Hands-on Introduction to Deep Learning

Convolutional Neural Networks

INSTITUT DU DÉVELOPPEMENT ET DES RESSOURCES EN INFORMATIQUE SCIENTIFIQUE

Objective of the section:

- Understand the functioning of convolutional neural networks
- Discover the origin of the performances of CNNs

Duration: ½ day

Document annex: PL2 folder

Aspects addressed:

- Convolution
- Notion of filter
- Padding
- Pooling
- Dilation
- Architectures of CNNs
- Residual connections (Resnet)
- Inception module
- Uses of CNNs

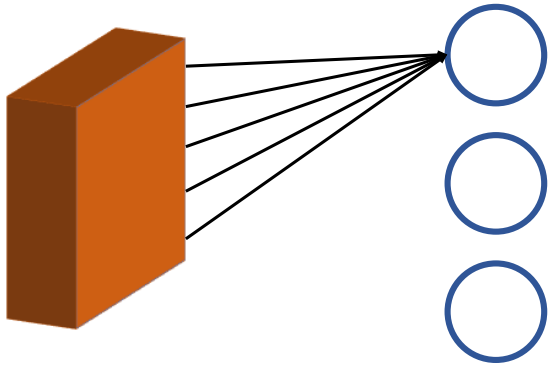


Image : 3 x 5 x 5

1 layer, 3 neurons :
75 weights per neuron

Motivations

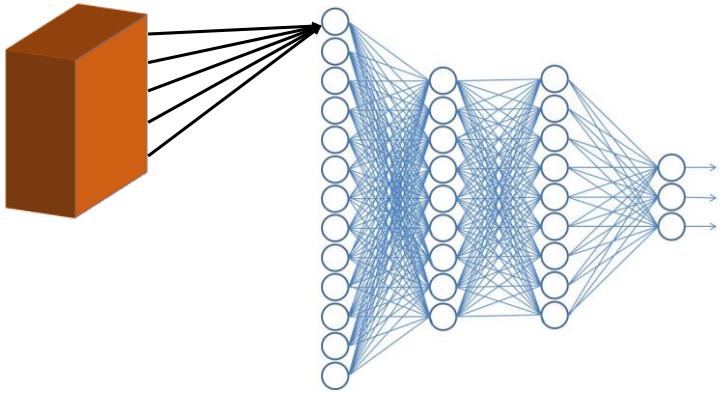


Image : 3 x Height x Width

Complex model

Motivations

Problem of "fully connected" networks: number of parameters

Example: 5x5 RGB image (3 channels):

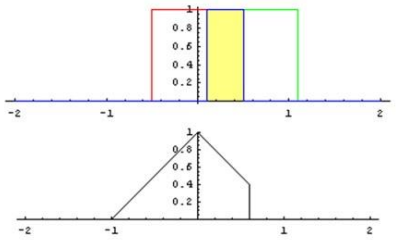
per neuron of the first layer 5x5x3=75 connections (parameters)

Image 1000*1000 RGB: per neuron of the first layer => 3*10^6 parameters

With several neurons per layer and several layers => impractical

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

$$f * g = \sum_{\substack{0 \le i \le n \\ 0 \le j \le m}} f(x_{i,j})g(x_{k-i,l-j})$$



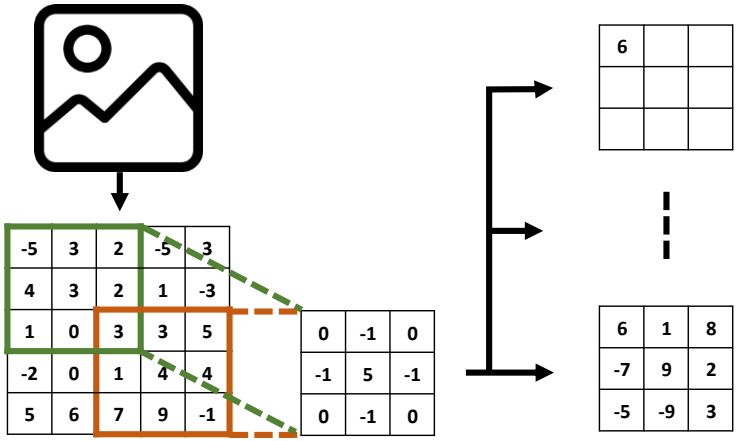
Convolution

Idea: use convolution

Inspiration from signal processing
Applying a filter to a signal can be used to emphasise or attenuate certain characteristics of the image.

