Chapter 3: CNN Convolutional Neural Network

The final list consisted of:

1. Introduction (Motivation: learning the features)
2. CNN Convolutional Neural Network (definition)
3. Convolution Layer
4. Pooling Layer
5. Full connected layer
6. Architecture of a CNN

1. Introduction

In traditional machine learning algorithms, the notion of *features* in vision is used to classify images. These methods consist of extracting the *features* of each image from the dataset **manually** by an expert, then training a classifier on these *features*. These supervised learning techniques can provide very good results, and their performance depends heavily on the quality of the *features* previously found. There are several methods for extracting and describing *features* (such as SIFT).

But in 2012, a revolution occurred: at the annual ILSVRC computer vision competition, a new ***deep learning*** algorithm broke records! This is a CNN **convolutional neural network**.

Convolutional neural networks have a methodology similar to that of traditional supervised learning methods: they receive input images, **automatically** detect the *features* of each of them, and then train a classifier on them. So the CNNs do all the tedious work of extracting and describing *features* themselves.

During the training phase, the classification error is minimized in order to optimize the classifier parameters and ***features***! In addition, the specific architecture of the network makes it possible to extract *features* of different complexities, from the simplest to the most sophisticated. The automatic extraction and prioritization of *features*, which adapt to the given problem, is one of the strengths of convolutional neural networks.

Today, convolutional neural networks, also called **CNN** or **ConvNet** for Convolutional Neural Network, are still the best performing models for image classification. This part is therefore naturally dedicated to them.

2. CNN Definition

What is the difference between a neural network and a convolutional neural network?

Convolutional neural networks refer to a subcategory of neural networks. However, CNNs are specifically designed to process input images. Their architecture is then more specific: it is composed of two main blocks.

1- The first block is the peculiarity of this type of neural networks, sinceit functions as a ***feature* extractor**. To do this, by applying convolutional filtering operations. The first layer filters the image with several convolution kernels, and returns *feature maps*, which are then normalized (with an activation function) and/or resized. This process can be repeated several times: we filter the *feature maps* obtained with new kernels, which gives us new *feature maps* to normalize and resize, and which we can filter again, and so on. Finally, the values of the last *feature maps* are concatenated in a vector. This vector defines the output of the first block, and the input of the second.
2- The second block: The input vector values are transformed (with several linear combinations and activation functions) to return a new output vector. This last vector contains as many elements as there are classes: the element *i* represents the probability that the image belongs to the class *i*

As with ordinary neural networks, the parameters of the layers are determined by backpropagation of the gradient: the cross entropy is minimized during the training phase. But in the case of CNNs, these parameters designate in particular the *features* of the images. See now the different types of layers of a CNN.

There are four types of layers for a convolutional neural network: the **convolution**, the ***pooling layer,*** the **ReLU correction** layer and the fully-connected ***layer***.

3. Convolution Layer

Convolution, from a simplistic point of view, is the act of applying a mathematical filter to an image. From a more technical point of view, it is a matter of dragging a matrix over an image, and for each pixel, use the sum of the multiplication of this pixel by the value of the matrix. This technique allows us to find parts of the image that might be of interest to us. Let's take the Figure below on the left as an example image and the Figure on the right as an example of

M.

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Simplistic example of the values of pixels of a 5x5 image

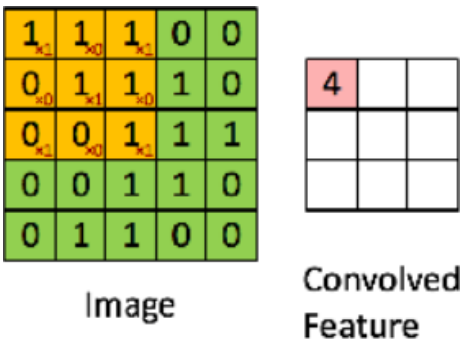
F.

