Analysis and Image Recognition

Using deep Learning and Neural Networks

EL GHAZI Abdou-rrahmane
GHEMRI Tedjani
PIVIDORI Barnabé

Master1 Ecometrie, Statistiques
2019-2020
I.S.F.A

Contents

Part I

Introduction

Artificial intelligence (AI) has been an important topic for many years. First considered as something impossible, many researchers and scientists tried to reach this goal: to make a computer "intelligent".

Before going any further, let's agree on a definition of "making a computer intelligent": does it mean that the computer should be able to score 20pts in mister Clot's exams? Or does it mean that it should be able to take control over the world as in "Terminator"? Not really... A computer or a program is "intelligent" if it can develop some criteria specific to human intelligence. Especially, our ability to learn, to think, to adapt... With machines equipped with the same cerebral abilities as human being, many tasks could be made without any human intervention. This ability transmitted would revolutionize our world. From our way of living, thinking by ourselves, to our ability to innovate or our way of working. Artificial intelligence could replace human beings in difficult tasks but also, it could make tasks that human beings could not do or that would take them too much time. (we could question ourselves on the repercussions that this revolution could have, especially in the world of work where many jobs could be deleted. This leads to a philosophic question, wouldn't this fast progress have some fatal side effects on our society. But this is not the subject of this research).

Now that you know what artificial intelligence is, you may ask yourself where is it in real life? Is it a scientific fantasy or are the giants of Silicon Valley about to conceive the next "Chappie"? As a matter of fact, without even noticing it, you are already using it every day. Facial and vocal recognition on cell phones are able to "learn" your face and your voice, to "think" then recognize you, and to adapt if one day you are a little less shaved than the usual or if you stand in a darker place.

As you may have already understood, this field is considered by many people as the future, and many actors invest a lot in artificial intelligence. Security, automobile, bank, finance, healthcare, marketing, tourism... As you can see, we could list application areas on pages as the scope is infinite, but this would be very unpleasant for you and this is not the point of this work. A lot of big companies exploit artificial intelligence, such as Amazon, Google, Apple, Facebook...

We cannot keep talking about artificial intelligence without mentioning Elon Musk. Considered as a genius for some people, as a madman for others, Elon Musk is a charismatic billionaire who's able to make headlines by smoking cannabis during an interview at Joe Rogan's, or alienate American stock exchange authority because of a tweet. If you don't know Elon Musk, you may at least have heard of one of his companies: creator of PayPal , SpaceX (astronautic leader), OpenAI (artificial intelligence leader), Neuralink (who works on integrating electronic components in brains) ... And also, his most famous company: Tesla. The cars conceived by Tesla are some real technologic gadgets: intern software to control the vehicle, Wi-Fi, possibility of controlling the car with your smartphone, and also, those are "autonomous" cars which means they can do many things by themselves. Those cars are currently categorized according to their autonomy level which can vary from 1 to 5, 5 meaning 100% autonomous, which is not reached yet. We estimate that the rate of AI equipments in cars should double from now to 2025.

Let's focus a little bit more on Telsa cars. Because of legislation and reliability, those cars cannot drive autonomously (even if they technically can on short durations) and human beings must always be on control. But those cars can "turn" the steering wheel, adapt the speed according to the limitations or their environment (cars, pedestrians...), change the line, overtake another car etc... And all of this is possible thanks to artificial intelligence.

We are going to ask you now to close your eyes and imagine what is for you the ideal futuristic car? Maybe some of you will think of a car that is able to prepare a cappuccino that you can sip while the car is driving on Ligurian Riviera roads, why not? But except for the barista side of your car, which, by the way, we're going to call BarisTata from now on, it must be able to drive you from a point A to a point B while you're totally not caring about the road. So it must be able to adapt to any traffic hazards, and we know that in some countries or cities, it's not given! Indian market would be hard to reach and for our BarisTata, that is the last straw.

When you drive, the sense that you use the most is the sight obviously, we use our sight to analyse potential dangers, read traffic signs... Our BarisTata needs a visual system that will be technologically translated by a batch of cameras laid out around the car. But once the environment is recorded by the cameras, how is our BarisTata supposed to know that what it sees is a car and that it needs to slow down? That what is crossing the street is a child running after his ball and that it has to stop straight away? That this is a stop sign and not a speed limitation sign? How does the car analyse what it "sees" through its cameras as a human being would?

In this work, we will try to answer all those questions. To simplify the data analysis, we decided to study the traffic signs recognition instead of the differentiation between a child running after his ball and a raccoon crossing the street. This is why in our study we'll focus on the development of a picture recognition program, especially recognition of traffic signs.

This study allowed us to approach and explore numerous topics such as Machine Learning and Neural Networks. But we will see that pictures are a special case and as of their classification, some methods are more adapted, in particular convolutional neural network (also known as CNN).

Part II

Theory

Chapter 1

Definition of some concepts

Now that we know what we want for our BarisTata, how can we provide it with the eyesight and switch it from Ray Charles to an eagle with a sharp vision? We could write a huge program composed of ifs and elifs that would consider every configuration and all possible and imaginable cases. But the possible field can be exhaustive and listing all those cases could be impossible. Fortunately for us, it doesn't work this way.

By the way, how does artificial intelligence work? We mentioned what an intelligent machine should be able to do: learning, thinking, adapting... But how is that concretely possible?

Let's start by a little leap back in time and a little of history: like in any other field, artificial intelligence went through different approaches and "beliefs", especially the two following approaches, which are connectionism and symbolism, the first one being widely developed these last years.

Symbolic AI can be distinguished by its willingness to explain things in a logical way, wich means that it is based on demonstrable methods and processes. It is declining for many years.

Connectionist AI is based on probabilities, it is more abstract, and leads to very interesting results but almost inexplicable on paper because it uses for example, by repetition, this self-learning of the machine. Connectionist AI dominates in data analysis and image processing, especially when using deep learning and neural networks.

In brief, symbolic AI consists in transmitting to the machine information and specific knowledge to solve targeted issues, whereas the connectionist approach consists in giving to the computer several examples with their solutions, which allows the computer to deduct solutions for any occurring issues.

Let's leave out our BarisTata and cars for a little moment, and to make you understand the difference between these two approaches, let's take the example of medicine field, and radiology in particular.

How can we diagnose cancer on a radiography? The first approach would consist in giving all useful information needed for a good recognition of an actual presence of cancer, based on radiologists processes to detect a cancer with visual analysis and imaging. The second approach would be to transmit to the computer an important database of imaging pictures mentioning if there is a cancer or not. This way, the computer would develop "its own criteria" to decide if there is a cancer or not, based on the results of its analysis of all the pictures we gave it beforehand.

We can already note all the disadvantages of these approaches. Symbolic approach, based only on information to configure beforehand in the machine, can have trouble transmitting this information sometimes complex and difficult to transpose. Visual recognition of a cancer by a human eye can be subjective or different according to the person. The radiologist can have knowledge through experience which even he himself can't explain or some knowledge he would use only on certain type of situations he is not even aware of. The limits of this approach are the difficulty of collecting and structuring human knowledge, especially language and visual recognition.

In the second approach, the lack of "logic" can question the basis of the software. The result is often presented in form of a percentage. Even if the machine recognizes 90% of the cancers we show it, we can still wonder if it is reliable because decision mechanisms are sometimes too much unfamiliar. Generally, we accept a percentage if it's similar to a human's.

With the current technics, we can let the machine find its own criteria. But for this, there are two ways of proceeding: one with unsupervised learning where we give it a database and the program will, by itself, create groups and gather people according to their resemblance. This technic can be useful in marketing or for sale for example: you have a database of all your customers and you ask your program to create 5 different groups to adapt your products to each category.

On the opposite, we have supervised learning where this time we will give our program variables and it will study them to come out with criteria that help him predict more individuals in the future.

We will be interested in this last technic for our BarisTata. Indeed, traffic signs are already categorized and governed by the law, we want to classify them according to these criteria. Our car must do it just like we did it when we learnt traffic regulations: learning all the signs and their meaning, again.., Again.., and AGAIN...

Chapter 2

Perceptron

Now we see things a little more clearly concerning our problem: we have to create a "classifier" thanks to supervised learning methods.

Let's put our BarisTata in the garage for the next few paragraphs and let's focus on a similar problem, but way simpler:

Since its creation in 1930, I.S.F.A organize a rugby match at the end of every year between two teams created by mister Clot: the green team versus the red team. But since he bets every year with mister Bienvenüe on the green team's victory, he distorts the team building so the red team is always composed of shorter and lighter students than the green one... and obviously, nobody knows about his little secret.

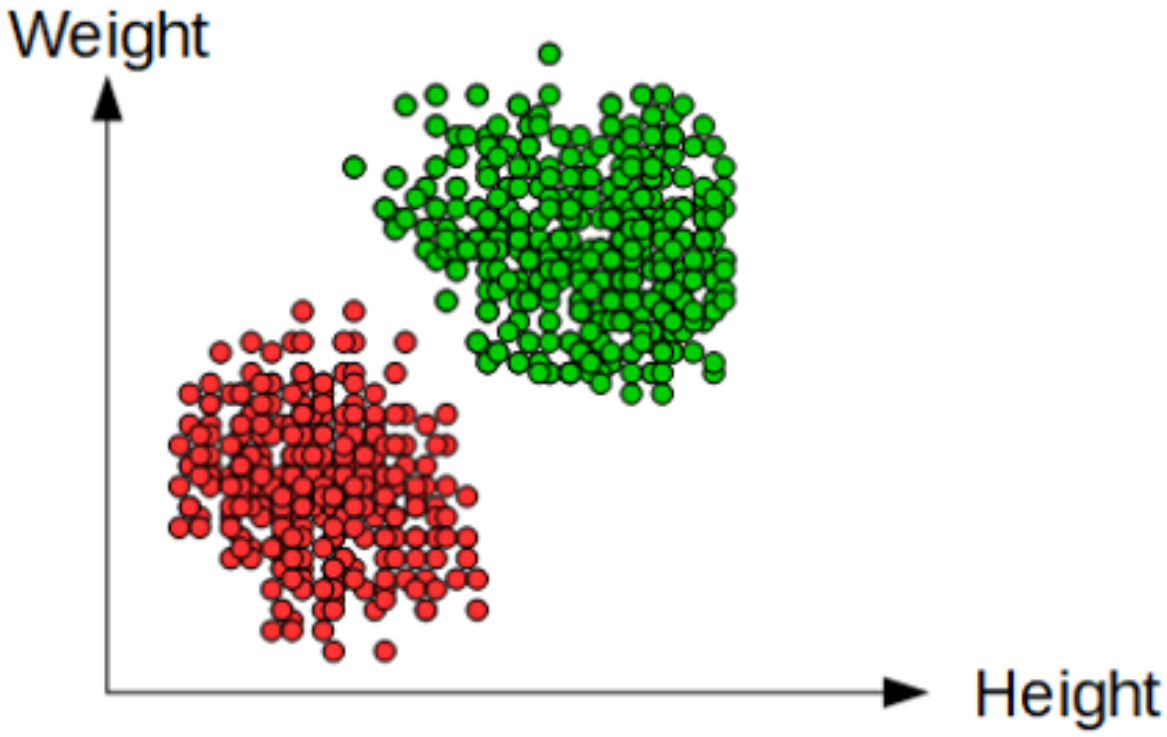

Figure 2.1: Distribution of teams according to height and weight

Frustrated by seeing him winning his bets every year, mister Bienvenüe would like to predict teams constitution and to know if a player P will be in the green or the red team.

For this type of problems, there is an algorithm called perceptron. This is the simplest neural network because it is a formal neuron, it is composed of only one neuron. This is a binary and linear classifier.

Here is its functioning: it takes as an input n variables (in our example, 2 variables: weight and height), it multiplies each variable by a weight 'Wi', it sums up all those values, and using a non-linear function F (called activation function, example: the hyperbolic tangent function denoted "tanh"), it returns a 0 or a 1 corresponding to the category of our people.

