.org)

Convolutions: sparse connections

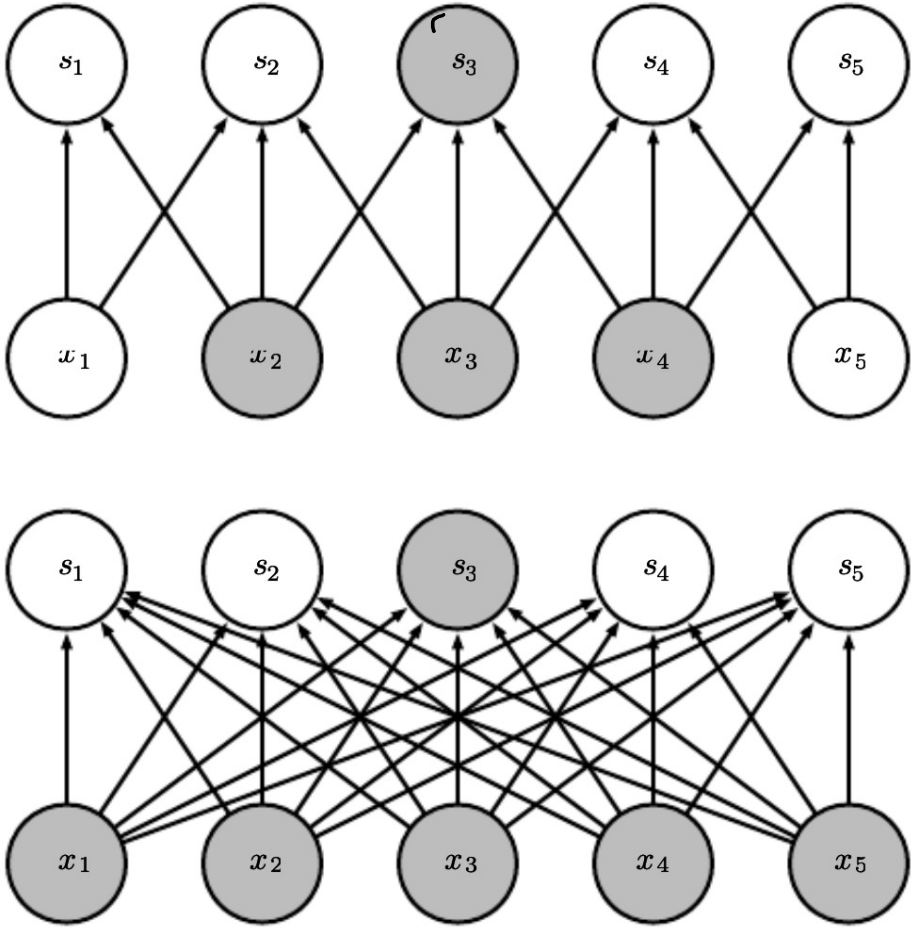


Figure 9.3: Sparse connectivity, viewed from above. We highlight one output unit, s_3, and highlight the input units in $\boldsymbol{x}$ that affect this unit. These units are known as the **receptive field** of s_3. *(Top)*When $\boldsymbol{s}$ is formed by convolution with a kernel of width 3, only three inputs affect s_3. *(Bottom)*When $\boldsymbol{s}$ is formed by matrix multiplication, connectivity is no longer sparse, so all the inputs affect s_3.

Chapter 9 – Deep Learning (www.deeplearningbook.org)

Receptive Field

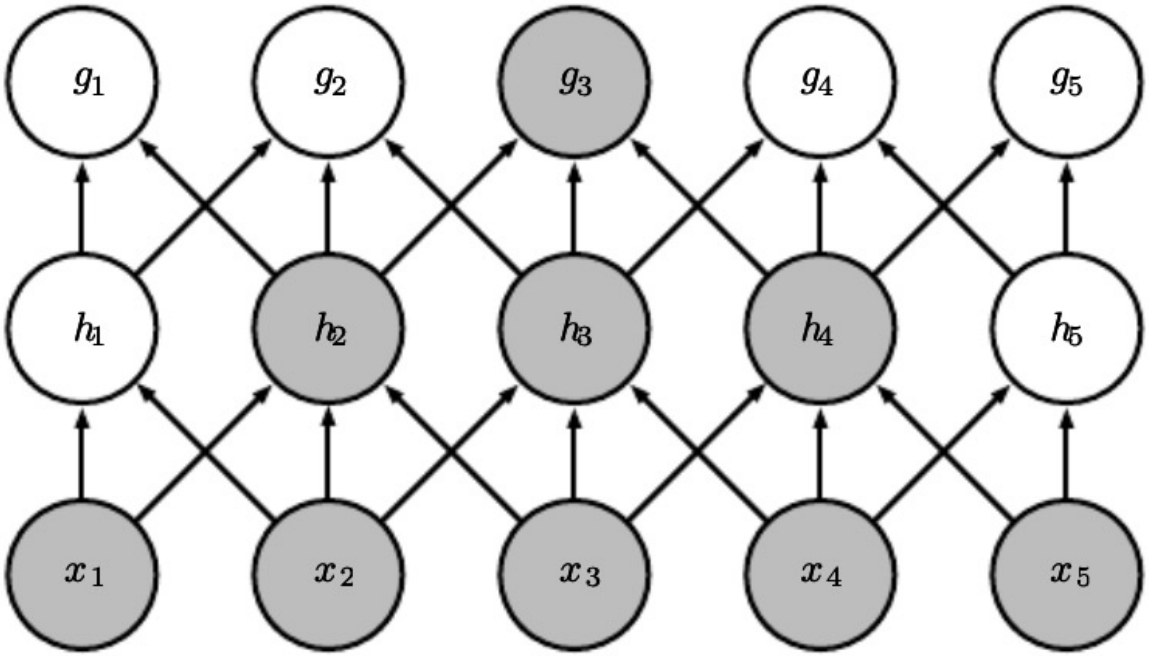


Figure 9.4: The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This effect increases if the network includes architectural features like strided convolution (figure 9.12) or pooling (section 9.3). This means that even though *direct* connections in a convolutional net are very sparse, units in the deeper layers can be *indirectly* connected to all or most of the input image.

