Neural Network applied to image recognition

Deep sea pictures classification

Project report

Seatech 2A - MOCA

2023 - 2024

Authors :
Fabien MATHÉ & Yohan STABLO

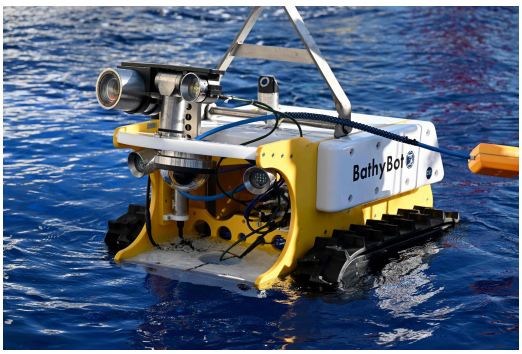
Advisor :
M. Mehmet ERSOY

MOCA - 2A

Abstract

ENGLISH

Approached by the **M.I.O.** - *the Mediterranean Institute of Oceanography* - to develop an AI-based tool that can detect fish in a picture, the goal of this report is to show how we created an application that can be used to process images from videos captured by ***BathyBot***, a deep sea exploration Remotely Operated underwater Vehicle - *ROV* - of the **CNRS**.



The BathyBot from the CNRS ©N. TUCAT

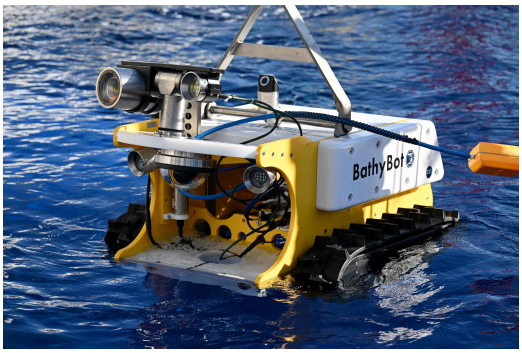
Today, the best way to process images is using AI. Even if it looks like a revolutionary concept, the source of AI took place in the 1950s with Alan Turing and other IT scientist of his time. Originally conceptualized as an autonomous machine that could write and read its own code, AI became through time a mathematical and numerical representation of a human brain, with neurons fully connected in networks. Those neural networks can learn by themselves from their errors using a descent gradient method in a back propagation algorithm.
To understand how an AI works, the best way is to create one using only a few basic functions and a lot of mathematical algorithms.
But there is a lot of libraries of functions that can be used to ease the creation of an AI that can process object detection in images, like **TensorFlow** and **Keras**, two *Python3* libraries dedicated to neural networks. Using them helps to create a Convolutional Neural Network - *a CNN* - that can deals with underwater pictures. After a tough training, this CNN could classify pictures captured by ***BathyBot*** as containing fish or not.
To go further, creating an object detection algorithm, using this CNN, which can find the location of a fish in a picture is the final achievement of this project.
And finally, all those things must be included in a user-friendly application to meet expectations of the **M.I.O.**

Keywords:

AI, Deep learning, Neural network, Convolutional neural network, Image recognition, Object detection, Fish, Deep sea, Underwater, Sea exploration

FRENCH

Approchés par le **M.I.O.** - *l'Institut Méditerranéen d'Océanologie* - pour développer un outil basé sur l'IA capable de détecter les poissons sur une image, l'objectif de ce rapport est de montrer comment nous avons créé une application pouvant être utilisée pour traiter les images issues de vidéos capturées par ***BathyBot***, un véhicule sous-marin télécommandé d'exploration en eaux profondes - *ROV* - du **CNRS**.



Le BathyBot du CNRS, ©N. TUCAT

Aujourd'hui, la meilleure façon de traiter les images est d'utiliser l'IA. Même si cela semble être un concept révolutionnaire, l'origine de l'IA remonte aux années 1950 avec Alan Turing et d'autres informaticiens de son époque. Initialement conceptualisée comme une machine autonome capable d'écrire et de lire son propre code, l'IA est devenue avec le temps une représentation mathématique et numérique d'un cerveau humain, avec des neurones entièrement connectés en réseaux. Ces réseaux neuronaux peuvent apprendre par eux-mêmes de leurs erreurs en utilisant une méthode de descente de gradient dans un algorithme de rétropropagation.
Pour comprendre comment fonctionne une IA, la meilleure façon est d'en créer une en n'utilisant que quelques fonctions de base et de nombreux algorithmes mathématiques. Mais il existe de nombreuses bibliothèques de fonctions qui peuvent être utilisées pour faciliter la création d'une IA capable de traiter la détection d'objets sur des images, comme **TensorFlow** et **Keras**, deux bibliothèques *Python3* dédiées aux réseaux neuronaux. Les utiliser aide à créer un réseau neuronal convolutif - *un CNN* - capable de traiter des images sous-marines. Après un entraînement intensif, ce CNN pourrait classifier les images capturées par ***BathyBot*** comme contenant des poissons ou non.
Pour aller plus loin, créer un algorithme de détection d'objets, en utilisant ce CNN, qui peut trouver l'emplacement d'un poisson sur une image est l'objectif final de ce projet. Et enfin, toutes ces choses doivent être incluses dans une application conviviale pour répondre aux attentes du **M.I.O.**

Mots-clés:

IA, Apprentissage profond, Réseau neuronal, Réseau neuronal convolutif, Reconnaissance d'images, Détection d'objets, Poisson, Eaux profondes, Sous-marin, Exploration marine

Acknowledgment

We extend our heartfelt appreciation to the following individuals whose guidance and support have been indispensable in the completion of this project:

Foremost, our deepest gratitude goes to our advisor, Mr ERSOY. We'd like to really thank him for finding us a very thrilling subject and for guiding us through this project, ensuring the smooth running of our work from A to Z. We are profoundly grateful for their mentorship and encouragement throughout this journey.

We are indebted to Mr GOLAY, Head of M.Sc Numerical Simulation Dpt, for the incredible university curriculum their created, allowing us to explore our subject through the deepest point.

Special recognition is owed to MIO Laboratory for creating the *BathyBot*, this so special ROV. Thanks to that, we could work with them for 3 exciting months on this project.

We would like to express our sincere appreciation to Yolhan MANNES, PhD student of Mr ERSOY, who help us finding new way of thinking during our brainstorming sessions.

Lastly, we wish to thank our families, and especially our 4 wonderful babies for their support and understanding throughout this process and to always be there for us.

To each of these individuals, we are truly thankful for their role in shaping this project and for their unwavering belief in our abilities.

Contents

Abstract 1

Acknowledgment 3

Introduction 5

I What is AI? 7
I.1 History of AI . . . 7
I.2 Perceptron or, the surface of deep learning . . . 8

II Deep dive in neural network 10
II.1 Neural network theory . . . 10
II.2 What really is back propagation? . . . 12
II.3 Results and review . . . 14
II.4 Comparison with TensorFlow . . . 17

III Two-dimensional image model 19
III.1 What is a Convolutional Neural Network (CNN) . . . 19
III.2 CNN Architecture . . . 19
III.3 Our CNN default model . . . 21
III.4 Dataset . . . 22
III.5 Towards the most effective classification model . . . 23
III.6 Conclusion . . . 29

IV Our app 32

V Development opportunities 33
V.1 Changing our architecture . . . 33
V.2 Data warfare . . . 33
V.3 Object detection . . . 34
V.4 Our HCR-CNN method . . . 35

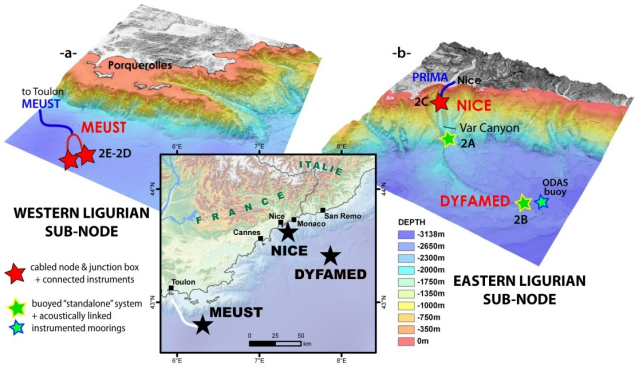
Conclusion 39

References 40

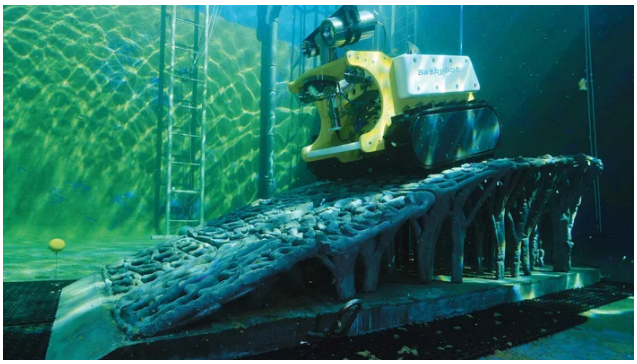
Introduction

The vast deep-sea environments remain largely inaccessible and often represent unexplored territories. The EMSO-LO cabled observatory is one of the few deep-water monitoring systems in the world. Located at a depth of 2500 meters in the Mediterranean, this observatory holds significant scientific importance, bringing together researchers from various disciplines to study this environment and assess the impacts of climate change.

Since April 19, 2023, *BathyBot*, a scientific robot from the **CNRS**, is working alongside the EMSO-LO off the coast of Toulon, France.



(a) EMSO observatory at Ligurian site ©EMSO



(b) BathyBot during a test in a basin at La Seyne-sur-Mer ©Dorian Guillemain

Figure .1: The environment of *BathyBot* in the Mediterranean sea

Its mission: collect video $6\,min$ a day to discover what is hiding deep-sea.

The problem: to know if there is fish in the video, someone must analyse every frame of it and say if there is a fish or not. After this first analyze, a scientist has to classify fish he discovered in the video. This is a very long and non-optimized process. That's where we arrived!

Expectation

So the **M.I.O** needed a tool that could preprocess the videos from ***BathyBot*** to select only the frames where there is one or more fish. The best way to achieve this is to use AI - *Artificial Intelligence*. Indeed, the **M.I.O** already have an AI-based tool named *ParticuleTrieur* that analyse plankton.
They want us to develop a tool inspired by their tool, and that's why we decided to focus our project on **neural networks** - *shorten as NN*.

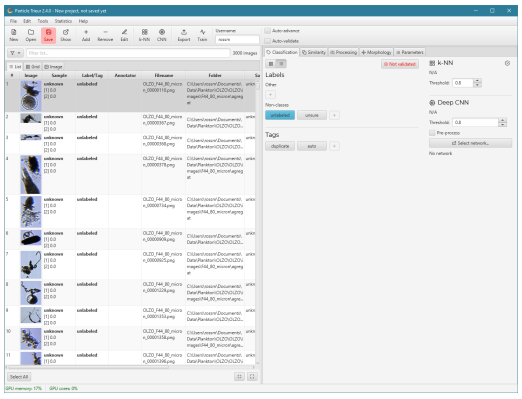


Figure .2: ParticuleTrieur overview ©

Our project goals

Today, AI is everywhere: on your phone or in your car but also where you don't expect to find it, like in data analysis or art creation.

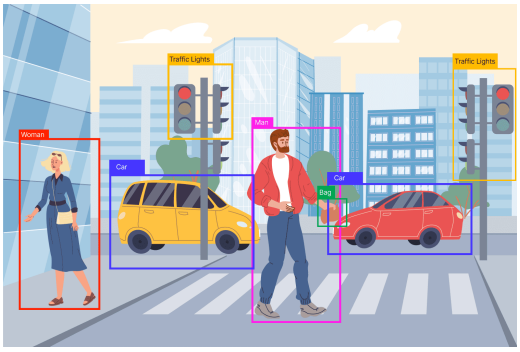


Figure .3: Object detection based on AI ©DeepLobe

Our goal is to create an AI that can process the videos from the *BathyBot* to detect frames where there is fish, extract them from the video and create a zoomed picture of the fish in order to analyze and classify it later.

Object recognition is also a very well known sector of AI, so there is a lot of models that are running it very well. But to do something really interesting, we decided to begin from scratch, using only our brains to develop our own AI.

First, we had to understand what is a neural network, from where it came and how it is used today.

Following those initial observations, we created a first neural network from scratch, without using any predefined tools to enforce our understanding of AI.

All this has led us to our first $2D$ image processing model that we optimized to process underwater pictures and find fish in them.

To complete our work, we have included the results of our project in an easy-to-use application.

Finally, because we found a lot of way to explore during our work, we found new perspectives to develop our AI, from enhancing dataset to creating an object detection algorithm.

All the code we developed during this project are available at this following GitHub address, due to appendix overfill: `https://github.com/Fabien-Math/HydroCognition`

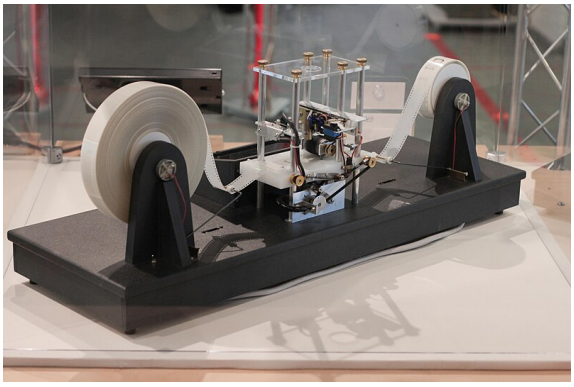
I What is AI?

To begin our work, we had to understand what is an AI - *an artificial intelligence*. Indeed, we're hearing a lot about it but we don't really know what it really is.
Let's first check a short definition: "*Artificial intelligence is the ability of a digital computer or computer-controlled robot to perform tasks commonly associated with intelligent beings*" [1, Britannica].