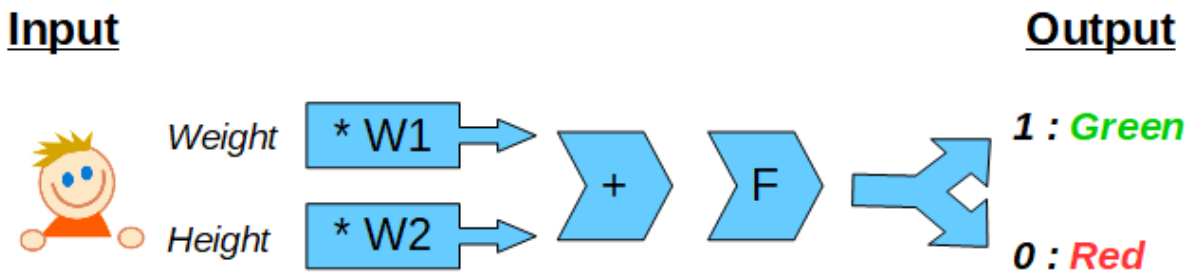

Figure 2.2: Perceptron diagram

Simple isn't it ? But I already know what's your next question : how are the 'Wi' weights chosen ? And from the beginning of this document you've been insisting on program's learning, there is no learning involved in here ? Indeed, you are totally right, here, there is no learning because it's the final model, but to arrive to this version it will practice, it will find himself without any 'human' help the right weights 'Wi'. How does it work concretely?

And this is where our supervised model and our students already "labeled" enter the track. Our little algorithm will initialize random values for 'Wi's, it will predict the students that it already knows, compare its predictions with the reality and measure his error with, as its name indicates, an error function. This function represents the «gap» between the predictions of the model and the reality of the labels. For this purpose there are many mathematical formulas such as the 'mean square errors' used for regressions, in the case of a classification it will often be the function of CrossEntropyLoss that will be used:

$$CrossEntropyLoss = -(y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i))$$

With this function, our algorithm knows for itself if its model matches the reality or not. In fact, if the model is good, the error is close to zero. But how to improve it? it could change the Wi weights randomly until it came across an infinitely small error but

it would be very long and as soon as we got a larger number of inputs it would become ineffective because the combinations are multiple : How do we know that we are optimizing our program by increasing W_96 and not W_504 ?

What is the goal of our training? To minimize our error function, we want it to be as small as possible. For this, we will use a Gradient algorithm. It's an iterative optimization algorithm that makes it possible to find the minimum of a function by repeating several times the same «operations».

First let's try to understand what the gradient is. Mathematically, the gradient of a function at one point is a vector of its partial derivatives at that point :

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ . \\ . \\ \frac{\partial f}{\partial x_n} \end{bmatrix}$$

The gradient will represent the variability of the function around this point. As a reminder, if your high school math classes go back too far, the derivative of a one-point function represents the "slope" of the function at that point. The gradient is widely used to represent force fields in physics, temperature or motion fields in meteorology, etc..

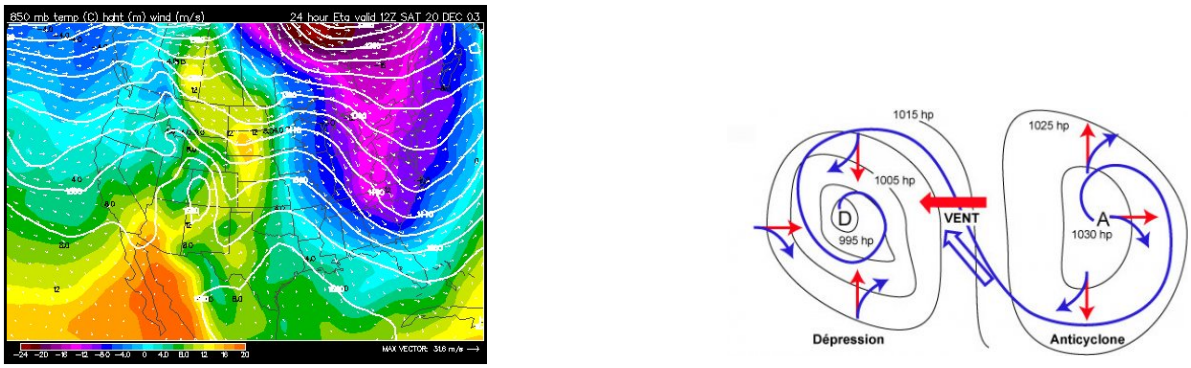

Figure 2.3: Thermal and wind gradient

So, if you understood our little explication, in this case the gradient will be used to « transform » our mathematical function into a « slope ». And we will use it to follow this «slope», always go down and eventually reach the bottom: the minimum of our function. It's a little bit like skiing, if you keep going down there is a point where you will inevitably reach the bottom of the resort ... But there may be some problems: if you land on a false platter you won't advance any more, or if you arrive in a small valley and you stop to have some mulled wine you will be stuck in this valley and you will not necessarily be at the bottom of the station. There are also the same problems in mathematics...

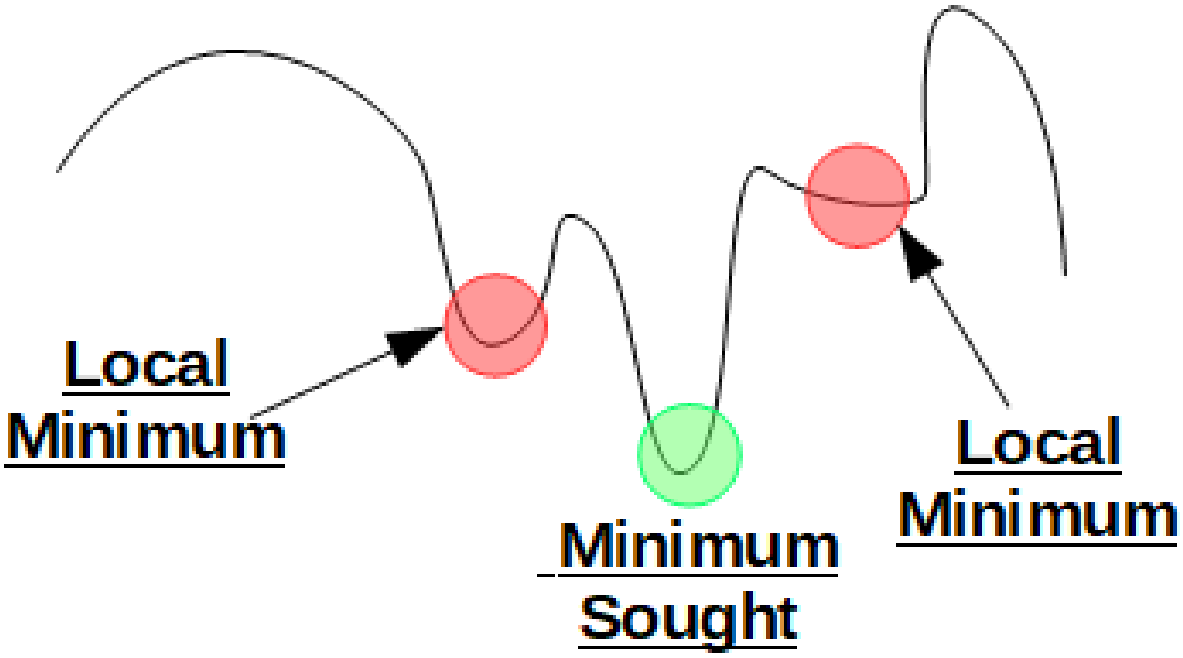

Figure 2.4: Different minimums of a function

To solve this problem, we usually add a term of inertia that will help the program get out of its local minimums. Fortunately for us, in this case, this type of problems happens very rarely.

We're almost done with the gradient, now we just have to explain the algorithm itself. Let us start in our function from a random point that we will call x_0. We calculate the gradient in this point to find «the descent» and then we move by a step proportional to a fixed 'η' (the smaller the 'η' the slower the descent, and the larger the step the more oscillation there will be during the descent), then we restart again and again until we get to the bottom!

Gradient algorithm

- Initialize with x_0 (randomly)
- Repeat
  $x_t + 1 = x_t - \eta \times \bigtriangledown f(x_t)$
- Until reached convergence

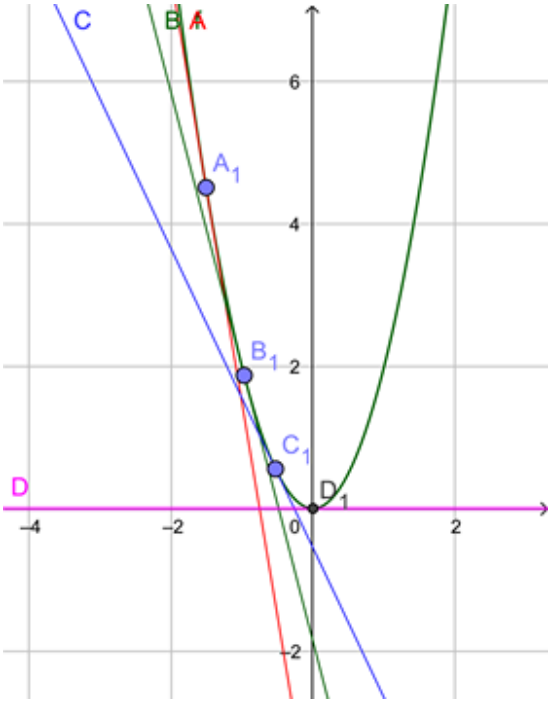

Figure 2.5: Application of the gradient method on the function $f(x) = 2x^2$

For our explanation, we have taken an example in 2 dimensions because it is easier to visualize but it is the same principle for 3,4, or 5000 dimensions.

Let's brief really quick: we know how to create a perceptron that is a binary linear classifier, we saw how to «evaluate» our model using a loss function, and finally we talked about the gradient algorithm that minimizes our loss function and therefore find our ideal 'Wi' weights.

So, we can finally turn on our car's engine and get back to our initial problem. Unfortunately, the perceptron model is not going to be useful for 2 reasons: first of all, it is a binary classifier, and the traffic signs are not limited to only two signs. Secondly and most important, the perceptron is a linear classifier. It is therefore effective only if the separation of our individuals is linear (a two-dimensional straight line, a three-dimensional plan etc.)

Going back to our rugby teams example, a perceptron can be effective because we can separate our 2 teams by a straight line.

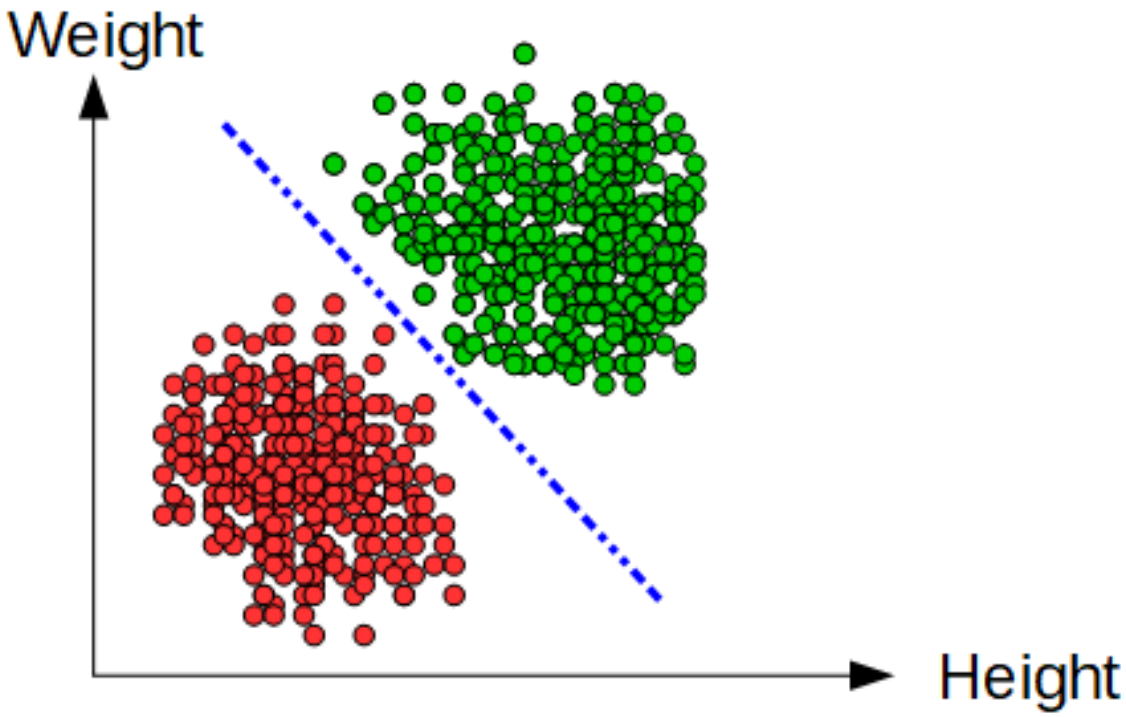

Figure 2.6: Linear Separation between the two teams

But what if the «border» between our individuals is not a straight line?