Chapter 9 – Deep Learning (www.deeplearningbook.org)

Convolutions: parameter sharing

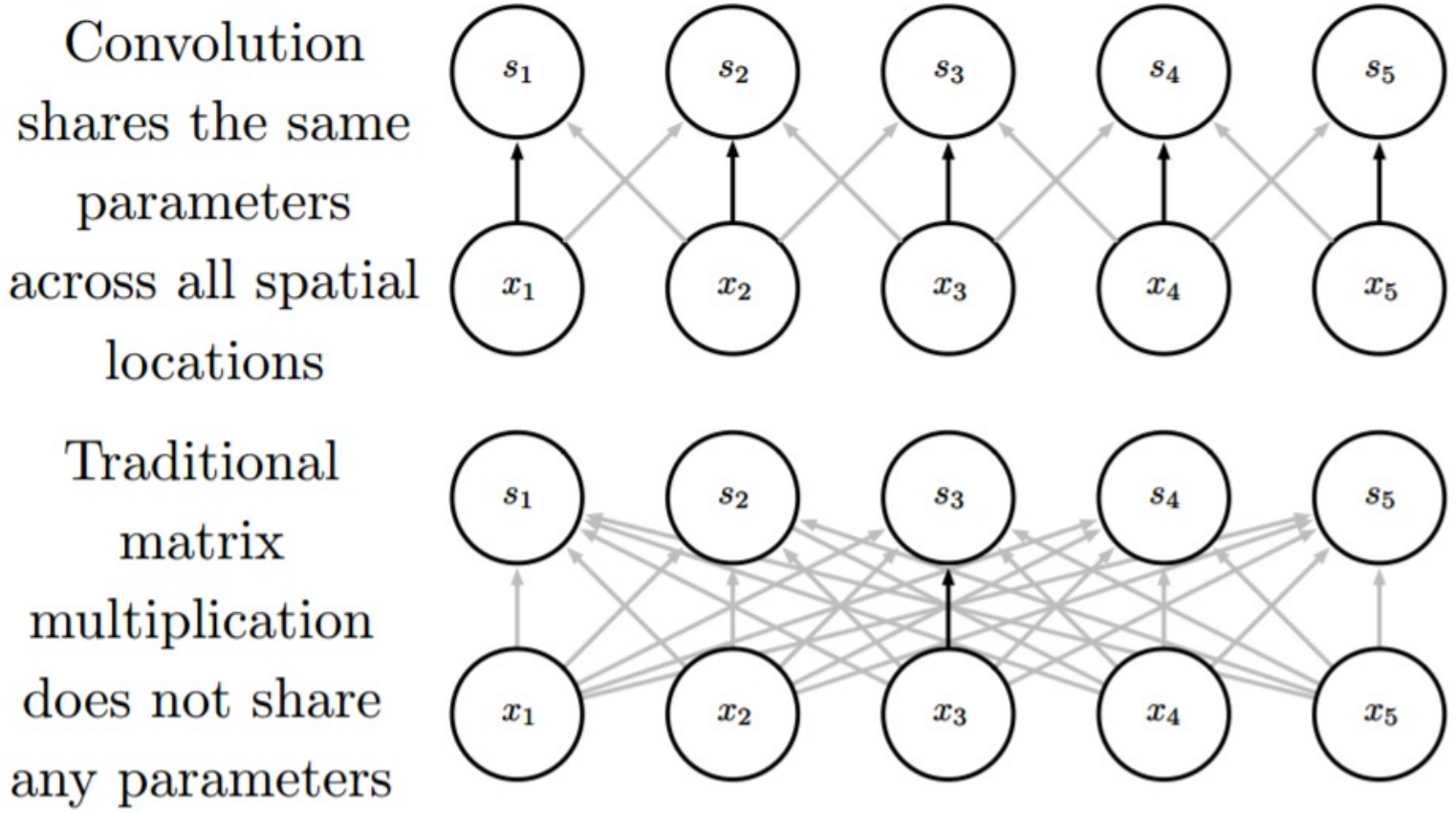


Figure 9.5: Parameter sharing. Black arrows indicate the connections that use a particular parameter in two different models. *(Top)*The black arrows indicate uses of the central element of a 3-element kernel in a convolutional model. Because of parameter sharing, this single parameter is used at all input locations. *(Bottom)*The single black arrow indicates the use of the central element of the weight matrix in a fully connected model. This model has no parameter sharing, so the parameter is used only once.

Chapter 9 – Deep Learning (www.deeplearningbook.org)

Effects of convolutions

Using convolutions has the following effects:

- Makes the number of parameters in a given layer independent from input size;
- Reduces the number of parameters in a given layer;
- Makes feature extraction invariant to x and y translation (in the convolutional network).

-> Since we have less parameters, we can build very deep models (e.g., 150 layers).

