Introduction to Convolutional Neural Networks

Table of contents

Introduction – 1

- Computer vision is a branch within computer science that enables computers to recognize the visual world
- Several machine learning and deep learning–based algorithms are available that help with building models to make predictions on images or videos
- Convolutional Neural Networks (CNNs) has shown excellent performance in this field — where they are the *de–facto* standard — and in many other machine learning tasks

Introduction – 2

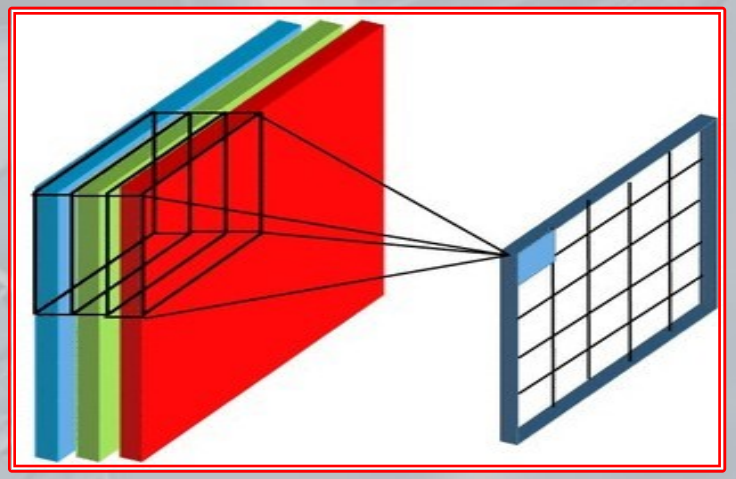
- The CNN connectivity model resembles the organization of the animal visual cortex: individual cortical neurons respond to stimuli only in a narrow region of the visual field, known as the receptive field
- The receptive fields of several neurons partially overlap such that they cover the entire visual field

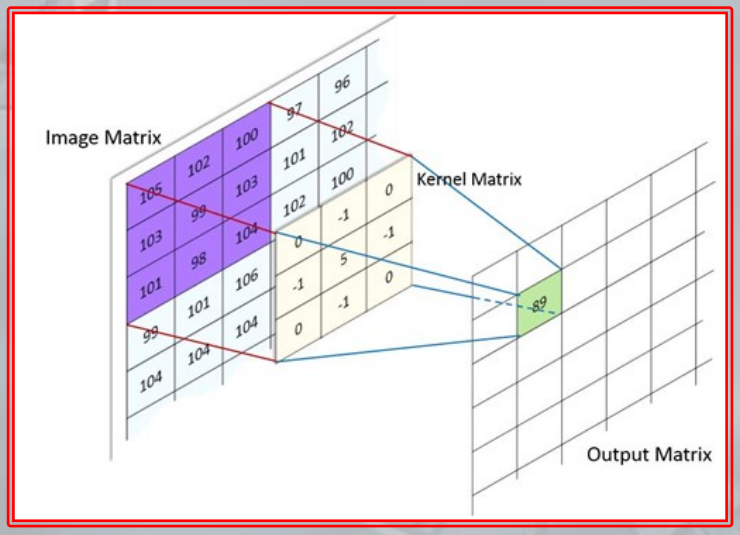
CNNs in a nutshell

- Convolutions: Especially useful for networks that analyze images or sequences
- Pooling and/or convolutions with stride: to reduce the size of the input
- ReLU or derived activation functions: to increase computational speed and alleviate the vanishing gradient problem
- Data augmentation: to artificially increase the number of training data and to reduce overfitting
- GPUs: to speed up training

Layer description: Convolutional layers – 1

- The input has a dimension $w \times h \times d$ and is transformed into an output of a size equal to $w \times h \times d'$, at most, where d' is the number of feature maps (or the number of kernels)
- Each feature map is the result of the convolution of the input with a 3D tensor (kernel) of dimensions $u \times v \times d$ ($u \ll w$ and $v \ll h$)
- The parameters are only $(u \times v \times d) \times d'$, (instead of $(w \times h \times d) \times (w \times h \times d')$, corresponding to a fully connected layer)