Convolution allows a filter to be applied to either temporal or spatial data.

The filter is called a kernel.



Convolution 2D

Convolution is also suitable for discrete data.

It can also be used regardless of the number of dimensions of the data.

Easily computable: it is a series of matrix products that can be parallelized.

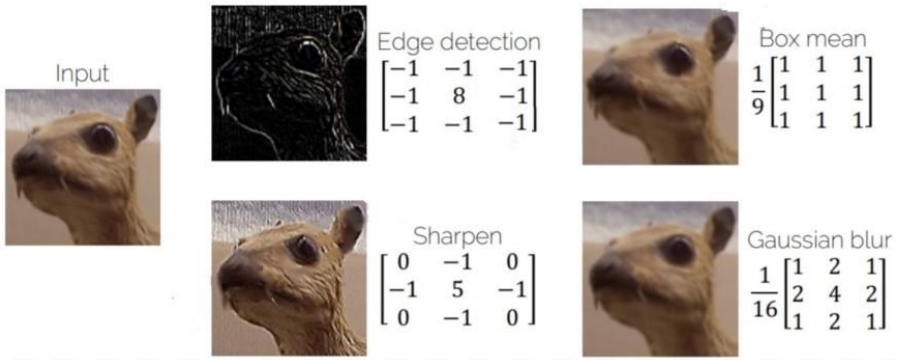
Allows weights to be pooled for multiple use.

Some shortcomings:

- Not robust to rotation
- To intensity changes, ...
- Axial symmetry (mirror inversion)
- ....

Fixes:

- Normalisation of data
- Data augmentation



Convolution 2D

Used extensively in computer vision, it allows an image to be filtered by a chosen kernel to :

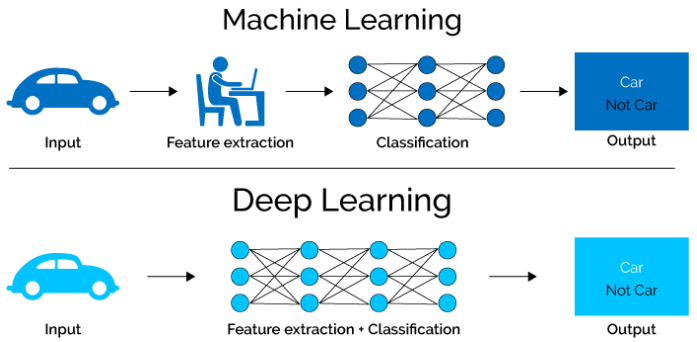
- highlight features such as edges
- modify the image by applying a form of blur.

As in image processing, the weight of these kernels can be chosen manually.

We can also consider the weights of these kernels as parameters that we will try to optimise as in dense neural networks.

Advantage: the weights of the neurons are shared.

The neural network learns to use filters to identify the features useful for its task.



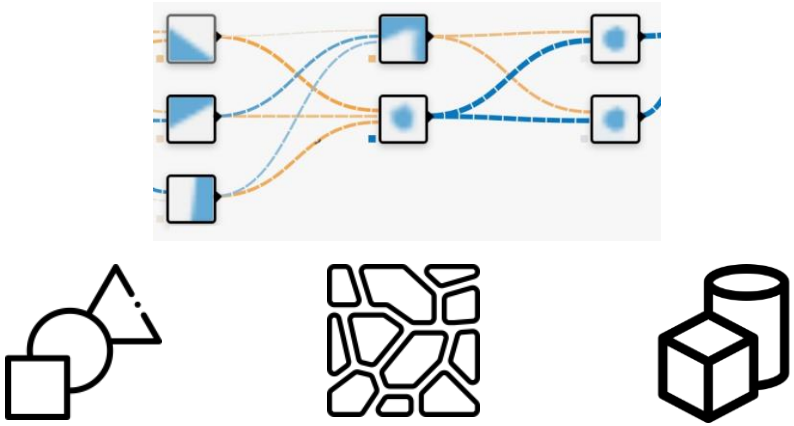
Convolutions role

Traditional (Machine Learning):