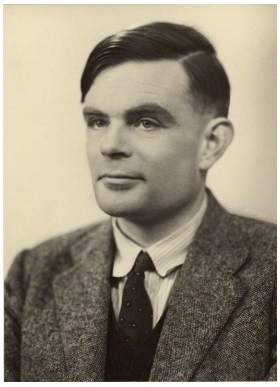
It's still confusing so we need a little bit of History to learn where AI took its roots.
After that, we'll explain the architecture of a neural network, the basis of today AI.
Finally, theory will be very useful to complete our understanding of machine learning.

I.1 History of AI

According to [2, *Larousse*], [3, *Harvard*] or [1, *Britannica*], the journey of modern AI began in the 1950s, with Alan Turing who first described the concept of Artificial Intelligence, based on his troubling concept: the well-named *Turing machine*.



(a) Turing Machine, reconstructed by Mike Davey



(b) Alan Turing ©National portrait Gallery

Figure I.1: The father of AI: Alan Turing and his revolutionary concept

The *universal Turing machine* is considered as the basis of AI. At this time, in 1936, the British mathematician outlined a computer-like machine that can read and modify a program of symbols stocked in a tape memory using a scanner moving forward and backward the tape: nothing very awesome yet.
The revolutionary concept he introduced was that the program also control the movement of the scanner. As a matter of fact, the machine can modify or even improve its own performance by modifying the script in the tape memory.

Using the work of Turing, 3 scientists from the *Research and Development Corporation* proved the concept of Turing with what a lot of people consider as the first working AI program, the *Logic Theorist*, which can reason logically by itself - *one of the most important aspect of intelligence for many of us.* This program was able to solve many mathematical problems without external help, which was a very big step to create a stand-alone machine.

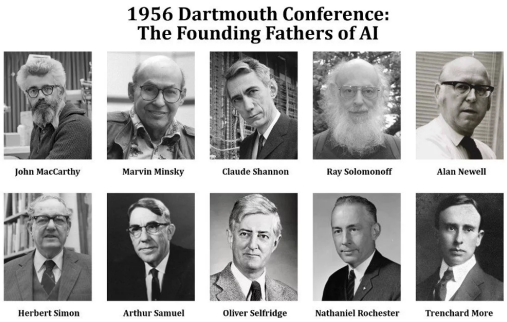


Figure I.2: The beginning of AI ©INDIAai

The *Logical Theorist* opened the gate to many application to logical self-minded computer. At that time, we couldn't talk about a *real intelligence*. Indeed, in the late 1950s and later, some of the greatest IT genius worked on a smart computer that could beat human at chess, considered as a game that can prove that you can think by yourself [2].
But even if **IBM** created *Deep Blue*, a supercomputer specialized in chess playing that beat Garry Kasparov - *the World Chess Champion from 1985 to 2000* - during their second game, chess computers weren't really intelligent: thanks to its 256 parallel processors, *Deep Blue* could study about 200 million potential moves per second. Assuming this, a computer chess use its power to predict moves that will most likely enable him to win the game [1] [2].

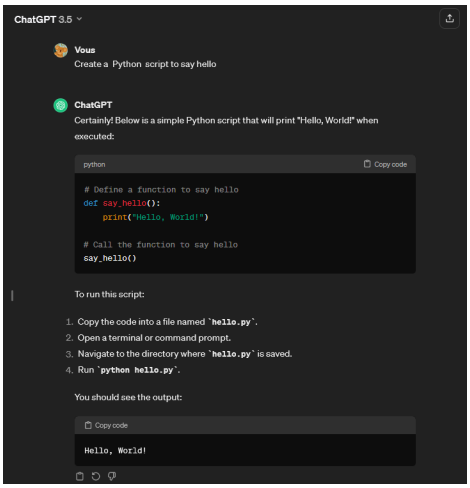


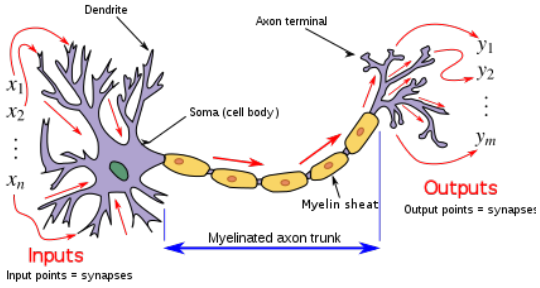
Figure I.3: An example of chatbot: *ChatGPT 3.5*

Chess computers as *Deep Blue* were a first step to create machines that think for themselves. But today, many AI-algorithms are playing with datas all around the world, like *ChatGPT* and *DALL-E* from **OpenAI** or *Gemini* from **Google**.
Those AI can understand requests from the users and answer in many different ways, as a human could, and can be used to do many many thing, not only one task. Today AI are way much sophisticated than **IBM** bots because they aren't based on the same architecture. *Deep Blue* and its predecessors were probabilistic machines using the brute force of an advanced computer to find a combination of move that may led them to win where *ChatGPT* and its friends are what we named ***Neural Networks***. Let's now talk more about neural network.

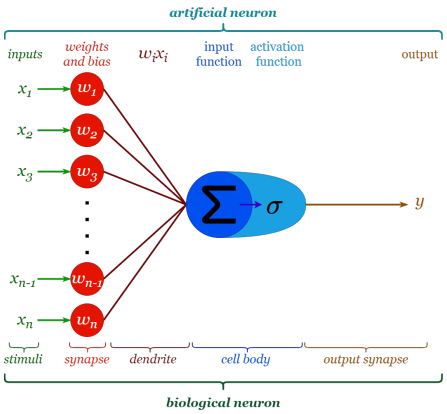
I.2 Perceptron or, the surface of deep learning

Artificial Intelligence has tried, since its beginning, to reproduce the way humans think. In 1943, following this idea, Warren McCulloch and Walter Pitts put forward the idea of simulate a brain using artificial neurons, which are *logical neurons* which response to stimulation [1] [2].

This concept is called ***McCulloch-Pitts neuron*** - *or MCP-neuron*. Technically, MCPs are mathematical activation functions with triggering threshold that respond to inputs and give an output. Those artificial neurons replicate biological neurons :



(a) A biological neuron
©Prof. Loc Vu-Quoc



(b) Schema of a perceptron

Figure I.4: How an artificial neuron work based on biology [4]

So a perceptron can be related to a neuron that respond to stimulation using a mathematical sum function - *the input function*.

Each input x_i has more or less influence on the output of the neuron and that's what is traduced by the *weights*: the more an input x_i has influence, the greater is its weight w_i. So at this point, a neuron receive n inputs x_i and adds them together with more or less weight to give its output:

$$y = \sum_{i=0}^{n} w_i x_i$$

Like in a brain, we don't want all the neurons to be activated in the same time and that's why we are using an *activation function*, σ. Those functions are mainly used to set an activation threshold in combination with a *bias* b:

$$y = \sigma\left(\sum_{i=0}^{n} w_i x_i - b\right)$$

There is a lot of activation functions but we concentrated ourselves only on 4 of them [5] [6]:

- **Rectifier Linear Unit:** $f(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x \geq 0 \end{cases}$, *most common activation function used in Deep Learning ;*
- **Sigmoid:** $\sigma(x) = \dfrac{1}{1+e^{-x}}$, *mainly used to make probability during a binary classification ;*
- **Hyperbolic tangent:** $\tanh(x) = \dfrac{2}{1+e^{-2x}} - 1$, *to normalize entry values between* -1 *and* 1 *;*
- **Softmax:** $\sigma(x_i) = \dfrac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ for $i = 1, \cdots, K$ and $x_i \in \mathbb{R}$, *used to provide a multi-class determination.*

Which are displayed here:

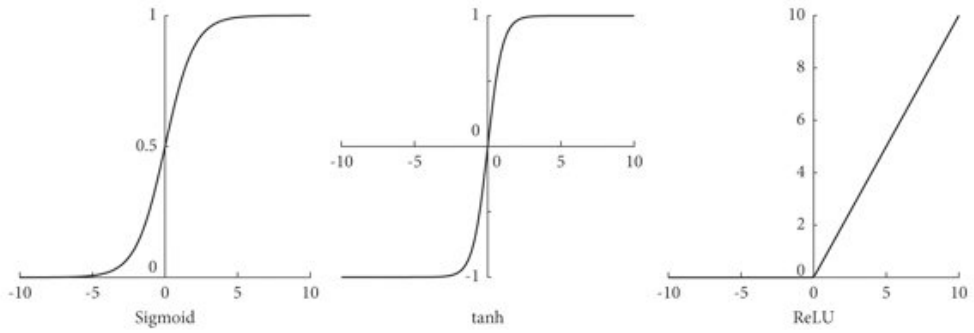


Figure I.5: Curves of the Sigmoid, Tanh, and ReLu activation functions
©Rui Jin and Qiang Niu

Perceptron - *or artificial neurons* - must be seen as the elementary blocks that will allow us to create a complex an fully functional brain. All today AI are based on well-assembled perceptron in different layers and to understand it better, designing an IT brain is the purpose of the next part.

II Deep dive in neural network

In the first part of this report, we discovered the origin of AI and the basis of artificial brains: the perceptron. For the moment, AI is nothing else than a mathematical function with multiple inputs and an output, nothing really *intelligent*.
But take a deep breath and dive in the beautiful world of neural networks with us!

II.1 Neural network theory

To go simply first, a neural network is a connection of many perceptrons spread in successive layers - *we will discuss Dense Neural Networks (DNN), i.e. fully connected neurons* [7] [8]. As a human, an AI has input ports - *eyes, ears and any other part of our body* - that give information to the brain - *the hidden layers* - to process them and give an output:

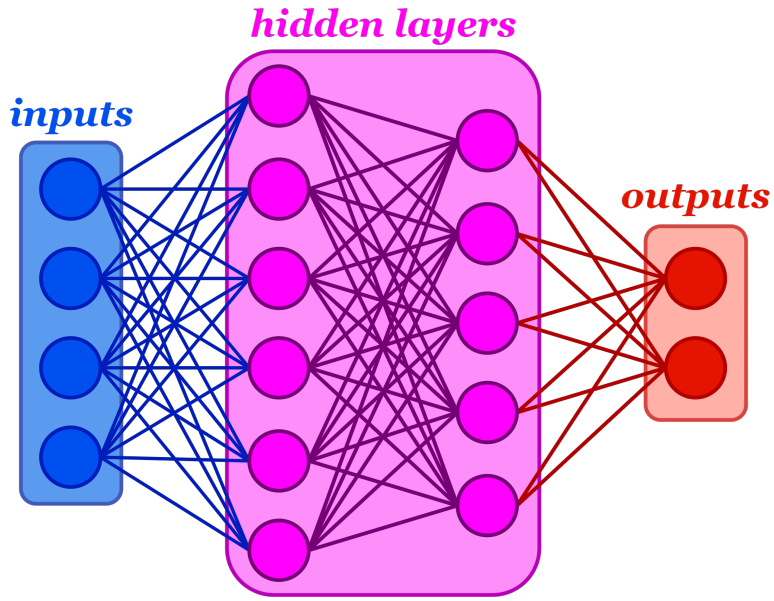


Figure II.1: Representation of a basic neural network

This schema shows exactly how a neural network: each neuron in each layer can be seen as a single perceptron - *described in the previous part* - which take as inputs all the outputs from the previous layer. In that case, each neuron has specific weight values associated to its inputs and a specific bias that ensure a threshold to its activation function.

The goal of a neural network is, depending the inputs, to give the correct output.
For example, an AI can be used to classify picture. What we want it to do is to say if there is or not a dog in an image. In that case, inputs are the pixels of the picture and output is the probability to have a dog in it.

To achieve this goal, the purpose is to *train* the model. Training is the longest part during development of an AI. Let's consider the following schema which represent the training cycle of a neural network:

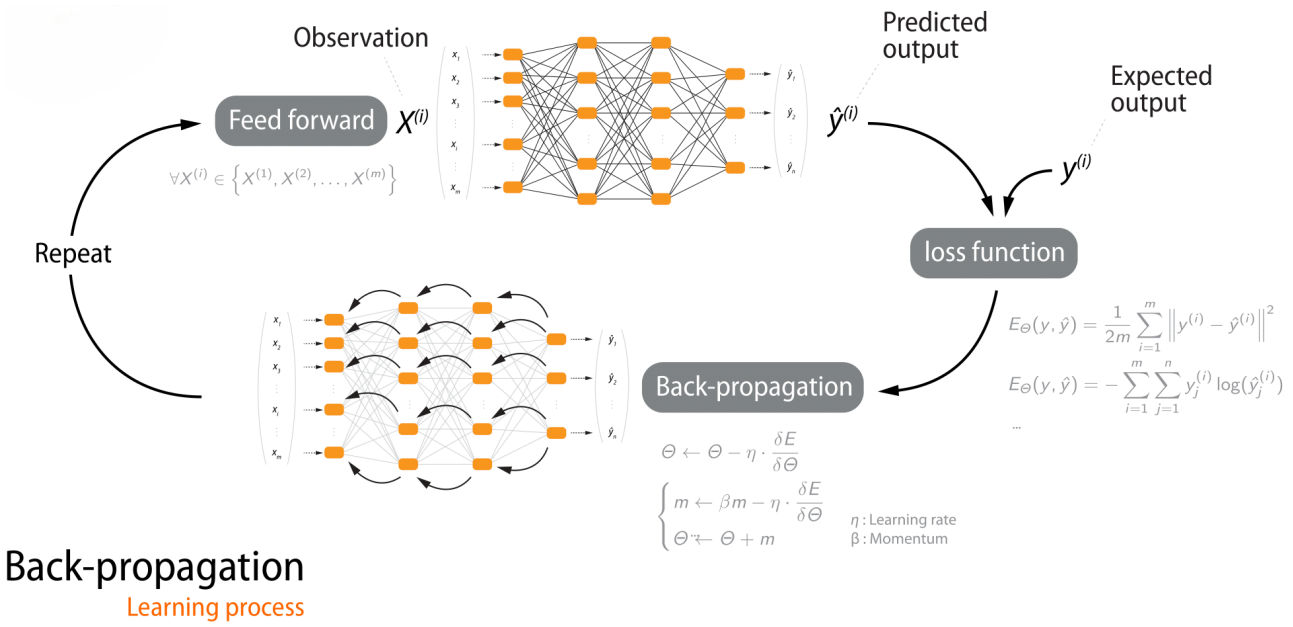


Figure II.2: Training cycle of a dense neural network ©[9, FIDLE]

A well-trained model is a network in which the weights and bias of each neuron are perfectly balanced to give the desired output in response to a certain input. And to find those correct parameters, you must use the training cycle presented above. So, we have to focus on it to understand how an AI is really working.