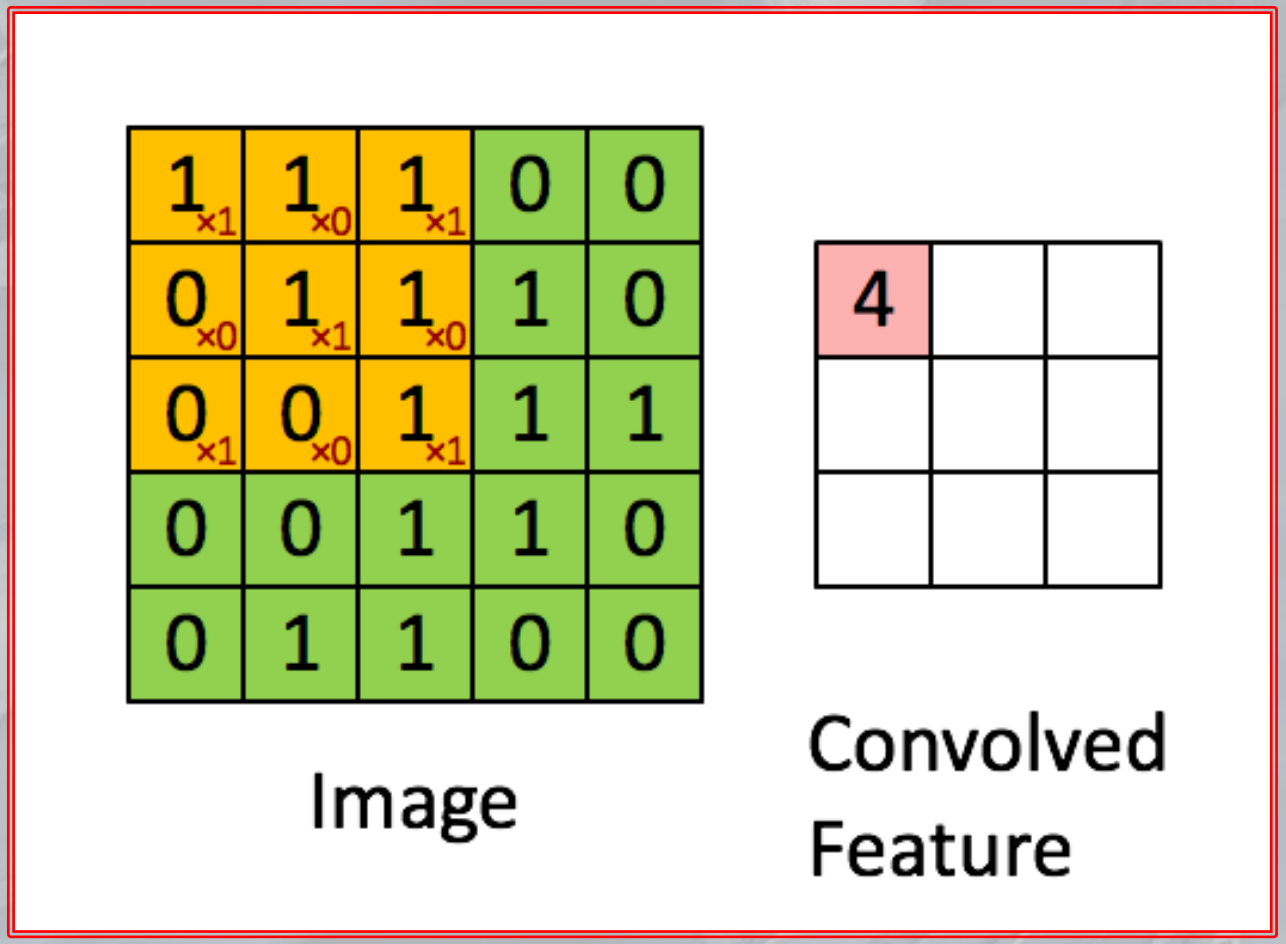




Layer description: Convolutional layers – 2

Kernel

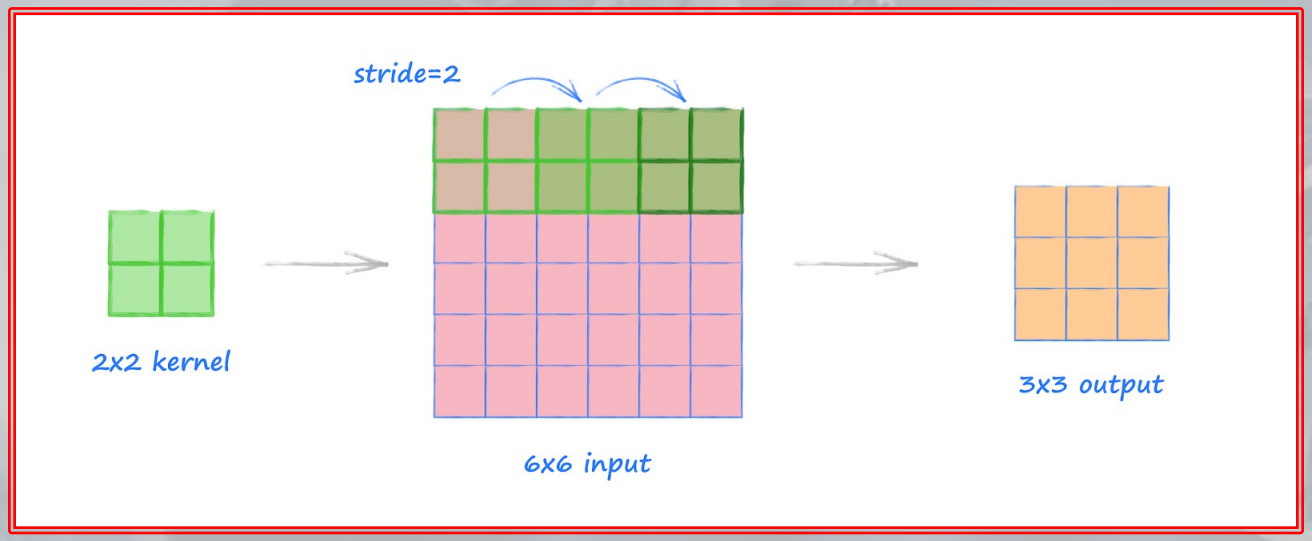
1	0	1
0	1	0
1	0	1



- Considering $d=1$, in a fully–connected layer, 10,000 weights would be required for processing an image of 100×100 pixels; however, with a 5×5 kernel, only 25 weights are required to process 5×5 tiles