1	0	1
0	1	0
1	0	1

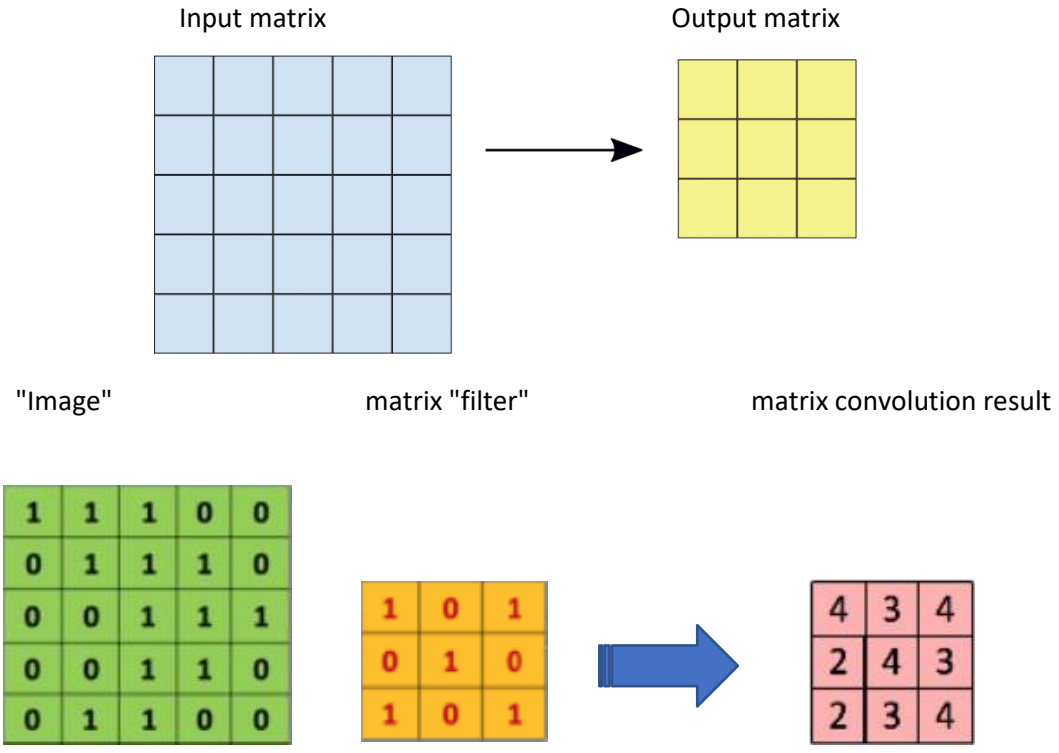
Example of values of a matrix used as a filter

In the case of Figure 17, the values are binary. In a real case, the values should vary between 0 and 255. In Figure 18, the values are represented by 1s and 0s. In a real case, these values are continuous and can be positive or negative.

Apply the filter on the image: in the image matrix M, we can see that each value of the pixels of the tile image (the orange boxes) is multiplied by each corresponding value of the filter (1x1, 1x0, 1x1 ….). Then add all these values to obtain a single value ’4’ which will be part of a new convolved image.



The filter should move one box at each iteration until the first line is finished. When we finish the first row, the filter “goes down” by one box and the same procedure is repeated for each row and column. Slow animation



Noted that a 3x3 convolution of depth 1 performed on a 5x5 input feature map, also of depth 1. Since there are nine possible 3x3 slots to extract the tiles from the 5x5 feature map, this convolution generates a 3x3 output feature map.

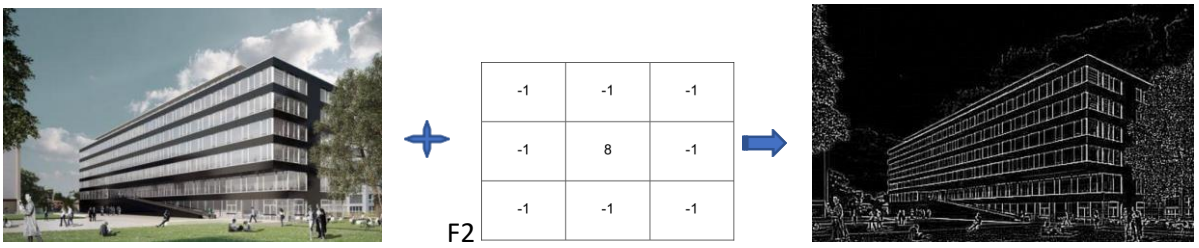
A convolutional neural network contains multiple filters and these filters are applied
Original image After the first step we have so many new images

switching. **The convolutional phase can also be seen as layers of neurons hidden** where each neuron is only connected to a few neurons in the next layer.

