# Welcome to CNN Tutorial

Saraei Thamer
**Contact me :** thamer.saraei@USherbrooke.ca
thamer.assarray@etudiant-enit.utm.tn

**Join me** : 

March 14, 2019

## 1 Before you start

In the late 1980s, ***Yan Le Cun*** [1] developed a particular type of network called the **Convolutional Neural Network(CNN)**, these networks are a particular form of multilayer neural network whose architecture of connections is inspired by that of the visual cortex of mammals. For example, each element is connected only to a small number of adjacents elements in the previous layer. In 1995, Yan le cun and two other engineers developed an automatic check reading system that was widely deployed around the world.

At the end of the 1990s, this system read between 10 and 20 percent of all checks issued in the United States. But these methods were rather difficult to implement with computers of the time, and despite this success, convolutional networks and neural networks more generally were abandoned by the research community between 1997 and 2012.

## 2 Introduction

Multi Layer perceptrons or MLPs have shown their effectiveness as a learning technique for data classification. They are indeed able to approximate complex non-linear functions in order to process large data.

In the context of image classification, two approaches are possible:

- ☐ Extract features directly from the data. Classically, these features are extracted by a user-chosen algorithm. The characteristic vectors obtained are then presented at the input of a neural network.

[1] http://yann.lecun.com/

- □ Present the input image of a neural network. The image, however, needs to be vectorized, that is to say shaped into a vector whose size is equal to the number of pixels of the image.

In the first case, the network is content to perform a classification of feature vectors. The hotspot (feature extraction) is left to the discretion of the user, and the choice of algorithm for extracting features is crucial.

In the second case, several problems arise:

- Classically, the layers of a neural network are completely connected, that is to say that the value of a neuron of a layer n will depend on the values of all the neurons of the n-1 layer. Thus the number of connections (and therefore weight, parameters) can be very large. For example, for an images size 15*15, the input dimension of MLP is 225. If the hidden layer has 100 neurons, the the number of parameters of this layer is 100*225 = 22500. The number of parameters will increase exponentially with the size of the input (images). This great complexity of the network requires many learning samples, which is often not the case. The network will therefore tend to over-learn, and will therefore propose a poor generalization capacity.
- Another defect of MLPs for an application to images is that they are not invariant to transformations of the input, which happens very often with images (slight translations, rotations).
- Finally, MLPs do not take into account the correlation between pixels in an image, which is a very important element for pattern recognition.

Convolutional Neural Network (CNN) is an extension of MLPs that effectively responds to major MLP defects. They are designed to automatically extract the characteristics of input images, are invariant to slight distortions of the image, and implement the concept of weight sharing to significantly reduce the number of network parameters. This sharing of weights also makes it possible to take strong account of the local correlations contained in an image. Convolutional neural networks were initially inspired by the discovery by *Hubel and Wiesel* [2] of neurons sensitive to local and selective orientation in the visual system of the cat.
An important advance has been made by *Y. Lecun et al.* [3] with the use of a convolutional neural network whose learning has been realized by back propagation (backpropagation). This model has notably been applied successfully for the recognition of handwritten characters [4]

# 3 Convolutional Neural Network

Convolutional neural networks are to date the most powerful models for classifying images. They have two distinct parts. In input, an image is provided in the form of a

[2]https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/
[3]https://www.cs.rit.edu/ mpv/course/ai/lecun-90c.pdf
[4]http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

matrix of pixels. It has 2 dimensions for a grayscale image. The color is represented by a third dimension, of depth 3 to represent the fundamental colors [Red, Green, Blue].

The first part of a CNN is the actual convolutive part. It functions as a feature extractor of images. An image is passed through a succession of filters, or convolution kernels, creating new images called convolution maps. Some intermediate filters reduce the resolution of the image by a local maximum operation. In the end, the convolution maps are laid flat and concatenated into a characteristic vector, called the CNN code.