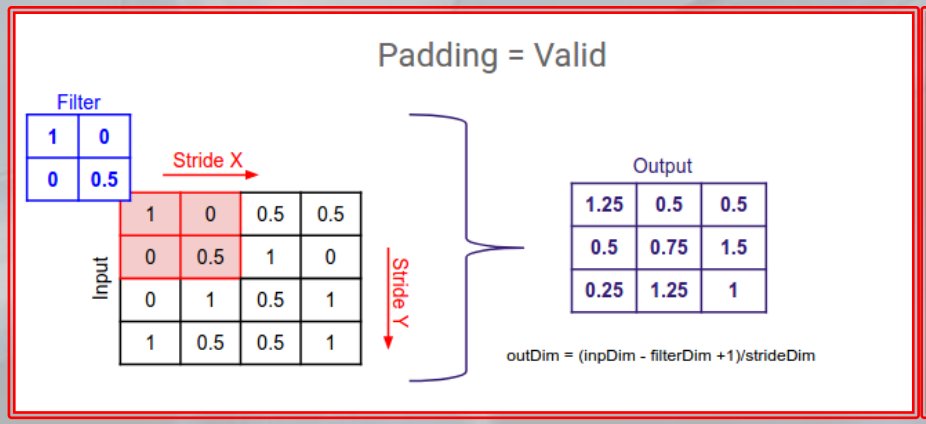
Stride in convolutional layers

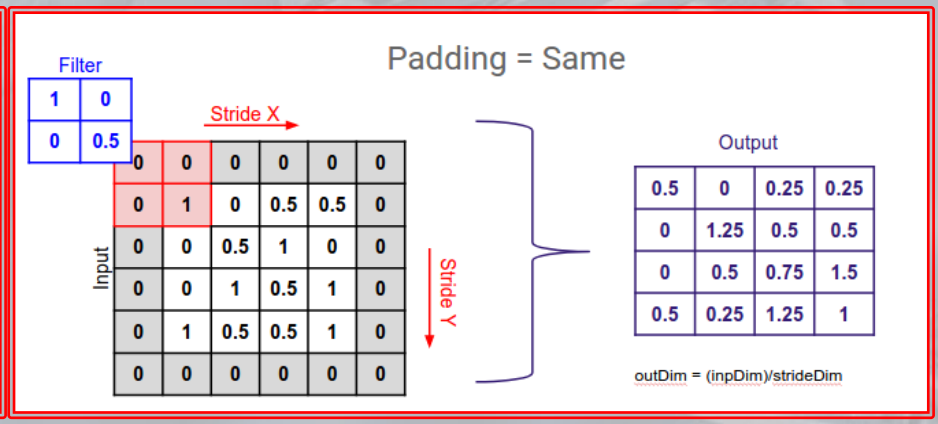
- The stride of a convolutional layer specifies how much the kernel is moved at each step
  - Normally, a unitary stride is used
  - A larger value can be employed to have less overlap between the receptive fields
  - In this case, the resulting feature map will be smaller than the input, since we are "skipping" some positions



Padding in convolutional layers

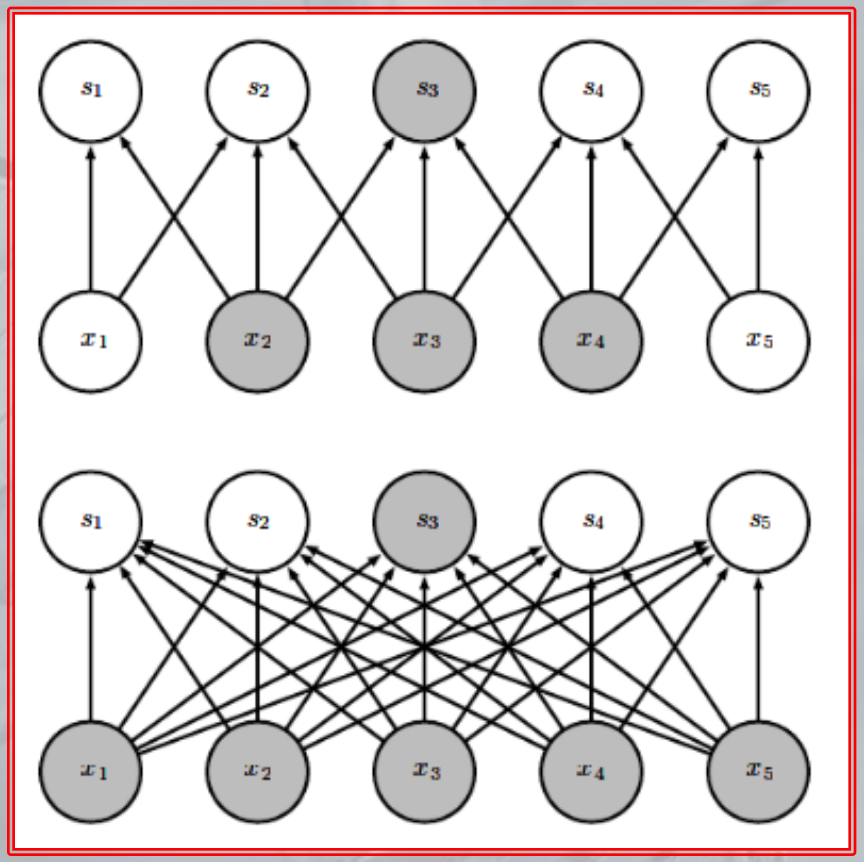
- How to manage the calculation of the elements on the border of the image? If the kernel is centered on a border element, part of it falls out of the image
- Two possible solutions:
  - Padding Valid – Kernels are applied only in "valid" locations; the output produced will have a smaller size than the input
  - Padding Same – Extra values (often zeros) are added to the outside in order to perform the calculation also on the border; in doing so, the dimensions of the image (with unitary stride) remain the same





