Visualizing and Understanding Convolutional Networks

Othmane Aouassar [1], Nisrine Madhar [1], Gea Romain [1]

[1]Institut National des sciences appliquées, Applied mathematics departement

{aouassar,madhar,gea}@etud.insa-toulouse.fr

May 30, 2019

Abstract

Convolution Neural networks have proved a high capacity for image recognition and classification. As yet, scientists have not been able either to explain their performance or to give possible methods for improvement. Our project focuses on the first of these two issues. For a given neural network using convolution, we decided to study the output of every single layer to understand what features are caught, revealing what features the layer is able to recognize. We first started with a basic autoencoder model to familiarize ourselves with several manipulations and then we moved to a more complex model, LeNet5. We also examined the output produced with varying parameters such as the activation and optimizing function to determine which one the layer uses to produce a high quality output. This will help to understand the purpose of every layer, which may lead to an optimal use of these layers and their parameters.

Acknowledgement

First and foremost, we would like to express our thanks of gratitude to our supervisor Mr. Guillouet, not only for giving us the chance to work on this project, but also for his support throughout the semester.

We would like to thanks the researchers team (Mr. Besse Mr.Guillouet) that made accessible considerable resources through Wikistat.

1 Introduction

During the past decade, Convolution neural networks have been very widely in image recognition, and have proved exceedingly efficient. However scientists haven't been able to understand this concept fully, or to come up with features or concepts that could help in understanding it or improving it. Therefore, the aim of our project is to try to find out what is actually behind the CNN. We have chosen to work on MNIST data, a database of handwritten digits, because it is accessible. Plus, it is a huge database with 70,000 images available. To this end, we started by first studying accessible models that rely on convolution layers to develop our familiarity with the different libraries, functions and parameters, also, to get a concrete view of the construction and outputs of such neural networks. Then, we analyzed the intermediate outputs of these models more deeply to try to get an explanation of how these neural networks work and how these stacking of layers are able to draw meaningful conclusions. In order to define the specialization of each filter, we began with a first method which aims to define the active filters first on the integral data and then on segregated data. Finally, so as to be able to visualize the features caught by the layers, we applied deconvolution to the outputs.

2 Building blocks

2.1 Neural networks

The first neural network algorithm appeared in 1958, created by Franck Rosenblatt. Originally, the neural network is very simple, it has only one output, which is binary and all inputs are connected. Today's neural networks are much more complex, due to the addition of floating inputs, and additions of parameters such as the notion of bias. The most famous neural network is the multilayer perceptron, which is an artificial system that learns through experience. It is organized in 3 different layers: the entrance, the intermediate layer which is normally not visible, and the exit. We will focus on this intermediate layer which is the core of the neural network. To facilitate the understanding of its functioning, we can start from a basic example:

Let x_0, x_1 and x_2, three different input signals. They are then transmitted to the heart of the network, weighted respectively by w_0, w_1 and w_2. These weights have a main role because they allow adaption throughout the learning, and we note that $\forall i, x_i$ is in [0,1] or [-1,1]. The sum of all these weighted signals is then calculated, and the bias b is added which will allow the activation function (function that will be defined later) to make a better learning. This activation function is then applied to obtain our output signal. In brief :

- **Step 1 :** Input : x_0, x_1, x_2
- **Step 2 :** Middle layer : $\sum_i w_i x_i + b$
- **Step 3 :** Output : $y = f_{activation}(\sum_i w_i x_i + b)$

We have defined the structure of the neural network. Now, we must detail the notion of learning. We have seen that the only variables in the perceptron are actually the weights and the bias. Updating these weights represents a large proportion of the work necessary to make the model fit the data : this is called gradient descent. For that, let's take a propagation, analyze the error committed by the algorithm by evaluating the impact that each weight had on the error, and increase or decrease this weight in order to obtain a better answer : one speaks here of backpropagation of the gradient.

The perceptron can be seen as a system of linear equations whose variables are the weights and the bias. To find the minimum error taking into account all the weights, we need to derive this system according to the weights.

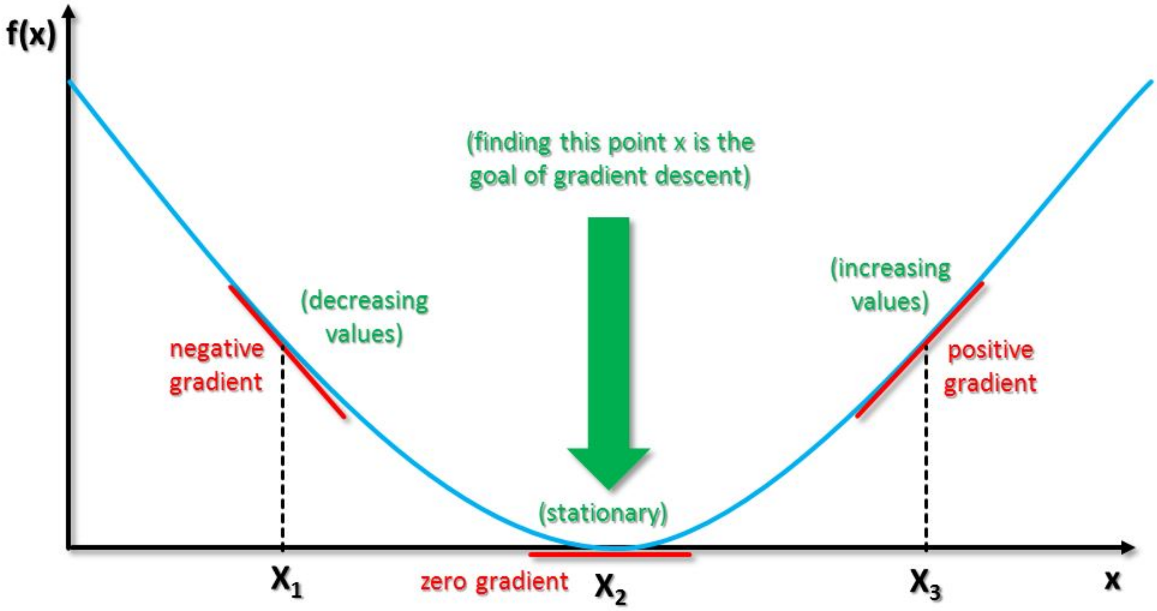


Figure 1: Backward propagation of the gradient

By doing a partial derivative of the error with respect to a weight, we obtain the value of the slope of the dependence of our system with respect to this variable. If it is positive, it will necessarily go downhill in order to reduce the error, whereas if it is negative, it will necessary increase to get closer to the correct curve. This method is visible in figure. 1. Thus, the gradient descent makes it possible to evaluate the impact of a weight on the error made and therefore to adapt it to provide a better output response.

