ALGERIAN REPUBLIC DEMOCRATIC AND POPULAR MINISTRY OF HIGH EDUCATION AND SCIENTIFIC RESEARCHES

University of Mohamed Khider Biskra

Faculty of Exact Science and Science of Life and Nature

Department of Computer Science

Dissertation For Master Degree Graduation In Computer Science

Field Computer Graphics & Computer Vision

Title:

Character Recognition In The Image

Using Deep Learning

Realized by:

Barkat Ishak

Defended the 06/07/2019, in front of the jury composed of:

Academic year 2018-2019

Dedication

This research work is dedicated to:
My beloved mother, my first teacher who always had faith in me.
My respected father whose support helped me to reach my goals.
My lovely sisters Sara, Nafla and Takia.
My dear brothers Immed and mehdi for their encouragement.
To all my uncles and aunties.
Last but not least, to my best friends and colleagues Amir, Mostapha, Seddik, Hazem, Hatem, Lotfi, Yasmine, Roufaida and Khaoula, And to all those whom I love and cherish.

Acknowledgement

Before all, my deep and sincere praise to Allah the almighty for providing me with strength and patience to accomplish this research work.
My thanks go to my supervisor Zerari Abd el Moumen for his valuable guidance. Special thanks would go to the members of the jury namely, Mokhtari Bilel, Chighoub Rabia for their efforts to evaluate my research work.
I also express my gratitude to my classmate Halimi Roufaida for her tips through this work.

Abstract

The human brain as complex product of evolutionary success, has acquired and is empowered by a sophisticated learning intuition which consists of the broader memory and the creativeness of imagination, both grant flexibility and an extreme potential of pattern recognition and acquisition, that's aside, a computer is only suitable for information retrieval and is as good as the program it runs, which makes it's window of usage very limited to fewer tasks, but is that the apex of computer science? What if computers were able to learn and recognize unknown patterns without the constant teaching and bypass first-hand programming?.

Key-words: Deep learning, Image processing, Character recognition, Convolution neural networks.

Contents

List of Tables

List of Figures

General Introduction

The technological developments of the last twenty years have allowed digital systems to invade our daily lives. There is more to demonstrate that digital occupies a place of choice in the world today. Among the major components of digital systems, great importance is given to the image. The representation and processing of digital images is the subject of very active research at present. The processing of images is a very vast field that has known, and still knows, an important development for a few decades.

Artificial intelligence has come a long way in recent years, it has applications everywhere. Deep learning is one of the most popular learning techniques today. Deep learning has absolutely dominated the field of processing and thus recognition of images. Computer vision (image recognition) is one of the hottest topics in artificial intelligence. Its power is so revolutionary that almost every day we are seeing technical advances in image processing services. Today, image recognition techniques based on Deep Learning are increasingly used in industry with for example applications in medical imaging and security. One of the classes of deep learning that represents an interesting method for image processing and recognition is that of convolutional neural networks.

Among the technologies of image recognition, character recognition has become a very interesting and challenging field of study. Character recognition is a computer process that makes it possible to recognize, in an image, the letters composing a text. This makes it possible to transform an image file into a text file. The main interest of this technique is to be able to then search in a text, as well as to select words or sentences of the same text. When we talk about character recognition in images, we need a reliable and accurate system that really extracts accurate results in a timely manner.

In this master project, we focus on character recognition using deep learning in order to improve the accuracy and training time of this kind of character recognition. To do this, we propose architectures in the CNN method based on certain types of character recognition; handwritten characters in Arabic and Latin, as another type one has optical character recognition.

Our memory and organized in 3 chapters with a general introduction and a conclusion. In the first chapter, we present the field of digital image recognition. Through the second chapter, we provide an introduction to the field of deep learning. The third chapter deals with the design of our solution and the implementation of our new architectures and the results obtained.

Chapter 1

Recognition in the digital image

1.1 Introduction

Image recognition technology has great potential for widespread adoption in various sectors. In fact, it is not a technology of the future, but it is already our present. Image recognition or computer vision is a technical discipline that looks for ways to automate all the work that a human visual system can do. Considering the growing potential of computer vision, many companies are investing in image recognition to interpret and analyze data primarily from visual sources, including for medical image analysis, identifying objects in autonomous cars, face detection for security purposes, etc. So the need to process images quickly became obvious.

This chapter provides an introduction to recognition in the digital image by focusing in particular on the basic techniques used for image processing and character recognition in digital images.

1.2 Image processing

1.2.1 Photo-metric property of the human visual system

1.2.1.1 Human eye

The human eye is an organ that reacts with light and allows light perception, color vision and depth perception [1].
Its the gateway to one of our five senses. Also an organ that reacts with light. It allows light perception, color vision and depth perception. A normal human eye can see about 10 million different colors! There are many parts of a human eye, and that is what we are going to cover in this atom [2].

A- Properties
Contrary to what you might think, the human eye is not a perfect sphere, but is made up of two differently shaped pieces, the cornea and the sclera. These two parts are connected by a

ring called the limb-us. The part of the eye that is seen is the iris, which is the colorful part of the eye. In the middle of the iris is the pupil, the black dot that changes size. The cornea covers these elements, but is transparent. The fund's is on the opposite of the pupil, but inside the eye and can not be seen without special instruments. The optic nerve is what conveys the signals of the eye to the brain. is a diagram of the eye. The human eye is made up of three coats:

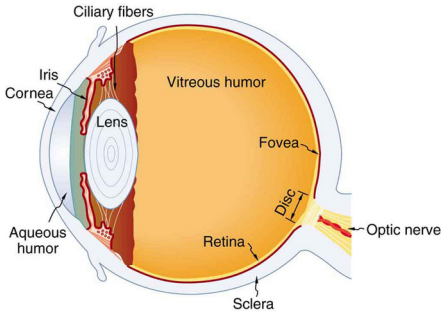


Figure 1.1: The Human Eye [2].

- Outermost Layer – composed of the cornea and the sclera.
- Middle Layer – composed of the choroid, ciliary body and iris.
- Innermost Layer – the retina, which can be seen with an instrument called the ophthalmoscope.

Once you are inside these three layers, there is the aqueous humor (clear fluid that is contained in the anterior chamber and posterior chamber), vitreous body (clear jelly that is much bigger than the aqueous humor), and the flexible lens. All of these are connected by the pupil.

B- Dynamics

Whenever the eye moves, even just a little, it automatically readjusts the exposure by adjusting the iris, which regulates the size of the pupil. This is what helps the eye adjust to dark places or really bright lights. The lens of the eye is similar to one in glasses or cameras. The human eye is had an aperture, just like a camera. The pupil serves this function, and the iris is the aperture stop. The different parts of the eye has different refractive indexes, and this is what bends the rays to form an image. The cornea provides two-thirds of the power to the eye. The lens provides the remaining power. The image passes through several layers of the eye, but happens in a way very similar to that of a convex lens. When the image finally reaches the retina, it is inverted, but the brain will correct this. shows what happens.

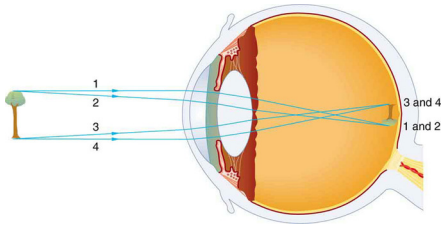


Figure 1.2: Vision Diagram [2].

1.2.1.2 Color vision

A- Definition of Colors

Colors is the characteristic of human visual perception described through color categories, with names such as red, orange, yellow, green, blue, or purple [3]. This perception of color derives from the stimulation of cone cells in the human eye by electromagnetic radiation in the visible spectrum. Color categories and physical specifications of color are associated with objects through the wavelength of the light that is reflected from them. This reflection is governed by the object's physical properties such as light absorption, emission spectra, etc... [4].

Using the cone cells in the retina, we perceive images in color, each type of cone specifically sees in regions of red, green, or blue.

With human eyesight, cone cells are responsible for color vision. From there, it is important to understand how color is perceived. Using the cone cells in the retina, we perceive images in color. Each type of cone specifically sees in regions of red, green, or blue, (RGB), in the color spectrum of red, orange, yellow, green, blue, indigo, violet [5].

The colors in between these absolutes are seen as different linear combinations of RGB. This is why TVs and computer screens are made up of thousands of little red, green, or blue lights, and why colors in electronic form are represented by different values of RGB. These values are usually given in the value of their frequency in log form [2].

B- Colors Perception

Although Aristotle and other ancient scientists had already written on the nature of light and color vision, it was not until Newton that light was identified as the source of the color sensation. In 1810, Goethe published his comprehensive Theory of Colors in which he ascribed physiological effects to color that are now understood as psychological.

In 1801 Thomas Young proposed his trichromatic theory, based on the observation that any color could be matched with a combination of three lights. This theory was later refined by James Clerk Maxwell and Hermann von Helmholtz. As Helmholtz puts it, "the principles of Newton's law of mixture were experimentally confirmed by Maxwell in 1856. Young's theory of color sensations, like so much else that this marvelous investigator achieved in advance of his time, remained unnoticed until Maxwell directed attention to it [6].

At the same time as Helmholtz, Ewald Hering developed the opponent process theory of color, noting that color blindness and afterimages typically come in opponent pairs (red-green, blue-orange, yellow-violet, and black-white). Ultimately these two theories were synthesized in 1957 by Hurvich and Jameson, who showed that retinal processing corresponds to the trichromatic theory, while processing at the level of the lateral geniculate nucleus corresponds to the opponent theory [7].

In 1931, an international group of experts known as the (International Commission of Lighting (ICL)) developed a mathematical color model, which mapped out the space of observable colors and assigned a set of three numbers to each.

Color in the eye

The ability of the human eye to distinguish colors is based upon the varying sensitivity of different cells in the retina to light of different wavelengths. Humans are trichromatic—the retina contains three types of color receptor cells, or cones. One type, relatively distinct from the other two, is most responsive to light that is perceived as blue or blue-violet, with wavelengths around 450 nm; cones of this type are sometimes called short-wavelength cones, S cones, or blue cones. The other two types are closely related genetically and chemically: middle-wavelength cones, M cones, or green cones are most sensitive to light perceived as green, with wavelengths around 540 nm, while the long-wavelength cones, L cones, or red cones, are most sensitive to light is perceived as greenish yellow, with wavelengths around 570 nm.

Light, no matter how complex its composition of wavelengths, is reduced to three color components by the eye. Each cone type adheres to the principle of univariance, which is that each cone's output is determined by the amount of light that falls on it over all wavelengths. For each location in the visual field, the three types of cones yield three signals based on the extent to which each is stimulated. These amounts of stimulation are sometimes called tristimulus values.

Color in the brain

While the mechanisms of color vision at the level of the retina are well-described in terms of tristimulus values, color processing after that point is organized differently. A dominant theory of color vision proposes that color information is transmitted out of the eye by three opponent processes, or opponent channels, each constructed from the raw output of the cones: a red–green channel, a blue–yellow channel, and a black–white "luminance" channel. This theory has been supported by neurobiology, and accounts for the structure of our subjective color experience. Specifically, it explains why humans cannot perceive a "reddish green" or "yellowish blue", and it predicts the color wheel: it is the collection of colors for which at least one of the two color channels measures a value at one of its extremes.

The exact nature of color perception beyond the processing already described, and indeed the status of color as a feature of the perceived world or rather as a feature of our perception of the world—a type of qualia—is a matter of complex and continuing philosophical dispute.

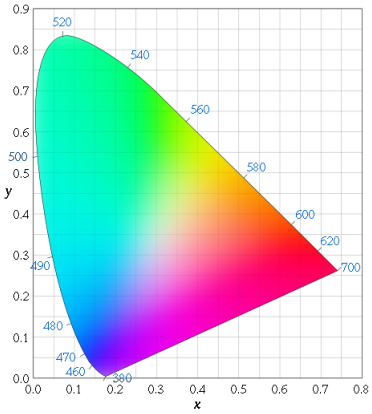


Figure 1.3: CIE 1931 color space chromaticity diagram[6]

C- Additive, Subtractive coloring and Structural color

Additive color is light created by mixing together light of two or more different colors. Red, green, and blue are the additive primary colors normally used in additive color systems such as projectors and computer terminals, figure 1.4.

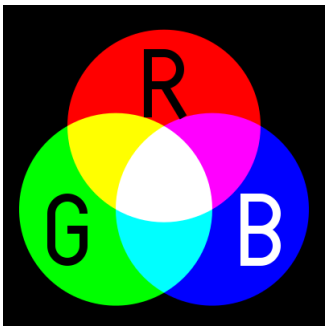


Figure 1.4: Additive color mixing: combining red and green yields yellow; combining all three primary colors together yields white [4].

Subtractive coloring uses dyes, inks, pigments, or filters to absorb some wavelengths of light and not others. The color that a surface displays comes from the parts of the visible spectrum that are not absorbed and therefore remain visible. Without pigments or dye, fabric fibers, paint base and paper are usually made of particles that scatter white light (all colors) well in all directions. When a pigment or ink is added, wavelengths are absorbed or "subtracted" from white light, so light of another color reaches the eye.