To simplify things, training consist of giving the network a lot of inputs where the desired output is known, then find when it has made mistake and correct its parameters accordingly. Now, let's go deeper!

II.1.1 Feed forward

First of all, we must remember that a DNN is just a mathematical function with parameters. So when we give inputs to this function, it give us back an output depending of those parameters. During training, we use inputs for which we know the output that the network should give us, it is called a *training dataset* in order to correct the parameters of the function.

The training is a series of ***epochs***. An epoch is a complete passage of the entire dataset in the network. And those epochs are divided in ***batches*** which are the number of entry given to the model before modifying its parameters to reduce memory use [10].

II.1.2 Comparison and loss function

During a batch, when the model is fed and give an output $\hat{Y}$, this output is compared to the desired output Y. The ***loss function*** is used to evaluate the accuracy of the model on the batch. There is a lot of loss function, as the *Mean Squared Error*:

$$MSE(\hat{Y}, Y) = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y}_i - Y_i)^2$$

where N is the number of input data from the batch, $\hat{Y}_i$ is the model output for the i^{th} data and Y_i is the desired output associated [9].

So we have to minimise this function over to fit the model well, and that is where *back propagation* take place.

II.1.3 Back propagation

Even if this looks like an awful word, the ***back propagation*** is nothing else than minimising the loss function using a ***gradient descent method*** from the output layer to the input one. This part is maybe the most important from the entire training process, and this is why we will discuss it in more detail in the next part.

II.2 What really is back propagation?

So back propagation is a *gradient descent method*. In other terms, we want to minimise the loss function of the model and to do it, we have to correct each parameter depending its influence on this loss function.

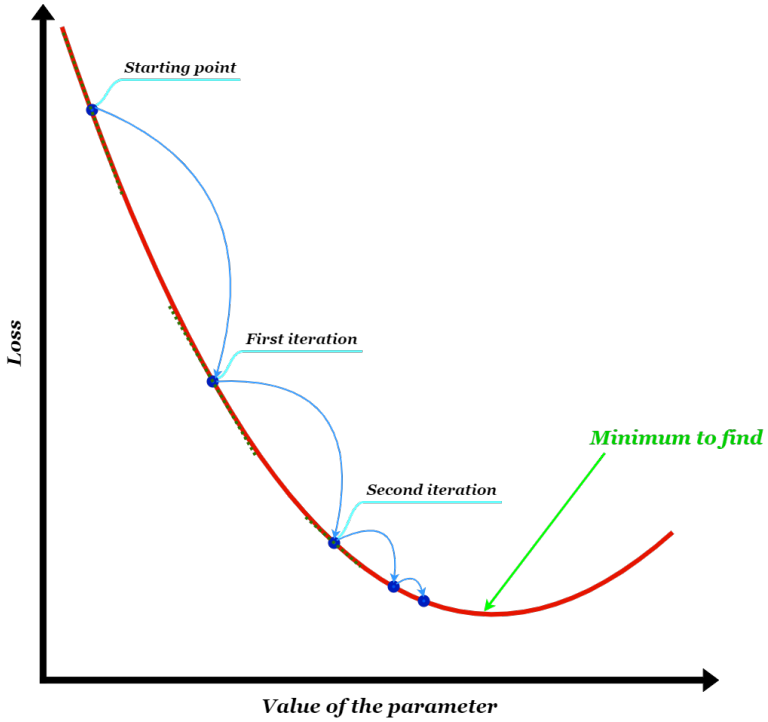


Figure II.3: Illustration of a gradient descent method to find the minimum of a function

II.2.1 Purpose of back propagation

Mathematically, assuming that there is a n-dimensional function f with n which take as input $v \in \mathbb{R}^n$. This function could be expressed using its n variables x_i for $1 \leq i \leq n$, and its gradient could be seen as the partial derivatives of f with respect of its n variables [8] [11] [12]:

$$\nabla f(v) = \begin{pmatrix} \dfrac{\partial f}{\partial x_1}(v) \\ \dfrac{\partial f}{\partial x_2}(v) \\ \vdots \\ \dfrac{\partial f}{\partial x_n}(v) \end{pmatrix}$$

Considering this, the gradient descent method is used to minimise the parameters of the loss function C - *between the output layer and the desired output.* Those parameters are all weights w and bias b of the model. At each iteration $k+1$, the algorithm is the following for a parameter x - *which can be a weight or a bias*:

$$x_{k+1} = x_k - \alpha \nabla C(x_k)$$

NOTE 1: The learning rate α

The parameter α is the learning rate (LR), a value used to modify the influence of the gradient. A too small LR implies a too long convergence time ; a too big LR increases the risk of missing the minimum point

II.2.2 Calculate the gradient

Even if this algorithm looks very simple, the big part of the job is to calculate the gradient of C. In fact, during a gradient descent, each layer has an influence on the previous one. So an easy way to calculate ∇C is to start from the output layer and go backward to the first hidden layer. But this method needs a very well indexation of each parameters of the networks, which is a tedious work.

All the following is inspirited by different works [8] [12] and will refer to the following schema for indexation:

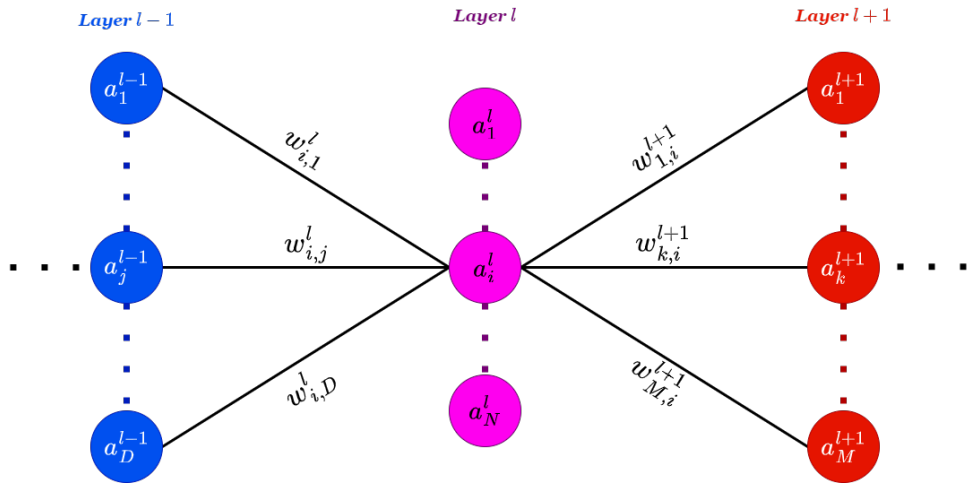


Figure II.4: Indexation of the parameters of a neural network

NOTE 2: Legend of the schema

On this schema, l is the index of the layer we are working in - $l \in |[0; L]|$ - and each neuron is indexed from 1 to D, N or M depending on the layer.

$a_i^l = \sigma(z_i^l)$

$z_i^l = (\sum_{j=1}^{D} w_{i,j}^l a_j^{l-1}) + b_i^l$

So let's find the gradient of C depending on the parameters associated with the i^{th} neuron of the l^{th} layer, a_i^l. Those parameters are its bias, b_i^l and the weights between it and the neurons from the previous layer $l-1$, i.e. $w_{i,j}^l$.

We came looking for prosperity but we found $\frac{\partial C}{\partial w_{i,j}^l}$:

$$\begin{aligned}\frac{\partial C}{\partial w_{i,j}^l} &= \frac{\partial a_i^l}{\partial w_{i,j}^l} \times \frac{\partial C}{\partial a_i^l} \\ &= \frac{\partial z_i^l}{\partial w_{i,j}^l}\frac{\partial a_i^l}{\partial z_i^l} \times \sum_{k=1}^{M}\left(\frac{\partial a_k^{l+1}}{\partial a_i^l}\frac{\partial C}{\partial a_k^{l+1}}\right)\end{aligned}$$

with:

$$\begin{aligned}\frac{\partial z_i^l}{\partial w_{i,j}^l} &= a_j^{l-1} \\ \frac{\partial a_i^l}{\partial z_i^l} &= \sigma'(z_i^l)\end{aligned}$$

and for the last sum:

$$\begin{aligned}&\frac{\partial C}{\partial a_k^{l+1}} \text{ , calculated in the } l+1^{th} \text{ layer} \\ &\frac{\partial a_k^{l+1}}{\partial a_i^l} = \frac{\partial z_k^{l+1}}{\partial a_i^l} \cdot \frac{\partial a_k^{l+1}}{\partial z_k^{l+1}} \\ &\qquad = w_{k,i}^{l+1} \cdot \sigma'(z_k^{l+1})\end{aligned}$$

to conclude:

$$\frac{\partial C}{\partial w_{i,j}^l} = a_j^{l-1} \cdot \sigma'(z_i^l) \times \sum_{k=1}^{M}\left(w_{i,j}^{l+1} \cdot \sigma'(z_k^{l+1}) \times \frac{\partial C}{\partial a_k^{l+1}}\right)$$

II.3 Results and review

Using this algorithm and all we learned about machine learning, we created a *Python3.11* script to get the hang of neural network theory. This script, named *BasicNN*, is a *Dense Neural Network* and its goal is to compare two numbers a and b and find the greater one. For instance, if the input of BasicNN is $[12.5, 58.47]$, it must give the output $[0, 1]$ which means $12.5 < 58.47$.

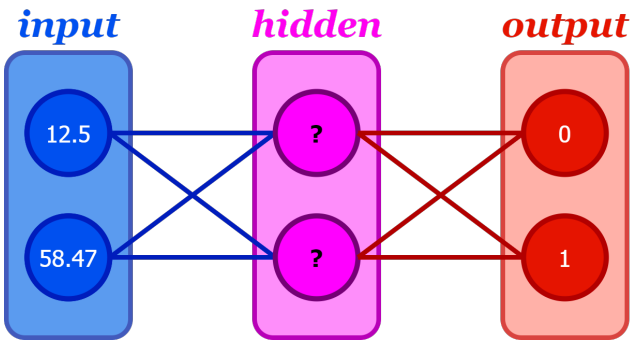


Figure II.5: Representation of *BasicNN*, a neural network that compare two numbers

Our script allow the user to make a complete custom network. But to have fast and easy to interpret results, we are using a very simple DNN with the architecture $2-2-2$, i.e. 2 input neurons - *the two numbers* -, a hidden layer with 2 neurons and the output layer.
This structure should be sufficient to compare 2 numbers because this task is very easy and could be translated by a mathematical function: $max(a,b) = \frac{a+b+|a-b|}{2}$ [13]

NOTE 3: Activation function and output

The activation function that we used is *sigmoid* for the last two layers. It gives, as output, the probability that the number a is the greater one but also the probability that the number b is the greater one. What we have as output are the probabilities of the input belonging to each of the classes "*a is the greater number*" and "*b is the greater number*".

II.3.1 First training of *BasicNN*

To start our journey with our baby AI, we decided to train it a lot, to see how far it could go. This first training session consists of 10000 batches of size 100 with random floating numbers between 0 and 1. Using those training parameters, here is the accuracy as a function of the number of batches:

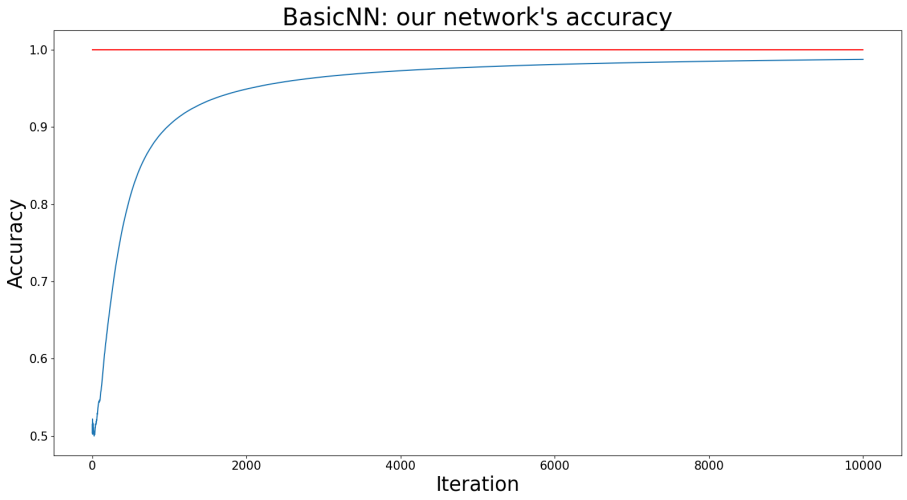


Figure II.6: Accuracy of our *BasicNN* script during its training

This model is very satisfying. According to Figure II.6, its accuracy reached 98.75%. But what we can see here is that after 4000 iterations, the accuracy is not moving very significantly.