Efficient convolution

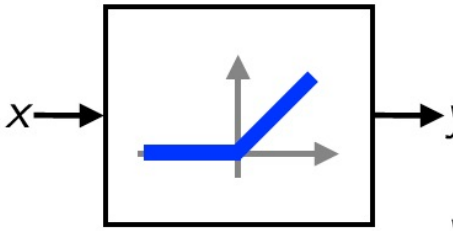
- Convolution in time-domain means multiplication in frequency domain
  - For some problem sizes, converting the signals to the frequency domain, performing multiplication in the frequency domain and transforming them back is more efficient
- A kernel is called separable if it can be written as a product of two vectors:

$$\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1 & 2 & 1\end{bmatrix} = \frac{1}{16}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}$$

  - Separable kernels can be stored with less memory if you store just the vectors
  - Moreover, processing using the vectors is faster than using naïve convolution

Convolutional Neural Networks

Activation Functions – Non Linearity



$$y = \frac{1}{1+e^{-x}}$$ sigmoid

$$y = \tanh(x)$$ hyperb. tan

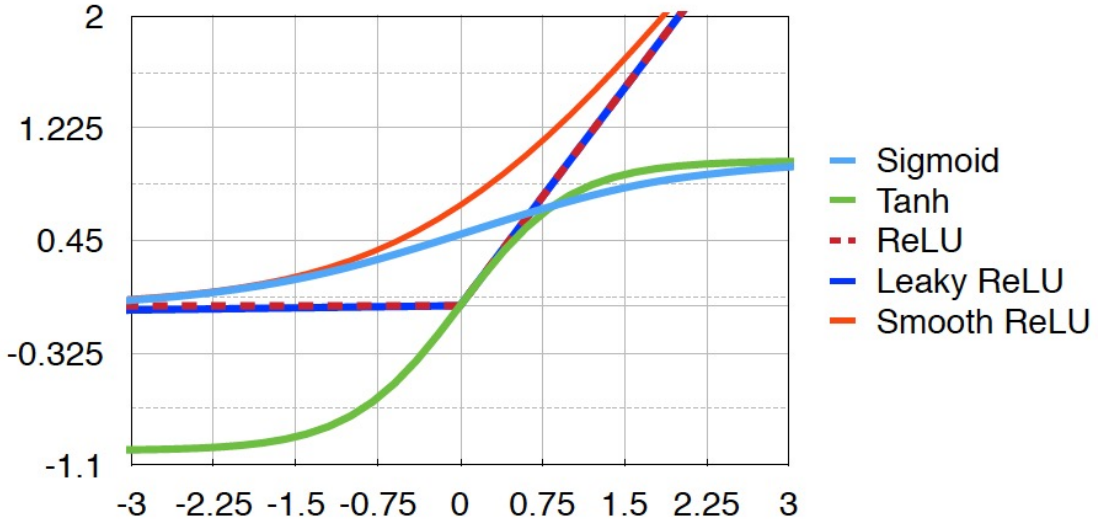
$$y = \max\{0, x\}$$ ReLU

$$y = \log(1+e^{x})$$ Soft ReLU

$$y = \epsilon x + (1-\epsilon)\max\{0, x\}$$ Leaky ReLU

Combination of linear function is a linear function. We need non linear functions on top of each convolutions.

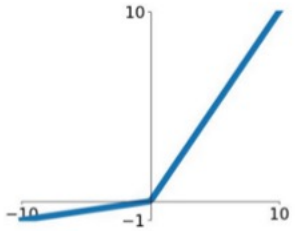
- ReLU and its variants
  - The common choice
  - Faster
  - Easier (in backpropagation etc.)
  - Avoids saturation issues

