



DEMOCRATIC AND REPUBLIC OF ALGERIA
MINISTRY OF HIGH EDUCATION AND SCIENTIFIC RESEARCH



University of Ibn Khaldoun - Tiaret
faculty of mathematics and computer science
computer science department

Dissertation

Submitted to the Department of **computer science** as a Partial Fulfillment
of the Requirement for the

Master Degree in computer engineering

Type: academic

Theme:

***Automatic recognition of noisy
digital images using Deep learning***

Presented by: Abdeldjallil AIDI

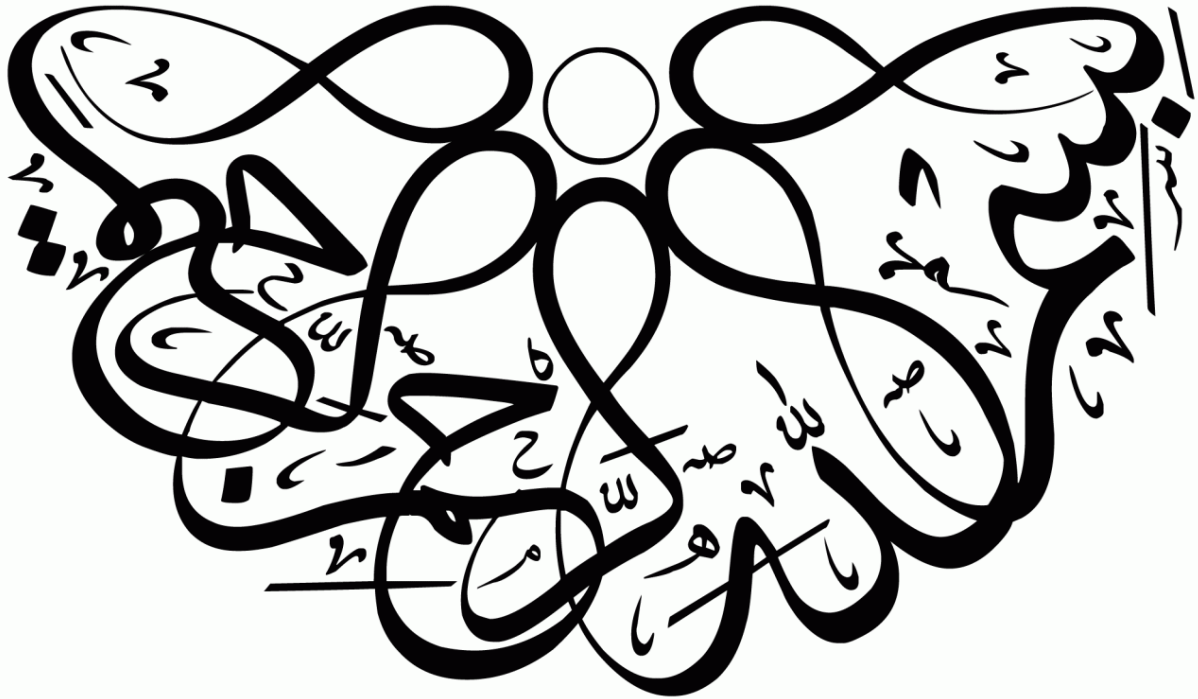
Supervised by: Mr. Karim MEZZOUG – MAA University of Ibn-Khaldoun-Tiaret

Board of Examiners:

Mr. Mohamed BAGHDADI – MAA University of Ibn-Khaldoun-Tiaret.

Mm. Abdia HAMDANI – MAA University of Ibn-Khaldoun-Tiaret.

Academic Year: 2019-2020



“The world that we have made as a result of the level of thinking we have done thus far creates problems that we cannot solve at the same level as the level we created them at.”

Albert Einstein¹

“ I am enough of the artist to draw freely upon my imagination. Imagination is more important than knowledge. Knowledge is limited. Imagination encircles the world.”

Albert Einstein²

1. The Journal of Transpersonal Psychology
Transpersonal Institute, 1969, 1-4, pp 124

2. The Saturday Evening Post, What Life Means to
Einstein: An Interview by George Sylvester Viereck, 1929
October 26

Acknowledgments:

First and foremost, I would like to thank **Mr. Karim MEZZOUG**, my supervisor. Who listened to me, accompanied me, guided me during all these stages, and understood my expectations, my different outlook on the studies that we have carried out. Thank you for all these constructive, rigorous discussions Thank you for being patient, understanding and structured. Thank you for helping me grows academically and professionally, trusting me and making my visits to the university not only rewarding but also enjoyable.

To **Mr. Mohamed BAGHDADI**: Thank you for doing us the honor of chairing this jury. I also thank you for continuing to fight for the good of your students. Thank you also for your insightful comments during the various laboratory presentations. You are an inspiration for future men in engineering. I express my gratitude to you.

My sincere thanks to **Mm. Abdia HAMDANI** for the honor, you have done to me agreeing to participate in my thesis jury as an examiner of my work, thank you for the time you spent reading this thesis, and for the suggestions and judicious remarks you indicated to me. This dissertation would not have come to fruition without your kindness and guidance.

Of course, I will not be without mentioning the two important persons in all my life which are my parents **Abdeldkader** and **Hafidha** and all the members of my family, who have always encouraged me in my projects and offered the means to carry them out. I also thank them for their hospitality when I needed to isolate myself to write this manuscript.

Finally, I would like to pay tribute to the colleagues from IBN KHALDOUN University. To those who have passed very quickly but have changed us a bit, to those who are there every day, to **Ilias Sid Ahmed MAKBOUL, Hakim ADIM, Mohamed MAHROUZ, Abdelmalek DAHI** and more others to always take the time to discuss our projects and make suggestions. To the people who mean the most to me. They will be recognized.

Abstract:

In the field of computer vision, image classification is one of the main problems that hold the lion's share of research. Since in most real-world scenarios within image classification applications there is no control over how the qualities of the images are given. Remarkably, it is crucial to consider that these images might be damaged by noise intentionally or unintentionally.

In this thesis, we try to clarify the effects of noise contained in the images in any way possible within image classification tasks by analyzing two different types of noise (S&P, Gaussian) with five different levels on three CNN models (XceptionNet, GoogleNet, ResNet) using the same parameters (Dataset, noise, and level of noise), and how denoising methods can help to alleviate this problem.

We perform our experiments with the Cifar10 dataset and two different denoising methods (one for each type of noise), our results show that noise in images can hinder classification tasks and cause it a problem (make it harder to separate classes). Although images were denoised, we were unable to reach the results obtained in the noise-free scenarios.

Keywords: Computer vision, Image Classification, Noise, Denoising, CNN.

3.4 - multi-resolution processing	21
3.5 - Image compression	22
4 - Computer vision	23
4.1 - segmentation, reconstruction, recognition	23
4.2-Applications of computer vision	25
5 - Conclusion	26
CH02: Image Classification	27
Introduction	27
2 - The motivations for image classification	28
3 - Classification /clustering	29
3.1 - Clustering	29
3.2 - Classification	29
3.2.1 - Binary classification	29
3.2.2 - Multi-class classification	30
4 - Machine learning for image classification	30
4.1 - The general process of machine learning Classification and its ingredients	31
4.2 - Some types of Machine learning problems	31
4.2.1 - Supervised learning	32
4.2.1.1 - Some supervised learning algorithms	33
A - K nearest neighbors (KNN)	33
B - Support Vector Machine (SVM)	33
C - Decision tree	34
4.2.2 - Unsupervised learning	34
4.2.2.1 - Some unsupervised learning algorithms	34
A- Hierarchical ascending classification	35
B - K-Means	35
C - Fuzzy C-means	35
4.2.3 - Semi-supervised learning	36
4.2.4 - Reinforcement learning	36
5 - Conclusion	36
CH03: Deep learning	37
Introduction	37
2 - Machine Learning	37
3 -Artificial Neural networks	38
3.1 - The biological neuron	39
3.2 - The perceptron model	40
3.2.1 - Limitations of the perceptron	45
3.3 - The multi-layered perceptron	45
3.3.1 - The Activation function	47
3.3.1.1 - Binary Step Function	48
3.3.1.2 - Linear Activation Functions	48
3.3.1.3 - Non-Linear Activation Functions	49
3.4 - Learning with back propagation	52
3.4.1 - Gradient descent	52
3.4.2 - Back propagation method	53

3.5 - Convergence of learning	55
3.5.1 - Problem of over-fitting	55
3.5.2 - Set of validation and cross-validation	56
3.5.3 - Early-stopping	57
3.5.4 - Regularization	57
3.6 - Alternatives to gradient descent	57
3.6.1 - Stochastic Gradient Descent	57
3.6.2 - Momentum	58
3.6.3 - Nesterov momentum	58
3.6.4 - Second-order methods	59
3.6.5 - Other optimization techniques	59
4 - Deep Artificial neural networks	59
4.1 - The interest of deep architectures	60
4.2 - Convolutional neural networks	61
4.2.1 - Convolutional layers	62
4.2.2 - Pooling Operation	63
4.3 - Recurrent neural networks	64
4.4 - Advanced techniques to improve learning	64
4.4.1 - Inception module (GoogleNet)	65
4.4.2 - Batch normalization	65
4.4.3 - ResNet	66
4.5 - Unsupervised neural networks	66
4.5.1 - Auto-encoders	66
4.5.2 - Generative antagonist networks	68
5 - Conclusion	70
CH04: Experiments and Desktop Application	71
Introduction	71
1 - Tools and Libraries	71
2.1 - Python	71
2.2 - Google Colaboratory	71
2.3 - Anaconda navigator	72
2.4 - Jupyter notebook	72
2.5 - TensorFlow	73
2.6 - Keras	73
2.7 - OpenCV	73
2.8 - PyQt	73
2.9 - Cifar-10 Dataset	74
3 - Experiments	74
4 - Implementation	76
5 - Results and Discussion	77
6 - Conclusion	81
Bibliographic references	81

List of figures:

Figure N°01: The succession from image processing to computer vision	04
Figure N°02: Image capture. A scene is illuminated by energy from a light source. Reflected light may fall on the detector and be captured.	06
Figure N°03: median filter example using a 3*3 sampling window	10
Figure N°04: Result of median filtering.	10
Figure N°05: (a) A representation of a narrow image histogram. (b) A representation of a widely distributed image histogram.	11
Figure N°06: Histogram equalization results.	13
Figure N°07: Two-dimensional filter response (a) high-pass, (b) low-pass	14
Figure N°08: Homomorphic filtering: (a) input image, (b) result of Homomorphic filtering	14
Figure N°09: noise classes.	15
Figure N°10: Plot of Probability Distribution Function. Where g = gray value, σ = standard deviation and μ = mean.	17
Figure N°11: Mean filter results.	18
Figure N°12: Gaussian filter results.	19
Figure N°13: Computer vision at the intersection of multiple scientific fields.	23
Figure N°14: Some types of Machine Learning problems	32
Figure N°15: Supervised learning	33
Figure N°16: Unsupervised learning	34
Figure N°17: Learning model seen as a black box	38
Figure N°18: Illustration of a biological neuron (up) and its mathematical model (down).	40
Figure N°19: Graphical representation of a perceptron described in the Email Classification Example.	41
Figure N°20: Example of gradient descent on one dimension	43
Figure N°21: Example of logic functions	45
Figure N°22: Example of a representation of an MLP	45
Figure N°23: Neural network representing the EXCLUSIVE OR function with two graphical representations	47
Figure N°24: The basic process carried out by a neuron in a neural network	47

Figure N°25: The function of activating binary step	48
Figure N°26: Linear Activation Function	48
Figure N°27: The output of a sigmoid neuron as t varies	49
Figure N°28: The output of a Tanh neuron as t varies	50
Figure N°29: The output of a ReLU neuron as z varies.	51
Figure N°30: Intermediate representation of the entry into layer 1	52
Figure N°31: Three models of classifiers at different learning levels	56
Figure N°32: Loss functions	56
Figure N°33: The difference between classic machine learning (up) and deep learning (down)	60
Figure N°34: Example from CNN called AlexNet.	61
Figure N°35: example of a convolution operation using sobel Filter	62
Figure N°36: Example of a pooling operation	63
Figure N°37: RNN layer	64
Figure N°38: Inception module in its simple version	65
Figure N°39: Example of a classic ResBlock with two intermediate layers	66
Figure N°40: Example of an auto-encoder	67
Figure N°41: Example of noise reduction with auto-encoder	67
Figure N°42: Architecture of GAN with generator and discriminator	68
Figure N°43: A mostly complete chart of artificial neural networks	69
Figure N°44: Google Colaboratory	71
Figure N°45: Anaconda navigator	72
Figure N°46: Jupyter notebook	72
Figure N°47: Represent the classes in the dataset, as well as 10 random images from each.	74
Figure N°48: XceptionNet saved History.	75
Figure N°49: GoogleNet saved History.	76
Figure N°50: ResNet saved History.	76
Figure N°51: graphical user interface.	77

Figure N°52: Comparison between accuracies of each model for different noise levels	78
Figure N°53: Result of NLM filtering	79
Figure N°54: accuracy and loss plots for each model when using Salt&Pepper noise with $p = 0.3$.	79
Figure N°55: accuracy and loss plots for each model when using Gaussian noise with $\sigma = 30$.	80

List of tables:

Table N°01: Differences between Classification and Clustering.	30
Table N°02: PSNR and SSIM for each noise level	77
Table N°03: Accuracy of each model when training and testing using the same dataset version.	78

Glossary of Acronyms and Abbreviations:

- ✓ **PDF** – Probability Density Function.
- ✓ **EM** – Expectation Maximization.
- ✓ **PCA** – principle component analysis.
- ✓ **LBP** – local Binary Patterns
- ✓ **HOG** – Histograms of Oriented Gradient
- ✓ **SM** – standard median
- ✓ **CT** – computed tomography
- ✓ **FBP** – filtered back projection
- ✓ **PET** – positron emission tomography
- ✓ **IR** – iterative reconstruction
- ✓ **AR** – Augmented Reality
- ✓ **VR** – Virtual Reality
- ✓ **KNN** – K nearest neighbors
- ✓ **ILSVRC** – Large Scale Visual Recognition Challenge
- ✓ **HE** – Histogram Equalization
- ✓ **BBHE** – Brightness Bi-Histogram Equalization
- ✓ **DSIHE** – Dualistic Sub Image Histogram Equalization
- ✓ **MHE** – Multi Histogram Equalization
- ✓ **MMBEBHE** – Minimum Mean Brightness Error Bi-Histogram Equalization
- ✓ **GHE** – Global Transformation Histogram Equalization
- ✓ **LHE** – Local Transformation Histogram Equalization.
- ✓ **IDBPHE** – Recursive Mean Separate Histogram Equalization
- ✓ **GPU** – Graphical processing unite.
- ✓ **SVM** – Support Vector Machine
- ✓ **AI** – Artificial Intelligence
- ✓ **ML** – Machine learning
- ✓ **DL** – Deep learning
- ✓ **MLP** – Multi-layer perceptrons
- ✓ **GD** – Gradient Descent
- ✓ **SGD** – Stochastic Gradient Descent
- ✓ **GAN** – Generative Antagonist Network
- ✓ **CNN** – Convolutional Neural Network
- ✓ **ANN** – Artificial Neural Network
- ✓ **RNN** – Recurrent Neural Network

General initiation:

As an indication, note that nearly 90% of the information received by humans is visual. The production of quality images, as well as their digital (and if possible) automatic processing, is therefore of considerable importance. Most scientific devices provide images (microscopes, telescopes, x-rays, Magneto-nuclear resonance ... etc.), and many fields of application use the image as a source of information and/or visualization.

In fact, the image is a collection of information which, at first, was presented on a photographic medium which allowed the delayed processing of the fleeting phenomenon, a detailed analysis of the recorded phenomena, and of course the archiving and the illustration. Digital Image processing was born from the idea of the need to replace the human observer with the machine. The image or signals from the sensors were then digitized for processing by the computer.

Man, always resorts to classification in his life, is he trying to answer the problems and questions about the categories of objects? That is to say carrying out the assignment of objects to their classes (formats, colors, sizes... etc.). Generally, observation bases characterize a particular domain (animals, fruit, patients, genes... etc.), where they are grouped into several classes. Automatic Image Classification is a pattern recognition application that automatically assigns a class to an image using a classification system.

Deep learning is a learning technique that enables a program, for example, to recognize the content of an image or to understand spoken language. Deep learning technology teaches you to represent the world. That is to say how the machine will represent the word or the image for example. This learning and classification system, based on digital "artificial neural networks" is a common technique in AI, allowing machines to learn and recognize objects. He deepens his understanding of the image with more and more precise concepts.

Our work is a part of the classification of images using deep learning. We aim to create a certain number of classifiers to images that contains noise and their restored versions, the results will be compared with each other to see the resilience of CNNs over distinct type of noise with different levels.

1 – Problematic:

In our humble thesis we try to answer an easy question, which is:

“Does noise in both training and test sets affect the performance of deep learning neural networks? “

2 – Related works

This is not the first time such experimentation is done to understand the impact of noisy images in the performance of CNNs within image classification tasks. There are papers studying the effects of noise in the capability of CNNs to learn [7].

Noise in images can hinder classification tasks; this knowledge is already discovered with systems that employ ANNs, and in systems that use hand-crafted features [18]. State of the art deep neural networks performance is affected when classifying images with lower quality compared with the original training set is showed by Dodge and Karam in [18], their experiments do not cover the presence of noisy images in the training set and how the model is doing with it.

While Paranhos da Costa et al in [37] takes in consideration that images in the training set might be affected by noise in somehow, they created noisy versions and generated their restored versions, they used hand-crafted features (LBP and HOG) and SVM classifiers where trained with each version of the training set.

T. Nazare et al in [20], their work is based on [37] they exploited deep neural networks.

As all we believe that noise in any way possible, makes classification more difficult, our experiments is based on [20], however there are two main differences. First, we exploited state of the art deeper artificial neural network architectures such as RESNET and XCEPTIONET, second, we train all the architecture using the same dataset version for better comparison.

3 – Plans:

In chapter N°01, we have introduced the basics that serve as a foundation for understanding different digital image processing techniques. Several classic processing methods have been proposed in the literature, we have presented some that we believe are the most common in the process of image processing and analysis.

In chapter N°02, the classification methods are quite numerous so that each method has certain advantages over the others for certain characteristics of the images. In this chapter we present an overview of image classification and several methods used in it.

In chapter N°03, Deep learning is one of the methods of machine learning. In this chapter we will explain what machine learning is and what it can do. We then present the model of the neural network which is the basis of deep learning. We will then show the different deep architectures of artificial neural networks and their usefulness. The purpose of this chapter is to get an overview of the value of deep learning, but also of its constraints.

Chapter N°04 is devoted to describe the tools used in our experiments, and a brief discussion over the results.

Introduction:

Vision, the most important of the five senses human have, thanks to ALLAH for an **eye** with more than six million cone cells, each one of them has one of three color-sensitive proteins known as opsins, these last ones change shape when hit by photons of light, triggering a cascade that produces electrical signals, which in turn transmit the messages to the brain for interpretation. Hence, a man can see a whole scene around fully and clearly. The idea of making this process possible for machines creates the field of computer vision.

Image digitization with data compression was performed for the first time in 1920 by British inventors Harry G. Bartholomew and Maynard D. McFarlane with the intention to send faxes from London to New York. Digital image processing focuses on two major tasks that are improvement of pictorial information for human interpolation and processing of image data for storage, transmission and representation for autonomous machine perceptions [01] to extract useful information with the help of computer vision algorithms.

The provenance of Computer vision goes back to an MIT undergraduate summer project in 1966¹. It can be defined as *a scientific field that extracts information out of digital images*, and through its applications, it can be defined as *building algorithms that can gain high-level understanding from the content of digital images or videos* and use it for other applications. Digital Images are an important type of digital media, and an essential tool for many scientists. As in astrophysics data from both satellites and distant stars and galaxies is collected in the form of images, and Medical imaging makes it possible to gather different kinds of information in the form of images, even from the inside of the body. By analyzing these images, it is possible to discover tumors and other disorders. For this to be achieved computer vision rely on image processing advanced techniques such as image enhancement, restoration and compression.

¹ Seymour A Papert. The summer vision project. 1966

2-Image processing VS computer vision:

In many contexts the two terms “Computer vision” and “image processing” are used almost as they pour into the same concept. They both involve practicing some computations on images. But are they really the same?

In **Image processing**, an image is given as input to be "processed", and an output image is returned after applying the transformations according to the context and the goals to be achieved. The transformations can be “smoothing”, “sharpening”, “contrasting” and “stretching”. Unlike **Computer vision**, it takes an image or a video as input, and the goal is to gain a high-level understanding about the image and its contents including being able to infer something about it. Computer vision uses image processing algorithms to solve some of its tasks.

The succession from image processing to computer vision can be classified into three levels low, mid and high-level processes [03] (see **Figure N°01**). In Low-level process both inputs and outputs are defined as images, such processes concern the image pre-processing operations such as contrast enhancement, noise reduction, and image sharpening. Mid-level processes concern operations such as segmentation, representation & description, and classification of objects. In the last step, High-level processes are carried out to make the sense, understanding, and autonomous navigation of individual objects for vision. Thus, Low-level to Mid-level processes are devoted to image processing while from Mid-level to High-level processes are devoted to computer vision.

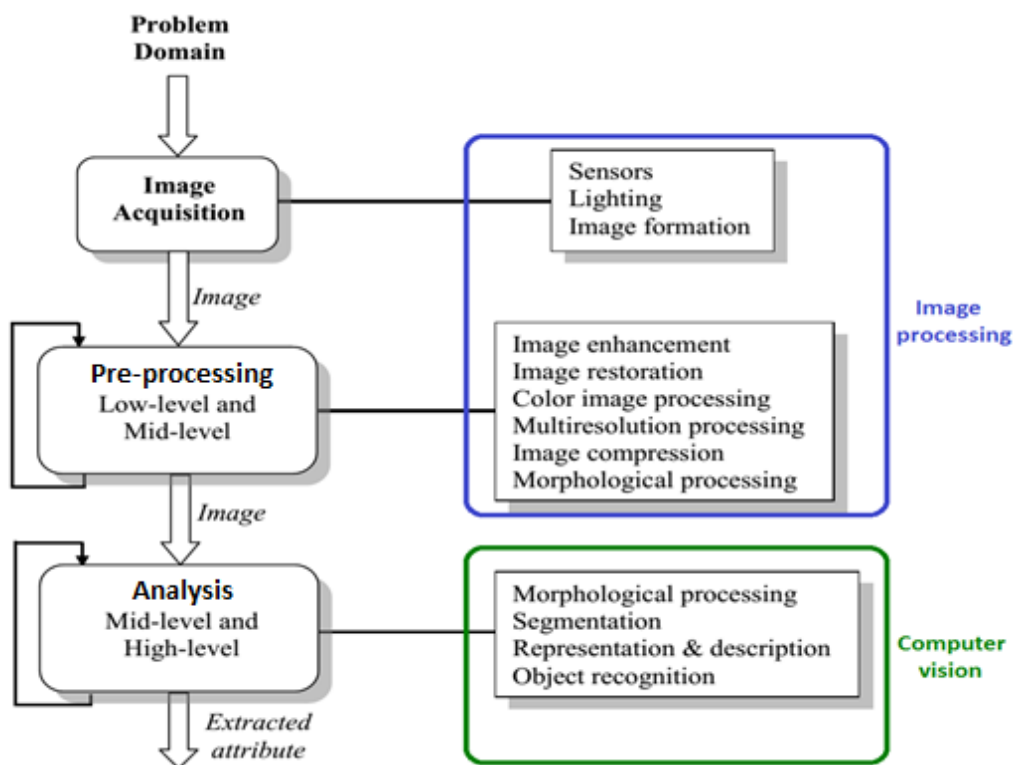


Figure N°01: The succession from image processing to computer vision.

3-Image processing:

Modern technology has made it possible to manipulate multidimensional signals with systems that range from simple digital circuits to advanced parallel computers [03]. The discipline of image processing is a vast one, encompassing signal processing techniques as well as techniques that are specific to images. Its applications range from medicine to entertainment, passing by geological processing and remote sensing. An image can be regarded as a function $f(x,y)$ of two variables x and y as in electrical engineering and computer science, it is a two-dimensional signal.

3.1 – Digital image:

In the real world an image can be a photograph, a painting, or even a dream, but in the world of computers, it is a set of dots called “pixels”. Such image is commonly known as digital image and formally defined as an array of pixels whose values specify the light intensity of the flux on the picture element represented by that pixel. As it said before the digital image is a function of two variables x and y which are responsible for the distribution (positions²) of the image pixels. Positions and value are positive scalars whose range depends on the digitization unit characteristics. The value of each pixel could be between 0 and 255³. Three different kinds of digital images could be illustrated:

- Black and white image,
- Gray value image, and
- Color image.

3.2 – image acquisition:

Before any procedure of image processing can began an image has to be captured and converted to digital form. This process is named image acquisition; its purpose is to transform a view of the real world into a digital image. However, a good understanding of the image formation process is required in quantitative analysis of any images requires. For an object in the three-dimensional world to become a digital image in the memory of a computer it must pass through three necessary steps:

- ***Becoming visible:*** by interacting with the light or more generally an electromagnetic radiation an object can be visible. The light collected by a camera system is determined by the optical properties of the material from which the object is made as well as by the illumination⁴ (see **Figure N°02**).
- ***Projection:*** The optical system collects the light rays reflected from the objects and projects the three-dimensional world onto a two-dimensional image plane.

² The position of the pixel is specified by the function $f(x, y)$ where x is the width, and y is the height.

³ 0 representing the darkest intensity, and 255 the lightest intensity.

⁴ Position and nature of the light or, more generally, radiation sources.

- **Digitization:** The continuous image on the image plane must be converted into image points on a discrete grid. Furthermore, the intensity at each point must be represented by a suitable finite number of gray values (Quantization) [04].

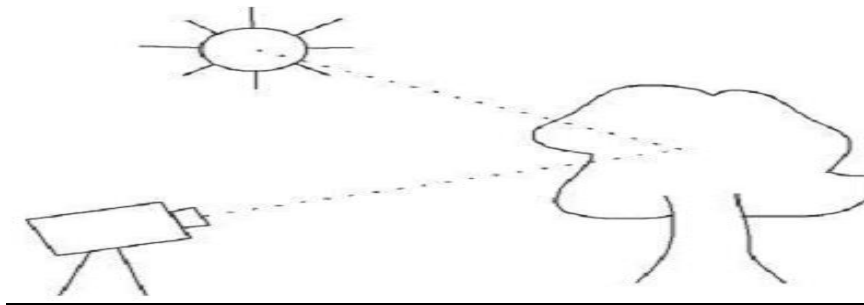


Figure N°02: Image capture. A scene is illuminated by energy from a light source. Reflected light may fall on the detector and be captured.

3.3 – image enhancement and restoration:

In the internet, in our phones and laptops, Millions of pictures ranging from biomedical images to the images of natural surroundings, such pictures might contain a lot of important information in diverse domains of application. The quality of the output image may be inferior to that of the original input picture when converted from one form to another by processes like imaging, scanning, or transmitting. Hence, there is a need to improve the quality of such images, so that the output image is better for human perception or machine analysis.

The realm of image enhancement covers contrast and edge enhancement, noise filtering, feature sharpening, and so on. These methods find applications in visual information display, feature extraction, object recognition, and so on [05] [06]. It mainly improves the visual appearance of the image or to make the original image more suitable for human or computer processing [05].

3.3.1 – Distinction between image enhancement and image restoration:

Sometimes we receive blurred or noisy images which are degraded by some degrading mechanism. A blur might be caused by defocused camera⁵, atmospheric degradation⁶, or a relative accelerated motion between the object and the focal plane of the lens of the camera during the capturing of a scene. In such cases the conventional enhancement techniques would not be suitable to clarify the object within the image but the restoration or reconstruction of the original scene might work very well if we can mathematically model the cause of degradation.

⁵ The camera is not appropriately focused in the moment the picture was taken.

⁶ An outdoor scene captured on a foggy winter morning.

It is needless to say, the reconstruction or restoration techniques are different from the enhancement techniques which are employed not necessarily to recover the original object from the scene but essentially to get a better-quality picture. Image restoration techniques seek to recover an image that has been degraded by a degradation phenomenon while the ultimate goal of image enhancement techniques is to improve the quality of the image.

3.3.2 - SPATIAL IMAGE ENHANCEMENT TECHNIQUES:

The spatial filtering techniques used for noise reduction (or smoothing) are as follows:

- Spatial low-pass, high-pass and Band-Pass filtering
- Unsharp masking and crisping
- Directional smoothing
- Median filtering

A- Spatial low-pass and high-pass Filtering:

In signal processing theory, Low-pass filtering attenuates the high-frequency components in the signal and is essentially equivalent to integrating the signal. Such integration implies summation and averaging the signal. Low-pass filter is also called blurring or smoothing filter, it is a spatial averaging operation. The simplest low-pass filter just calculates the average of a pixel and all of its eight immediate neighbors so the original value of the pixel will be replaced by the resulting value. In particular, this operation is useful in removing visual noise, which generally appears as sharp bright points in the image.

On the other hand, High-pass filtering of an image produces an output image in which the low spatial frequency components are attenuated; it is mostly used for edge enhancement to emphasize fine details in the image. Since the sharpness of an image is related to the content of high-frequency components, high -pass filtering is used to deblurring, while low -pass filtering leads to blurring.

Such a filter can easily be implemented by subtracting the low-pass output from its input. Typically, the low-pass filter would perform a relatively long- term spatial average, $((2q + 1) \times (2q + 1))$ with $q \in Z^+$ window [08].

B- Averaging and Spatial Low-Pass Filtering:

If the resulting image is a low pass filtered image, which means each pixel is replaced by a weighted average of its neighborhood pixels, the output image in this case is expressed as:

$$g(m, n) = \sum_{-q \leq k, i \leq q} a(k, i) f(m - k, n - i) \quad (1.1)$$

Where $f(m, n)$ and $g(m, n)$ are the input and output images respectively, W is a suitably chosen neighborhood around the pixel at location (m, n) , $f(m - k, n - i) \in W$, and $a(m - k, n - i)$ are the filter weights. In general, in spatial averaging filters, all the weights are assigned equal values. Hence the mathematical representation of the filtering becomes:

$$g(m, n) = \frac{1}{N} \sum_{-q \leq k, i \leq q} f(m - k, n - i) \quad (1.2)$$

Where N is the number of pixels in the neighborhood W .

The spatial averaging operation on an image may be used to smooth the noise. If the observed image is given as:

$$d(m, n) = f(m, n) + \mu(m, n) \quad (1.4)$$

Then the spatial average yields:

$$g(m, n) = \frac{1}{N} \sum_{-q \leq k, i \leq q} f(m - k, n - i) + \bar{\mu}(m, n) \quad (1.5)$$

Where $\bar{\mu}(m, n)$ is the spatial average of the noise component $\mu(m, n)$. If the noise has a variance σ^2 , then it can be shown that $\bar{\mu}(m, n)$ is zero mean and has variance $\frac{\sigma^2}{N}$. This implies that the image noise power is reduced by a factor equal to the number of pixels chosen in the neighborhood of the central pixel by performing the spatial averaging filtering.

The conventional spatial filtering utilizes an averaging procedure to generate the smoothed image. The weights used to average are image data invariant. Thus, all regions of the image which can be brought under an arbitrary neighborhood W are equally affected. In such a way, spatial filtering by averaging

(I) Does not take into account the effect of the difference of gray levels between the central pixel and a neighboring pixel.

(II) Does not always take into account the diminishing influence of the pixels that are situated in increasing distance from the central pixel [08].

C- Unsharp Masking and Crisping:

As we have observed from the above, a sharp image can be obtained by high-pass filtering a blurred image. Alternatively, subtracting a blurred version of the image from the original image may also lead to the sharpening of the image [08]. Here

$$v(m, n) = f(m, n) - g(m, n) \quad (1.6)$$

Where $g(m, n)$ is a low-pass-filtered version of the original image $f(m, n)$.

An Unsharp mask is simply another type of high-pass filter. Such mask is constructed by Low-pass filtering an image. Usually the mask is scaled before it is subtracted to make ease control in the amount of sharpening that is applied. Also, the strength of the low-pass filter used can also be adjusted⁷. From an alternative viewpoint, to result in a better high-contrast image a gradient or a high-pass signal may be added to the original image. The Unsharp masking operation can be represented by:

$$v(m,n) = f(m,n) + \gamma h(m,n) \quad (1.7)$$

where $\gamma > 0$ and $h(m,n)$ is a suitably defined gradient at (m,n) .

This is also referred to as high emphasis filter where the Low frequency components are retained while emphasizing the High frequency components of the image.

D- Directional Smoothing:

A blurred image is always the result when using a Low-pass filter and quite often the crisp edges are blurred by averaging. To minimize this effect, the directional averaging operation is often used to inhibit the edges from getting blur resulting from the smoothing operation. Spatial averages $g(m,n;\theta)$ are calculated in several directions θ as:

$$g(m,n;\theta) = -f(m-k, n-i) \quad (1.8)$$

Where $f(m-k, n-i) \in W_0$, and W_0 is the neighborhood selected in the direction θ . The key to the implementation of effective directional smoothing is to identify a specific direction θ^* for which $|f(m,n) - g(m,n;\theta^*)|$ is minimum.

E- Median Filter:

The standard median (SM) filter is a simple nonlinear smoother; It is particularly effective in reducing impulsive-type noise [21]. In median filtering the input pixel is replaced by the median of the pixels contained in the neighborhood [22]. Symbolically this can be represented as:

$$v(m,n) = \text{median}\{y(m-k, n-i) \mid y(k,i) \in W, -q \leq k, i \leq q\} \quad (1.9)$$

Where W is suitably chosen neighborhood. The algorithm for median filtering requires arranging the pixel gray values in the neighborhood in increasing or decreasing order and picking up the value at the center of the array (see **Figure N°03**). Generally, the size of the neighborhood is chosen as odd number so that a well-defined center value exists. If, however, the size of the neighborhood is even the median is taken as the arithmetic mean of the two values at the center.