The model has been trained on data between 0 and 1, so we should think it can't deal with data out of this range. So, to really conclude on the performance of *BasicNN*, here is its results on a validation test, constituted with a lot of various data that could cripple the model:

Validation Input	Output	True Output	Error
(1.000000, 100.000000)	(0.000, 1.000)	(0, 1)	(0.000085, 0.000079)
(1.000000, 1.000000)	(0.583, 0.417)	(0, 0)	(0.582652, 0.416836)
(0.100000, 1.000000)	(0.000, 1.000)	(0, 1)	(0.000086, 0.000079)
(0.999900, 1.000000)	(0.582, 0.418)	(0, 1)	(0.581614, 0.582123)
(1.000000, 1.000001)	(0.583, 0.417)	(0, 1)	(0.582642, 0.583154)
(0.120000, 0.090000)	(0.786, 0.213)	(1, 0)	(0.214122, 0.213358)
(5.000000, -4.000000)	(1.000, 0.000)	(1, 0)	(0.000101, 0.000093)
(-10000.000000, 10000.000000)	(0.000, 1.000)	(0, 1)	(0.000085, 0.000079)

Table II.1: Validation set for comparing 2 numbers by *BasicNN*

As we can see on the Table II.1, *BasicNN* is working well on most of the validation data outside its training range. But it seems to have issues on data between 0 and 1. We have to go further on the validation of our model.

II.3.2 Validation of *BasicNN*

To valid our model, we chose as acceptability criterion the following one, based on the error ε between the desired output y_{true} and the real output y:

$$\varepsilon = max(|y_{true} - y|) < 0.5$$

With this criterion, if the output of the network differ more than 0.5 from the desired output, it means that the model couldn't find the right class for the input. Here is an example of how our criterion is working:

Acceptability criterion

Here are the 2 outputs of the model: $p_a = \mathbb{P}(a > b)$ and $p_b = \mathbb{P}(b > a)$

Let's consider an input $X = (a, b)$ where $a > b$:

$$BasicNN(X) = \begin{cases} True\,, \text{if } p_a > 0.5 \text{ and } p_b < 0.5 \\ False\,, \text{if } p_a \leq 0.5 \text{ or } p_b \geq 0.5 \end{cases}$$

We define the ***acceptability factor*** as:

$$\theta(X) = 1 - \varepsilon > 0.5$$

So we first used this criterion on a large range of data, comparing all the numbers between -100 and 100. The greener the result is, the more acceptable is the output of the model and red translate the most unacceptable results, i.e. $\theta(X) \leq 0.5$:

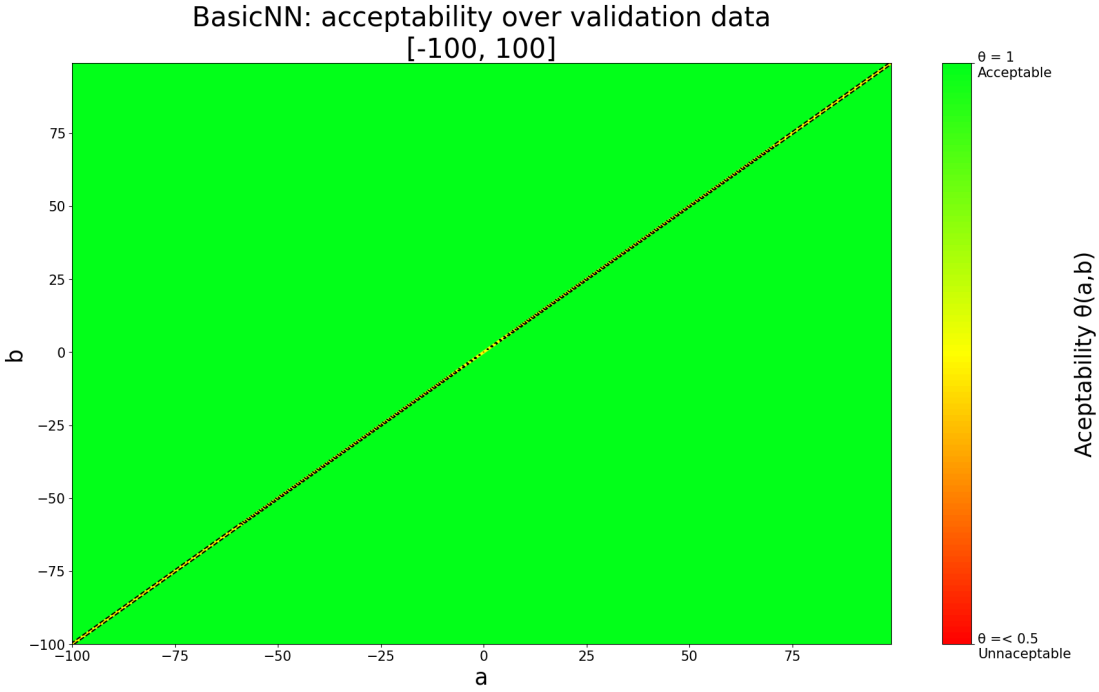
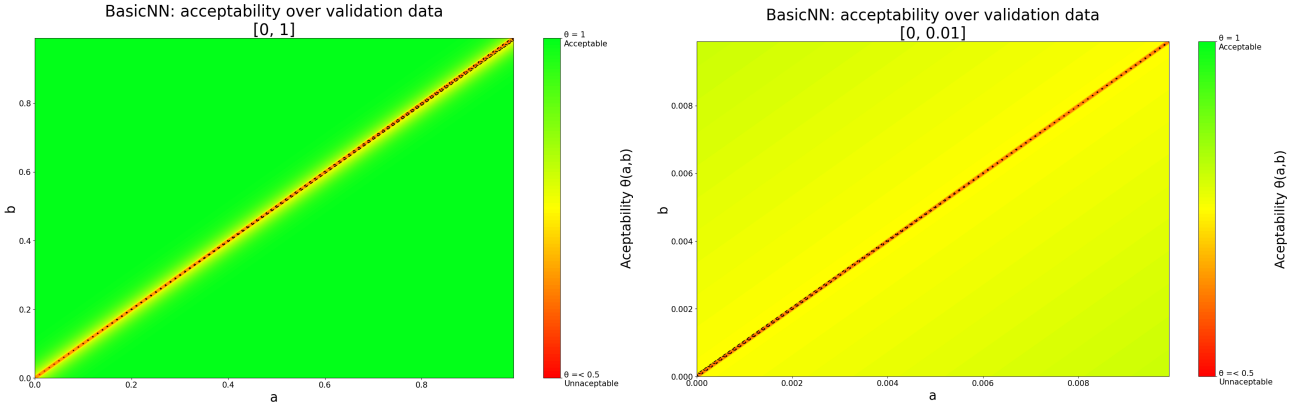


Figure II.7: Acceptability criterion on data from -100 to 100

This Figure II.7 shows us 2 main things:

- *BasicNN* perfectly compare numbers outside its training dataset ;
- Errors of prediction are mainly concentrated where the numbers compared are nearly the same.

Because the distribution of validity seems to be symmetrical about the point $(0, 0)$, we refine our validation set, between 0 and 1 first and then between 0 and 0.01, to find more detailed information about the model's error:



(a) Validation on $[0, 1]$

(b) Validation on $[0, 0.01]$

Figure II.8: Acceptability criterion on more restricted data

So, referring Figure II.8, we can conclude that our *BasicNN* model is doing a great job to analyze its inputs. In accordance with Figure II.6, our network is very accurate to classify inputs between the 2 classes, except when they are really close. This issue is mainly due to the construction of our dataset: the classes of the output are $a > b$ and $b > a$, there is no case for $a = b$, so the AI doesn't know what to do when the a and b are too close. This issue is well shown on Figure II.8 (b): a and b are very nearby, so the model is less accurate.

II.4 Comparison with TensorFlow

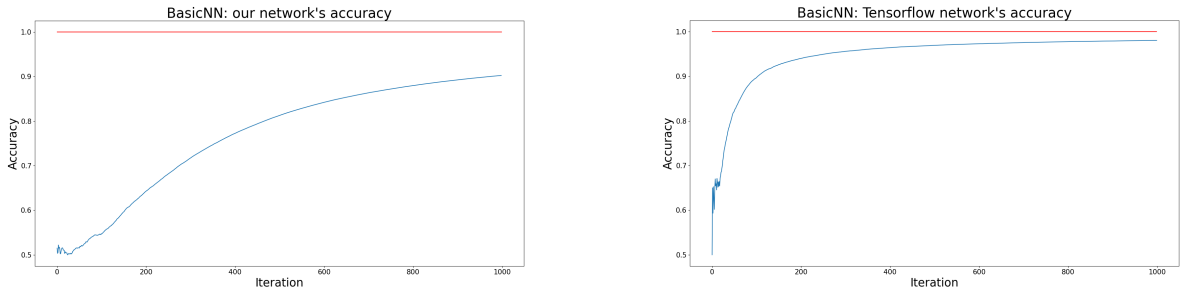
A lot of people worked before us on AI and neural networks, so there are many libraries that could help us make a better model than *BasicNN*. The one we decided to use is ***TensorFlow*** - *TF* - with ***Keras***, a very famous and user-friendly library than can allow us to completely create a neural network.

Figure II.9: ©TensorFlow

Another advantage of TensorFlow is its ease of use when you are developing an image recognition model, as we shall see later.

To compare our DNN with the TF's one, we recreated the same architecture that we used in *BasicNN* but with TF's intrinsic functions. After, we trained both models to judge their performances. What we compared are the accuracy of each model and their training time.

That is how we found those two accuracy graphics:



(a) Our *BasicNN* model

(b) TensorFlow's model

Figure II.10: Comparing the accuracy of the 2 models after $1,000$ batches

Indeed, TF's model reached an accuracy of 98.04% with $1,000$ batches of 100 inputs where, with the same training set, our model reached only an accuracy of 90.24%. *BasicNN* needed $10,000$ batches of size 100 to reach an accuracy of 98.75%, according to Figure II.6.
The problem is that, for a $1,000$ batches training set, *BasicNN* trained itself during $3.335\ s$ and it is still less accurate than TF's model which trained only for $0.783\ s$. To reach the same accuracy, our model took $34.43\ s$.

This issue must come from the gradient descent, even if we tried to use the same algorithm than in TensorFlow.
Firstly, it is not excluded that we haven't exactly the same loss function than in TF.
Secondly, TF can use a stochastic gradient descent which could be more optimized than a full gradient as we used. We tried to deactivate it, but we may have fail too.

Whatever the differences are, the gradient descent from TF is more optimized than our and, according to this comparison, we decided to use TensorFlow for the rest of our work.

III Two-dimensional image model

So far in this report, we discovered the origins of neural networks, we explained how it works and how they are trained. We also created a full connected neural network from scratch in *Python* to dive deeply in the comprehension of neural networks. Here, let's venture in the realms of visual AI analysis with the CNN.

III.1 What is a Convolutional Neural Network (CNN)

A Convolutional Neural Network is a machine learning model based on a linear operation: the convolution. This type of network is very well suited for analysing visual data because it identifies patterns and extracts features within it.

III.2 CNN Architecture

A convolutional Neural Network is made of several convolutional blocks, each one of them are composed at least of a 2D convolutional layer and a pooling layer.

III.2.1 2D Convolution layer

The convolution operation is done between what is called a kernel -or a filter- and the data to be processed. The kernel is a matrix, usually a square one of size 3, 5, 7 or even 9, where each value are carefully chosen to perform an operation on the data. In image processing, there are several kernel types for blurring, sharpening, edge detection and more.

Mathematically, the convolution operation can be written like below[14].
Let's consider a multichannel image I with d channels and a kernel K with size $k_w \times k_h$ (width $\times$ height), the computed

$$I'(x,y) = \sum_{i=1-\lfloor \frac{k_w}{2} \rfloor}^{\lfloor \frac{k_w}{2} \rfloor} \sum_{j=1-\lfloor \frac{k_h}{2} \rfloor}^{\lfloor \frac{k_h}{2} \rfloor} \sum_{k=1}^{d} I(x+i, y+j, k) \cdot K(i,j,k) \tag{1}$$

The convolution operation can be described as the sum of the wise-product between the kernel and the neighbours of the pixel treated. The figure below schematize the operation.

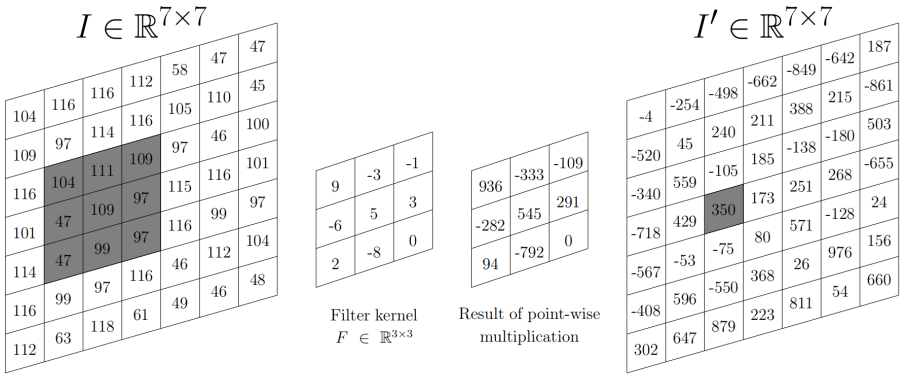


Figure III.1: Visualization of the convolution operation between a kernel of size 3×3 and a one channel 7×7 image ©[[14], Martin Thoma]

The 2D convolutional layer uses the convolution operation with a defined number of filters and return a *features map*.

III.2.2 Pooling layer

In CNNs, pooling layers summarize the outputs of neighbouring groups of neurons. It is called a *subsampling layer* because it reduces the size of the features map.

The most commons' algorithm of pooling are *average-pooling* and *max-pooling* [15], however $\mathcal{L}_2$ and stochastic pooling can be find[14]. For instance, a 2×2 *average-pooling* kernel averages 4 neighbouring elements of an input matrix to compute one element in the output matrix.