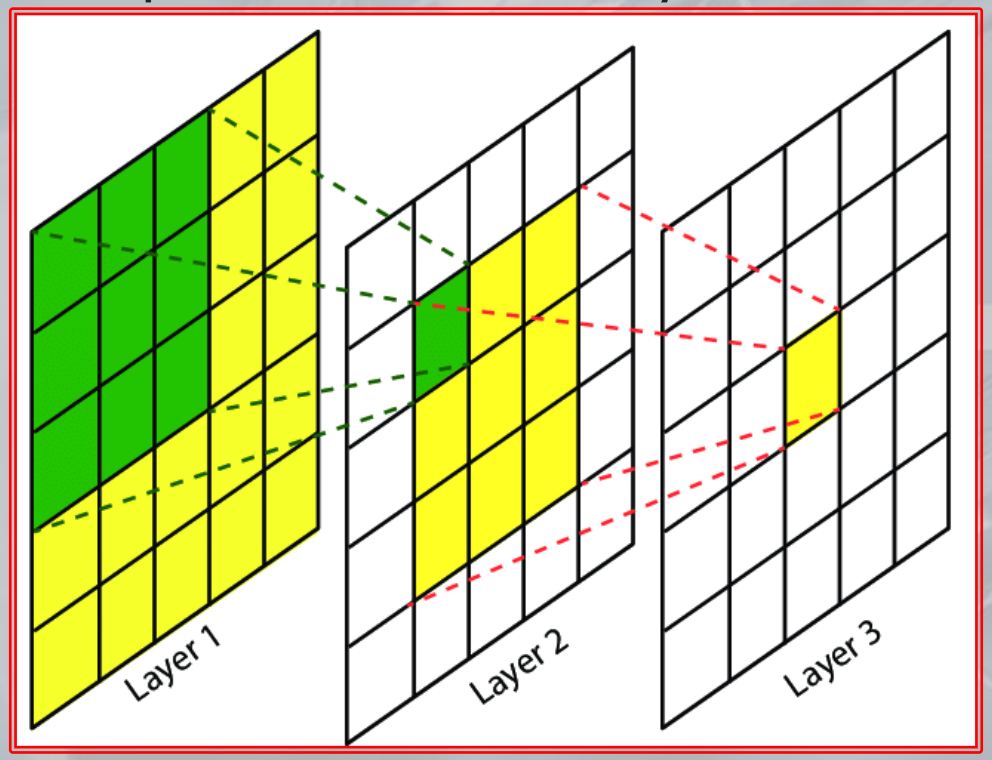
Sparse interactions in convolutional layers

- In convolutional layers, fewer units contribute to the input of the neurons of the next layer
  - Receptive field
- Example
  - When using convolutions with a size 3 kernel, only three inputs affect $S3$ (top)
  - When using a fully connected layer all inputs affect $S3$ (bottom)



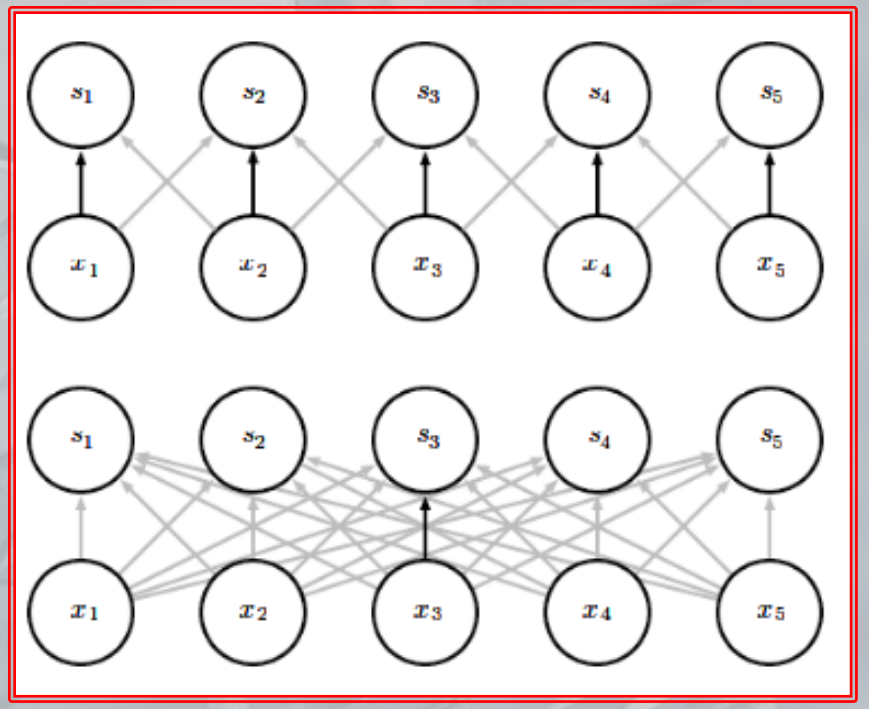
Receptive field in convolutional layers

- The application of multiple convolutional layers allows to broaden the receptive field
- Layer by layer, all inputs contribute to the calculation of the elements of a given feature map
- Higher–layer features are extracted from wider context windows, compared to lower–layer features