If the light is not a pure white source (the case of nearly all forms of artificial lighting), the resulting spectrum will appear a slightly different color. Red paint, viewed under blue light, may appear black. Red paint is red because it scatters only the red components of the spectrum. If red paint is illuminated by blue light, it will be absorbed by the red paint, creating the appearance of a black object, figure 1.5.

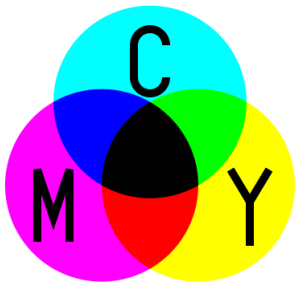


Figure 1.5: Subtractive color mixing: combining yellow and magenta yields red; combining all three primary colors together yields black [7].

Structural colors are colors caused by interference effects rather than by pigments. Color effects are produced when a material is scored with fine parallel lines, formed of one or more parallel thin layers, or otherwise composed of microstructures on the scale of the color's wavelength. If the microstructures are spaced randomly, light of shorter wavelengths will be scattered preferentially to produce Tyndall effect colors: the blue of the sky (Rayleigh scattering, caused by structures much smaller than the wavelength of light, in this case air molecules), the luster of opals, and the blue of human irises. If the microstructures are aligned in arrays, for example the array of pits in a CD, they behave as a diffraction grating: the grating reflects different wavelengths in different directions due to interference phenomena, separating mixed "white" light into light of different wavelengths. If the structure is one or more thin layers then it will reflect some wavelengths and transmit others, depending on the layers thickness [8].

Table of spectral or near-spectral colors
Most of the colors listed do not reach the maximal (spectral) colorfulness, or are not usually seen with it, but they can be saturated enough to be perceived closely to their dominant wavelength spectral colors. Ranges of wavelengths and frequencies are only approximate.
Wavelengths and frequencies in gray indicate dominant wavelengths and frequencies, not actual range of spectrum composing a specified color, which extends farther to both sides and is averaged by receptors to give a near-spectral appearance [9]. Also we can see different Color spaces in table 1.1.

1.2.2 Digital Camera

As we know already the real world contain one or more light sources that can intersects with surfaces, which can viewed or stored by an optic sensor (digital camera) a device with certain architecture, can gives us as an output a digital image.
A camera is the basic sensing element. In simple terms, most cameras rely on the property of light to cause hole/electron pairs (the charge carriers in electron- ics) in a conducting material. camera models developed by Healey and Kondepudy (1994), Tsin, Ramesh, and Kanade (2001), Liu,Szeliski, Kang et al. (2008), shows a simple version of the processing stages that occur in

Color term,light source, or dye	Sample	Wavelength, nm	Frequency, THz	Hue*h*	Comments
Red		740–625	405–479		A traditional, broad color term, which includes some nearby non-spectral hues. The short-wave boundary can extend to 620 or even about 610 nanometers
• Extreme spectral red = red (CIE RGB)	*	740	405	?	The exact spectral position has more influence on luminance
• red (Wide-gamut RGBprimary)[6]	*	= 700	= 428	?	than on chromaticity in this band; chromaticities are almost
• Helium–neon laser	*	633	473	?	the same for these two variants
• Some carmine dyes	*	NIR–602[7]	497-NIR	?	Near-spectral, but other parts of carmine (color) are purple
• red (sRGB primary)		614–609	488–492	0	Noticeably non-spectral
Orange		620–585 625–590[5]	483–512 479–508	0–30	The short-wave (yellowish) part corresponds to amber, the long-wave (reddish) side nears (or includes) RGB red above.
Yellow		585–560 590–565[5]	512–540 508–530		A traditional color term
• Sodium-vapor lamp		589	508	?	
• yellow (NCS)		?	?	50	Gold has almost identical chromaticity at h = 51
• Munsell 5Y for V = 10, C = 22[8]		577	519	?	
• process (canary) yellow		?	?	56	
• yellow (sRGB secondary)		570	?	60	
• Chartreuse yellow		?	?	68	
Lime		564	?	75	May be classified as either green or yellow
Green		565–###	530–###		A traditional, broad color term
• Chartreuse green		?	?	90	
• Bright green		556 - *$&#	?	96	
• Harlequin		552	?	105	
• green (sRGB primary)		549	547	120	Noticeably non-spectral
• green (Wide-gamut RGBprimary)[6]	*	525	571	?	Almost spectral
• Spring green (sRGB definition)	*	?	?	150	May lie rather far from the spectrum
• green (NCS)	*	?	?	160	
• Munsell 5G for V = 4, C = 29[8][9]	*	503	597	(?)= 163(extrap.)	
Cyan		500+–480[10]520–500[5]	593–624 576–600		Sometimes included (or overlaps) with blue , terminological distinction between the two is inconsistent
• Turquoise	*	?	?	175	Most of "turquoise" lies far away of the spectrum
• cyan (sRGB secondary)	*	488	?	180	Lie rather far from the spectrum
• process cyan	*	?	?	193	
Blue		490–450 500–435[5]	610–666 600–689		A traditional, broad color term, which used to include cyan
• blue (NCS)	*	?	?	197	Lies rather far from the spectrum
• Azure (sRGB definition)	*	488	614	210	May lie rather far from the spectrum
• Munsell 5B for V = 5, C = 20[8]	*	482	622	(?)= 225(extrap.)	
• blue (RGB primary)		466–436[11]	?	240(of sRGB)	May be classified as indigo or (if indigo is omitted) as violet
Indigo		446	672	(?)= 243(extrap.)	Definition is controversial, this wavelength least disputably belongs to "indigo"
Violet	*	450–400 435–380[5]	666–750 689–788	up to277(extrap.)	Far spectral violet is very dim and rarely seen. The term also extends to purples

Table 1.1: Table of spectral or near-spectral colors [9]

modern digital cameras. Chakrabarti, Scharstein, and Zickler (2009) developed a sophisticated 24-parameter model that is an even better match to the processing performed in today's cameras.

A simple model that accounts for the most important effects such as exposure (gain and shutter speed), nonlinear mappings, sampling and aliasing, and noise. As we see in figure 1.6 which is a representative pipeline of image sensing.

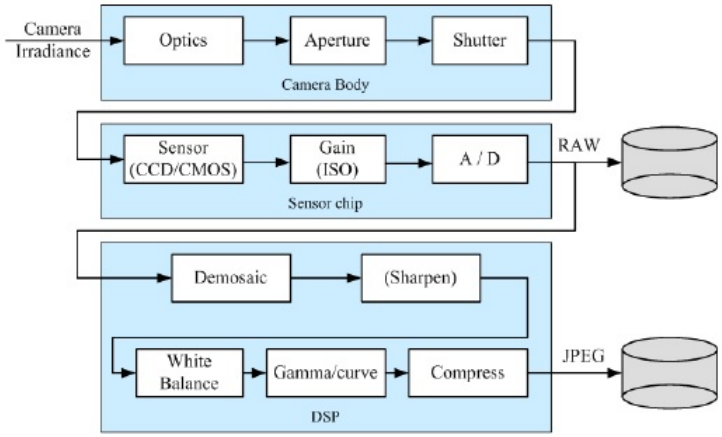


Figure 1.6: Image sensing pipeline, showing the various sources of noise as well as typical digital post-processing steps [1].

1.2.2.1 Sampling and aliasing

light impinging on the image sensor falls onto the active sense areas in the imaging chip? The photons arriving at each active cell are integrated and then digitized. However, if the fill factor on the chip is small and the signal is not otherwise band-limited, visually unpleasing aliasing can occur [7].

To explore the phenomenon of aliasing, let us first look at a one-dimensional signal as we see in figure 1.7, in which we have two sine waves, one at a frequency of

$$f = 3/4 \tag{1.1}$$

and the other at

$$f = 5/4 \tag{1.2}$$

If we sample these two signals at a frequency of

$$f = 2 \tag{1.3}$$

we see that they produce the same samples (shown in black), and so we say that they are aliased [1].

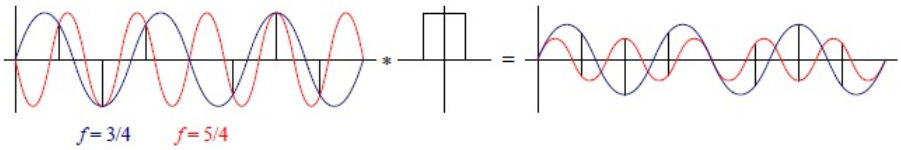


Figure 1.7: One-dimensional signal [1].

1.2.2.2 Color cameras

The design of RGB video cameras has historically been based around the availability of colored phosphors that go into television sets. When standard-definition color television was invented (NTSC), a mapping was defined between the RGB values that would drive the three color guns in the cathode ray tube (CRT) and the XYZ values that unambiguously define perceived color [9].

1.2.2.3 Compression

The last stage in a camera's processing pipeline is usually some form of image compression (unless you are using a lossless compression scheme such as camera RAW or PNG).
All color video and image compression algorithms start by converting the signal into YCbCr (or some closely related variant), so that they can compress the luminance signal with higher fidelity than the chrominance signal. (Recall that the human visual system has poorer frequency response to color than to luminance changes.) In video, it is common to subsample Cb and Cr by a factor of two horizontally; with still images (JPEG), the subsampling (averaging) occurs both horizontally and vertically see figure 1.8.

Figure 1.8: Image compressed with JPEG at three quality settings [1].

1.2.3 Feature extraction

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set [10].
When the input data to an algorithm is too large to be processed and it is suspected to be

redundant (e.g. the same measurement in both feet and meters, or the repetitiveness of images presented as pixels), then it can be transformed into a reduced set of features (also named a feature vector). Determining a subset of the initial features is called feature selection [11].

1.2.3.1 The Uses of feature extraction

The process of feature extraction is useful when you need to reduce the number of resources needed for processing without losing important or relevant information. Feature extraction can also reduce the amount of redundant data for a given analysis. Also, the reduction of the data and the machine's efforts in building variable combinations (features) facilitate the following learning and generalization steps in the machine learning process.

1.2.3.2 Practical Uses of Feature Extraction

- **Autoencoders:** The purpose of autoencoders is unsupervised learning of efficient data coding. Feature extraction is used here to identify key features in the data for coding by learning from the coding of the original data set to derive new ones.
- **Bag-of-Words:** A technique for natural language processing that extracts the words (features) used in a sentence, document, website, etc. and classifies them by frequency of use. This technique can also be applied to image processing.
- **Image Processing:** Algorithms are used to detect features such as shaped, edges, or motion in a digital image or video.

1.2.3.3 Feature extraction methods

One very important area of application is image processing, in which algorithms are used to detect and isolate various desired portions or shapes (features) of a digitized image or video stream.

A- Low-level

- Edge detection [12].
- Corner detection [13].
- Blob detection [14].
- Ridge detection [15].
- Scale-invariant feature transform [16].

B- Curvature

- Edge direction, changing intensity, autocorrelation [17].

C- Flexible methods

- Deformable, parameterized shapes.
- Active contours (snakes)

1.2.4 Image Segmentation

In computer vision, image segmentation is the process of partitioning a digital image into multiple segments (sets of pixels, also known as super-pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze [18][19].
Image segmentation is the task of finding groups of pixels that "go together". In statistics, this problem is known as cluster analysis and is a widely studied area with hundreds of different algorithms (Jain and Dubes 1988; Kaufman and Rousseeuw 1990; Jain, Duin, and Mao 2000;Jain, Topchy, Law et al. 2004) [1].
Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

Segmentation is a crucial step in image processing. To date, there are many segmentation methods, which can be grouped into four main classes:

- Region-based segmentation. Examples include: region-growing, decomposition / merge.
- Edge-based segmentation.
- classification or thresholding.
- Segmentation based on cooperation between the first three segmentations.

The figure 1.9 show different segmentation methods made by variation of algorithms.