- Specialised feature extractors
- Feature exploitation: predictive architecture (SVM, Random Forest, ..., or dense neural networks) to be trained

Deep Learning:

- Features extraction: architecture to be trained
- Feature exploitation: architecture to be trained

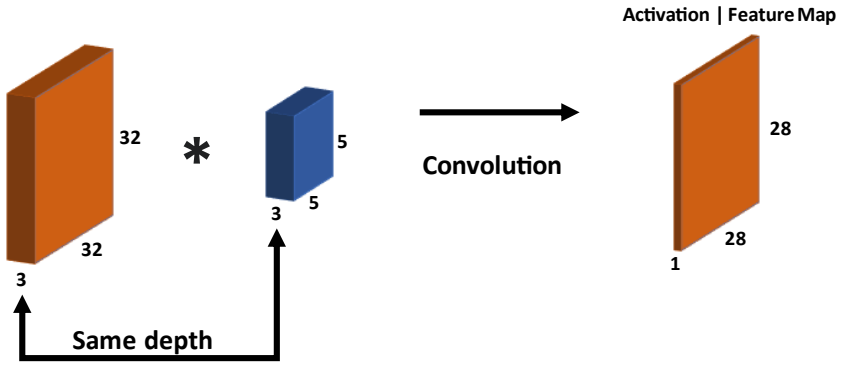


Convolutions role

As with dense neurons, we can chain convolutions and free ourselves from the need for human intervention to build the features the model needs.

Example on images:
The first convolution layers recognise simple shapes
The next layers combine the simple shapes and detect patterns, structures.
In a deep enough network, complete objects can be found in the last layers.
The deeper the model, the more complex combinations of features it can build.

Problem: Risk of overfitting
As for dense network, there is a risk for the model to overfit the features learned to patterns in the dataset thus inducing the inability to extrapolate to new and different data.

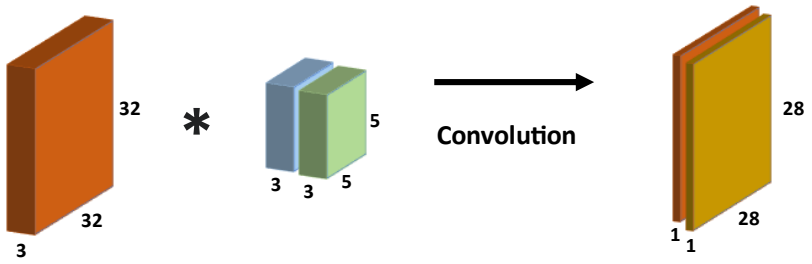


Convolution 3D

When applying a convolution, the kernel must have the same depth as the input.
Example: an RGB image has a depth of 3, the kernel must also have this depth.

Convolution of an input with a kernel generates an activation or feature map of depth 1. The output is so called because the output represents the areas in the image where the response is stronger or weaker (activation) relative to the filter (feature).

The dimensions of the output are reduced compared to the input because of the number of possible positions for the filter: 32 - (5-1) = 28 possible positions here



Kernel and activations map

It is possible to use several kernels at a given level.

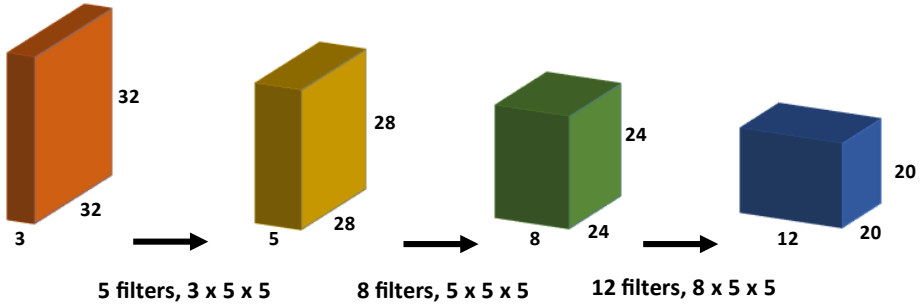
This allows several different features to be extracted at the same scale.