Effects of applying filters

To explain how a filter can find interesting parts in images. Let's take the previous Filter. Values that are at 1 form an "X". During the convolution phase, when this filter is applied to a shape that corresponds exactly to an "X", then the value obtained will be higher. The less the shape of the image corresponds to the shape of the filter, the lower the value obtained. So we can reduce the size of the image and bring out the interesting elements.

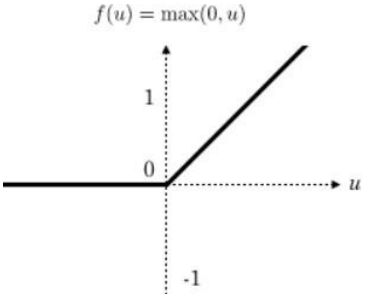
In order to demonstrate the effects of a convolution phase, the following figure, which is a synthetic image of building B, will be modified with several mathematical filters. For example, if we want to find edges, we can use the values of the F2 filter.



The filters are also adapted to each learning iteration because the values of the mathematical filters used are weights as in multilayer neural networks

ReLu – Rectified Linear Unit

ReLu is a function that must be applied to each pixel of an image after convolution, and replaces each negative value with a 0. If this function is not applied, the created function will be linear and the XOR problem persists since in the convolution layer, no activation function is applied.



ReLu is widely used in convolutional neural networks because it is a fast function to compute: () = () (0, ()). Its performance is therefore better than other functions where expensive operations have to be carried out.

How to choose features?

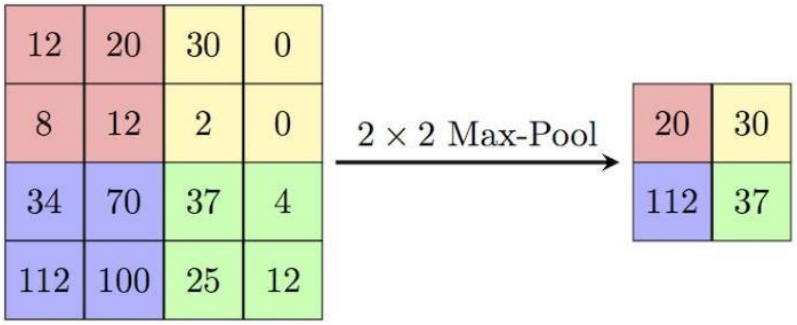
Unlike traditional methods, features are not pre-defined according to a particular formalism (e.g. SIFT), but learned by the network during the training phase! The filter cores designate the weights of the convolution layer. They are initialized and then updated by backpropagation of the gradient.

This is the strength of convolutional neural networks: they are able to determine on their own the discriminating elements of an image, adapting to the problem posed. For example, if the question is to distinguish cats from dogs, automatically defined features can describe the shape of ears or paws.

Pooling

This type of layer is often placed between two convolution layers: it receives several *feature maps* as input, and applies the ***pooling*** operation to each of them. The *pooling* operation consists of reducing the size of the images, while preserving their important characteristics. To do this, the image is cut into regular cells, then the maximum value is kept within each cell.

The method used consists in imagining a window of 2 or 3 pixels that slides over an image, as for convolution. But, this time, we do steps of 2 for a size 2 window, and steps of 3 for 3 pixels. The window size is called "kernel size" and the footsteps are called
"strides" For each step, we take the highest value among those present in the window and this value constitutes a new pixel in a new image. This is called Max Pooling.



The *pooling* layer reduces the number of parameters and calculations in the network. This improves the efficiency of the network and avoids over-learning.

The maximum values are located less accurately in the *feature maps* obtained after *pooling* than in those received as input – this is actually a great advantage! Indeed, when you want to recognize a dog for example, its ears do not need to be located as precisely as possible: knowing that they are located roughly next to the head is enough!

Thus, the *pooling* layer makes the network less sensitive to the position of the *features*: the fact that a *feature* is located a little higher or lower, or even that it has a slightly different orientation should not cause a radical change in the classification of the image.

5. Fully-connected layer

The fully-connected layer is always the last layer of a neural network. This type of layer receives an input vector and produces a new output vector. To do this, it applies a linear combination and then optionally an activation function to the values received as input.