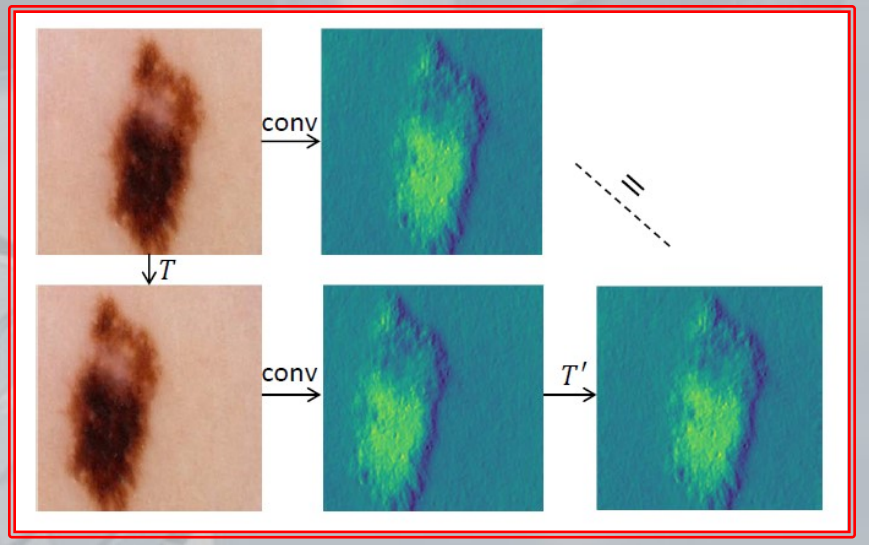
Weight sharing in convolutional layers

- In convolutional layers, weight sharing is used as a "trick" to reduce the number of parameters
  - Each kernel weight is used for each input element
- Example
  - Convolutional model with a kernel with size 3: a single parameter is used many times (top)
  - Fully connected model: no sharing of parameters; each parameter is used only once (bottom)



Equivariant representation in convolutional layers

- Weight sharing guarantees a model which is equivariant to translations
- The kernel "learns to extract" particular features of the image, regardless of the position considered within it



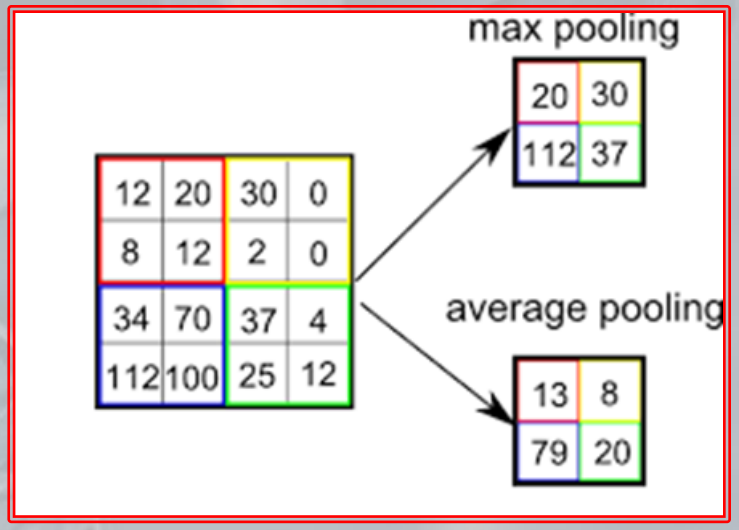
- Informally, a function is equivariant when applying a transformation and then computing the function produces the same result as computing the function and then applying the transformation
- Specifically, a function f is equivariant to a function g if

$$f(g(x)) = g(f(x))$$

Definition – A function is said to be an equivariant map when its domain and codomain are acted on by the same symmetry group, and when the function commutes with the action of the group

Pooling

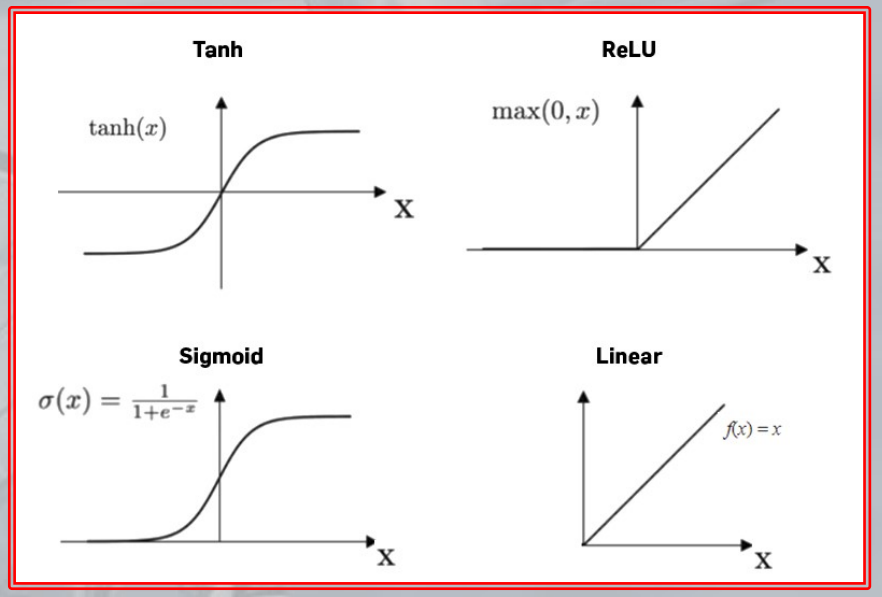
- Often used after a series of convolutional layers to reduce the size of feature maps
- Pooling with dimension $n \times n$ and with stride m: place a window $n \times n$ every m points