To compute the error resulting from our algorithm, we must compare the answer obtained with the expected answer, so we need a loss function. For instance, if the neural network output is response vector $y = (y_1, ..., y_n)$, it must be compared to the expected theoretical vector. To compute this error, it is sufficient to use the norm L_2 of distance between the vectors.

However, it is important not to overload the network with layers or variables, as this may lead to over-learning. The answer obtained would then be very close to reality, but the algorithm would not have enough freedom to perform well on other data sets. Therefore, we need to find a balance between how many unknown parameters we provide and the quality of the desired response.

2.2 Convolutionnal networks

A convolutional network is an algorithm based on deep learning, which takes as a parameter an image bank on which it trains, then reproduce the same processes on images that it does not know.

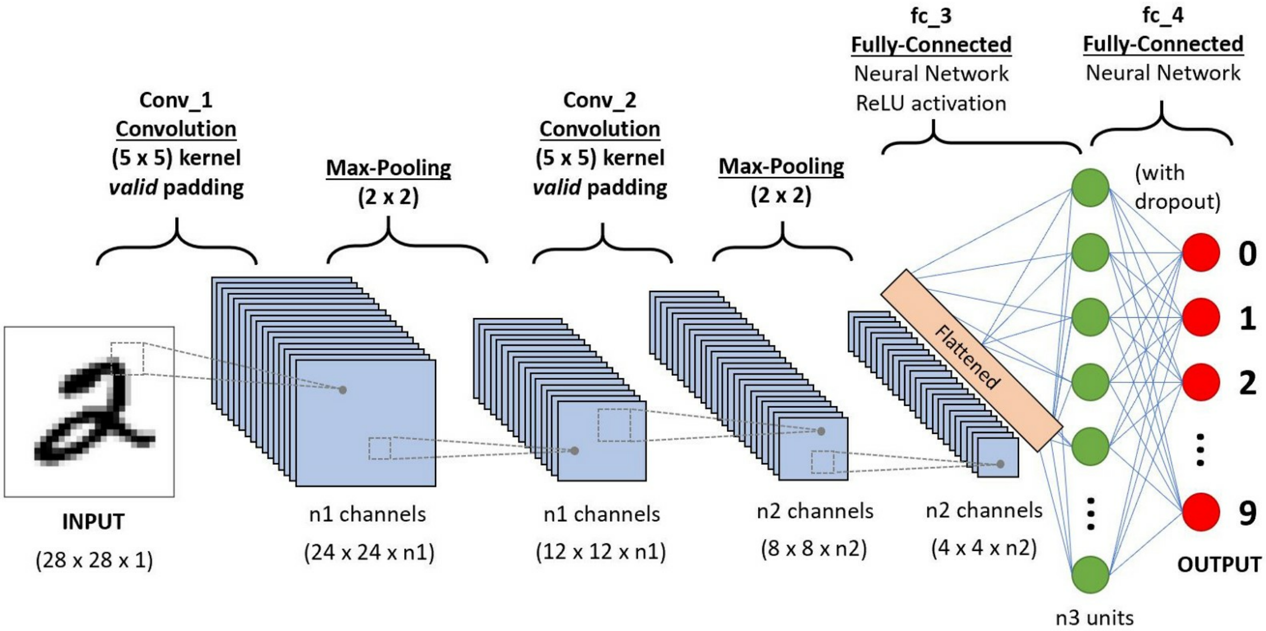


Figure 2: A CNN sequence to classify handwritten digits, from [4]

Convolutional networks are widely used in image analysis and recognition. The two main characteristics of convolution networks are the use of pattern filters and the implementation of pooling. At the entrance of this special neuron network, an image is provided (which can be seen as a matrix), it's also composed of filters that will help analyze the image. Each filter will help detecting a particular pattern on the input image.

Max-Pooling is a method that reduces the size of an image, keeping only the largest and most representative pixels in the image.
A convolution layer is characterized by three parameters: *depth*, *pitch*, and *margin*.

- The depth of the layer counts the number of neurons associated with a filter.
- The step controls the overlap of the receiver fields.
- The margin that is characterized by the zero padding allows adding zeros around the output image to obtain an output of the same size as the input.

In a convolution network, it is not uncommon to interpose pooling layers between two convolution layers in order to avoid over-learning. Pooling also allows us to gain in computing power. However, it is also possible to use another pooling function rather than maxpooling. This technique is effective on banks of data like MNIST's because the images are similar and this pooling adapts to the elastic deformations of the images.

2.3 Some vocabulary

- **Activation function:** this notion came from biology more precisely from the communication between two biological neurons, it is a kind of decision rule that defines whether the signal which a neuron want to convey to an other neuron is worth transmitting to the other neuron. We can distinguish two major sorts of activation function, linear and non linear functions. The latter are the most used.

  Here is an example of activation function :

  One of the most famous activation functions is the **Heaviside function** of the form

  $$\forall x \in \mathbb{R}, H(x) = \begin{cases} 0 \text{ if } x < 0 \\ 1 \text{ if } x \geqslant 0 \end{cases}$$

  ReLU =

  $$max(0, x)$$

  it returns 0 if x is negative and x otherwise. The special feature of this activation function is that it can avoid saturation.

  In fact, other functions can also be used to enhance the nonlinear properties of the network, such as the hyperbolic tangent function.

- **Pooling :** this an important concept in convolution neural networks, which is actually a form of down sampling. There are many different forms of nonlinear pooling functions, and "Max pooling" is the most common. It divides the input image into several rectangular areas and outputs a maximum value for each sub-area. Here is a clear approach on how Maxpooling works :

  Maxpooling : For each 2x2 size patch of the input (in this case) this layer substitutes the input patch with its maximum in the output.

  In addition to the maximum pooling, the pooling layer can also use other pooling functions, such as "average pooling" or "L2-norm pooling". In the past, the use of average pooling was more extensive, but recently average pooling has been less common because the largest pooling performs better in practice.