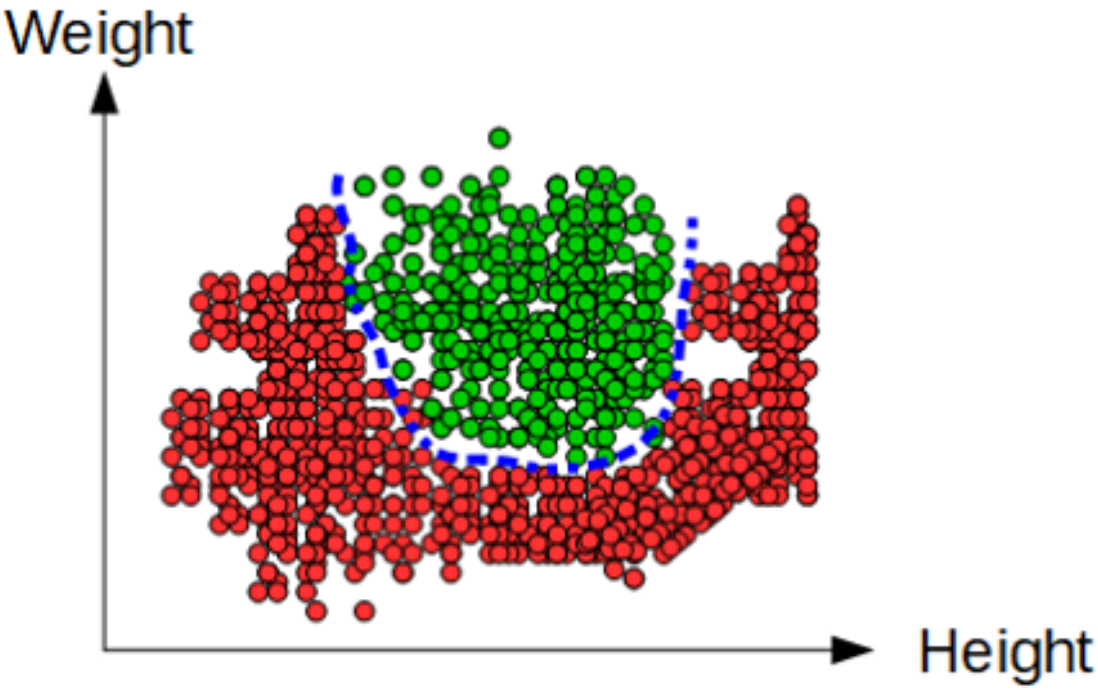

Figure 2.7: Non-linear Separation between two categories of individuals

This kind of separation is the most frequent in real life.

Chapter 3

Neural networks

For this type of problem, we must find another method and put aside the perceptron. But does that mean we have to throw away everything we've seen so far ?
Fortunately for us the answer is NO: many concepts already established in the preceding chapters will be very useful for us for later on.
So, what should we do now?
We must find a way to classify individuals when the border between them is not linear.

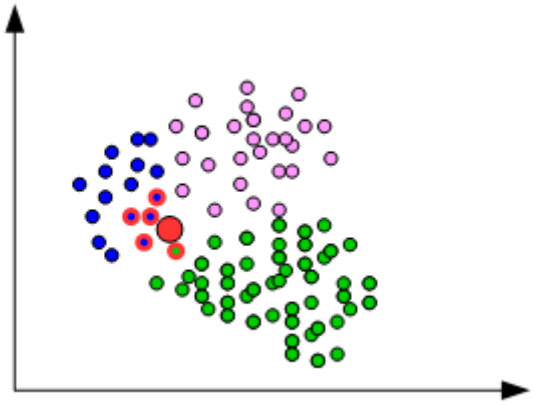

Figure 3.1: 5_means to classify the red point

Let's take a look to the example above, we want to identify the big red point. What category do you think it's going to be classified in? the blue one, the pink one or the green one? If we use the 5 means method, the big red point actually belongs to the blue category, because the 5 closest points to it are blue.

This method could be very useful for us. Indeed, if we use a big database with different traffic signs, we could identify a new sign just by looking at the closest signs... Simple, efficient and fast! Did you just say fast? Not so sure ... For example, if we use a perceptron, and give it a database of 20 000 photos that it will practice on until its weights are established, then it will never need this base ever again to predict new elements. 20 000 photos of 75 pixels by 75 pixels photos (which is really low quality for photos). Each photo

will be made up of $75 \times 75 \times 3 = 16875$ elements ($\times 3$ for the colors), our database will therefore be made up of 337 500 000 numbers... That's a lot to compare it all every time ... Plus, that's not really an AI, our algorithm doesn't really learn anything with this method, it just compares each picture everytime. Thats why we will ask you to completely forget this method.

So we're going to focus on neural algorithms. Why is our brain capable of doing such complex things? Because it is composed of thousands, no, hundreds of billions of neurons. If our brain was a perceptron and composed of a single neuron, I am not really convinced that we would have managed to survive so far ... it might have been beneficial for the planet but you would not be there to read this magnificent report ! For comparison, a snail has about 11 000 neurons.
So, let's do the same for our model! Let's superimpose the neurons!
We end up with what we call a "neural network", a model composed of multiple neurons stacked on several layers.

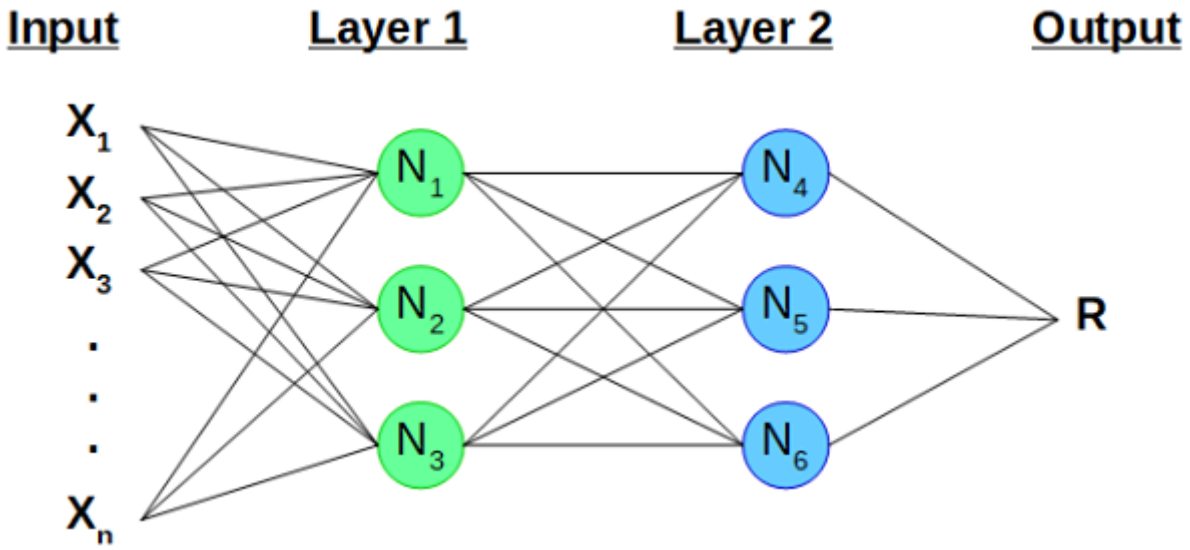

Figure 3.2: Neural network diagram

There it is, a nice neural network! Each neuron acts as a perceptron, it takes as input all the values of the previous layer, applies the weights and sums, applies an activation function, and returns a value to transmit to the next layer. Until we get the final result 'R'.

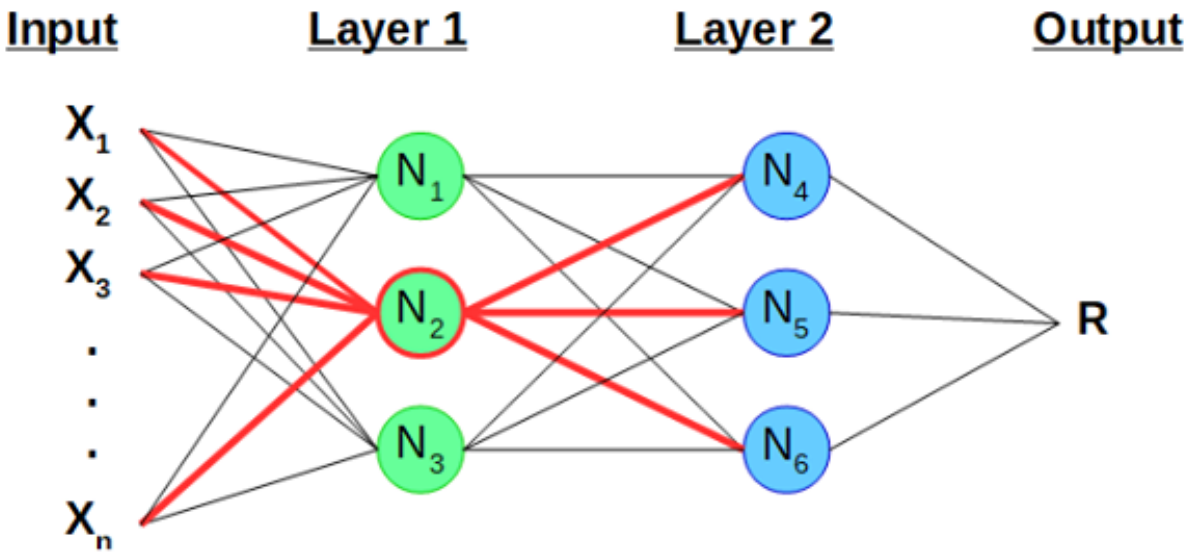

Figure 3.3: Connexions of the neuron N_2

There is an exemple with the neuron N_2 that takes n variables as input, transforms them, and transmits its outcome to the next layer (Layer 2).

To evaluate our model, the principle and the method will be the same as of a perceptron. We have a function such as the crossentropy function that will measure the error of our model. But for learning we have a slight problem, if you remember what we were saying during the chapter 'Percepron', we talked about the gradient method which is used to minimize the error function by using partial derivatives. And in theory, it should be the same principle in a network: we want to find which neuron to act on to minimize the loss. However, in a neural network it quickly becomes complicated to calculate the gradient of the entire network. Because the partial derivatives depend on the previous layers and therefore, they are quick to become very complex.

Fortunately for us, the gradient backpropagation algorithm exists and is used in neural networks to adjust each neuron in order to minimize loss.

So, now we are in possession of models able to identify problems and much more complex borders between individuals. We are no longer limited to binary and linear problems.

