University of Bordeaux

Collège Sciences et Technologie

Design of a Convolutional Neural Network (CNN)

Author:

COTTAIS Déborah

UE : DEA
18-02-2019

Table of Contents

1 Introduction

In this project, we will try to design a convolutional Neural Network (CNN) in order to classify the fruits in Fruits 360 dataset. To do that, we have a dataset which contains images of 95 types of fruits such as Apples (Golden, Red Yellow,...), Apricot, Banana, and so more ... There are in total : 65 429 images of which 48 905 are for the training data and 16 421 are for the test. The size of these images is 100 x 100 pixels.

2 Neural Networks : Deep Learning

2.1 Neural Networks

First of all, a definition of the term neural network is needed. The idea behind neural networks is to simulate (like a simplified but still faithful copy) interconnected brain cells. This simulation is done inside a computer so that it learns, recognizes models and makes decisions like a human brain. Computer simulations are simply accumulations of algebraic variables and mathematical equations that make it possible to relate these simulations to each other (ie numbers whose values change constantly). A conventional neuron network has thousands of artificial neurons called 'units' arranged in a series of layers like represented in *figure 1.* Each of these layers connecting to the layers at its sides. Some of these layers are the input units, they receive information that the network will try to process. Other units are on the opposite side of the network (output) and indicate how they react to the information they have learned. Between the input and output units are one or more layers of 'hidden' units, forming the great majority of this artificial network. Their role is to turn input units into something that output units can use. Since most networks are fully interconnected, each hidden unit and each output unit is connected to each layer unit at its side.

The question now is, how does a neural network really work, and how is it able to learn things?

The information flows through a network of neurons in two ways. When he learns (so when he is trained) or when he is functioning normally (after training). The information is entered into the network via the input units. This triggers the hidden unit layers as well as those arriving at the output units. However, not all units send a signal continuously. Indeed, each unit receives unit entries placed to its left, and the entries are multiplied by the

weight of the connections they browse. Each unit adds up all the entries it receives in this way. If the sum is greater than a certain threshold value, the unit sends a signal and triggers the units to which it is connected (ie those to its right).

For a neural network to learn, there must be a feedback element. In the same way that a child is told if what he is doing is right or wrong. In this way, the more the difference in what has been done and the result is great, the more radically the answer will be changed. This feedback process is called backward propagation. This involves comparing the output a network produces with the output it was supposed to produce.

The difference between the two can then be used to change the weights of the connections between the network units, from the output units to the input units while passing through the hidden units and this in 'reverse'. Thus, the back propagation will cause the network to learn, reducing the difference between the actual output and the expected output to the point where the two coincide. Therefore, the network will find the 'right' answer as it should.

Once the network has been educated with enough learning examples, it reaches a point where it can be presented with a whole new set of inputs that it has never encountered before.

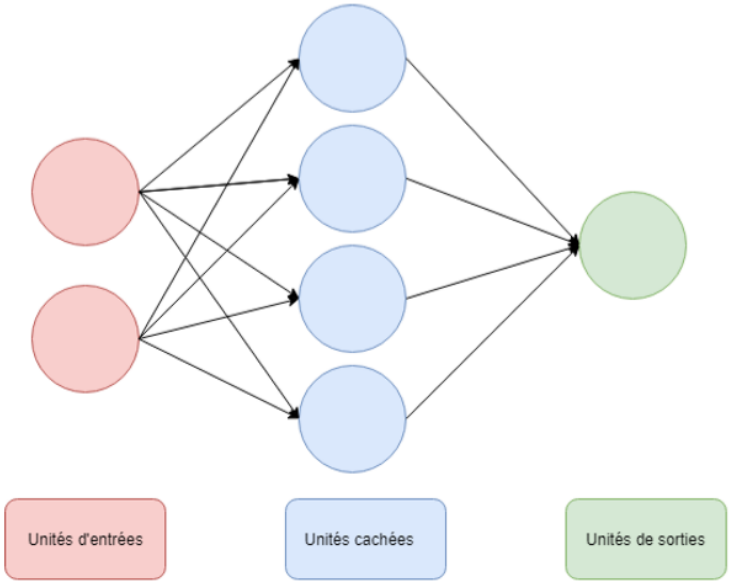

Figure 1: Schema of a basic neural network

2.2 CNN architecture

One of the most popular applications of neural networks in deep learning is that of image recognition like in this project.

The idea is to be able to classify one or several images, for example: is this photo of a cat ? To do this, it is crucial at first to convert our image into a matrix (the dimensions being the number of pixels) which will be the input of our classifier (Linear / Nonlinear Regression, Perceptron, Neural Network etc ...).