⁷ A very strongly blurred mask can be used to remove large-scale brightness differences, while a slightly blurred mask will sharpen fine detail.

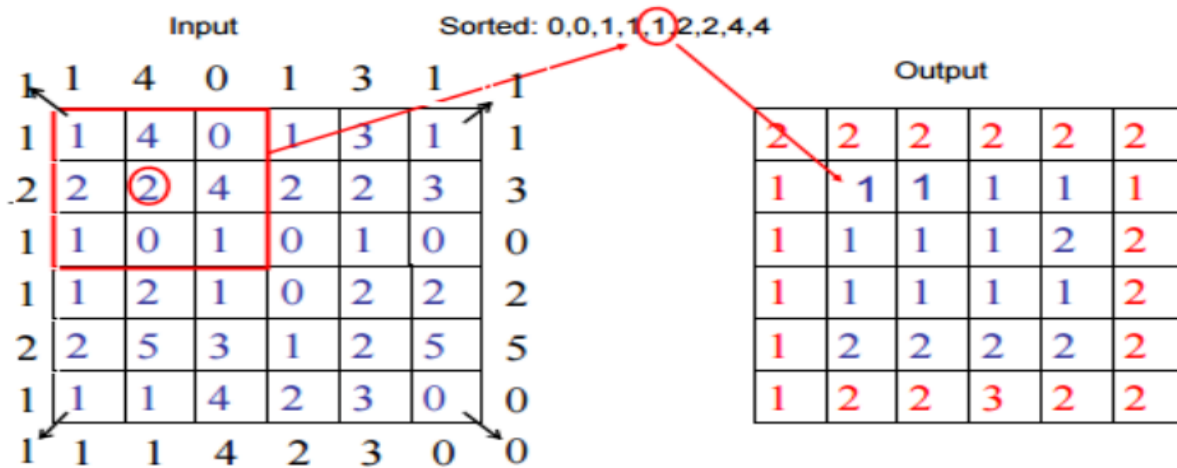


Figure N°03: median filter example using a 3*3 sampling window.

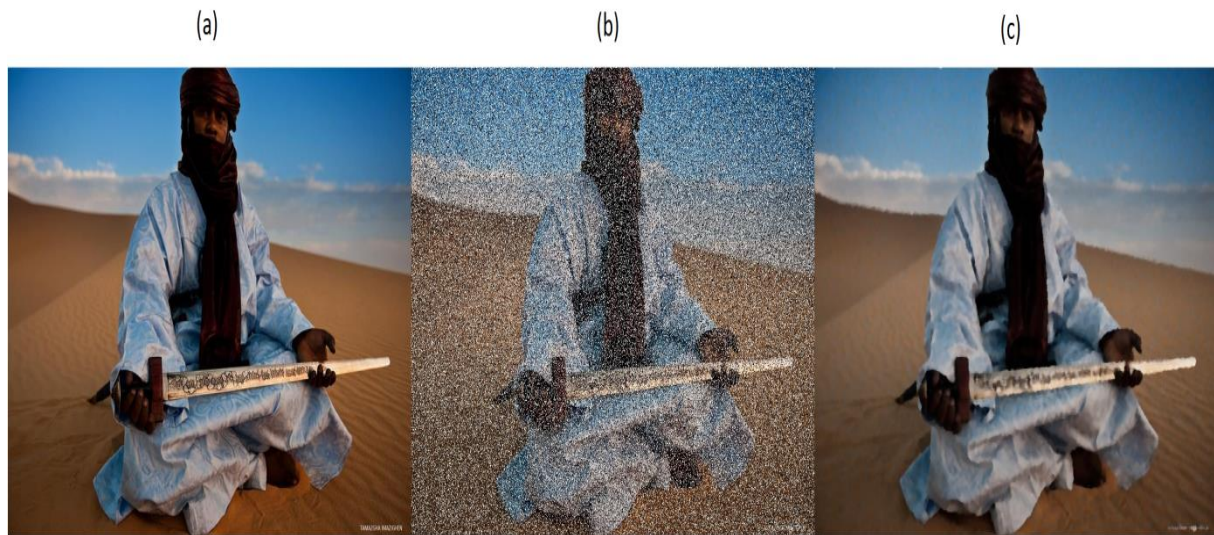


Figure 04: Result of median filtering: (a) original image, (b) salt and pepper noisy image, (c) result of median filtering.

Some of the properties of median filter are:

- It is a nonlinear filter.
- It is useful in removing isolated lines or pixels while preserving spatial resolution. It is found that median filter works well on binary noise but not so well when the noise is Gaussian.
- Its performance is poor when the number of noise pixels is greater than or equal to half the number of pixels in the neighborhood.

3.3.3 - HISTOGRAM-BASED CONTRAST ENHANCEMENT:

Images with low contrast are usually captured in dark or bright environments. In such images a large number of pixels occupy only a small portion of the available range of intensities. So, preprocessing of such images becomes necessary to make the images suitable for other image processing applications. Through histogram modification, the dynamic range of gray levels can be increased by reassigning each pixel with a new intensity value. The principle here is to stretch the dynamic range of the pixel values in a suitable way.

The information conveyed by an image is related to the probability of occurrence of gray levels in the form of histogram in the image [05]. By uniformly distributing the probability of occurrence of gray levels in the image, the perception of the information content in the image becomes much easier.

A- Image Histogram:

The histogram of an image is a discrete function. It represents the relative frequency of occurrence of the various gray levels in the image. From a mathematically point of view, for a digital image with gray level in the range $[0, L - 1]$, discrete function of the histogram is as follow:

$$p(r_k) = \frac{n_k}{N}$$

Where r_k is the k^{th} gray level and n_k is the number of pixels in the image with that gray level. N is the total number of pixels in the image. It may be noted that $k = 0, 1, \dots, L - 1$.

An image is poorly visible; the most likely reason is that the image histogram is narrow (see **figure N°05 (a)**) which means pixels that represent different objects or different parts of an object have a very close⁸ gray level. Similarly, a widely distributed histogram (see **figure N°05 (b)**) means that almost all the gray levels are present in the image which leads to a clearer image.

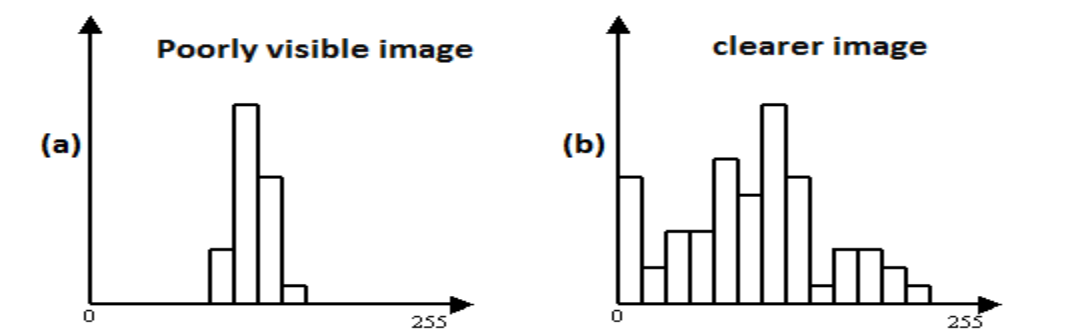


Figure N°05: (a) A representation of a narrow image histogram.
 (b) A representation of a widely distributed image histogram.

⁸ The difference in gray levels present in the image is generally low.

B- Histogram Equalization (HE):

Histogram equalization [09] is widely used technique for contrast enhancement in a variety of applications due to its simple function and effectiveness. It focuses on adjusting the gray scale of the image so that the gray level histogram of the input image is mapped onto a uniform histogram [01] [10]. The goal of **HE** is to obtain a uniform histogram for the output image.

Let the variable r represents a random variable which indicates the gray level of an image. First, assuming that r is continuous and lies within the closed interval $[0,1]$ with $r = 0$ representing black and $r = 1$ representing white. For any r in the specified interval let us consider a transformation of the form: $s = T(r)$.⁹

It is assumed that the transformation T satisfies the following criteria:

- $T(r)$ is a single valued function, monotonically increasing in the interval
- $T(r)$ lies between 0 and 1

The first condition preserves the order from black to white in the gray scale, and the second one guarantees that the function is consistent with the allowed range of pixel gray values. The inverse transform from s to r can be represented by:

$$r = T^{-1}(s). \quad (1.10)$$

Let the original and transformed gray levels be characterized by their probability density functions $p_r(r)$ and $p_s(s)$ respectively. Then from elementary probability theory, if $p_r(r)$ and $p_s(s)$ are known and if $T^{-1}(s)$ satisfies the first condition stated above then the probability density function of the transformed gray level is given by:

$$p_s(s) = \left[p_r(r) \frac{d_r}{d_s} \right]_{r=T^{-1}(s)} \quad (1.11)$$

If the transformation is given by:

$$s = T(r) = \int_0^r P_r(w)dw \quad (1.12)$$

Then substituting $\frac{d_r}{d_s} = p_r(r)$ in **Eq. 1.11** we obtain $P_s(s) = 1$. Thus, it is possible to obtain a uniformly distributed histogram of an image by the transformation described by **Eq. 1.12**.

From the above discussions, it is clear that using a transformation function equal to the cumulative distribution of T (as given by **Eq. 1.12**) produces an image whose gray levels have a uniform density. This implies that such a transformation results in an

⁹ $T(r)$ is a transformation that produces a level s for every pixel value r in the original image.

increase in the dynamic range of the pixel gray values which produce a pronounced effect on the appearance of the image.

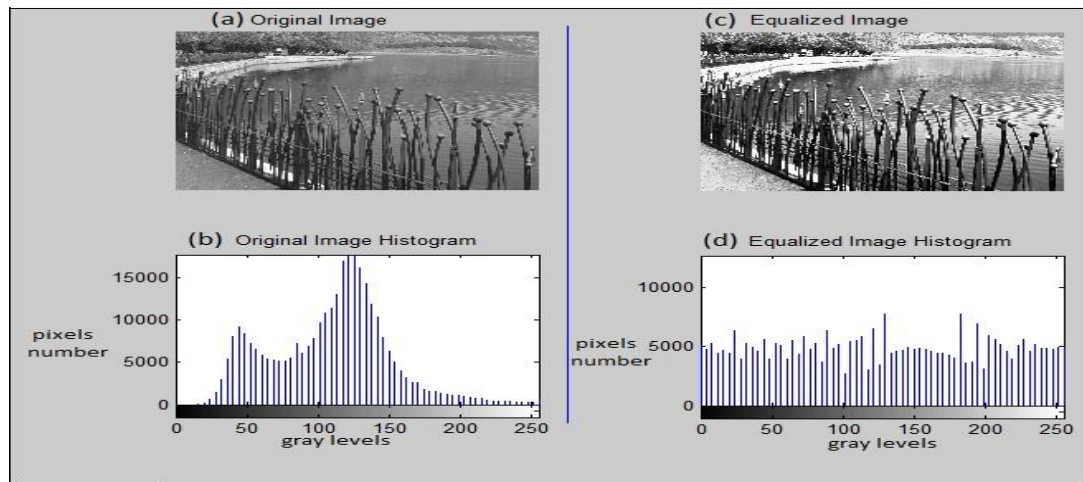


Figure N°06: Histogram equalization results: (a) original image, (b) histogram of the original image. (c) Enhanced image. (d) Equalized histogram.

C- guidance:

There are many other HISTROGRAM-BASED CONTRAST ENHANCEMENT techniques such as BBHE, DSIHE, MHE, MMBEBHE, RMSHE, GHE, LHE and IDBPHE

- **BBHE:** Brightness Bi-Histogram Equalization [11] [12].
- **DSIHE:** Dualistic Sub Image Histogram Equalization [13] [14].
- **MHE:** Multi Histogram Equalization [15].
- **MMBEBHE:** Minimum Mean Brightness Error Bi-Histogram Equalization [16].
- **GHE:** Global Transformation Histogram Equalization [17].
- **LHE:** Local Transformation Histogram Equalization.
- **IDBPHE:** Recursive Mean Separate Histogram Equalization [19] [14].

3.3.4- FREQUENCY DOMAIN METHODS OF IMA ENHANCEMENT:

Enhancement in the frequency domain is accomplished by high-pass, low-pass, and band-pass filtering of the original image. The task of enhancement in frequency domain implicates computing the Fourier transform of the image $f(x, y)$ (i.e. $F(u, v)$) and the filter transfer function $H(u, v)$ and taking the inverse Fourier transform of the product $F(u, v)H(u, v)$ [23]. The variations in gray level in an image represent the frequency component present in the image. A uniformly homogeneous image with constant gray value has 0 frequencies, while an image with adjacent black-and-white image has high spatial frequencies.

The convolution theorem states that the convolution in spatial domain is equivalent to multiplication in frequency domain. This implies that

$$G(u, v) = H(u, v)F(u, v) \quad (1.13)$$

where $G(u, v)$, $H(u, v)$, and $F(u, v)$ are the Fourier transforms of $g(x, y)$, $h(x, y)$, and $f(x, y)$ respectively. Taking the inverse Fourier transform of $G(u, v)$, we get

$$g(x, y) = \mathcal{I}^{-1}[H(u, v)F(u, v)] \quad (1.14)$$

It may be observed that by suitable selection of $h(z, y)$, we get a resultant image $g(z, y)$ which is an enhanced version of the original image $f(z, y)$. As- summing $f(x, y)$ to be point source of light, i.e., a unit impulse function whose Fourier transform is unity,

$$G(u, v) = H(u, v)$$

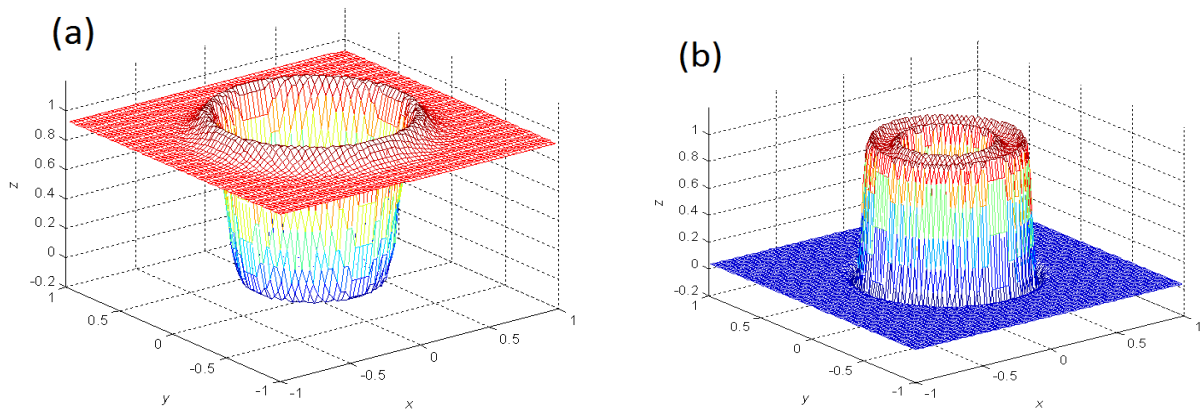


Figure N°07: Two-dimensional filter response (a) high-pass, (b) low-pass.

3.3.4.1 - Homomorphic Filter:

Homomorphic filters are widely used for compensating the effect of non-uniform illumination in an image. Pixel intensities in an image represent the light reflected from the corresponding points in the objects. An image $f(z, y)$ may be characterized by two components:

- (1) The amount of source light incident on the scene being viewed.
- (2) The amount of light reflected by the objects in the scene.

These portions of light are called the illumination and reflectance components, and are denoted $i(x, y)$ and $r(x, y)$ respectively. The functions $i(x, y)$ and $r(x, y)$ combine multiplicatively to give the image function $f(x, y)$:

$$f(x, y) = i(x, y)r(x, y) \quad [05] \quad \text{where } 0 < i(x, y) < \alpha \text{ and } 0 < r(x, y) < 1$$



Figure N°08: Homomorphic filtering: (a) input image, (b) result of Homomorphic filtering.

3.3.5 - NOISE MODELING:

During acquisition, coding, transmission, or processing, Noise is always present in digital images and very difficult to eliminate without the prior knowledge of noise model. Removal of noise from an image is one of the most important tasks in digital image processing. The main problem here is to succeed in eliminating as much noise as possible, while preserving the structures and details of the image. There are several approaches towards removing noise from an image depending on the nature of the noise, such as additive or multiplicative type of noise which is primarily inherent to the imaging procedure.

3.3.5.1 - Types of Noise in an Image and Their Characteristics:

Similar to other digital signals, digital images are also sometimes inadvertently corrupted by unwanted signals, called noise. Digital images are often deprived by different types of noise such as impulse noise during its acquisition and transmission phase. Such deterioration negatively influences the performance of many image processing techniques and a preprocessing module to filter the images is often required [24] [25].

Here is a description of the four important classes of noise:

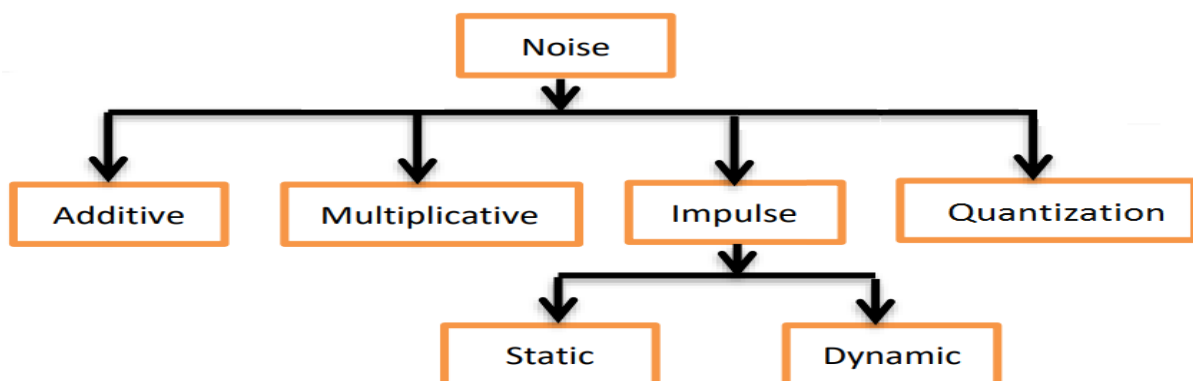


Figure N°09: noise classes.

- **Additive noise:** Sometimes images are degraded by noises generated from sensors which are mostly thermal whit Gaussian; it is essentially additive and

signal independent mathematically described by $g(x,y) = f(x,y) + n(x,y)$, where $g(x,y)$ is the result of the original image function $f(z,y)$ corrupted by the additive Gaussian noise $n(x,y)$.

- **Multiplicative noise:** it causes a convolution of the signal with the noise in the frequency domain and is therefore harder to remove than additive noise, which is merely added to the signal in both time and frequency [26]. The graininess noise from photographic plates and speckle noise from the imaging systems are multiplicative noise, which may be represented as $g(x,y) = f(x,y) * n(x,y)$, where $n(x,y)$ is the multiplicative noise.
- **Impulse noise:** quite often noises are impulse noise generated from sensors. Impulse noise can be assumed as an additive noise [27], and randomly damages the pixel, at random positions [28]. Normally, it appears as black and white speckles on the image [29]. Which can be modeled as $g(x,y) = (1 - p)f(x,y) + p.i(x,y)$ where $i(x,y)$ is the impulsive noise and p is a binary parameter that assumes the values of either 0 or 1.
- **Quantization noise:** it is essentially a signal dependent noise. This noise is characterized by the size of signal quantization interval. Such noise produces image-like artifacts and may produce false contours around the objects. The quantization noise also removes the image details which are of low-contrast [05].

3.3.6 – image restoration:

Removal, reduction of degradations, or improvement of the quality of images acquired by optical, electro-optical or electronic means is one of the most important tasks in digital image processing. Images may be hindered due to several reasons, e.g.:

- Imperfection of the imaging system
- Imperfection in the transmission channel
- Degradation due to atmospheric condition
- Degradation due to relative motion between the object and the camera

3.3.6.1 – Noise classification :

A- Salt and Pepper noise:

It is also called impulse noise, random noise or spike noise, S&P noise model, only a proportion of the entire image pixels are corrupted whereas other pixels are non-noisy. A standard S&P noise value may be either minimum (0) or maximum (255). The common intensity value for pepper noise is close to 0 and for salt noise is close to 255. Furthermore, the unaffected pixels remain unchanged.

$$n(x,y) = \begin{cases} 0, & \text{Pepper noise} \\ 255, & \text{Salt noise} \end{cases}$$

Where $n(x, y)$ the value intensity of the pixel.

B- Gaussian noise:

Named after Carl Friedrich Gauss, it is a statistical noise having a probability density function (PDF) equal to that of the normal distribution, which is also known as the Gaussian distribution. In other words, the values that the noise can take are Gaussian distributed.

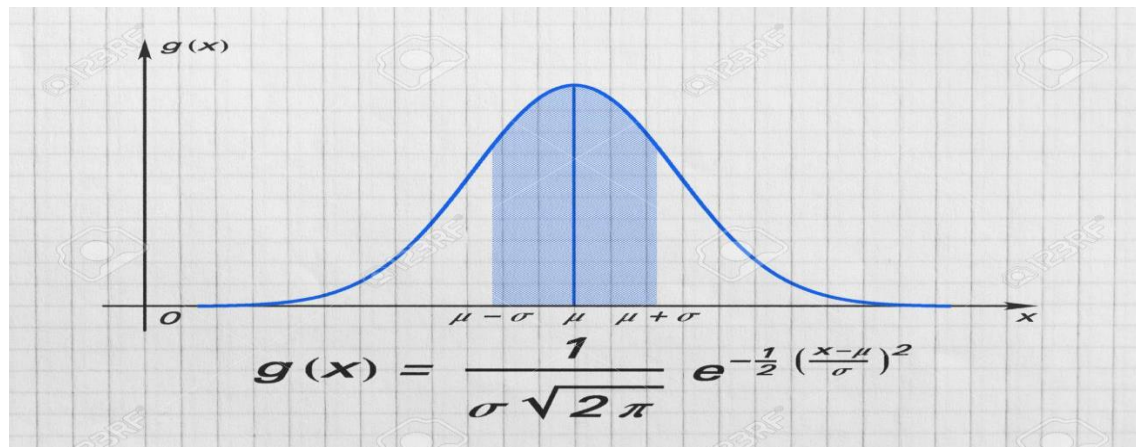


Figure N°10: Plot of Probability Distribution Function. Where g = gray value, σ = standard deviation and μ = mean.

The main sources of Gaussian noise in digital images arise during acquisition¹⁰, high temperature, and transmission¹¹. In digital image processing, Gaussian noise can be reduced using a spatial filter, but when smoothing an image, an unwanted result can cause the edges of the image to be erased and fine scaled. Conventional spatial filtering techniques for noise removal include: averaging (convolution filtering), median filtering, and Gaussian smoothing.

C- Poissonian Noise:

The appearance of this noise is seen due to the statistical nature of electromagnetic waves such as x-rays, visible lights and gamma rays. The x-ray and gamma ray sources emitted number of photons per unit time. These rays are injected in patient's body from its source, in medical x rays and gamma rays imaging systems. These sources are having random fluctuation of photons. Result gathered image has spatial and temporal randomness. This noise is also called as quantum (photon) noise or shot noise.

D- speckle noise:

Speckle noise is a common phenomenon in all coherent imaging systems like laser, acoustic, SAR and medical ultrasound imagery [40]. This noise is multiplicative noise [41],

¹⁰ for example, sensor noise caused by poor lighting.

¹¹ The noise may be caused by electronic circuit noise.

it can exist similar in an image as Gaussian noise. Its probability density function follows gamma distribution, [38] [39].

$$F(g) = \frac{g^{\sigma-1}}{(\sigma-1)! \sigma^\sigma} e^{-\frac{g}{\sigma}} \quad [42]$$

Where g is the gray level and σ is the variance.

3.3.6.2 – RESTORATION TECHNIQUES:

A- Mean Filter:

The Mean filter is a linear average filter. Here, the filter calculates the average value of the corrupted image in a predetermined area. Then, the intensity value of the central pixel is replaced by this average value. The average filter is a simple spatial filter. It is a sliding window filter that replaces the central value of the window with the average of all pixel's values in the kernel or window. The window is usually square, but it can be any shape.

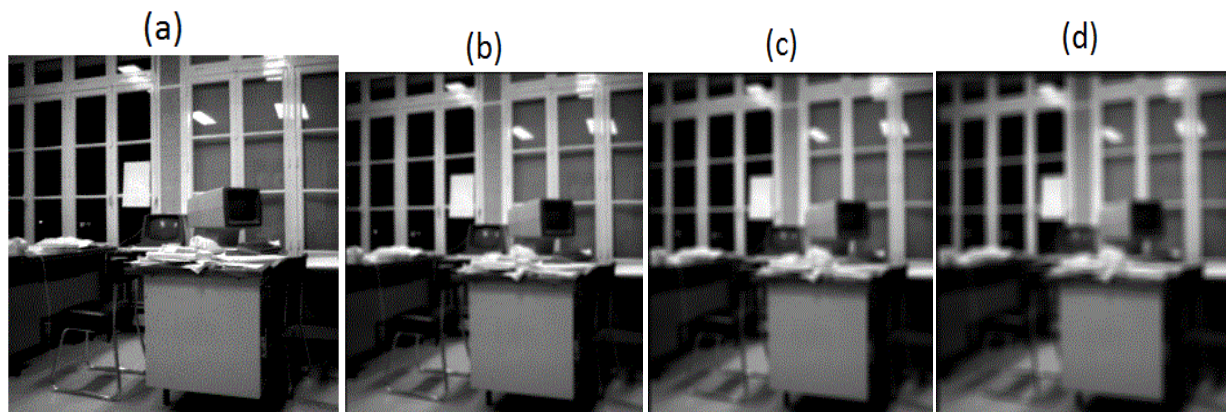


Figure N°11: Mean filter results (a) original image, (b) image with mean filter of 3*3 window, (c) image with mean filter of 5*5 window, (d) image with mean filter of 7*7 window.

The disappearance of fine details and the spreading of the contour as a function of the size of the kernel are striking in these examples. This is one of the drawbacks of linear filters. However, a finer choice of the kernel coefficients can make it possible to attenuate these defects to a certain extent. In fact, the Mean filter does not take into account the correlations between the pixels of an image; its performance in terms of maintaining image quality is therefore poor. The smooth filter presented in the following paragraph is an example of a linear filter whose coefficients, chosen with more care, allow less coarse processing of the image.

B- Gaussian filter (smooth filter):

Like the Mean filter, the effect of the smooth filter increases with the size of its core. The outlines and fine details are however better preserved than with the averaging device: thus, in the case of smooth $7 * 7$, there is no impression of blur on the window bars, which is annoying with the Mean filter, by using a Gaussian weighting, the smooth filter takes better account of the correlations between pixels, in particular for a texture (the gray level correlation function for a texture is often modeled by a Gaussian).

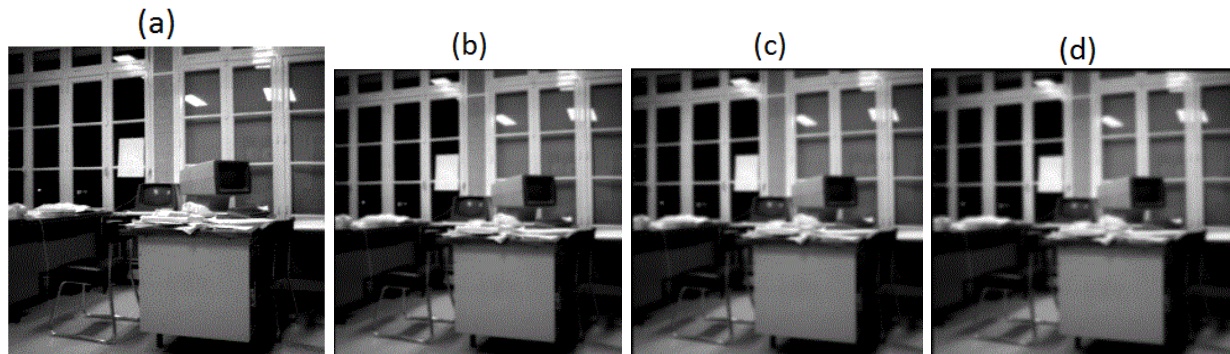


Figure N°12: Gaussian filter results (a) original image, (b) image with Gaussian filter of $3*3$ window, (c) image with Gaussian filter of $5*5$ window, (d) image with Gaussian filter of $7*7$ window.

The smooth filter is a good example of the performance that can be achieved with a linear finite impulse response filter. The big advantage of these filters is their ease of design and implementation, but they cannot be used for too fine work (the deterioration of the contours which they induce for example, will prevent a fine segmentation of the images). These limitations have therefore led to the design of non-linear filters such as median filter.

C- Inverse filter:

Suppose that h is the blur operator (unknown or given by calibration of measuring devices) and that we do not take into account the noise. The blurred image f verifies $f = h * u$ where u is the original image (which we want to find). An equivalent formulation is $F = H.U$ that means $U = \frac{F}{H}$ where F denotes the Fourier transform of f [44].

The reverse filter is the simplest of the filters. Under certain conditions, it can give very good results. It therefore consists of calculating $\frac{1}{H}$ and applying it to the blurred image. This is the best filter to deconvolve a noiseless image. However, it is not always possible to calculate $\frac{1}{H}$ because H can cancel out. An alternative is the **Van Cittert algorithm**¹².

¹² Van Cittert's algorithm is based on the frequency formulation of a convolution.

D- Wiener Filter:

The most important technique for removal of blur in images due to linear motion or unfocussed optics is the Wiener filter. From a signal processing standpoint, blurring due to linear motion in a photograph is the result of poor sampling. Each pixel in a digital representation of the photograph should represent the intensity of a single stationary point in front of the camera. Unfortunately, if the shutter speed is too slow and the camera is in motion, a given pixel will be an admixture of intensities from points along the line of the camera's motion. When the image is blurred by a known low-pass filter, it is possible to recover the image by inverse filtering.

However, inverse filtering is very sensitive to additive noise. The approach of reducing one degradation at a time allows us to develop a restoration algorithm for each type of degradation, and simply combine them. The Wiener filtering executes an optimal tradeoff between inverse filtering and noise smoothing. It removes the additive noise and inverts the blurring simultaneously [43].

E- Patch-dictionary based image recovery:

Varied versions of algorithms have been proposed for dictionary learning such as KSVD [30]. The idea is to combine the gains of patch methods such as NL-Means, with efficient learning methods in redundant dictionaries [31]. Dictionary learning was introduced by B. A. Olshausen and D. J. Field in articles [32] and [33]. The BM3D method (Block Matching 3D) introduced by Dabov, Foi, Katkovnik, and Egiazarian [34] in 2007, then extended by the same authors in [35] in 2008 and in the article [36] in 2009, this method which mixture of various techniques, is based on block processing and gives the best digital results in the state of the art of image denoising.

F- Non-Local Means filter (NL-Means):

This method is close to Denoising by dictionary learning. The major difference is that the image itself in some sort provides the dictionary. NL-Means is a patch method which is based on the self-similarity of the image. It was introduced by Buades, Coll, and Morel in 2005 [45]. Kervrann and Boulanger, Awate and Whitaker proposed similar methods in 2006 [46]. This method is more frequently referred to as NL- Means. The iterative approach of NL-Means was proposed by Brox and Cremers in 2007 [47], then detailed by Brox, Kleinschmidt, and Cremers in 2008 [48]. In 2009, Tschumperlé and Brun tried to link the patch denoising methods and the anisotropic diffusion methods [49]. Louchet and Moisan studied the association between total variation, Bayesian models and patch approaches in 2011 [50].

As shown by Buades et al. in [51], the principle of NL-Means consists in taking advantage of a redundancy of the long-distance information that can be found in the images, and then it is a question of finding similar pixels (patches) in the image, and calculating their weighted average according to their similarity with the pixel witch

meant to be denoised, The similarity between two pixels i and j depends on the similarity of the intensity gray level vectors $v(N_i)$ and $v(N_j)$, where N_k denotes a square neighborhood of fixed size and centered at a pixel k .

The definition of NL-Means as introduced by Buades et al [51] is

$$NL[v](i) = \sum_{j \in N_i} w(i, j)v(j) \quad (1.15)$$

Where N_i is a set of neighboring pixels of i the family of weights $\{w(i, j)\}_j$ depend on the similarity between the pixels i and j , and satisfy the usual conditions

$$0 \leq w(i, j) \leq 1 \text{ and } \sum_j w(i, j) = 1.$$

$$w(i, j) = \frac{1}{Z(i)} e^{-\frac{\|v(N_i) - v(N_j)\|_{2,a}^2}{h^2}}$$

$$Z(i) = \sum_j e^{-\frac{\|v(N_i) - v(N_j)\|_{2,a}^2}{h^2}}$$

Where $a > 0$ is the standard deviation of the Gaussian kernel, and the parameter h acts as a degree of filtering. It controls the decay of the exponential function and therefore the decay of the weights as a function of the Euclidean distances.

3.4 – multi-resolution processing:

Wavelets have become an important mathematical tool; they have established themselves in various fields of application. By their extraordinary aptitude for approximation and concentration of energy, the implication of the wavelet transform in the various algorithms imposing in compression of images becomes an essential tool. Therefore, it has been applied to almost all technical fields including image denoising, numerical integration, and pattern recognition [56].