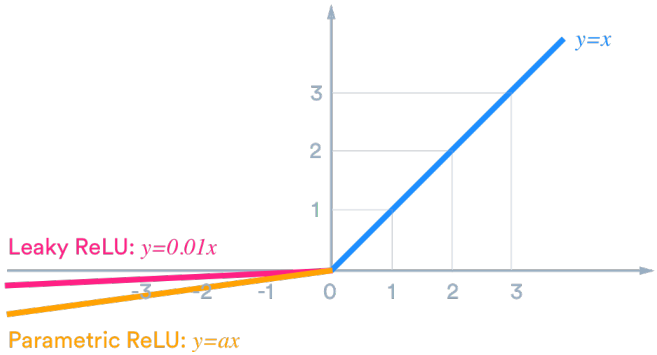


Considering data Centered to 0

Leaky ReLU

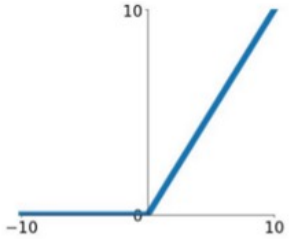
$$\max(0.1x, x)$$





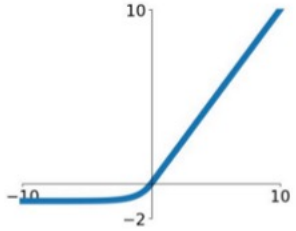
ReLU

$$\max(0, x)$$



ELU

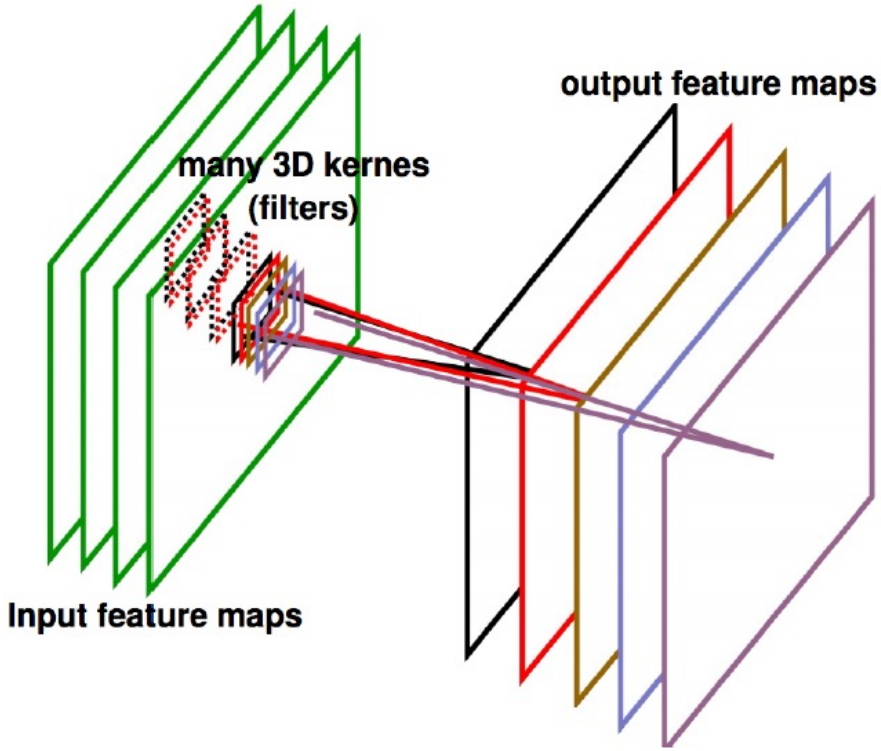
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \leq 0 \end{cases}, \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases}. \qquad (15)$$

Convolutional Neural Networks

Filters and Feature Maps – Convolutional Layer
(i.e., linear transformation + activation)



Every filter captures a specific feature of the precedent layer.

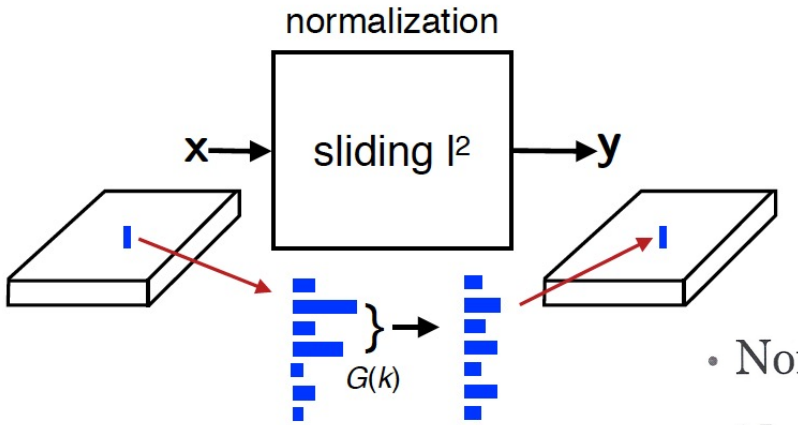
The CNN learns many filters per layer.

The output of each filter after the activation function is called ***feature map***.

Convolutional Neural Networks

Local Feature Normalization – Normalization Layer

Across feature channels rather than spatially



$$y_{ijk} = x_{ijk} \left(\kappa + \alpha \sum_{q \in G(k)} x_{ijq}^2 \right)^{-\beta}$$

- Normalize the responses of neurons
- Necessary especially when the activations are not bounded (e.g., a ReLU unit)

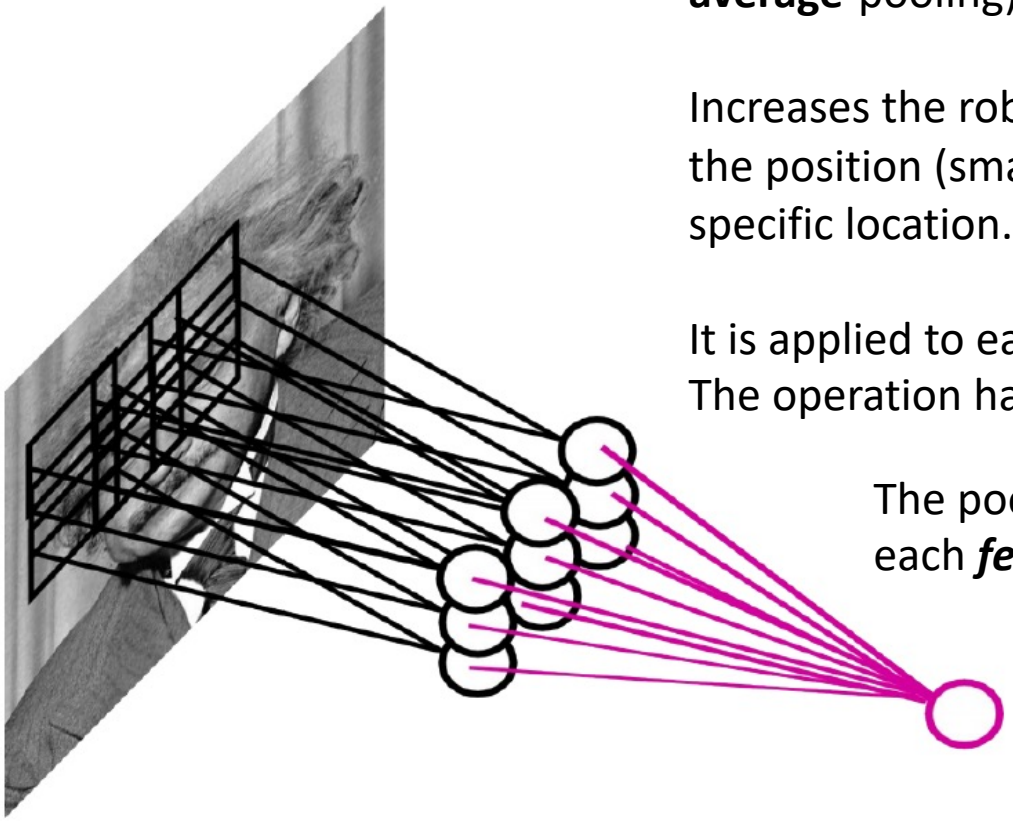
Operates at each spatial location independently