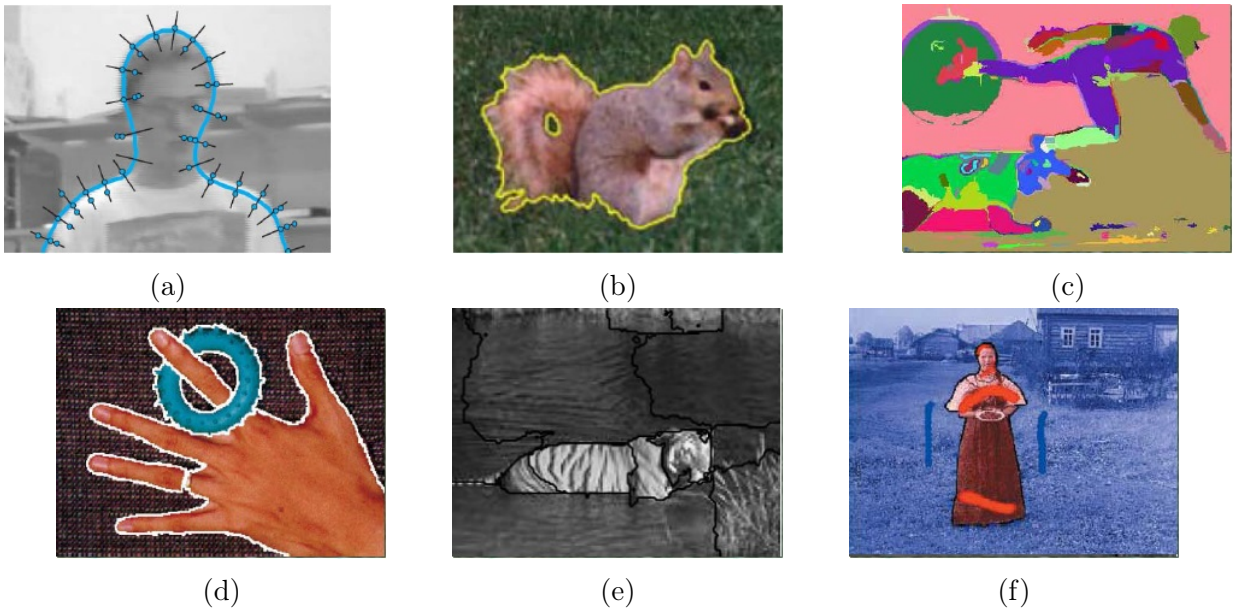


Figure 1.9: Some popular image segmentation techniques: (a) active contours (Isard and Blake 1998) c 1998 Springer; (b) level sets (Cremers, Rousson, and Deriche 2007) c 2007 Springer; (c) graph-based merging (Felzenszwalb and Huttenlocher 2004b) c 2004 Springer; (d) mean shift (Comaniciu and Meer 2002) c 2002 IEEE; (e) texture and intervening contour-based normalized cuts (Malik, Belongie, Leung et al. 2001) c 2001 Springer;(f) binary MRF solved using graph cuts (Boykov and Funka-Lea 2006) c 2006 Springer [1].

1.3 Recognition in the image

Image recognition is a term for computer technologies that can recognize certain people, animals, objects or other targeted subjects through the use of algorithms and machine learning concepts. The term image recognition is connected to computer vision, which is an overarching label for the process of training computers to see like humans, and image processing, which is a catch-all term for computers doing intensive work on image data [20].

1.3.1 Implementation of the image recognition Algorithms

We can find many uses that implement the recognition on : see the figure 1.10.

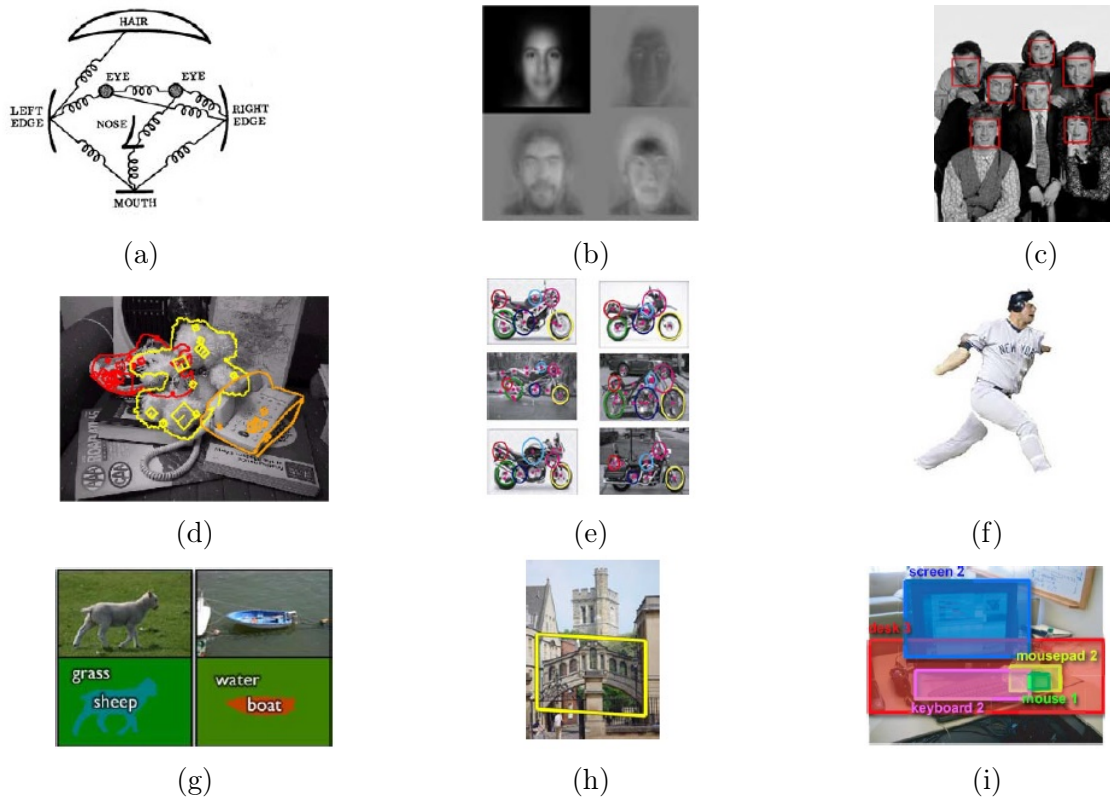


Figure 1.10: Recognition: face recognition with (a) pictorial structures (Fischler and Elschlager 1973) c 1973 IEEE and (b) eigenfaces (Turk and Pentland 1991b); (c) realtime face detection (Viola and Jones 2004) c 2004 Springer; (d) instance (known object) recognition (Lowe 1999) c1999 IEEE; (e) feature-based recognition (Fergus, Perona, and Zisserman 2007); (f) region-based recognition (Mori, Ren, Efros et al. 2004) c2004 IEEE; (g) simultaneous recognition and segmentation (Shotton, Winn, Rother et al. 2009) c2009 Springer; (h) location recognition (Philbin, Chum, Isard et al. 2007) c2007 IEEE; (i) using context (Russell, Torralba, Liu et al. 2007) [1].

1.3.1.1 The uses of images recognition

Needs that allows us to use images recognition methods and algorithm throw the commercial life or the regular one, from the most uses we have :

- Face detection.
- Object recognition.
- Species recognition.

A- Face detection

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images.[1] Face detection also refers to the psychological process by which humans locate and attend to faces in a visual scene [21].

Face-detection algorithms focus on the detection of frontal human faces. It is analogous to image detection in which the image of a person is matched bit by bit. Image matches with the image stores in database. Any facial feature changes in the database will invalidate the matching process. A lot of application which implement the face detection methods:

- Facial motion capture [22].
- facial recognition system [23].
- Photography [24].
- Marketing [25][26].

B- Object recognition

Object Recognition is a technology in the field of computer vision. It is considered to be one of the difficult and challenging tasks in computer vision.

Object Recognition refers to the capability of computer and software systems to locate objects in an image/scene and identify each object. Object Recognition has been widely used for face Recognition, vehicle Recognition, pedestrian counting, web images, security systems and driver-less cars. There are many ways object Recognition can be used as well in many fields of practice, as we see the figure 1.11.

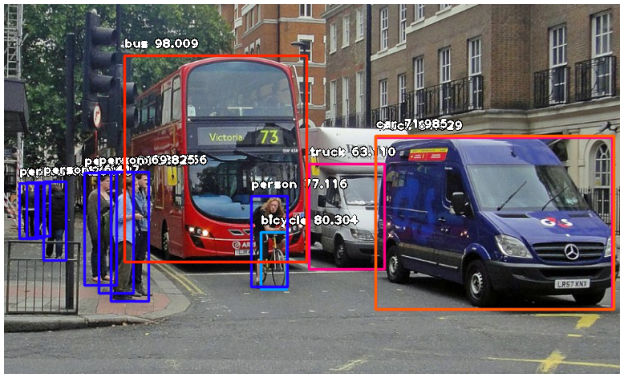


Figure 1.11: Multiple object recognition in image [27].

Object recognition methods has the following applications:

- Activity recognition [28].
- Automatic target recognition [29].
- Global robot localization [30].
- Object Counting and Monitoring [31].
- Computer-aided diagnosis [32].

C- Species recognition

Species recognition one of the most important and useful techniques during life time, it dose concern the species for everything regarding the differences of huge number of types such as animals, plants, fishes, trees, etc..., the species recognition methods will take care of identifying certain spice and classify it as the related species. Important application through this process:

- Forest species recognition [33].

- Plant species recognition [34].
- Fish species recognition [35].
- Animals species recognition [36].

1.3.1.2 Recognition in images problems

Many problems and difficulties facing the recognition in images methods and techniques, thought we could summarize as :

A- Compression

A modern trend in image storage and transmission is to use digital techniques. Digitizing a television signal results in -100 megabits per second. But channel bandwidth is expensive. So for applications such as teleconferencing, one wants to use a channel of 64 kilo-bits per second. For other applications such as videophone and mobile videophone, even lower channel bandwidths kilo-bits per second) are desirable [37].

B- Enhancement

Image enhancement is the process of adjusting digital images so that the results are more suitable for display or further image analysis. For example, you can remove noise, sharpen, or brighten an image, making it easier to identify key features [38]. Many algorithms have been devised to remove these degradation's. The difficult problem is how to remove degradation's without hurting the signal. For example, noise-reduction algorithms typically involve local, averaging or smoothing which, unfortunately, will blur the edges in the image. Adaptive methods have been investigated-e.g., smoothing less near the edges. However, they are generally effective only if the degradation is slight. A challenging problem is then how to enhance severely degraded images [37].

C- Recognition

Typically, a recognition system needs to classify an unknown input pattern into one of a set of pre-specified classes. The task is fairly easy if the number of classes is small and if all members in the same class are almost exactly the same. However, the problem can become very difficult if the number of classes is very large or if members in the same class can look very different. Thus, a most challenging problem is how to recognize generic objects.

D- Visualization

Commonly, visualization is considered as a part of computer graphics. The main task is to generate images or image sequences based on three-dimensional object and scene models. A challenging problem is how to model dynamic scenes containing nonrigid objects (such as clothing, hair, trees, waves, clouds, etc...). The models have to be realistic, and yet the computation cost has to be reasonable [37].

1.4 Characters recognition in images

1.4.1 Definition

Character recognition , often abbreviated as CR, is the mechanical or electronic conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example from a television broadcast) [39].
Character recognition is a process which allows computers to recognize written or printed characters such as numbers or letters and to change them into a form that the computer can use [40].

1.4.2 Characters recognition types

Characters recognition is generally an "offline" process, which analyses a static document. Handwriting movement analysis can be used as input to handwriting recognition [41]. Instead of merely using the shapes of glyphs and words, this technique is able to capture motions, such as the order in which segments are drawn, the direction, and the pattern of putting the pen down and lifting it. This additional information can make the end-to-end process more accurate. This technology is also known as "on-line character recognition", "dynamic character recognition", "real-time character recognition", and "intelligent character recognition".

As we can mention types :

- Optical character recognition (OCR) – targets typewritten text, one glyph or character at a time.
- Optical word recognition – targets typewritten text, one word at a time (for languages that use a space as a word divider). (Usually just called "OCR".)
- Intelligent character recognition (ICR) – also targets handwritten print-script or cursive text one glyph or character at a time, usually involving machine learning [42].
- Intelligent word recognition (IWR) – also targets handwritten print-script or cursive text, one word at a time. This is especially useful for languages where glyphs are not separated in cursive script [43].

1.4.3 Character recognition Techniques

Character recognition during been developed a lot of techniques that accomplish the purpose of getting good recognition results, well known techniques :

1.4.3.1 Pre-processing

Character recognition software often "pre-processes" images to improve the chances of successful recognition. Techniques include [44]:

- De-skew – If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counterclockwise in order to make lines of text perfectly horizontal or vertical.
- Despeckle – remove positive and negative spots, smoothing edges.
- Binarisation – Convert an image from color or greyscale to black-and-white (called a "binary image" because there are two colors). The task of binarisation is performed as a simple way of separating the text (or any other desired image component) from the background [45]. The task of binarisation itself is necessary since most commercial recognition algorithms work only on binary images since it proves to be simpler to do so [46][47].
- Line removal – Cleans up non-glyph boxes and lines [48].
- Layout analysis or "zoning" – Identifies columns, paragraphs, captions, etc. as distinct blocks. Especially important in multi-column layouts and tables.
- Line and word detection – Establishes baseline for word and character shapes, separates words if necessary.
- Script recognition – In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script [49].
- Character isolation or "segmentation" – For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.
- Normalize aspect ratio and scale [50].

Segmentation of fixed-pitch fonts is accomplished relatively simply by aligning the image to a uniform grid based on where vertical grid lines will least often intersect black areas. For proportional fonts, more sophisticated techniques are needed because whitespace between letters can sometimes be greater than that between words, and vertical lines can intersect more than one character [51].