The concept of multi-resolution processing is due to Stephane Mallat [54] and Yves Meyer [55]. A multi-resolution analysis of $L^2(\mathbb{R})$ is defined as a sequence of closed subspaces V_j of $L^2(\mathbb{R})$, $j \in \mathbb{Z}$, with the following properties [54] [55].

- $V_j \subset V_{j+1}$.
- $v(x) \in V_j \Leftrightarrow v(2x) \in V_{j+1}$.
- $v(x) \in V_0 \Leftrightarrow v(x + 1) \in V_0$.
- $\bigcup_{j=-\infty}^{+\infty} V_j$ is dense in $L^2(\mathbb{R})$ and $\bigcap_{j=-\infty}^{+\infty} V_j = \{0\}$

– A scaling function $\varphi \in V_0$, with a non-vanishing integral, exists such that the collection $\{ \varphi(x \leftrightarrow l) \mid l \in \mathbf{Z} \}$ is a Riesz basis of V_0 .

In a rapidly developing field, scientists are working hard to discover new technologies, and make it available in papers and several good ones concerning wavelets are already available, such as [57, 58, 59, 60, 61, 62]. Of these, [60] involves a brief introduction to multi-resolution processing (analysis), [57] define wavelets from an approximation theory point of view, [58] review the continuous and discrete wavelets, [62] focuses on the construction of wavelets, [59] looks at wavelets from a signal processing point of view and [61] compares wavelets with Fourier techniques.

3.5 – Image compression:

In an image, there is usually a relationship between the pixels, hence the existence of information redundancy. The goal of compression is to minimize or even eliminate this redundancy, the objective of which is to achieve the best possible fidelity rate for an available storage or transmission capacity.

There are two types of compression:

- **Lossless compression:** the reconstruction of the data is identical to the originals but the compression rate is very low.
 - No change in the image.
 - Only changes the way it is encoded on disc.
 - Display (= decompression) of the image: identical to the original.
- **Lossy compression:** Removes information less essential to the human eye.
 - Reduction of the number of data
 - Higher compression ratio than lossless compression.

The higher the compression ratio, the greater the level of loss and the more the image quality is degraded. [63]

There are many ways of encoding, representing and compressing images. There are a multitude of formats to compensate for the diversity of images, different in size, number of colors and method of representation (vector and bitmap). Each format is complementary to the others. Today, with the considerable technological advance, algorithms allow themselves to perform numerous calculations to compress and decompress an image. The ratios are getting larger and larger, without significant and real loss of image quality.

The latest few years show that compression methods (fractal¹³ and wavelet) are particularly promising. They show the need to associate computer programming with

¹³ An enlargement or reduction of a set of identical patterns by transformations: translations, rotations, symmetries.

mathematics. The future of compression can only be done through mathematical algorithms, and advances in mathematical research will surely lead to a breakthrough in image and file compression.

4 – Computer vision:

Computer vision is located in the heart of the intersection of several scientific fields (see **Figure N°13**). Neuroscience can help by first understanding human vision also we can see computer vision as a subfield of computer science, and for the developing of computer vision algorithm an algorithm theory or machine learning are essential.

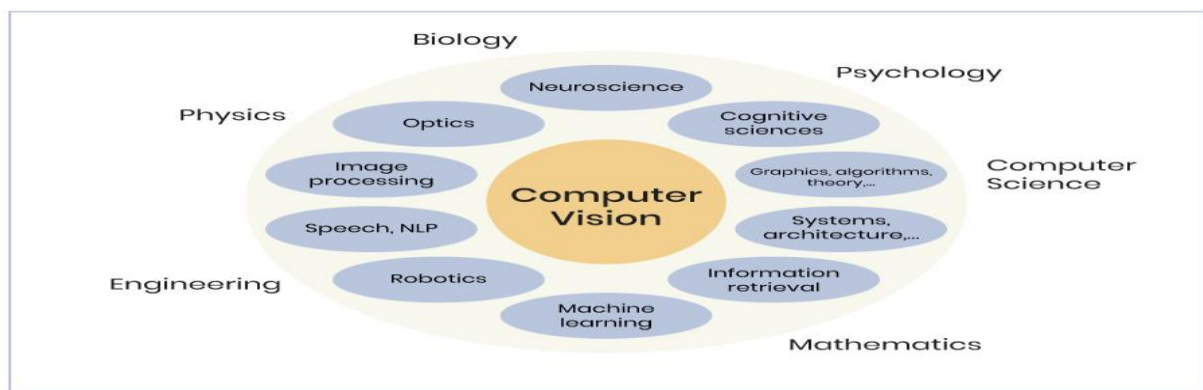


Figure N°13: Computer vision at the intersection of multiple scientific fields.

4.1 – segmentation, reconstruction, recognition:

In practice, David Marr's paradigm [64] translates into three stages of processing: segmentation, reconstruction and recognition.

- **Image Segmentation:** To be able to correctly process the mass of multimedia data conveyed throughout the day around the world segmentation is necessary. Image segmentation is the most important operation in an image processing system because it is located at the intersection between image processing and analysis. As a result, many researchers have worked on the development of dedicated methods and algorithms. In addition, it has been an important area of research for several years. As proof, the number of published works dealing with this problem is difficult to assess. This is the consequence of several elements: the diversity of the images, the complexity of the problem, the evolution of computing machines, and a fairly empirical evaluation of the results, there are many segmentation methods, which can be grouped into three main classes:
 1. **The edge approach:** we consider that the primitives to be extracted are the lines of contrasts separating regions of different and relatively homogeneous gray levels, or else regions of different texture that can be grouped into three categories: derivative methods, Analytical methods and deformable methods.

2. **The regions approach:** image segmentation by the region approach consists of dividing the image into regions. Several techniques of this approach are to be distinguished: Region growth, Segmentation by classification, Segmentation by division fusion.
 3. **Cooperative approach:** depending on the way in which two region and edge segmentation processes are made to cooperate, three different approaches can be distinguished: sequential cooperation, cooperation by merging results and mutual cooperation.
- **Image Reconstruction:** CT image reconstruction is a mathematical process that generates tomographic images from X-ray projection data acquired from many different angles around the patient. Image reconstruction has fundamental impacts on the quality of the image and therefore on the radiation dose. For a given radiation dose, it is desirable to reconstruct images with the lowest possible noise without sacrificing image precision and spatial resolution. Reconstructions that improve image quality can result in a reduction in radiation dose because images of the same quality can be reconstructed at a lower dose. At the origin of the method of reconstructing an object image from its projections (Tomography) is Radon's work on the determination of functions from their integrals in certain directions (1917). In 1956 Bracewell demonstrated the relationship between the Fourier transform and the Radon transform which gave rise to the 2 or 3D image reconstruction algorithm by the filtered back projection (FBP) method. It was not until 1963 that the first applications of medical tomography were carried out using X-rays. The main results of the time were due to Kuhn for obtaining the first tomographic images by simple rear projection and Cormack for the application of Radon's work to X-ray acquisitions. From 1970 the first computed tomography images were published and the development of the first X-ray scanners began. Neutrons as well as other radiations and particles were subsequently used as exploration projectiles for the development of several tomography techniques. The main techniques, currently used in various fields of industry and science, are: neutron tomography, electron tomography (electron microscope), nuclear magnetic resonance tomography and positron emission tomography (PET). Here we focus on the technique of Transmission Tomography using a neutron beam for the exploration of matter. Two major categories of reconstruction methods exist, analytical reconstruction and iterative reconstruction (IR).
 - **Image recognition:** A subcategory of Computer Vision and Artificial Intelligence represents a set of image detection and analysis methods to enable the automation of a specific task. It is a technology that is able to identify places, people, objects and many other types of elements within an image and draw conclusions by analyzing them. Theoretically, image recognition is based on Deep Learning, which is a sub-category of Machine Learning, refers to a set of techniques and technologies for machine learning, based on artificial neural networks (further informations are available in **CHAPTER 03**).

4.2-Applications of computer vision:

The number of images uploaded in the internet around the world is growing exponentially, cameras are everywhere there are images in Instagram, Pinterest and Twitter, videos on YouTube, feeds of security cameras, medical and scientific images, to understand their content computer vision is essential. Here is a non-exhaustive list of applications of computer vision:

- **Scene recognition:** is one of the hallmark tasks of computer vision. For instance, a photo of landmark can be compared to billions of photos on Google to find the best matches. We can then identify the best match and deduce the location of the photo.
- **Face detection:** it has been used for multiple years in cameras to take better pictures and focus on the faces. Smile detection can allow a camera to take pictures automatically. Face recognition is more difficult than face detection, but with the scale of today's data, companies like Facebook are able to get very good performance. Finally, we can also use computer vision for biometrics, using unique iris pattern recognition or fingerprints.
- **Mobile visual search:** With computer vision, we can do a search on Google using an image as the query.
- **Self-driving cars:** Autonomous driving is one of the hottest applications of computer vision. Companies like Tesla, Google or General Motors compete to be the first to build a fully autonomous car.
- **Automatic checkout:** Amazon Go is a new kind of store that has no checkout. With computer vision, algorithms detect exactly which products you take and they charge you as you walk out of the store¹⁴.
- **Augmented Reality:** AR is also a very hot field right now, and multiple companies are competing to provide the best mobile AR platform. Apple released **ARKit** in June 2017 and has already impressive applications¹⁵.
- **Virtual Reality:** VR is using similar computer vision techniques as AR. The algorithm needs to know the position of a user, and the positions of all the objects around. As the user moves around, everything needs to be updated in a realistic and smooth way.
- **Facial recognition:** Moscow is using facial recognition technology to monitor whether people are complying with quarantine orders. In China, algorithms have been programmed to recognize people wearing masks, and report anyone with a fever¹⁶.

¹⁴ see their video [here](https://www.amazon.com/b?node=16008589011) (<https://www.amazon.com/b?node=16008589011>) accessed 05 October 2020

¹⁵ check out the different [apps](https://www.madewitharkit.com/) (<https://www.madewitharkit.com/>) accessed 05 October 2020

¹⁶ Using facial recognition against covid-19 (<https://www.dw.com/en/using-facial-recognition-against-covid-19/av-53868752>) accessed 05 October 2020

5 – Conclusion:

We wanted this chapter to be a brief introduction to concepts related to the field of image processing and computer vision. The different definitions that are developed there are those of the elementary knowledge of this discipline.

Introduction:

Information today takes many forms. Textual information is certainly the most widespread. However, with the growth of the Internet and multimedia tools, textual information is no longer the only way to convey knowledge. The sound and especially the image take more and more importance. Don't we say that in some cases an image is worth a thousand texts? This state of affairs has resulted in an increasingly perceptible need to develop tools capable of processing the image, indexing it, finding it in a database, recognizing its shape, etc.

A quick review of the state of the art in artificial VISION shows us, for example, that the indexing of images is currently the subject of very abundant research in the field of image processing and computer vision. Several methods are proposed there for associating with an image a set of descriptors of its content, in order to measure the resemblance with the descriptors corresponding to a request. By way of example, we can cite the approaches by query [65], by examples and counter examples [65], by navigation [66] etc. Some methods combine several approaches such as the example approach with the navigation approach.

The aim of indexing images is to be able to find an image. A first technique consists in annotating the images, that is to say associating them with a small text or several key words on which we will carry out research subsequently. Another way is to search directly for images from their actual content and no longer from added data. This objective is broken down into two families of problems of different complexity:

- Be able, following an indexing to find an image starting from a practically identical image or simply starting from a single fragment.
- Be able to find similar classes, even if the images are very different in terms of the signal: mountain landscape, etc.

For this matter, a classification method is essential to create similarity classes. To this end, several classification methods have been proposed to date; they are inspired from well-known techniques in pattern recognition and computer vision.

2 - The motivations for image classification:

The goal of image classification is to develop a system capable of automatically assigning a class to an image. Thus, this system makes it possible to perform an expert task which can be costly to acquire for a human being due in particular to physical constraints such as concentration, fatigue or the time required by a large volume of image data.

The applications of automatic image classification are numerous and range from document analysis to medicine and the military [67]. Thus we find applications in the medical field such as the recognition of cells [68], tumors in mammograms [69]; in agriculture such as pollen classification [70], recognition of soil type and grains [71] [72], the classification of herbs [73]; in the document domain such as handwriting recognition for checks, postal codes [74], cards [75]; in the urban domain such as the recognition of traffic signs [76], the recognition of pedestrians [77] [78] [79] [80], vehicle detection [81], building recognition [82] to aid in localization; in the field of biomarkers such as face recognition [83] [84], fingerprints, irises [85] [86]. The common point of all these applications is that they require the establishment of a processing chain from the available images composed of several steps in order to provide a decision as an output. Each step in the establishment of such a classification system requires the search for appropriate methods for optimal overall performance; namely the feature extraction phase and the learning phase. Typically, we have image data from which we need to extract relevant information translated into digital vectors. This extraction phase allows us to work in a digital space. It is then a matter of developing, in the learning phase, from these initial data, a decision function to decide whether a new data belongs to one of the classes involved.

The feature extraction phase can be preceded by a so-called pre-processing phase. The purpose of this phase is to clean up the image (as mentioned in the first chapter of this thesis), that is, to isolate the informative or interesting content in the image [87]. This operation thus makes it possible to conceal or attenuate any information likely to harm the description of the relevant content during the feature extraction phase. We thus find techniques of noise attenuation, edge reinforcement, image improvement techniques such as contrast enhancement, reduction of the image size by binarization [88], the reduction of the image to its visual primitives such as skeletonization [89] [90] [91] or even the extraction of contours using filtering techniques. The reader can refer to a state of the art of pretreatment techniques in [05] and [87].

3 - Classification /clustering:

3.1 - Clustering:

Clustering consists of grouping individuals into classes. Other terms such as segmentation, vector quantization, digital taxonomy and unsupervised learning are also used to refer to this same process. The terminology used often depends on the context of use.

Indeed, the clustering can feed various and varied applications in the fields of exploration, data analysis, etc. These applications range from image segmentation and shape classification to document indexing and other data mining uses.

3.2 - Classification:

Classification consists of assigning individuals to pre-existing classes. These classes can optionally be obtained by clustering. We then say that there is a learning process. It is a Supervised Learning task where output is having defined labels (discrete value)

It can be a binary or multi-class classification. In binary classification, the model predicts 0 or 1; yes, or no, but in case of multi-class classification, the model predicts more than one class.

3.2.1 - Binary classification:

A classification problem in which the label space is binary, that is, $Y = \{0, 1\}$ is called a binary classification problem. It's a type of classification task that predicts one of two mutually exclusive classes. For example, a machine learning model that classifies emails as "junk" or "legitimate".

Other examples of binary classification:

- Identify whether a painting was painted by Picasso or not.
- Identify whether or not an image contains a giraffe.
- Identify whether or not a molecule can treat depression.
- Identify if a financial transaction is fraudulent.

3.2.2 - Multi-class classification:

A classification problem in which the label space is discrete and finite, that is, $Y = \{1, 2, \dots, C\}$ is called a multi-class classification problem. It is the number of classes.

Here are some examples of multi-class classification problems:

- Identify in which language a text is written.
- Identify which of the 10 Arabic numerals is a handwritten number.
- Identify a facial expression from a predefined list of possibilities (anger, sadness, joy, etc.).
- Identify what species a plant belongs to.
- Identify the objects present in a photograph.

PARAMENTER	CLASSIFICATION	CLUSTERING
Type	used for supervised learning	Used for unsupervised learning
Basic	process of classifying the input instances based on their corresponding class labels	grouping the instances based on their similarity without the help of class labels
Need	it has labels so there is need of training and testing dataset for verifying the model created	there is no need of training and testing dataset
Complexity	more complex as compared to clustering	less complex as compared to classification
Example Algorithms	Logistic regression, Naive Bayes classifier, Support vector machines etc.	k-means clustering algorithm, Fuzzy c-means clustering algorithm, Gaussian (EM) clustering algorithm etc.

Table 01: Differences between Classification and Clustering.

4 - Machine learning for image classification:

Machine learning is the “Field of study that gives computers the ability to learn without being explicitly programmed.” Arthur Samuel (1959)

“A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.” Tom Mitchell (Machine Learning 1997).

Classification involves assigning each pixel in the image a class (label). This assignment can be made based on regions for which we know the classes of belonging at first, so we speak of supervised or unsupervised classification.

This part of the thesis allows you to familiarize yourself with the various notions related to machine learning, in its supervised and unsupervised forms. All of the concepts discussed can be further explored by reading [92], whose notations we take here.

4.1 – The general process of machine learning Classification and its ingredients:

There are four main steps to the currency machine learning process:

1. Choice of data.
2. Calculation of the similarities between the n individuals from the initial data.
3. Choice of a classification and execution algorithm.
4. Interpretation of the results:
 - Evaluation of the quality of the classification,
 - Description of the classes obtained.

Machine learning is based on two fundamental pillars:

- On the one hand, the data, which are the examples from which the algorithm will learn.
- On the other hand, the learning algorithm, which is the procedure that we run on this data to produce a model. Running a learning algorithm on a data set is called training.

These two pillars are equally important. On the one hand, no training algorithm will be able to create a good model from data that is not relevant, it is the concept of garbage in; garbage out which states that a training algorithm to which data is supplied poor quality will not be able to do anything with them other than poor quality predictions. On the other hand, a model learned with an unsuitable algorithm on relevant data will not be of good quality.

4.2 – Some types of Machine learning problems:

Machine learning is quite a large field, and in this section, we list the broadest classes of problems it is interested in.

Learning algorithms can be categorized according to the type of learning they employ, if the classes are predetermined and the examples labeled then we speak of supervised learning. When the system or the operator only has examples, but not labels, and the number of classes and their nature have not been predetermined, we speak of unsupervised learning. furthermore, there is also in between, when the examples are half labeled it's called Semi-supervised learning

The idea of designing agents capable of learning like children rather than attempting to directly reproduce the intelligence of an adult could be realized through the Reinforcement learning.

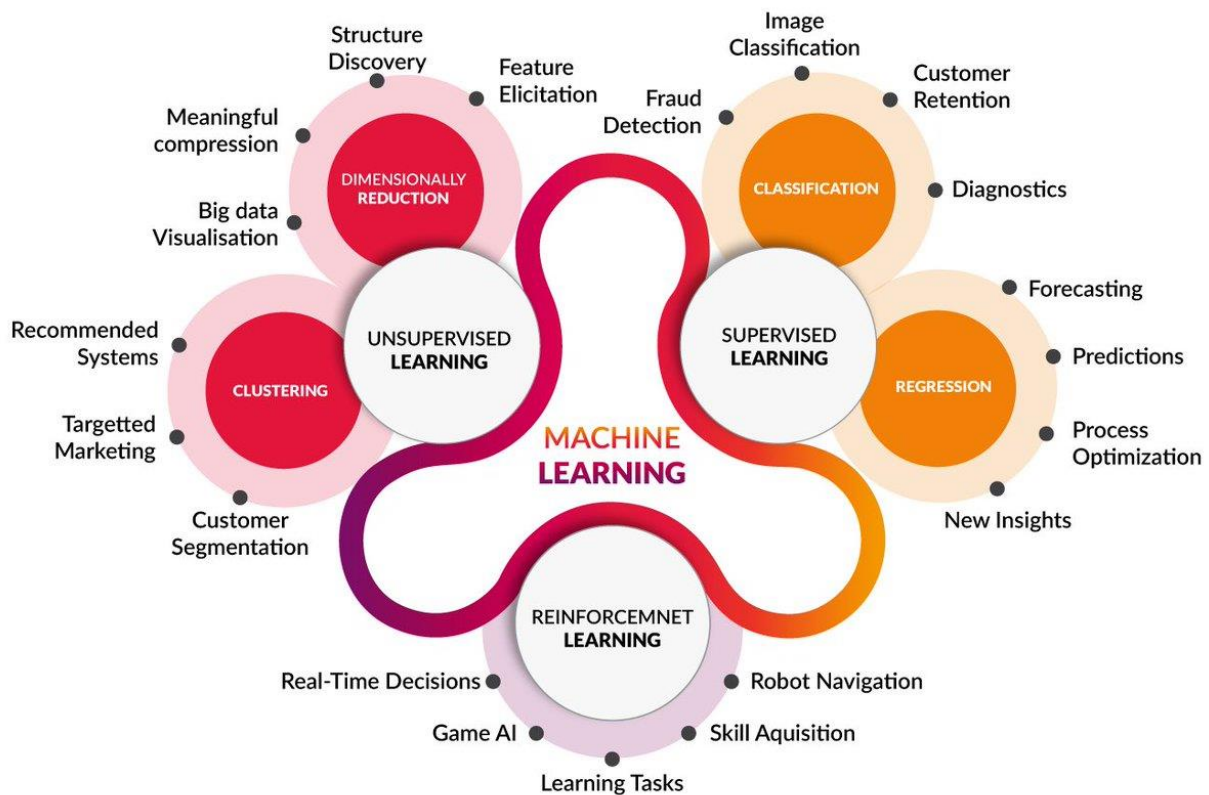


Figure 14: Some types of Machine Learning problems¹⁷.

4.2.1 - Supervised learning:

Supervised learning is perhaps the easiest type of machine learning problem to grasp: its purpose is to learn how to make predictions, from a list of labeled examples, i.e. accompanied by the value to predict (see **Figure N°15**). The labels serve as a "teacher" and oversee the learning of the algorithm.

We call supervised learning the branch of machine learning which is interested in problems that can be formalized as follows: given n observations $\{x_i\}_{i=1, \dots, n}$ described in a space X , and their labels $\{y_i\}_{i=1, \dots, n}$ described in a space Y , we suppose that the labels can be obtained from the observations thanks to a fixed and unknown function $\varphi: X \rightarrow Y : y_i = \varphi(x_i) + \epsilon_i$, where ϵ_i is a random noise. It is then a question of using the data to determine a function $f: X \rightarrow Y$ such that, for any pair $(x, \varphi(x)) \in X \times Y$, $f(x) \approx \varphi(x)$.

¹⁷ This Picture was taken from (<https://medium.com/ml-research-lab/machine-learning-algorithm-overview-5816a2e6303>) accessed 05 October 2020.

There are generally three types of problems to which supervised learning is applied: supervised classification, regression, and time series. These three types of problems are differentiated according to the type of labels provided by the oracle. In the framework of this thesis, we will only be interested in classification.

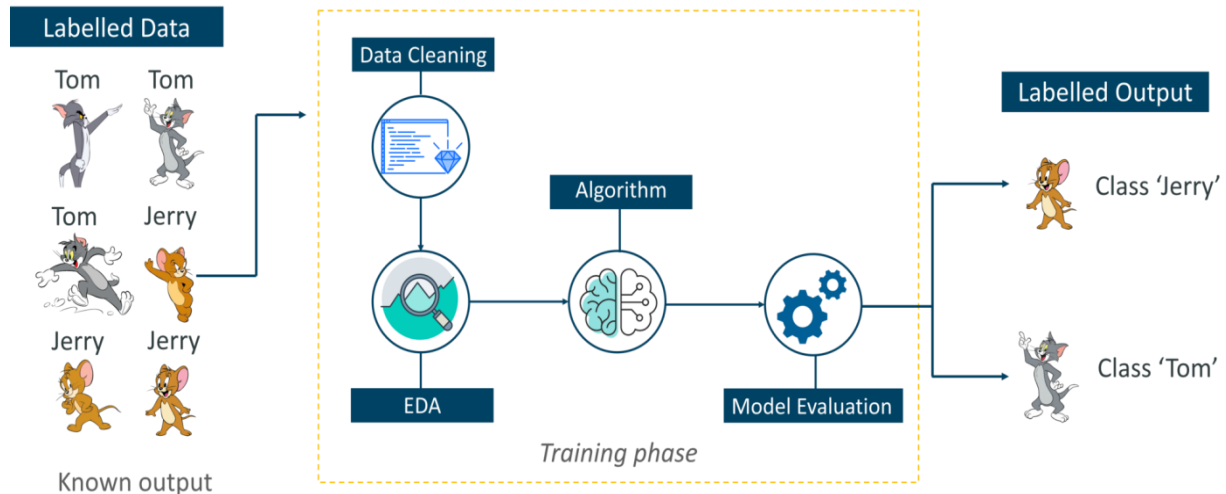


Figure N°15¹⁸: Supervised learning.

4.2.1.1 – Some supervised learning algorithms:

Most supervised learning algorithms attempt to find a model (a mathematical function) that explains the relationship between input data and output classes. These sets of examples are therefore used by the algorithm. We can therefore cite supervised pixel classification algorithms such as Decision Tree [93] [94], The K nearest neighbors (KNN) [156] or Support vector machine (SVM) [159].

A- K nearest neighbors (KNN):

It's a very simple and straightforward approach. It does not require learning but simply the storage of learning data. Its principle is as follows. Data of unknown class is compared to all stored data. For the new data, we choose the majority class from among its K closest neighbors (it can therefore be heavy for large databases) in the sense of a chosen distance.

B- Support Vector Machine (SVM):

Support vector machine (SVM) is one of the most popular methods in the family of supervised approaches, and of kernel-based, classification methods. It was developed by Vapnik in 1995, and remains to this day one of the most popular algorithms. Used, especially for pattern recognition

¹⁸ This picture was taken from (<https://www.edureka.co/blog/introduction-to-machine-learning/>). Accessed 05 October 2020

C- Decision tree:

A decision tree is a decision support tool representing a set of choices in the graphic form of a tree. The various possible decisions are located at the ends of the branches (the “leaves” of the tree), and are reached according to decisions made at each stage. The decision tree is a tool used in various fields such as security, data mining, medicine, etc. It has the advantage of being readable and quick to execute. It is also a representation that can be calculated automatically by supervised learning algorithms.

4.2.2 – Unsupervised learning:

If only unlabeled examples are available, and if the classes and their number are unknown, it is called unsupervised learning, or clustering. In this case, the learning comes down to targeting the homogeneous groups of examples existing in the data, that is to say to identifying groups, and that the most different examples are separated into different groups, the notion of similarity being most often reduced to a function of distance between pairs of examples.

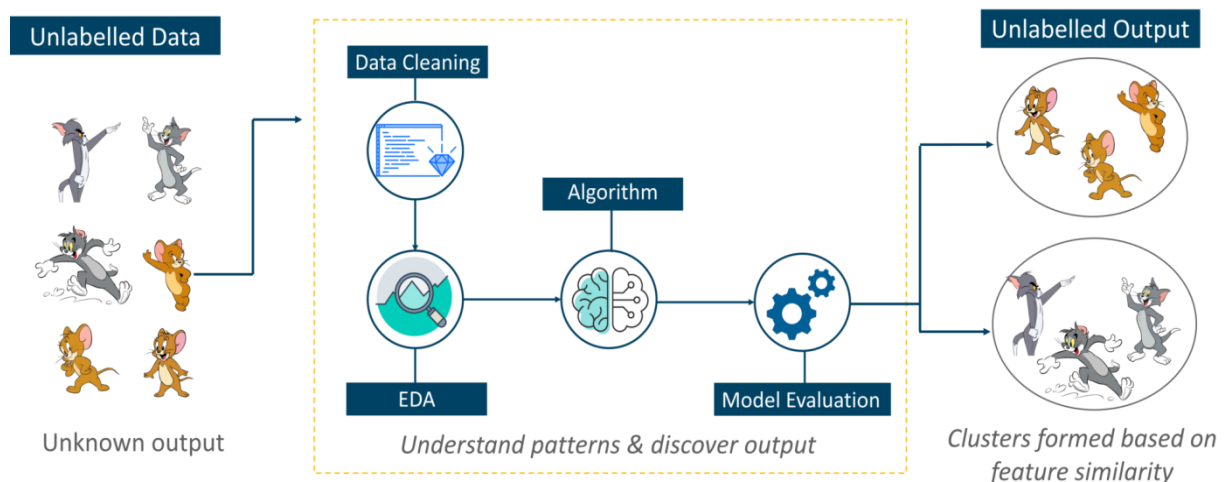


Figure 16: ¹⁹Unsupervised learning

4.2.2.1 – Some unsupervised learning algorithms:

We can therefore cite unsupervised pixel classification algorithms such as Fuzzy C-Means proposed by Joe Dunn in 1974, k-means and ascending hierarchical classification.

¹⁹ This picture was taken from (<https://www.edureka.co/blog/introduction-to-machine-learning/>). Accessed 10 October 2020

A- Hierarchical ascending classification:

This involves iteratively grouping the individuals together, starting from the bottom (the two closest) and gradually building a tree, or dendrogram, finally grouping all the individuals into a single class, at the root. This supposes knowing how to calculate, at each stage or grouping, the distance between an individual and a group as well as that between two groups. This therefore requires for the user of this method, to make an additional choice: how to define the distance between two groups knowing that of all the pairs of individuals between these two groups.

The ascending hierarchical classification is an iterative classification method whose principle is simple.

- We start by calculating the dissimilarity between the N objects.
- Then the two objects are grouped together, the grouping of which minimizes a given aggregation criterion, thus creating a class comprising these two objects.
- The dissimilarity between this class and the N-2 other objects is then calculated using the aggregation criterion. Then the two objects or object classes are grouped together, the grouping of which minimizes the aggregation criterion.

We continue in this way until all the objects are grouped together.

B- K-Means:

The k-means algorithm developed by McQueen in 1967 [95], one of the simplest unsupervised learning algorithms, called the mobile center algorithm [96] [97], he attributes each point in a cluster whose center (centroid²⁰) is closest. The center is the average of all points in the cluster; its coordinates are the arithmetic mean for each dimension separately from all points in the cluster, hence each cluster is represented by its center of gravity.

C- Fuzzy C-means:

(FCM) is an unsupervised fuzzy classification algorithm. Coming from the C-means algorithm, it introduces the notion of fuzzy set in the definition of classes: each point in the set of data belongs to each cluster with a certain degree, and all the clusters are characterized by their center of gravity. Like other unsupervised classification algorithms, it uses a criterion of minimizing intra-class distances and maximizing inter-class distances, but giving a certain degree of belonging to each class for each pixel. This algorithm requires prior knowledge of the number of clusters and generates the classes by an iterative process while minimizing an objective function. Thus, it makes it possible to obtain a fuzzy partition of the image by giving each pixel a degree of belonging (between 0 and 1) to a given class. The cluster with which a pixel is associated is the one with the highest degree of membership.

²⁰ Is a point (either imaginary or real) at the center of a cluster.

4.2.3 – Semi-supervised learning:

As you might expect, semi-supervised learning involves learning labels from a partially labeled dataset. The first advantage of this approach is that it avoids having to label all of the training examples, which is relevant when it is easy to accumulate data but their labeling requires a certain amount of human labor. Take image classification, for example: it's easy to get a database of hundreds of thousands of images, but having the label we're interested in for each one can be very labor intensive. Additionally, labels given by humans are likely to reproduce human biases, which a fully supervised algorithm will in turn reproduce. Sometimes semi-supervised learning avoids this pitfall.

4.2.4 – Reinforcement learning:

In reinforcement learning, the learning system can interact with its environment and perform actions. In return for these actions, he gets a reward, which can be positive if the action was a good choice, or negative if it wasn't. The reward can sometimes come after a long series of actions; this is the case, for example, for a system that learns to play go or chess. Thus, learning in this case consists in defining a policy, that is to say a strategy to systematically obtain the best possible reward.

The main applications of reinforcement learning are in games (chess, go, etc.) and robotics. This subject goes well beyond the scope of this thesis.

5 – Conclusion:

We have seen a generality on the designs of the classification methods and a superficial overview on the principles of the first great approach which infers from a sample of classified examples a procedure (decision function) for classifying the new unlabeled examples. Discrimination (or supervised methods) can be based on notions of proximity (closest neighbors) or even on research in hypothesis spaces (decision trees, artificial neural networks).

We saw that clustering allows objects (individuals or variables) to be grouped into a limited number of groups or classes (clusters). Classification consists of grouping the pixels of the image with fairly similar characteristics, into subsets of classes.

This list of possible learning tasks is not exhaustive. Likewise, it is possible to find cases of so-called semi-supervised learning, in which the data information is not complete [100] [101]. In the chapter that follows we will see the learning model of deep neural networks. This is used for many different tasks these days, whether supervised or not.

Introduction:

Before we get to what Deep learning (DL) is, we need to introduce two main concepts: the first is the AI (artificial intelligence), and the second is the concept of ML (machine learning).

Artificial Intelligence (AI) is a large field, where we try to make machines mimic human behavior with the aim of making them so powerful to perform many types of tasks such as problem solving, knowledge representation, voice recognition, and many others. The basic idea is to put knowledge into the machine. Thanks to these two areas, there are sophisticated systems capable of changing their behavior without the need to make changes to their code, but only to their training data. So, with this wave of advanced machine learning techniques taking AI a step forward, where does DL fit in or what does DL bring in there.

Nowadays, DL is the center of attention since its realization is much more important than any other machine learning algorithm in such complex tasks, for example, we mark the following:

- **Image processing and object recognition** in [98] which show us progress in using deep convolutional networks for object recognition, and the adoption of deep learning by the computer vision community.
- **Speech recognition and signal processing** in [99] which present the results obtained in phonetic classification for automatic speech recognition as the first industrial application of deep learning.

2 – Machine Learning:

Machine learning is driven by completing tasks that are difficult to define exhaustively or by simple rules in traditional programs. For example, developing an AI algorithm to play the game of Go while respecting the rules of the game is relatively simple to program because the set of rules can easily be defined (Each player plays a single pawn in turn, he does not can only place his pawn on a valid square, etc.). However, optimally playing the AI machine in order to achieve victory is impossible to define simply. This is because the game of Go does not have a known optimal strategy, which makes it difficult to create a series of rules for the machine to follow in order to win. In addition, the number of possible states of the game as well as the possibilities in each state is so large that it is impossible to describe everything in a typical program for an AI.