Normalise groups G(k) of feature channels

Groups are usually defined in a **sliding window manner**

Convolutional Neural Networks

Spatial Pooling Layer



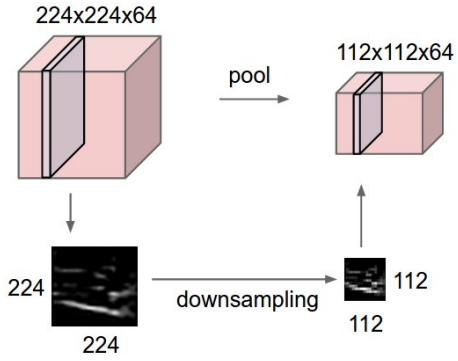
CNN achitecture has also **pooling layers**.

A poooling **layer takes regions of a feature map and select a unique representative value** (**max**-pooling, **average**-pooling).

Increases the robustness of the features with respect to the position (small shift). Reduce dependence of a specific location. Reduce computational complexity.

It is applied to each channel indipendendly.
The operation has to be differentiable (es. max pooling)

The pooling can be done by spatial sampling of each ***feature map***.

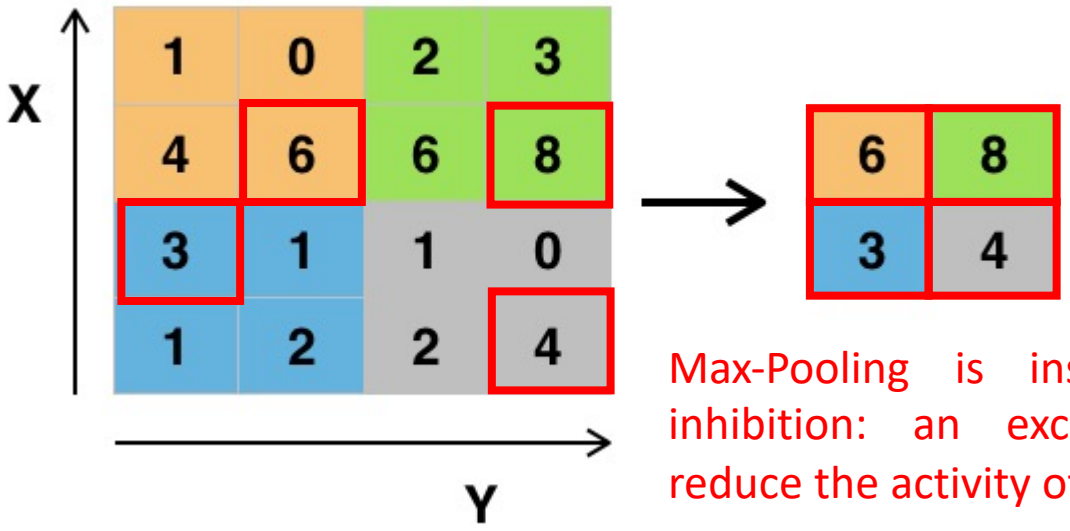


Max-Pooling

We would like our convolutional networks to be invariant to small shifts in the x and y axes (e.g., the image is not perfectly centered around the object).

To achieve this, a max pooling operation is usually employed. In its simplest form, max pooling subsamples the input feature map selecting local maxima.

Output depth dimension remain the same.

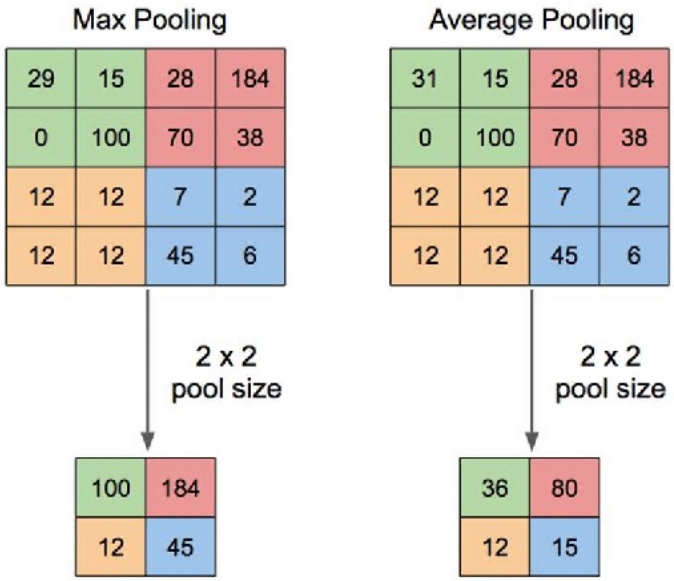


Pooling can be overlapping or non-overlapping

Es. non-overlapping: max pooling with 2x2 filter and stride 2

Max-Pooling is inspired by lateral inhibition: an excited neuron can reduce the activity of its neighbours.

Max Pooling vs Avg Pooling



Pooling layers compress features extracted.

Max pooling: if the block have mostly small activations, but some large activations, with max pooling we loose the information of small activations.

Avg pooling: in previous case, we retain some information about low activation. But if positive large activations are balanced by negative activations the output looks no active due AVG.