Now we can finally take our BarisTata, take out our tools, and put our hands in the sludge? We have one last theoretical point to address and after we promise to move into practice and tuning.

Why is this kind of model effective? Because it can return the same output for 2 similar individuals using all the neurons that acts as some sort of function. This similarity comes obviously with the input variables, which in our case are photos, they will be therefore the pixels. However, to recognize photos, comparing the photos pixel by pixel

and finding similarities between them, aren't enough... because by doing this there are many parameters that we completely overlook: the centering of the photos, the sharpness, the brightness, the tilt ... Our neural networks could work on simple photos very well centered, taken under the same conditions ... However, if we wish to identify signs while our BarisTata is running we realize that these conditions will never be fully validated.

This is why, in this context we are shifting to the most recommended ones, the convolutional networks. We reassure you right now: they are neural networks as well, so everything that we have just seen in this chapter is important!

Chapter 4

Convolutional neural networks

In everyday life, how do you recognize a rabbit? You will identify some forms and characteristics specific to this animal, it is small, hairy, has a small tail and large ears ... You know that it's a rabbit and not a zebra. And guess what, a convolutional neural network works the same way!

To put it in a simple way, a convolutional neural network will "apply filters" on the photos to identify certain paterns specific to the images that it is looking for. In our example of rabbits, we can think that it will apply a filter that highlights their large ears and if it detects these shapes it deduces that it is indeed a rabbit in the photo.

These filters, called convolutions, replace from now on the neurons in our network:

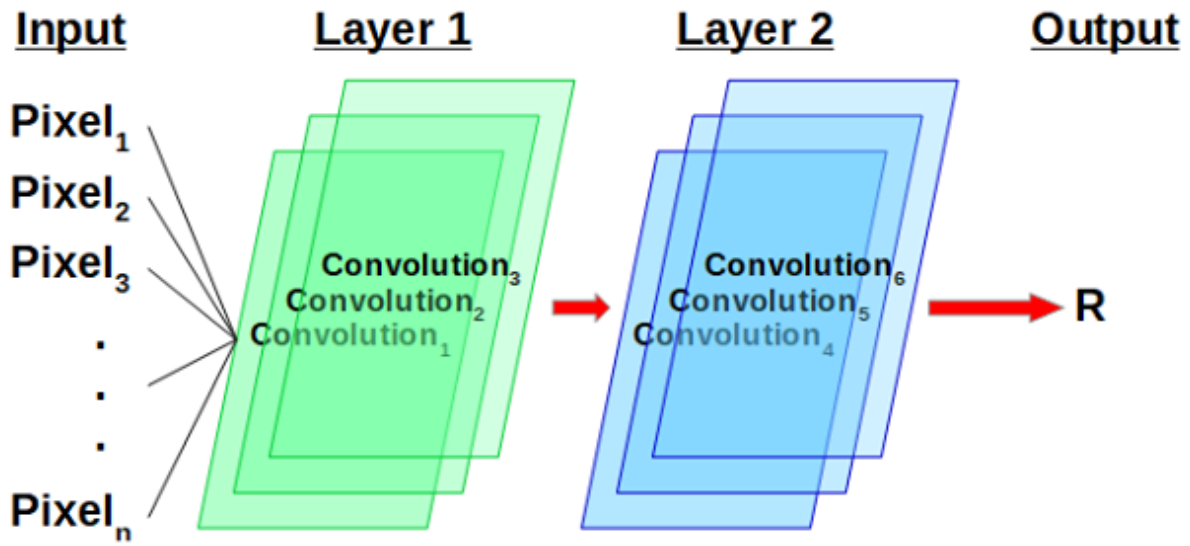

Figure 4.1: Convolutional neural network diagram

Now let's try to understand what is actually happening with our filters: A convolution is an integer square matrix, here is for example a convolution of 3×3:

2	-1	0
1	3	-1
0	2	1

Figure 4.2: Example of a convolution

This filter will then "replace" the weights of our previous networks and just like before, we will sum the multiplications of the pixels of our photo by this filter and we will "walk" our filter through the photo, thus obtaining a new photo... You're lost? Indeed it's not very clear ... An example will surely speak more to you:
We remind you that a photo is made of 3 numbers for red, green and blue but for the example we will only take one number. Here is a beautiful photo of Nathalie Portman ... Yes yes I assure you it's her ... With a little bit of imagination!

9	85	10	64	64
71	63	56	84	97
12	65	17	10	97
54	97	62	34	46
43	79	5	74	26

Figure 4.3: Nathalie Portman's picture

Let's start by applying our filter on the upper right corner of our photo:

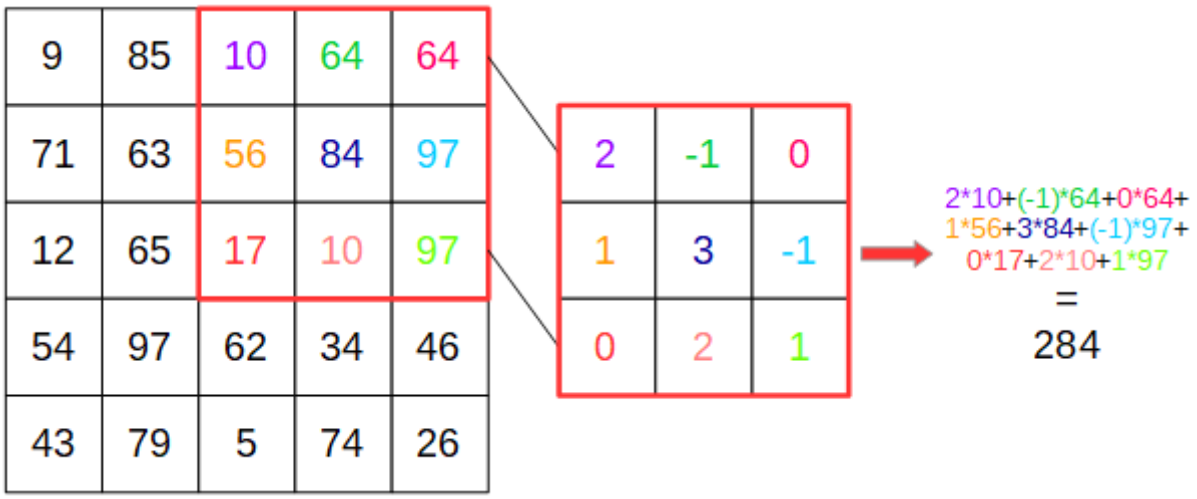

Figure 4.4: Application of the convolution on the upper right corner of Nathalie's picture

We get the value 284... Now let's do this on our entire photo:

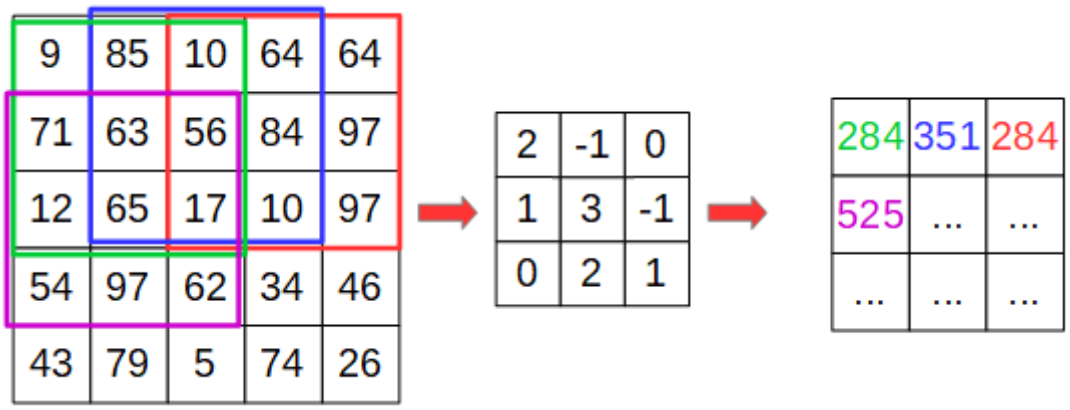

Figure 4.5: Application of the convolution on Nathalie's picture

So, we get as an output a photo a little bit smaller than the original. So, obviously with our example the output numbers are bigger than 255, and therefore cannot be pixels, but it was just to help you understand the principle.

The principle of a convolutional network is to pass a certain number of filters on our photos to obtain smaller but "filtered" photos that highlight some characteristics specific to our desired individuals. After all our convolutions we get a photo much smaller than our original photo but much deeper (the number of filters obtained, this can be hundreds of convolutions that are applied). After that we will pass all our final pixels in one or several layers of classical neural networks as we saw in the previous chapter. This is roughly what our networks is going to look like:

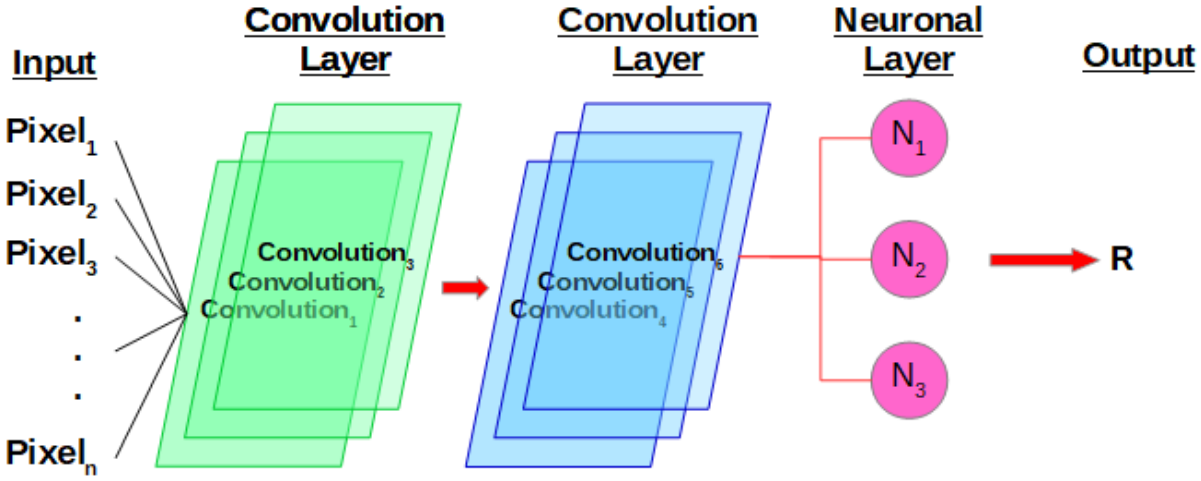

Figure 4.6: Diagram of two convolutions plus one Neuronal Layer

Surely you must be rightly wondering, how we should choose the number of layers? The number of convolutions? The number of neurons?.. There are no actual rules ... You have to fumble, try and find the right numbers. This will be part of the parameters that we will have to play with to find the most optimized model for our problem.

So that's it? we finally have a completed model that will work on our BarisTata? In theory yes, but we are missing one last notion that we have not seen yet during this thesis and which is however fundamental when we are talking about machine learning: Overfitting. If we randomly take a school subject, for example Mr. Clot's subject 'Data Analysis'... To score a good mark and learn well, it is obviously not recommended to arrive to the test by just asking the same morning at 7:30am a quick summary of the all the courses of the semester to your friend, because by doing that, you 'under learn' the subject and risk getting a grade very close to 0. But on the other hand, it is useless to memorise the annals by heart (which would serve as a database) because once in front of your copy, you would be unable to adapt to the new questions. This is what overfitting means: having a model that sticks "too much" to the database and therefore doesn't represent the reality anymore.
Let's take the following for example:

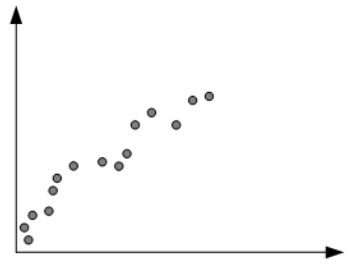

Figure 4.7: Prediction model

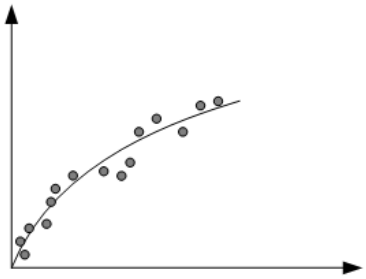

Figure 4.8: Nice regression

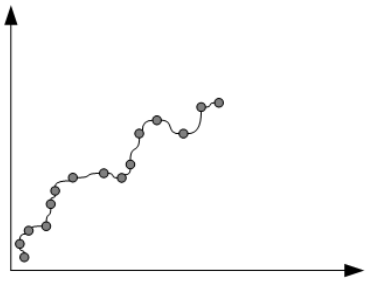

Figure 4.9: 'Overfitting'!!