Let's take another example with computer assisted vision. Recognizing the face of a human being in an image may seem like a simple task to us because we can do it without thinking. But when we combine the wide variety of possible faces with all of the layouts of those faces in an image, it's impossible to simply describe these by the value of each pixel in the image. In both of these situations, machine learning can train statistical models so that they find on their own the knowledge necessary to accomplish these tasks using examples in our data.

A formal definition of machine learning has been proposed by **T. Mitchell**: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ". We will call this computer program capable of learning (statistical model or learning model). Let us first see this model as a black box, able to take as input data from the outside (for example, images from a camera, network traffic from a router, etc.) and return an output (for example, AI decision making, description of an image, etc.). This model has parameters θ which make it possible to influence its output depending of the input (see **Figure N°17**).

As explained in Michell's definition, learning requires experience E . This typically consists of a D_{train} learning Dataset that the model analyzes during the learning process. This learning of a task T is done using a cost function J . This cost function is calculated on a D_{valid} dataset, separate from D_{train} in order to measure the performance of the learned model; it is the performance measure P . During training, the model must therefore be able to modify its parameters θ using the D_{train} training dataset to improve its performance measured by P . The fact that the performance is measured on a D_{valid} dataset separate from D_{train} implies a capacity for generalization of the model, which means an ability to respond to cases that he did not see during his experience E . The goal of the model is to make its parameters θ tend towards an optimal θ^* which minimizes J on D_{valid} (the measure P).

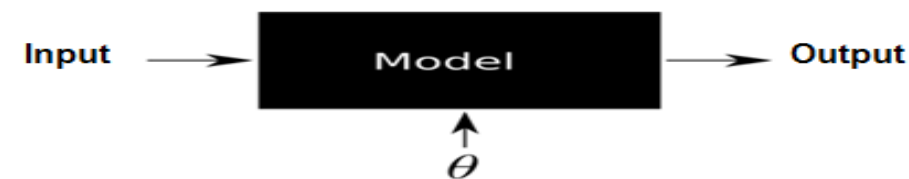


Figure N°17: Learning model seen as a black box.

3 -Artificial Neural networks:

Artificial Neural networks are a machine learning models capable of representing a relationship between data from a X space and an output space Y . They are used in many fields, such as computer-assisted vision [102] [103] [104], natural language processing [105], audio analysis [106] [107], but also to develop AI machines capable of playing in games [83] or used as a personal assistant (such as Amazon Alexa, Apple Siri, Microsoft Cortana or the Google Assistant).

The basic unit of calculation is the neuron. This takes several signals as input and interprets them to send a new signal to other neurons or to the output of the neural network, i.e. the output of the model. There are lots of architectures for building ANNs (see **Section 4** of this chapter).

Before we get to introduce artificial neural networks, we will go through several biological concepts about neurons, then, start by presenting a model made up of a single neuron, called the perceptron model. This will allow us to highlight the basic mechanisms of any artificial neural network.

3.1 – The biological neuron:

The human brain is the best model of the machine, an incredibly fast all-rounder and above all endowed with an incomparable capacity for self-organization. Its behavior is much more mysterious than the behavior of its basic cells. It is made up of a large number of basic biological units (1,000 to 10,000 synapses per neuron). Nerve cells called "neurons" are the building blocks of the central nervous system. They are made up of three essential parts: the cell body, the dendrites and the axon [139].

- A. **The cell body:** It contains the nucleus of the neuron and performs the biochemical transformations necessary for the synthesis of enzymes and other molecules that ensure the life of neurons. Its shape is pyramidal or spherical in most cases; it often depends on its position in the brain. This cell body is a few microns in diameter [140].
- B. **The dendrites:** Each neuron has a hair of dendrites. These are thin tubular extensions, a few tenths of microns in diameter and a few tens of microns in length. They are the main receptors of the neuron which are used to pick up the signals which reach it [140].
- C. **The axon:** The axon, which is strictly speaking the nerve fiber, serves as a means of transport for the signals emitted by the neuron. It differs from dendrites by its shape and the properties of its outer membrane. Indeed, it is generally longer than the dendrites, and branches out at its end, where it communicates with other neurons, while the ramifications of the dendrites occur rather near the cell body.

To form the nervous system, neurons are connected to each other in complex spatial distributions. Transmission between two neurons is not direct. In fact, there is an intercellular space of a few tens of Angstroms²¹ (10⁻⁹ m) between the axon of one neuron and the dendrites of another neuron. The junction between two neurons is called the synapse [139] [140].

²¹ A unit of length equal to one hundred-millionth of a centimeter, 10⁻¹⁰ meter, used mainly to express wavelengths and inter-atomic distances.

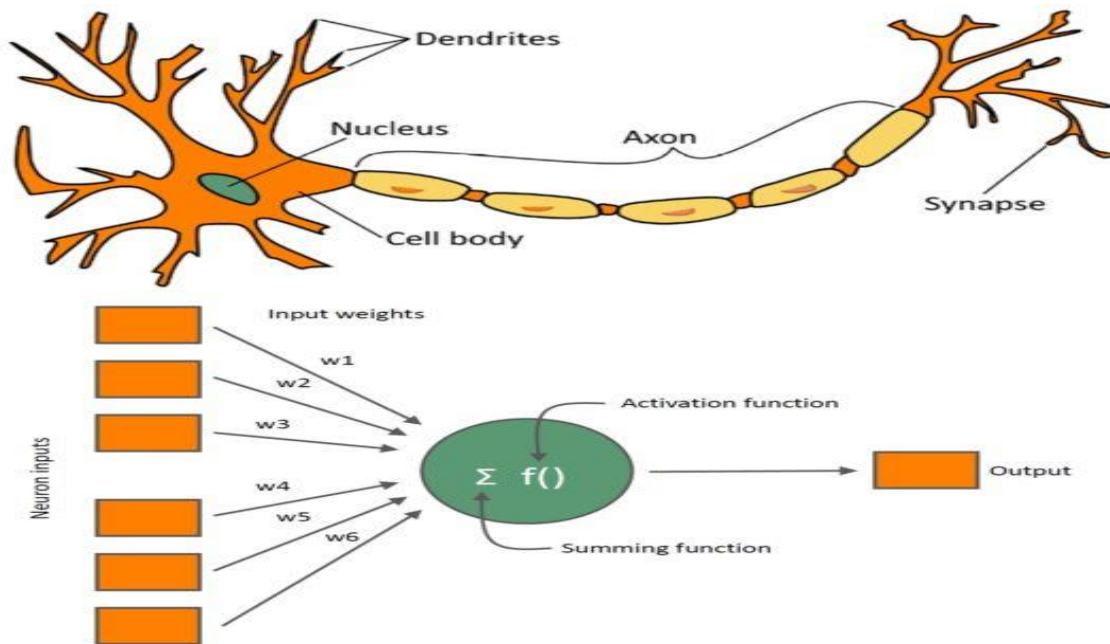


Figure N°1822: Illustration of a biological neuron (up) and its mathematical model (down).

3.2 – The perceptron model:

In its simplest version, the perceptron is a neural network made up of only one neuron, which takes n binary data as input. Each of its inputs i is weighted by a weight noted w_i . The neuron can take the states "1" or "0" (respectively active or non-active) depending on its weighted inputs and a bias noted $\beta \in \mathbb{R}$. This state represents the output of the model. It is therefore possible to represent the perceptron as a parametric function $f_{\theta}: \{0, 1\}^n \rightarrow \{0, 1\}$ with θ the set of its parameters, i.e. the bias β and the weights (w_1, \dots, w_n) . The output of a perceptron for an input vector $x \in \{0, 1\}^n$ is calculated such that:

$$f(x, w) = H(z(x, w) + \beta),$$

With $H(t)$ the Heaviside function defined for all $t \in \mathbb{R}$ as $H(t) = I_{\{t > 0\}}$ and $z(x, w)$ the weighted sum of the inputs:

$$z(x, w) = \sum_{j=1}^n w_j x_j.$$

Intuitively, the weights w_1, \dots, w_n represent the importance given to each entry for perceptron activation. As a reminder, I_A is the indicator function which is equal to 1 if condition A is verified and 0 if not. The β bias can be seen as adding a threshold to the difficulty of activating the perceptron. Indeed, if the weighted sum $z(x, w)$ exceeds $-\beta$ (the opposite of bias), the perceptron activates otherwise it remains inactive.

²² This picture was taken from (<https://www.ee.co.za/article/application-of-machine-learning-algorithms-in-boiler-plant-root-cause-analysis.html>) accessed 05 October 2020

To simplify the notations, we will include the bias β in the weight vector by adding an input constant $x_0 = 1$ (the bias therefore becomes w_0). We obtain the parametric function $f_{\theta}: \{0, 1\}^{n+1} \times \mathbb{R}^{n+1} \rightarrow \{0, 1\}$ such that

$$f(\mathbf{x}, \mathbf{w}) = H(z(\mathbf{x}, \mathbf{w})) = H\left(\sum_{j=0}^n w_j x_j\right) \quad (3.1)$$

Example of e-mail classification Consider a perceptron modeled by the function f which aims to classify the e-mails received with labels "Important" or "Not important". The x_j entries are the characteristics of received e-mails, such as "Sent by a contact", "Contains an attachment", "Is an automatic reply". Email is tagged as "Important" when the neuron is active and "Not important" when the neuron remains inactive.

Let us fix the weights associated with $w_1 = 2, w_2 = 1, w_3 = -1$ and the bias at $\beta = -0,5$ (or $w_0 = -0.5$), as described in **Figure N°19**. In this example, we can see that an email is tagged as important if it is sent by a contact or contains an attachment, but it is not an automatic reply.

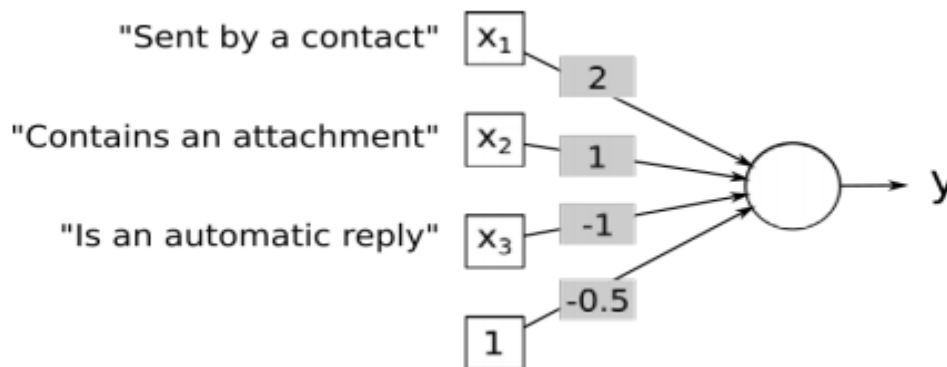


Figure 19: Graphical representation of a perceptron described in the Email Classification Example²³.

Now suppose we have no knowledge of what makes mail important and what not. More precisely, we do not know the weights w_j associated with each entry x_j . However, we do have access to a dataset of e-mails, hand-labeled by users with the labels "Important" and "Not important" as well as the characteristics associated with each (for example, "sent by a contact", "contains an attachment", ...). It is then possible to extract knowledge concerning the description of an important email from these examples by looking for the weights associated with each connection that best correspond to this dataset. This research phase will correspond to the learning of the perceptron. Once this process is completed on this dataset, the perceptron should be able to generalize this classification on new incoming mails.

²³ The squares represent the entrance to the perceptron and the circle represents the neuron. The weights are written on the connections between the inputs and the neuron.

Learning requires a function differentiable at any point to calculate the output of the perceptron. Instead of using Heaviside's function as an activation function, we will introduce the sigmoid function (see **section 3.3.1**).

The training of the parameters \mathbf{w} is done by minimizing a J_{train} cost function on the D_{train} training dataset. This function represents errors made by the model (with its current settings) on the classification of emails in our training dataset. In our example, we'll define the cost function as:

$$J_{\text{train}}(\mathbf{W}) = \frac{1}{K} \sum_{k=1}^K L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \mathbf{w})$$

With $\mathbf{x}^{(k)}$ and $\mathbf{y}^{(k)}$ which are respectively the characteristics and the label of the k^{th} example of the learning dataset. L corresponds to the function used to calculate the error on an example of the dataset. In our case we choose as function:

$$L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \mathbf{w}) = \frac{1}{2} (f(\mathbf{x}^{(k)}, \mathbf{w}) - \mathbf{y}^{(k)})^2$$

The J_{train} cost function therefore represents the root mean square error between the output given by the perceptron and the expected value on the entire learning dataset. The goal of training is to find the vector of parameters \mathbf{w}^* which minimizes J_{train} . We start by initializing the value of the parameters \mathbf{w} by the null vector:

$$\mathbf{w}_0 = (0, \dots, 0)^\top$$

With \mathbf{w}_0 denoting the state of \mathbf{w} at time 0 which means at the start of learning. The vector of parameters is iteratively modified using the following rule:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \nabla J_{\text{train}}(\mathbf{w}_t)$$

\mathbf{w}_t Represents the state of \mathbf{w} at iteration t , $\alpha \in (0, 1]$ is a coefficient called learning rate, and $\nabla J_{\text{train}}(\mathbf{w}_t)$ is the gradient of the cost function for state \mathbf{w}_t , defined by:

$$\nabla J_{\text{train}}(\mathbf{w}_t) = \left(\frac{\partial J_{\text{train}}(\mathbf{w}_t)}{\partial w_1}, \dots, \frac{\partial J_{\text{train}}(\mathbf{w}_t)}{\partial w_n} \right)^\top$$

This method is a first-order minimization algorithm called gradient descent. The vector of parameters \mathbf{w} moves opposite the gradient of the loss function in order to find a local minimum (see **Figure N°20**).

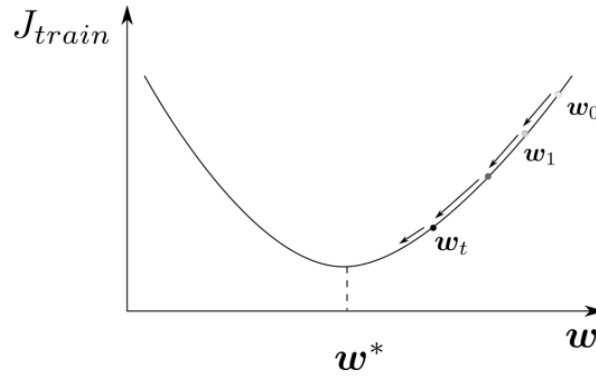


Figure 20: Example of gradient descent on one dimension

The α hyper-parameter is used to modulate the displacement step in the parameter space. The larger α , the greater the modifications made to the vector \mathbf{w} during an iteration. We will see in **Section 3.6** that this parameter is important to guarantee good convergence of learning on an interesting local minimum. The calculation of the gradient $\nabla J_{train}(\mathbf{w}_t)$ requires calculating for each weight w_j its partial derivative $\partial J_{train} / \partial w_j$:

$$\begin{aligned} \frac{\partial J_{train}}{\partial w_j} &= \frac{\partial}{\partial w_j} \left(\frac{1}{K} \sum_{k=1}^K L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \mathbf{w}) \right) \\ &= \frac{1}{K} \sum_{k=1}^K \frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \mathbf{w})}{\partial w_j} \end{aligned}$$

This partial derivative is the average of the partial derivatives of the error function L on each example in the training dataset. Using the composite function derivation theorem twice on the error function L , we can develop it in these three terms:

$$\frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \mathbf{w})}{\partial w_j} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial z} \frac{\partial z}{\partial w_j} \quad (3.2)$$

With:

$$L \equiv \frac{1}{2} (f - y)^2, \quad f \equiv \frac{1}{1 + e^{-z}}, \quad \text{and } z = \sum_{i=0}^n w_i x_i^{(k)}$$

The first term represents how much the error made for example k depends on the output of the perceptron. This derivative can be calculated as being:

$$\begin{aligned} \frac{\partial L}{\partial f} &= \frac{\partial}{\partial f} \left(\frac{1}{2} (f - \mathbf{y}^{(k)})^2 \right) \\ &= f - \mathbf{y}^{(k)} \end{aligned} \quad (3.3)$$

The second term is the derivative of the activation function, that is, the sigmoid function:

$$\begin{aligned}\frac{\partial f}{\partial z} &= \frac{\partial}{\partial z} \left(\frac{1}{1 + e^{-z}} \right) \\ &= \frac{e^{-z}}{(1 + e^{-z})^2}\end{aligned}\quad (3.4)$$

The last term represents the partial derivative of the sum of the weighted inputs w_j :

$$\frac{\partial z}{\partial w_j} = \frac{\partial}{\partial w_j} \left(\sum_{i=0}^n w_i x_i^{(k)} \right)$$

The weighted input $w_i x_i$ is the only non-constant term of this sum as a function of w_j . We therefore obtain:

$$\frac{\partial z}{\partial w_j} = x_j^{(k)} \quad (3.5)$$

From relations (3.3), (3.4) and (3.5), we obtain:

$$\frac{\partial L}{\partial w_j} = (f - y^{(k)}) \frac{e^{-z}}{(1 + e^{-z})^2} x_j^{(k)} \quad (3.6)$$

The partial derivative $\frac{\partial L}{\partial w_j}$ can be rewritten as:

$$\frac{\partial L}{\partial w_j} = \delta^{(k)} x_j^{(k)} \quad (3.7)$$

With the term $\delta^{(k)} = (f - y^{(k)})e^{-z}/(1 + e^{-z})^2$ which does not depend on j . In terms of computation, it is possible to compute $\delta^{(k)}$ only once for all the inputs j for each example k . The final partial derivative is:

$$\frac{\partial J}{\partial w_j} = \frac{1}{K} \sum_{k=1}^K \delta^{(k)} x_j^{(k)}$$

Performing an iteration of the gradient descent requires calculating the error made by the perceptron on each example in the training dataset. This error then makes it possible to calculate the partial derivative for each input and therefore to have the gradient of the vector of parameters. The gradient descent algorithm can therefore update the parameters and start a new iteration. The gradient descent continues to iterate until a sufficiently interesting parameter vector is obtained, described by a stop criterion. We will see in **section 3.5** how this stopping criterion can be defined.

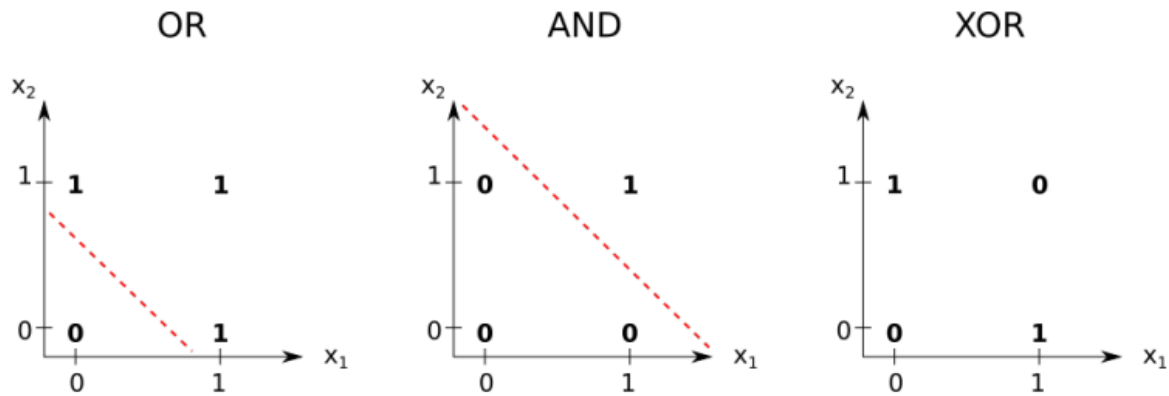


Figure N°21: Example of logic functions: the output of the OR functions and the AND functions are linearly separable while the outputs of the EXCLUSIVE OR function cannot be linearly separated. The perceptron therefore cannot represent this function. When $x_1 = 0$, the output of the perceptron must increase if x_2 increases, and if $x_1 = 1$ the output of the perceptron must decrease if x_2 increases. The first requires that $w_2 > 0$ while the second requires $w_2 < 0$.

3.2.1 - Limitations of the perceptron:

The perceptron is a so-called linear classifier, which means it classifies data from a linear combination of its inputs. It is therefore unable to classify data into non-linearly separable classes (non-separable with a hyper-plane in data space). For example, it is not possible to represent the EXCLUSIVE OR function with a perceptron (see **Figure N°21**).

3.3 - The multi-layered perceptron:

Multi-layer perceptrons, also called **MLP**, are more general neural networks than the perceptron. They are made up of a multitude of interconnected neurons organized in successive layers. An MLP can be represented by an acyclic graph in which each node represents a neuron. The oriented arcs represent the relationships between each neuron: an **arc** from node i to node j means that neuron j takes the activation value of neuron i as input. **Figure N°22** shows a graphical representation of an MLP with 3 layers having 3, 4, 2 neurons respectively.

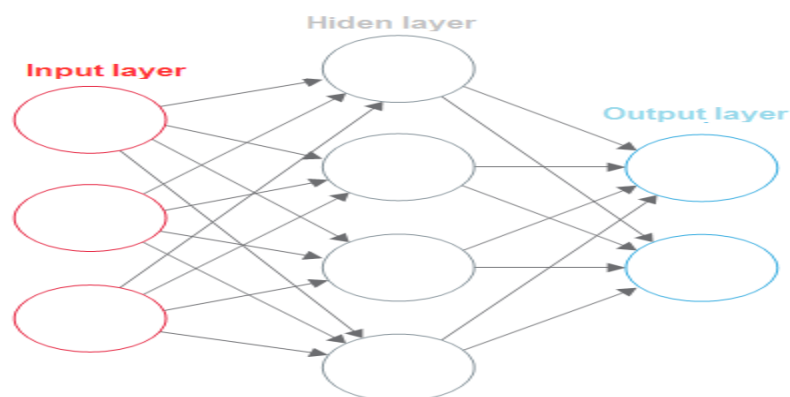


Figure N°22: Example of a representation of an MLP

As shown in **Figure N°22**, each neuron in the first layer takes as input, the input of the MLP. Each subsequent layer receives as input the activation values of the previous layer (that is, the vector containing the values of each neuron in the previous layer). The output of the MLP is made up of the activation value of each neuron in the last layer, called the output layer. The activation vector of layer l , composed of s neurons can be calculated according to the input vector $\mathbf{e} \in \mathbb{R}^p$ (MLP inputs or the activation vector of the previous layer), as follows:

$$\mathbf{a}^{(l)} = f^{(l)}(\mathbf{e}, W^{(l)}, \beta^{(l)}) = \phi^{(l)}(W^{(l)}\mathbf{e} + \beta^{(l)})$$

With $W = (\mathbf{w}_1, \dots, \mathbf{w}_s)^\top \in \mathbb{R}^{s \times p}$ and $\beta^{(l)} = (\beta_1, \dots, \beta_s)^\top \in \mathbb{R}^s$ with respectively \mathbf{w}_i the vector of the weights of the neuron i and β_i its bias. The function $\phi^{(l)}$ is an activation function applied individually to each neuron of layer l . In an MLP, all neurons in the same layer have the same activation function $\phi^{(l)}$.

The MLP is represented by a function f which takes as input data $\mathbf{x} \in \mathbb{R}^n$ and a set of parameters $\theta = \{W^{(l)}, \beta^{(l)} \mid l \in \{1, \dots, L\}\}$, corresponding to the set of matrices $W^{(l)}$ and vectors $\beta^{(l)}$ for all layers $l = 1, \dots, L$, and gives as output $\hat{y} \in \mathbb{R}^m$. As described in the equation of a perceptron (see **Eq 3.1**), it is possible to include the vector $\beta^{(l)}$ in the matrix $W^{(l)}$ by adding a constant input for each layer (which adds a column to each matrix $W^{(l)}$). The function f is a composition of functions $f^{(l)}$ associated with each layer l of the network. For example, with a three-layer MLP, we have:

$$\hat{y} = f(\mathbf{x}, \theta) = f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}, W^{(1)}), W^{(2)}), W^{(3)})$$

An MLP with a number of layers greater than or equal to 2 is a universal approximator of functions, which means it is able to represent all kinds of functions if its parameters are correctly adjusted (under certain conditions on the activation function of the hidden layers [108]). To illustrate this, we take the example of the EXCLUSIVE OR function with two inputs. Consider an MLP with two inputs and two layers, made up of 2 and 1 neurons. The parameters of the MLP are:

$$W^{(1)} = \begin{bmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{bmatrix}$$

$$W^{(2)} = [0 \quad 1 \quad -2]$$

We use rectified linear units (called ReLU) used regularly in modern ANNs such as in [98] [104]. This type of neuron uses the activation function $\phi(z) = \max\{0, z\}$ (see **section 3.3.1**). The entire MLP is shown graphically in **Figure N°23**.

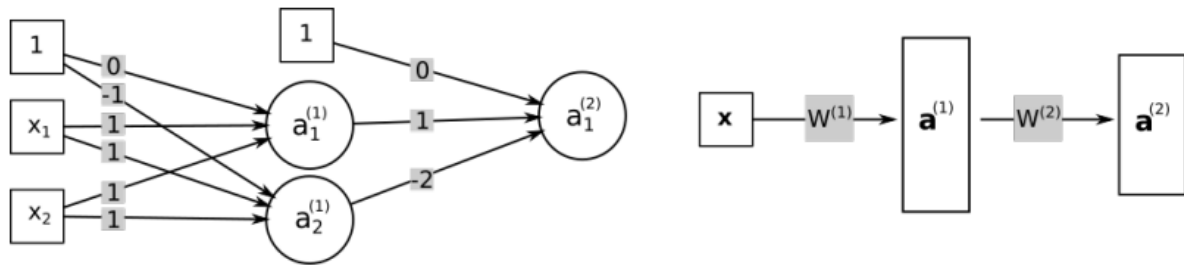


Figure N°23: Neural network representing the EXCLUSIVE OR function with two graphical representations. On the left, each neuron is represented by a circle. The weights are represented on the connections between neurons. Likewise, the biases represented by the connection between a constant (squares) and each neuron. To the right, in this graphic style, each layer is represented by a rectangle. Parameter matrices can be indicated on the connections between layers. The advantage of this second representation is that it is more compact than the first.

The middle layers of this MLP transform the representation space of the input data as shown in **Figure N°30**. This new representation space makes it possible to linearly separate the outputs of the targeted function (the EXCLUSIVE OR function in this example). Intermediate layers can be seen as representations of higher-level inputs.

3.3.1 – The Activation function:

Activation functions are functions used in ANNs to compute the weighted sum of input and biases, of which is used to decide if a neuron can be fired or not. It manipulates the presented data through some gradient processing usually gradient descent and afterwards produces an output for the neural network, which contains the parameters in the data. These AFs are often referred to as a transfer function in some literature.

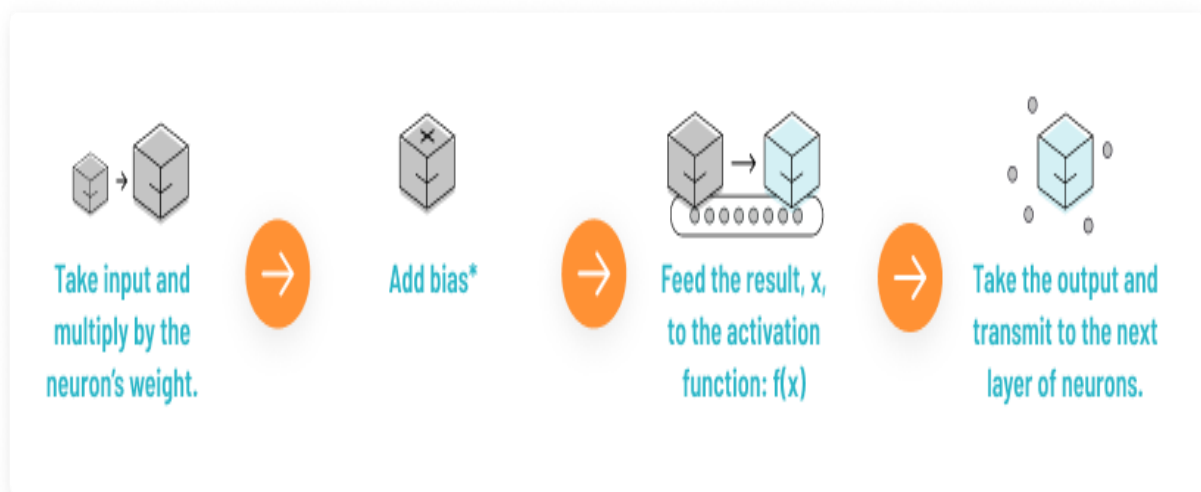


Figure N°24: The basic process carried out by a neuron in a neural network.

There are three general types of activation functions:

3.3.1.1 - Binary Step Function:

A binary step function is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer (back to our email example we used the Heaviside Step function).

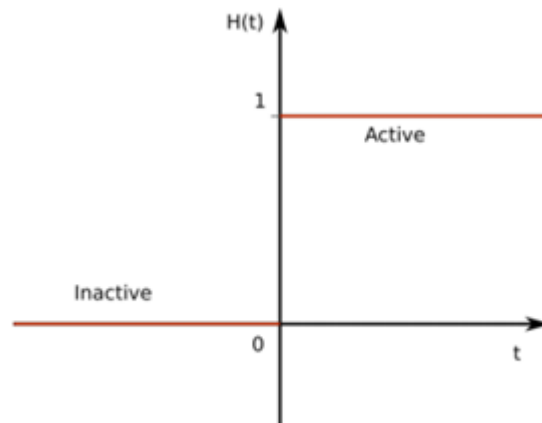


Figure N°25: The function of activating binary step.

The problem with a step function is that it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.

3.3.1.2 - Linear Activation Function:

A linear transform is basically the identity function $f(x) = cx$, It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.

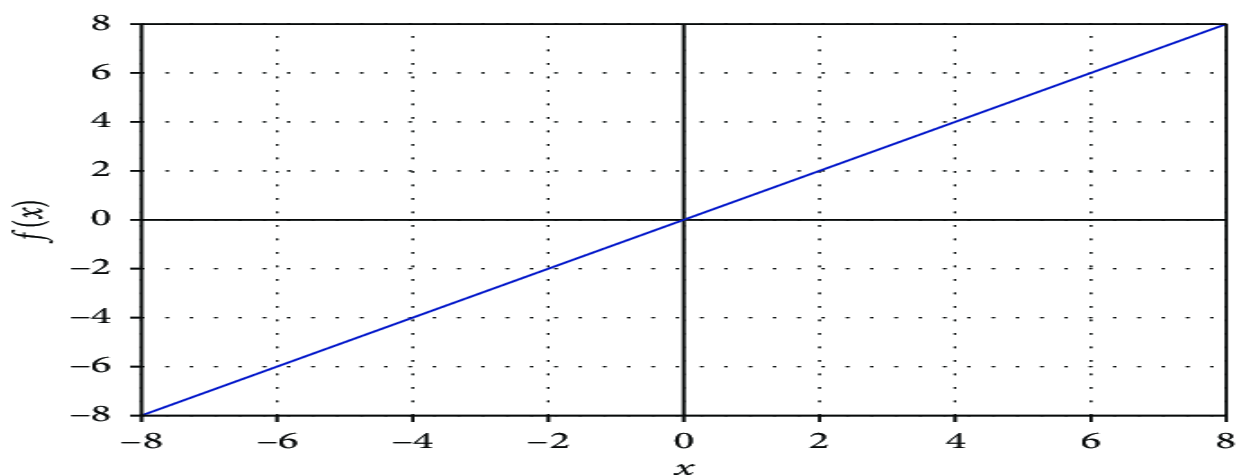


Figure N°26: Linear Activation Function.

3.3.1.3 – Non-Linear Activation Functions:

Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality. Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.

Here 4 Common Nonlinear Activation Functions:

Sigmoid / Logistic: The Sigmoid AF is sometimes referred to as the logistic function or squashing function in some literature [109]. The Sigmoid function research results have produced three variants of the sigmoid AF, which are used in DL applications. The Sigmoid is a non-linear AF used mostly in feed-forward neural networks. It is a bounded differentiable real function, defined for real input values, with positive derivatives everywhere and some degree of smoothness [110]. The Sigmoid function is given by the relationship

$$s(t) = \frac{1}{1 + e^{-t}}$$

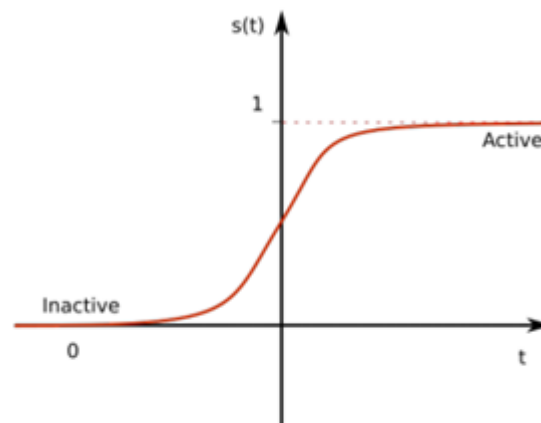


Figure N°27: The output of a sigmoid neuron as t varies

This one has an "S" shape close to the function of Heaviside. The advantage of the sigmoid function, however, is that it is derivable at all points (unlike the Heaviside function). So going back to our emails example the function defined by the perceptron with the sigmoid function is therefore the following:

$$f(\mathbf{x}, \mathbf{w}) = \frac{1}{1 + e^{-z(\mathbf{x}, \mathbf{w})}}$$

The output of the perceptron is now set to] 0, 1 [. It can be interpreted as the probability that the neuron will activate depending on the input.