Pooling over different convolutions – Invariance to different transformation

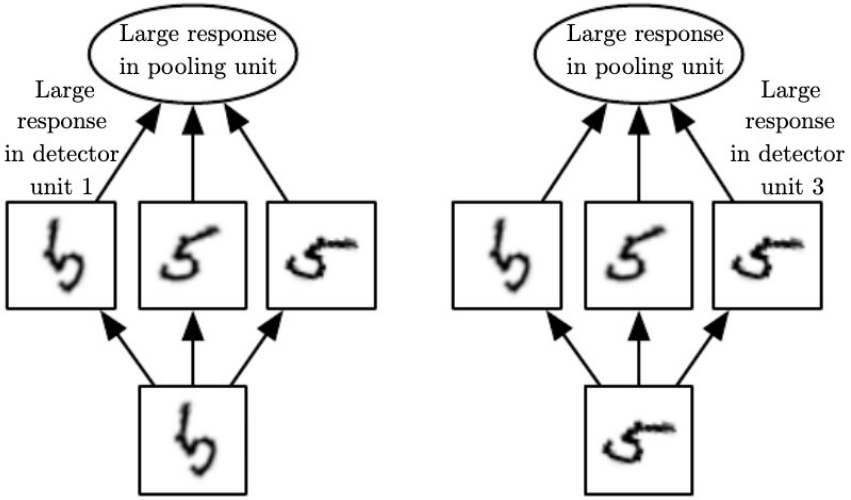


Figure 9.9: Example of learned invariances. A pooling unit that pools over multiple features that are learned with separate parameters can learn to be invariant to transformations of the input. Here we show how a set of three learned filters and a max pooling unit can learn to become invariant to rotation. All three filters are intended to detect a hand written 5. Each filter attempts to match a slightly different orientation of the 5. When a 5 appears in the input, the corresponding filter will match it and cause a large activation in a detector unit. The max pooling unit then has a large activation regardless of which detector unit was activated. We show here how the network processes two different inputs, resulting in two different detector units being activated. The effect on the pooling unit is roughly the same either way. This principle is leveraged by maxout networks (Goodfellow *et al.*, 2013a) and other convolutional networks. Max pooling over spatial positions is naturally invariant to translation; this multichannel approach is only necessary for learning other transformations.

Pooling can downsample

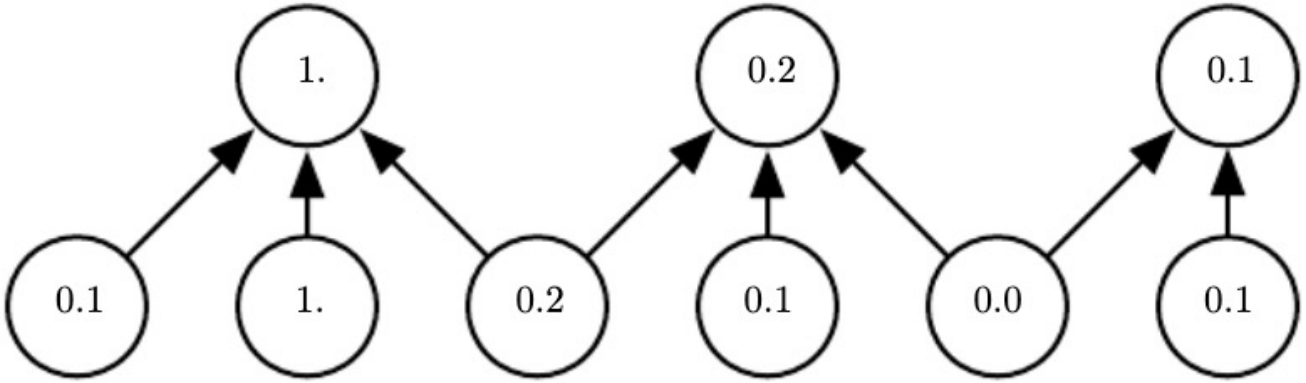


Figure 9.10: Pooling with downsampling. Here we use max pooling with a pool width of three and a stride between pools of two. This reduces the representation size by a factor of two, which reduces the computational and statistical burden on the next layer. Note that the rightmost pooling region has a smaller size but must be included if we do not want to ignore some of the detector units.

For many tasks, pooling is essential for handling inputs of varying size. For example, if we want to classify images of variable size, the input to the classification layer must have a fixed size. This is usually accomplished by varying the size of an offset between pooling regions so that the classification layer always receives the same number of summary statistics regardless of the input size. For example, the final pooling layer of the network may be defined to output four sets of summary statistics, one for each quadrant of an image, regardless of the image size.

Backpropagation for pooling

- E.g., for a max operation:
- Remember the derivative of $max(x, y)$:
  - $f(x, y) = \max(x, y)$
  - $\nabla f = [1(x \geq y),\ 1(y \geq x)]$
- Interpretation: only routing the gradient to the input that had the biggest value in the forward pass.
- This requires that we save the index of the max activation (sometimes also called *the switches*) so that gradient "routing" is handled efficiently during backpropagation.