Generally, the pooling kernel does not overlap neurons already summarized, we call the *stride* the amount of neuron to shift before summarize others. In practice, according to this article[16], it is more difficult to overfit a neural network with overlapping pooling.

Pooling can be visualize in Figure III.2.

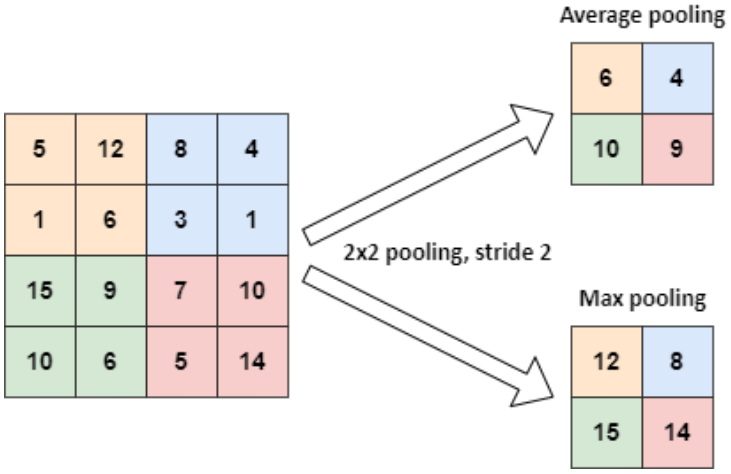


Figure III.2: Visualization of 2×2 pooling apply to a features map with stride equal 2

III.2.3 Dense layer

The dense layer is a basic layer composed of several perceptrons. It is the last layer of the CNN which determinate the class of the image.
CNN model are a least composes of these deferents layers, but to optimize performances, some other layers can be added.

III.2.4 Other layers

The *Dropout layer* improves the performance by randomly set neurons to zero in each layers during training[14][15][16]. It prevents overfitting and complex co-adaptation between neurons[14][16]. The *Normalization layer* it is not necessary when we use ReLU activation function, but it helps generalization. [16]

III.3 Our CNN default model

In this work, we defined a default CNN, it is inspired by the readings we have done[14][17][18]. This model served as a basis for comparison with all *submodels* created to see the influence of hyperparameter on the performance. This default model has 3 convolutional blocks, each block are composed of a 2D convolutional ReLU activation layer and a max-pooling layer. After these 3 blocks, there is one flatten layer which convert the resulted matrix into a single array, an all connected 64 neurons layer with ReLU activation and a classifier head with *softmax* activation function.

Layer type	Output shape	Number of parameters
Input	$256 \times 144 \times 3$	0
Convolution 1	$254 \times 142 \times 32$	896
Max pooling 1	$127 \times 71 \times 32$	0
Convolution 2	$125 \times 69 \times 64$	$18,496$
Max pooling 2	$62 \times 34 \times 64$	0
Convolution 3	$60 \times 32 \times 64$	$36,928$
Max pooling 3	$30 \times 16 \times 64$	0
Flatten	$30,720$	0
Dense	64	$1,966,144$
Classifier head	2	130

Table III.1: Summary of our default CNN

See Figure III.3 for visualization of our default CNN.

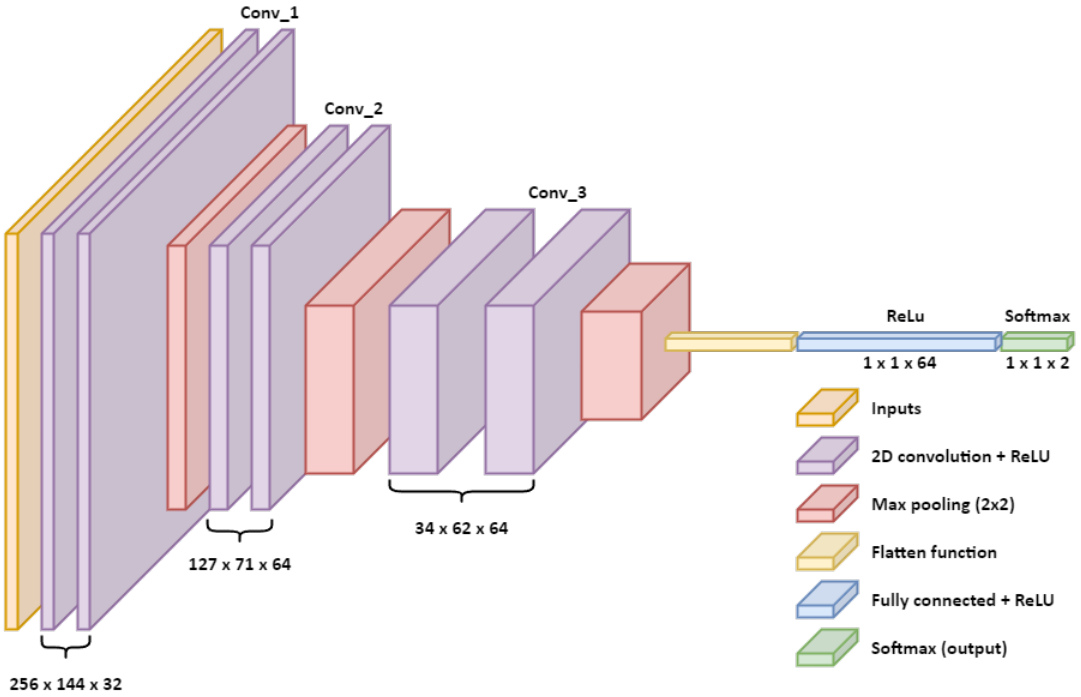


Figure III.3: Visualization of our default CNN

To validate that this model is able to recognize something, we first trained it on CIFAR-10, a dataset that is commonly considered as a standard for image recognition [19]. It appeared that, after less than 10 epochs, the model validation accuracy reached more than 71.41%: this default model architecture can successfully classify the different images of the CIFAR-10 dataset. That will be the starting point of our own architecture.

NOTE 4: Greyscale images

In the following, we will discuss the influence of dataset on performance. To do it, we create another CNN for greyscale image. The only things that change are the *Output shape* of the *Input* layer which become $256 \times 144 \times 1$ and the *Number of parameters* in the *Convolution 1* layer which decrease to 360.

III.4 Dataset

A dataset is a set of organized data which have the same format. It can be text, images, numerical data or whatever else [20]. On a dataset, each data is annotated, i.e. each data are linked to an output class of the AI which is the desired output for this input.

In machine learning, a dataset contains the inputs of a neural network during its training, so it is essential to have a good dataset. Indeed, the more data are various, the more you can train your model to various cases. Thanks to this, the network could be more efficient when facing to data he never encountered because its parameters must have been updated by a quite similar case from the dataset.

So finding a lot of data that correspond to what your model will do is crucial. For us, we had to find two type of pictures: pictures with fish and picture without fish. The problem was that nobody created a such dataset on the internet before, so we had to create our own fish dataset - *OFD.*
To achieve this, we combined two datasets:

1. the Fish Computer Vision Project, with various fish species ;
2. the Large Scale Underwater Image dataset, which we sorted to retain only pictures without fish.

To validate our model, we also found two videos deep sea, one captured by the *BathyBot* [21] and an other one [22]. It will be the reference to judge the accuracy of our AI because it must have been designed and trained to recognize fish in this environment.

The combination of the two previous datasets gave us a training set of 6341 pictures. The problem was that it isn't enough to train a network well. To counter this issue, we had to do what is named *data augmentation* [14] [16]. The goal of this technique is to artificially create data.
To achieve this, there is a lot of possibilities, sorted in ascending order of new data creation:

- flipping the images ;
- scale them up or down ;
- rotate them ;
- adjust their color parameters (contrast, brightness, saturation, etc...) ;
- crops them in different part ;
- many more others.

But because our computers can't allow the model to process too many pictures during its training - *limited by memory use* -, we just used a horizontal flip. With this method, we have a larger dataset of $12,682$ pictures to train our AI.

Finally, we build three type of dataset to find the most efficient type of entry for our network. Each dataset has the same pictures but with different coloration:

1. Vanilla dataset: raw pictures with their original RGB colors ;
2. Greyscale images: conversion from RGB pictures to greyscaled ones, represent the brightness of the picture ;
3. Preprocessed images: application of a Sobel filter to highlight edges in the pictures.

III.5 Towards the most effective classification model

After understanding the way CNN works and building the dataset, we tried to find the most effective model for our application, finding a fish in a deep underwater image. To do it, we explore the influence of hyperparameters on performances.
Hyperparameters are all parameters outside the model, for instance, batch size, activation function, pooling function and number of neurons in dense layer and many others. TensorFlow has many hyperparameters, all will not be discussed in the following.

III.5.1 Finding the best dataset format

In this section, all images have the same format: 256px in width and 144px in height. To find the best dataset format, we always use our default CNN built in subsection III.3.

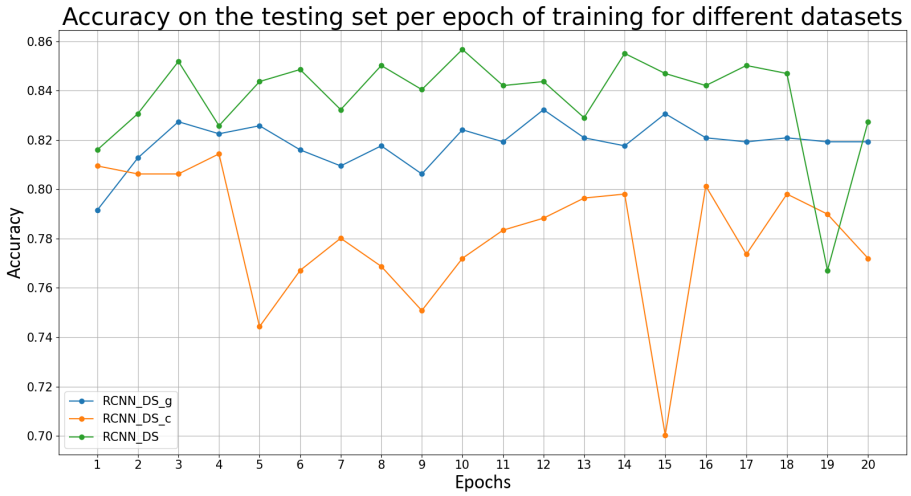


Figure III.4: Accuracy of the neural network on the testing set per epoch of training for different dataset

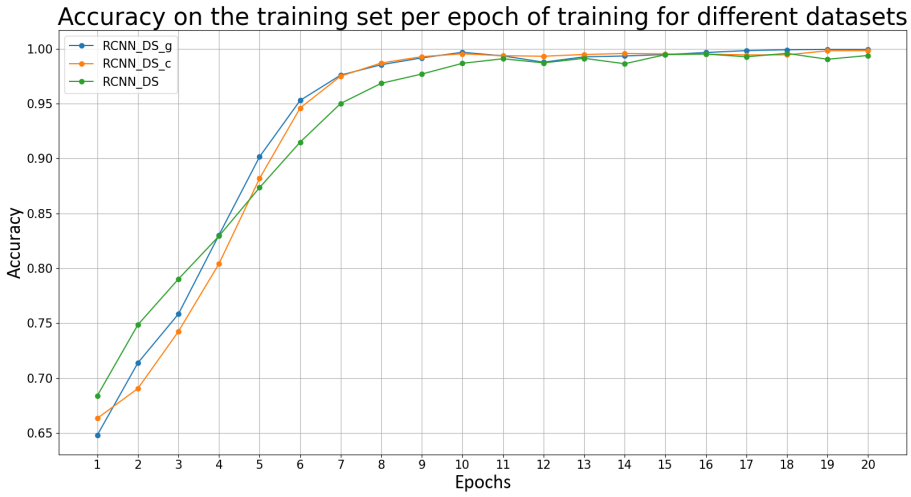


Figure III.5: Accuracy of the neural network on the training set per epoch of training for different dataset

On the Figure III.5 graph, we can see that it is not necessary to go further 10 epochs to have good estimation of the performance of our neural network.

Moreover, Figure III.4 shows that the best dataset to be exploited is the vanilla one with RGB images. The CNN performs, on the 10 last epoch, 1.3% better than the CNN trained by the greyscale dataset and 5.5% better than the CNN trained with the contour detection preprocessing dataset.

Table III.2 gives us a summary of the performance on different dataset format.

Dataset name	Description	Maximum training accuracy (%)	Maximum testing accuracy (%)
DS	Vanilla dataset with RGB images	99.7	85.7
DS_g	Greyscale images	99.9	83.2
DS_c	Preprocessed images, mask of the contours	99.9	81.4

Table III.2: Table summarizing performance of the default model on different dataset

We then use the vanilla dataset to explore other hyperparameter influence.

III.5.2 Influence of the batch size

As a reminder, the batch size is the number of images in one set of training before adapting the weight and bias of the neural network.