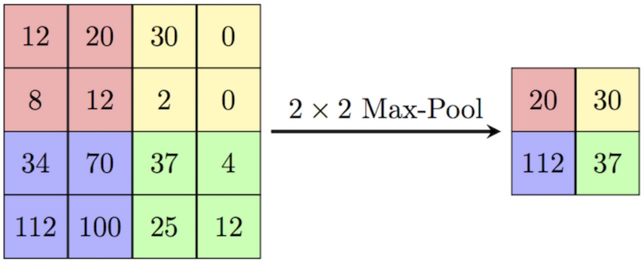


Figure 3: MaxPooling

- **The loss function layer :** is used to determine how the training process "punishes" the difference between the network's predictions and the real results. It is usually the last layer of the network. A variety of different loss functions are available for different types of tasks. For example, the Softmax cross entropy loss function is often used to select one of the K categories, while the Sigmoid cross entropy loss function is often used for multiple independent two classification problems. Euclidean loss function is often used for the problem that the value range is any real number

  Softmax(x) =

  $$\frac{e^{x_j}}{\sum_j e^{x_j}}$$

  the x_j refers to the weights of the neuron. We give to this function a vector and as output we get a vector where each represents the probability that our initial input belongs to the i category.

  Categorical crossentropy : is commonly used in categorical classification, it evaluates between two probability distributions (that of expected values and that of the predicted values).

- ***Convolution (Conv2D)*** : the aim of this layer is to apply a filter/kernel to an input image, the characteristics of this filter are given as parameters, here we have a kernel as 3x3 matrix.

- ***Dense***: this layer consists of a linear operation in which every input is connected to every output by a weight, which is why it is also called fully connected layer.

- ***Dropout*** : this layer shuts down a ratio of neurons, 25% in the present case so as to avoid over fitting phenomena.

- ***Flatten*** : this layer reshapes a matrix into a vector.

3 Methods

3.1 Model Set up

Training data

As mentioned before, we worked on MNIST data, that is to say that our neural network has been trained and tested on this kind of data. In MNIST database each images is stored as a vector of shape 784. This format is not suitable for a 2D-convolution layer, which is why we have chosen to reshape the image into shape 28×28. Moreover since our data are black and white images, the data images are of shape $28 \times 28 \times 1$.

To be able to use our model we first need to make it train on some data and then to test its abilities on a test data different from the data it has been trained on. This is why we have two data-samples.

The first is our training data which is a sample of 60,000 black and white images of digits of size $28 \times 28 \times 1$, and the second on is a sample of 10,000 of black and white images of digits of the same size..

Model

The following model corresponds to the model suggested in [1]

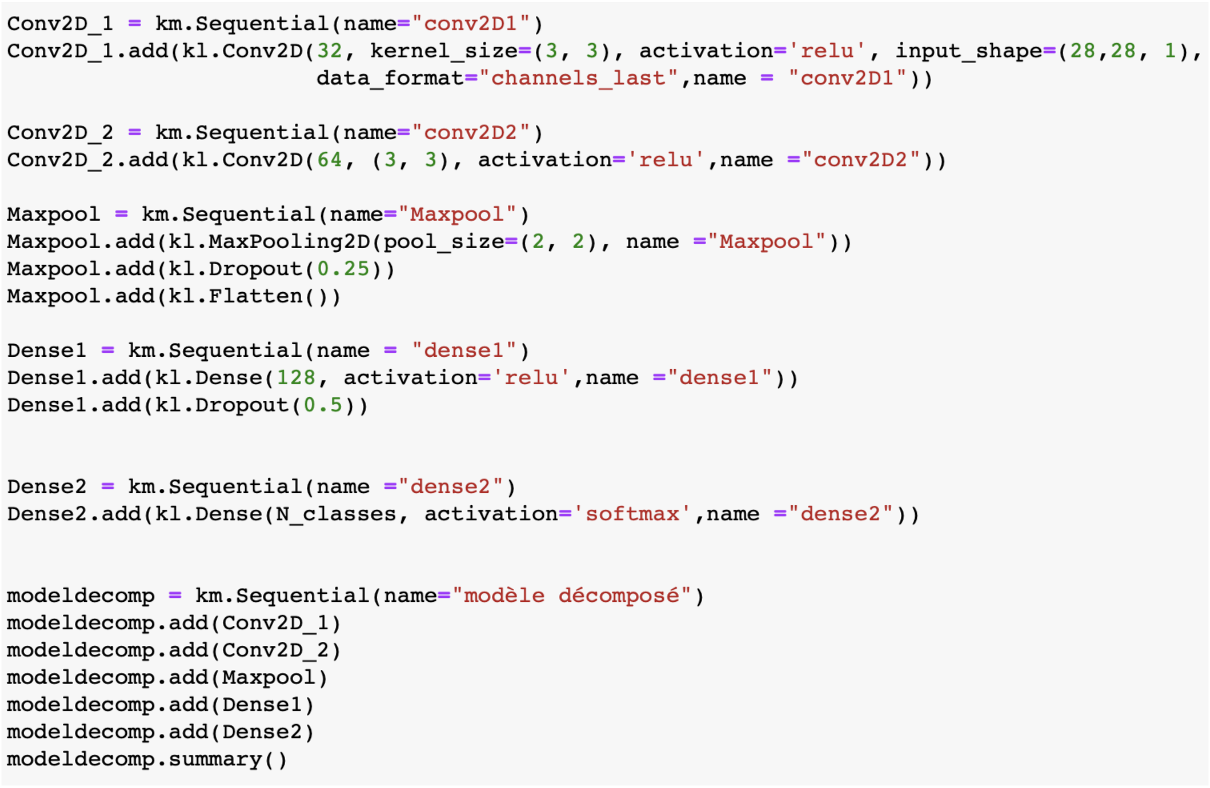
The model we are considering has 8 layers, (the utility of each one of them has been introduced in the previous section with respect to the definitions in [5]. Our neural network follows this pattern :

- A convolution layer
- A second convolution layer
- A max pooling layer
- A dropout layer where 25% of neurons are set to zero.
- A flatten layer
- A dense layer
- A dropout layer where 50% of neurons are set to zero
- A second dense layer

This model aims to classify MNIST data. In other words, for a given image of digit the model is able to say to which digit it corresponds.

As you can see on the figure 4, each layer has its own options, which gives us an insight into the methods and function considered.