Hyperbolic Tangent Function (Tanh): The hyperbolic tangent function is another type of AF used in DL and it has some variants used in DL applications. The hyperbolic tangent function known as Tanh function, is a smoother [150] zero-centered function whose range lies between -1 to 1, thus the output of the Tanh function is given by

$$\tanh(t) = \left(\frac{e^t - e^{-t}}{e^t + e^{-t}} \right)$$

The tanh function became the preferred function compared to the sigmoid function in that it gives better training performance for multi-layer neural networks. However, the tanh function could not solve the vanishing gradient problem suffered by the sigmoid functions as well. The main advantage provided by the function is that it produces zero centered output thereby aiding the back-propagation process.

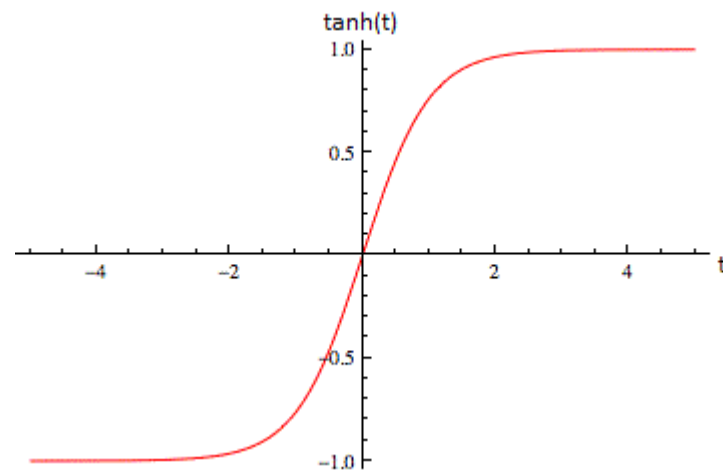


Figure N°28: The output of a Tanh neuron as t varies

Rectified Linear Unit (ReLU) Function: The rectified linear unit (ReLU) activation function was proposed by Nair and Hinton 2010, and ever since, has been the most widely used activation function for deep learning applications with state-of-the-art results to date [152]. The ReLU is a faster learning AF [150], which has proved to be the most successful and widely used function [151]. It offers the better performance and generalization in deep learning compared to the Sigmoid and tanh activation functions [153]. The ReLU represents a nearly linear function and therefore preserves the properties of linear models that made them easy to optimize, with gradient-descent methods [92]. The ReLU activation function performs a threshold operation to each input element where values less than zero are set to zero thus the ReLU is given by

$$\varphi(z) = \max(0, z) = \begin{cases} z_i, & \text{if } z_i \geq 0 \\ 0, & \text{if } z_i < 0 \end{cases}$$

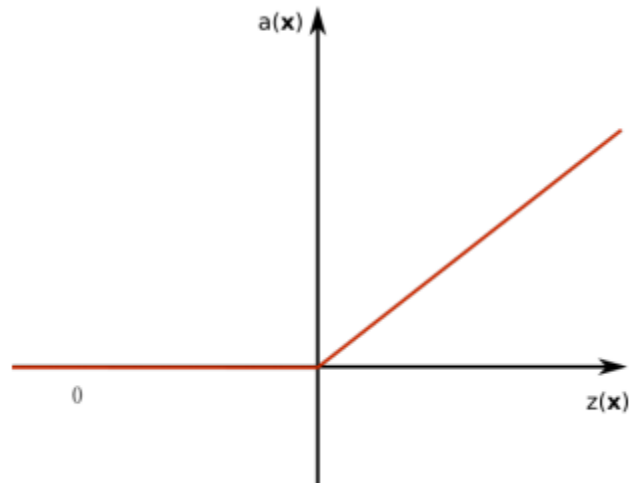


Figure N°29: The output of a ReLU neuron as z varies.

Softmax Function: The Softmax function is another type of activation function used in neural computing. It is used to compute probability distribution from a vector of real numbers. The Softmax function produces an output which is a range of values between 0 and 1, with the sum of the probabilities been equal to 1. The Softmax function [5] is computed using the relationship

$$f(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^k \exp(x_j)}$$

Where x The input vector to the Softmax function, made up of (x_0, \dots, x_k) , k is the number of classes in the multi-class classifier, and the term on the bottom of the formula $\sum_{j=1}^k \exp(x_j)$ is the normalization term. It ensures that all the output values of the function will sum to 1 and each be in the range $(0, 1)$, thus constituting a valid probability distribution.

Likewise, there are more activation functions such as [154]:

- Hard Sigmoid Function.
- Sigmoid-Weighted Linear Units (SiLU).
- Derivative of Sigmoid-Weighted Linear Units (dSiLU).
- Hard Hyperbolic Function.
- Leaky ReLU (LReLU).
- Parametric Rectified Linear Units (PReLU).
- Randomized Leaky ReLU (RRReLU).
- S-shaped ReLU (SReLU).
- Exponential Linear Units (ELUs).
- Scaled Exponential Linear Units (SELU).
- The Exponential Linear Squashing (ELiSH).
- Softplus Function.
- Softsig.

3.4 – Learning with back propagation:

Now that we have defined the MLP model, we will see that it allows us to approximate a function $g: \mathbb{R}^n \rightarrow \mathbb{R}^m$ with a set of parameters θ . A manual adjustment of the parameters θ in order to find θ^* , such that $f(\mathbf{x}, \theta^*) \approx g(\mathbf{x})$.

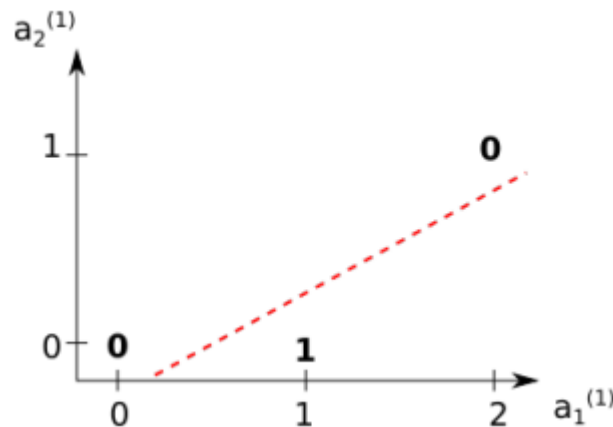


Figure N°30: Intermediate representation of the entry into layer 1. In this representation, the classes are linearly separable unlike the original representation of the data shown in **Figure N°21**. This is why the 2-layer MLP is able to represent the EXCLUSIVE OR function (unlike the perceptron).

For all x expert knowledge of the function g . In addition, the size of θ is of the order of $O(n^2)$, with n the number of neurons in the MLP. This represents up to a few million values in modern MLPs. It is therefore necessary to use an automatic optimization method to get closer to θ^* .

3.4.1 – Gradient descent:

MLP is a model suitable for supervised learning. We will see in **Section 4.5** how neural networks can perform unsupervised tasks. In this section, we will show how to teach an MLP a regression task, using the gradient descent (GD) and back-propagation method. Let be an MLP which must approximate an unknown function $g: \mathbb{R}^n \rightarrow \mathbb{R}^m$ and f the function defined by this MLP. To evaluate the ability of $f(\cdot, \theta)$ to approximate the target function g , we introduce the following cost function:

$$J^*(\theta) = E[L(\mathbf{x}, \mathbf{y}, \theta)] = \int L(\mathbf{x}, \mathbf{y}, \theta) p(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y}$$

With L the errors function of the example (x, y) . A commonly used error function, which we will take in this example, is the least square function, defined as follows:

$$L(\mathbf{x}, \mathbf{y}, \theta) = \frac{1}{2} \sum_{i=1}^m (f(x_i, \theta) - y_i)^2$$

The training consists in minimizing the cost function J^* , which is, minimizing the error expectation of f given θ . To do this, we use the empirical cost function on a D_{train} training set composed of the doublet $(\mathbf{x}^{(k)}, \mathbf{y}^{(k)})$, such that $g(\mathbf{x}^{(k)}) = \mathbf{y}^{(k)}$. The empirical cost function is defined as the mean error of the MLP for the examples in the learning dataset:

$$J(\theta) = \frac{1}{|D_{\text{train}}|} \sum_{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \in D_{\text{train}}} L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta)$$

The goal is to find θ^* which minimizes the empirical cost function. This minimization is done by the method called gradient descent, which we saw in the perceptron learning algorithm (see **Section 3.2**). The following update rule is applied to each iteration for each of the $W^{(l)}$ matrices:

$$W_{t+1}^{(l)} = W_t^{(l)} + \alpha \Delta W_t^{(l)}$$

With:

$$\Delta W_t^{(l)} = \begin{bmatrix} \frac{\partial L(\theta)}{\partial w_{11}^{(l)}} & \dots & \frac{\partial L(\theta)}{\partial w_{1p}^{(l)}} \\ \vdots & \vdots & \vdots \\ \frac{\partial L(\theta)}{\partial w_{s1}^{(l)}} & \dots & \frac{\partial L(\theta)}{\partial w_{sp}^{(l)}} \end{bmatrix}$$

3.4.2 – Back propagation method:

As with the perceptron, the partial derivative of the cost function can be calculated as the average of the partial derivative of the error function on each example of D_{train} .

$$\begin{aligned} \frac{\partial J}{\partial w_{ij}^{(l)}} &= \frac{\partial}{\partial w_{ij}^{(l)}} \left(\frac{1}{|D_{\text{train}}|} \sum_{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \in D_{\text{train}}} L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta) \right) \\ &= \frac{1}{K} \sum_{(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}) \in D_{\text{train}}} \frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta)}{\partial w_{ij}^{(l)}} \end{aligned}$$

The back propagation process is used to calculate the partial derivative $\partial L^{(k)} / \partial w_{ij}^{(l)}$ for all $w_{ij}^{(l)}$ given an input $\mathbf{x}^{(k)}$ and the associated label $\mathbf{y}^{(k)}$. The back-propagation is calculated in 2 stages:

- **Forward propagation:** the activation value $\mathbf{a}^{(l)}$ is calculated for each layer l , from the first hidden layer to the output layer, based on the input $\mathbf{x}^{(k)}$ of the MLP.
- **Back propagation:** the error term for each neuron is calculated from the output layer to the first layer, comparing the output $\hat{\mathbf{y}}^{(k)}$ of the MLP (i.e., the activation $\mathbf{a}^{(d)}$ of the output layer for input $\mathbf{x}^{(k)}$ with the expected output $\mathbf{y}^{(k)}$).

The error term is calculated as follows:

$$\delta_j^{(l)} = \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} = \begin{cases} (a_j^{(l)} - y_j) \phi_j^{(l)'}(z_j^{(l)}) & \text{if } l \text{ is an output layer,} \\ \left(\sum_{p=1} \delta_p^{(l+1)} w_{jp}^{(l)} \right) \phi_j^{(l)'}(z_j^{(l)}) & \text{if } l \text{ is a hidden layer} \end{cases}$$

With:

$$L \equiv \frac{1}{2} \sum_{o=1}^m (a_o^{(d)} - y_o)^2, \quad a_j^{(l)} \equiv \phi_j(z_j^{(l)}), \text{ and } z_j^{(l)} \equiv \sum_{p=0}^u w_{pj} a_p^{(l-1)}$$

Given the activation $a_i^{(l-1)}$ of the i^{th} neuron of layer $l - 1$ and the error term $\delta_i^{(l)}$ of the j^{th} neuron of layer l , it is possible to calculate the partial derivative $\partial L^{(k)} / \partial w_{ij}^{(l)}$:

$$\frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta)}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

Proof: The proof is comparable to the description of the gradient descent of the perceptron. We use the compound function derivative theorem twice to get:

$$\frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta)}{\partial w_{ij}^{(l)}} = \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial w_{ij}^{(l)}}$$

The last term represents the partial derivative of the sum of the weighted inputs w_{ij} :

$$\frac{\partial z}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \left(\sum_{p=1}^u w_{pj} a_p^{(l-1)} \right)$$

The weighted input $w_{pj} a_p^{(l)}$ is the only term of the non-constant sum as a function of w_{ij} . We therefore obtain:

$$\frac{\partial z}{\partial w_{ij}} = a_i^{(l-1)} \quad (3.8)$$

Let $\delta_j^{(l)} = \frac{\partial L}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial z_j^{(l)}}$. If j is a neuron of the last layer, that is, $l = d$

$$\begin{aligned} \delta_j^{(l)} &= \frac{\partial}{\partial a_j^{(l)}} \left(\frac{1}{2} \sum_{p=0}^m (a_p^{(l)} - y_o^{(k)})^2 \right) \frac{\partial}{\partial z_j^{(l)}} \left(\phi_j^{(l)}(z_j^{(l)}) \right) \\ &= (a_j^{(l)} - y_j) \phi_j^{(l)'}(z_j^{(l)}) \end{aligned} \quad (3.9)$$

Otherwise, j belongs to a hidden layer (i.e., $1 \leq l < d$), and we can expand $\delta_j^{(l)}$:

$$\begin{aligned} \delta_j^{(l)} &= \sum_{t=1} \frac{\partial L}{\partial a_t^{(l+1)}} \frac{\partial a_t^{(l+1)}}{\partial a_j^{(l)}} \phi'(z^{(l)}) \\ &= \sum_{t=1} \frac{\partial L}{\partial a_t^{(l+1)}} \frac{\partial a_t^{(l+1)}}{\partial z_t^{(l)}} \frac{\partial z_t^{(l+1)}}{\partial a_j^{(l)}} \phi'(z^{(l)}) \end{aligned}$$

The term $\left(\frac{\partial L}{\partial a_t^{(l+1)}} \frac{\partial a_t^{(l+1)}}{\partial z_t^{(l)}}\right)$ is the error term $\delta_t^{(l+1)}$ for the neuron t of the layer $l + 1$. So, we have:

$$\delta_j^{(l)} = \sum_{t=1} \delta_t^{(l+1)} w_{jt} \phi'(z^{(l)}) \quad (3.10)$$

From **Eq: 3.8, 3.9** and **3.10**, we have:

$$\frac{\partial L(\mathbf{x}^{(k)}, \mathbf{y}^{(k)}, \theta)}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

With:

$$\delta_j^{(l)} = \begin{cases} (a_j^{(l)} - y_j) \phi'(z^{(l)}) & \text{si } l = d \\ \sum_{t=1} \delta_t^{(l+1)} w_{jt} \phi'(z^{(l)}) & \text{sinon.} \end{cases}$$

3.5 – Convergence of learning:

Gradient descent learning optimizes neural network parameters with respect to the empirical error function J_{train} . Due to the model of neural networks, this function is generally non-convex. That is, it contains several local minimums. In practice, it is not necessary to reach an overall minimum on the J_{train} error function because this generally leads to a case of over-learning, as we will see in the next paragraph. The learning rate α is an important parameter to take into account. Too small, the learning is slow and the risk of falling into an uninteresting local minimum is high, and too large, the search for the parameters is likely to diverge [111] [112]. We will see in **Section 3.6** methods for automatically adapting the learning rate during gradient descent.

3.5.1 – Problem of over-fitting:

A classic problem in machine learning is the over-fitting (**over-learning**) of the training set. This problem occurs when the learned model begins to adapt to the particular cases of the dataset to the detriment of the general case. This scenario is repeated in **Figure N°31**. This phenomenon stems from an insufficiently large learning dataset compared to the complexity of the learning model. Since ANNs are very complex models with a particularly large number of parameters, their training requires special attention to this phenomenon of over-training, in particular in the case where the training dataset is relatively small.

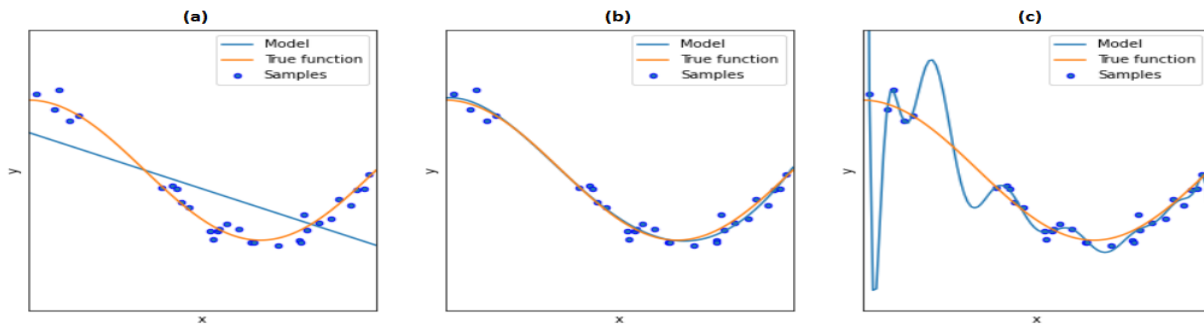


Figure N°31²⁴: Three models of classifiers at different learning levels: a) under-learning classifier, b) well-learned classifier, and c) over-learning classifier. The purple and orange dots are the data for the two different classes.

3.5.2 – Set of validation and cross-validation:

In machine learning models, such as ANNs, the phenomenon of over-learning can be detected by the use of a data set not used during training, called a validation set (approximately between 10% and 50% of the learning set). During training, the loss function is calculated on the validation set on a regular basis as a simple observation (it is not used for the calculation of the gradient). Over-learning is observed when the loss function begins to move up on the validation set as it continues to decline on the learning set (see **Figure 30**) This breaking point marks the model's specialization on learning data at the expense of the model's generalizability (i.e., the ability to process new data).

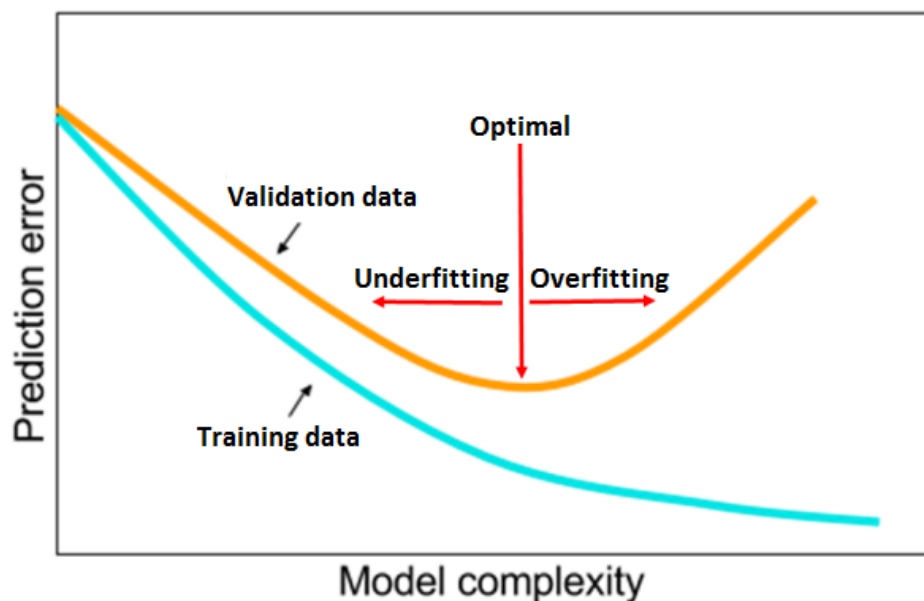


Figure N°32: Loss function calculated on the D_{train} training set (curve in blue) and on the validation set D_{valid} (curve in orange) according to the model complexity. When the loss function on D_{valid} begins to rise while on D_{train} continues to descend, the model enters an over-fitting phase.

²⁴ This picture was taken from <https://datascience.foundation/sciencewhitepaper> accessed 09 October 2020

3.5.3 – Early-stopping:

The simplest method to avoid over-learning is to stop the gradient descent when the loss function calculated on the validation set starts to increase (starts going up) and the loss function calculated on the training set continues to increase (keeps going down). Methods have been proposed to automatically detect this breaking point during training [113].

3.5.4 – Regularization:

Since over-fitting is due to the learning of a model that is too complex compared to the initial problem, techniques to avoid it consist in penalizing this complexity of the model. For neural networks, this consists in penalizing excessively large connection weights by adding a regularization term to the cost function J to be minimized. This term penalizes the weights of too strong connections. Another regularization solution, called Dropout²⁵, has been proposed specifically for neural networks [114]. It consists in "disconnecting" neurons taken at random, at each iteration, on a temporary basis. These neurons therefore do not participate in the output of the neural network during an iteration of learning. The other neurons must therefore compensate for this absence. This has the effect of making the neural network more robust against noise and thus avoiding over-fitting.

Learning the other neurons must therefore compensate for this absence. This has the effect of making the neural network more robust against noise and thus avoiding over-fitting.

3.6 – Alternatives to gradient descent:

3.6.1 – Stochastic Gradient Descent:

Gradient descent is a very popular optimization algorithm, but there are many variations that are used to train neural networks. An existing variant is the stochastic version of gradient descent which we will call stochastic gradient descent. Let J be the cost function to be minimized as a function of a vector of parameters \mathbf{w} such that:

$$J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^m L_i(\mathbf{w})$$

With L_i is the i^{th} observation made on the learning set (typically the error made by the i^{th} data in the learning set). Instead of calculating the gradient of $J(\mathbf{w})$ over the set of observations L_i to modify the parameters \mathbf{W} , this gradient is approximated by calculating it on a single observation L_i . At each iteration, the parameters \mathbf{W} are modified as follows:

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \alpha \nabla L_i(\mathbf{W}_t)$$

This method requires calculating the gradient just for a single entry of the training set, greatly reducing the cost per iteration. It is particularly useful when the dataset is not fully accessible or too large to be stored.

²⁵ Dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random.

In addition, the stochastic gradient descent allows, through shorter iterations, to approach the optimal solution of \mathbf{w} more quickly. However, this approximation of the gradient on a single observation implies doing a larger number of iterations with a smaller learning rate.

A good compromise is to take, not a single observation, but a batch (that is, between a few tens or hundreds of observations) of a size b . The gradient is then better approximated, which makes it possible to use a more reasonable learning rate to quickly converge.

3.6.2 – Momentum:

It is still possible to accelerate the gradient descent by the momentum method [158]. The name of this method comes from the field of physics, where momentum represents inertia²⁶ in the movement of an object. The idea is to keep the inertia also in the gradient search by keeping track of the last modifications to calculate the displacement of the parameters of the function. This method is generally associated with gradient descent with small batches. The update rule is then the following:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \alpha \Delta \mathbf{w}_t$$

With:

$$\Delta \mathbf{w}_t = \gamma \Delta \mathbf{w}_{t-1} + \frac{1}{b} \sum_{i \in X_t} \nabla L_i(\mathbf{w}_t)$$

With X_t the data batch used in iteration t , and $\gamma \in]0,1]$. The closer γ is to 1, the more momentum the search in parameter space has.

3.6.3 – Nesterov momentum:

A variant of the momentum method is called the Nesterov momentum method proposed in the work of Sutskever et al. [157] to improve the learning of neural networks. Instead of calculating the gradient for $\tilde{\mathbf{w}}_t$ at iteration t , the authors propose to calculate the latter at point \mathbf{w}_t , which corresponds to the current parameters θ_t plus the momentum, that is to say such that:

$$\tilde{\mathbf{w}}_t = \mathbf{w}_t + \gamma \Delta \mathbf{w}_{t-1}$$

The modification to iteration t is therefore equal to:

$$\Delta \mathbf{w}_t = \beta \Delta \mathbf{w}_{t-1} + \frac{1}{b} \sum_{i \in X_t} \nabla L_i(\tilde{\mathbf{w}}_t)$$

²⁶ A tendency to do nothing or to remain unchanged.

3.6.4 – Second-order methods:

Gradient descent is a so-called first-order method, which means it optimizes the parameters of a function using its first derivative. There are other optimization methods such as Newton's method which uses the second derivative of the function to be minimized in order to find an extremum²⁷.

The calculation of the Hessian²⁸ or its approximation requires calculating the matrix of the partial derivatives second order which corresponds to a matrix of size $n \times n$ at each iteration (with n the number of parameters of the neural network). In the case of modern neural networks, we recall that n can be of the order of 10^6 see 10^7 . These techniques are efficient but require approximating the Hessian by different methods in order to be able to operate on classical neural networks. Currently, they are rarely used in neural network training because other methods have been shown to be equally effective.

3.6.5 – Other optimization techniques:

In order to accelerate the gradient descent, many other methods have been proposed for neural networks such as AdaGrad [116], Adadelta [118], RMSProp²⁹, or Adam [117]. One of the most widely used is the Adam optimization method [117]. This method is based on gradient descent with small batches and uses the same principles as AdaDelta. The idea is to adapt the learning rate from an estimate of the first and second moment of the gradient (unlike the momentum method which only uses the first moment). The estimation of the first and second moment, however, requires maintaining and updating two additional variables for each parameter of the neural network. This type of method has the advantage of being relatively robust and makes it possible to automatically adapt the learning rate during learning for each weight.

4 – Deep Artificial neural networks:

In **Section 2.2** we saw how it was possible to learn models such as neural networks to perform certain tasks. However, neural networks have for a very long time been limited in their architectures, especially with regard to their depth, which means the number of layers that they can learn. This limitation collapsed in the years 2010-2012 with the arrival of much larger datasets (such as [119]) accompanied by larger computing and storage capacities. This advance has also been made possible by different neural network architectures, which are easier to learn and better suited to certain types of data.

²⁷ The maximum or minimum value of a function.

²⁸ The elements of the Hessian matrix consist of the second derivatives of the error measure with respect to the weights and thresholds in the network [138].

²⁹ This method proposed by G. Hilton has not been published but a description is available on the page: http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides lec6.pdf Accessed 10 October 2020

4.1 – The interest of deep architectures:

In classical learning algorithms, characteristics must be extracted from the raw data in order to perform the learning task. The goal is to have a higher-level representation of the data. For example, in the field of image analysis, a first step consists in calculating the points of interest (such as SIFT [120]) and grouping them into bags of words to train a classical learning model such as a decision tree, an SVM [115], a forest of random trees or even a neural network.

Extracting features from raw data requires good knowledge of this data and the learning task, as well as engineering work to adapt the extraction methods. This operation is relatively expensive to set up, depends on the context and a bad extraction of the characteristics leads to very poor performance in terms of learning. The idea of deep architectures is to integrate this feature extraction, normally done "by hand", through a learning process in the first layers of the neural network (see **Figure N°33**).

In **Section 3.3**, we saw that intermediate layers makes it possible to transform the representation of the input data into a higher-level representation. During the learning phase, each layer of an MLP learns a representation of its input that should be of interest to subsequent layers. The information in each of these layers will get higher and higher.

The term deep therefore refers to the number of layers of deep neural networks between the input and the output layer. A network with only one hidden layer is called a shallow network, and conversely, a network with more than 2 hidden layers is called deep. Nowadays, it is possible to find networks with a hundred, or even a thousand layers for the deepest [104] [103].

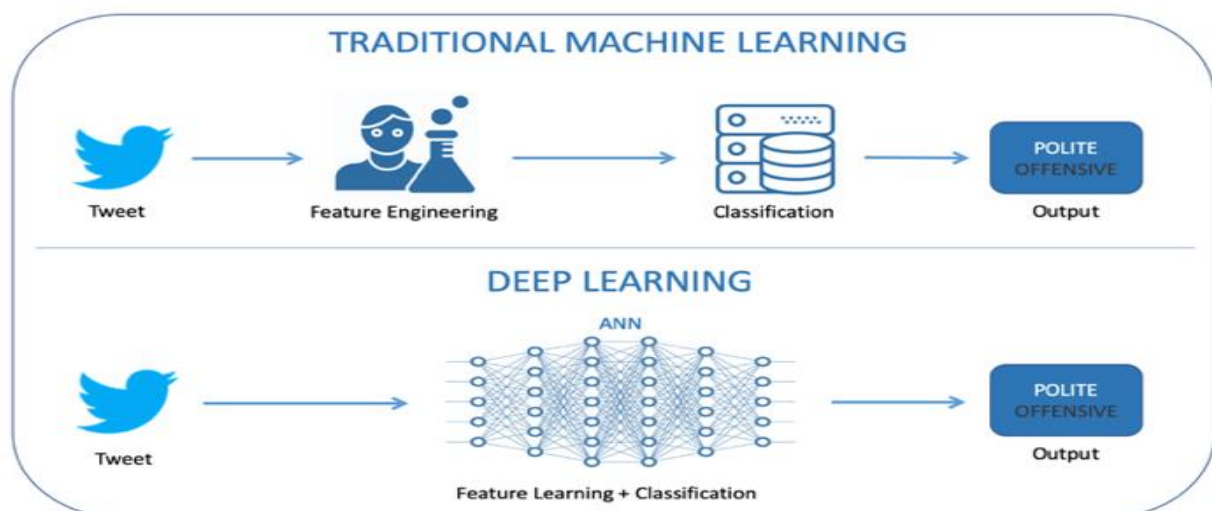


Figure N°33³⁰: The difference between classic machine learning (up) and deep learning (down)

³⁰ This picture was taken from(<https://thenewstack.io/demystifying-deep-learning-and-artificial-intelligence/>) accessed 10 October 2020

4.2 – Convolutional neural networks:

Convolutional neural networks (CNNs) were introduced by Lecun et al. [121]. The particularity of CNNs is the use of the convolution operation in the first intermediate layers of the neural network. Originally, this operation was used as a filter in the field of image or sound in order to highlight patterns or reduce a type of noise.

In CNNs, the model itself learns the filters of the different convolutions in order to highlight the patterns of the input data that are used in subsequent layers. A classic CNN is generally made up of four types of layers:

- The convolutional layers, which contain several convolution operations applied to the same input,
- The layers of pooling operations,
- The activation layers,
- The fully connected layers.

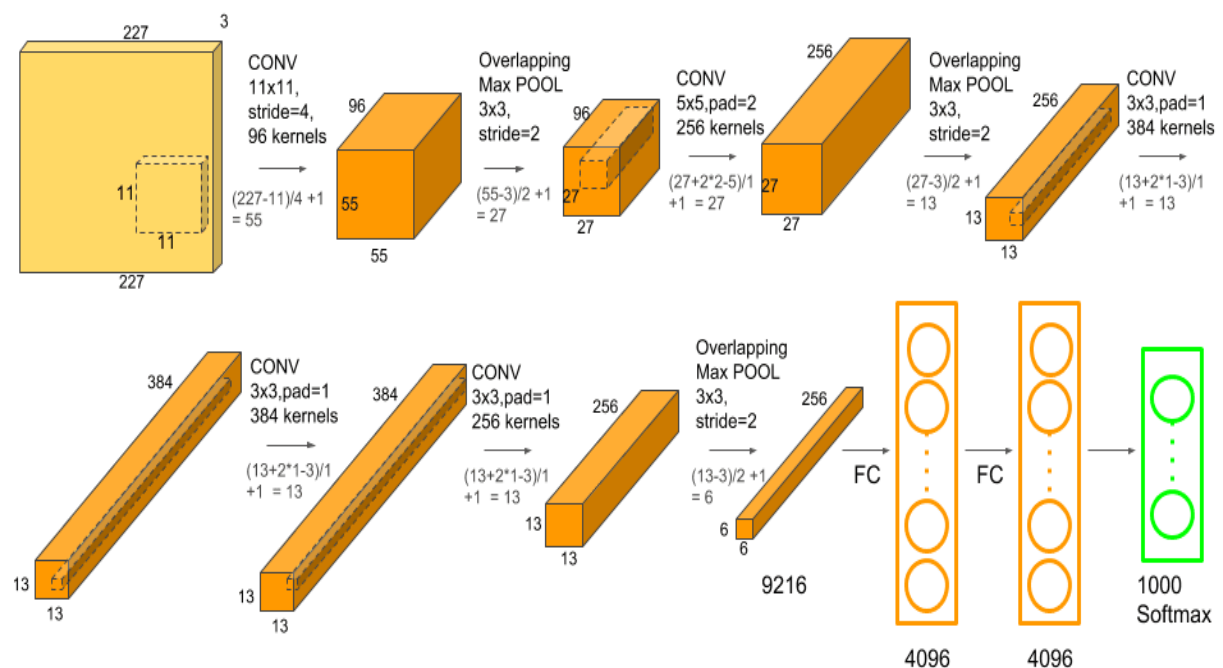


Figure N°34³¹: Example from CNN called AlexNet [63]. In this representation, neurons are organized according to the dimensions of width, height and depth. Unlike fully connected layers, convolutional layers keep information spatially coherent. Each dimension size is shown in the figure. As input, the neural network takes an image of 227 × 227 pixels with 3 color channels.

³¹ This picture was taken from (<https://neurohive.io/en/popular-networks/alexnet-imagenet-classification-with-deep-convolutional-neural-networks/>) accessed 10 October 2020

4.2.1 - Convolutional layers:

Originally, the convolution operation is used on temporal (sound) or spatial (images) data as a linear filter. In this section, we will take the example of a 2D convolution operation, used on data such as images $X = (x_{i,j,z})_{1 \leq i \leq h, 1 \leq j \leq l, 1 \leq z \leq c}$ (with $h \times l$, the dimensions of the image and c the number of channels). This operation is defined by a kernel $A = (a_{i,j,z,k})_{1 \leq i \leq m, 1 \leq j \leq n, 1 \leq z \leq c, 1 \leq k \leq f}$ where $m \times n$ is the width and the height filter and f is the number of filters, as well as a bias $\beta \in \mathbb{R}^f$.

The output of the convolution operation $Y \in \mathbb{R}^{h \times l \times f}$ is calculated as:

$$y_{i,j,k} = \sum_{i'=0}^m \sum_{j'=0}^n \sum_{z=0}^c x_{i+i',j+j',z} a_{i',j',z,k} + \beta_k$$

In order to simplify the formula, we have not taken into account the handling of the "edges" of the input image in the convolution operation. Convolutions are also applicable on one-dimensional (like sound [107]) or three-dimensional (like video or 3D scanner) data.