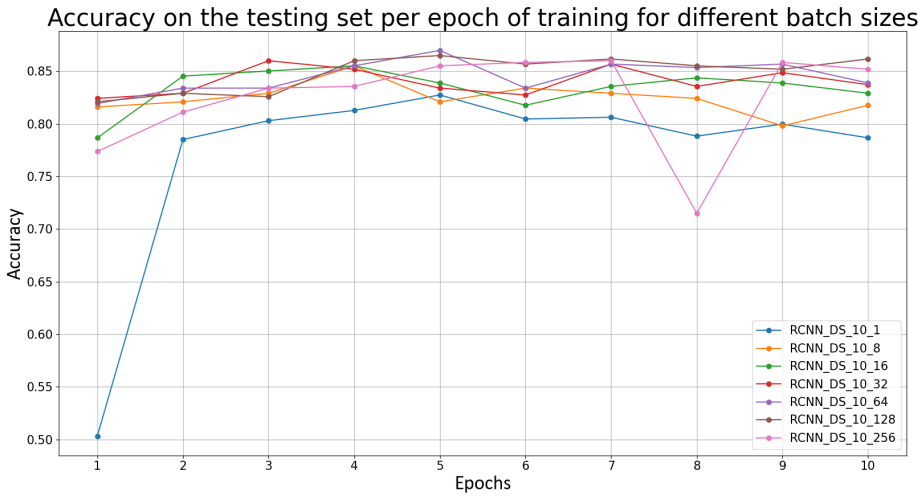


Figure III.6: Accuracy of the neural network on the testing set per epoch of training for different batch size

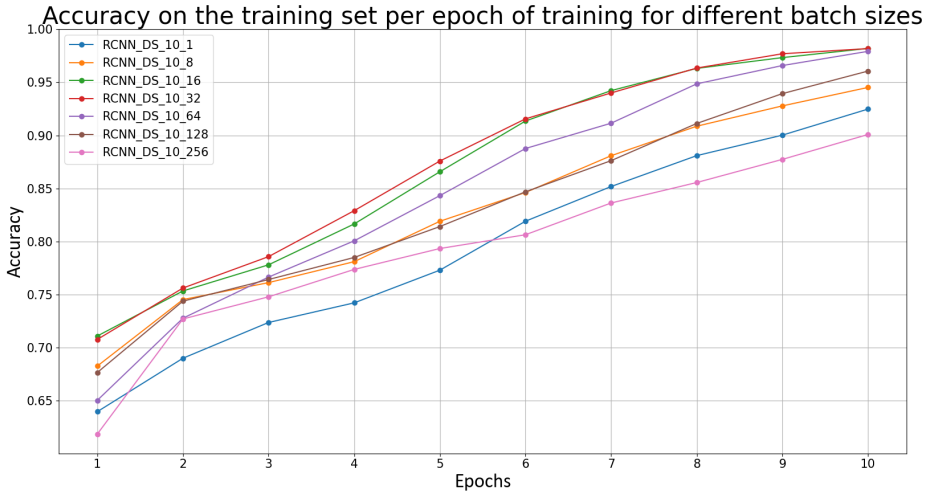


Figure III.7: Accuracy of the neural network on the training set per epoch of training for different batch size

According to Figure III.6 and Figure III.7, we can draw the Table III.3 to summarize performances.

Batch size	**Maximum training accuracy (%)**	**Maximum testing accuracy (%)**	**Average time per image (ms)**
1	92.5	82.7	14.9
8	94.5	85.5	7.6
16	98.2	85.5	6.4
32 (default model)	98.2	86.0	6.2
64	97.9	87.0	6.2
128	96.1	86.5	7.3
256	90.1	86.0	6.7

Table III.3: Table summarizing the performance of the model over different batch size

Table III.3 shows that 64 and 32 images per batch are the best for accuracy performance. 64 images per batch has the best performance for the maximum testing accuracy, this accuracy is the one we try to maximize to have the model as general as possible.
We therefore decided to use a batch of 64 images for our optimized model.

III.5.3 Influence of the pooling function

Here we are going to test the result of the two main used pooling function over the performance of our model. The accuracy performance can be seen on Figure III.8 and Figure III.9.

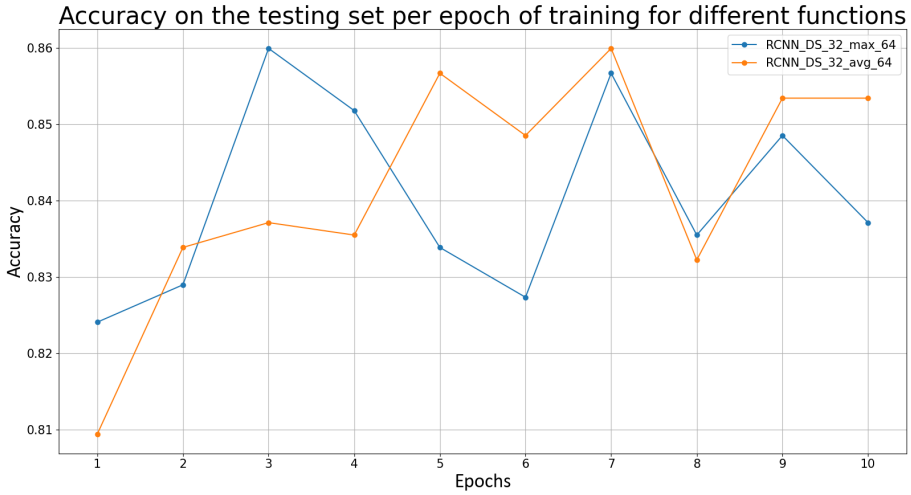


Figure III.8: Accuracy of the neural network on the testing set per epoch of training for different pooling functions

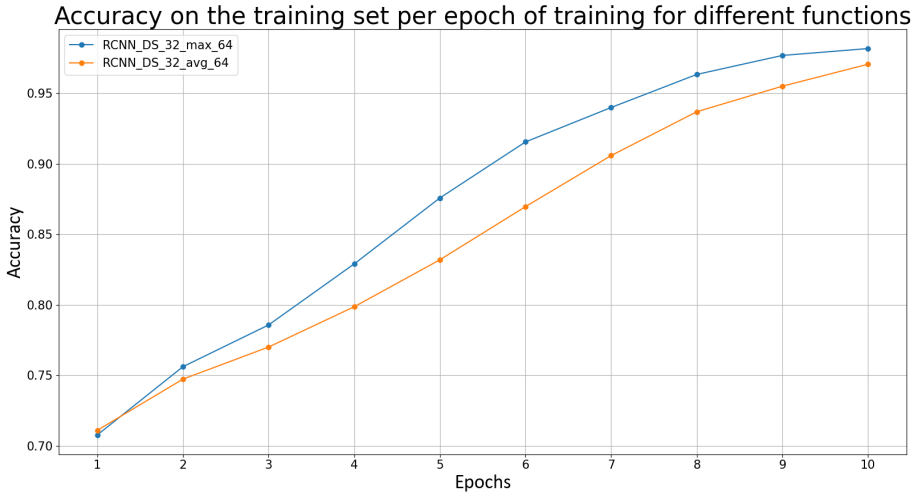


Figure III.9: Accuracy of the neural network on the training set per epoch of training for different pooling functions

See Table III.4 for maximum accuracies on the two different pooling functions.

Model name	**Description**	**Maximum training accuracy (%)**	**Maximum testing accuracy (%)**
CCNN_max_64 (default model)	Max-pooling	98.2	86.0
CCNN_avg_64	Average-pooling	97.1	86.0

Table III.4: Table summarizing performance of the model over different pooling function

As we see in Table III.4, the two function performs quite equivalent, both are use in different stage-of-the-art model [14].
In our case, we select the *max-pooling* layer because it reaches faster good accuracy.

III.5.4 Influence of the *Dense* layer

The *Dense* layer summarize the features found in the CNN block to decide if there is a fish or not in our images. We have therefore tried different type of *Dense* layer. All model names are described in the Table III.5.

Model name	Description
CCNN_64_1 (default model)	One *dense* layer with 64 neurons
CCNN_128_1	One *dense* layer with 128 neurons
CCNN_32_2	Two *dense* layers with 32 neurons
CCNN_64_2	Two *dense* layers with 64 neurons

Table III.5: Description of model name for *dense* layers influence

The result of its training is plotted on Figure III.10 and Figure III.11. The results are summarized in Table III.6

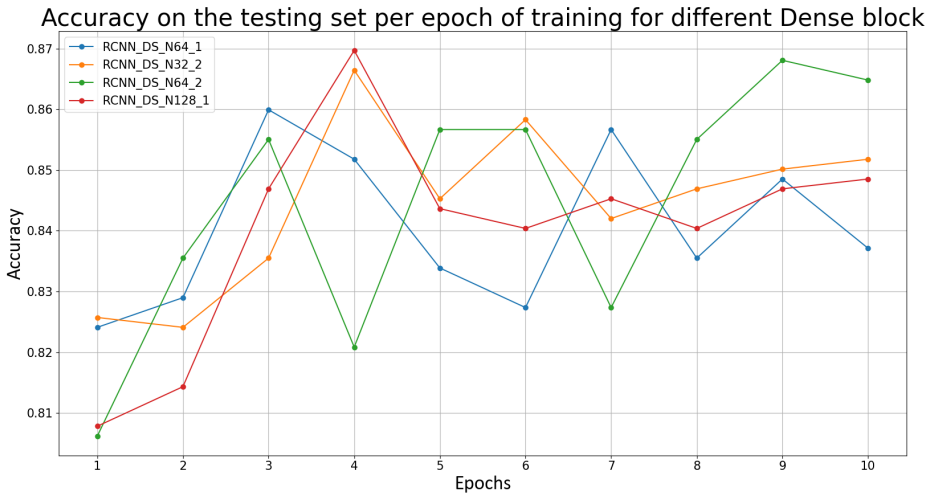


Figure III.10: Accuracy of the neural network on the testing set per epoch of training for different *Dense* layer

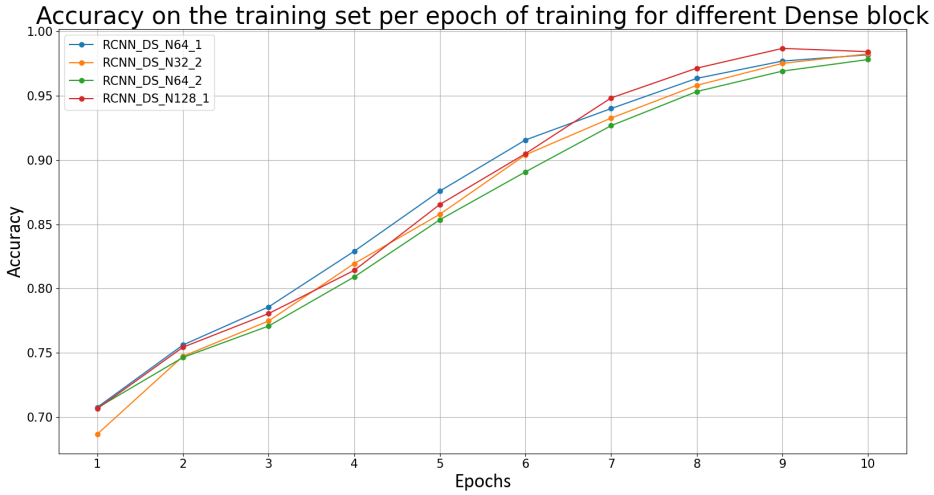


Figure III.11: Accuracy of the neural network on the training set per epoch of training for different *Dense* layer

Model name	**Maximum training accuracy (%)**	**Maximum testing accuracy (%)**	**Number of parameters**
CCNN_64_1 (default model)	98.2	86.0	2,022,594
CCNN_128_1	98.7	87.0	3,988,866
CCNN_32_2	98.3	86.6	1,040,514
CCNN_64_2	97.8	86.8	2,026,754

Table III.6: Table summarizing performance of the model over different dense layer

According to Table III.6, the model the most effective for detecting fish in an image is *CCNN_128_1*, it reaches 98.7% accuracy on the training set and 87.0% on the testing set. These are good performances, however, the number of parameters to train double and the time and resources increases. We therefore choose the model *CCNN_64_2*, it combines "low" number of parameters with high maximum testing accuracy.

III.6 Conclusion

After all these comparisons, we are now able to choose the best direction to build a model that combine performance and low execution time. Its performances and structure are given in Table III.7.

Layer type	Output shape	Number of parameters
Input	$256 \times 144 \times 3$	0
Convolution 1	$254 \times 142 \times 32$	896
Max pooling 1	$127 \times 71 \times 32$	0
Convolution 2	$125 \times 69 \times 64$	$18,496$
Max pooling 2	$62 \times 34 \times 64$	0
Convolution 3	$60 \times 32 \times 64$	$36,928$
Max pooling 3	$30 \times 16 \times 64$	0
Flatten	$30,720$	0
Dense	64	$1,966,144$
Dense	64	$4,160$
Classifier head	2	130

Table III.7: Summary of our optimized Convolutional Neural Network after all comparison

The total number of parameters is $2,026,754$.

This model performs very well. The maximum accuracy on the training set is 99.96%, on the testing set, the model achieve an accuracy of 87.0%.

See Figure III.12 and Figure III.13 for performance per epoch.

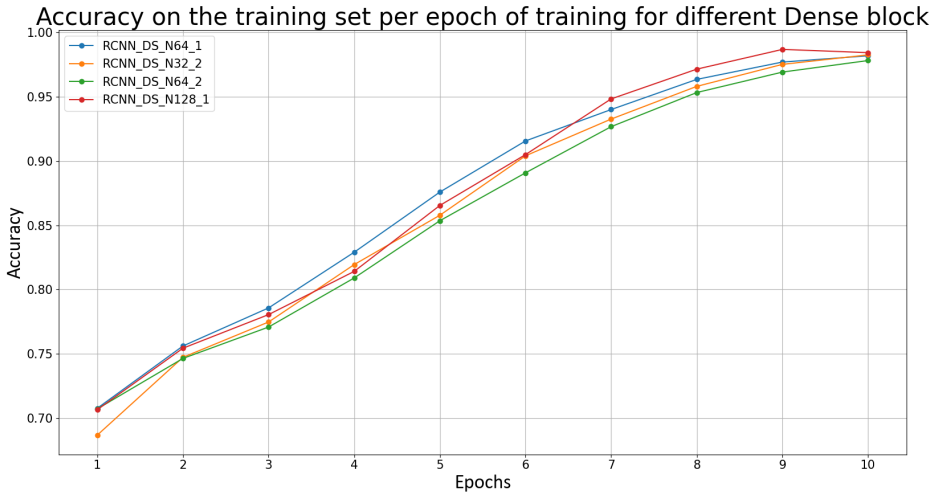


Figure III.12: Accuracy of the neural network on the testing set per epoch of the optimized model

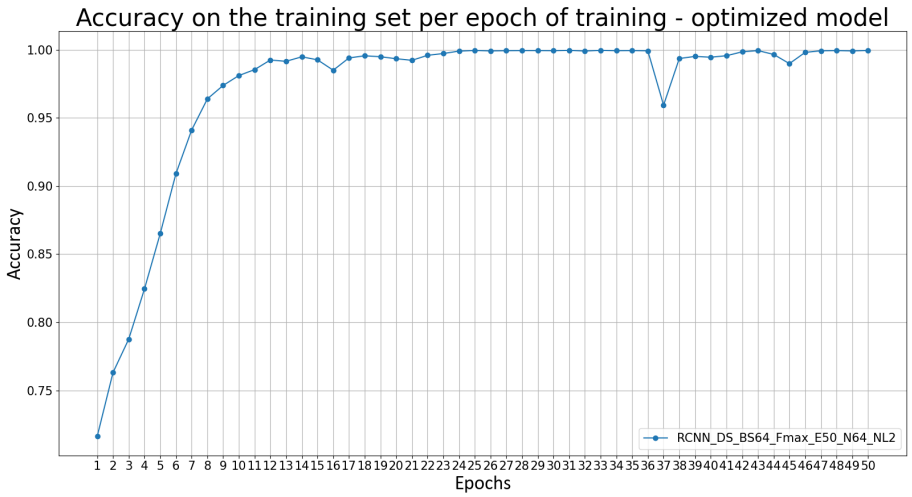


Figure III.13: Accuracy of the neural network on the training set per epoch of the optimized model

NOTE 5: Validation on CIFAR-10

Just to compare, we trained this model on CIFAR-10 again. It is still performant, but a little less than before optimization: 71.17% of accuracy.