Here, we provide some information on how to compute the numbers of parameters involved in each layer. And also, we explain why the input and output shape of each layer changes. We will give the techniques for the main layers, since the idea is the same for the remaining layers. *We will mainly focus on convolution and dense layers.*



```
Conv2D_1 = km.Sequential(name="conv2D1")
Conv2D_1.add(kl.Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,28, 1),
                       data_format="channels_last",name = "conv2D1"))

Conv2D_2 = km.Sequential(name="conv2D2")
Conv2D_2.add(kl.Conv2D(64, (3, 3), activation='relu',name ="conv2D2"))

Maxpool = km.Sequential(name="Maxpool")
Maxpool.add(kl.MaxPooling2D(pool_size=(2, 2), name ="Maxpool"))
Maxpool.add(kl.Dropout(0.25))
Maxpool.add(kl.Flatten())

Dense1 = km.Sequential(name = "dense1")
Dense1.add(kl.Dense(128, activation='relu',name ="dense1"))
Dense1.add(kl.Dropout(0.5))


Dense2 = km.Sequential(name ="dense2")
Dense2.add(kl.Dense(N_classes, activation='softmax',name ="dense2"))


modeldecomp = km.Sequential(name="modèle décomposé")
modeldecomp.add(Conv2D_1)
modeldecomp.add(Conv2D_2)
modeldecomp.add(Maxpool)
modeldecomp.add(Dense1)
modeldecomp.add(Dense2)
modeldecomp.summary()
```

Figure 4: Model

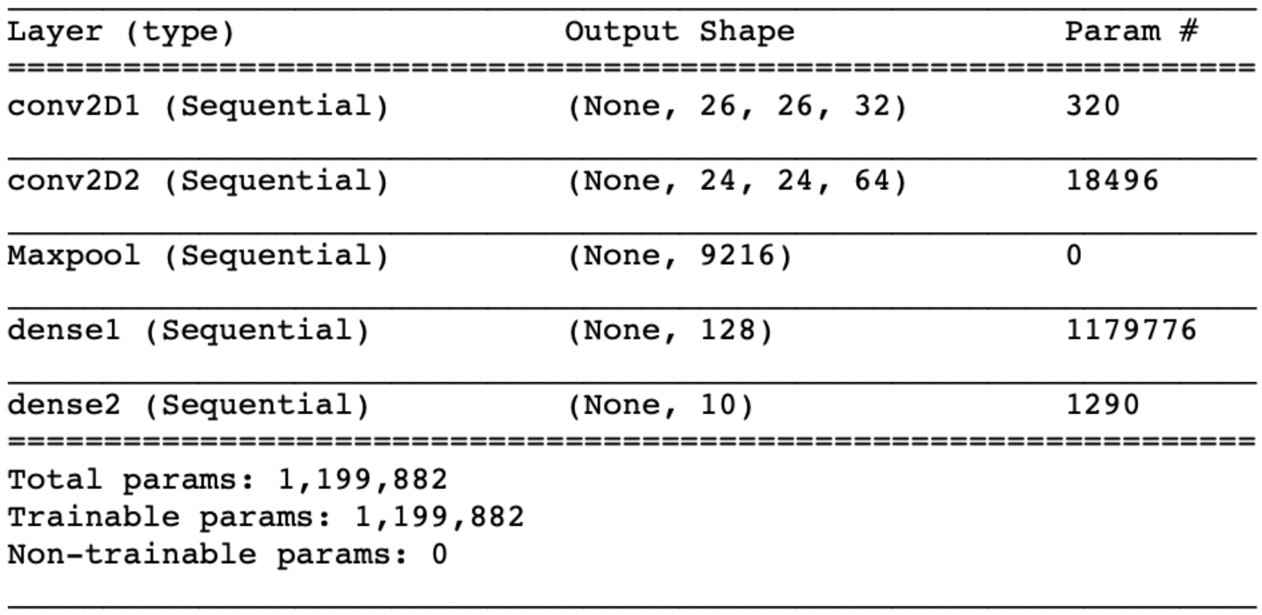
The first layer (which is a 2-D convolution layer) takes as an input the initial image which is stored as matrix of shape 28 × 28 with one channel since the images we are considering are only black and white.
The output shape and the parameters involved in each layer are given is the summary table below.

As we can see on the summary table for each image given to this first layer 32 output of size 26 × 26 are produced. In other words, we obtain an image of shape 26 × 26 with 32 channel which is rewritten as 26 × 26 × 32. The reduction of the image shape is due to the filter size we have considered (3 × 3), whereas the 32 filters we obtain are due to the option we specified during the model construction. This first layer has 32 filters and each one is applied to the input image.
To sum-up for each image given to this layer, we get 32 outputs since the image input will be going through 32 filters : the layer produces 32 outputs of size 26 × 26, for each image.

Parameters computation [3]

Convolution layer's parameters

The number of parameters stands for the number of neurons involved in the layers. Hence our first layer which is a convolution layer has 320 parameters. We can compute it manually



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2D1 (Sequential)         (None, 26, 26, 32)        320
_________________________________________________________________
conv2D2 (Sequential)         (None, 24, 24, 64)        18496
_________________________________________________________________
Maxpool (Sequential)         (None, 9216)              0
_________________________________________________________________
dense1 (Sequential)          (None, 128)               1179776
_________________________________________________________________
dense2 (Sequential)          (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

Figure 5: Summary Table

using the following formula :

$$\# \text{ parameters} = \text{output\_channels} \cdot (\text{input\_channels} \cdot \text{filter\_size} + 1).$$

In the present case the output channels is 32, the input channels is 1 and the filter size here is 3×3. Therefore, when we apply this formula we get :

$$\begin{aligned} \# \text{ parameters} &= 32 \times (1 \times (3 \times 3) + 1) \\ &= 320 \end{aligned}$$

Which is exactly the number of parameters shown in the summary table above.

Dense layer parameters

For the dense layer the number of parameters is computed using :

$$\# \text{ parameters} = \text{output\_size} \cdot (\text{input\_size} + 1)$$

If we consider the dense layer of our model, the input shape is the output size of the previous layer 9216 and the output size of this dense layer is 128 :

$$\begin{aligned} \# \text{ parameters} &= 128 \cdot (9216 + 1) \\ &= 1,179,776 \end{aligned}$$