1.4.3.2 Character recognition

There are two basic types of core CR algorithm, which may produce a ranked list of candidate characters [52].
Matrix matching involves comparing an image to a stored glyph on a pixel-by-pixel basis; it is

also known as "pattern matching", "pattern recognition", or "image correlation". This relies on the input glyph being correctly isolated from the rest of the image, and on the stored glyph being in a similar font and at the same scale. This technique works best with typewritten text and does not work well when new fonts are encountered. This is the technique the early physical photocell-based CR implemented, rather directly.
Feature extraction decomposes glyphs into "features" like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient. These features are compared with an abstract vector-like representation of a character, which might reduce to one or more glyph prototypes. General techniques of feature detection in computer vision are applicable to this type of CR, which is commonly seen in "intelligent" handwriting recognition and indeed most modern CR software [53]. Nearest neighbour classifiers such as the k-nearest neighbors algorithm are used to compare image features with stored glyph features and choose the nearest match [54].
Software such as Cuneiform and Tesseract use a two-pass approach to character recognition. The second pass is known as "adaptive recognition" and uses the letter shapes recognized with high confidence on the first pass to recognize better the remaining letters on the second pass. This is advantageous for unusual fonts or low-quality scans where the font is distorted (e.g. blurred or faded) [51].

1.4.3.3 Post-processing

CR accuracy can be increased if the output is constrained by a lexicon – a list of words that are allowed to occur in a document [44]. This might be, for example, all the words in the English language, or a more technical lexicon for a specific field. This technique can be problematic if the document contains words not in the lexicon, like proper nouns. Tesseract uses its dictionary to influence the character segmentation step, for improved accuracy [51].
The output stream may be a plain text stream or file of characters, but more sophisticated CR systems can preserve the original layout of the page and produce, for example, an annotated PDF that includes both the original image of the page and a searchable textual representation.
"Near-neighbor analysis" can make use of co-occurrence frequencies to correct errors, by noting that certain words are often seen together [55].
Knowledge of the grammar of the language being scanned can also help determine if a word is likely to be a verb or a noun, for example, allowing greater accuracy.

1.4.3.4 Application-specific optimizations

The major CR technology providers began to tweak CR systems to deal more efficiently with specific types of input. Beyond an application-specific lexicon, better performance may be had by taking into account business rules, standard expression.
This strategy is called "Application-Oriented CR" or "Customized OCR", and has been applied to OCR of license plates, invoices, screenshots, ID cards, driver licenses, and automobile

manufacturing.

1.4.4 Applications

Character recognition (CR) engines have been developed into many kinds of domain-specific CR applications, such as receipt CR, invoice CR, check CR, legal billing document CR.

They can be used for:

- Data entry for business documents, e.g. check, passport, invoice, bank statement and receipt.
- Automatic number plate recognition.
- In airports, for passport recognition and information extraction.
- Automatic insurance documents key information extraction.
- Extracting business card information into a contact list [56].
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg.
- Make electronic images of printed documents searchable, e.g. Google Books.
- Converting handwriting in real time to control a computer (pen computing).
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR [57][58][59].
- Assistive technology for blind and visually impaired users.

1.5 Conclusion

Image recognition is the process of identifying and detecting an object or feature of a digital image or video. This concept is used in many applications such as factory automation, toll monitoring and security monitoring systems. In addition, images have naturally invaded our daily lives so their treatment has become quite important in many areas.

Through the next chapter, we will present a technology that has absolutely dominated computer vision in recent years, which is the deep learning.

Chapter 2

Deep Learning

2.1 Introduction

In recent years, much of the modern innovation in image recognition is based on deep learning technology, an advanced type of machine learning and the modern wonder of artificial intelligence. The deep learning approach to image recognition involves the use of convolutional neural networks.

In this chapter, we will provide a detailed explanation of how image recognition works, as well as the Deep Learning technology that powers it.

2.2 Neural networks

2.2.1 Definition

Neural networks are a computer model that shares certain properties with the animal. brain in which many single units work in parallel, without centralized control unit. The weightings between the units are the main means of long-term calculation. storing information in neural networks. Updating weights is the primary means of neural network learning new information [60].

2.2.2 Neural Networks Architectures

The way in which the neurons of a neural network are structured is intimately linked with the learning algorithm used on the network, we focus attention to network architectures (structures). In general, we can identify three fundamentally different classes of networks architectures:

- Single-Layer Feedforward Networks.
- Multilayer Feedforward Networks.
- Recurrent Networks.

2.2.2.1 Single-Layer Feedforward Networks

Neurons are organized as layers. In the simplest As a layered network, we have a source node input layer that projects directly to an output layer of neurons (compute nodes). It is illustrated in the figure 2.1.

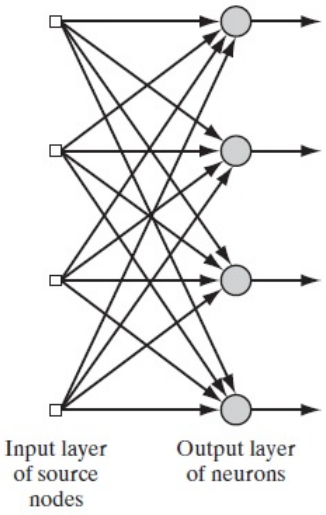


Figure 2.1: Feedforward network with a single layer of neurons [61].

2.2.2.2 Multilayer Feedforward Networks

Multilayer Feedforward Networks is distinguished by the presence of one or more hidden layers, The architectural chart of the figure 2.2 illustrates the structure of a multilayered anticipation neural network. for the case of a single hidden layer.

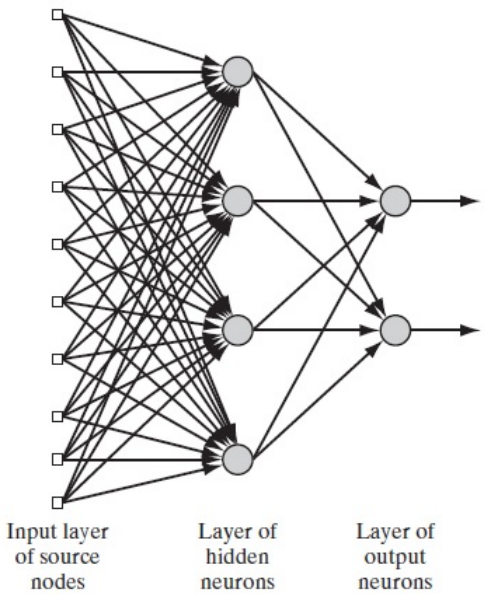


Figure 2.2: Fully connected feedforward network with one hidden layer and one output layer [61].

2.2.2.3 Recurrent Networks

A current neural network is distinguished from an anticipatory neural network in that it has at least one feedback loop [62]. For example, a recurrent network may consist of a single layer of neurons with each neuron supplying its output signal to the inputs of all other neurons, as shown in the architectural graph of figure 2.3.

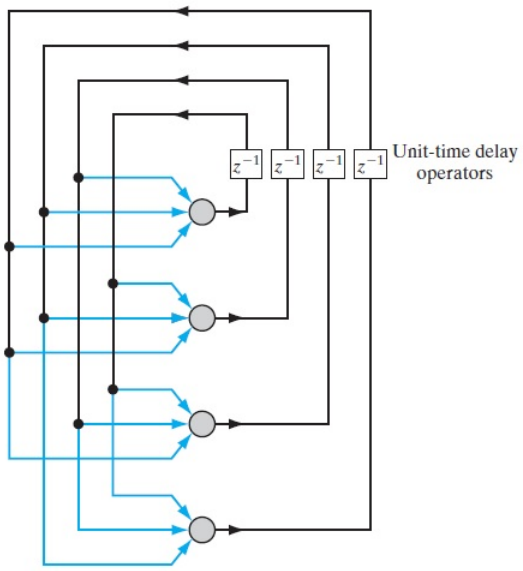


Figure 2.3: Recurrent network with no self-feedback loops and no hidden neurons.[62]

On the figure 2.4, we illustrate another class of recurrent networks with hidden neurons. The illustrated feedback connections also come from hidden neurons. from the output neurons.

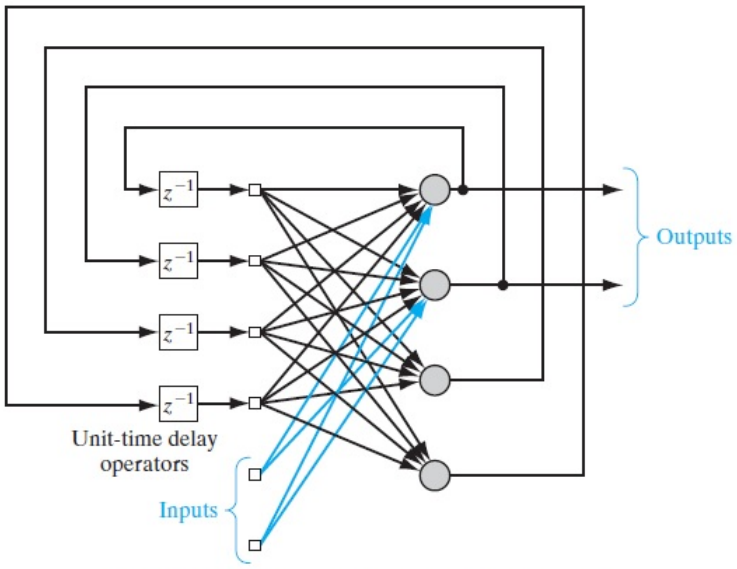


FIGURE 18 Recurrent network with hidden neurons.

Figure 2.4: Recurrent network with hidden neurons.[62]

2.2.3 Automatic learning types

Very important and essential type for learning, we must quote them:

- Supervised learning.
- Unsupervised learning.
- Reinforcement learning.
- Semi-supervised learning.

2.2.3.1 Supervised learning

Supervised learning is a prediction function that consists of annotated examples and their turn which constitute a learning base methods. The goal of supervised learning is then to build, from the learning base, classifiers, or ranking functions. Such a function allowed us to recognize an object we have-descriptions, figure 2.5.

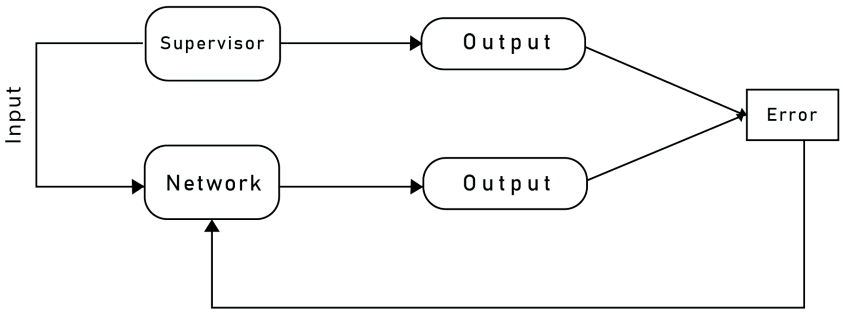


Figure 2.5: Supervised Learning.

2.2.3.2 Unsupervised learning

Unsupervised learning the reverse of supervised learning. it is a question of finding a structure which corresponds to the problem to be predicted but starting from the non-annotated data, it is not possible to make the algorithm a desired output. the absence of annotated data or learning base which means unsupervised learning, figure 2.6.

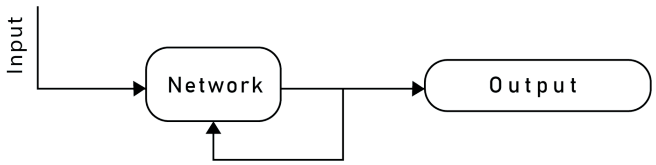


Figure 2.6: UnSupervised Learning.

2.2.3.3 Reinforcement learning

For reinforcement learning, for an autonomous agent (robot, etc.), to learn the actions to be taken, from experiments, so as to optimize a quantitative reward over time. The agent is immersed in an environment, and makes his decisions based on his current state. In return, the environment provides the agent with a reward, which can be positive or negative. The agent searches, through iterated experiments, a decision-making behavior (called strategy or policy, and which is a function associating with the current state the action to be performed) optimal, in that it maximizes the sum of rewards over time [63]. See figure 2.7

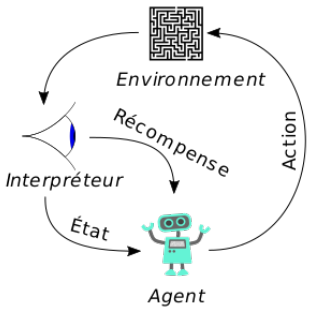


Figure 2.7: Reinforcement learning [64].

2.2.3.4 Semi-supervised learning

Semi-supervised learning uses a set of labeled and untagged data, so it is between supervised learning that uses only labeled data and unsupervised learning that uses only non-labeled [65]. Semi-supervised learning aims to solve problems of pattern recognition with a small amount of data labeled for a large number of points without labels. These latter points used to unravel the underlying structure of the data it is relevant to use similarity graphs to represent this structure [66]. The use of untagged data in combination with tagged data has been shown to significantly improve the quality of learning. Another benefit is that data labeling requires a

human user. When data sets become very large, this operation can be tedious. In this case, semi-supervised learning, which requires only a few labels, is of obvious practical interest [65].