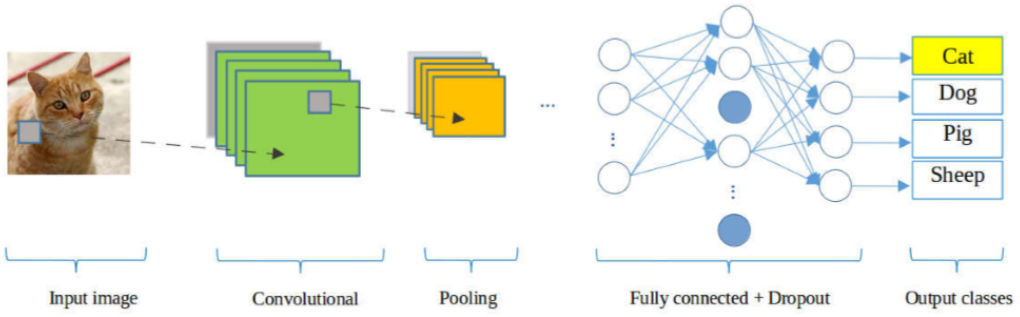

Figure 2: Architecture Overview

There are three main types of layers to build CNN architectures :

- convolutional layer
- pooling layer
- fully-connected layer

The first step is the core of CNN with the parameters consist of a set of learnable filters. Convolutional layer :

- accept a input of (W1 x H1 x D1)
- Hyper-parameters: Number of filters K, filter size F, the stride S, amount of zero padding P
- Output with the size of (W2 × H2 × D2), where : W2=(W1F+2P)/S+1 H2=(H1F+2P)/S+1 and D2=K

Convolution is the fact of highlighting some well chosen characteristics in the source image in order to have the same image but with a sort of filter. Therefore, during this first step, several convolution filters are applied to the initial image, thus leading to the generation of several distinct images at the form level.

The second step allows to reduce the spatial size of the representation, to reduce the amount of parameters and computation and to control overfitting. There are two common types of pooling: MAX and AVERAGE. This step have zero parameters and :

- accept a input of (W1 x H1 x D1)
- Hyper-parameters: filter size F, the stride S
- Output with the size of (W2 $\times$ H2 $\times$ D2), where : W2=(W1F)/S+1 H2=(H1F)/S+1 and D2=D1

The last step of bulding CNN architecture have full connections to all activations in previous layer and their computation is the same as in regular Neural Network. The output of this network is finally reduced to a probability which makes it possible to specify to which degree the image in question is indeed a cat.

3 Results

To build CNN architecture, PyTorch framework has been used and also some packages like Anaconda3, numpy, and torchvision . In this part, I describe the different results given with my CNN architecture.

In order to have the best classification percentage : the best score of accuracy and also the smallest scores of loss, I had to find the best combination of hyper-parameters by changing their values like the number of epochs, the number of lr and also the input and output values.

epochs	lr	momentum	kernel	input1	output1	input2	output2	FC1_in	FC1_out	FC2_in	FC2_out	FC3_in	FC3_out	accuracy
2	0,01	0,9	5	3	6	6	16	16x22x22	120	120	84	84	83	2%
5	0,01	0,9	5	3	6	6	16	16x22x22	120	120	84	84	83	2%
5	0,001	0,9	5	3	6	6	16	16x22x22	120	120	84	84	83	90%
5	0,001	0,9	5	3	70	70	140	140x22x22	300	300	200	200	83	99.472%
5	0,0001	0,9	5	3	70	70	140	140x22x22	300	300	200	200	83	99.497%
5	0,001	0,9	5	3	50	50	100	100x22x22	300	300	200	200	83	99.488%

Figure 3: Some results of accuracy with different values of hyper-parameters

Input1 or 2 and output 1 or 2 represent the number of channels (image channel). The FC1, FC2 and FC3 values are for the affine operation (y=Wx+b).

Like we can see on this table, when we increase just the number of epochs and we don't change the others hyper-parameters, there is no change for the accuracy percentage (it stays the same : 2%). However, when we also increase the lr hyper-parameters, we can see a big evolution. With this combination (epochs=5 and lr=0.001), we have a percentage of accuracy of 90%. Like we can see on the *figure 4* : to the left, we have lr=0.001 and epochs=2 and to the right, we have lr=0.01 and epochs=5. We can see that the higher the number is, better is the accuracy and the loss (with lr=0.01, the loss is 4.4 whereas lr=0.001, the loss is 0.7). Moreover, if we increase the number of epochs to 5 for lr=0.001, the accuracy percentage stays the same, but the number of loss decrease and so better is ([5,10000] loss: 0.064).