Indeed, our predictions will no longer be true to reality ... They will follow the training database "too well".

We will see during the construction of our model what to add and what to pay attention to, to avoid this. But for now let's finally get our hands dirty starting the eye surgery of our Baristata!

Part III

Practice

Chapter 1

Data

The first step is obviously to find a database containing photos of traffic signs. The larger our database, the more our model will be able to train and adapt to different photos, we are talking about thousands of photos here. Ideally, our photos should already be labeled. Because if we have to say for each one of our 50,000 photos whether it is a prohibited direction sign or a stop sign, the conception of our database will become very long and tedious.

There are many websites that offer free databases, in particular Kaggle, which is a DataScience website that offers numerous competitions around Data. Therefore, there are many databases available on this site. There are also many government sites that offer Databases: data.gouv.fr is a site that offers data of the French government, of INSEE ... There is the American version: www.data.gov, canadian one: open.canada.ca... Unfortunately on all these sites we did not find our happiness, we ended up finding it on this site:

https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html

This database is composed of 26,640 photos in ppm format, divided into 43 different "types of signs". From the german signage, which is very similar to the french one, this base is perfect for us. Let's look at the head of the database once the zip file has been extracted:

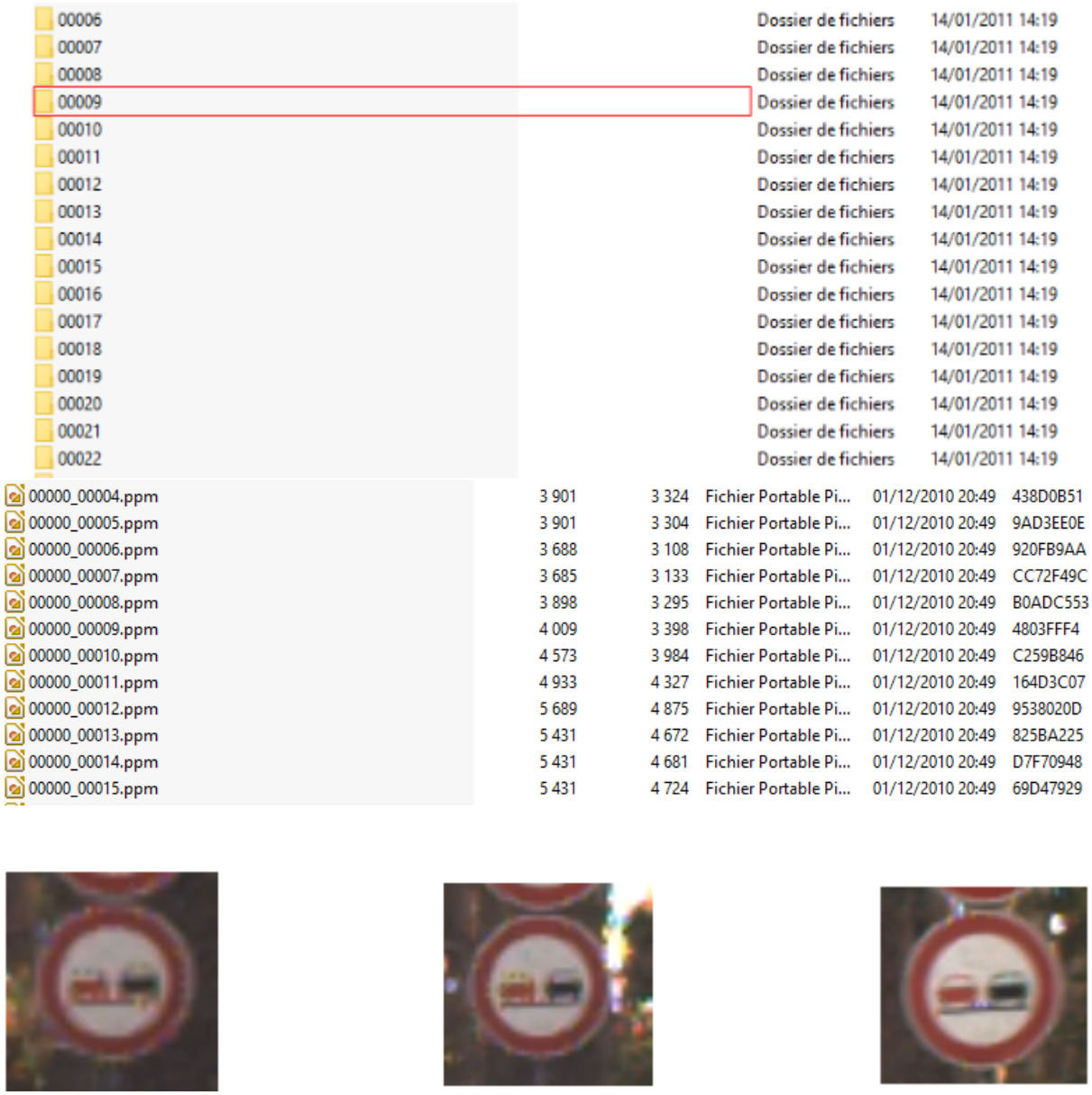

Figure 1.1: The database folder

Each folder contains one different type of signs, let's start renaming each folder to see a little bit clearer:

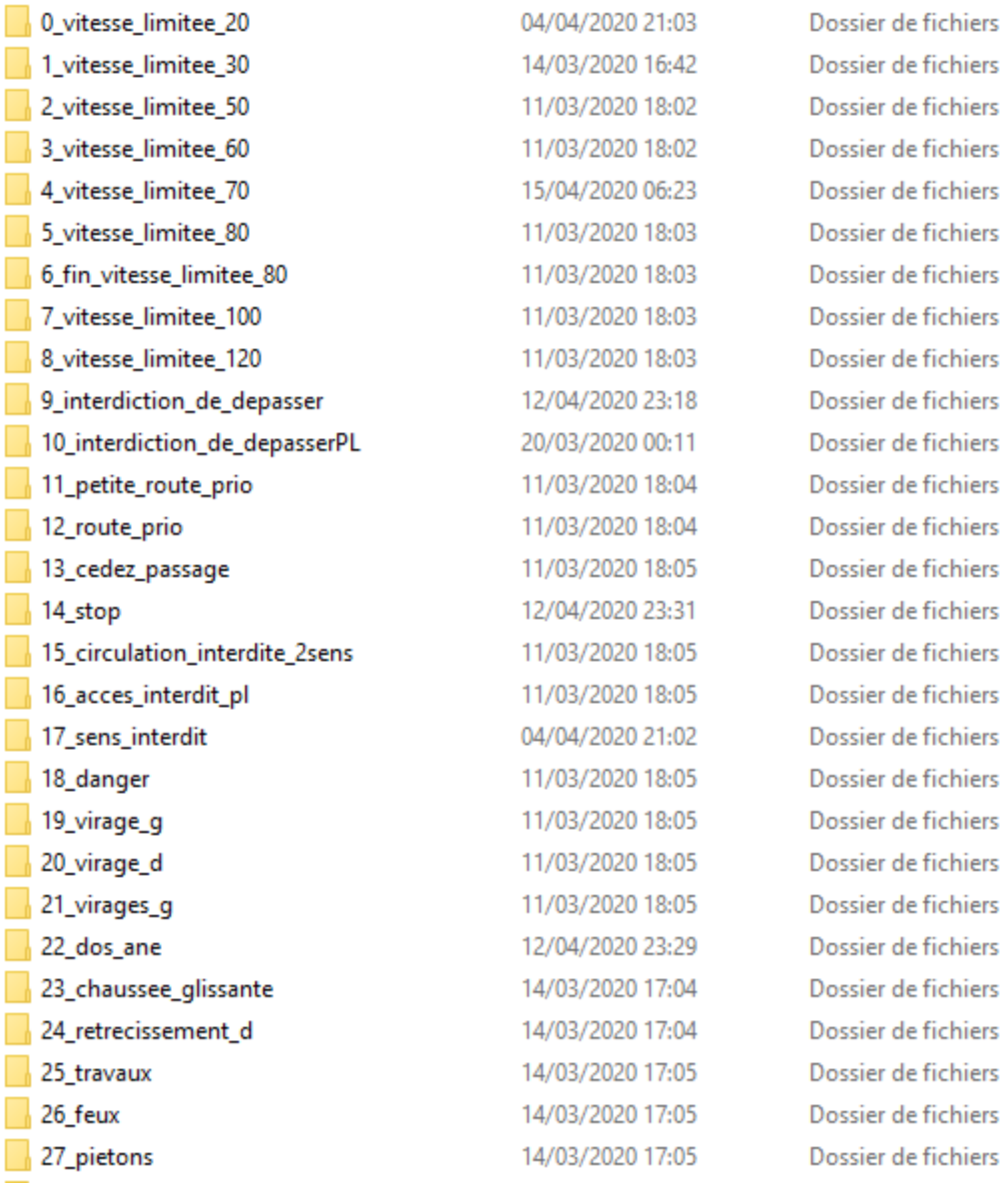

Figure 1.2: Renamed database folder

To carry out this project, we chose to work in python because it is the language that offers the most tools in machine learning, DataScience ... So, now we have to create a small program that allows us to import all of these photos into a large table and associate a label to it that matches its folder name.