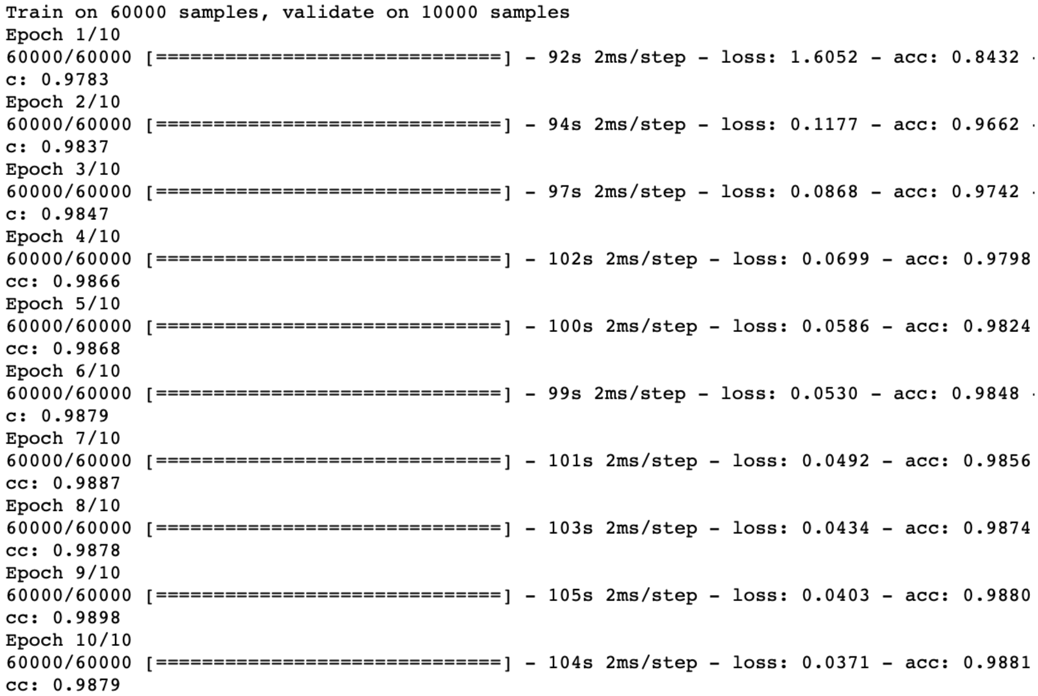
Remark: *We have chosen to decompose the model to facilitate the intermediate output treatment.*

Model's performance

Before proceeding further, we had to make sure that our model was performing successfully. So we used two indicators that allow us to asses the performance of our model.

On the one hand we have the classification error which is computed with respect to the loss function : we expect this value to decrease through the training or the testing session. And on the other hand, we have the validation score, which is expected to increase through the training or the testing session. We chose to examine these indicators for each epoch and for both the training and the testing session, to observe their behaviour throughout the epochs.

As you can see on the table below, as expected the loss value drops significantly from 1.6 to 0.0371, whereas the accuracy value increases throughout the epochs from 0.8432 to 0.9881 which is satisfying.

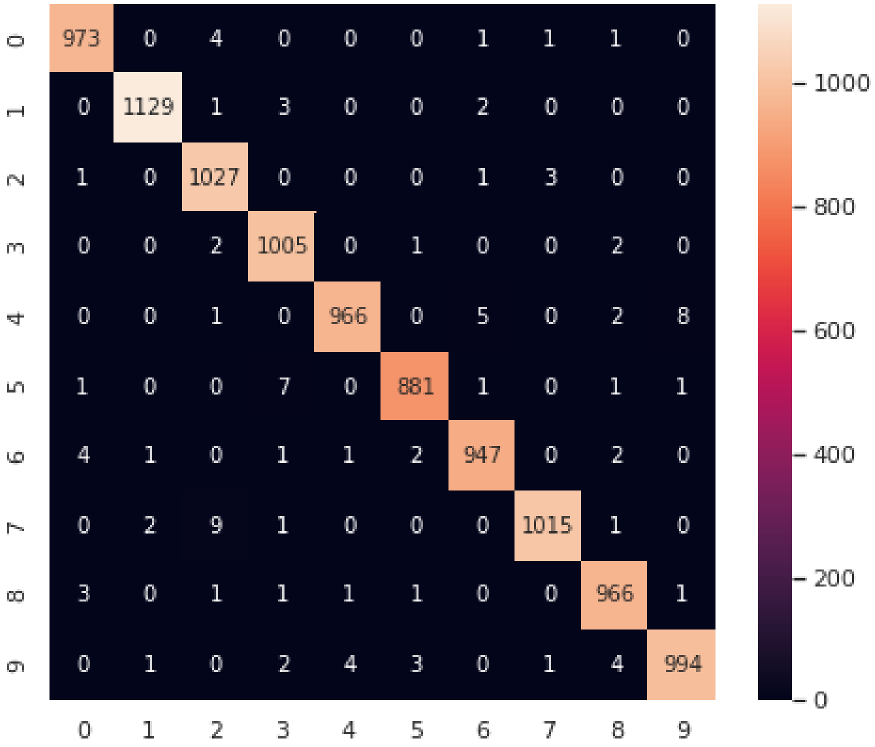


```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 92s 2ms/step - loss: 1.6052 - acc: 0.8432 ·
c: 0.9783
Epoch 2/10
60000/60000 [==============================] - 94s 2ms/step - loss: 0.1177 - acc: 0.9662 ·
c: 0.9837
Epoch 3/10
60000/60000 [==============================] - 97s 2ms/step - loss: 0.0868 - acc: 0.9742 ·
c: 0.9847
Epoch 4/10
60000/60000 [==============================] - 102s 2ms/step - loss: 0.0699 - acc: 0.9798
cc: 0.9866
Epoch 5/10
60000/60000 [==============================] - 100s 2ms/step - loss: 0.0586 - acc: 0.9824
cc: 0.9868
Epoch 6/10
60000/60000 [==============================] - 99s 2ms/step - loss: 0.0530 - acc: 0.9848 ·
c: 0.9879
Epoch 7/10
60000/60000 [==============================] - 101s 2ms/step - loss: 0.0492 - acc: 0.9856
cc: 0.9887
Epoch 8/10
60000/60000 [==============================] - 103s 2ms/step - loss: 0.0434 - acc: 0.9874
cc: 0.9878
Epoch 9/10
60000/60000 [==============================] - 105s 2ms/step - loss: 0.0403 - acc: 0.9880
cc: 0.9898
Epoch 10/10
60000/60000 [==============================] - 104s 2ms/step - loss: 0.0371 - acc: 0.9881
cc: 0.9879
```

Figure 6: Loss and accuracy values throughout training.

Another way to verify that the classification is working well is to consider the confusion matrix. The latter corresponds to a matrix such that its cell ($C_{i,j}$) contains the number of observations known to be in group i but predicted to be in group j. Thus the coefficients on its diagonal represent the number of elements the model has managed to predict exactly. A good model should have a confusion matrix with only coefficients on the diagonal and few numbers elsewhere.

The confusion matrix of our model is given in the figure below. As you might notice, our model manages to predict well all of the digits since the confusion matrix has large number in the range of 1,000 and negligible numbers elsewhere.



	0	1	2	3	4	5	6	7	8	9
0	973	0	4	0	0	0	1	1	1	0
1	0	1129	1	3	0	0	2	0	0	0
2	1	0	1027	0	0	0	1	3	0	0
3	0	0	2	1005	0	1	0	0	2	0
4	0	0	1	0	966	0	5	0	2	8
5	1	0	0	7	0	881	1	0	1	1
6	4	1	0	1	1	2	947	0	2	0
7	0	2	9	1	0	0	0	1015	1	0
8	3	0	1	1	1	1	0	0	966	1
9	0	1	0	2	4	3	0	1	4	994

Figure 7: Confusion matrix.