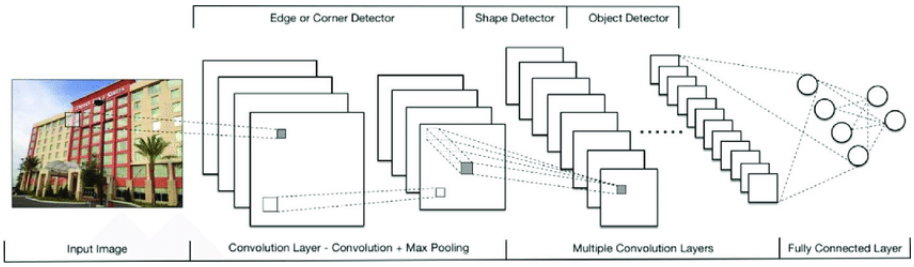



Figure 1: Standard architecture of a convolutional neuron network

This CNN code at the output of the convolutive portion is then connected to the input of a second portion, consisting of fully connected layers (multilayer perceptron). The role of this part is to combine the characteristics of the CNN code to classify the image.

The output is a last layer with one neuron per category. The numerical valyes obtained are generally normalized between 0 and 1, of sum 1, to produce a probability distribution of the categories.

## 3.1 Convolutional neural network architecture

Convolutional neural networks are based on multilayer perceptron (MLP), and are inspired by vertebrate visual cortex behavior. Although effective for image processing, MLPs have great difficulty managing large images, which is due to the exponential growth in the number of connections with the size of the image.

For example, if we take an image of 32x32x3 size (32 wide, 32 high, 3 color channels), a single fully connected neuron in the first hidden layer of the MLP would have 3072 entries (32 * 32 * 3). A 200x200 image would thus result in treating 120,000 entries per neuron which, multiplied by the number of neurons, becomes enormous.

CNNs aim to limit the number of entries while maintaining the strong "spatially local" correlation of natural images. In contrast to MLPs, CNNs have the following distinctive features:

- **'3D volumes of neurons' :** The neuron layer is no longer simply a surface (perceptron), but becomes a volume with a depth. If we consider a single CNN

receptor field, the n associated neurons (on the depth) form the equivalent of the first layer of an MLP.

- **'Local connectivity':** Thanks to the receiver field which limits the number of neuron inputs, while maintaining the MLP architecture, the CNNs ensure that the 'filters' produce the strongest response to a spatially localized input pattern which leads to a parsimonious representation of the entrance. Such a representation takes up less space in memory. In addition, since the number of parameters to be estimated is reduced, their (statistical) estimate is more robust for a fixed data volume (compared to an MLP).
- **'Shared weight':** In CNNs, the filtering parameters of a neuron (for a given receiver field) are identical for all other neurons of the same nucleus (processing all other receptive fields of the image). This setting (weight vector and bias) is defined in a "function card". This means that all neurons in a given convolutional layer detect exactly the same characteristic. By multiplying the receiver fields, it becomes possible to detect elements independently of their position in the visual field, which induces a translation invariance property.

Together, these properties allow convolutional neural networks to obtain a better generalization (in terms of learning) about vision problems. Weight sharing also greatly reduces the number of free parameters to learn, and thus the memory requirements for network operation. The decrease of the memory footprint makes it possible to learn larger networks that are often more powerful.

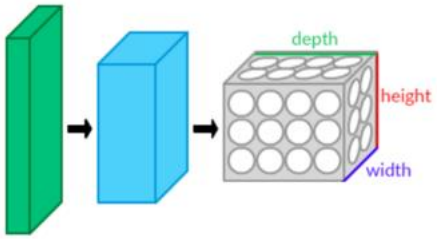



Figure 2: A layer of CNN in 3 dimensions: (Green = input volume, blue = receiver field volume, gray = CNN layer, circles = independent artificial neurons)

An CNN architecture is formed by a stack of independent processing layers:

- Convolution layer (**CONV**) that processes data from a receiver field.
- Pooling layer (**POOL**), which compresses information by reducing the size of the intermediate image (often by downsampling).
- The correction layer (**ReLU**), often referred to as abuse 'ReLU' with reference to the activation function (linear grinding unit).
- The "fully connected" (**FC**) layer, which is a perceptron type layer.
- The loss layer (**LOSS**).

### 3.1.1 Convolution layer (CONV)

The convolution layer is the basic building block of a CNN. Three parameters make it possible to dimension the volume of the convolution layer **the depth**, **the step** and **the margin**.