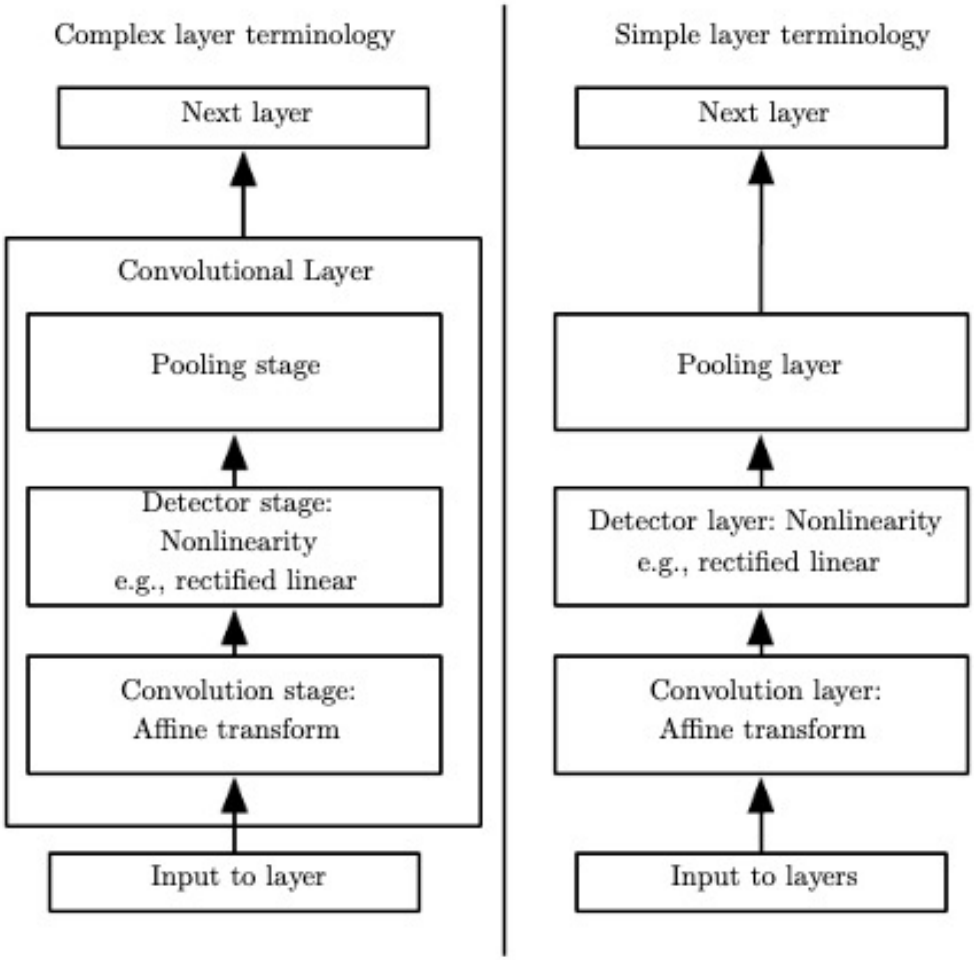
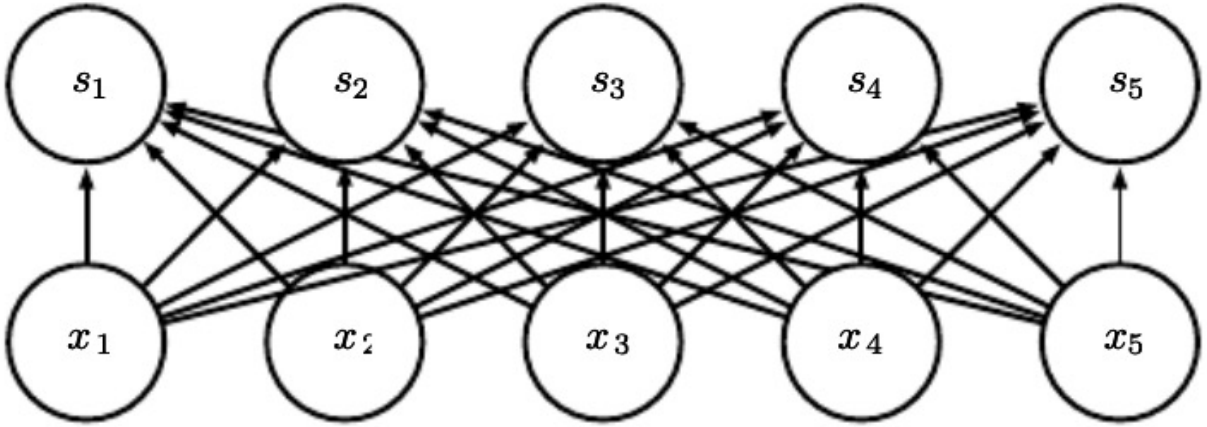


Figure 9.7: The components of a typical convolutional neural network layer. There are two commonly used sets of terminology for describing these layers. *(Left)*In this terminology, the convolutional net is viewed as a small number of relatively complex layers, with each layer having many "stages." In this terminology, there is a one-to-one mapping between kernel tensors and network layers. In this book we generally use this terminology. *(Right)*In this terminology, the convolutional net is viewed as a larger number of simple layers; every step of processing is regarded as a layer in its own right. This means that not every "layer" has parameters.