Each kernel will generate an output of depth 1.

With multiple kernels, we can stack the feature maps to form a new output with multiple features per location.

It is the set of kernels applied to an input that we call a convolutional layer defined by:

- The number of kernels
- The parameters (size, stride,…) of the kernels



Dimensions evolution

When applying convolutions on the previous activation maps, the size of the desired output must be considered to choose the size of the following filters.

By chaining convolutions this way, it is possible to obtain a representation of the input data as a feature vector.

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

Output : $(\frac{N+2\cdot P-F}{S}+1)\times(\frac{N+2\cdot P-F}{S}+1)$

- **Entrée : 5 x 5**
- **Filtre : 3 x 3**
- **Padding : 1**
- **Stride : 1**
- **Sortie : 5 x 5**

Padding

Problems:

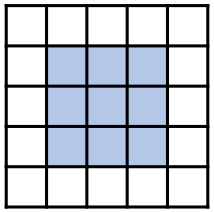
- Input dimensions reduced rapidly
- Centre pixels used several times
- Corner pixels used only once
- Creation of a bias towards the "centre" of the data

Solution: Padding
Extend the data with artificial elements.
Benefits :

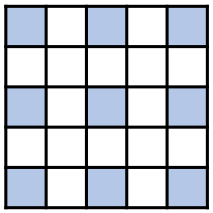
- Prevents reduction in size
- Uses 2 times more values on the edges
- Uses 4 times more values in the corners

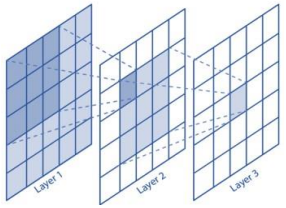
Striding :
This is the step of moving the kernel when calculating the convolution.
Increasing this value forces the combination of spatially more distant features.

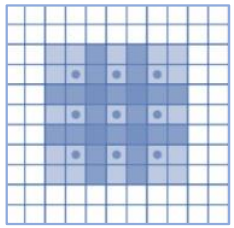
D = 1



D = 2







Dilated convolution

Receptive field:
The region that has an influence in the production of a feature.

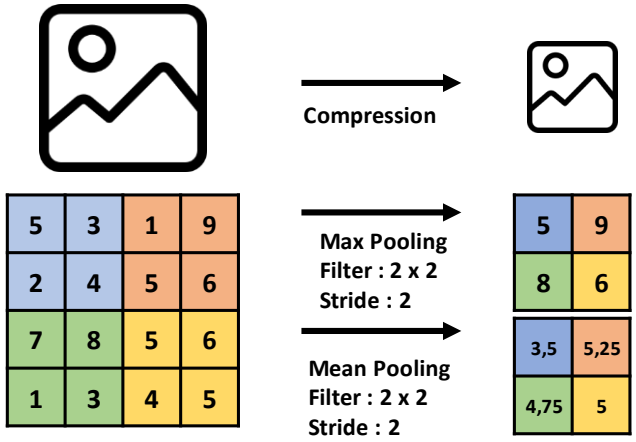
Influence of receptive field – an example:
In semantic segmentation, the objective is to decide the class of each pixel. Having a larger receptive helps to add the context around each pixel for the model to take a decision.

To increase the receptive field, multiple actions can be taken as changing the size of the kernel, using pooling, using dilated convolution.

Dilated convolution: A technique that expands the kernel by inserting holes between its consecutive elements.

Benefits:
Allows to extend the size of the receptive field
Without increasing the number of parameters

It is extensively used in concepts as spatial pyramid.