The last fully-connected layer makes it possible to classify the image at the input of the network: it returns a vector of size N , where N is the number of classes in our image classification problem. Each element of the vector indicates the probability for the input image to belong to a class.

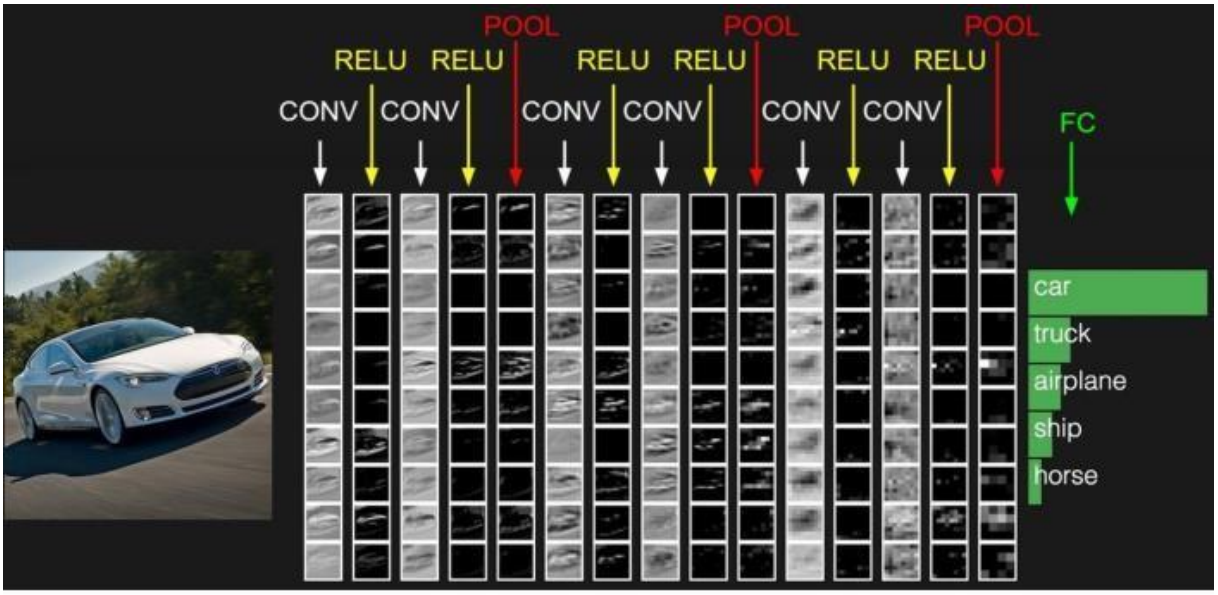
For example, if the problem consists in distinguishing cats from dogs, the final vector will be of size 2: the first element (respectively, the second) gives the probability of belonging to the class "cat" (respectively "dog"). Thus, the vector [0.9 0.1] means that the image has a 90% chance of representing a cat.

To calculate the probabilities, the fully-connected layer therefore multiplies each input element by a weight, sums, and then applies an activation function (logistics if N=2 , softmax if N>2): This processing amounts to multiplying the input vector by the matrix containing the weights. The fact that each input value is connected with all output values explains the term fully-connected.

6. Architecture of a CNN

A convolutional neural network can have several convolutional steps, ReLu and Pooling. A rule to follow is that the ReLu function must be applied after a convolution step in order to have a non-linear response, but Pooling is not mandatory.

After going through all the steps of convolution, ReLu and Pooling, we can move on to image classification. The last phase consists of sending all the pixels into a multilayer neural network. Since we have been able to recover the most important parts of an image that we have condensed, the classification phase will be much more efficient than using an artificial neural network without convolution.



A CNN is simply a stack of multiple layers of convolution, pooling, ReLU correction, and fully-connected. Each image received as input will therefore be filtered, reduced and corrected several times, to finally form a vector. In the classification problem, this vector contains the probabilities of belonging to the classes.