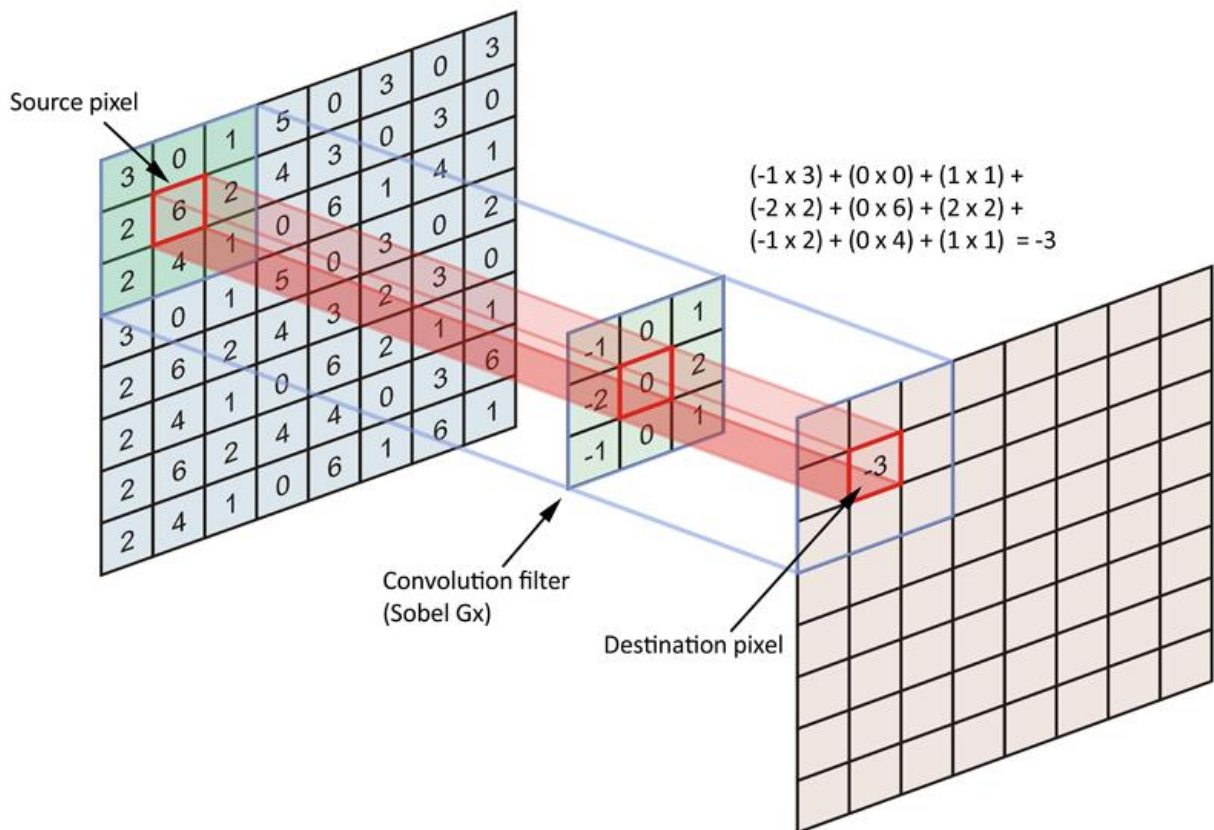


Figure N°35: example of a convolution operation using sobel ³² Filter

³² The Sobel filter is an operator used in image processing for edge detection. This is one of the simplest operators which, however, gives correct results.

4.2.2 – Pooling Operation:

Convolutional layers can be followed by a pooling operation. It aims to reduce the dimension of the layers of neurons by grouping together the information present on neurons close to each other. The principle is to move a sliding window over the neurons and apply a pooling operation to it. There are different types of pooling operations such as maximum or average function. An example of a pooling operation is shown in **Figure N°36**.

The fully connected layers and the activation layers are identical to MLPs (respectively the classic neural layers and the activation function applied to an entire layer). Fully connected layers are usually placed at the end of CNNs (just before the output layer). They make it possible to correlate all the patterns detected by the convolutional layers in the previous layers. Activation layers are usually placed after each convolutional layer and each fully connected layer. Activation layers and pooling layers are not neural layers because they contain no connections to learn (which means no trainable parameters).

Unlike MLP, the number of parameters to learn in CNNs is generally lower, but the number of operations remains higher. This is because the filters, generally small in size, are shared by neurons of one or more dimensions of the output layer. CNNs are mainly used in the field of imaging, where they overtake other learning methods [98] [104]. They are also used in the field of sound [107] or video. Variants also exist for the analysis of arbitrary graphs [122].

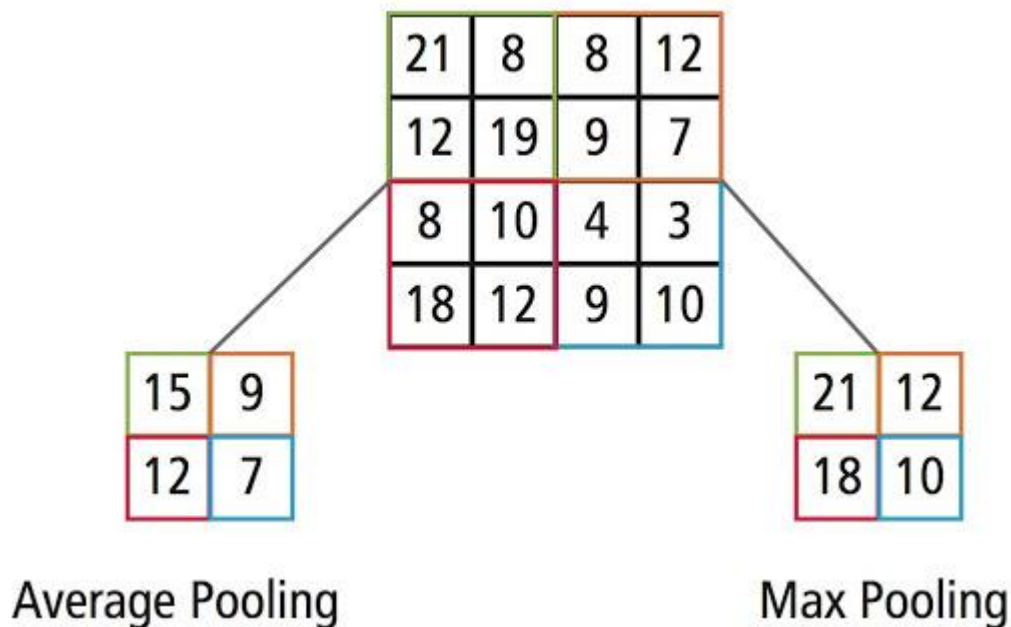


Figure N°36: Example of a pooling operation with the maximum pooling (right) and average pooling (left). The sliding window is 2×2 in size and moves 2 by 2 along the y axis and 2 by 2 along the x axis.

4.3 – Recurrent neural networks:

While CNNs are primarily used to bring out spatially close relationships (such as relationships between close pixels in an image), Recurrent Neural Networks (RNNs) were developed to keep temporal context for each input event. They have been particularly used for the analysis of time series, audio data, or text where context is important in order to analyze each new entry. The idea is to keep information over time inside the layers of neurons to give context to the data being analyzed. The output of the RNN at time t will depend not only on the input at time t but also on the state of the RNN calculated at time $t - 1$.

In its simplest version, a layer of an RNN can be described as a fully connected layer l which takes as input the previous layer $l - 1$ at time t concatenated at the output of itself (which means the layer l) at time $t - 1$. **Figure 35** shows an RNN layer.

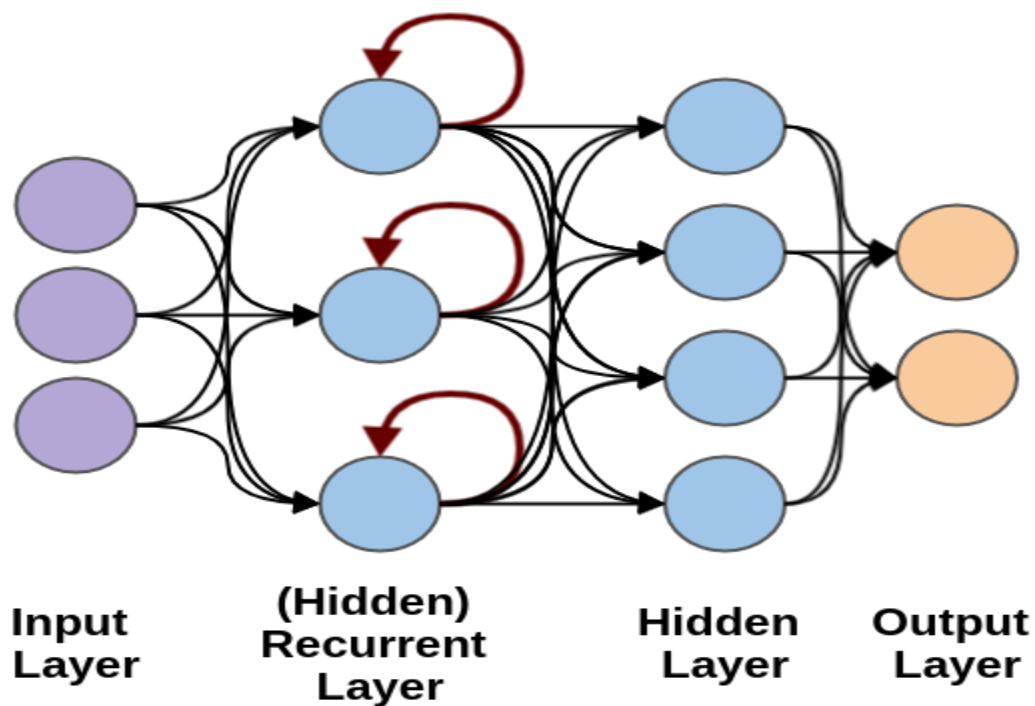


Figure N°37: RNN layer.

4.4 – Advanced techniques to improve learning:

The latest neural networks, especially in the field of imaging, have an extremely large number of layers (up to a hundred layers for some network [103, 104]). This has the effect of considerably reducing the gradient calculated in the lower layers of the network. To answer this problem, many solutions have been proposed in recent years.

4.4.1 – Inception module (GoogleNet):

Proposed in 2014 by C. Szegedy et al. [104], the neural network called GoogleNet won the ILSVRC³³(Image Classification Challenge on the ImageNet Dataset) challenge that same year by proposing multiple improvements to the CNN architecture. Most notable is the use of branched modules called an Inception module.

The basic idea is to multiply filters with different sizes. In **Figure N°38 (a)**, the module contains 4 branches, 3 layers of convolutions with filters of size 5×5 , 3×3 and 1×1 , and a pooling operation with windows of size 3×3 . The 1×1 convolution operations consist in relating only the different channels on the same position of the image. In order to reduce the amount of computation and the number of parameters to learn, the authors propose to add inexpensive 1×1 convolutions to reduce the number of channels before the 3×3 and 5×5 convolution operations. This version of the Inception modulus with dimension reduction is shown in **Figure N°38 (b)**.

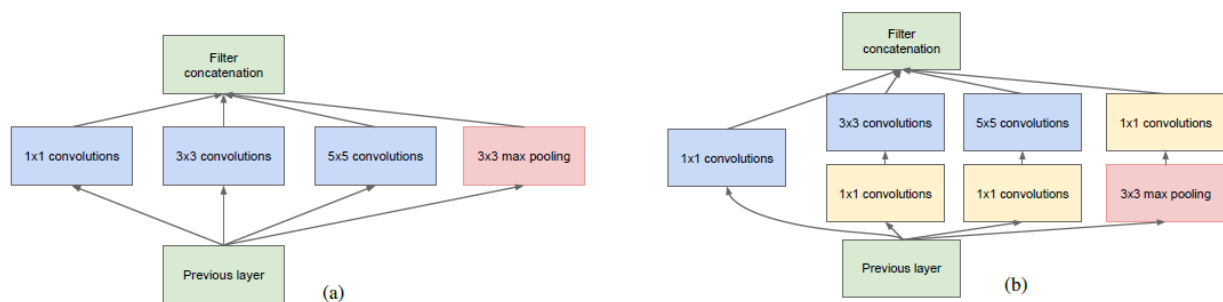


Figure N°38: Inception module in its simple version (a) and with dimension reduction (b). In this graphic representation, the Conv 3x3 block represents a convolution layer with filters of size 3 by 3. Max-pooling represents a pooling operation with the maximum function.

4.4.2 – Batch normalization:

The batch normalization technique was proposed in 2015 by S. Ioffe et al. [123]. The goal is to get around the problem of learning successions of layers dependent on each other. When a layer l is modified during training, the intermediate representation of the input data of the $l + 1$ layer is found to be modified and therefore it must re-learn its parameters according to this new representation. One solution to reduce the disturbances due to this change of intermediate representation at the input of a layer consists in normalizing the activation of each neuron of a layer following the activation of this one over a whole batch. Once neuron activation is normalized, it is "denormalized" using two variables μ_i, β_i for each neuron, which the network must additionally learn.

This denormalization is necessary to keep the representation capacity of the neural network. This method has the effect of making learning more stable with respect to the initialization of the deep neural network and the chosen learning rate (the latter may therefore be higher). Batch normalization is now used in much of state-of-the-art neural networks.

³³ A competition to evaluate algorithms for object detection and image classification at the ImageNet dataset.

4.4.3 – ResNet:

To reduce the disappearance of the gradient over a large number of layers K . Kaiming He and al. [103] propose to modify the output of certain blocks of layers of the neural network to obtain:

$$\mathbf{y} = f(\mathbf{x}) + \mathbf{x}$$

With \mathbf{x} , the input to the block, $f(\mathbf{x})$ the useful function, (which means the one used to complete the task). The idea of including the identity in the output function makes it possible not to lose information on the input data, over the layers of the neural network. This modification is made using a type of block called Residual block (used in neural networks called *Residual Neural Network*) depicted in **Figure 39**. The idea is therefore to bypass the learning operation by adding the input to the output of the block (usually followed by an activation function). The authors have shown that they are able to train artificial neural networks with up to 1,202 hidden layers using this method.

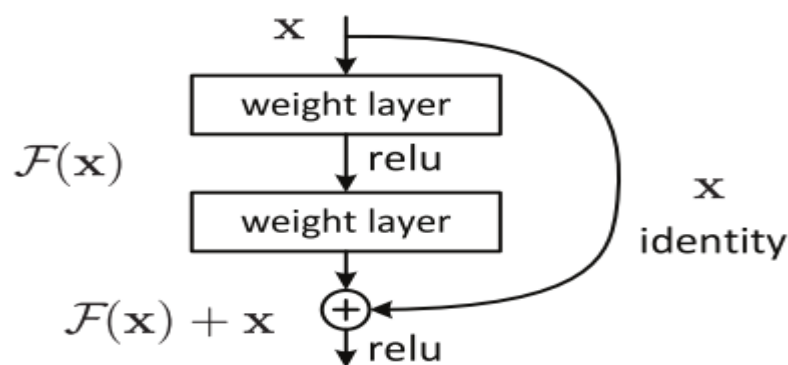


Figure N°39: Example of a classic ResBlock with two intermediate layers. The function $f(x)$ is represented by the two layers of neurons above.

4.5 – Unsupervised neural networks:

We saw at the beginning of the chapter that when no label there is available in the D_{train} learning dataset, it is not possible to use the so-called supervised learning methods seen so far. In this section, we will look at neural network architectures suitable for unsupervised learning.

4.5.1 – Auto-encoders:

Auto-encoders are artificial neural networks separated into two parts: an encoder and a decoder (see **Figure N°40**). The goal is to learn how to reduce the number of dimensions in an interesting way by encoding. The encoder is a neural network that transforms the input data $x \in \mathbb{R}^n$ into a new space \mathbb{R}^m with $n > m$. Conversely, the decoder transforms the data x from the \mathbb{R}^m space back to the \mathbb{R}^n space.

The learning of these two neural networks is done simultaneously. The goal is to minimize the decoder "reconstruction" error on the data encoded by the encoder:

$$J = \|x - f_{\theta}(g_{\theta}(x))\|^2$$

With $g_{\theta}(x)$, the parametric function representing the encoder with its parameters θ and $f_{\theta}(x)$ the parametric function representing the decoder with its parameters θ .

Since the encoding (i.e. the intermediate representation on \mathbb{R}^n) is a smaller space, in terms of dimension, than the input space, the neural network (encoder and decoder) must learn to compress using better input information in order to restore it during decoding with the least possible loss.

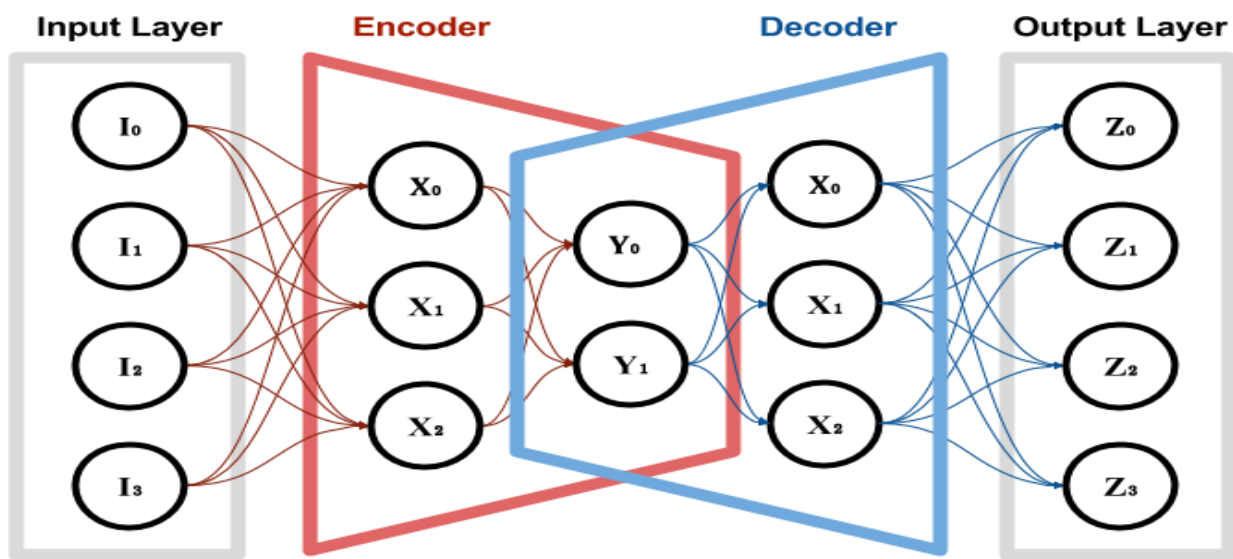


Figure N°40: Example of an auto-encoder. The "Encoder" and "Decoder" blocks both represent artificial neural networks.

The idea of the auto-encoder is a bit similar to a PCA (principle component analysis): one seeks to understand the discriminating characteristics of the data by simply propagating the values from the input to the output, but with an internal layer of dimension smaller than the input (while the output is the same size as the input). Once the model has been trained, it can possibly be used to detect anomalies in a set of data. For example, because it tries to keep only the main information, it eliminates "noise" in data.

Below is an example with image recognition:

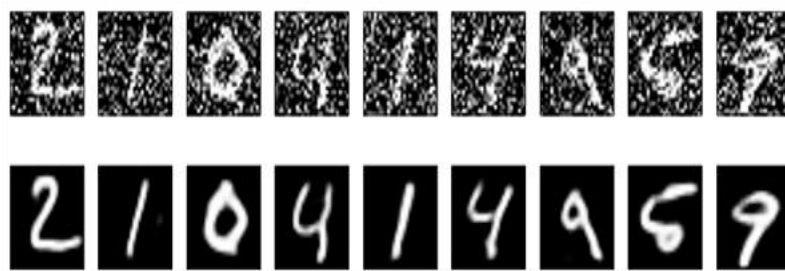


Figure N°41: Example of noise reduction with auto-encoder.

4.5.2 – Generative antagonist networks:

Generative antagonist networks (called GANs) were introduced by I. Goodfellow [124]. The principle of antagonistic artificial networks is the minimization of multiple functions with antagonistic objectives. They have been used to make counterexamples in order to deceive classical artificial neural networks [125], or to automatically find encryption techniques [126]. In the case of generative antagonistic networks, the idea is to train a neural network to generate realistic examples that could belong to the learning dataset. This neural network is decomposed into two different models: a generator and a discriminator (see **Figure N°42**). The generator takes as input a vector of random variables and associates it with an output in the X data space. The discriminator is a binary classifier that takes data from the learning dataset and data from the generator output as input. Its goal is to learn to differentiate between these two types of data. The purpose of the generator is to trick the discriminator when it generates data. During training, the distribution of this output data gradually approaches the distribution of the data in the learning dataset.

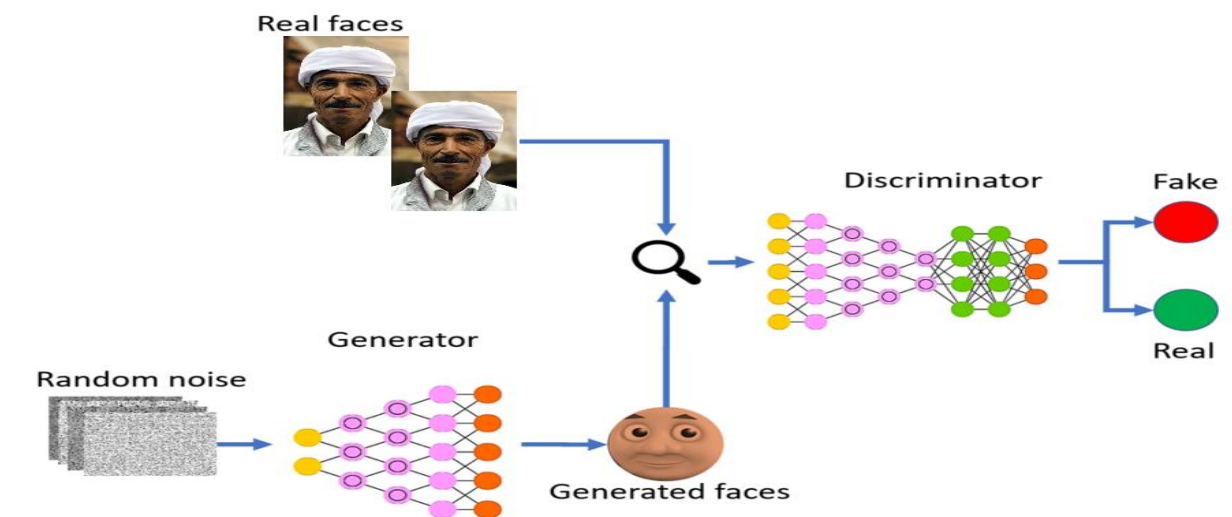


Figure N°42: Architecture of GAN with generator and discriminator.

This type of neural network has become very popular in research. Many improvements have been proposed such as [128] [127] [129] [130]. All these different variants are also used in many applications [131] [132] [133].

The mere purpose of generating new data has limited relevance in the use of GANs. Existing variations in which additional information is given to the generator in order to produce data. For example, in the work of A. Odena et al. [134], the authors propose to add the data class at the input of the generator. This allows the generator to generate a particular type of data, in addition to improving its learning. With a fairly similar principle, it is possible to generate an image from a text [135]. This is encoded and given to the input of the generator as well as inside the discriminator. GANs can also be used to increase the resolution of an image using that image as input [137]. Likewise, some GANs can be used to change the style of an image to that of an artist [136].

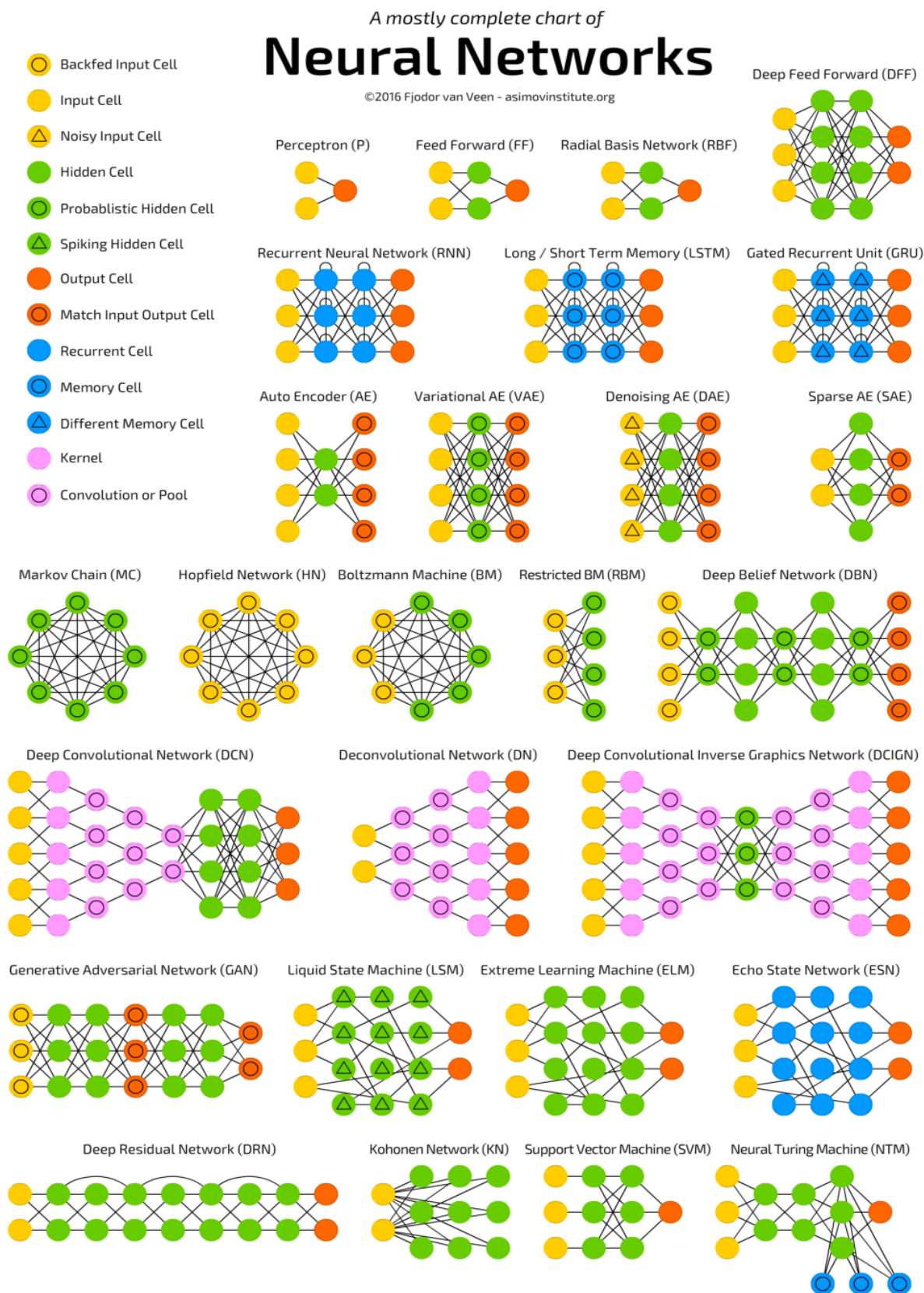


Figure N°43³⁴: A mostly complete chart of artificial neural networks.

³⁴ This picture was taken from <https://www.asimovinstitute.org/neural-network-zoo/> accessed 09 October 2020

5 – Conclusion:

In this chapter we saw an overview of the state of the art of deep learning. It is particularly rich and the number of applications has been increasing steadily over the past few years. Artificial Neural networks can be learned in a supervised or unsupervised manner. Learning is done by gradient descent (or a variant of this method) on a large as possible dataset.

Introduction:

After we presented, the general concepts of Image Processing and Computer Vision Especially Image Classification and the techniques of Deep Learning, this chapter is devoted to present the tools and libraries used in our experiments and a humble discussion about the results.

2 – Tools and Libraries:

2.1 – Python:

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms [141].

2.2 – Google Colaboratory:

Colaboratory, or “Colab” for short, is a product from Google Research. Colab allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis and education. More technically, Colab is a hosted Jupyter notebook service that requires no setup to use, while providing free access to computing resources including GPUs [142].

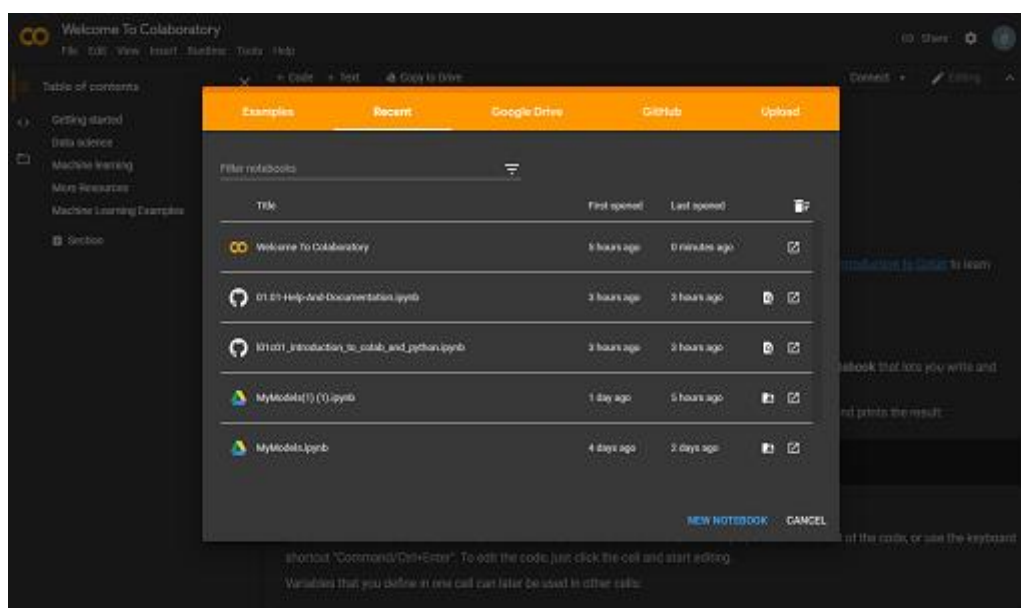


Figure N°44: Google Colaboratory

2.3 – Anaconda navigator:

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda® distribution that allows you to launch applications and easily manage conda packages, environments, and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, macOS, and Linux [143].

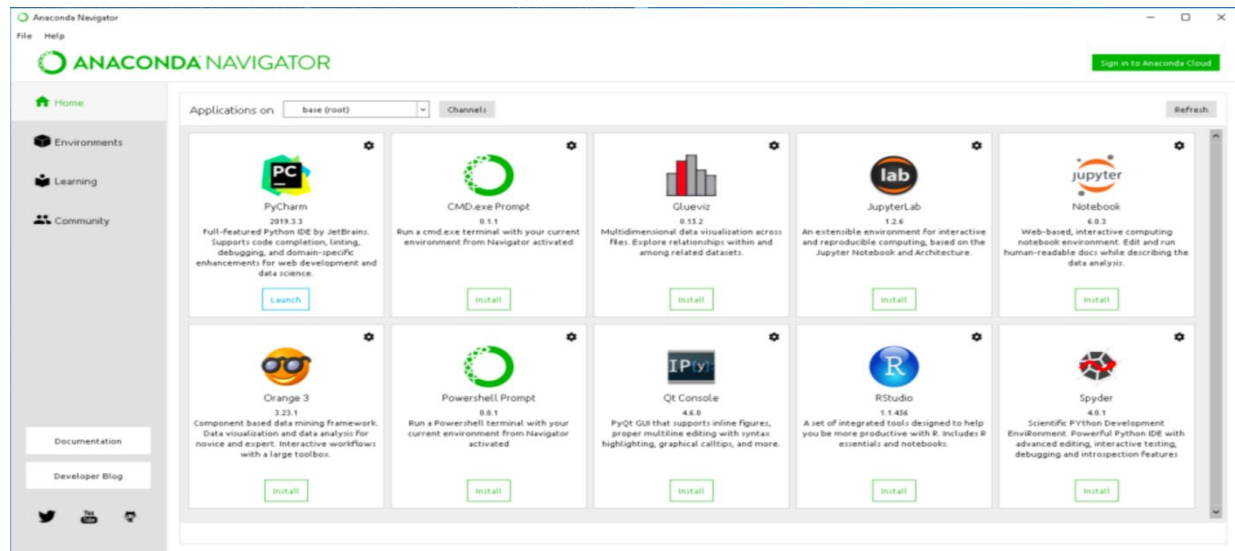


Figure N°45: Anaconda navigator

2.4 – Jupyter notebook:

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more [144].

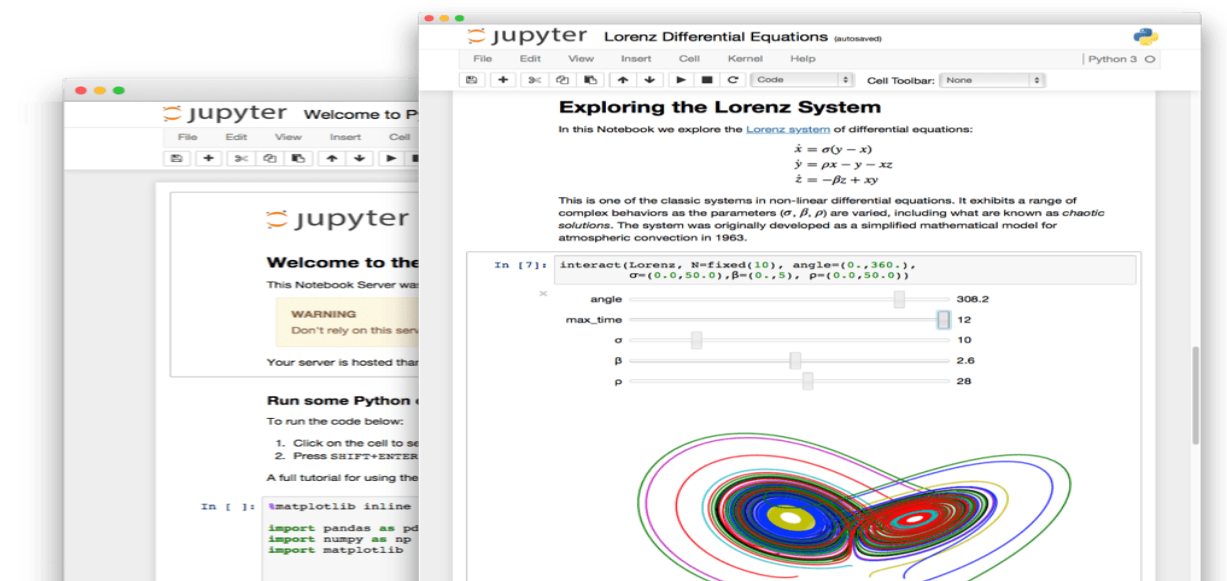


Figure N°46: Jupyter notebook

2.5 – TensorFlow:

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications [145].