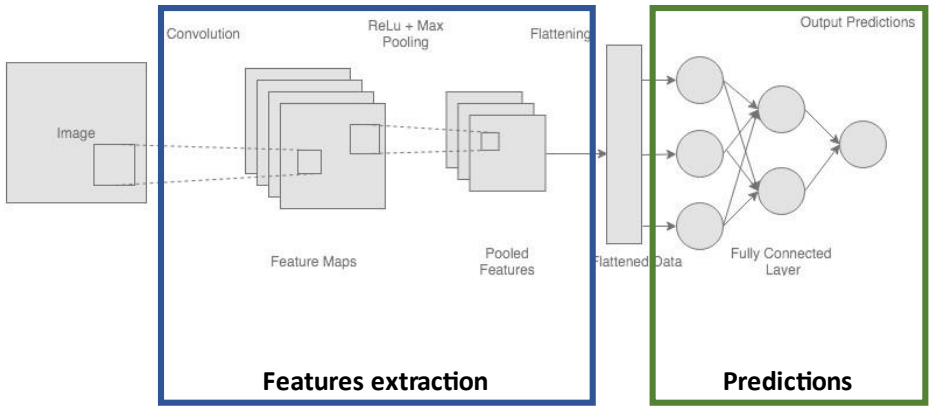
Pooling

A variety of characteristics are desired. But we also want to limit the amount of information by limiting redundancy.

To achieve this, a pooling step is applied which allows the input to be sub-sampled and the number of features to be reduced.

Two popular methods:

- Max pooling: Feature reduction by selecting the one with the highest response locally.
- Mean pooling: Feature aggregation by locally averaging the features.

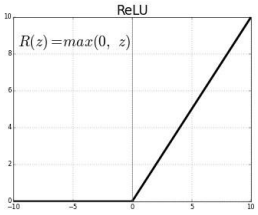


Final Fully connected layers prototype

Convolutions (followed by an activation function) and pooling can be combined to obtain features representing the input data.

It is common to aim at obtaining a vector which is then used with a few dense layers (usually between 1 and 4 layers) to perform the desired predictive task.

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



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

Activation function

Just as for the fully connected layers, it is necessary to break to generate non-linearity by applying an activation function.

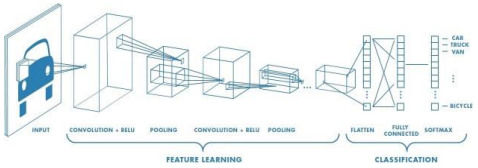
For CNNs, it is typically the Relu function which is used.

```python
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
```




```
torch.nn.Conv2d(
        in_channels,
        out_channels,
        kernel_size,
        stride=1,
        padding=0,
        dilation=1,
        groups=1,
        bias=True,
        padding_mode='zeros',
        device=None,
        dtype=None)
```

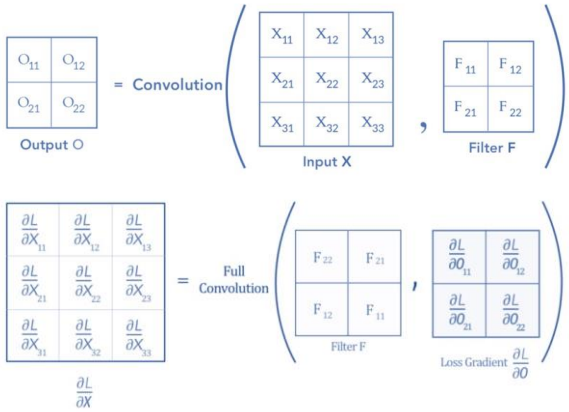
Basic CNN implementations (Pytorch)

It is this basic sequence that allows the construction of a CNN including the LeNet model. Today, the most recent models integrate other techniques to improve performance and training.

One of the interests of CNNs: A form of explicability

It is possible to visualise the activation maps or kernels to see which features are learned and which areas of the image have the strongest activations.

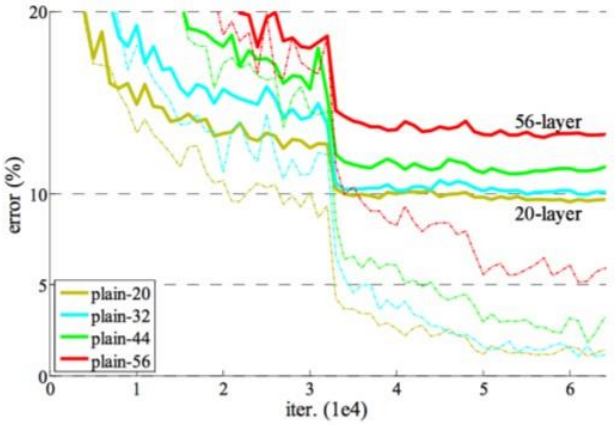
The implementation of models is nowadays facilitated using libraries as Pytorch or Tensorflow. They allow to define a model architecture and forward pass. The backward pass is then generated using automatic differentiation.