For this, we will need 3 packages:

- Glob2: A package that allows us to easily browse through folders and spot paterns in files
- Numpy: A very useful and known package that facilitates the manipulation of vectors and matrixes

- PIL: A package that allows us to manipulate images.

let's start by retrieving all our photos :

```
files = glob2.glob("D:/Cours/M1/TER/GTSRB/Training/**/*.ppm")
```

This command stores all the paths of all our photos in the 'files' array. Glob2 allows us to collect all the files that end with a '.ppm' and the ** that precede it indicates that we are looking in all the training subfolders. So, we have to collect all our photos with a small line. We just have to "read" them and convert them to pixels. For this, we will directly use the PIL package:

```
def Import_X(files):
    t=[]
    for file in files:
        t.append(np.array(Image.open(file).resize((75,75))))
    return t
```

Figure 1.3: Import_X function

This small function allows us to browse through all our paths, and for each path we store the photo pixels in an array. When the photo is opened, it is resized to 75 * 75 to standardize all of our photos. We must now identify the type of traffic signs of each photo in our database. For this, we will use the folders that we renamed previously to identify each photo. Here is a code that identifies by a number ranging from 0 to 42 the type of traffic signs of each photo in our database.

```
def Y_Transformation (files):
    t=[]
    for file in files:
        if("0_vitesse_limitee_20" in file):
            x=0
        elif("1_vitesse_limitee_30" in file):
            x=1
        elif("2_vitesse_limitee_50" in file):
            x=2
        elif("3_vitesse_limitee_60" in file):
            x=3
        elif("4_vitesse_limitee_70" in file):
            x=4
        elif("5_vitesse_limitee_80" in file):
            x=5
        elif("6_fin_vitesse_limitee_80" in file):
            x=6
        elif("7_vitesse_limitee_100" in file):
            x=7
        elif("8_vitesse_limitee_120" in file):
            x=8
        elif("9_interdiction_de_depasser" in file):
            x=9
        elif("10_interdiction_de_depasserPL" in file):
            x=10
        elif("11_petite_route_prio" in file):
            x=11
        elif("12_route_prio" in file):
            x=12
        elif("13_cedez_passage" in file):
            x=13
        elif("14_stop" in file):
            x=14
        elif("15_circulation_interdite_2sens" in file):
            x=15
        elif("16_acces_interdit_pl" in file):
            x=16
        elif("17_sens_interdit" in file):
            x=17
        elif("18_danger" in file):
            x=18
        elif("19_virage_g" in file):
            x=19
        elif("20_virage_d" in file):
            x=20
        elif("21_virages_g" in file):
            x=21
        elif("22_dos_ane" in file):
            x=22
        elif("23_chaussee_glissante" in file):
            x=23
        elif("24_retrecissement_d" in file):
            x=24
        elif("25_travaux" in file):
            x=25
        elif("26_feux" in file):
            x=26
        elif("27_pietons" in file):
            x=27
        elif("28_enfants" in file):
            x=28
        elif("29_velos" in file):
            x=29
        elif("30_neige" in file):
            x=30
        elif("31_animaux_sauvages" in file):
            x=31
        elif("32_fin_interdictions" in file):
            x=32
        elif("33_obligation_d" in file):
            x=33
        elif("34_obligation_g" in file):
            x=34
        elif("35_obligation_tout_droit" in file):
            x=35
        elif("36_obligation_tout_droit_droite" in file):
            x=36
        elif("37_obligation_tout_droit_gauche" in file):
            x=37
        elif("38_contournement_d" in file):
            x=38
        elif("39_contournement_g" in file):
            x=39
        elif("40_rond_point" in file):
            x=40
        elif("41_fin_inter_depasser" in file):
            x=41
        elif("42_fin_inter_depasser_pl" in file):
            x=42
        t.append([x])
    return t
```

Figure 1.4: Y_Transformation function

So, we can now create our database properly with these 2 functions:

```
X=np.array(Import_X(files))
Y=tf.keras.utils.to_categorical(Y_Transformation(files))
```

We store our photos in the form of pixels in a vector called X and the label of this photo is located at the same index of the vector called Y. With the function Y_Transformation presented just above, we get a number to identify our signs, but in neural networks , we need to do some one hot encoding to categorize individuals. In other words, our label must be a binary representation with a single '1'. For example, to code a program that identifies if it is a photo of a cat or a dog, instead of having 0 and 1, we would have [0,1] and [1,0]. In our application, we will therefore have vectors of 42 zero with a 1 at the index of the desired type of signs.

To do this, we don't need to code it, a function of the 'tensorflow' package already does this. The very popular package 'tensorflow' will be the essential package for this project. Very complete, it is useful in everything related to supervised learning and DataScience, it contains many libraries such as Keras that is very useful for deep learning, plus, it supports both convolution as well as recurrent neural networks. It is this library that the to_categorial function comes from, which directly does one hot encoding from our traffic signs numbers.

So, we have our database containing all of our traffic signs, the first thing we're going to do is keep some of it that our model will never see. Keeping a part for a final test which allows us to test the presence of 'Overfitting'. If, for example, our model is 95% reliable, but on data that he has never seen before he has only 85% good response, we know that he has 'overfitted'. For this, we will isolate 10% of our database that will be reserved for a final test. We do this directly with sklearn's train_test_split command:

```
X_train, X_test_final, Y_train, Y_test_final = train_test_split(X, Y, test_size=0.1)
```

This command directly allows to randomly draw 10% of X and Y, that will be placed in X_test_final and Y_test_final. The remaining 90% will be placed in X_train and Y_train The last step before finishing the preparation of our data is to divide our two tables of X by 255 to have all our values are restricted between 0 and 1:

```
X_train=X_train/255
X_test_final=X_test_final/255
```

So, our data is ready to be used ... we just have to create our model ... Fortunately, with Tensorflow this is quite easy!

Chapter 2

First Model

Remember, when we first talked about 'overfitting' we told you that we would see later on how to solve it. You know that our model will be composed of different convolutional layers and some "normal" neural layers. To prevent the over learning, we will add after each convolutional layer a pooling layer. There are different types of pooling, but the most used is maxPooling. It is an area that scans our photo and keeps the maximum values only stored inside of it. Again, an example will be more telling... Let's apply a 2×2 pooling on our image of Nathalie Portman:

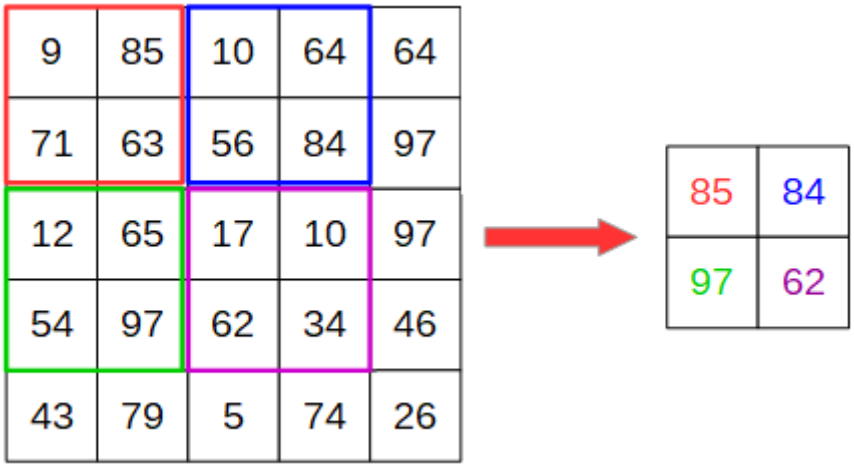

Figure 2.1: Application of 'Pooling' on Nathalie's picture

It's very simple, the only "difficulty" is the way to manage the edges. There are different ways to do that, we can complete our photo with zeros, so that that our pooling can go on until the end:

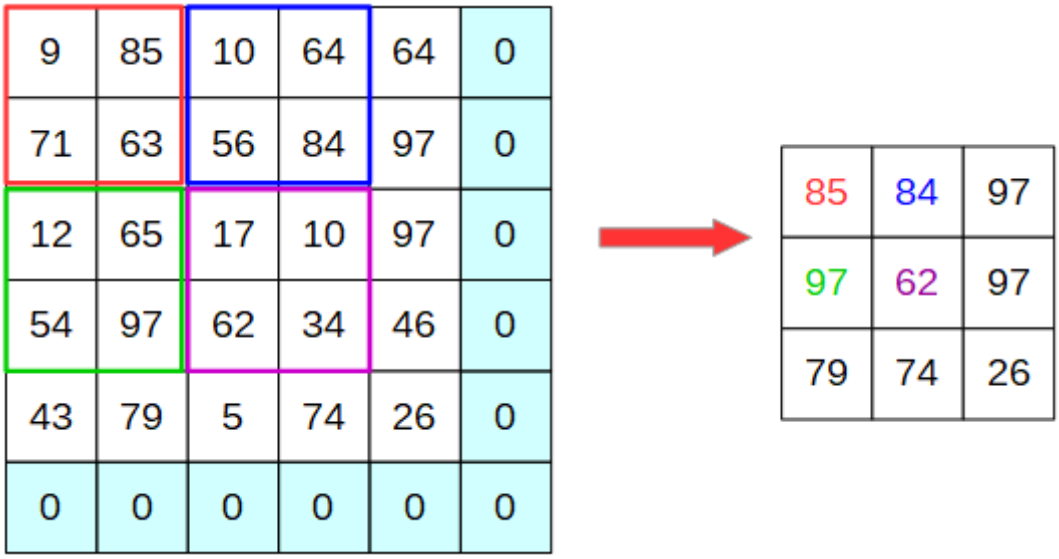

Figure 2.2: Adding zeros on the edge and applying the 'Pooling'

Or, we can, as in the first example, "delete" the sides, this is the choice we will make. So, we end up with an photo that becomes smaller and smaller as the layers go by, but also deeper and deeper (stacking filters). In addition to preventing overfiting, poooling will allow us to considerably reduce the size of our photos and this will allow us to go faster in our calculations, while keeping only the important elements of our photo.

For our first model, we will create 3 onvolutional layers. Tensorflow allows us to do this very easily, let's start by creating the model:

```
model = tf.keras.Sequential()
```

We add a convolution layer :

```
model.add(tf.keras.layers.Conv2D(16,(3,3),activation="relu"))
```

The first parameter, 16, is the number of convolutions that will be present in our layer. Next we have the size of our convolution, which is (3×3), then we specify the activation function that we want. The "relu" function is the most used for convolutions.

$$f(x) = x^+ = max(0, x)$$

We place a layer to normalize the image resulting from our convolutions, then we put a pooling layer:

```
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))
```

For pooling, we specify the size (2×2), then we choose how we want to manage the edges (as explained above) thanks to the padding variable.

It's that simple to create one complete layer of convolution with tensorflow. Let's create 2 more, one of 32 convolutions and one of 64 convolutions. So we end up with our 3 layers of convolutions :

```
model.add(tf.keras.layers.Conv2D(16,(3,3),activation="relu"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))

model.add(tf.keras.layers.Conv2D(32,(3,3),activation="relu"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))

model.add(tf.keras.layers.Conv2D(64,(3,3),activation="relu"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))
```

Figure 2.3: 3 convolutional layers of Tensorflow

We remind you that after each layer, our photo becomes smaller but deeper, the more we advance in our network the more filters we can apply.

For this model, all the parameters (number of layers, number of convolutions per layer, size of the convolutions, size of the pools, etc.) were chosen at random. Then it will be up to us to play with these parameters to find the most efficient model.

After our "convolution" part, we have to add a "classical" neural network part. To do this, we must first "flatten" our photo

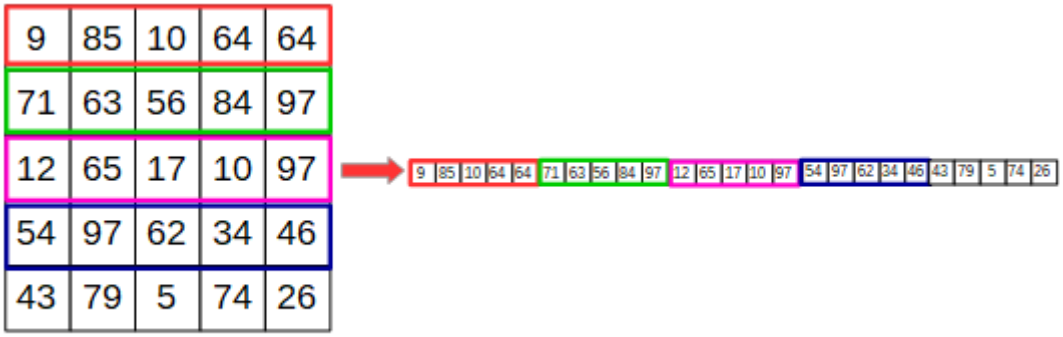

Figure 2.4: 'Flattening' Nathalie's picture

We do it like this usingTensorflow:

```
model.add(tf.keras.layers.Flatten())
```

Now, we can add 2 layers of fully connected neurons :

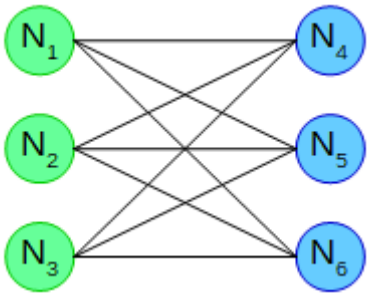

Figure 2.5: 2 fully connected neuronal layers

The commands in python are :

```
model.add(tf.keras.layers.Dense(10000, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(5000, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))
```

Figure 2.6: 2 fully connected layers of Tensorflow

So the first layer contains 10,000 neurons and the second one 5,000. You can see that we no longer have a pooling layer but a dropout layer. As the name suggests, this layer deletes neurons. The function takes as parameter the percentage of neurons thrown away, here it is 50%. This layer has the same role as the pooling but for fully connected networks it will reduce the risk of overfitting.
Now we just have to add the last output layer. Our output is made of 43 neurons (one per type of sign):

```
model.add(tf.keras.layers.Dense(43,activation='softmax'))
```

Activation is no longer the same, it's 'softmax' instead of 'relu'!
Let's compile our model

```
model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])
```

We give it the categorical crossentropy function for loss. We choose the adam function as an optimizer and the accuracy in metrics. Again, these parameters are those recommended for convolutional models but we could test other ones.
This is the moment we have all been waiting for since the beginning of this work!!! Let's start the training and look at the performance of our model!

```
history = model.fit(X_train, Y_train,validation_split=0.2,epochs=10, batch_size=64)
```

When we start the training, we use 2 very important parameters: the number of epochs and the batch size. The number of epochs is the number of times our model will train on the entire database. The batch is the size of the batches that our model is going to train on.