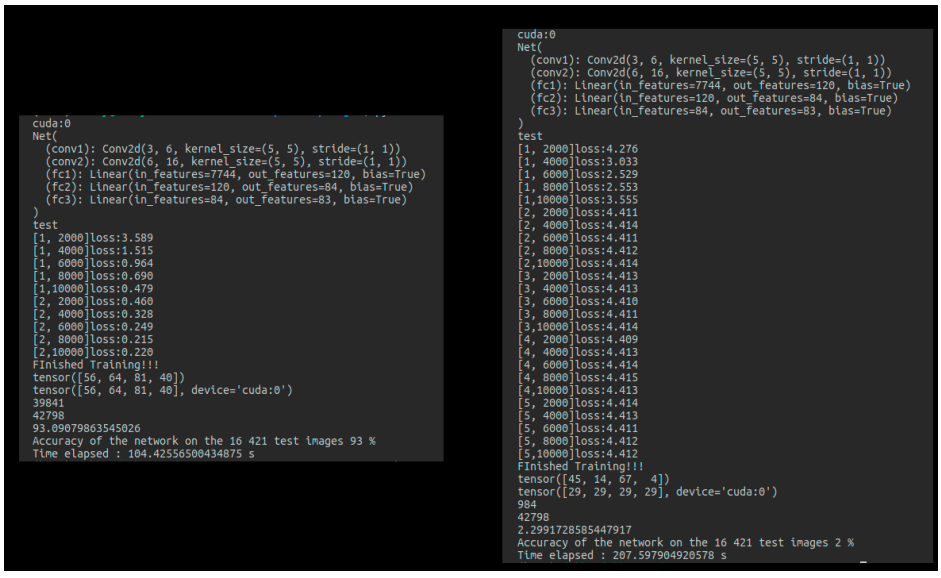

Figure 4: Comparaison between 2 values of lr

Then, if we also change the values of the different input and output, that allows to increase the percentage of accuracy of our model to 99% and also decrease the number of loss ([5,10000]loss:0.007) like we can see on the *figure 5.*

```
cuda:0
Net(
  (conv1): Conv2d(3, 70, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(70, 140, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=67760, out_features=300, bias=True)
  (fc2): Linear(in_features=300, out_features=200, bias=True)
  (fc3): Linear(in_features=200, out_features=83, bias=True)
)
test
[1, 2000]loss:2.901
[1, 4000]loss:0.976
[1, 6000]loss:0.437
[1, 8000]loss:0.282
[1,10000]loss:0.198
[2, 2000]loss:0.131
[2, 4000]loss:0.145
[2, 6000]loss:0.058
[2, 8000]loss:0.056
[2,10000]loss:0.081
[3, 2000]loss:0.062
[3, 4000]loss:0.026
[3, 6000]loss:0.071
[3, 8000]loss:0.059
[3,10000]loss:0.010
[4, 2000]loss:0.010
[4, 4000]loss:0.012
[4, 6000]loss:0.023
[4, 8000]loss:0.014
[4,10000]loss:0.008
[5, 2000]loss:0.011
[5, 4000]loss:0.010
[5, 6000]loss:0.009
[5, 8000]loss:0.009
[5,10000]loss:0.007
FInished Training!!!
tensor([42, 63, 82, 69])
tensor([42, 63, 82, 69], device='cuda:0')
42572
42798
99.47193794102529
Accuracy of the network on the 16 421 test images 99 %
```

Figure 5: The best combination of hyper-parameters (less loss and best accuracy)

Moreover, if we increase again the lr hyper-parameters to 0.0001, we have also 99% of accuracy. However, the number of loss is bigger than with a small value of lr (0.001) and the calculation time of the program is much slower refer to *figure 6*.

```
cuda:0
Net(
  (conv1): Conv2d(3, 70, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(70, 140, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=67760, out_features=300, bias=True)
  (fc2): Linear(in_features=300, out_features=200, bias=True)
  (fc3): Linear(in_features=200, out_features=83, bias=True)
)
test
[1, 2000]loss:4.347
[1, 4000]loss:2.740
[1, 6000]loss:1.297
[1, 8000]loss:0.821
[1,10000]loss:0.566
[2, 2000]loss:0.325
[2, 4000]loss:0.242
[2, 6000]loss:0.176
[2, 8000]loss:0.138
[2,10000]loss:0.108
[3, 2000]loss:0.092
[3, 4000]loss:0.065
[3, 6000]loss:0.076
[3, 8000]loss:0.044
[3,10000]loss:0.058
[4, 2000]loss:0.030
[4, 4000]loss:0.031
[4, 6000]loss:0.028
[4, 8000]loss:0.024
[4,10000]loss:0.018
[5, 2000]loss:0.016
[5, 4000]loss:0.027
[5, 6000]loss:0.023
[5, 8000]loss:0.027
[5,10000]loss:0.016
FInished Training!!!
tensor([39, 45, 62, 33])
tensor([39, 45, 62, 33], device='cuda:0')
42583
42798
99.4976400766391
Accuracy of the network on the 16 421 test images 99 %
```

Figure 6: Result with lr=0.0001

With all these modifications, we can conclude that more we increase the number of epochs, lr and also the input and output values, more the accuracy is better and more the number of lost is small therefore more the model is accurate. To resume, the best combination is :

- epochs=5
- lr=0.001
- momentum=0.9
- kernel=5
- input1=3
- output1=70=input2
- output2=140
- FC1input=140x22x22
- FC1output=300=FC2input
- FC2output=200=FC3input
- FC3output=83

4 Conclusion

The main of this project is to build CNN architecture to classify some fruits in different classes. Thanks to this algorithm, we can classify at 99% of accuracy the fruits in the fruits-360 database.