Convolution backpropagation

Pavithra Solai. "Convolutions and Backpropagations'

For convolutions, the backward pass is facilitated by the fact that we can still apply the chain rule and determinate that the gradient of the output can also be obtained by doing a convolution.



Depth problem

He, Kaiming et al. "Deep residual learning for image recognition

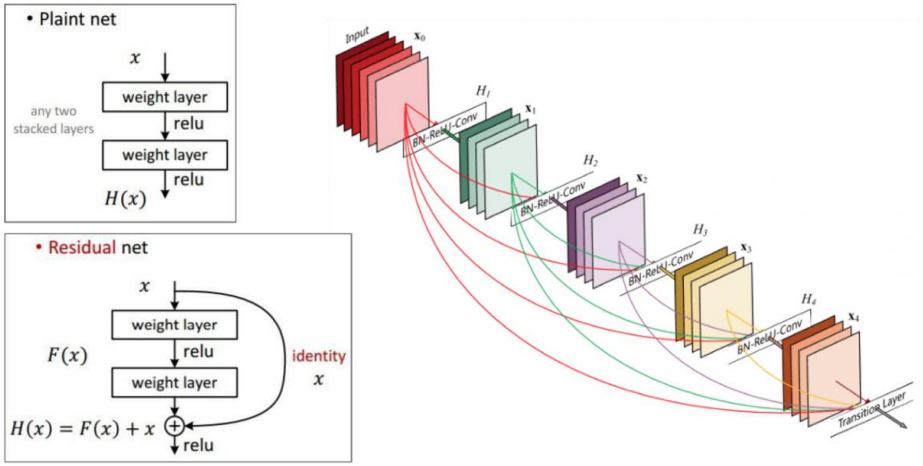
Limitations:
The more layers one uses, the more difficult the training is, as can be seen on the graph, the training for a complex network converges to less efficient parameters

Explanation:
A layer is built from the previous layer, accumulated errors from successive layers can create undesirable effects
Gradient explosion:

- Error in the gradients
- Less efficient network
- Accumulation of errors and very high gradients
- Difficulties in finding the right relations between complex features

Disappearance of gradients:

- Sigmoids have low derivative values.
- During backpropagation, this effect is multiplied by the gradients, which stops the training of certain neurons.



He, Kaiming et al. "Deep residual learning for image recognition

Classic CNN: Each layer only takes as input the output of the previous layer

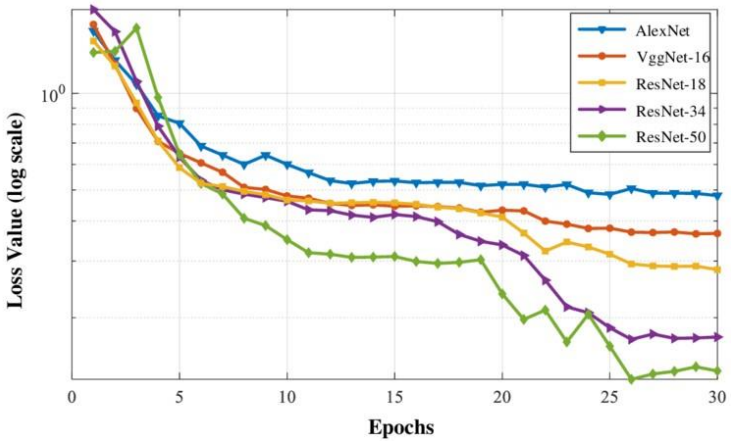
Idea: create a shortcut to keep the value of the previous layer in the output

Biological inspiration: we find neurons whose connections "jump" over intermediate neurons in the cerebral cortex

We obtain a so-called residual architecture; we no longer conceive of a layer as a function that gives F(x) where x is the result of the previous level
We consider a layer that gives x + F(x)