Let's say our database contains 10,000 photos. If we tell it 10 epochs with batches size 100, he will take 100 photos, test what it learnt on these 100 photos, modify its convolutions, then take another batch of 100 other photos and start again and again until it has seen the 10,000 photos (so it will make a total 100 packs of 100).Now it finished one epoch, so it will start all over again 10 times!

Split validation is used to control overfitting: at the end of each epoch it test what it learnt on a part of the database that it has never seen before (here it's 20%). This result is very important because the more training there is, the more our model's precision increases because it adapts in order to increase its precision. But if at the end of an epoch the model is 90% accurate but only on the validation batch, on photos he has never seen before, it is only valid at 60%, our model is either not efficient or there is overfitting.

Let's go !!! Press enter and execute our code!

```
Epoch 1/10
19180/19180 [==============================] - 151s 8ms/sample - loss: 0.9860 - acc: 0.8052 - val_loss: 8.4515 - val_acc: 0.1902
Epoch 2/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.1792 - acc: 0.9558 - val_loss: 0.2103 - val_acc: 0.9477
Epoch 3/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.1238 - acc: 0.9722 - val_loss: 0.9215 - val_acc: 0.8776
Epoch 4/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.1121 - acc: 0.9743 - val_loss: 0.1364 - val_acc: 0.9737
Epoch 5/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.1068 - acc: 0.9780 - val_loss: 0.1559 - val_acc: 0.9739
Epoch 6/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.0706 - acc: 0.9868 - val_loss: 0.2035 - val_acc: 0.9706
Epoch 7/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.0995 - acc: 0.9801 - val_loss: 0.1485 - val_acc: 0.9758
Epoch 8/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.0735 - acc: 0.9856 - val_loss: 0.1080 - val_acc: 0.9817
Epoch 9/10
19180/19180 [==============================] - 152s 8ms/sample - loss: 0.0605 - acc: 0.9879 - val_loss: 0.2052 - val_acc: 0.9771
Epoch 10/10
19180/19180 [==============================] - 153s 8ms/sample - loss: 0.0711 - acc: 0.9875 - val_loss: 0.2824 - val_acc: 0.9683
Sur le Test final de 2664 images l'accuracy est de 96.69669669669669
```

Figure 2.7: The results of our first model

Let's analyze the result! At the end of each epoch, we have 4 values. The loss, which decreases progressively, which makes sense because our model adapts in order to reduce this loss. The accuracy, which increases with the epochs. And the same 2 values this time calculated on the validation batch. We arrive to an accuracy of 98% and 96% on the validation, moreover, we remind you that we had put on the side at the very beginning 10% of our database to do a final verification test.
We test our model on this part with the following lines of code:

```
c=0
for i in range (len(X_test_final)):
    im=X_test_final[i]
    im = np.reshape(im,[1,75,75,3])
    if(Y_test_final[i][model.predict_classes(im)]==1.0):
        c=c+1
print("Sur le Test final de",len(Y_test_final),"images l'accuracy est de",(c/len(Y_test_final))*100)
```

Figure 2.8: Final test program

Our final test conforms to validation because it gives us an accuracy of 96.7% Let's look at the evolution of our 2 accuracies (Classical accuracy and validation accuracy) on a graph:

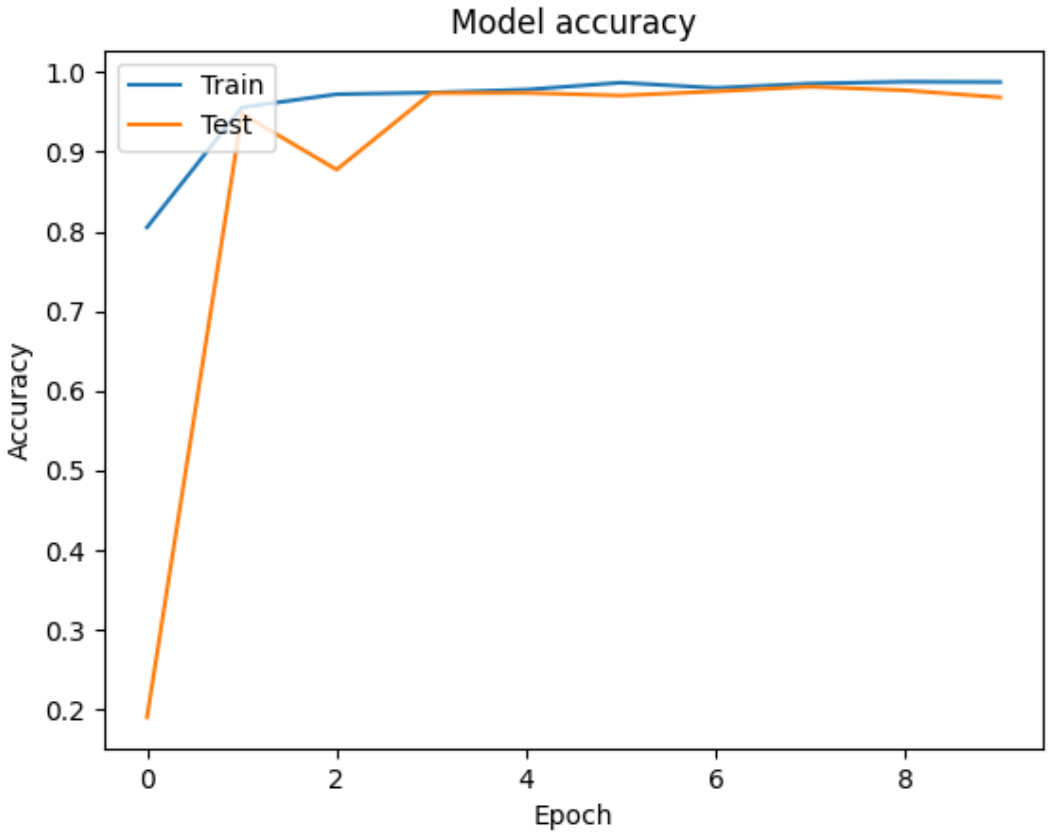

Figure 2.9: Accuracy's evolution depending on the epochs

Not so bad ! Our validation (in orange) does not depart from our training (in blue) so in theory there is no overfitting!

But let's test our model on real conditions! To do this, we have collected 7 photos of traffic signs taken from 'google map' and we will pass them under the headlamps of our BarisTata. Here is the little program that does this after having of course imported our photos

```
c=0
for i in range (len(X)):
    im=X[i]
    im = np.reshape(im,[1,75,75,3])
    im=im/1
    if(Y[i]==[model.predict_classes(im)]):
        c=c+1
    else:
      print("BarisTata n'a pas reconnu ===>",files[i])
print(c,"/",len(X))
```

The results are :

```
BarisTata n'a pas reconnu ===> /Users/mai2011428/Ter/Test_Google/17_sens_interdit.jpg
BarisTata n'a pas reconnu ===> /Users/mai2011428/Ter/Test_Google/9_interdiction_de_depasser.jpg
BarisTata n'a pas reconnu ===> /Users/mai2011428/Ter/Test_Google/40_rond_point.JPG
BarisTata n'a pas reconnu ===> /Users/mai2011428/Ter/Test_Google/14_stop.jpg
BarisTata n'a pas reconnu ===> /Users/mai2011428/Ter/Test_Google/34_obligation_g.jpg
2 / 7
```

Figure 2.10: BarisTata's results on some google's pictures

Hmm .. not that great! So we need to find a more efficient model. Now, it's the time to tinker with all of our parameters, play the mad scientist and test lots of different combinations.

Chapter 3

Improvement

Given the training time of these models: from 30 minutes for the fastest ones to several hours for some others on our own personal computers. We decided to do all of our training on the isfa server to reduce our training time by 4 times. In addition, the server offered us the possibility of launching several models at the same time. After each execution, the results and the graphics generated by our programs were automatically saved in our personal space, so we were not obliged to stay in front of our machine.

All these calculations being greedy in resources, and having received an email from Mr. Clot kindly informing us of our excessive use of the server, we have most often run all of our models at night, in order not to interfere with the use of the server by other students.

After very numerous tests, we arrived at the following model:

```
model = tf.keras.Sequential()

model.add(tf.keras.layers.Conv2D(16,(3,3),activation="relu"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))

model.add(tf.keras.layers.Conv2D(32,(3,3),activation="relu"))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.MaxPool2D(pool_size=(2, 2), padding='valid'))

model.add(tf.keras.layers.Flatten())


model.add(tf.keras.layers.Dense(2000, activation='relu'))
model.add(tf.keras.layers.BatchNormalization())
model.add(tf.keras.layers.Dropout(0.5))

model.add(tf.keras.layers.Dense(43,activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer=tf.keras.optimizers.Adam(),metrics=['accuracy'])

history = model.fit(X_train, Y_train,validation_split=0.2,epochs=15, batch_size=150)
```

Figure 3.1: Final model

We had already chosen not to reduce our photos to a size of (75×75) but (100×100), the server being more powerful, we could analyze more detailed photos. The program above only contains the model because to prepare the data, apart from the size of the

photos, everything else is similar to what we did before.

Therefore, In this model, we have only 2 layers of convolutions, with respectively 16 and 32 convolutions per layer. Followed by a single neuronal layer made of 2,000 elements.

In addition, we have made 15 epochs with batches of size 150 ... Now I stop my Blabla, I don't want to make you wait any longer, here is the results:

```
Epoch 1/15
19180/19180 [==============================] - 105s 6ms/sample - loss: 0.8032 - acc: 0.7987 - val_loss: 4.1555 - val_acc: 0.1424
Epoch 2/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0968 - acc: 0.9723 - val_loss: 4.7339 - val_acc: 0.2475
Epoch 3/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0342 - acc: 0.9909 - val_loss: 2.6895 - val_acc: 0.4852
Epoch 4/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0266 - acc: 0.9929 - val_loss: 1.3814 - val_acc: 0.6987
Epoch 5/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0269 - acc: 0.9930 - val_loss: 0.0999 - val_acc: 0.9729
Epoch 6/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0210 - acc: 0.9942 - val_loss: 0.1635 - val_acc: 0.9575
Epoch 7/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0298 - acc: 0.9913 - val_loss: 0.1314 - val_acc: 0.9643
Epoch 8/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0173 - acc: 0.9952 - val_loss: 0.0774 - val_acc: 0.9789
Epoch 9/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0124 - acc: 0.9967 - val_loss: 0.0602 - val_acc: 0.9846
Epoch 10/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0117 - acc: 0.9967 - val_loss: 0.0819 - val_acc: 0.9766
Epoch 11/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0169 - acc: 0.9954 - val_loss: 0.1822 - val_acc: 0.9527
Epoch 12/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0255 - acc: 0.9917 - val_loss: 0.0913 - val_acc: 0.9746
Epoch 13/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0135 - acc: 0.9958 - val_loss: 0.0787 - val_acc: 0.9783
Epoch 14/15
19180/19180 [==============================] - 107s 6ms/sample - loss: 0.0070 - acc: 0.9981 - val_loss: 0.0714 - val_acc: 0.9825
Epoch 15/15
19180/19180 [==============================] - 106s 6ms/sample - loss: 0.0134 - acc: 0.9962 - val_loss: 0.0669 - val_acc: 0.9831
Sur le Test final de 2664 images l'accuracy est de 98.57357357357357
```

Figure 3.2: Results of our final model

So, we have an accuracy of 99.6% and on validation it reaches 98.3%... not bad! Our final test gives us 98.5% good answers on 2664 images.