- The size of the feature map is reduced by a factor of m
- The most common pooling functions are:
  - Max pooling – the content of the window is summarized considering its maximum value
  - Average Pooling – the content of the window is summarized by averaging the values it contains

Activation functions

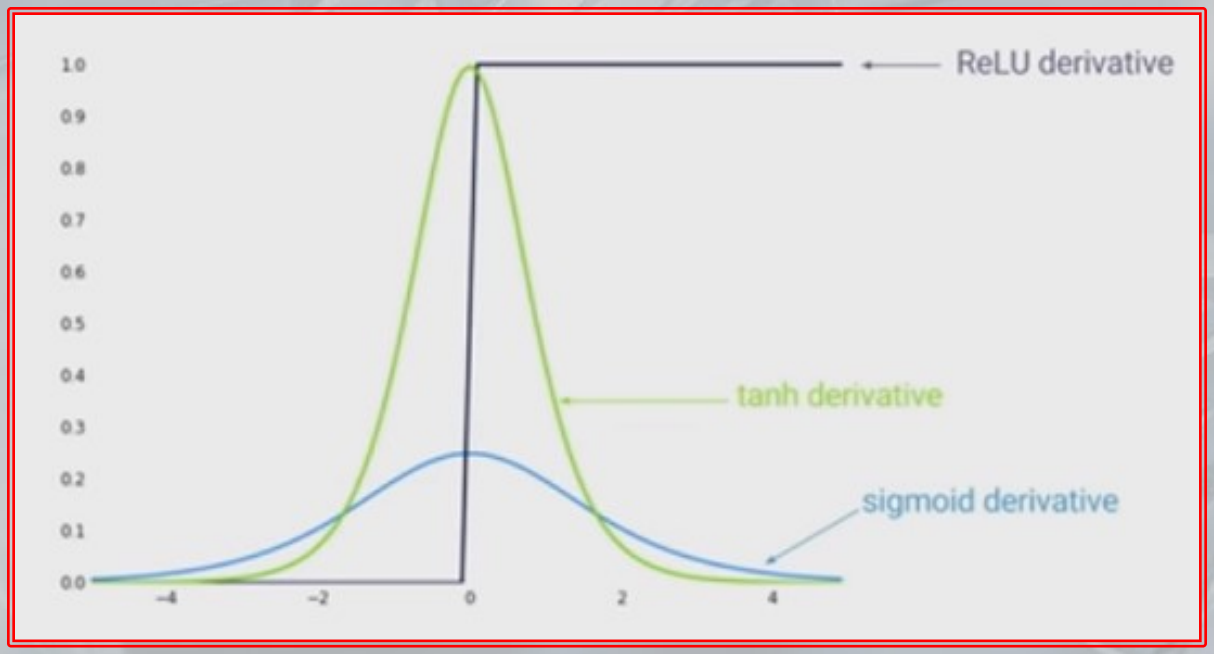
- They serves to construct non–linear models
- The use of the ReLU (Rectified Linear Unit) function avoids neuron saturation
  - It speeds up the calculation and reduces the problem of vanishing gradient



Sigmoidal activation functions generate gradients in the interval (0, 1)
BackPropagation calculates the gradients using the chain rule: for each layer the gradient is multiplied by the gradients calculated on the previous layer ⇨ the gradient progressively reduces and the error signal decreases moving towards the deeper layers