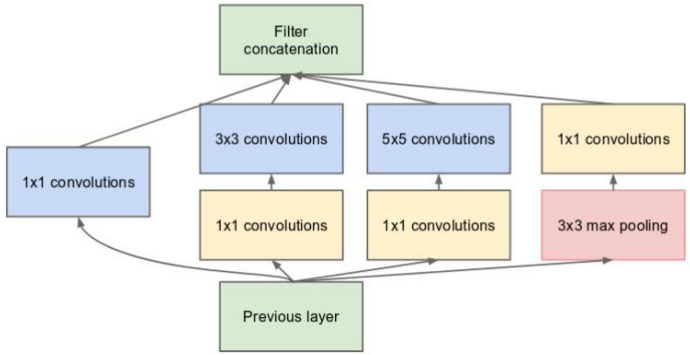
Dual purpose:

- Combining features of different levels
- Learning aid with “expressway” for backpropagation of gradients



He, Kaiming et al. "Deep residual learning for image recognitior

This method significantly improves learning performance and maintains a consistent progression when increasing the depth of the model.



Inception module

Szegedy, Christian, et al. "Going deeper with convolutions

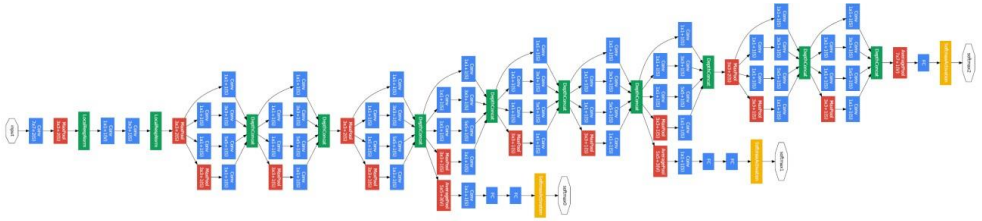
Problem: The kernels have a size that induces a bias towards the desired feature size.

How to reduce this effect?

- By using multiple kernel sizes.

How to use several kernel sizes?

- Use convolutions with a kernel 1x1 which standardizes the size of the outputs
- Concatenate the outputs once they are the same size

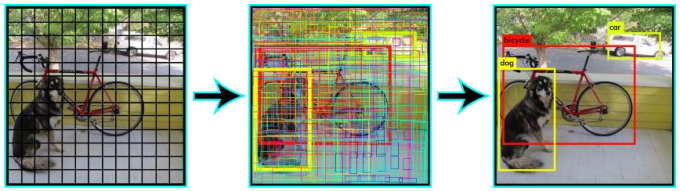


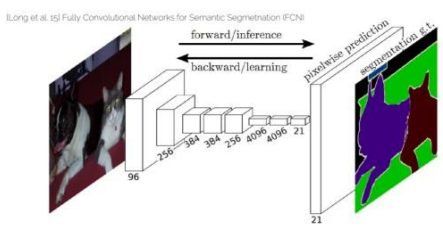
GoogLeNet

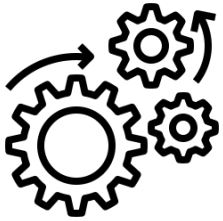
Szegedy, Christian, et al. "Going deeper with convolutions

The GoogLeNet architecture uses the inception module.
This architecture has shown impressive performance demonstrating the usefulness of this type of feature combination while reducing memory occupancy and training times compared to traditional models such as VGG.

To help the training, they also introduced multiple “heads”: final outputs from which a loss will be calculated. This way, the gradients are more easily retropropagated.







Applications

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection."
Long, et al "Fully convolutional networks for semantic segmentation

The use of CNNs is not limited to image classification.

CNNs can be used for a wide variety of tasks:

- Semantic segmentation
- Object detection
- Encoders - Decoders (compression, decompression of information)
- Transformation of data
- …

For this, it is necessary to think about the task to achieve (dimensions of the inputs, outputs, kernels, layers, etc).

The use of CNNs is not limited to images. Convolution is not specific to the size of images and can very well be used for vectors for example and various type of data.