2.3 Deep Learning

2.3.1 Definition

Deep learning is a special type of machine learning that reaches great power and flexibility in learning to represent the world as a nested hierarchy of concepts, each concept being defined in relation to simpler concepts and more abstract representations calculated in less abstract terms [67].

2.3.2 Deep Learning Architectures

This one goes even the major architecture of deep learning as follows:

- Unsupervised Pretrained Networks (UPNs).
- Recurrent Neural Networks.
- Recursive Neural Networks.
- Convolutional Neural Networks (CNNs).

2.3.2.1 Unsupervised Pretrained Networks (UPNs)

In this type we will deal with 3 architecture specify:

- Autoencoders.
- Deep Belief Networks (DBNs).
- Generative Adversarial Networks (GANs).

A- Autoencoders

Autoencoders are simple learning circuits that aim to turn inputs into outputs with the least amount of distortion. Although conceptually simple, they play an important role in machine learning. Auto-encoders were introduced in the 1980s by Hinton and the PDP to tackle the problem of backward propagation without a teacher, using the input data as a teacher [68]. This is summarized in the figure 2.8.

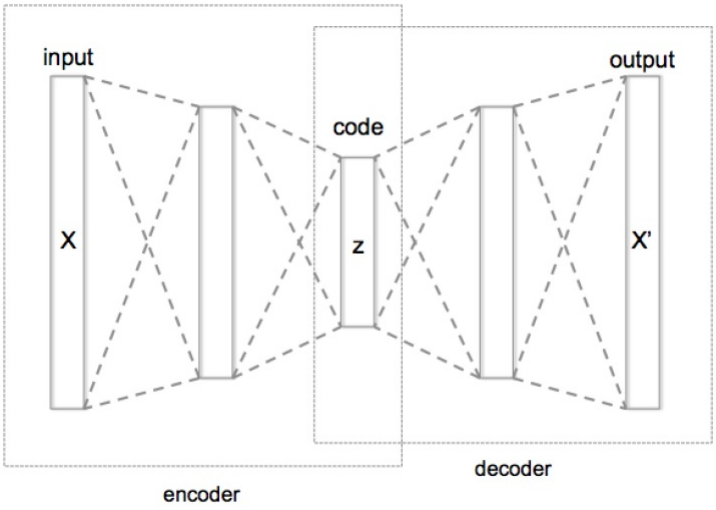


Figure 2.8: Autoencoders Structures [68].

B- Deep Belief Networks (DBNs)

Deep Belief Networks (DBNs) is a stack of many Restricted Boltzmann Machines (RBMs) such that the hidden layer of each RBM acts as a visible layer for the next RBM. The visible layer of the first RBM is also a visible layer of DBN and all other layers are masked layers. DBN is formed by forming one RBM at a time. Once, the first RBM has been formed, the samples are simply routed to it and the output produced at the hidden layer of it is served as input to the visible layer of the next RBM and so on. This is called a pre-formation layer of DBN [69]. The figure 2.9 shows the basic architecture of the DBN used in the proposed DBN-WP.

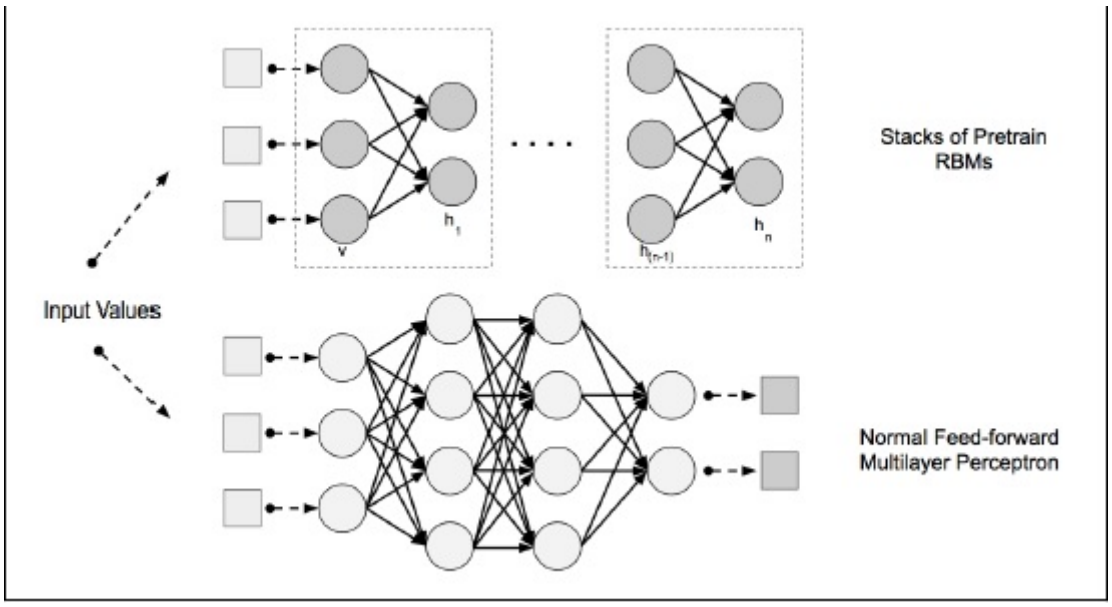


Figure 2.9: DBN architecture [60].

C- Generative Adversarial Networks (GANs)

Generic adversarial networks (GAN) are a emergent technique both semi-supervised and unsupervised learning. They achieve this implicitly modeling of large data distributions. Offers in 2014 [70]. figure 2.10 could gives us general idea about the operation of the GANs.

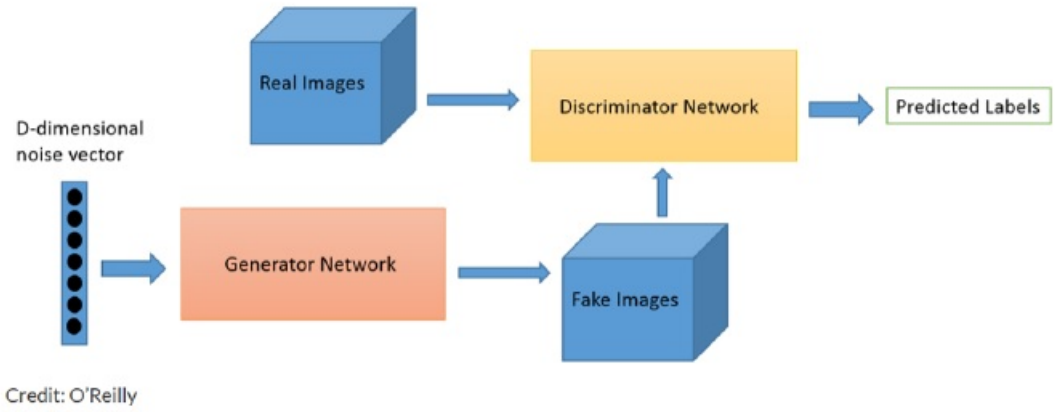


Figure 2.10: Basic idea behind GANs [71].

2.3.2.2 Recurrent Neural Networks (RNNs)

Recurrent neural networks belong to the family of feed-forward neural networks. They are different from other feed-forward networks in their ability to send information over time [60]. Technical RNNs have been applied to a wide variety of problems [72]. Recurrent networks can generally be classified into globally recurrent networks, in which the feedback connections between each neuron are allowed, and networks recurrent locally, globally and prospectively with the dynamics performed within the neural models [73]. RNNs have architictures to vary the figure 2.11 shows an example of the RNN completely connected to each other.

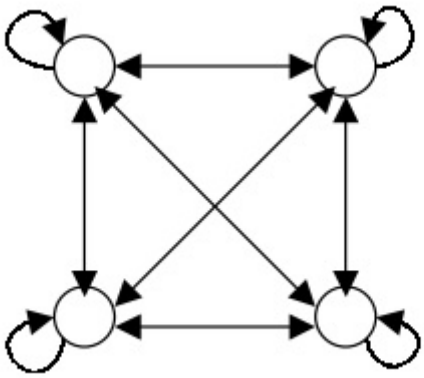


Figure 2.11: An example of a fully connected recurrent neural network [72].

The figure 2.12 shows an example of the RNN has simple related and partial, Although some nodes are part of a feedforward structure.
The weights are represented by (C1 and C2).

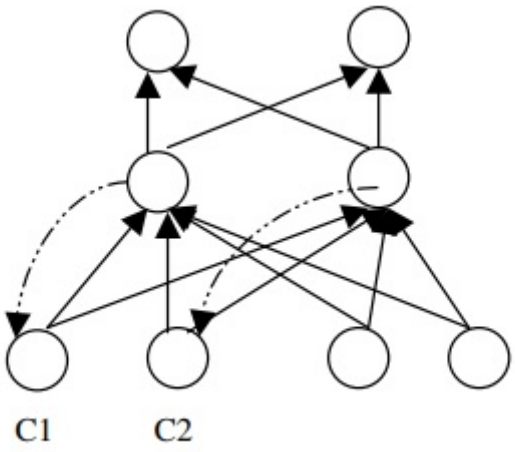


Figure 2.12: An example of a simple recurrent network [72].

2.3.2.3 Recursive Neural Networks (RNNs)

Recursive neural networks are nonlinear adaptive models that are: able to learn deep structured information [74]. Seem very suitable for classification.
Recursive neural networks have been used in some applications, including: sentiment analysis, image description and paraphrase detection [75]. The figure 2.13 we can even the simplest architecture for RNNs.

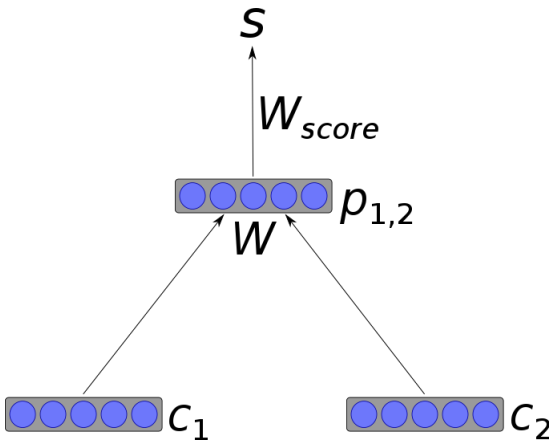


Figure 2.13: A simple recursive neural network architecture [76].

2.3.2.4 Convolutional Neural Networks (CNNs)

CNNs is a type of acyclic artificial neural network (feed-forward), in which the connection pattern between neurons is inspired by the animal's visual cortex. Neurons in this region of the brain are arranged so that they correspond to overlapping regions when paving the visual field [77].
The goal of a CNN is to learn higher-order features in the data via convolutions. They are well suited to object recognition with images and consistently top image classification competitions. They can identify faces, individuals, street signs, platypuses, and many other aspects of visual data. CNNs overlap with text analysis via optical character recognition, but they are also useful

when analyzing words6 as discrete textual units. They're also good at analyzing sound. The figure 2.14 illustrates, CNNs are good at building position and (somewhat) rotation invariant features from raw image data.

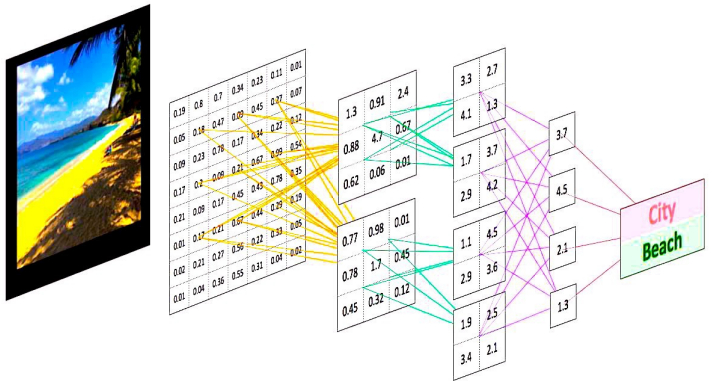


Figure 2.14: CNNs and computer vision [60].

A- Biological Inspiration

The biological inspiration for CNNs is the visual cortex in animals. The cells in the visual cortex are sensitive to small sub regions of the input. We call this the visual field (or receptive field). These smaller sub regions are tiled together to cover the entire visual field. The cells are well suited to exploit the strong spatially local correlation found in the types of images our brains process, and act as local filters over the input space. There are two classes of cells in this region of the brain. The simple cells activate when they detect edge-like patterns, and the more complex cells activate when they have a larger receptive field and are invariant to the position of the pattern [60].