Rectified Linear Unit

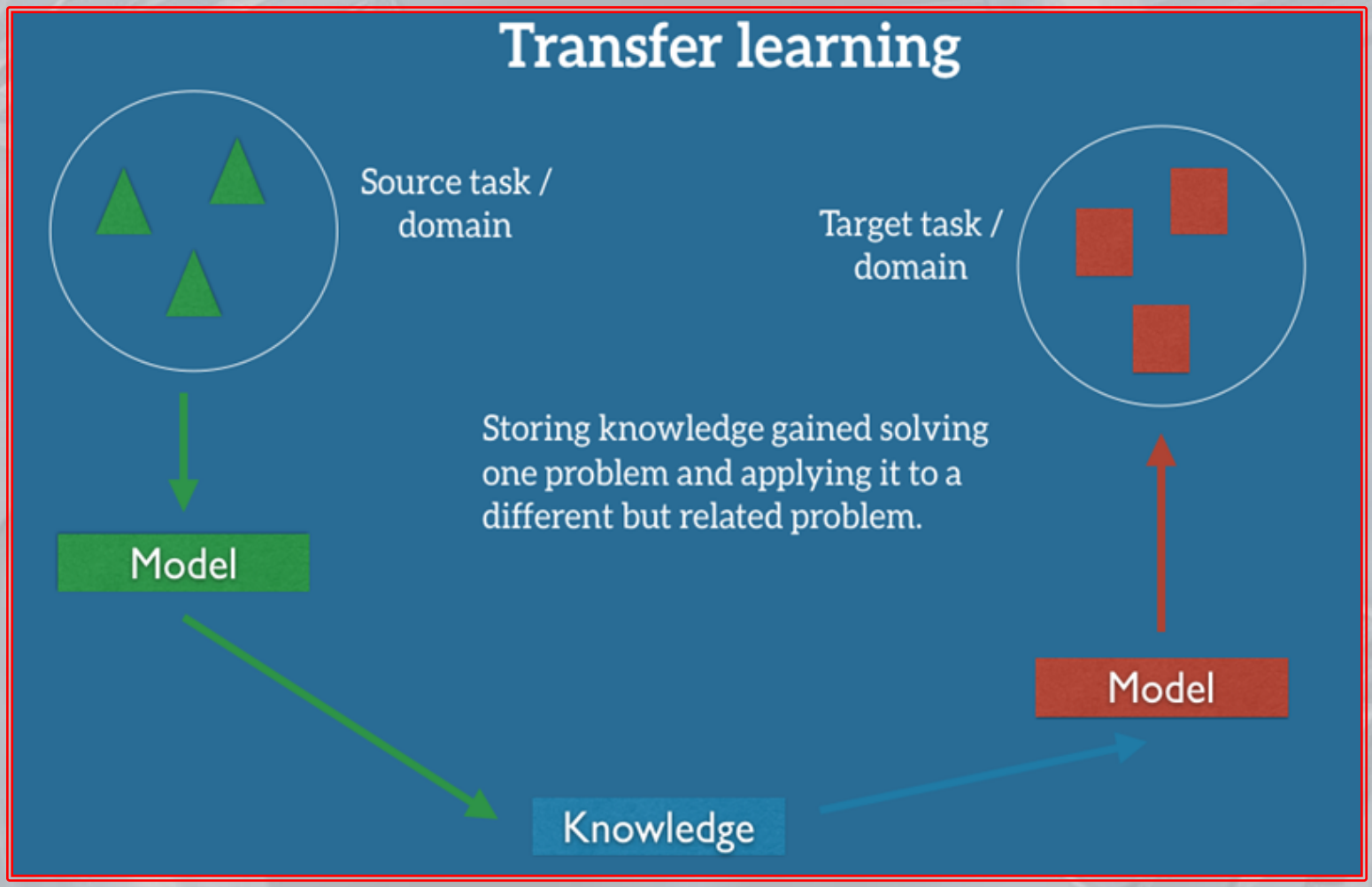
- The calculation of the derivative is simpler (with respect, for example, to a sigmoid or hyperbolic tangent)
- The derivative is 0 or 1 and multiplying by 1 does not decrease the error that is backpropagated
- Non–differentiability in zero is, theoretically, a problem, but in practice it never is



How to overcome the lack of data

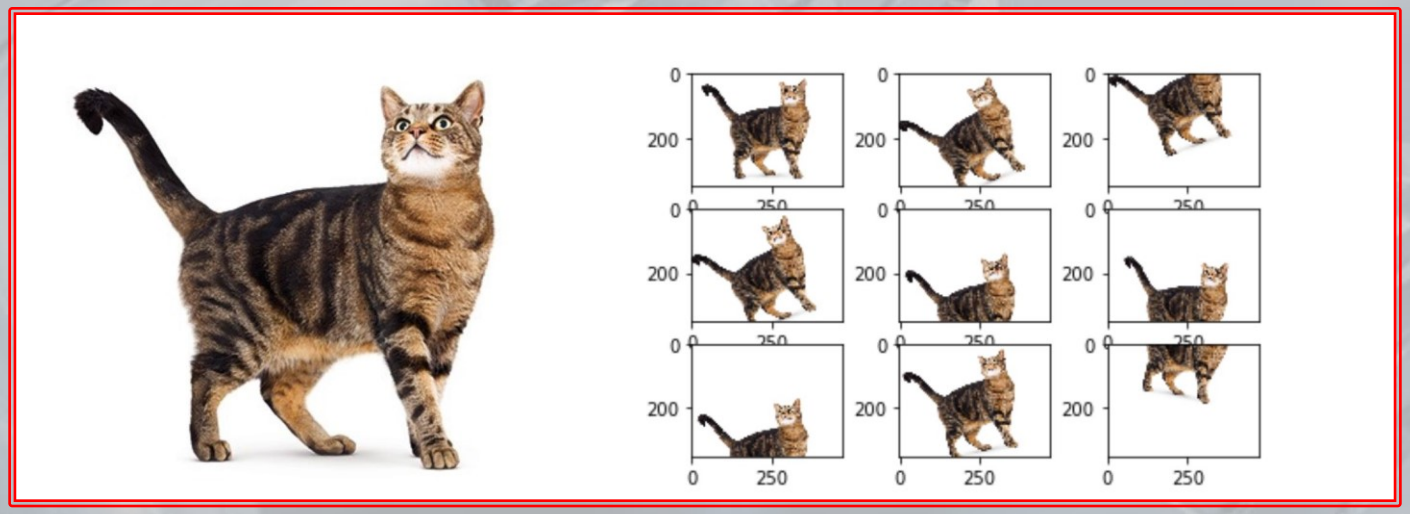
- Frequently, in real problems (especially in the medical field), the available data are not sufficient to train a deep network
- Transfer Learning – the basic premise of transfer learning is simple: take a model trained on a large dataset and transfer its knowledge to a smaller dataset; the model trained on the task for which there are many annotated data is reused as a "starting point" to be subsequently refined for solving a different problem, for which there are few examples
- Data Augmentation – the examples of the training set are increased, transforming the original data
- Generation of synthetic data – a set of synthetic data is generated together with the relative targets; generally, synthetic data are used to pre–train the network