1. **Depth of the layer:** number of convolution nuclei (or number of neurons associated with the same receiver field).
2. **Step:** Controls the overlap of the receiver fields. The smaller the step, the more overlapping the receiver fields and the greater the output volume.
3. **The margin** (at 0) or **zero padding**: sometimes, it is convenient to put zeros at the border of the input volume. The size of this 'zero-padding' is the third hyper parameter. This margin makes it possible to control the spatial dimension of the output volume. In particular, it is sometimes desirable to keep the same area as that of the input volume.

If the step and the margin applied to the input image make it possible to control the number of receiving fields to be managed (treatment surface), the depth makes it possible to have a notion of output volume, and in the same way that an image can have a volume, if we take a depth of 3 for the three RGB channels of a color image, the convolution layer will also have a depth output. That is why we speak rather of "volume of exit" and "volume of entrance", because the entry of a convolution layer can be either an image or the exit of another layer of convolution.

The spatial size of the output volume can be calculated according to the size of the input volume *Wi* the processing surface *K* (number of receiver fields), the Step *S* with which they are applied, and the size of the margin *P*.

The formula for calculating the number of neurons in the output volume is :

$$W0 = \frac{Wi - K + 2 * P}{S} + 1 \tag{1}$$

If *W0* is not integer, the peripheral neurons will not have as much input as the others. It will therefore increase the size of the margin (to recreate virtual inputs).

Often, we consider a step **S** = 1, so we calculate the margin as follows: $P = \frac{K-1}{2}$ if you want an output volume of the same size as the input volume. In this particular case the layer is said to be "locally connected".

### 3.1.2 Pooling layer (POOL)

Another important concept of CNNs is pooling, which is a form of subsampling of the image. The input image is cut into series of rectangles of *n* non-overlapping side pixels (Pooling). Each rectangle can be seen as tile. The output signal of tile is defined according to the values taken by different pixels of tile.

Pooling reduces the spatial size of an intermediate image, reducing the amount of parameters and calculation in the network. It is therefore common to periodically insert a pooling layer between two successive convolutional layers of a CNN architecture to control the overfitting (over-learning). The pooling operation also created a form of translation invariance.

The pooling layer operates independently on each depth slot of the input and resizes it only at the surface level. The most common form is a stacking layer with tiles of size 2x2 (width / height) and as the output value the maximum input value. In this case we speak of "Max-Pool 2x2".

It is possible to use other pooling functions than the maximum. One can use an "Average Pooling" (the output is the average of the values of the input patch), "L2-normpooling". In fact, even if initially the average pooling was often used it turned out that the max pooling was more effective because it increases more significantly the importance of strong activations. In other circumstances tochastic pooling can be used.

Pooling allows big gains in computing power. However, because of the aggressive reduction in the size of the representation (and therefore the associated loss of information), the current trend is to use small filters (2x2 type). It is also possible to avoid the pooling layer but this implies a greater risk of over-learning.

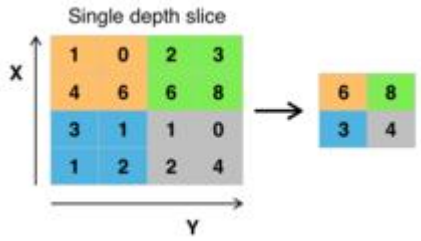



Figure 3: Pooling with a 2x2 filter and a step of 2

### 3.1.3 Correction layers (RELU)

It is possible to improve the processing efficiency by interleaving between the processing layers a layer that will operate a mathematical function (activation function) on the output signals.
The function ***ReLU*** (abbreviation for rectified linear units): **F (x) = max (0, x)**. This function forces neurons to return positive values.

### 3.1.4 Fully connected layer (FC)

After several layers of convolution and max-pooling, the high-level reasoning in the neural network is via fully connected layers. Neurons in a fully connected layer have

connections to all the outputs of the previous layer. Their activation functions can therefore be calculated with a matrix multiplication followed by a polarization shift.

#### 3.1.5 Loss Layer (LOSS)

The loss layer specifies how network training penalizes the difference between the expected and actual signal. It is normally the last layer in the network. Various loss functions suitable for different tasks can be used. The ”Softmax” function is used to calculate the probability distribution on the output classes.