B- CNN Architecture Overview

CNNs transform the input data from the input layer through all connected layers into a set of class scores given by the output layer. There are many variations of the CNN architecture, but they are based on the pattern of layers, as demonstrated in figure 2.15

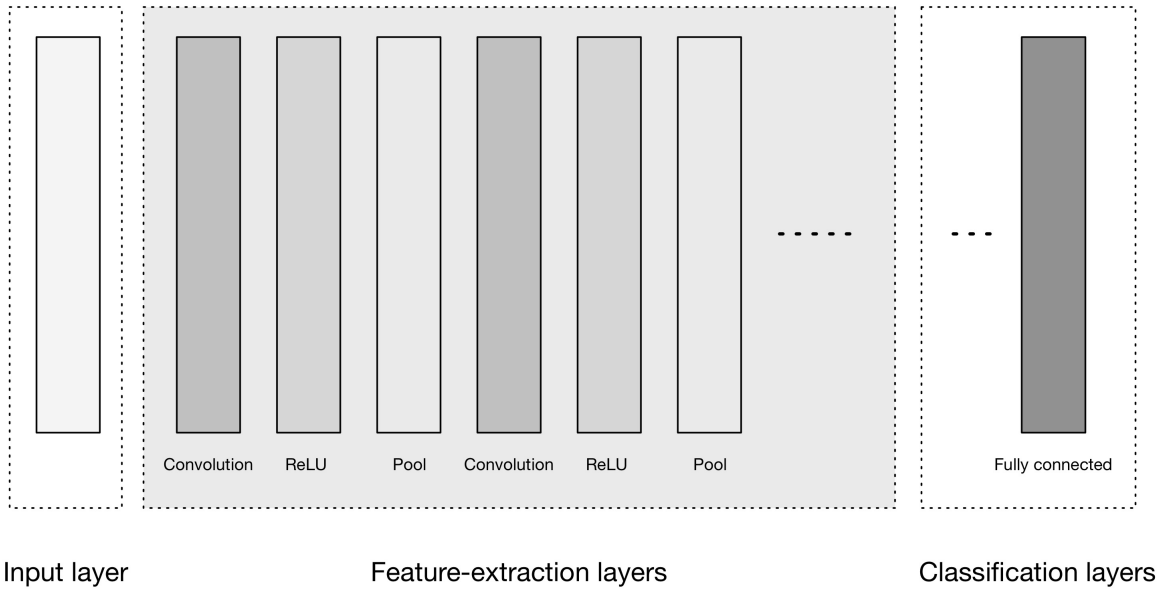


Figure 2.15: High-level general CNN architecture [60].

Input Layers

Input layers are where we load and store the raw input data of the image for processing in the network. This input data specifies the width, height, and number of channels. Typically, the number of channels is three, for the RGB values for each pixel.

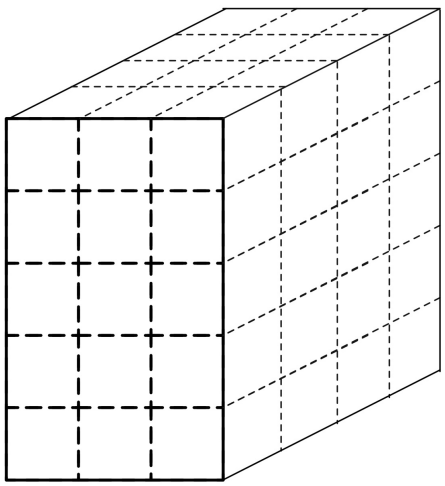


Figure 2.16: Input layer 3D volume [60].

Convolutional Layers

Convolutional layers are considered the core building blocks of CNN architectures. As figure 2.17 illustrates, convolutional layers transform the input data by using a patch of locally connecting neurons from the previous layer. The layer will compute a dot product between the region of the neurons in the input layer and the weights to which they are locally connected in the output layer.

Convolution : Is defined as a mathematical operation describing a rule for how to merge two sets of information. Figure 2.18 is known as the feature detector of a CNN. The input to a convolution can be raw data or a feature map output from another convolution. It is often interpreted as a filter in which the kernel filters input data for certain kinds of information, for example, an edge kernel lets pass through only information from the edge of an image [60].

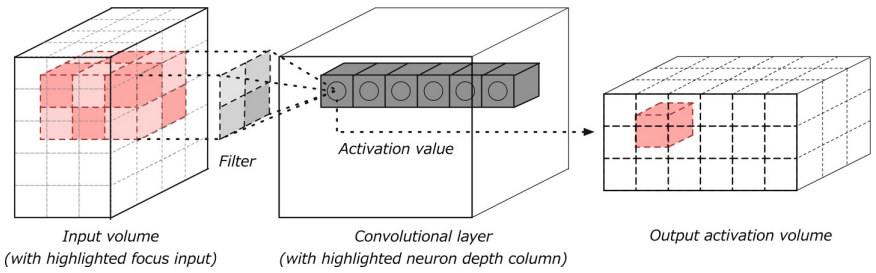


Figure 2.17: Convolution layer with input and output volumes [60].

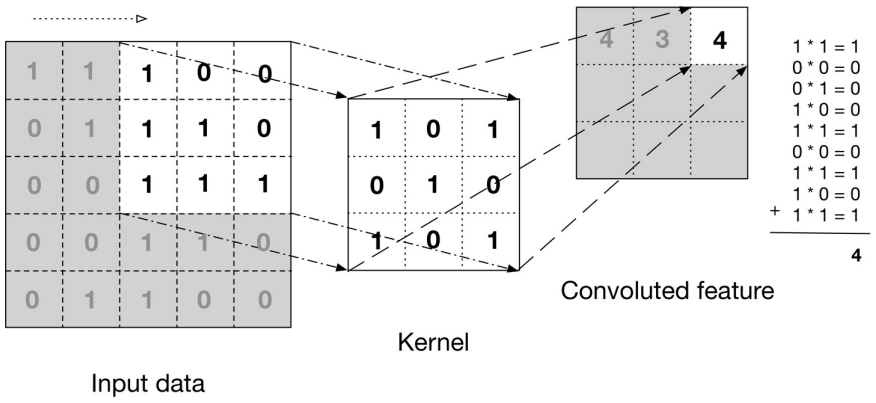


Figure 2.18: The convolution operation [60].

Filters : The parameters for a convolutional layer configure the layers set of filters. Filters are a function that has a width and height smaller than the width and height of the input volume. Filters (e.g, convolutions) are applied across the width and height of the input volume in a sliding window manner, as demonstrated in figure 2.18. Filters are also applied for every depth of the input volume. We compute the output of the filter by producing the dot product of the filter and the input region [60].

Activation maps : We slide each filter across the spatial dimensions (width, height) of the input volume during the forward pass of information through the CNN. This produces a two-dimensional output called an activation map for that specific filter. Figure 2.19 depicts how this activation map relates to our previously introduced concept of a convoluted feature. The activation map on the right in figure 2.19 is rendered differently to illustrate how convolutional activation maps are commonly rendered in the literature.

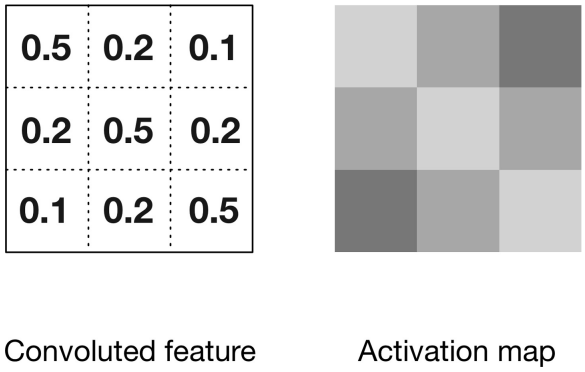


Figure 2.19: Convolutions and activation maps [60].

Parameter sharing : CNNs use a parameter-sharing scheme to control the total parameter count. This helps training time because we'll use fewer resources to learn the training dataset. To implement parameter sharing in CNNs, we first denote a single two-dimensional slice of depth as a depth slice. We then constrain the neurons in each depth slice to use the same weights and bias. This gives us significantly fewer parameters (or weights) for a given convolutional layer [60].

Convolutional layer hyper-parameters : Following are the hyper-parameters that dictate the spatial arrangement and size of the output volume from a convolutional layer are:

- Filter (or kernel) size (field size).
- Output depth.
- Stride [78].
- Zero-padding [79].

Pooling-layers : Pooling-layers are commonly inserted between successive convolutional layers. We want to follow convolutional layers with pooling layers to progressively reduce the spatial size (width and height) of the data representation. Pooling layers reduce the data representation progressively over the network and help control over-fitting. The pooling layer operates independently on every depth slice of the input [60]. Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called sub-sampling or down-sampling which reduces the dimensionality of each map but retains the important information. Spatial pooling can be of different types [80] :

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling take the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

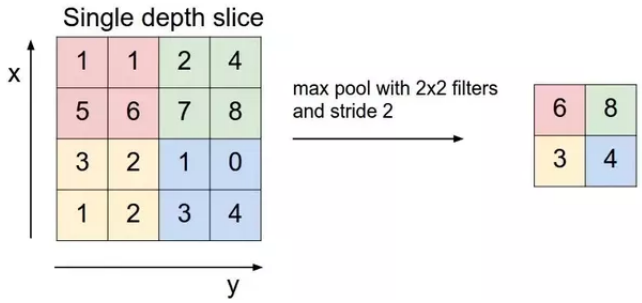


Figure 2.20: Max pooling [80].

Fully connected layer

We use this layer to compute class scores that we'll use as output of the network (e.g., the output layer at the end of the network). The dimensions of the output volume is [1 * 1 * N], where N is the number of output classes we're evaluating. This layer has a connection between all of its neurons and every neuron in the previous layer.

Fully connected layers have the normal parameters for the layer and hyperparameters. Fully connected layers perform transformations on the input data volume that are a function of the activation's in the input volume and the parameters (weights and biases of the neurons).

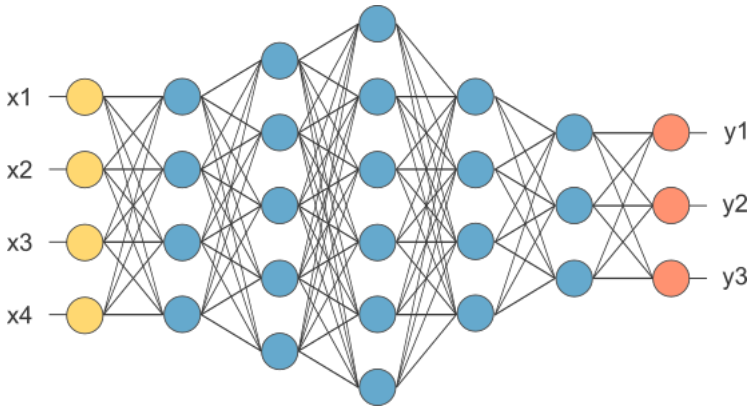


Figure 2.21: After pooling layer, flattened as FC layer [80].

In the above diagram, feature map matrix will be converted as vector (x1, x2, x3, . . .). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc...,

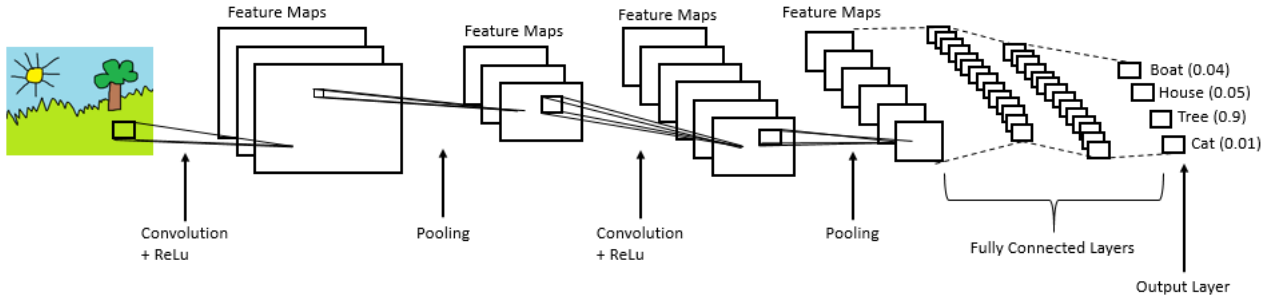


Figure 2.22: Complete CNN architecture [80].

C- Popular architectures of CNNs

Following is a list of some of the more popular architectures of CNNs.

- LeNet [81].
  - One of the earliest successful architectures of CNNs
  - Developed by Yann Lecun
  - Originally used to read digits in images
- AlexNet [82].
  - Helped popularize CNNs in computer vision
  - Developed by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton
  - Won the ILSVRC 2012
- ZF Net [83].
  - Won the ILSVRC 2013
  - Developed by Matthew Zeiler and Rob Fergus
  - Introduced the visualization concept of the Deconvolutional Network
- GoogLeNet [84].
  - Won the ILSVRC 2014
  - Developed by Christian Szegedy and his team at Google
  - Codenamed “Inception,” one variation has 22 layers
- VGGNet [85].
  - Runner-Up in the ILSVRC 2014
  - Developed by Karen Simonyan and Andrew Zisserman
  - Showed that depth of network was a critical factor in good performance
- ResNet [86].
  - Trained on very deep networks (up to 1,200 layers)
  - Won first in the ILSVRC 2015 classification task

2.3.3 Deep Learning Application Domain

2.3.3.1 Bioinformatics

Bioinformatics is a multi-disciplinary field of biotechnology research More generally, bioinformatics is the application of statistics and computer science to biological science. biotechnology where biologists, physicians, computer scientists, mathematicians, physicists and bioinformaticians work together to solve a scientific problem posed by biology.
The term bioinformatics can also describe, by abuse of language, all the computer applications resulting from this research. The use of the term bioinformatics is documented for the first time in 1970 in a publication [87]. By Paulien Hogeweg and Ben Hesper (Utrecht University, The Netherlands), with reference to the study of information processes in Bioinformatic systems [88].
This domain ranges from genome analysis to modeling the evolution of an animal population in a given environment, including molecular modeling, image analysis, genome assembly and reconstruction. phylogenetic trees (phylogeny). This discipline constitutes the "in silico biology", by analogy with in vitro or in vivo [89].