3.2 Filters specializations

After becoming familiar with the model and its parameters, we developed some algorithms that aim to identify which filters are activated. By an active filter, we mean that the filter is able to catch some interesting features. Therefore, we say that a filter active if the norm of its output image is greater than a given level. We have chosen this level as the average norm. For instance filters 0,4,7 are active:

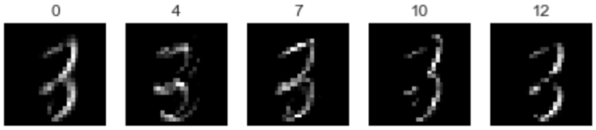


Figure 8: Output of active filters.

3.2.1 Segregated data

For each digit, using the algorithm *Segregation_ label* we isolate the images with the same label. This way, we obtained 10 sub-samples, one sample of images for each digit. For each sample and for each filter we compute the norm of the output and the average norm for this filter and for this digit. Then, using our algorithm *selection_4* we only select the index of a filter with a norm higher than average. We repeat this for all the images in the sample, obtaining for each image of the same label the filters that have been activated. Our algorithm *conversion* enables us to count the number of occurrences of each activated filter. In the figure 3 below, we can see for images with label 0 for example that a wide number of filters have never been activated (number of occurrences equal to 0) such as filter number 6,13,14,22,23 and others. The same conclusions are drawn for the remaining labels, but this method did not allow us to figure out what features the filters are able to catch.

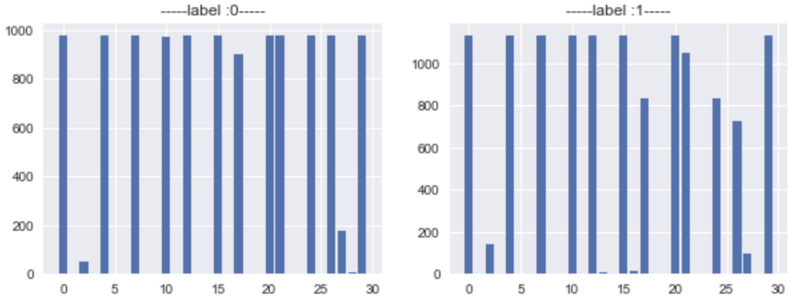


Figure 9: Occurence of active filters for images with label 0 and 1.

3.2.2 Data completeness

In this section, we expand the idea of active filter mentioned before to the integral data. The logic behind the selection of active filter is the same except that this time the level is the average norm on all the output of all the images. Then for given filter we count how much times the filter have been activated for each label. In the figure 10 below we observe filter 2 and 4 have never been activated as stated in the section before. Meanwhile filter number 1 has been activated for all the labels.

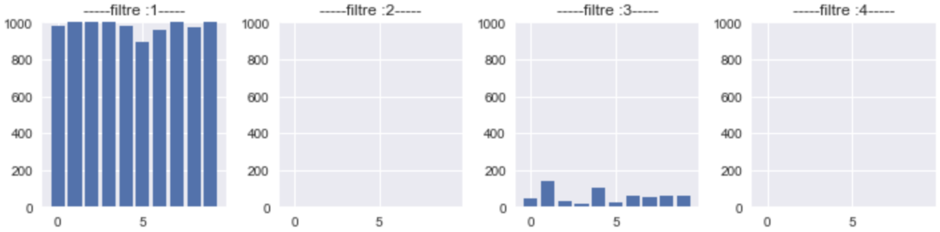


Figure 10: For filter 1,2,3,4 the labels for which these filters have been activated.

3.2.3 Distribution of active filter

As we have mentioned earlier, the limit we set to say that a filter is active or not is the mean but we are not sure of the relevance of this decision-making rule. So, we decided to plot the distribution of the norm of each filter using the boxplot as graphic representation.

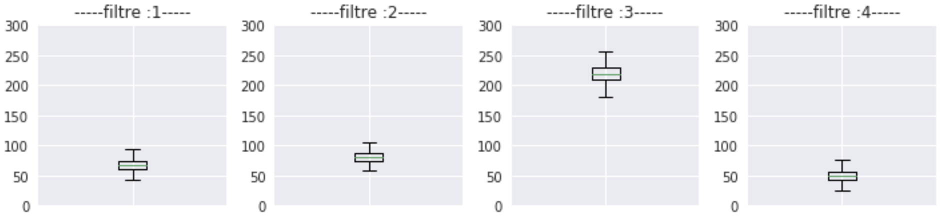


Figure 11: Filter norm distribution

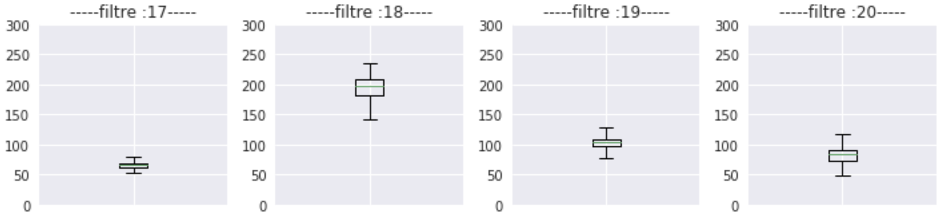


Figure 12: Filter norm distribution

For the following boxplots, we worked on a sample composed by only images with label= 0,(representing a zero). For instance, for filter 1 we have computed the filter norm for all the images, using this value we were able to plot a boxplot, we iterate this procedure for all filters and later on on all the labels.
Comments : We only represent some of filters, we chose to represent the different boxplots using the same scale, to be able to see them all.
We can see from both figures 11 and 12 that the distribution of filter's norm is symetric. In fact, it is the same that we have the number of image above and below the median (green line). Hence, we can say the deciding-rule we have set earlier is acceptable.

As you might have noticed, the previous methods did not allow us to show which features are caught by the different filters.
In order to gain a clearer view of the features caught by the filters, in the next section we will introduce the deconvolution concept that will enable us to reach our aim.