### 3.2 Examples of CNN models

The most common form of a CNN architecture stacks a few Conv-ReLU layers, follows them with Pool layers, and repeats this pattern until the input is reduced in a space of a sufficiently small size. At one point, it is common to place fully connected layers (FCs). The last fully connected layer is connected to the output. Here are some common CNN architectures that follow this pattern:

- INPUT $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ FC
- INPUT $\rightarrow$ [CONV $\rightarrow$ RELU $\rightarrow$ POOL] * 2 $\rightarrow$ FC $\rightarrow$ RELU $\rightarrow$ FC. Here, there is a unique layer of CONV between each POOL layer.
- INPUT $\rightarrow$ [CONV $\rightarrow$ RELU $\rightarrow$ CONV $\rightarrow$ RELU $\rightarrow$ POOL] * 3 $\rightarrow$ [FC $\rightarrow$ RELU] * 2 $\rightarrow$ FC. Here, there are two stacked CONV layers before each POOL layer.

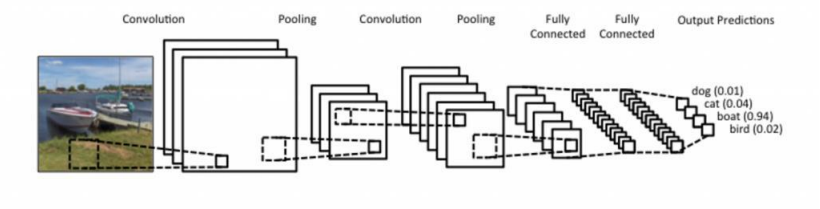



Figure 4: Examples of CNN models

### 3.3 Choice of parameters

CNNs use more parameters than a standard MLP. Even if the usual rules for learning rates and regularization constants still apply, it is necessary to take into consideration the notions of number of filters, their form and the form of max pooling.

#### 3.3.1 Number of filters

As the size of the intermediate images decreases with the depth of processing, layers near the input tend to have fewer filters while layers closer to the output may have

more. To equalize the calculation at each layer, the product of the number of features and the number of pixels processed is generally chosen to be nearly constant across the layers.To preserve the input information, it would be necessary to maintain the number of intermediate outputs (intermediate number multiplied by the number of pixel positions) to be increasing (in the broad sense) from one layer to another. the number of intermediate images directly controls the power of the system, depends on the number of available examples and the complexity of the processing.

#### 3.3.2 Filter form

Filter forms vary widely in the literature. They are usually chosen based on the dataset. The best results on MNIST images (28x28) are usually in the 5x5 range on the first layer, while natural image data sets (often with hundreds of pixels in each dimension) tend to use more large filters of first layer of 12x12, even 15x15. The challenge is therefore to find the right level of granularity so as to create abstractions at the appropriate scale and adapted to each case.

#### 3.3.3 Max Pooling Form

The typical values are 2x2. Very large input volumes may justify 4x4 pooling in the first layers. However, the choice of larger shapes will greatly reduce the size of the signal, and may result in the loss of too much information.

# 4 Some Maths: How Convolutional Neural Networks Work

In this section, we are going to learn about convolution, which is the first step in the process that convolutional neural networks undergo. We'll learn what convolution is, how it works and what elements are used in it.

## 4.1 Convolution Operation

We denote $X$ a table $n \times m$, $f$ a filter $nf \times mf$ with $nf \leq n$ , $mf \leq m$ , the convolution of $X$ by $f$ for a pixel of coordinates $i, j$ where $i \in [\![0, n\text{-}1\ ]\!]$ and $j \in [\![0, m\text{-}1\ ]\!]$ is defined by :

$$(X \otimes f) = \sum_{k=-\frac{nf}{2}}^{\frac{nf}{2}} \sum_{l=-\frac{mf}{2}}^{\frac{mf}{2}} X[i{+}k, j{+}l] \times f[k + \frac{nf}{2}, l + \frac{mf}{2}] \tag{2}$$

In the case above, if $i{+}k$ or $j + l$ comes out of the image $X$ (that is, $i{+}k \geq n$ for example) then we take $X[i{+}k, j{+}l] = 0$, we speak of "zero-padding". Other padding is possible (by taking the value of the nearest pixel for example).

We can also choose to do the calculation only on the pixels $i, j$ such that $i + k$ and $j + l$ are always in the image: there will be then reduction of the output dimension.