2.3.3.2 Health

Deep learning has been particularly effective in medical imaging, thanks to the availability of high-quality images and the ability of convolutional neural networks to classify images. For example, deep learning can be as effective (or even more effective) than a dermatologist in classifying skin cancers. Several providers have already received FDA approval for the use of deep learning algorithms to perform diagnostics, including image analysis for oncology and retinal diseases. Deep learning is also making major strides in improving the quality of health services by anticipating medical events through electronic medical records [90].

2.3.3.3 Robotics

A large number of recent developments in robotics have been achieved through artificial intelligence and deep learning. For example, AI has allowed robots to detect and react to their environment. This ability extends the range of their functions, from moving houses on the floors to sorting and handling irregular, fragile or tangled objects. Picking a strawberry is an easy task for humans, but much more complicated for robots. The progress of AI will evolve the capabilities of robots. The evolution of AI will allow robots of the future to become better assistants for man. They will not only be used to understand and answer questions, as is the case for some robots. They will also be able to respond to voice commands and gestures, and even anticipate the next move of a person with whom they work. Today, collaborative robots are already working alongside humans to perform separate tasks tailored to their abilities [91].

2.3.3.4 Visual and vocal recognition

In general, a method that contains different method and different use for example, visual recognition of a road sign that are elements of road signs placed on the side of roads. They designate both the device on which is implanted a road signal and the signal itself, by a robot or a car autonomous [92].
Automatic speech recognition (often improperly called speech recognition) is a computer-based technique that allows the human voice captured by a microphone to be analyzed for transcription as machine-readable text [93].

2.3.4 The challenges of deep learning

2.3.4.1 The quantity of data used

Deep learning algorithms are trained to learn gradually using data. Large sets of data are needed for the machine to deliver the desired results. Since the human brain needs many experiences to learn and derive information, the artificial artificial neural network requires a large amount of data. The more powerful the abstraction you want, the more parameters you need to adjust, and more settings require more data.

2.3.4.2 The learning time too expensive

Once the datasets are in hand, their use to form deep learning networks may require several days on large groups of processors and GPUs, emerging techniques such as transfer learning and Generative adversarial networks are promising to overcome this challenge.

2.3.4.3 Optimization of the hyper-parameter

One of the reasons deep learning works so well is the large number of interconnected neurons, or free parameters, that capture subtle nuances and variations in the data. However, this also means that it is more difficult to identify hyper-parameters, parameters whose values must be set before training. The process is more an art than science. There is also a risk of over-adaptation of data, especially when the number of parameters greatly exceeds the number of independent observations.

2.3.4.4 Difficulty in understanding how information arrives

Because of the large number of layers, nodes, and connections, it is difficult to understand how deep learning networks get to information. While this may not be so important in applications such as photo tagging on social media, understanding of the decision-making process becomes very important in critical applications such as predictive maintenance or decision making. clinical.

2.3.4.5 Deep learning is sensitive

Deep learning networks are very sensitive to the butterfly effect - small variations in the input data can lead to radically different results, making them unstable as a result. This instability also opens up new attack surfaces for hackers. In so-called conflicting attacks, researchers have shown that, by adding an imperceptible amount of noise, it is possible to mislead deep learning networks to result in completely incorrect information. everything without even accessing the system.

2.3.5 Related works

2.3.5.1 Image colorization

In image colorization, the goal is to produce a colored image given a grayscale input image. This problem is challenging because it is multimodal – a single grayscale image may correspond to many plausible colored images. As a result, traditional models often relied on significant user input alongside a grayscale image [94].

Recently, deep neural networks have shown remarkable success in automatic image colorization – going from grayscale to color with no additional human input. This success may in part be due to their ability to capture and use semantic information (i.e. what the image actually is) in colorization, although we are not yet sure what exacly makes these types of models perform so well.

Tung Nguyen, Kazuki Mori and Ruck Thawonmas, proposed a method using deep learning techniques for colorizing grayscale images. By utilizing a pre-trained convolutional neural network, which is originally designed for image classification, able to separate content and style of different images and recombine them into a single image [95].

2.3.5.2 Scene Labelling

Each pixel is labelled with the category of the object it belongs to in scene labelling. Clement Farabet et al proposed a method using a multiscale convolutional network that yielded record accuracies on the Sift Flow Dataset (33 classes) and the Barcelona Dataset (170 classes) and near-record accuracy on Stanford Background Dataset (8 classes). Their method produced 320 X 240 image labelling in under a second including feature extraction [96].

2.3.5.3 Image Classification

Compared with other methods CNNs achieve better classification accuracy on large scale datasets due to their capability of joint feature and classifier learning. Krizhevsky et al. develop the AlexNet and achieve the best performance in ILSVRC 2012. Following the success of the AlexNet, several works made significant improvements in classification accuracy by reducing filter size 32 or expanding the network depth [82].

2.3.5.4 Document Analysis

Many traditional handwriting recognizers [97]. Use the sequential nature of the pen trajectory by representing the input in the time domain. However, these representations tend to be sensitive to stroke order, writing speed and other irrelevant parameters. This system AMAP preserves the pictorial nature of the handwriting images. This model uses a combination of CNNS, Markov models, EM algorithm and encoding of words into low resolution images to recognize handwriting [98]. Using this system error rates dropped to 4.6per and 2.0per from 5per for word and character errors respectively.

2.3.5.5 Optical Character Recognition

OCR is used to identify the character from human written text. To recognize the text segmentation of character is important stage. So here, we addressed different techniques to recognize the character. This document also presents comparison of different languages for character and numeral recognition with its accuracy achieved by different writer. Segmentation problem of each language were different also handwritten character was also varied user to user, so it is necessary to make OCR systems more effective and accurate for segmentation. Comparative study concludes that deep learning technique gives good segmentation and gives better result in case with large dataset compares to other techniques [99].

2.3.5.6 Text Classification

NLP tasks deal with sentences and documents which are represented as a matrix at the input. Each row of a matrix corresponds to a token, which essentially is a word or in some approaches a character. Thus each row is a vector that represents a token. These vectors are generally low-dimensional representations called as word embedding. Word embedding is a set of language modelling and feature learning technique in NLP where words from the vocabulary are mapped to vectors of real-numbers in a low-dimensional space [100].

2.4 Conclusion

The field of computer vision is moving from statistical methods to deep learning methods of neural networks. There are still many difficult problems to solve in computer vision. Nevertheless, deep learning methods provide state-of-the-art results on specific problems. In this chapter, we have detailed the deep learning technology and how the performance of image recognition has exploded with the arrival of deep learning by presenting some related work.

Based on our study of this area, we found that existing work has certain limitations despite the benefits that offer them. This prompted us to propose our contribution, which will be described in the following chapter.

Chapter 3

Design, implementation and results

3.1 Introduction

Character recognition has attracted considerable attention in recent decades. Researchers have made significant breakthroughs in this area with the rapid development of deep learning algorithms and the most successful applications in this field. So far, they are using neural network-based methods. The convolutional neural networks have been very successful in various image recognition applications, including those of handwritten character recognition.

In our project, we were interested in creating a character recognition system that uses image processing methods based on deep learning. To achieve our goal, we propose in this chapter new architecture applied in the convolutional neuron network (CNN) method.

3.2 System Design

3.2.1 Methodology

In our system, we are going to use few of image processing methods such as pre-processing, adjustments, also the data augmentation etc.., also we gonna use an appropriate supervised deep learning method named Convolution neural network(CNN), which gives us big benefits in image recognition.
This technique allows us to make character recognition which we want to accomplish by creating and designing a descent training architecture seeking for better accuracy results, using benchmarks datasets for training session, which this method takes as an input in sort of images stored and organised as classes.

3.2.2 Global system design

Generally, Our system character recognition in images will be following a certain steps, as we represent the global architecture in figure 3.1.

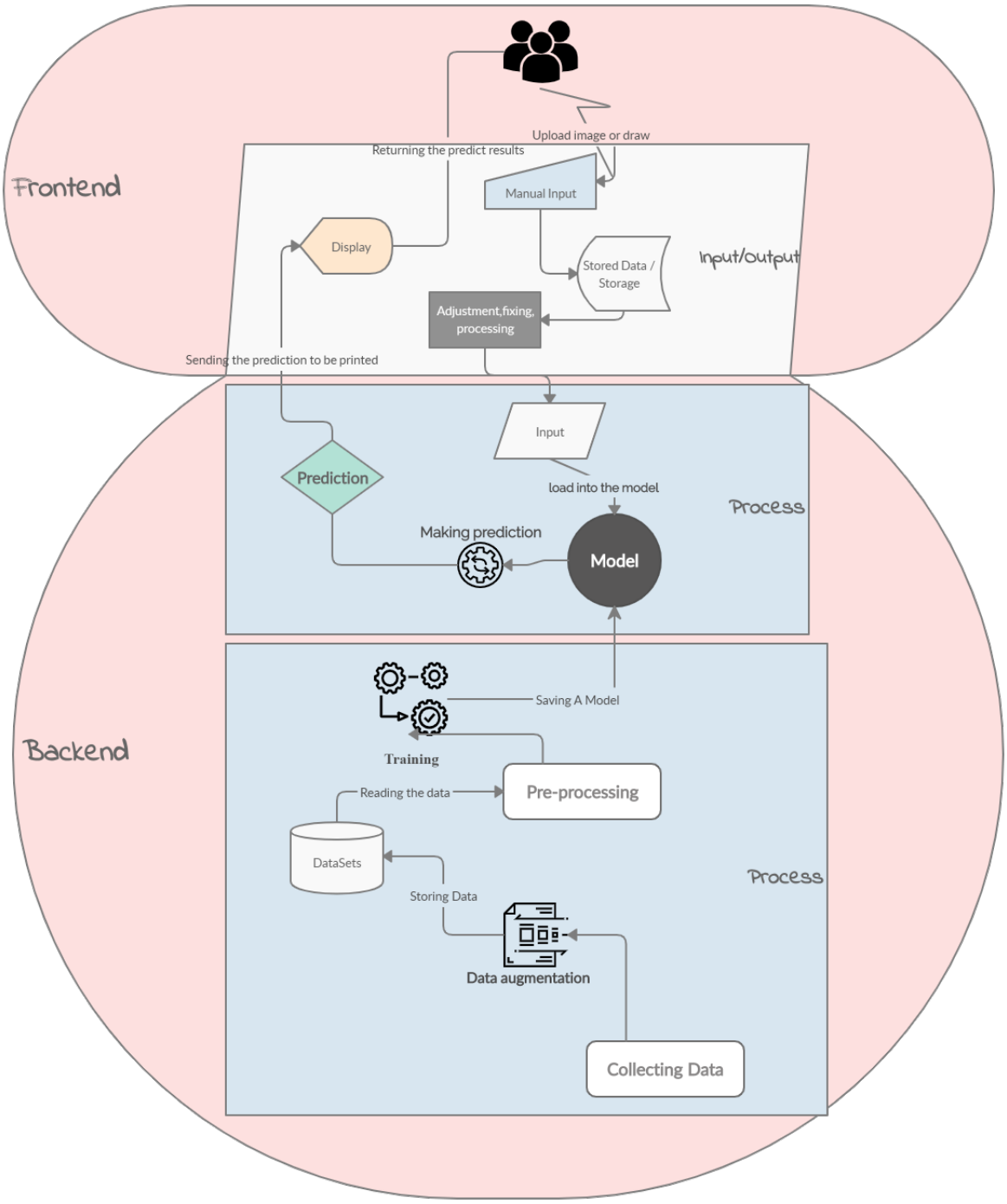


Figure 3.1: Global system Design.

As we noticed in the previous system architecture, two phases are part of system :

the front-end: (User Interface) which is the important phase for the user where the interaction and the requests of files are done, also the displaying and the storage of images and results after getting them from the back-end phase.

the Back-end: the most important phase in the system, during this process we will do the data collecting including data pre-processing and augmentation, and then the training phase which gives us as an output a model that we gonna use later-on the prediction phase, the prediction model takes as input images and then results as output.