IV Our app

The goal of our project was to classify image from a video. To improve the usability of our neural network, we decided to build an application called ***Hydro'Cognition***.

This application is made with python using mainly two libraries: *TKinter* and *CustomTKinter*, to preview video, we use an open-source code [23].

The application is composed of 5 compartments, the right part of the application is dedicated to image selection and image visualization.
On the left side, from top to bottom, we can find a frame to choose which model is loaded, a frame to process video or images and a frame of logging. In this last frame, all information are printed to ensure no one is missed.

All functionalities of our application are listed below:

- Load custom model
- Preview video
- Process video
- Load images
- Preview images
- Process images

See Figure IV.1 to preview the application.

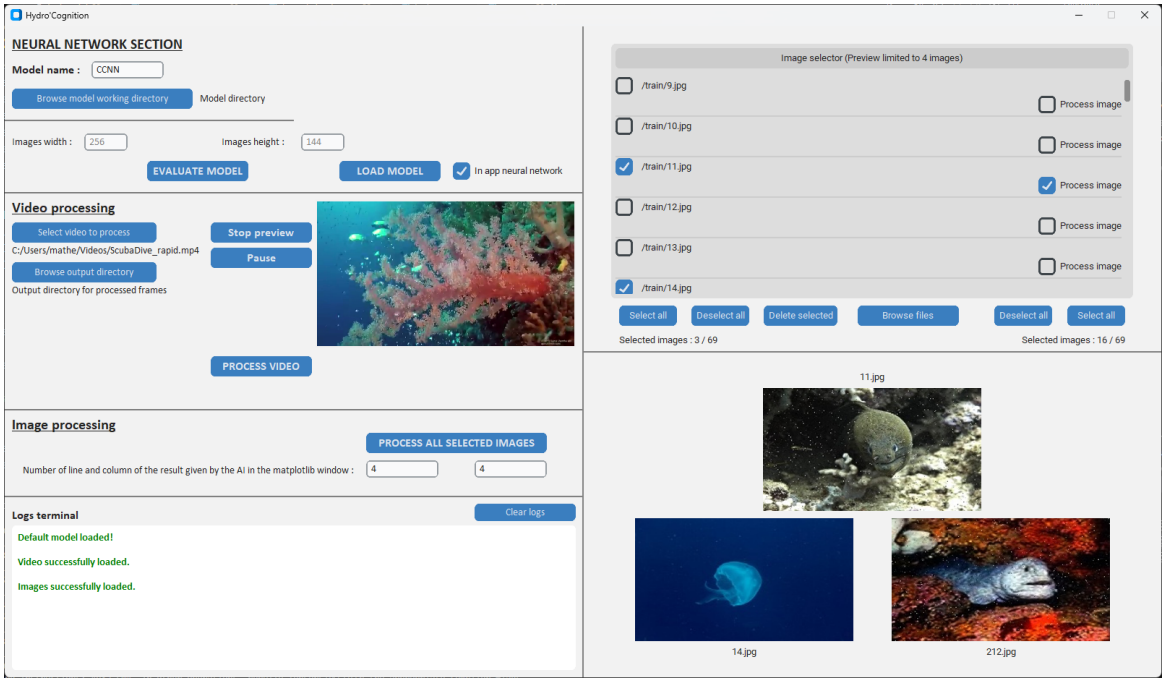


Figure IV.1: Preview of the ***Hydro'Cognition*** application

All codes and our application can be found open source on *GitHub* here. How the application works is also described there.

V Development opportunities

To go further in our project, we identified some points to enhance our model. We haven't taking too much time on developing those points, but we will discuss them here.

V.1 Changing our architecture

As developed in the subsection III.5, we took a lot of time trying to find the best hyperparameter for our model. But there are two things left to do:

1. adding a Max-Pooling layer as entry ;
2. comparing combination of hyperparameter.

Firstly, adding a Max-Pooling as entry of our model could be a good idea. Indeed, as developed in the subsubsection III.2.2, a max-pooling layer increases contrast in a picture. So using it as entry, before the first convolutional layer, should help the model finding point of interest in the processed images.
After a quick test, this improves network performance on the validation set by $+0.85\%$. Even if it isn't as efficient as expected, it could be a solution to improve our model.
This experiment gave us a glimpse on what should happen if we also tried to change the architecture of the image processing part of our model, i.e. the number of pooling or convolutional layers. But it may have taken too much time to deal with.

Secondly, during our performance comparison in subsection III.5, we found the best hyperparameter of our model **separately**. The fact is a neural network is fully connected, so each hyperparameter affects all the other. We should have run a larger comparison campaign to really understand the influence between the parameters of our model architecture and find the best combination of hyperparameter.
Unfortunately, a such campaign is too heavy to be conducted manually by a human. So to find the best CNN model to fit our problem, it could be technically possible to use a *genetic algorithm* to automatically find the best model [24]. But this is an algorithm which requires too much memory to be conducted on our own.

V.2 Data warfare

In machine learning, data are the sinews of war, as explained in subsection III.4: the more you have precise data, the more your model could be well-trained.
And one huge issue of our project is the dataset we used: it is not suited to the purpose of our project. Indeed, the fish pictures of our dataset aren't the type of fish or environment the *BathyBot* may encounter deep sea: fish aren't same shaped in the depths, there is less coral or other water plant and the colors at $2,000\,m$ depth aren't the same as near the surface.

To counter those problems, we may have some solution:

1. recolour training images to match colours found deep sea, but fish shapes will still be wrong ;
2. find a more accurate dataset with images recorded by camera or ROV really deep ;
3. use black and white pictures instead of coloured ones to be sure that the AI detect the shapes of the fish instead of some common colour schemes between fish pictures.

V.3 Object detection

Finally, we wanted to go further than the project proposed by the MIO: more than detect a fish in a video, we also wanted to find out where in each frame it can be found. This technique is named *object detection*.

V.3.1 What is object detection?

In terms of computer vision, there is 3 main techniques [25] [26]:

1. Image classification: what our model is actually doing, i.e. labelling images with different classes depending on what the network recognize in each picture ;
2. Object detection: that is what we wanted to do, i.e. identifying and locating objects in a picture by putting them in different boxes ;
3. Image segmentation: similar but more difficult than object detection because more precise, identifying each pixel of an object in a picture.

Here is a figure from IBM to represent the difference between those 3 techniques:

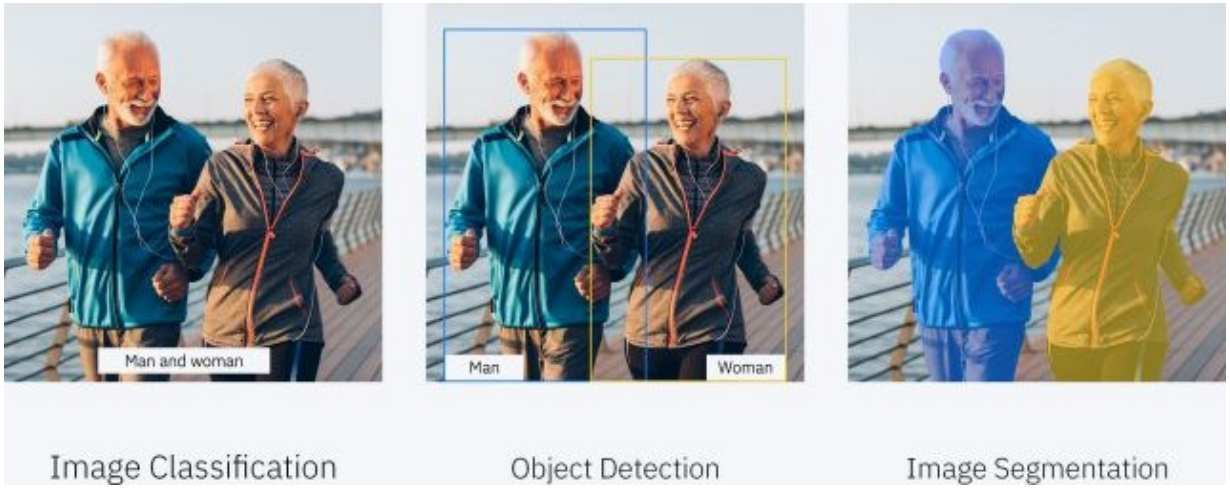


Figure V.1: Different computer vision techniques ©IBM

So our goal was to implement an object detection in our model, to find in *BathyBot* videos where are fish and directly extract parts of the image with fish in order to make classification easier.

V.3.2 Different object detection algorithms

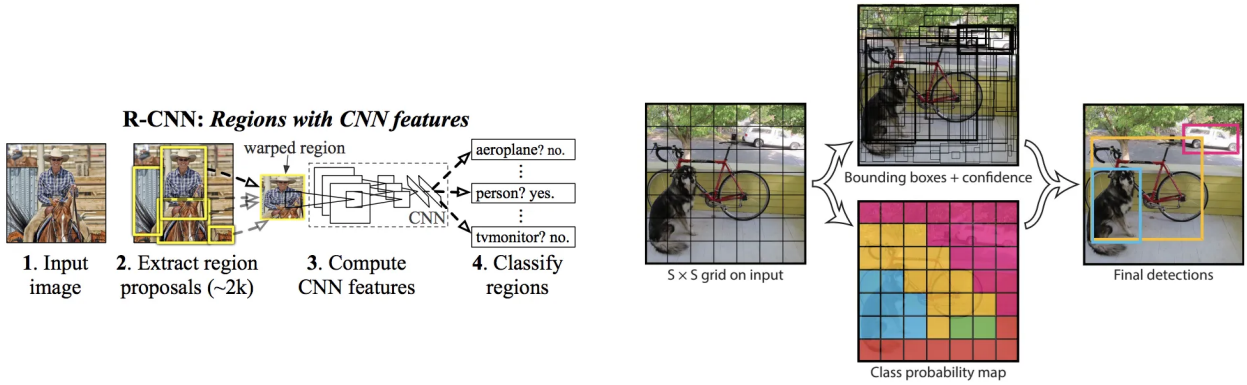
Object detection is the focus of many sectors as autonomous driving, medicine or surveillance service, so a lot of different algorithms exist today. But there is 2 of them which are really dominating computer vision:

- **R-CNN**, or *Region-based Convolutional Neural Network* ;
- **YOLO**, or *You Only Look Once*.

For **R-CNN**, the input image is sliced in interest region pictures in which there could an object to classify. After this extraction, each region is forwarded to a trained CNN to be classified. At the end, the model give, for each region, its most probable class.
It is a very cumbersome method because each region must be treated by the CNN as an input, so

real time detection is quite complicated to achieve. There are many R-CNN based method, like *Fast R-CNN*, that try to fasten it.

To use object detection in real time, a **YOLO** model is exactly what you need. Indeed, the input image is sliced in a grid and forwarded to a CNN. This CNN give, as output, for each square of the grid, its class probability. By post-treatment, it is easy to construct boxes around same class squares that are adjacent.
The difference with R-CNN is that YOLO treats the image as once, without preprocessing function, and learn to find details to determine the object's class [27]. This difference of processing is shown in the following figure:



(a) Schema of a R-CNN processing

(b) Schema of a YOLO processing

Figure V.2: Difference of algorithm between 2 object detection algorithm, R-CNN and YOLO ©R. GANDHI

For us, due to a short time to achieve our goal, we decided to follow a mix between R-CNN and YOLO methods, as explained below.

V.4 Our HCR-CNN method

We decided to create our own object detection method, based on an R-CNN algorithm but using a YOLO type construction. We didn't find a such method as described below, so we decided to name it **HCR-CNN**, the *Hydro'Cognition Region-based Convolutional Neural Network*.

V.4.1 Algorithm description

As a remainder, the aim of this algorithm is to find where a fish is located in a picture.

To do it, the algorithm will resize each input image of size $n \times m$ as an image of size $256 \times 144\,px$. After resizing the input, it is sliced in a grid of size 4×4, i.e. in 16 rectangle of size $64 \times 36\,px$

Then, each region constituted of $i \times i$ rectangle for $1 \leq i \leq 4$ is forwarded to a CNN trained to recognize fish in a picture and based on our previous study.

Finally, the network gives us the region where the probability of having a fish is the highest. By using this, our model must be able to find, in each image it will process, if there is a fish or not and where it is located.