More visually:

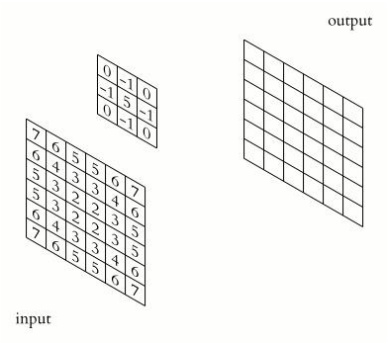



Figure 5: Convolution Operation

Effect of convolutions, example on images:

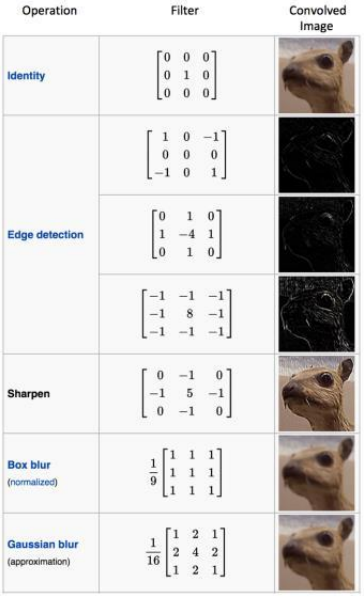



Figure 6: Image processing

***N.B*** : **A Convolution Neuron**

We do not choose the **filters**, we learn them !!! These are **trainable parameters of the network**.

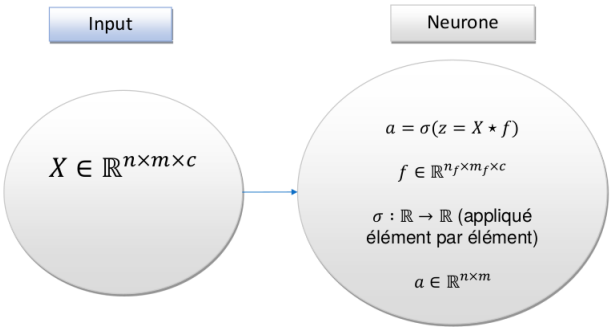
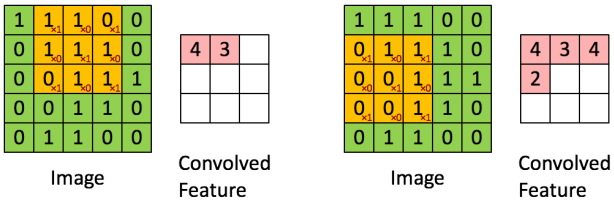


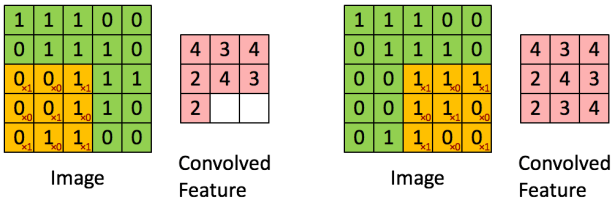

Figure 7: A convolution Neuron

$c$ is called number of channels :

- $c = 1$ for a grayscale image
- $c = 3$ for a RGB image (one channel per color)
- In the intermediate layers, $c$ corresponds to the number of neurons of the previous layer.




(a) ConvStep1 (b) ConStep3




(c) ConvStep6 (d) ConvStep8

Figure 8: Convolution With Filter 2*2

## 4.2 Activation Function : ReLU

In practice, at the end of a convolution, an activation function is applied (in general **RELU**: Rectified Linear Unit): *f (x) = max (0, x)*.

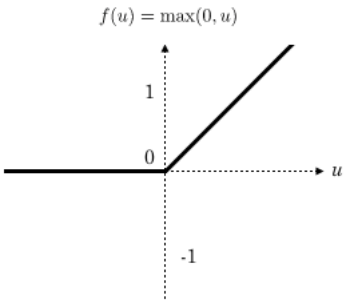


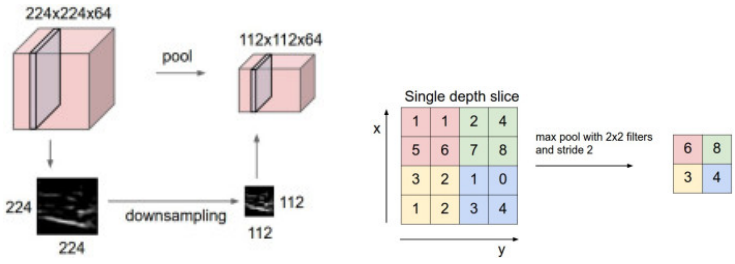

Figure 9: Look of ReLU function

The ReLU correction layer therefore replaces all negative values received as inputs with zeros. It plays the role of activation function.