Let's look at the evolution of accuracy:

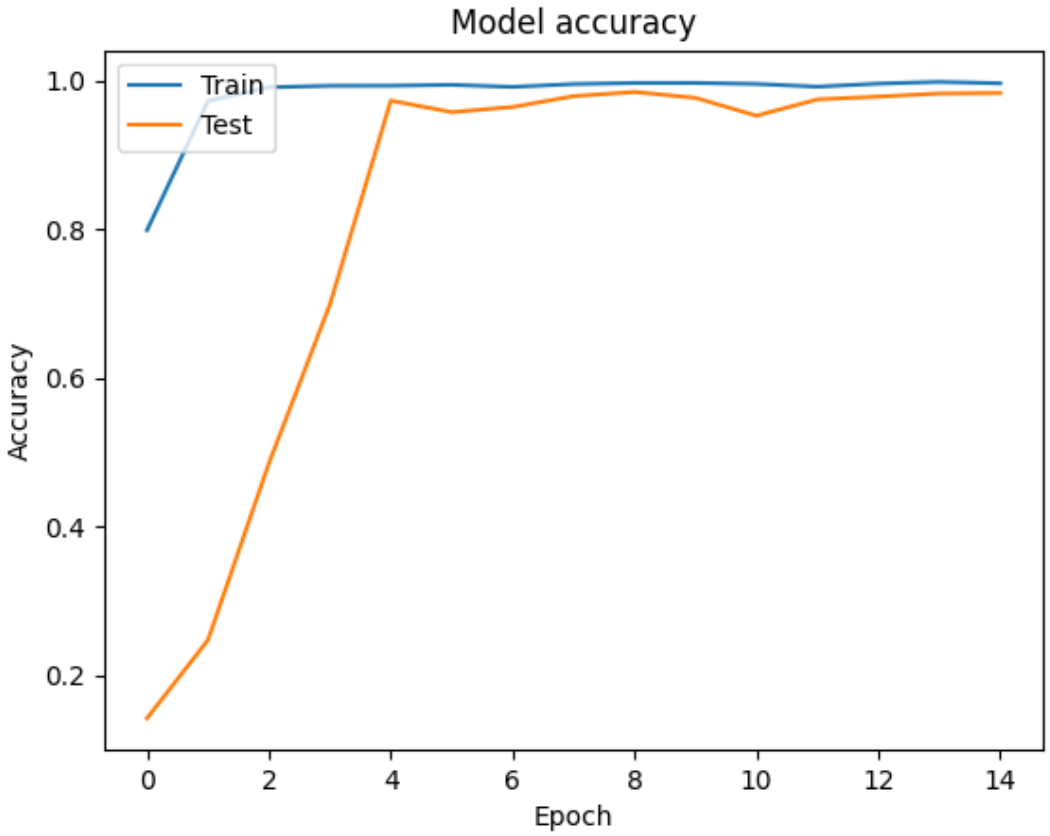

Figure 3.3: The evolution of our final model's accuracy

The accuracy on the validation batch and the accuracy on the training converge together. So, there is no overfitting.

Let's test our model on our google photos :

```
7 / 7
```

It recognizes 7 photos out of 7, so this is the model that we will retain!

Our entire program is available in the annexed link.

Part IV

Conclusion

The results obtained during this project are satisfying. We managed to create from scratch our own program of image recognition of traffic signage. In order to achieve this, we had to mobilize our knowledge in computer science, especially python, and mathematics, in order to be able to understand, analyse, and reproduce in practice the different concepts and methods that we were able to discover. . This upstream research, and this permanent reflectionthroughout our work, have taught us a lot about the subject that we can, to say it again, has fascinated us in every way.

Nevertheless, this ongoing reflection leads us to ask ourselves whether our model is "perfect". The answer is no, our model could be perfected, as could, to a lesser extent, most existing models on the market.

To optimize our model, there are different points that we could work on. As we mentioned, the database is very important in these programs. Our database was quite large, but if we could have had more diversity or if we could have found an even larger database, our program would have been better. The same goes for the training of our model. We were able to run our training a lot of times, but with even more appropriate equipment or extra time, we could have made this model even better. We can also simply ask ourselves if our code could have been improved. It's true that we did some optimization work on it, but we are convinced that this code could have been even better, just like most of codes that exist.

To carry out this project, we first decided to work each one of us on his own to increase our personal knowledge and to be more efficient in our interventions within the group. But at the same time, we had immediately set up an online group discussion to be able to exchange in case of difficulties, new proposals, relevant documents to share between us etc...

Secondly, in addition to our online group discussion, we had a folder called "TER Project" on Google Drive, where we shared our research and also our progress, so each member could be kept informed at any time of the project's progress, and could also contribute to the new information. For example, we could modify a document that was already there by adding text and changing its color, so we could all notice the changes. Moreover, to use the ISFA server, we changed the access authorizations to allow each person of the group to intervene on the files present on our personal spaces.

We then set up meetings on weekdays outside of school hours, or on weekends, while keeping the online system we had in place.

The current context in France has turned our way of living and working upside down. Since March 17th, the confinement has had the effect of developing telework in France. To a lesser extent, it is the same for our working group. We could no longer see each other during meetings, so we organized our meetings by videoconference. We mainly used Skype,Discord and Kast to exchange all 3 of us by video via the internet, as well as to share our computer screens.

Our way of working at the very beginning of the project allowed us to adapt properly to

this new context. We were already used to working from home and sharing and exchanging with each other online. So we had to adapt to work exclusively this way. This adaptation appeared to be less difficult than expected for our group.

In spite of this exceptional situation, we have kept a permanent contact with Mr. Denis CLOT, our supervisor, who has always been available for us throughout the realization of our project. We were able to exchange directly and very quickly by e-mail when adjustments were necessary, and we programmed several videoconferences thanks to Webex. When we encountered difficulties in the realization of the code, or when we had questions about the right methodology to adopt, we were able to exchange and move in the right direction. This was also the case when using the ISFA server, where we were able to count on his good recommendations.

Carrying out this project thus allowed us to confront the complexity of building our own convolutional neural network program, and the right attitude to adopt for the good progress of this one.

Appendix A

Current context

The current context in France has turned our way of living and working upside down. Since March 17th, the confinement has had the effect of developing telework in France. To a lesser extent, it is the same for our working group. We could no longer see each other during meetings, so we organized our meetings by videoconference. We mainly used Skype,Discord and Kast to exchange all 3 of us by video via the internet, as well as to share our computer screens.

Our way of working at the very beginning of the project allowed us to adapt properly to this new context. We were already used to working from home and sharing and exchanging with each other online. So we had to adapt to work exclusively this way. This adaptation appeared to be less difficult than expected for our group.

Appendix B

Links

B.1 Photos

Links to the different sources of the photos used throughout this work:

1. https://pixabay.com/fr/vectors/gar%C3%A7on-dessin-anim%C3%A9-enfants-1299574/
2. https://www.weather.gov/source/zhu/ZHU_Training_Page/Miscellaneous/Heights_Thicknesses/thickness_temperature.htm#GRADIENTS
3. http://www.ppmenegoz.com/spip.php?article75
4. https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/GTSRB-Training_fixed.zip

B.2 Documents

All the documents used in the conception of this project:

1. https://pages.isfa.fr/~denis/docs/ml.pdf?fbclid=IwAR2GcEfnqZSRpaB4_KRc0MDUjgVMpB7QjZJ-vGg7HRbrSvTpr-xRT_rUbtI
2. https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/5082166-quest-ce-quun-reseau-de-neurones-convolutif-ou-cnn?fbclid=IwAR2GcEfnqZSRpaB4_KRc0MDUjgVMpB7QjZJ-vGg7HRbrSvTpr-xRT_rUbtI
3. https://stanford.edu/~shervine/l/fr/teaching/cs-230/pense-bete-reseaux-neurones-convolutionnels?fbclid=IwAR3tyqctLUbYWVov7ei2d6LbwvKB7qPnJe9lqs4zNnIgXkyr8E3qxt1ZzEE
4. https://openclassrooms.com/fr/courses/4470531-classez-et-segmentez-des-donnees-visuelles/4470538-comprenez-comment-un-ordinateur-voit-une-image?fbclid=IwAR1jD8mwY-zaDq9DHq14eEL4RgjhB_PB36LCiAUruSMDLb_f6S71n5G1ZLw

5. http://vision.gel.ulaval.ca/~cgagne/enseignement/apprentissage/A2018/presentations/iam-sem11-reseauconv.pdf?fbclid=IwAR3_kwgVdyFoOzNeRRzStFvKSi-8WNM6yNLNJCmsoKs4wtq-EmSU1xjrxsg

B.3 Videos

Links to all the videos used for a better understanding of the different concepts:

- Introduction to perceptron
  1. https://www.youtube.com/watch?v=RNYT9bECfOo&feature=share&fbclid=IwAR3SX6L97B5LjlJDlfeJbTIcQWbdrDIrZv19XyQCBj1JSj_uRggl4Gc1G5Y
- Introduction to neural networks and deep learning
  1. https://www.youtube.com/watch?v=gPVVsw2OWdM&fbclid=IwAR3wsoR3wf_BjJMRFqb6jt_HQTFABu-7S5JSiE0Q3qRy75R1QSIcJ_P1eNQ
  2. https://www.youtube.com/watch?v=T-wpJPZV8io&fbclid=IwAR12XKtA1WkWMS9Zu0Pza4yxhU2z8vb_wxutsyvmQPCHhq2FdfQKKNNX5eU
  3. https://www.youtube.com/watch?v=GvQwE2OhL8I&feature=share&fbclid=IwAR1OaG47rn-CO0JHAXR5S2lRcwVhHAhW24uvFooF8HN1b46ISzC_InlaNOM
- Introduction to convolution neural networks, especially for image classification
  1. https://www.youtube.com/watch?v=2JEtEdsLdoo&fbclid=IwAR1WPlfwnZuPlYncEIhL-Tqi8UTwFxEz6XB39Qloi5YyoSFt1paOkoMSJXk
  2. https://www.youtube.com/watch?v=zG_5OtgxfAg&fbclid=IwAR3SX6L97B5LjlJDlfeJbTIcQWbdrDIrZv19XyQCBj1JSj_uRggl4Gc1G5Y
  3. https://www.youtube.com/watch?v=kFcviv7p2_s&fbclid=IwAR3GlvNX-Pl0XgGzn5XK-9kGxUk4OMVa8KeNdhcuZInlipHOTqjfLdp5Ud0
  4. https://www.youtube.com/watch?v=YRhxdVk_sIs&feature=share&fbclid=IwAR0l6_D5Z_1q98T5XShhk3hvmmvv6zdHPxo2GNoCfsVwi5-KnLrUasZxOaU
- Help in understanding how image classification works
  1. https://developers.google.com/machine-learning/practica/image-classification?hl=fr&fbclid=IwAR3sQj6PmYqz2ES5a-1UrMeHTgaJbAe06VN3p6bDt8TE29cBORMLDXob-FI

B.4 The program

Link to access our entire program:

1. https://pages.isfa.fr/~mai2011428/ter.html