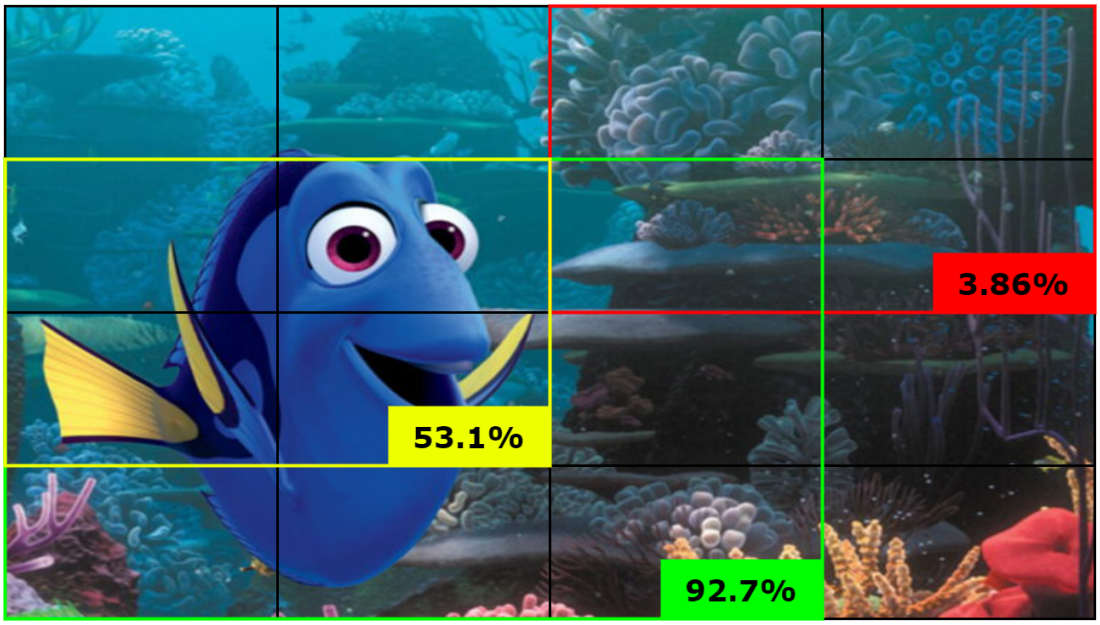


Figure V.3: Representation of how HCR-CNN could find a fish in a picture

V.4.2 Fish occurrence probability and acceptability threshold

Fish occurence

To compute the position of our fish, we create a (4×4) interest matrix M. Each term of it correspond to each 16^{th} of the image processed, these terms are called *unit*.
Each time we compute a sub-image, we add in all *units* cover by the sub-image the probability of occurrence of a fish in this sub-image. Because some sub-images can cover several *units* of the interest matrix, see Figure V.4, this probability is element-wise divided by a normalization matrix. This matrix represents the number of time we add a probability to each *units*.

Normalization matrix :

$$\begin{pmatrix} 4 & 6 & 6 & 4 \\ 6 & 10 & 10 & 6 \\ 6 & 10 & 10 & 6 \\ 4 & 6 & 6 & 4 \end{pmatrix}$$

Example of an interest matrix:

$$\begin{pmatrix} 1 & 0.99999992 & 0 & 0 \\ 0.90665629 & 0.99976473 & 0.90004299 & 0 \\ 0 & 0.90016988 & 1 & 1 \\ 0 & 0 & 0.98753373 & 0.99999371 \end{pmatrix}$$

Acceptability threshold

If a fish is detected in the full image, we compute the sub-images. The criteria for determining if a fish is within a specific area of the image is established as follows:

$$tresh = min(M) \times (1 - \theta) + max(M) \times \theta$$

Where M is the interest matrix and θ a modifiable parameter selected to 0.5.

This is a relative threshold which excludes all *units* with the lowest probability of occurrence of a fish.

V.4.3 Implementation

Always using *Python*, Figure V.4 show all sub-images to be treated by the CNN.

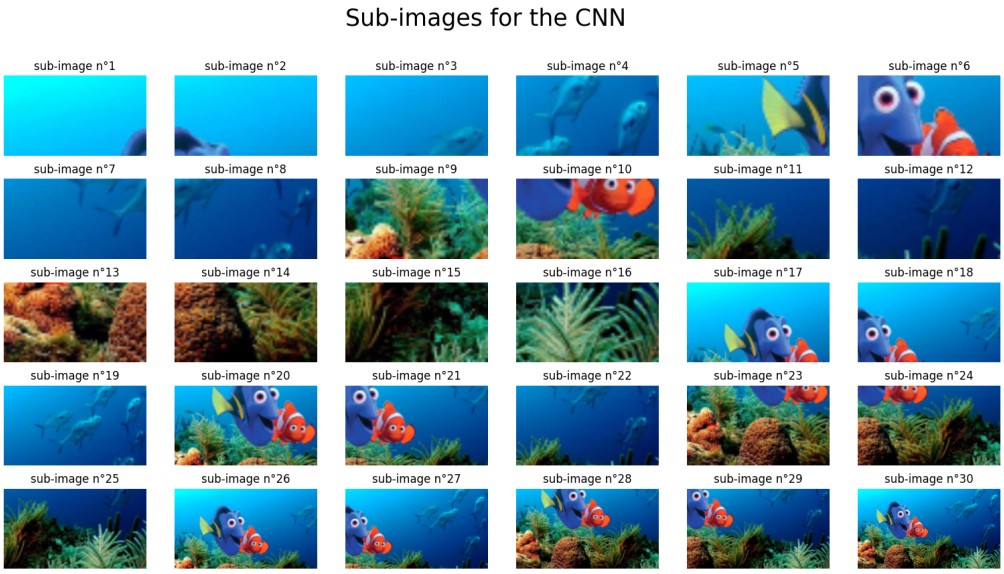
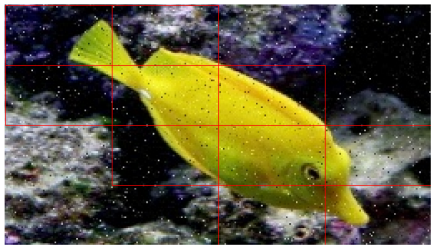


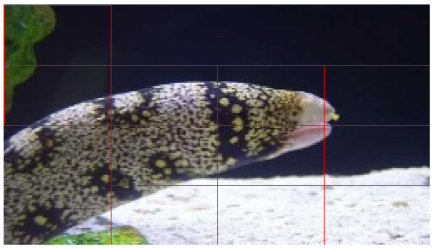
Figure V.4: Visualization of all sub-images after being sliced

As all our scripts, this one is available under the name *HCR-CNN* in the GitHub repository named HydroCognition.

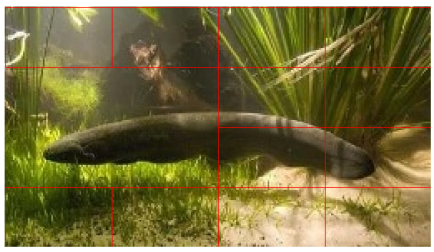
Unfortunately, we didn't have the time to develop a large scale analysis on the performance of our *HCR-CNN* model. However, we tested our model on several images. Performance is not as high as we expected, Figure V.5 show how it performs on different type of images.



(a) *HCR-CNN* - Very good case



(b) *HCR-CNN* - Average case



(c) *HCR-CNN* - Bad case

(d) *HCR-CNN* - Very bad case

Figure V.5: Processing of different images by the model *HCR-CNN*

In these cases, we can see that the model performs very well on easy images.
Nevertheless, its average performance are quite frustrating, in the Figure b, we can see that it takes the ground for a fish. Because, we trained our CNN on entire images and not sub-images, it does not perform very well.

A way to improve would be to train the CNN with sub-images and full images. Then, it may achieve a better rate for finding fish.

NOTE 6: Development in progress

We're still developing this algorithm and aiming to enhance its accuracy.

Conclusion

The main goal of our project was to propose an AI-based tool that could process videos from a deep sea ROV and extract only the frames with fish.

First, to create a tool using AI, we had to really understand what an AI is. According to deep researches, we discovered that *artificial intelligence* is nothing else than mathematical probabilistic functions connected in networks as a brain. It acts like a numerical one, and that's why the black box behind an AI is called a ***neural network***.

And as a human being, an AI must learn to be better: the back-propagation is here for that. Back-propagation is a term used in IT jargon, but it is nothing else than a gradient descent algorithm that modifies parameters of a neural network. The goal of this gradient is to find the minimum of a loss function between what a network gives as output and a desired output.
Knowing that, a well-trained AI is then a neural network with parameters that provide as output the most probabilistic case it found during processing data.

All we previously discovered about AI led us to this point: we created a fully functional neural network from scratch, but it is less performant than predefined libraries as ***TensorFlow*** with ***Keras***, the environment we used in the remaining part of our project.
Indeed, we had to process images and classify them, and that requires using a more specified AI: a Convolutional Neuron Network. This specific part of AI science focus on classifying pictures depending on what they contain. It uses graphic functions - *convolution, pooling, and many others* - and a classic neural network to find patterns in images in the same class.

After determining the best CNN architecture we must use and training it on a dedicated dataset, we developed a user-friendly application to help to process images using our AI: ***Hydro'Cognition***. This is the outcome of our project: an easy-to-use tool based on a CNN that can find fish in pictures with an accuracy of 87%.

To go further...

But to go further, we decided to overtake the initial goal of our project. Classify pictures between those without fish and those with fish is good, but it will be better to locate fish in the picture. That why we developed an object detection algorithm, the HCR-CNN, which is a Region-based CNN with inspiration from YOLO picture slicing.

This model give some good results, but there is still a problem with the dataset we used: not enough data and not enough accurate with the purpose of our project. The dataset, in the end, doesn't correspond to deep sea environment, only to fish and corals that can be found near the surface.

To enhance our model, the easiest and less time-consuming method should be to find a better dataset, 100% dedicated to underwater fish recognition in the depths.

References

[1] B.J. Copeland. Artificial intelligence. britannica.com, July 20, 1998, update: November 28, 2023. [Accessed: March 12, 2024].

[2] Larousse. Intelligence artificielle. larousse.fr, unknown. [Accessed: March 12, 2024].

[3] R. Anyoha. The history of artificial intelligence. sitn.hms.harvard.edu, August 28, 2017. [Accessed: March 12, 2024].

[4] Datascientest. Perceptron: Concept, function, and applications. datascientest.com, January 18, 2023. [Accessed: March 12, 2024].

[5] T. Keldenich. Fonction d'activation: Comment elle fonctionne? inside-machinelearning.com, February 8, 2021. [Accessed: April 8, 2024].

[6] Wikipedia. Activation function. wikipedia.com, February 5, 2014, update: January 20, 2024. [Accessed: February 15, 2024].

[7] Datascientest. Dense neural networks: Understanding their structure and function. datascientest.com, March 5, 2024. [Accessed: March 11, 2024].

[8] G. Sanderson. Neural networks lessons. 3blue1brown.com, April 14, 2017. [Accessed: February 17, 2024].

[9] CNRS/MIAI-Grenoble. Formation introduction au deep learning. FIDLE 2023/2024, November 2023 - May 2024. [Accessed: February 28, 2024].

[10] Datascientist. Epoch : An essential notion in real-time programming. datascientest.com, June 2, 2023. [Accessed: April 10, 2024].

[11] Britannica's Editor. Gradient (mathematics). britannica.com, January 25, 2024. [Accessed: February 28, 2024].

[12] R. Kwiatkowski. Gradient descent algorithm — a deep dive. towardsdatascience.com, May 22, 2021. [Accessed: February 28, 2024].

[13] K. Biswas ; S. Kumar ; S. Banerjee ; A.K. Pandey. Smooth maximum unit: Smooth activation function for deep networks. openaccess.thecvf.com, 2022. [Accessed: April 12, 2024].

[14] Martin Thoma. Analysis and optimization of convolutional neural network architectures. Masters's thesis, Karlsruhe Institute of Technology, Karlsruhe, Germany, June 2017.

[15] Qing Li, Weidong Cai, Xiaogang Wang, Yun Zhou, David Dagan Feng, and Mei Chen. Medical image classification with convolutional neural network. In 2014 13th international conference on control automation robotics & vision (ICARCV), pages 844–848. IEEE, 2014.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Communications of the ACM, 60(6):84–90, 2017.

[17] TensorFlow. Convolutional neural network (cnn). tensorflow.org. [Accessed: March 21, 2024].

[18] TensorFlow. Tensorflow 2 quickstart for experts. tensorflow.org. [Accessed: March 6, 2024].

[19] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[20] Datascientest. What is a dataset? how do i work with it? datascientest.com, September 25, 2023. [Accessed: April 13, 2024].

[21] CNRS (MIO-DT-INSU). Bathybot à 2500m en mediterranée. youtube.com, May 10, 2023. [Accessed: February 15, 2024].

[22] marineinstitueIRL. Dive deep with the rov holland 1. youtube.com, June 8, 2021. [Accessed: April 13, 2024].

[23] Paul. Tkvideoplayer. https://pypi.org/project/tkvideoplayer/, January 20, 2024. [Accessed: March 20, 2024].

[24] Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Completely automated cnn architecture design based on blocks. IEEE transactions on neural networks and learning systems, 31(4):1242–1254, 2019.

[25] J. MUREL ; E. KAVLAKOGLU. What is object detection? ibm.com, January 3, 2021. [Accessed: April 3, 2024].

[26] J. GALLAGHER. R-cnn, fast r-cnn, faster r-cnn, yolo — object detection algorithms. blog.roboflow.com, August 22, 2023. [Accessed: April 3, 2024].

[27] R. GANDHI. What is object detection? the ultimate guide. towardsdatascience.com, July 9, 2018. [Accessed: April 3, 2024].