## 4.3 Pooling: operation used to reduce the size

A convolution layer can also be followed by a ***pooling*** layer. Pooling (or sometimes called subsampling) layer make the CNN a little bit translation invariant in terms of the convolution output. The purpose of pooling is to reduce the size of the "images", looking for more 'rude' details and bigger 'structures' in the image but also to overcome the phenomenon of "overfitting".

Just as for convolution we apply a filter that we drag on the image and (in the "max pooling" version) we keep the max value on each window. We also talk about "Down-Sampling" or "Sub-Sampling"




(a) Down Sampling (b) Pooling with filter 2*2

Figure 10: Pooling Operation

- **Max-pooling of size *l*:** we take the maximum element of each sub-array of size *l*

- **Sum-pooling of size *l*:** we sum up all the elements of each sub-table of size $l$

What the Max-pooling operation does ?.

The max-pooling operation, at a given position, outputs the maximum value of the input, that falls within the kernel. So mathematically,

$$h_{i,j} = max(X[i+k, j+l-1] \; \forall 1 \leq k \leq n, 1 \leq l \leq m) \tag{3}$$

# 5 Implementing and Running a CNN with Keras

Here we will briefly discuss how to implement a CNN. Knowing the basics is not enough, we also should understand how to implement a model using a standard deep learning library like *Keras* [5]. Keras is a wonderful tool, especially to quickly prototype models, to see them in action.

First we define the Keras API we want to use. We will go with the sequential API.

Listing 1: Define a sequential model

```
model = Sequential()
```

Then we define a convolution layer as follows:

Listing 2: Added a convolution layer

```
model.add(Conv2D(32, (3,3), activation='relu', input_shape=[28, 28,1]))
```

Here, 32 is the number of kernels in the layer, (3,3) is the kernel size (height and width) of the convolution layer. We use the non-linear activation Relu and the input shape [28, 28, 1] which is [image height, image width, color channels]. Note that the input shape should be the shape of the output produced by the previous layer. So for the first convolution layer we have the actual data input. For the rest of layer, it will be the output produced by previous layer. Next we discuss how to implement a max pooling layer:

Listing 3: Add a max pool layer

```
model.add(MaxPool2D())
```

Here we don't provide any parameters as we're going to use default values provided in Keras. If you do not specify the argument, Keras will use a kernel size of (2,2) and a stride of (2,2). Next we define the fully-connected layers. However before that we need to flatten our output, as the fully connected layers process 1D data:

Listing 4: FC

```
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

[5] https://keras.io/

Here we define two fully-connected or dense layers. The first fully connected layer has 256 neurons and uses Relu activation. Finally we define a dense layer with 10 output nodes with softmax activation. This acts as the output layer, that will activate a particular neuron for images having the same object. Finally we compile our model with,

Listing 5: Compiling

```
model.compile(optimizer='adam', loss='categorical_crossentropy',
 metrics=['accuracy'])
```

Here we say use the Adam optimiser (to train the model), with the cross entropy loss and use the accuracy of the model to evaluate the model. Finally we can train and test our model using data. We are going to use *MNIST dataset* [6]. The MNIST dataset contains images of hand written digits (0–9) and the objective is to classify the images correctly by assigning the digit, that the image represents. Next we train our model by calling the following function:

Listing 6: Training

```
model.fit(x_train, y_train, batch_size = batch_size)
```

And we can test our model with some test data as below:

Listing 7: Testing

```
test_acc = model.evaluate(x_test, y_test, batch_size=batch_size)
```

We will run this for several epochs, and this will allow you to increase your models performance.

# References


[1] LeCun, Yann, et al. *"Gradient-based learning applied to document recognition."* . Proceedings of the IEEE 86.11 (1998): 2278-2324.

[2] Zeiler, Matthew D., et al. *"Deconvolutional networks."* Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[3] Représentation d'un réseau de neurones et des couches intermédiaires sur les chiffres MNIST,
`http://scs.ryerson.ca/ aharley/vis/conv/flat.html`


[6] http://yann.lecun.com/exdb/mnist/