Transfer learning



Data augmentation

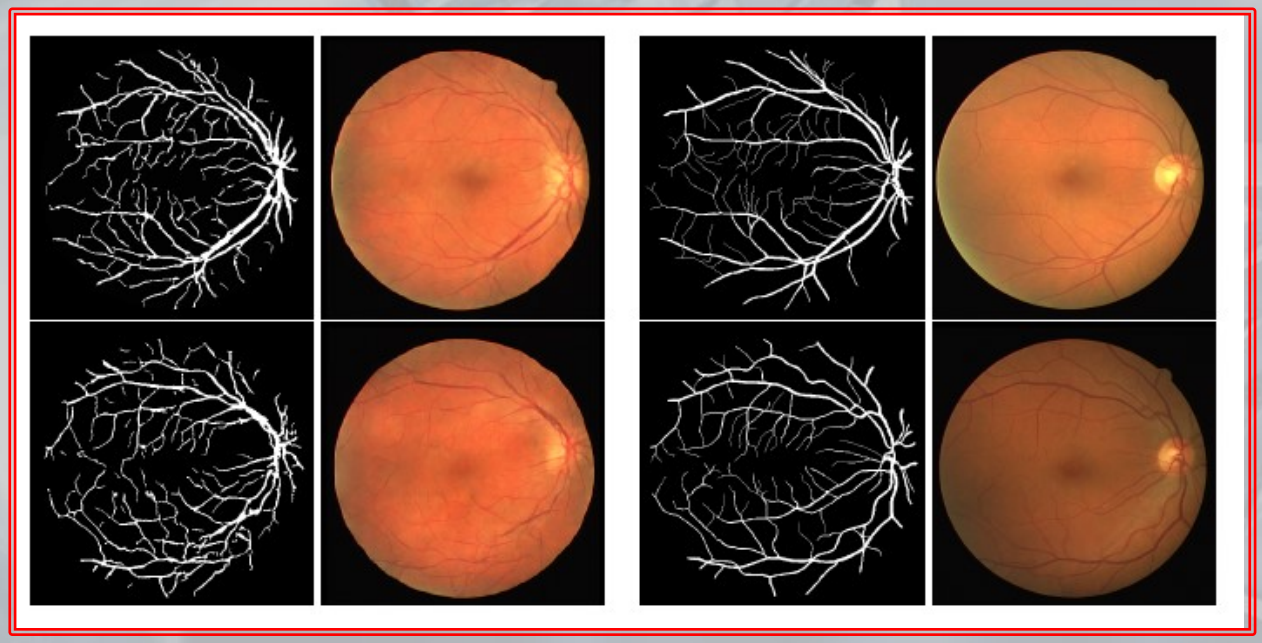
- Increasing the size of the training set by modifying an image through rotations, symmetries (mirroring), color modification, etc.
- The target, in this case, derives directly from the annotation of the original image



Generation of synthetic data

- There are specialized deep architectures to solve this problem
- GANs (Generative Adversarial Networks) are composed of a "generator" network and a "discriminator" network which serve, respectively, to generate new data starting from a known distribution and to evaluate their "fidelity to the originals"

Generated images



Real images

Deep network design

- To create and train a deep network, the following hyperparameters must be defined:
  - Network architecture (number of layers, type of layers, number of units for each layer, etc.)
  - Activation functions (tanh, sigmoid, ReLU, or similar)
  - Output layer
  - Loss function (or cost function, or error function)
  - Optimizer
  - Learning rate

Output layer

- The role of the output layer is to apply a final transformation to the data before producing the model output
- The most used output layers are:
  - Linear layer – It realizes an affine transformation
$$\hat{y} = W^T h + b$$
  - Sigmoid layer – It is used in classification tasks
$$\hat{y} = \sigma(W^T h + b)$$
  - Softmax layer – It is a "generalization" of the sigmoid function, which can be used to represent a probability distribution on a discrete variable with n possible values

$$\hat{y} = \frac{e^{z_i}}{\sum_j e^{z_j}} \text{ dove } z = W^T h + b$$

Loss function

- The loss function quantifies the error between predicted and expected outputs
  - It measures model performance
  - The error will be used by the optimizer to update the parameters
- Example: Mean Square Error (MSE)

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{y}_i - \widehat{\boldsymbol{y}}_i)^2$$

Optimizer

- The most famous training technique for neural networks is BackPropagation
  - The input patterns are propagated forward to calculate the relative outputs
  - Based on the outputs, the error is calculated and propagated backwards by updating the weight values (Stochastic Gradient Descent – SGD)
- Various techniques proposed to improve SGD:
  - Momentum – It keeps track of recent updates and allows the attenuation of fluctuations in descending the gradient
  - Adaptive Moment Estimation (Adam): It calculates learning rates adaptively and separately for each parameter

Example of a deep architecture