2.6 – Keras:

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result as fast as possible is key to doing good research* [146].

2.7 – OpenCV:

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc. OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies [147].

2.8 – PyQt:

PyQt is the contraction of two words: on the one hand, Python (the programming language used) reputed to be very easy to learn; on the other, Qt, an extremely complete framework (mainly for graphical interfaces), but written in C++. PyQt serves as a connecting layer between these two worlds and brings Qt to the Python environment.

Qt is a multiplatform library, recognized above all for its features to help design graphical interfaces. However, Qt can do much more: this library comes with modules for access to SQL databases, a full reusable web browser, a help system, multimedia features. For some time now, it has offered new, more integrated and higher-level features, such as access to mapping and location tools, wireless communication (NFC, Bluetooth), graphics, and data visualization...etc. Also, its environment is very rich, with many other extension libraries available [148].

2.9 – Cifar-10 Dataset [149]:

The CIFAR-10 dataset consists of 60000 32x32 Colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. The classes are completely mutually exclusive. There is no overlap between automobiles and trucks. "Automobile" includes sedans, SUVs, things of that sort. "Truck" includes only big trucks. Neither includes pickup trucks.

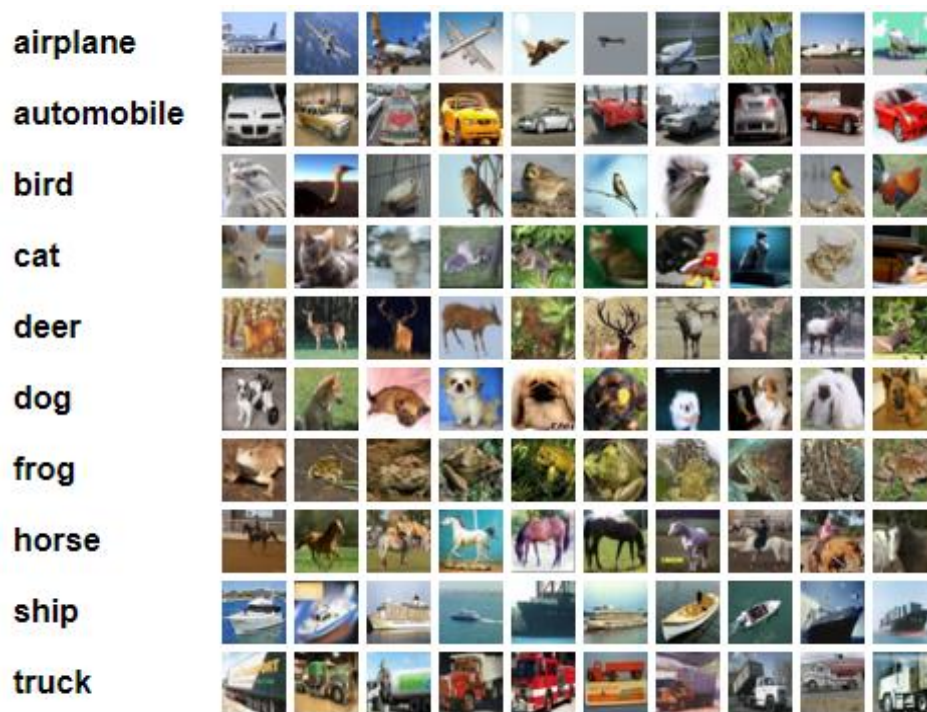


Figure N°47: Represent the classes in the dataset, as well as 10 random images from each.

3 – Experiments:

The main focus throughout this thesis has been to show if noisy images in any way has impact over the performance of deep learning model in image classification. So as a first step to our experimental study we created noisy and restored versions of the original publicly-available dataset selected for our experiment (Cifar-10). For that to be realized three copies of our dataset were degraded by a Gaussian noise with standard deviation $\sigma = [10, 30, 50]$, and another three were hampered by Salt & Pepper noise with the amount $p = [0.1, 0.3, 0.5]$, then filtering methods are applied to restore the versions affected by noise, as a result we have 13 dataset version (original, six noisy datasets and six restored versions).

CH04: Experiments and Desktop Application

The dataset versions affected by Gaussian noise were restored using NLM (Non Local Means) algorithm [03]. To perform the NLM denoising method we used a 7×7 patch, an 11×11 window, and for the parameter h , it was set equal to standard deviation of the Gaussian noise used to hinder the version being restored. Going to the Salt & Pepper noise, all restored versions were generated using a 3×3 Median Filter.

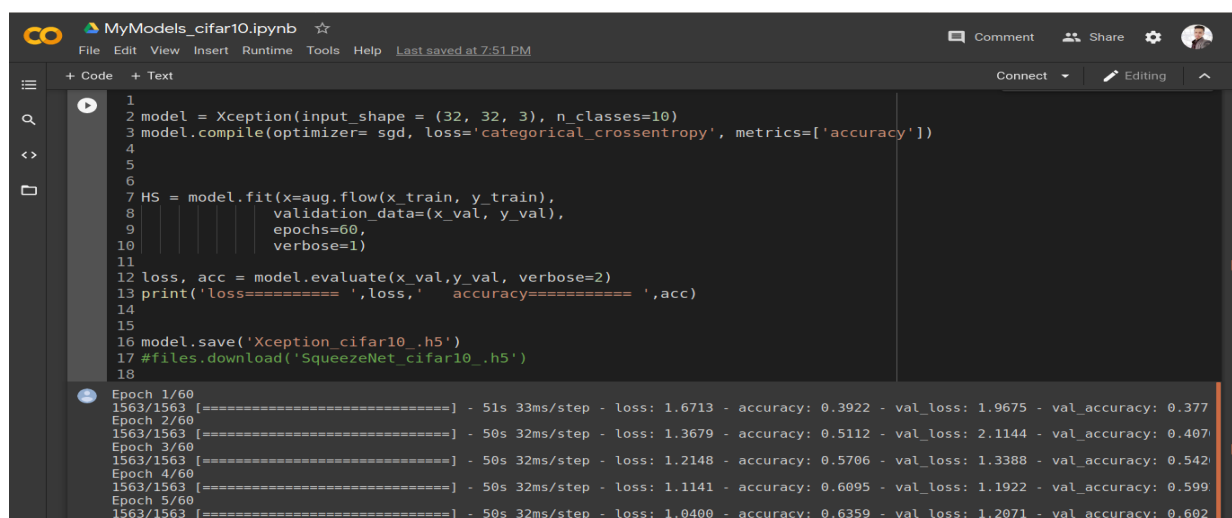
The second step is to learn a classifier for each dataset version creating 13 classifier by each one of the three selected architectures for our study. The architectures were implemented using Keras library.

As a third step to our study, we analyze the differences between learned models. To do that we will begin our scrutiny by comparing classification accuracies in two steps:

- 1. Inter-models:** the goal of it is to visualize which architecture is better resilient when the noise appear in both training and test set with the same level of noise, and see how much harder classifying these datasets gets.
- 2. Intra-models:** here we compare the obtained accuracies with the same model using three datasets (noise-free, noisy, denoised) the goal is to see how the image quality affect the models, and how much the denoising methods are helping in the cases that of noisy images.

Our models were all trained using data augmentation with Keras module. All the models were trained using Google Colab platform with a single 12GB NVIDIA Tesla K80 GPU that can be used up to 12 hours continuously. **XceptionNet** took around 50 seconds per Epoch to train each model with data augmentation (see **Figure N°48**), **GoogleNet** took 47 seconds per Epoch to train each model with data augmentation(see **Figure N°49**), and for the **ResNet**; it took 47 seconds per Epoch to train each model with data augmentation(see **Figure N°50**). We store the weights of the CNN model and record the accuracy and loss for every epoch to be able to evaluate and compare the results for each model.

Additionally, we have created a desktop application that allows you to recognize images or test models even train them.

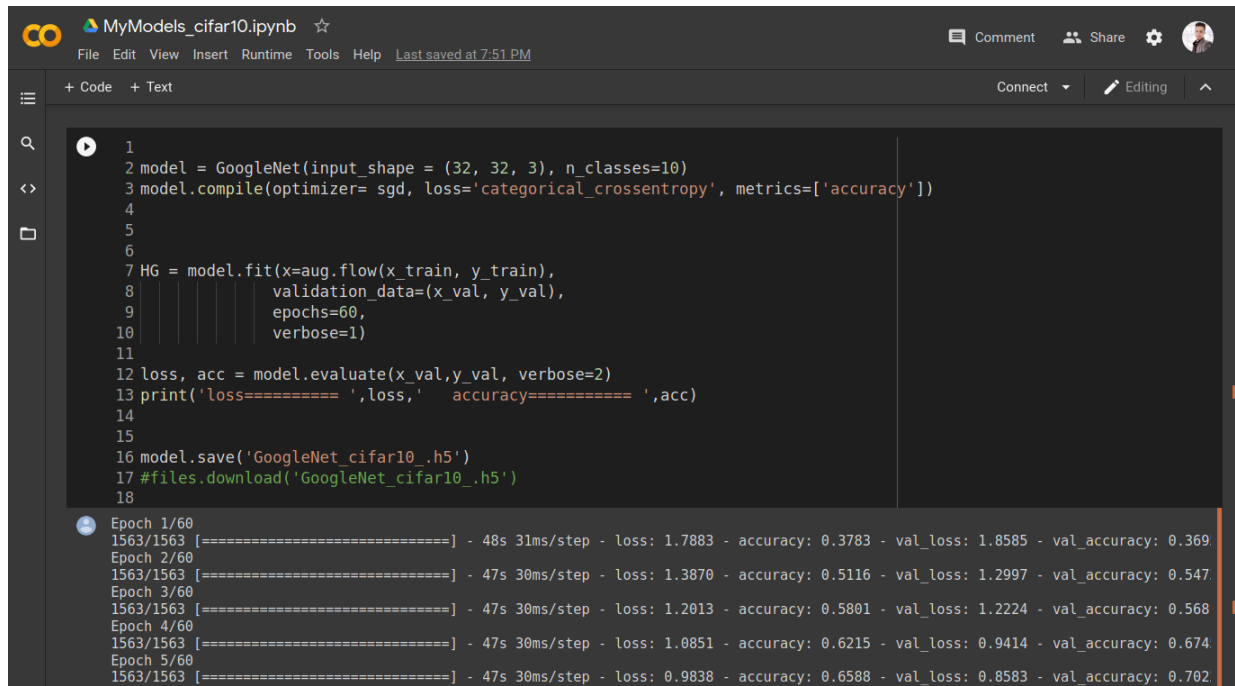


```
1
2 model = Xception(input_shape = (32, 32, 3), n_classes=10)
3 model.compile(optimizer= sgd, loss='categorical_crossentropy', metrics=['accuracy'])
4
5
6
7 HS = model.fit(x=aug.flow(x_train, y_train),
8               validation_data=(x_val, y_val),
9               epochs=60,
10              verbose=1)
11
12 loss, acc = model.evaluate(x_val,y_val, verbose=2)
13 print('loss===== ',loss,' accuracy===== ',acc)
14
15
16 model.save('Xception_cifar10_.h5')
17 #files.download('SqueezeNet_cifar10_.h5')
18
```

Epoch	Step	Time	Loss	Accuracy	Val Loss	Val Accuracy
Epoch 1/60	1563/1563	51s 33ms/step	1.6713	0.3922	1.9675	0.377
Epoch 2/60	1563/1563	50s 32ms/step	1.3679	0.5112	2.1144	0.407
Epoch 3/60	1563/1563	50s 32ms/step	1.2148	0.5706	1.3388	0.542
Epoch 4/60	1563/1563	50s 32ms/step	1.1141	0.6095	1.1922	0.599
Epoch 5/60	1563/1563	50s 32ms/step	1.0400	0.6359	1.2071	0.602

Figure N°48: XceptionNet saved History.

CH04: Experiments and Desktop Application

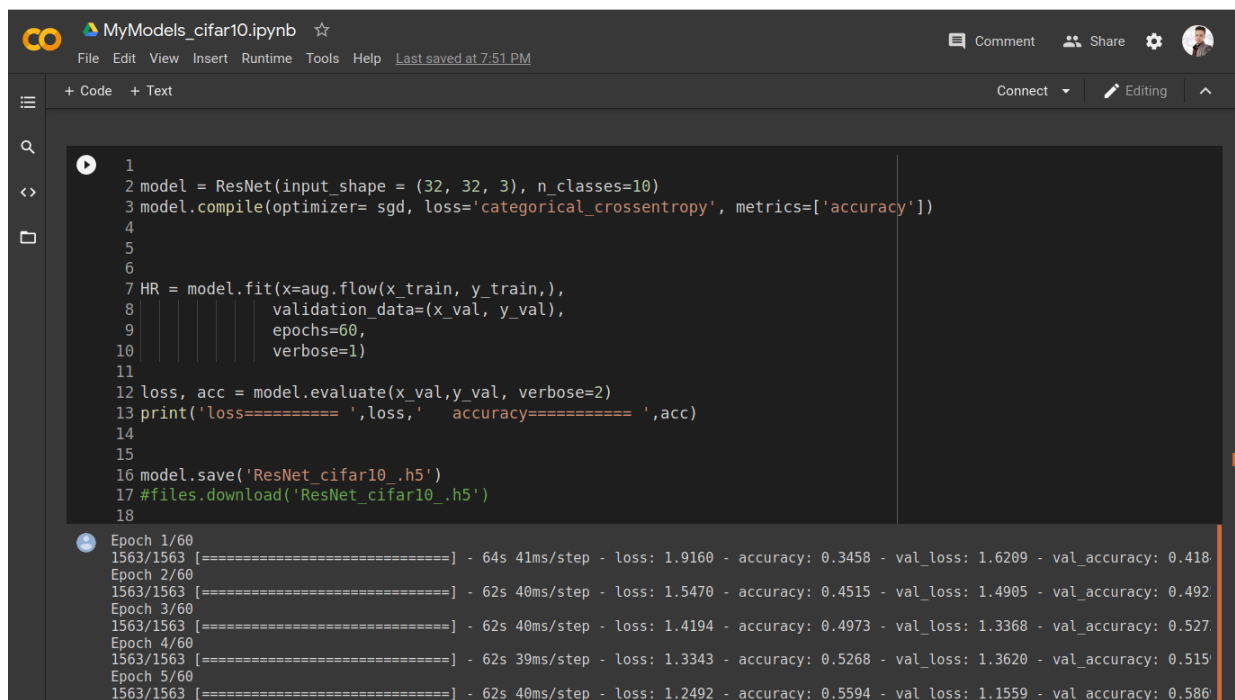


The screenshot shows a Jupyter Notebook titled 'MyModels_cifar10.ipynb'. The code defines a GoogleNet model and trains it for 60 epochs. The output shows the training progress for each epoch, including loss, accuracy, and validation loss/accuracy.

```
1
2 model = GoogleNet(input_shape = (32, 32, 3), n_classes=10)
3 model.compile(optimizer= sgd, loss='categorical_crossentropy', metrics=['accuracy'])
4
5
6
7 HG = model.fit(x=aug.flow(x_train, y_train),
8               validation_data=(x_val, y_val),
9               epochs=60,
10              verbose=1)
11
12 loss, acc = model.evaluate(x_val,y_val, verbose=2)
13 print('loss===== ',loss,' accuracy===== ',acc)
14
15
16 model.save('GoogleNet_cifar10_.h5')
17 #files.download('GoogleNet_cifar10_.h5')
18
```

Epoch 1/60
1563/1563 [=====] - 48s 31ms/step - loss: 1.7883 - accuracy: 0.3783 - val_loss: 1.8585 - val_accuracy: 0.369
Epoch 2/60
1563/1563 [=====] - 47s 30ms/step - loss: 1.3870 - accuracy: 0.5116 - val_loss: 1.2997 - val_accuracy: 0.547
Epoch 3/60
1563/1563 [=====] - 47s 30ms/step - loss: 1.2013 - accuracy: 0.5801 - val_loss: 1.2224 - val_accuracy: 0.568
Epoch 4/60
1563/1563 [=====] - 47s 30ms/step - loss: 1.0851 - accuracy: 0.6215 - val_loss: 0.9414 - val_accuracy: 0.674
Epoch 5/60
1563/1563 [=====] - 47s 30ms/step - loss: 0.9838 - accuracy: 0.6588 - val_loss: 0.8583 - val accuracy: 0.702

Figure N°49: GoogleNet saved History.



The screenshot shows a Jupyter Notebook titled 'MyModels_cifar10.ipynb'. The code defines a ResNet model and trains it for 60 epochs. The output shows the training progress for each epoch, including loss, accuracy, and validation loss/accuracy.

```
1
2 model = ResNet(input_shape = (32, 32, 3), n_classes=10)
3 model.compile(optimizer= sgd, loss='categorical_crossentropy', metrics=['accuracy'])
4
5
6
7 HR = model.fit(x=aug.flow(x_train, y_train,),
8               validation_data=(x_val, y_val),
9               epochs=60,
10              verbose=1)
11
12 loss, acc = model.evaluate(x_val,y_val, verbose=2)
13 print('loss===== ',loss,' accuracy===== ',acc)
14
15
16 model.save('ResNet_cifar10_.h5')
17 #files.download('ResNet_cifar10_.h5')
18
```

Epoch 1/60
1563/1563 [=====] - 64s 41ms/step - loss: 1.9160 - accuracy: 0.3458 - val_loss: 1.6209 - val_accuracy: 0.418
Epoch 2/60
1563/1563 [=====] - 62s 40ms/step - loss: 1.5470 - accuracy: 0.4515 - val_loss: 1.4905 - val_accuracy: 0.492
Epoch 3/60
1563/1563 [=====] - 62s 40ms/step - loss: 1.4194 - accuracy: 0.4973 - val_loss: 1.3368 - val_accuracy: 0.527
Epoch 4/60
1563/1563 [=====] - 62s 39ms/step - loss: 1.3343 - accuracy: 0.5268 - val_loss: 1.3620 - val_accuracy: 0.515
Epoch 5/60
1563/1563 [=====] - 62s 40ms/step - loss: 1.2492 - accuracy: 0.5594 - val_loss: 1.1559 - val accuracy: 0.586

Figure N°50: ResNet saved History.

4 - Implementation:

Our model architectures are based on [155] they were adjusted to recognize tiny images; we created two methods to add noise and filter it using OpenCV, and some relevant functionalities to save models and their related plots. Also, a graphical user interface was implemented using PyQt (see Figure N°51).

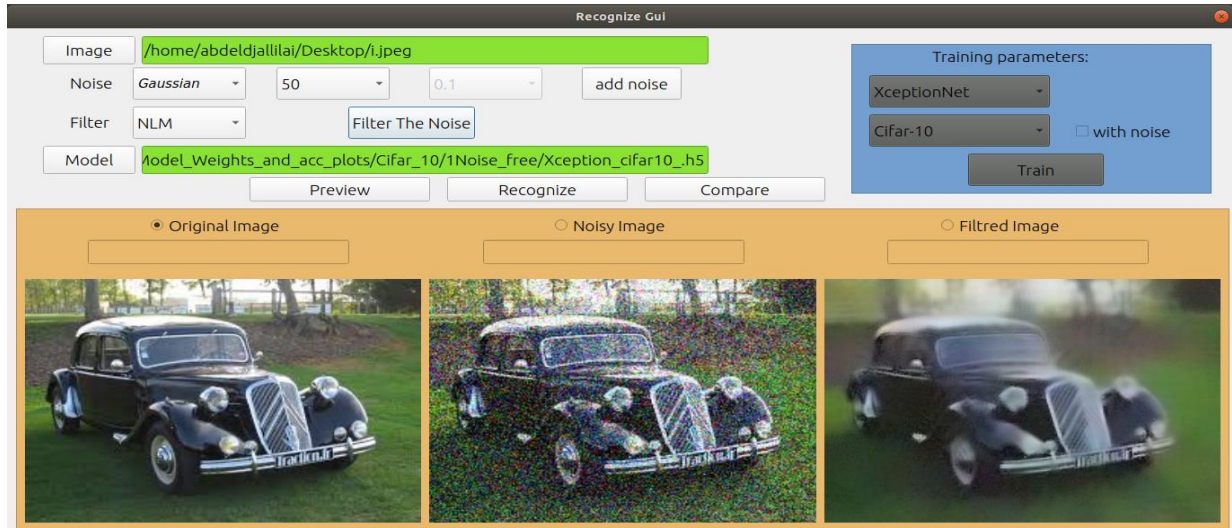


Figure N°51: graphical user interface.

5 – Results and Discussion:

To evidently visualize the impact of noise and filtering methods in the image quality, the structural similarity index measure (SSIM) and the Peak signal to noise ratio (PSNR) values are shown in **Table N° 2**. Smaller values in both the PSNR and SSIM indicate less similarity between the images of the original dataset version and the version compared to. By comparing the results while training and testing our chosen models with the same dataset (same noise and level) we can see how much harder separating between classes gets for these models and how the denoising methods are helping override this hurdle.

			SSIM		PSNR	
			noisy	filtered	noisy	filtered
Cifar-10 Data set	Salt and Pepper	P = 0.1	0.76	0.97	15.26	25.39
		P = 0.2	0.60	0.95	12.46	23.47
		P = 0.3	0.48	0.93	10.90	21.19
		P = 0.4	0.41	0.88	9.85	18.92
		P = 0.5	0.34	0.82	9.08	16.87
	Gaussian	$\sigma = 10$	0.87	0.87	16.92	17.17
		$\sigma = 20$	0.73	0.77	14.22	15.50
		$\sigma = 30$	0.58	0.67	12.45	15.29
		$\sigma = 40$	0.44	0.54	11.20	14.62
		$\sigma = 50$	0.31	0.45	10.28	13.77

Table N°02: PSNR and SSIM for each noise level

In general, the presence of noise in the dataset, even when restored using denoising methods, makes the classification task hard. To better understand this, **Table N°03 and Figure N°52** present the accuracies of the models trained with datasets affected by noise and the restored versions also.

CH04: Experiments and Desktop Application

		XceptionNet		GoogleNet		ResNet		
		noisy	filtered	noisy	filtered	noisy	filtered	
Cifar-10 Data set	original	85.89		84.96		77.46		
	Salt and Pepper	P = 0.1	77.12	81.15	77.98	80.12	66.07	73.04
		P = 0.2	74.47	78.50	72.11	77.84	62.89	72.38
		P = 0.3	66.26	76.14	67.59	75.61	58.22	69.38
		P = 0.4	62.77	72.88	62.48	74.80	51.43	67.33
		P = 0.5	57.02	74.43	57.06	73.30	49.04	64.94
	Gaussian	$\sigma = 10$	76.76	69.92	73.70	68.02	68.11	63.29
		$\sigma = 20$	75.24	70.45	73.82	71.42	68.21	63.61
		$\sigma = 30$	60.37	62.54	66.58	58.97	62.31	56.98
		$\sigma = 40$	52.33	55.69	51.44	44.99	47.32	52.15
$\sigma = 50$		27.97	51.54	25.60	51.28	23.19	46.83	

Table N°03: Accuracy of each model when training and testing using the same dataset version.

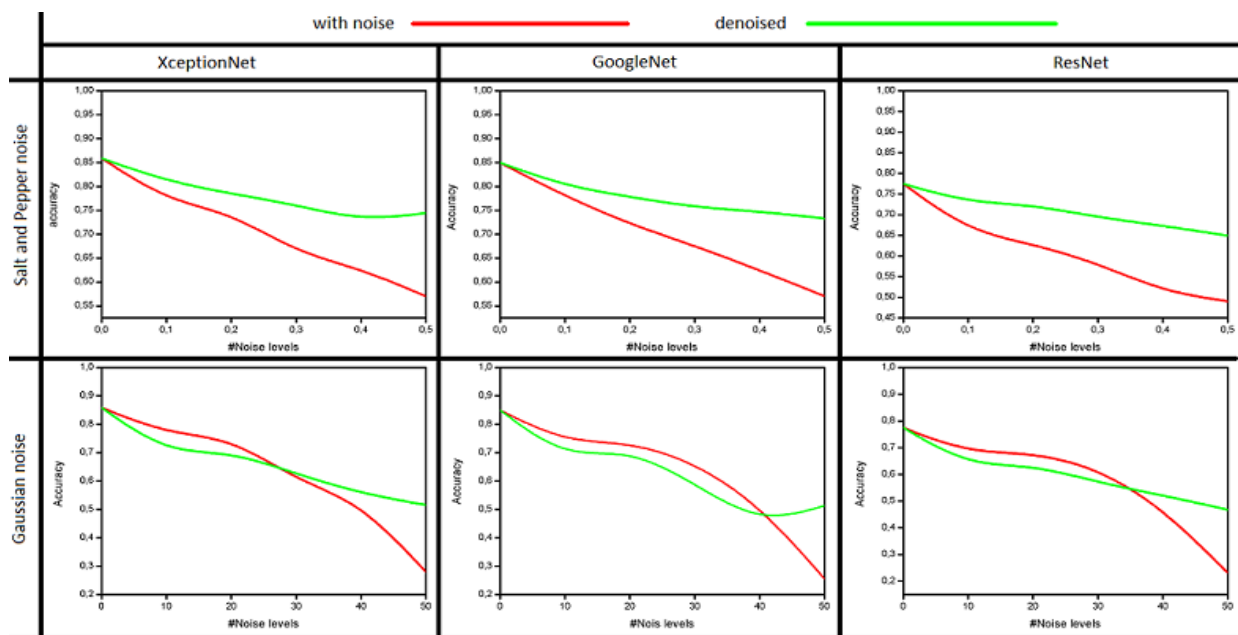


Figure N°52: Comparison between accuracies of each model for different noise levels

In this comparison, the networks trained with the original dataset is used as a baseline scenario, given that these networks have no previous knowledge of any type of noise, while the others have already seen noisy images in some level. Therefore, networks trained with noisy images have an advantage when dealing with noise in future data even when it occurs at a different level. For the XceptionNet, the model trained on the original data obtained an average accuracy of 85.89%, which is the best overall result. On the GoogleNet, the best average result of 84.96% was obtained by the model trained with original data. Lastly, with the ResNet, the model trained on the original dataset obtained an overall 77.46% accuracy.

After adding noise, the accuracy of the models will go down and of course the more noise the less accuracy we get after training.

CH04: Experiments and Desktop Application

As we can see the most of accuracies resulted when using NLM are smaller than accuracies using Gaussian noise, this might be due to the fact that such methods generate blur when denoising images, removing relevant details as shown in **Figure N°53**.

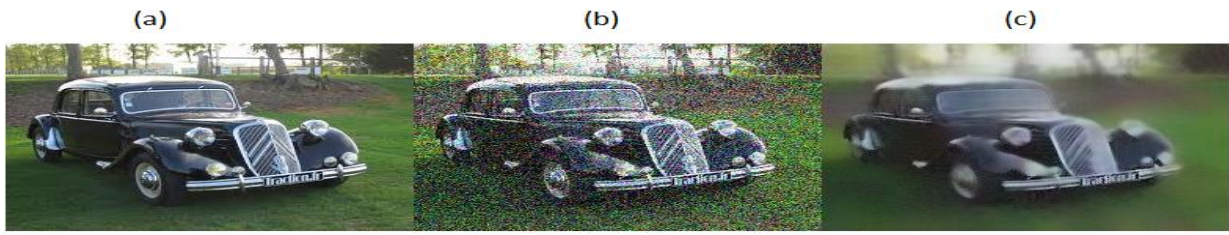


Figure N°53: Result of NLM filtering: (a) original image, (b) Gaussian noisy image, (c) result of NLM filtering.

Due to the big number of loss and accuracies plots we chose to show just two samples (the middle level of every type of noise) see **Figures N°51,52**.

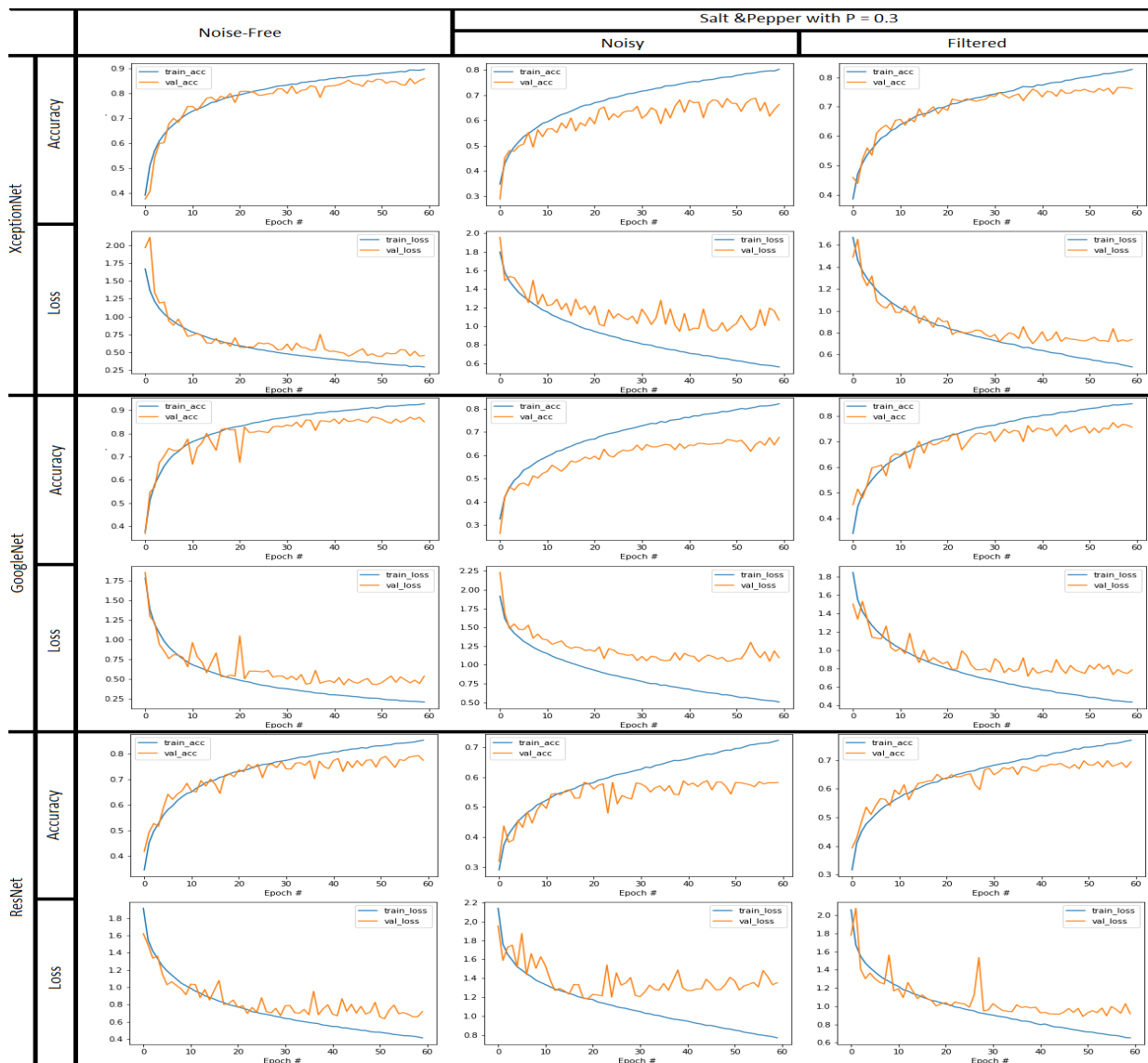


Figure N°54: accuracy and loss plots for each model when using Salt&Pepper noise with $p = 0.3$.