3.3 Deconvolution

Until now, we have worked only on the convolutional part and presented the main ideas on how the convolutional networks work in order to obtain an output classification of the input data.
In this part, we will focus especially on the filters of the convolutional network, in order to understand how they work and how they can help the neural networks to deduce the classification of the data.
Also, we assume that the convolution operation done using different filters corresponds to a linear application, the corresponding matrix is orthogonal. We will then be using

the opposite operation of the convolution, which is the deconvolution. The motivation for this deconvolutional operation is that we hope that the filters learned by the convolutional network will detect some specific patterns on the data, which would helps us have a better understanding of the behaviour of each filter.
To do this analysis, we decided to visualize what exactly each filter recognizes on the input image, and we took several examples. Let us explain further how exactly this operation will be done depending on which layer we are working on:
For the first layer, composed of 32 layers, the deconvolution operation will not be difficult to apply. Actually, since the shape of each picture on the data base is (28,28,1), that means that the input image has only one channel (the depth of the image), then the filters of this layer will have the same number of channels, more specifically, the shape of each filter on this first layer is (3,3,1). As a result, the output of this layer has the following shape: (26,26,32). Indeed, this output is the result of a concatenation, or by abuse of language a superposition, of the convolution operation between the input data and all the 32 filters of the layer.

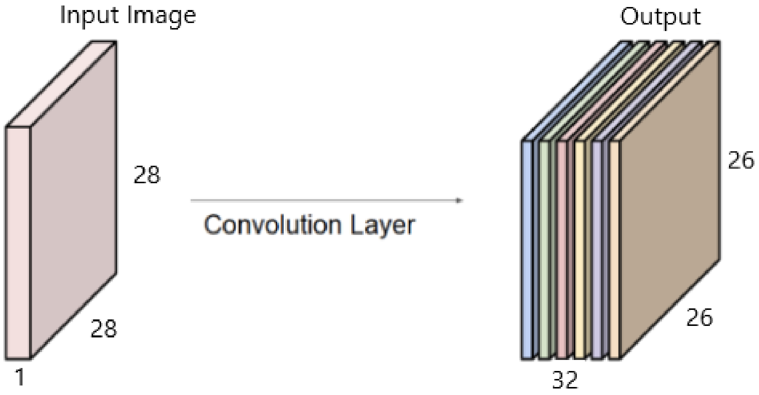


Figure 13: output shape of the first layer

Thus, the deconvolution operation corresponds to the convolution of the output using the transpose of the filters.
Let's take the example of the 13th layer: in order to visualize the detected patterns, we need to apply a normal convolution of the image corresponding to the 13th channel of the output, with the transpose of this filter.
Concerning the second layer, the corresponding deconvolution operation relays on the same principal as the previous idea, but because of some technical limitations, this has been more complicated to set up in practice. We decided then to work on Autoencoder

3.4 Autoencoder

Using to the tutorial provided by [2], we were able to understand this concept.
Autoencoder is an data compressing algorithm, which has three interesting characteristics :

- ***Specific to data*** : This algorithm will compress data that have the same characteristics as the data they have been training on.

- ***Loss***: Like other compressing algotithms the output does have not the same quality as the inputs. This is why this algotrithm is said to be lossy.

- ***Neural network*** : The algorithm will be learned automatically using training and test data.

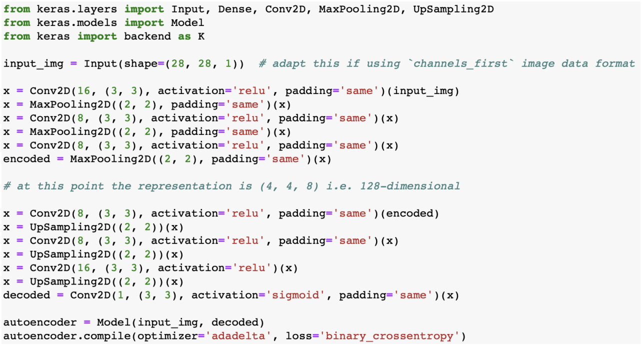
An Autoencoder is made up of two main components :

- ***Encoder*** : This part will be in charge of compressing the input into a latent space. This space will catch the main features of the treated data.

- ***Decoder***: This part will try to construct the input using the encoder's output.

Note that the encoder and the decoder are neural networks.

In the figure 14. below we can see how we build the auto encoder using convolution layers.



```
from keras.layers import Input, Dense, Conv2D, MaxPooling2D, UpSampling2D
from keras.models import Model
from keras import backend as K

input_img = Input(shape=(28, 28, 1))  # adapt this if using `channels_first` image data format

x = Conv2D(16, (3, 3), activation='relu', padding='same')(input_img)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = MaxPooling2D((2, 2), padding='same')(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2, 2), padding='same')(x)

# at this point the representation is (4, 4, 8) i.e. 128-dimensional

x = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)
x = UpSampling2D((2, 2))(x)
x = Conv2D(8, (3, 3), activation='relu', padding='same')(x)
x = UpSampling2D((2, 2))(x)
x = Conv2D(16, (3, 3), activation='relu')(x)
x = UpSampling2D((2, 2))(x)
decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(x)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy')
```

Figure 14: Construction of Autoencoder

After training the model on some training data, we obtain the following results :

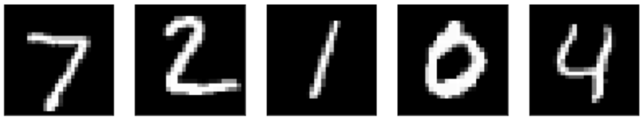


Figure 15: Input

As you can see the decoder's outputs came close to the initial output. Nevertheless, we notice that we have lost quality as expected.

Next, it was important for us to study the principal characteristics of our filters we had. We aim to classify the filters according to their elements of recognition. Thus, once we had studied the active or inactive filters for some images in our MNIST database, we decided to put in parallel the activation of the filters and the response to these filters by the process of deconvolution that the we presented in the previous section. This allows for example to show that some filters capture the horizontal elements, others vertical, oblique.

Figure 16: Encoder outputs

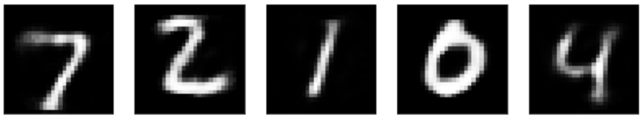


Figure 17: Decoder outputs

4 Results and Analysis: filters features

4.1 Deconvolution and Active filters

We noticed a kind of incoherence between the results provided by the active filter method and deconvolution results for the first convolution layer, to the extent that for label 3, for instance, the active filters methods states that filter 0 is seldom used therefore we deduce that they do not carry interesting information. But while using the deconvolution we realize that we are able to construct an image on which we can clearly distinguish the digit « 3 » . This leads us to two conclusions : Either the decision-making rule we have set to say if the filter is active or not is not efficient and should be replaced, the amount of information gathered by the filter does not go hand by hand with its quality. Hence, if the information caught by the filter is not important in terms of quantity (but highly relevant), we will be able to reconstruct the number using this filter easily.