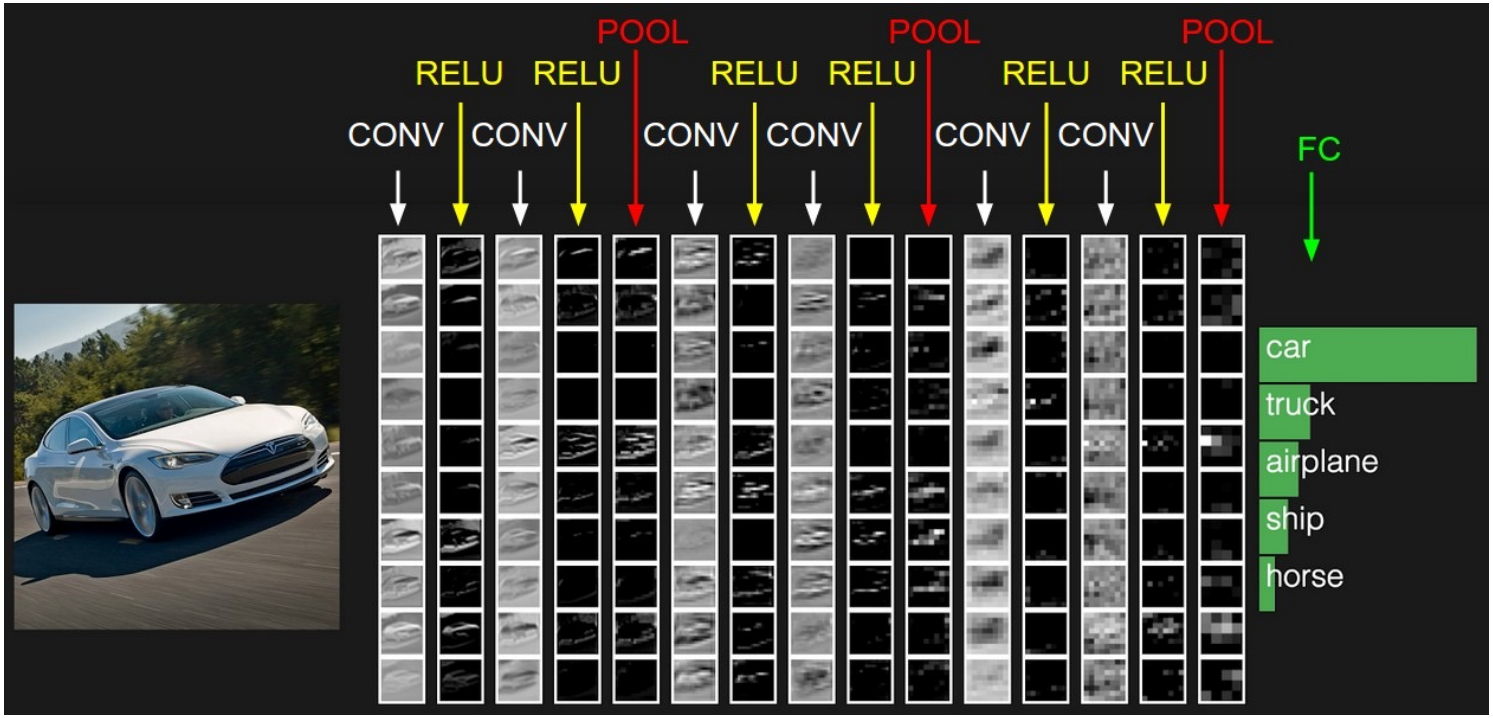
Fully-Connected Layer (FC Layers)

- At the top of the network for mapping the feature responses to output labels of the classes (i.e., the input to the last layer is fully connected)
- Can be many FC layers



Convolutional Neural Networks

Fully Connected Layer



The last layer is **fully connected (FC)** and the number of neurons are equal to the number of classes. The output of the last layer is the probablility score with respect to the classes.

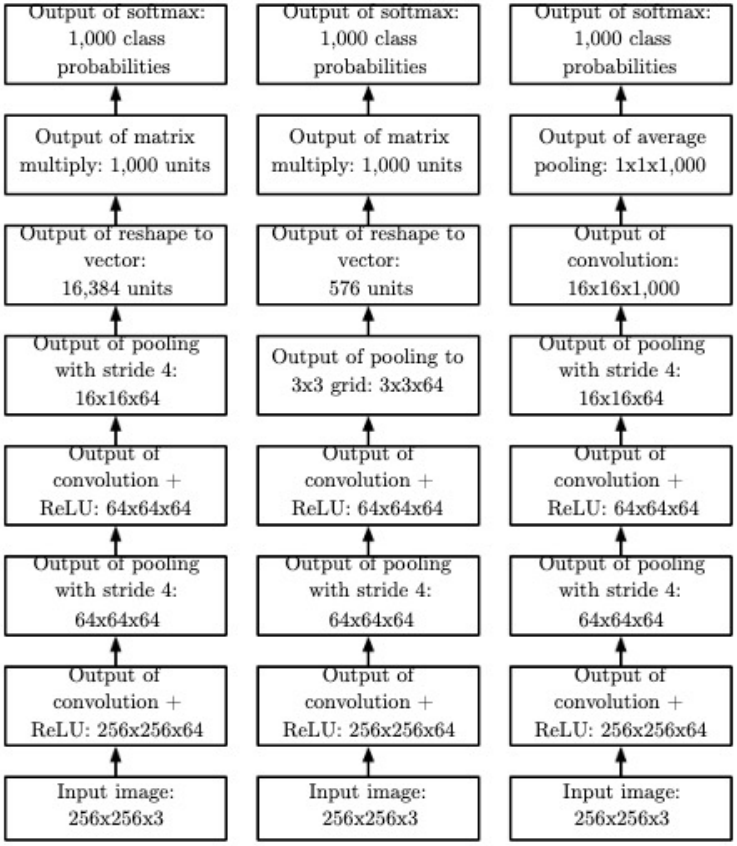


Figure 9.11: Examples of architectures for classification with convolutional networks. The specific strides and depths used in this figure are not advisable for real use; they are designed to be very shallow to fit onto the page. Real convolutional networks also often involve significant amounts of branching, unlike the chain structures used here for simplicity. *(Left)*A convolutional network that processes a fixed image size. After alternating between convolution and pooling for a few layers, the tensor for the convolutional feature map is reshaped to flatten out the spatial dimensions. The rest of the network is an ordinary feedforward network classifier, as described in chapter 6. *(Center)*A convolutional network that processes a variably sized image but still maintains a fully connected section. This network uses a pooling operation with variably sized pools but a fixed number of pools, in order to provide a fixed-size vector of 576 units to the fully connected portion of the network. *(Right)*A convolutional network that does not have any fully connected weight layer. Instead, the last convolutional layer outputs one feature map per class. The model presumably learns a map of how likely each class is to occur at each spatial location. Averaging a feature map down to a single value provides the argument to the softmax classifier at the top.

Examples of CNN in 3D

- Demo