CH04: Experiments and Desktop Application

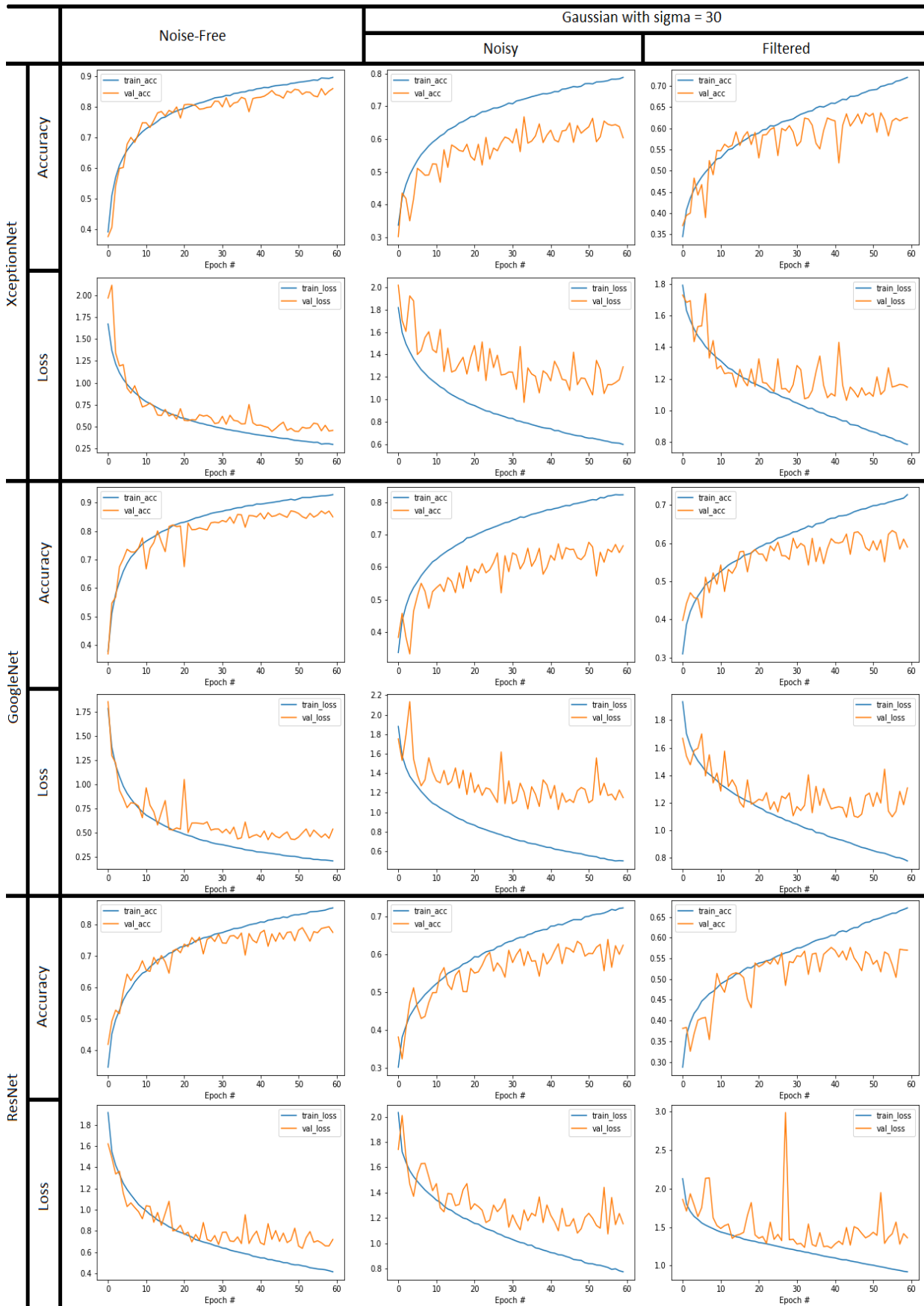


Figure N°55: accuracy and loss plots for each model when using Gaussian noise with $\sigma = 30$.

6 – Conclusion:

Training deep convolutional neural networks with datasets after adding noise to their images at a known level has shown how hard classifying these images gets, our study covered two types of noise with five levels for each type, likewise you can test other types of noise with different models. Our results show that classification is being hardened using datasets affected by noise in both training and test sets.

Regarding the denoising algorithms, datasets restored using Median filtering were able to improve the accuracies compared to datasets with Salt&Pepper noise as well the quality of images, while in the other part datasets versions that were hindered with Gaussian noise gave better results than their restored counterparts using NLM filter. Hence, better results might be achieved using other filtering methods with different parameter choice.

As future work, we aim to use bigger datasets like Cifar-100, even bigger like Image-net and other developed models that might be more resilient to noise.

Bibliographic references:

- [1] - R. C. Gonzalez and R. E. Woods, *Digital image processing*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2008.
- [2] - E. R. Davies, *Computer vision: theory, algorithms, practicalities*, Fifth edition. London, United Kingdom ; Cambridge, MA, United States: Elsevier/Academic Press, 2018.
- [3] - D. E. Dudgeon and R. M. Mersereau, *Multidimensional digital signal processing*. Englewood Cliffs, NJ: Prentice-Hall, 1984.
- [4] - B. Jähne, *Digital image processing: concepts, algorithms, and scientific applications*, 3rd ed. Berlin ; New York: Springer-Verlag, 1995.
- [5] - R. C. Gonzalez and R. E. Woods, *Digital image processing*, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2002.
- [6] - W. K. Pratt, *Digital image processing: PIKS Scientific inside*, 4th ed., Newly updated and rev. Ed. Hoboken, N.J: Wiley-Interscience, 2007.
- [7] - A. J. Bekker and J. Goldberger, "Training deep neural-networks based on unreliable labels," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, Mar. 2016, pp. 2682–2686, doi: 10.1109/ICASSP.2016.7472164.
- [8] - T. Acharya and A. K. Ray, *Image Processing: Principles and Applications*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2005.
- [9] - P. NATTAPONG, "Improvement of Histogram Equalization for Minimum Mean Brightness Error," in *Proceedings of the 2007 WSEAS Int*, Gold Coast, Australia, Jan. 2007, pp. 135–140.
- [10] - A. Rosenfeld and A. C. Kak, *Digital picture processing*, 2nd ed. New York: Academic Press, 1982.
- [11] - N. Sengee, A. Sengee, and H.-K. Choi, "Image contrast enhancement using bi-histogram equalization with neighborhood metrics," *IEEE Trans. Consumer Electron.*, vol. 56, no. 4, pp. 2727–2734, Nov. 2010, doi: 10.1109/TCE.2010.5681162.
- [12] - K. Vinay and B. Himani, "Performance Evaluation of Contrast Enhancement Techniques for Digital Images," *IJCST*, vol. 2, no. 1, Mar. 2011.
- [13] - Yu Wang, Qian Chen, and Baomin Zhang, "Image enhancement based on equal area dualistic sub-image histogram equalization method," *IEEE Trans. Consumer Electron.*, vol. 45, no. 1, pp. 68–75, Feb. 1999, doi: 10.1109/30.754419.
- [14] - Tae Keun Kim, Joon Ki Paik, and Bong Soon Kang, "Contrast enhancement system using spatially adaptive histogram equalization with temporal filtering," *IEEE Trans. Consumer Electron.*, vol. 44, no. 1, pp. 82–87, Feb. 1998, doi: 10.1109/30.663733.
- [15] - D. Menotti, L. Najman, J. Facon, and A. A. Araujo, "Multi-Histogram Equalization Methods for Contrast Enhancement and Brightness Preserving," *IEEE Trans. Consumer Electron.*, vol. 53, no. 3, pp. 1186–1194, Aug. 2007, doi: 10.1109/TCE.2007.4341603.
- [16] - V. Buzuloiu, "Adaptive-neighborhood histogram equalization of color images," *J. Electron. Imaging*, vol. 10, no. 2, p. 445, Apr. 2001, doi: 10.1117/1.1353200.
- [17] - N R. Mokhtar *et al.*, "Image Enhancement Techniques Using Local, Global, Bright, Dark and Partial Contrast Stretching For Acute Leukemia Images," in *Proceedings of the World Congress on Engineering 2009*, London, U.K., Jul. 2009, vol. 1.
- [18] - S. Dodge and L. Karam, "Understanding how image quality affects deep neural networks," in *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, Lisbon, Portugal, Jun. 2016, pp. 1–6, doi: 10.1109/QoMEX.2016.7498955.
- [19] - Soong-Der Chen and Abd. R. Ramli, "Contrast enhancement using recursive mean-separate histogram equalization for scalable brightness preservation," *IEEE Trans. Consumer Electron.*, vol. 49, no. 4, pp. 1301–1309, Nov. 2003, doi: 10.1109/TCE.2003.1261233.
- [20] - T. S. Nazaré, G. B. P. da Costa, W. A. Contato, and M. Ponti, "Deep Convolutional Neural Networks and Noisy Images," in *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, Cham, 2018, pp. 416–424, doi: 10.1007/978-3-319-75193-1_50.
- [21] - T. S. Huang, Ed., *Two-dimensional digital signal processing II: transforms and median filters*. Berlin ; New York: Springer-Verlag, 1981.
- [22] - T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Trans. Acoust., Speech, Signal Process.*, vol. 27, no. 1, pp. 13–18, Feb. 1979, doi: 10.1109/TASSP.1979.1163188.
- [23] - T. Lindeberg and A. Almansa, "Fingerprint enhancement by shape adaptation of scale-space operators with automatic scale selection," *IEEE Trans. on Image Process.*, vol. 9, no. 12, pp. 2027–2042, Dec. 2000, doi: 10.1109/83.887971.
- [24] - M. T. Yildirim, A. Basturk, and M. E. Yuksel, "Impulse Noise Removal From Digital Images by a Detail-Preserving Filter Based on Type-2 Fuzzy Logic," *IEEE Trans. Fuzzy Syst.*, vol. 16, no. 4, pp. 920–928, Aug. 2008, doi: 10.1109/TFUZZ.2008.924358.
- [25] - T. Mélange, M. Nachtegaal, and E. E. Kerre, "Fuzzy Random Impulse Noise Removal From Color Image Sequences," *IEEE Trans. on Image Process.*, vol. 20, no. 4, pp. 959–970, Apr. 2011, doi: 10.1109/TIP.2010.2077305.
- [26] - B. Boashash, *Time-frequency signal analysis and processing: a comprehensive reference*, 2. edition. London: Academic Press Inc, 2015.
- [27] - E. Abreu, M. Lightstone, S. K. Mitra, and K. Arakawa, "A new efficient approach for the removal of impulse noise from highly corrupted images," *IEEE Trans. on Image Process.*, vol. 5, no. 6, pp. 1012–1025, Jun. 1996, doi: 10.1109/83.503916.
- [28] - Haidi Ibrahim, "An efficient implementation of switching median filter with boundary discriminative noise detection for image corrupted by impulse noise," *Sci. Res. Essays*, vol. 6, no. 26, Nov. 2011, doi: 10.5897/SRE11.856.
- [29] - Nicholas Sia Pik Kong and Haidi Ibrahim, "The effect of shape and weight towards the performance of Simple Adaptive Median filter in reducing impulse noise level from digital images," in *2nd International Conference on Education Technology and Computer*, Shanghai, China, Jun. 2010, vol. 5, pp. 118–121, doi: 10.1109/ICETC.2010.5530039.
- [30] - M. Aharon, M. Elad, and A. Bruckstein, "K-SVD: An Algorithm for Designing Overcomplete Dictionaries for Sparse Representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006, doi: 10.1109/TSP.2006.881199.

CH04: Experiments and Desktop Application

- [31] - Joseph Salmon, "Agrégation d'estimateurs et méthodes à patch pour le débruitage d'images numériques," Phd, Diderot-Paris VII, Paris, 2010.
- [32] - B. A. Olshausen and D. J. Field, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, Jun. 1996, doi: 10.1038/381607a0.
- [33] - B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?," *Vision Research*, vol. 37, no. 23, pp. 3311–3325, Dec. 1997, doi: 10.1016/S0042-6989(97)00169-7.
- [34] - K. Dabov, A. Foi, V. Katkovnik, and K. Egiazarian, "Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering," *IEEE Trans. on Image Process.*, vol. 16, no. 8, pp. 2080–2095, Aug. 2007, doi: 10.1109/TIP.2007.901238.
- [35] - Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, Vladimir Katkovnik, and Karen Egiazarian, "A nonlocal and shape-adaptive transform-domain collaborative filtering," presented at the Workshop on Local and Non-Local Approx, Tampere, Finland, 2008.
- [36] - Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian, "Bm3d image denoising with shape-adaptive principal component analysis," *Inria*, Apr. 2009.
- [37] - G. B. P. da Costa, W. A. Contato, T. S. Nazare, J. E. S. B. Neto, and M. Ponti, "An empirical study on the effects of different types of noise in image classification tasks," *arXiv:1609.02781 [cs]*, Sep. 2016, Accessed: Sep. 27, 2020. [Online]. Available: <http://arxiv.org/abs/1609.02781>.
- [38] - F. Benzarti and H. Amiri, "Speckle Noise Reduction in Medical Ultrasound Images," *arXiv:1305.1344 [cs]*, May 2013,
- [39] - Salivahanan S, Vallavaraj A, and Gnanapriya C, *Digital signal processing*. NewDelhi: Tata McgrawHill Education, 2001.
- [40] - L. Gagnon and A. Jouan, "Speckle filtering of SAR images: a comparative study between complex-wavelet-based and standard filters," San Diego, CA, Oct. 1997, pp. 80–91, doi: 10.1117/12.279681.
- [41] - Sarita Veera Dangeti, "Denoising techniques - a comparison," LSU Master's These, Louisiana State University and Agricultural and Mechanical Colleg, 2003.
- [42] - A. K. Boyat and B. K. Joshi, "A Review Paper: Noise Models in Digital Image Processing," *arXiv:1505.03489 [cs]*, May 2015.
- [43] - More Manisha and Shivale Nitin, "Linear Filtering Based Image Restoration with Image De-Blurring Toolkit," *IJSR*, vol. 5, no. 7, Jul. 2016.
- [44] - M. Bergounioux, *Introduction au traitement mathématique des images - méthodes déterministes*. Heidelberg: Springer, 2015.
- [45] - A. Buades, B. Coll, and J. M. Morel, "A Review of Image Denoising Algorithms, with a New One," *Multiscale Model. Simul.*, vol. 4, no. 2, pp. 490–530, Jan. 2005, doi: 10.1137/040616024.
- [46] - C. Kervrann and J. Boulanger, "Optimal Spatial Adaptation for Patch-Based Image Denoising," *IEEE Trans. on Image Process.*, vol. 15, no. 10, pp. 2866–2878, Oct. 2006, doi: 10.1109/TIP.2006.877529.
- [47] - Thomas Brox and Daniel Cremers, Eds., "Iterated nonlocal means for texture restoration," in *International Conference on Scale Space and Variational Methods in Computer Vision*, Berlin ; New York, 2007, pp. 13–24.
- [48] - T. Brox, O. Kleinschmidt, and D. Cremers, "Efficient Nonlocal Means for Denoising of Textural Patterns," *IEEE Trans. on Image Process.*, vol. 17, no. 7, pp. 1083–1092, Jul. 2008, doi: 10.1109/TIP.2008.924281.
- [49] - D. Tschumperle and L. Brun, "Non-local image smoothing by applying anisotropic diffusion PDE's in the space of patches," in *2009 16th IEEE International Conference on Image Processing (ICIP)*, Cairo, Nov. 2009, pp. 2957–2960, doi: 10.1109/ICIP.2009.5413453.
- [50] - C. Louchet and L. Moisan, "Total Variation as a Local Filter," *SIAM J. Imaging Sci.*, vol. 4, no. 2, pp. 651–694, Jan. 2011, doi: 10.1137/100785855.
- [51] - A. Buades, B. Coll, and J.-M. Morel, "A Non-Local Algorithm for Image Denoising," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA, 2005, vol. 2, pp. 60–65, doi: 10.1109/CVPR.2005.38.
- [52] - J. P. Serra, *Image analysis and mathematical morphology*. London ; New York: Academic Press, 1982.
- [53] - J. Serra, "Morphological optics," *Journal of Microscopy*, vol. 145, no. 1, pp. 1–22, Jan. 1987, doi: 10.1111/j.1365-2818.1987.tb01312.x.
- [54] - S. G. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 11, no. 7, pp. 674–693, Jul. 1989, doi: 10.1109/34.192463.
- [55] - Y. Meyer, *Ondelettes et opérateurs Volume 1 et 2: Ondelettes, Opérateurs de Calderon-Zygmund*. 1997.
- [56] - M. Misiti, Y. Misiti, G. Oppenheim, and J.-M. Poggi, Eds., *Wavelets and their Applications*. London, UK: ISTE, 2007.
- [57] - R. A. DeVore and B. J. Lucier, *Wavelets, in Acta numerica 1*. Cambridge: Cambridge University Press, 1991.
- [58] - C. E. Heil and D. F. Walnut, "Continuous and Discrete Wavelet Transforms," *SIAM Rev.*, vol. 31, no. 4, pp. 628–666, Dec. 1989, doi: 10.1137/1031129.
- [59] - O. Rioul and M. Vetterli, "Wavelets and signal processing," *IEEE Signal Process. Mag.*, vol. 8, no. 4, pp. 14–38, Oct. 1991, doi: 10.1109/79.91217.
- [60] - G. Strang, "Wavelets and Dilation Equations: A Brief Introduction," *SIAM Rev.*, vol. 31, no. 4, pp. 614–627, Dec. 1989, doi: 10.1137/1031128.
- [61] - G. Strang, "Wavelet transforms versus Fourier transforms," *Bull. Amer. Math. Soc.*, vol. 28, no. 2, pp. 288–306, Apr. 1993, doi: 10.1090/S0273-0979-1993-00390-2.
- [62] - R. S. Strichartz, "How To Make Wavelets," *The American Mathematical Monthly*, vol. 100, no. 6, pp. 539–556, Jun. 1993, doi: 10.1080/00029890.1993.11990449.
- [63] - T.-W. Lin, "Compressed quadtree representations for storing similar images," *Image and Vision Computing*, vol. 15, no. 11, pp. 833–843, Nov. 1997, doi: 10.1016/S0262-8856(97)00031-0.
- [64] - D. Marr, *Vision: a computational investigation into the human representation and processing of visual information*. Cambridge, Mass: MIT Press, 2010.
- [65] - J. Martinez and S. Guillaume, "Colour image retrieval fitted to «classical» querying," in *Image Analysis and Processing*, vol. 1311, A. Del Bimbo, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 14–21.

CH04: Experiments and Desktop Application

- [66] - J. Martinez and E. Loisant, "Browsing image databases with Galois' lattices," in *Proceedings of the 2002 ACM symposium on Applied computing - SAC '02*, Madrid, Spain, 2002, p. 791, doi: 10.1145/508791.508944.
- [67] - R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*, 2nd ed. New York: Wiley, 2001.
- [68] - Keyzers Daniel, Dahmen Jörg, and Ney Hermann, *Invariant Classification of Red Blood Cells: A Comparison of Different Approaches*. Bildverarbeitung für die Medizin, 2001.
- [69] - Zaiane Osmar, Antonie Luiza, and Coman, Alexandru, "Mammography Classification By an Association Rule-based Classifier," in *Proceedings of the Third International Workshop on Multimedia Data Mining, MDM/KDD'2002, July 23rd, 2002*, Edmonton, University of Alberta, Canada, 2002, pp. 62–69.
- [70] - M. Rodriguez-Damian, E. Cernadas, A. Formella and R. Sa-Otero, "Pollen classification using brightness-based and shape-based descriptors," *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004., Cambridge, 2004*, pp. 212-215 Vol.2, doi: 10.1109/ICPR.2004.1334098 .
- [71] - T. Maenpää, M. Pietikainen, and J. Viertola, "Separating color and pattern information for color texture discrimination," in *Object recognition supported by user interaction for service robots*, Quebec City, Que., Canada, 2002, vol. 1, pp. 668–671, doi: 10.1109/ICPR.2002.1044840.
- [72] - T. Mäenpää, J. Viertola, and M. Pietikäinen, "Optimising Colour and Texture Features for Real-time Visual Inspection," *Patt. Anal. App.*, vol. 6, no. 3, pp. 169–175, Dec. 2003, doi: 10.1007/s10044-002-0179-1.
- [73] - K. H. G., . Mohd. M. M., . A. H., and . S. R., "Scale Invariant Feature Transform Technique for Weed Classification in Oil Palm Plantation," *J. of Applied Sciences*, vol. 8, no. 7, pp. 1179–1187, Jul. 2008, doi: 10.3923/jas.2008.1179.1187.
- [74] - Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, Dec. 1989, doi: 10.1162/neco.1989.1.4.541.
- [75] - Ø. Due Trier, A. K. Jain, and T. Taxt, "Feature extraction methods for character recognition-A survey," *Pattern Recognition*, vol. 29, no. 4, pp. 641–662, Apr. 1996, doi: 10.1016/0031-3203(95)00118-2.
- [76] - Y. B. Lauziere, D. Gingras, and F. P. Ferrie, "A model-based road sign identification system," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Kauai, HI, USA, 2001, vol. 1, p. I-1163-I-1170, doi: 10.1109/CVPR.2001.990662.
- [77] - Suard Frédéric, "Méthodes à noyaux pour la détection de piétons.," Phd, LITIS - INSA of Rouen, France, 2006.
- [78] - F. Suard, V. Guigue, A. Rakotomamonjy, and A. Benschrair, "Pedestrian detection using stereo-vision and graph kernels," in *IEEE Proceedings. Intelligent Vehicles Symposium, 2005.*, Jun. 2005, pp. 267–272, doi: 10.1109/IVS.2005.1505113.
- [79] - F. Suard, A. Rakotomamonjy, A. Benschrair, and A. Broggi, "Pedestrian Detection using Infrared images and Histograms of Oriented Gradients," in *2006 IEEE Intelligent Vehicles Symposium*, Jun. 2006, pp. 206–212, doi: 10.1109/IVS.2006.1689629.
- [80] - C. Hilario, J. M. Collado, J. M. Armingol, and A. de la Escalera, "Pedestrian Detection for Intelligent Vehicles Based on Active Contour Models and Stereo Vision," in *Computer Aided Systems Theory – EUROCAST 2005*, Berlin, Heidelberg, 2005, pp. 537–542, doi: 10.1007/11556985_70.
- [81] - P. Negri, X. Clady, S. M. Hanif, and L. Prevost, "A Cascade of Boosted Generative and Discriminative Classifiers for Vehicle Detection," *EURASIP J. Adv. Signal Process.*, vol. 2008, no. 1, p. 782432, Dec. 2008, doi: 10.1155/2008/782432.
- [82] - H. Shao, T. Svoboda, and L. Gool, "ZuBuD Zurich Buildings Database for Image Based Recognition," 2003.
- [83] - P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Dec. 2001, vol. 1, p. 1–I, doi: 10.1109/CVPR.2001.990517.
- [84] - P. Viola and M. J. Jones, "Robust Real-Time Face Detection," *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154, May 2004, doi: 10.1023/B:VISI.0000013087.49260.fb.
- [85] - A. Rad, R. Safabakhsh, N. Qaragozlou, and M. Zaheri, "Fast iris and pupil localization and eyelid removal using gradient vector pairs and certainty factors," in *The Irish Machine Vision and Image Processing Conf*, 2004, pp. 82–91.
- [86] - G. Guo and M. J. Jones, "Iris Extraction Based on Intensity Gradient and Texture Difference," in *2008 IEEE Workshop on Applications of Computer Vision*, Jan. 2008, pp. 1–6, doi: 10.1109/WACV.2008.4544018.
- [87] - P. Bolon *et al.*, *Analyse d'images : Filtrage et segmentation*. MASSON, 1995.
- [88] - N. Otsu, "A Threshold Selection Method from Gray-Level Histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979, doi: 10.1109/TSMC.1979.4310076.
- [89] - G. S. di Baja and E. Thiel, "Skeletonization algorithm running on path-based distance maps," *Image and Vision Computing*, vol. 14, no. 1, pp. 47–57, Feb. 1996, doi: 10.1016/0262-8856(95)01039-4.
- [90] - A. L. Blum and P. Langley, "Selection of relevant features and examples in machine learning," *Artificial Intelligence*, vol. 97, no. 1–2, pp. 245–271, Dec. 1997, doi: 10.1016/S0004-3702(97)00063-5.
- [91] - L. Lam, S.-W. Lee, and C. Y. Suen, "Thinning methodologies-a comprehensive survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 9, pp. 869–885, Sep. 1992, doi: 10.1109/34.161346.
- [92] - Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning.*, vol. 1. Cambridge: The MIT Press, 2016.
- [93] - A. D. Gordon, L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, "Classification and Regression Trees.," *Biometrics*, vol. 40, no. 3, p. 874, Sep. 1984, doi: 10.2307/2530946.
- [94] - G. V. Kass, "An Exploratory Technique for Investigating Large Quantities of Categorical Data," *Applied Statistics*, vol. 29, no. 2, p. 119, 1980, doi: 10.2307/2986296.
- [95] - J. MacQueen, "Some methods for classification and analysis of multivariate observations," presented at the Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 1967.
- [96] - J. P. Benzécri, *L'analyse des données: leçons sur l'analyse factorielle et la reconnaissance des formes et travaux du Laboratoire de statistique de l'Université de Paris VI*. Dunod, 1973.
- [97] - G. Celeux, E. Diday, and G. Govaert, "Classification automatique de données environnement statistique et informatique," *undefined*, 1989.
- [98] - A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
- [99] - G. Hinton *et al.*, "Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012, doi: 10.1109/MSP.2012.2205597.

CH04: Experiments and Desktop Application

- [100] - D. P. Kingma, D. J. Rezende, S. Mohamed, and M. Welling, "Semi-Supervised Learning with Deep Generative Models," *arXiv:1406.5298 [cs, stat]*, Oct. 2014.
- [101] - Olivier Chapelle, Jason Weston, and Bernhard Schölkopf. 2002. Cluster kernels for semi-supervised learning. In *Proceedings of the 15th International Conference on Neural Information Processing Systems (NIPS'02)*. MIT Press, Cambridge, MA, USA, 601–608. .
- [102] - O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," in *Proceedings of the British Machine Vision Conference 2015*, Swansea, 2015, p. 41.1-41.12, doi: 10.5244/C.29.41.
- [103] - K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015,
- [104] - C. Szegedy *et al.*, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, Jun. 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.
- [105] - R. Collobert and J. Weston, "A unified architecture for natural language processing: deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning - ICML '08*, Helsinki, Finland, 2008, pp. 160–167, doi: 10.1145/1390156.1390177.
- [106] - A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 3, pp. 328–339, Mar. 1989, doi: 10.1109/29.21701.
- [107] - D. Amodei *et al.*, "Deep Speech 2: End-to-End Speech Recognition in English and Mandarin," *arXiv:1512.02595 [cs]*, Dec. 2015.
- [108] - K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991, doi: 10.1016/0893-6080(91)90009-T.
- [109] - J. Turian, J. Bergstra, and Y. Bengio, "Quadratic features and deep architectures for chunking," in *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Short Papers on - NAACL '09*, Boulder, Colorado, 2009, p. 245, doi: 10.3115/1620853.1620921.
- [110] - J. Han and C. Moraga, "The influence of the sigmoid function parameters on the speed of backpropagation learning," in *From Natural to Artificial Neural Computation*, Berlin, Heidelberg, 1995, pp. 195–201, doi: 10.1007/3-540-59497-3_175.
- [111] - Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," in *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Müller, Eds. Berlin, Heidelberg: Springer, 1998, pp. 9–50.
- [112] - T. Schaul, S. Zhang, and Y. LeCun, "No More Pesky Learning Rates," *arXiv:1206.1106 [cs, stat]*, Feb. 2013,
- [113] - L. Prechelt, "Automatic early stopping using cross validation: quantifying the criteria," *Neural Networks*, vol. 11, no. 4, pp. 761–767, Jun. 1998, doi: 10.1016/S0893-6080(98)00010-0.
- [114] - N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *JMLR.org*, Jan. 01, 2014.
- [115] - B. Schölkopf, A. J. Smola, F. Bach, and M. D. of the M. P. I. for B. C. in T. G. P. B. Scholkopf, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.
- [116] - Duchi John, Hazan Elad, and Singer Yoram, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization" *Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.
- [117] - D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017,
- [118] - M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method," *arXiv:1212.5701 [cs]*, Dec. 2012,
- [119] - O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *Int J Comput Vis*, vol. 115, no. 3, pp. 211–252, Dec. 2015, doi: 10.1007/s11263-015-0816-y.
- [120] - D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, Sep. 1999, vol. 2, pp. 1150–1157 vol.2, doi: 10.1109/ICCV.1999.790410.
- [121] - Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.
- [122] - M. Niepert, M. Ahmed, and K. Kutzkov, "Learning Convolutional Neural Networks for Graphs," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, NY, USA, Jun. 2016, vol. 48.
- [123] - S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," in *Proceedings of the 32Nd International Conference on International Conference on Machine Learning*, Lille, France, Mar. 2015, vol. 37.
- [124] - I. J. Goodfellow *et al.*, "Generative Adversarial Networks," *arXiv:1406.2661 [cs, stat]*, Jun. 2014,
- [125] - N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The Limitations of Deep Learning in Adversarial Settings," in *2016 IEEE European Symposium on Security and Privacy (EuroSP)*, Mar. 2016, pp. 372–387, doi: 10.1109/EuroSP.2016.36.
- [126] - M. Abadi and D. G. Andersen, "Learning to Protect Communications with Adversarial Neural Cryptography," *arXiv:1610.06918 [cs]*, Oct. 2016,
- [127] - Martin Arjovsky, S. CHINTALA, and Léon Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," *arXiv:1701.04862 [cs, stat]*, Jan. 2017
- [128] - M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein Generative Adversarial Networks," in *International Conference on Machine Learning*, Jul. 2017, pp. 214–223, Accessed: Sep. 17, 2020. [Online]. Available: <http://proceedings.mlr.press/v70/arjovsky17a.html>.
- [129] - T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," *arXiv:1606.03498 [cs]*, Jun. 2016
- [130] - M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Curran Associates, Inc., 2017, pp. 6626–6637.
- [131] - L. Ma, X. Jia, Q. Sun, B. Schiele, T. Tuytelaars, and L. Van Gool, "Pose Guided Person Image Generation," *arXiv:1705.09368 [cs]*, Jan. 2018,

CH04: Experiments and Desktop Application

- [132] - Y. Jin, J. Zhang, M. Li, Y. Tian, H. Zhu, and Z. Fang, "Towards the Automatic Anime Characters Creation with Generative Adversarial Networks," *arXiv:1708.05509 [cs]*, Aug. 2017,
- [133] - C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, Barcelona, Spain, Dec. 2016, pp. 613–621, Accessed: Sep. 17, 2020. [Online].
- [134] - A. Odena, C. Olah, and J. Shlens, "Conditional Image Synthesis with Auxiliary Classifier GANs," in *International Conference on Machine Learning*, Jul. 2017, pp. 2642–2651,
- [135] - S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, "Generative adversarial text to image synthesis," in *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, New York, NY, USA, Jun. 2016, pp. 1060–1069, Accessed: Sep. 17, 2020. [Online].
- [136] - J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation Using Cycle-Consistent Adversarial Networks," in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2242–2251, doi: 10.1109/ICCV.2017.244.
- [137] - C. Ledig *et al.*, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul. 2017, pp. 105–114, doi: 10.1109/CVPR.2017.19.
- [138] - C. Bishop, "Exact Calculation of the Hessian Matrix for the Multilayer Perceptron," *Neural Computation*, vol. 4, no. 4, pp. 494–501, Jul. 1992, doi: 10.1162/neco.1992.4.4.494.
- [139] - K. Gurney, *An introduction to neural networks*. London: UCL Press, 1997.
- [140] - I. B. Levitan and L. K. Kaczmarek, *The neuron: cell and molecular biology*, Fourth edition. Oxford ; New York: Oxford University Press, 2015.
- [141] - "The Python Tutorial — Python 3.8.6rc1 documentation." <https://docs.python.org/3/tutorial/> (accessed Sep. 24, 2020).
- [142] - "Colaboratory – Google." <https://research.google.com/colaboratory/faq.html#resource-limits> (accessed Sep. 24, 2020).
- [143] - "Anaconda Navigator — Anaconda documentation." <https://docs.anaconda.com/anaconda/navigator/> (accessed Sep. 24, 2020).
- [144] - "Project Jupyter." <https://www.jupyter.org> (accessed Sep. 24, 2020).
- [145] - "TensorFlow," *TensorFlow*. <https://www.tensorflow.org/> (accessed Sep. 24, 2020).
- [146] - K. Team, "Keras documentation: About Keras." <https://keras.io/about/> (accessed Sep. 24, 2020).
- [147] - "OpenCV." <https://opencv.org/about/> (accessed Sep. 24, 2020).
- [148] - P. Denis and T. Cuvelier, *Créer des applications graphiques en Python avec PyQt5*. 2017.
- [149] - "CIFAR-10 and CIFAR-100 datasets." <https://www.cs.toronto.edu/~kriz/cifar.html> (accessed Sep. 24, 2020).
- [150] - Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, May 2015, doi:10.1038/nature14539.
- [151] - P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for Activation Functions," *arXiv:1710.05941 [cs]*, Oct. 2017, Accessed: Sep. 27, 2020. [Online]. Available: <http://arxiv.org/abs/1710.05941>.
- [152] - V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, Haifa, Israel, Jun. 2010, pp. 807–814, Accessed: Sep. 26, 2020. [Online].
- [153] - G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout," in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013, pp. 8609–8613, doi: 10.1109/ICASSP.2013.6639346.
- [154] - C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *arXiv:1811.03378 [cs]*, Nov. 2018, Accessed: Sep. 27, 2020. [Online]. Available: <http://arxiv.org/abs/1811.03378>.
- [155] - "Machine-Learning-Tokyo/DL-workshop-series," *GitHub*. <https://github.com/Machine-Learning-Tokyo/DL-workshop-series> (accessed Oct. 01, 2020).
- [156] - P. Cunningham and S. J. Delany, "k-Nearest Neighbour Classifiers: 2nd Edition (with Python examples)," *arXiv:2004.04523 [cs, stat]*, Apr. 2020, Accessed: Oct. 05, 2020. [Online]. Available: <http://arxiv.org/abs/2004.04523>.
- [157] - I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International Conference on Machine Learning*, May 2013, pp. 1139–1147, Accessed: Oct. 09, 2020. [Online]. Available: <http://proceedings.mlr.press/v28/sutskever13.html>.
- [158] - D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986, doi: 10.1038/323533a0.
- [159] - Vapnik, Vladimir N. *The Nature of Statistical Learning Theory*. Springer New York, 1995. DOI.org (Crossref), doi:10.1007/978-1-4757-2440-0.