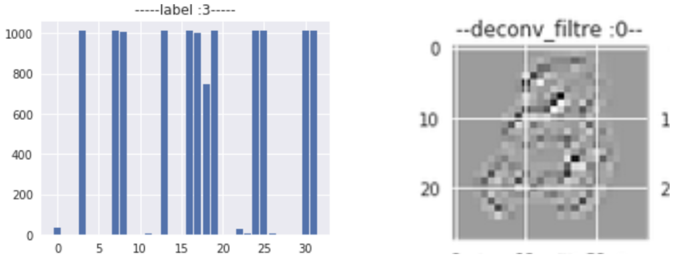


Figure 18: Active filters for label 3 (left), Deconvolution output for filter 0 (right).

Paradoxically, the filters that were told active a several times (more than 1000 times), do carry information but this information is of poor quality. In fact, the deconvolution output shows that the information carried by this filter neither useful to recognize interesting features nor useful in recognising the whole information contained in the number.

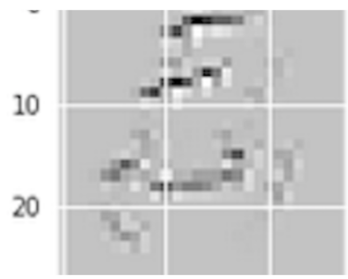


Figure 19: Deconvolution output for filter 23.

However, we have notice that among the active filters, some are able to caught special features. For instance, the filter 18 seem to focus and is able to recognize horizontal lines. We can observe this on the deconvolution of this filter applied to images of digits with horizontal lines such as "3" and "7".

On the figures 20. and 21. below, we can see that when we apply deconvolution to filter 18 on a "3" or "7" image, we only get horizontal lines appearing which means that the filter is only catching this feature.

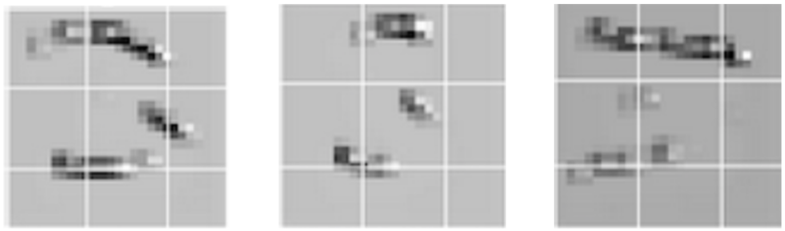


Figure 20: Deconvolution output for filter 18 on "3" images.

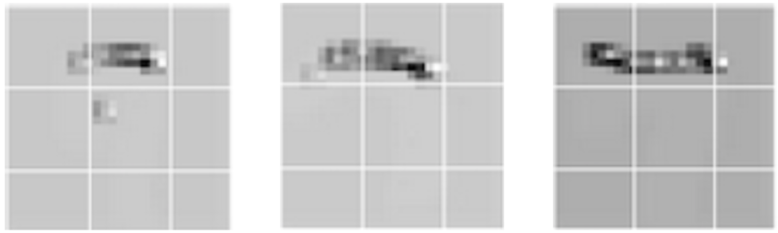


Figure 21: Deconvolution output for filter 18 on "7" images.

Filter {10} represents another filter that captures special patterns, this filter is able to recognize oblique lines. As you can see on the figures below when an image of a number with oblique lines goes through this filter, only this special feature is caught. We are able to visualize this using deconvolution on filter 10.

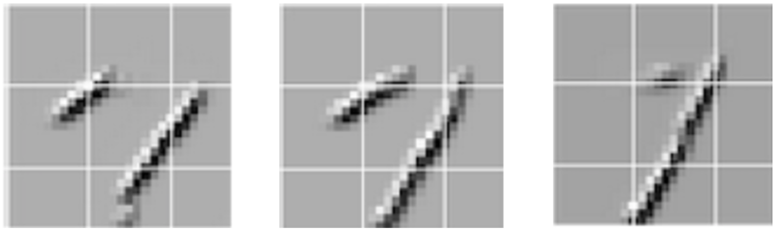


Figure 22: Deconvolution output for filter 10 on "7" images.

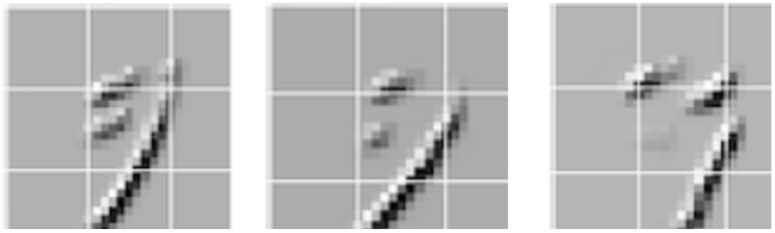


Figure 23: Deconvolution output for filter 10 on "9" images.

As you can see on the figure above only oblique lines when filter 10 is applied on images of "7" or "9".

Note that these observations are only valid for this version of the run of our model, if we re-run it is quite likely that these filters do no longer caught the pattern mentioned earlier.

5 Conclusion and Discussion

The original goal of the Artificial Neural Networks (ANN) approach was to solve problems in the same way that a human brain would. However, over time, attention has moved to performing specific tasks, leading to deviations from biology. Convolutional Neural Networks (CNN) have been used on a variety of tasks, and on this paper, we decided to explain, in simple terms, the main concepts behind these CNNs, starting from basic definitions as building blocks, to analyzing the behaviors of the filters on the network.

Also, we would like to make a point on the fact that, throughout the project, we have faced several challenges. We had to learn a new programming language, get used to new libraries, and to use new concepts and tools that we were not familiar with. In addition, since numerous mathematicians have failed to construct a whole solid theory behind Artificial Neural Networks, it is doubly difficult to make a true/simple interpretation of the results : therefore, this work is mainly based on heuristical explanations.

Let it be clear that there are several details we have oversimplified or skipped : the main goal was to represent the results and conclusions of the different experiments.

There still many areas to explore about this subject, and many experiments to be done in order to improve our understanding of these efficient tools.

References

[1] Philippe Besse, Brendan Guillouet, and Béatrice Laurent. Wikistat 2.0: Educational resources for artificial intelligence. *arXiv preprint arXiv:1810.02688*, 2018.

[2] Francois Chollet. Building autoencoders in keras. *The Keras Blog.[Keras]*, 2016.

[3] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[4] Sumit Saha. A comprehensive guide to convolutional neural networks. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, Published the 15/12/2018.

[5] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *Dept. of Computer Science.*