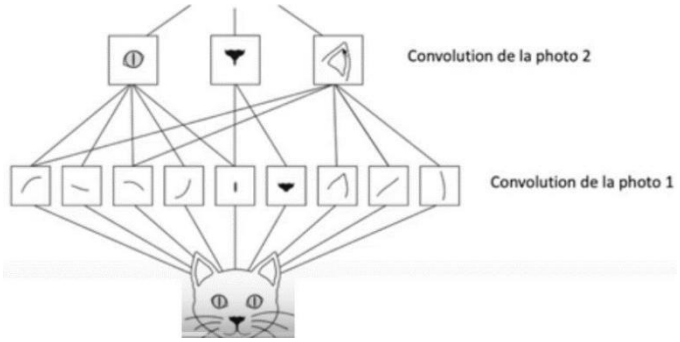
All convolutional neural networks must start with a convolution layer and end with a fully-connected layer. Intermediate layers can stack in different ways, provided that the output of one layer has the same structure as the input of the next. For example, a fully-connected layer, which always returns a vector, cannot be placed before a pooling layer, since the latter must receive a 3D matrix.

Typically, a neural network stacks multiple convolution and ReLU correction layers, then adds a pooling layer (optional), and repeats this pattern several times; then, it stacks fully-connected layers.

The more layers there are, the deeper the neural network is: we are in the middle of **Deep Learning**!

The first convolution layer learns simple features, which represent rudimentary structural elements of the image (contours, corners...) The higher the convolution layers, that is to say far from the input of the network, the more complex the features learned: these consist of the simpler features of the previous layers. A square is an example of a complex feature, formed of contours and corners.

The highest convolution layers therefore learn sophisticated features: for example convolution layer 2, in the case of cat recognition below, they can correspond to the ears, nose or eye.



Setting up layers

A convolutional neural network differs from another by the way the layers are stacked, but also parameterized. The convolution and *pooling* layers indeed have **hyperparameters**, that is to say parameters whose value you must define beforehand.

> *Note: The features of the convolution layer and the weights of the fully-connected layer are not hyperparameters, sincethey are learned by the neural network during the training phase. The size of the feature maps at the output of the convolution and pooling layers depends on the hyperparameters.*

Each image (or feature map) is $W \times H \times D$

where W is its width in pixels, H is its height in pixels and D is the number of channels (1 for a black and white image, 3 for a color image).

The convolution layer has four hyperparameters:

1. The number of ***K*** filters
2. The size ***F*** of the filters: each filter is of dimensions $F \times F \times D$ pixels.
3. The step ***S*** with which the window corresponding to the filter is dragged on the image. For example, a step of 1 means moving the window one pixel at a time.
4. The *zero-padding* ***P***:: a black contour of thickness P pixels is added to the image at the input of the layer. Without this contour, the output dimensions are smaller. Thus, the more convolution layers are stacked with P=0, the more the image at the input of the network shrinks. So we lose a lot of information quickly, which makes the task of *feature* extraction difficult.

For each input image of size *W*×*H*×*D*, the convolution layer returns a matrix with dimensions of $W_c * H_c * D_c$, où $W_c = \frac{W-F+2P}{S} + 1$, $H_c = \frac{H-F+2P}{S} + 1$ et $D_c = K$.

Choose : $P = \frac{F-1}{2}$ et $S = 1$ thus makes it possible to obtain *feature maps* of the same width and height than those received as input.

The pooling layer has only two hyperparameters:

- The size *F* of the cells: the image is cut into square cells of size *F*×*F* pixels
- The *S* pitch: the cells are separated from each other by *S* pixels

For each input image of size *W*×*H*×*D*, the *pooling* layer returns a matrix of dimensions: $W_p * H_p * D_p$, où $W_p = \frac{W-F}{S} + 1$, $H_p = \frac{H-F}{S} + 1$ et $D_p = D$

Just like stacking, the choice of hyperparameters is made according to a classic scheme:

- For the convolution layer, the filters are small and dragged onto the image one pixel at a time. The *zero-padding* value is chosen so that the width and height of the input volume are not changed at the output. In general, we then choose *F=3, P=1*, *S=1 or F=5, P=2, S=1*
- For the pooling layer, *F=2 and S=2* is a wise choice. This eliminates 75% of the input pixels. We can also find *F=3 and S=2*: in this case, the cells overlap. Choosing larger cells causes too much loss of information, and yields poorer results in practice

That's it, you have the basics to build your own CNN! In practice, it is advisable not to create a convolutional neural network from A to Z to solve your problem: the most effective strategy is to take an existing network that classifies a large collection of images well (such as ImageNet) and apply ***Transfer Learning*** – see the next section!