3.2.3 Detailed system design

3.2.3.1 Back-end

The back-end phase its the main body of the system where we gonna pass thought important steps described in following :

1. The collection and Preparation of datasets.
2. Pre-processing the datasets.
3. Training.
4. Use of the output model.

Also the figure 3.2 illustrate the back-end phase

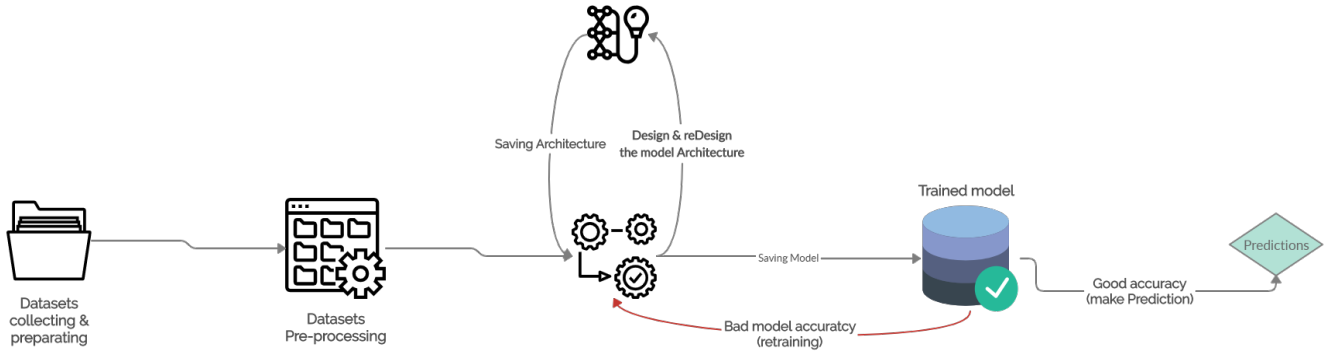


Figure 3.2: Back-end system architecture.

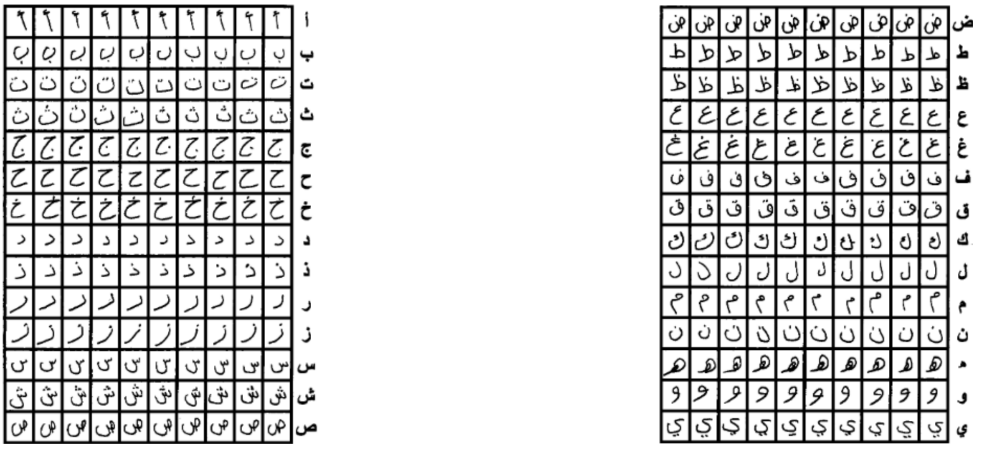
A- The collection and Preparation of datasets

In our system we are going to work on three different type of characters (handwritten Latin characters, optic Latin characters and Arabic handwritten characters), for that we need to collect and prepare three different datasets.

Arabic Characters

We are going to use a datasets created by Ahmed El-Sawy, Mohamed Loey Benha University and Hazem EL-Bakry Mansoura University, The dataset is composed of 16,800 characters written by 60 participants, the age range is between 19 to 40 years, and 90% of participants are right-hand. Each participant wrote each character (from 'alef' to 'yeh') ten times on two forms as shown in figure. 3.3a & 3.3b. The forms were scanned at the resolution of 300 dpi. Each block is segmented automatically using Matlab 2016a to determining the coordinates for each block. The database is partitioned into two sets: a training set (13,440 characters to 480 images per class) and a test set (3,360 characters to 120 images per class). Writers of training set and test set are exclusive. Ordering of including writers to test set are randomized to make sure that writers of test set are not from a single institution (to ensure variability of the test set). Creating the proposed database presents more challenges because it deals with many issues such as style of writing, thickness, dots number and position. Some characters have different

shapes while written in the same position. For example the teh character has different shapes in isolated position [101].



(a) (b)

Figure 3.3: Data collection for Arabic characters [101].

Handwritten Latin Characters

We used a standard benchmark dataset for learning, classification and computer vision systems, labeled as EMNIST created by Gregory Cohen, Saeed Afshar, Jonathan Tapson, André van Schaik.

The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format and dataset structure that directly matches the MNIST dataset [102].

The dataset is provided in two file formats. Both versions of the dataset contain identical information, and are provided entirely for the sake of convenience. The first dataset is provided in a Matlab format that is accessible through both Matlab and Python (using the scipy.io.loadmat function). The second version of the dataset is provided in the same binary format as the original MNIST dataset [103].

There are six different splits provided in this dataset. A short summary of the dataset is provided below:

- EMNIST ByClass: 814,255 characters. 62 unbalanced classes.
- EMNIST ByMerge: 814,255 characters. 47 unbalanced classes.
- EMNIST Balanced: 131,600 characters. 47 balanced classes.
- EMNIST Letters: 145,600 characters. 26 balanced classes.
- EMNIST Digits: 280,000 characters. 10 balanced classes.
- EMNIST MNIST: 70,000 characters. 10 balanced classes.

The full complement of the NIST Special Database 19 is available in the ByClass and ByMerge splits. The EMNIST Balanced dataset contains a set of characters with an equal number of

samples per class. The EMNIST Letters dataset merges a balanced set of the uppercase and lowercase letters into a single 26-class task. The EMNIST Digits and EMNIST MNIST dataset provide balanced handwritten digit datasets directly compatible with the original MNIST dataset. the most relaible for us and the one we gonna use its EMNIST Balanced.

	Type	No.Classes	Training	Testing	Total
By Class	Digits	10	344,307	58,646	402,953
	Uppercase	26	208,363	11,941	220,304
	Lowercase	26	178,998	12,000	190,998
	Total	62	**731,668**	**82,587**	**814,255**
By Merge	Digits	10	344,307	58,646	402,953
	Letters	37	387,361	23,941	411,302
	Total	47	**731,668**	**82,587**	**814,255**
MNIST	Digits	10	60,000	10,000	70,000

Table 3.1: Breakdown of the number of available training and testing samples in the NIST special database 19 using the original training and testing splits [103].

Table 3.1 Shows the breakdown of the original training and testing sets specified in the releases of the NIST Special Database 19. Both the By Class and By Merge hierarchies contain 814,255 handwritten characters consisting of a suggested 731,668 training samples and 82,587 testing samples. It should be noted however, that almost half of the total samples are handwritten digits.

Name	Classes	No.Training	No.Testing	Validation	Total
Complete	62	697,932	116,323	No	814,255
Merge	47	697,932	116,323	No	814,255
Balanced	47	112,800	18,800	Yes	131,600
Digits	10	240,000	40,000	Yes	280,000
Letters	37	88,800	14,800	Yes	103,600
MNIST	10	60,000	10,000	Yes	70,000

Table 3.2: Structure and organization of the EMNIST datasets[103].

Table 3.2 Contains a summary of the EMNIST datasets and indicates which classes contain a validation subset in the training set. In these datasets, the last portion of the training set, equal in size to the testing set, is set aside as a validation set. Additionally, this subset is also

balanced such that it contains an equal number of samples for each task. If the validation set is not to be used, then the training set can be used as one contiguous set.

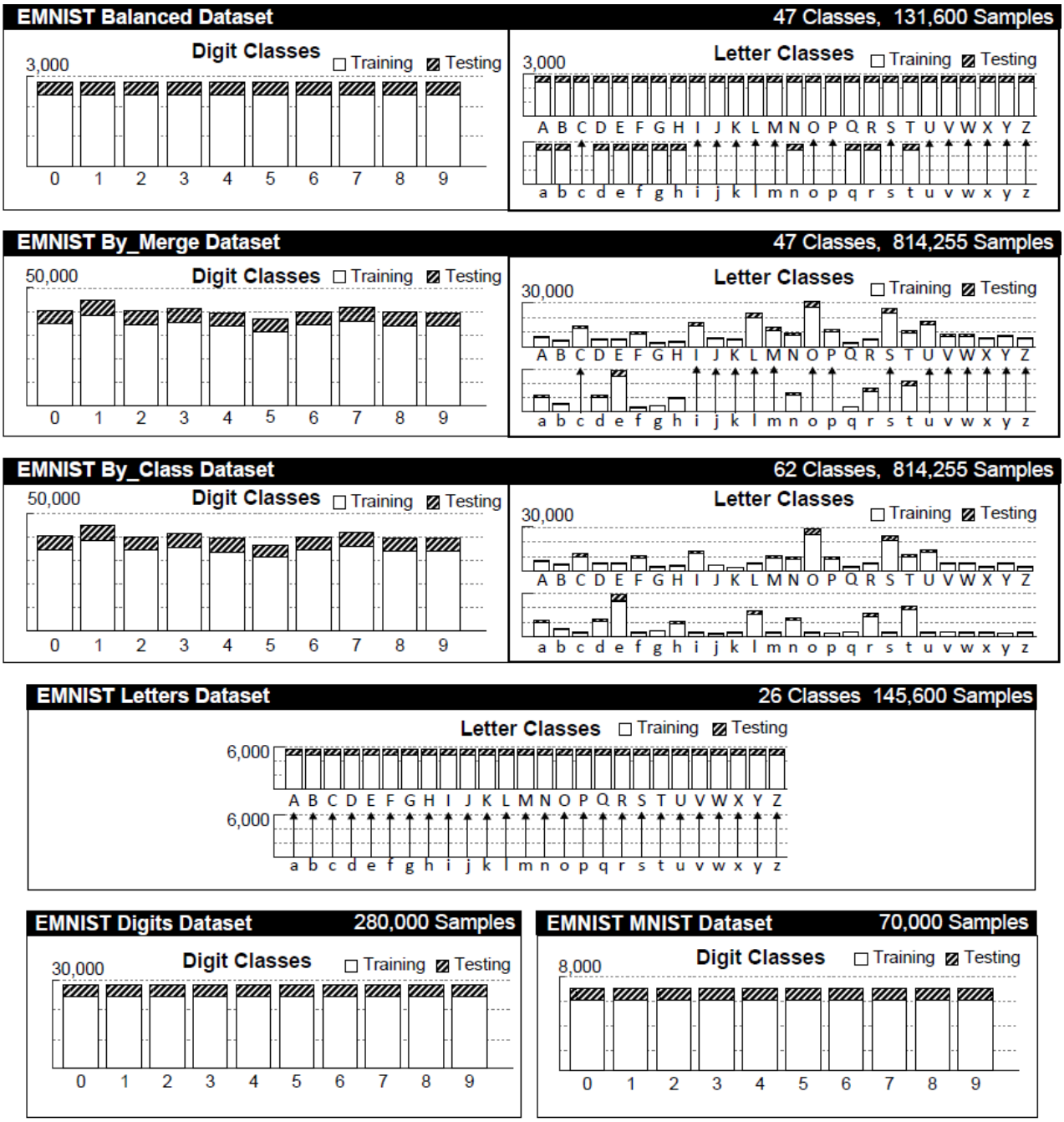


Figure 3.4: Visual breakdown of the EMNIST datasets[103]

The class breakdown, structure and splits of the various datasets in the EMNIST dataset are shown. Each dataset contains handwritten digits, handwritten letters or a combination of both. The number of samples in each class is shown for each dataset and highlights the large variation in the number of samples in the unbalanced datasets. The training and testing split for each class is also shown using either solid or hatched portions of the bar graphs. In the datasets that contain merged classes, a vertical arrow is used to denote the class into which the lowercase letter is merged.

The EMNIST By Merge an EMNIST By Class datasets both contain the full 814,255 char-

acters contained in the original NIST Special Database 19. The primary differences between the two datasets are the number of classes, the number of samples per class, and the order in which the characters appear in the dataset. The By Class dataset contains the full 62 classes comprising 10 digit classes, 26 lowercase letter classes, and 26 uppercase letter classes. The By Merge dataset merges certain uppercase and lowercase characters, containing 10 digit classes and 47 letter classes. The number of characters per class varies dramatically throughout both datasets, as is clearly visible in figure 3.4. The datasets also contain vastly more digit samples than letter samples.

Optic Latin Characters

Unlike previous datasets for this we are going to use a method called:

Data generator: A function which generates images with a random characters or text of random size, thickness, and font, the more we increase the differ entices (font, size, characters, etc...) the more we increase the complexity of your dataset, with training we could recognize many different types, otherwise we could have just one font. In this system the dataset contains the following characters:

"0 1 2 3 4 5 6 7 8 9 a b c d e f g h i j k l m n o p q r s t u v w x y z A B D E F G H J K L M N Q R T Y . ( ) %"

In total 56 different characters are considered for the dataset. Uppercase characters which look like its lowercase has been discarded. The dataset contains each character in 15 different fonts, 7 different orientations, and 5 different font thickness.

Size of the dataset = 56*15*7*5 = 29400

Number	Font Name	Size	Orientation
1	Archivo Regular		
2	Calibri		
3	Comic Neue		
4	ConvertOne Regular		
5	FjallaOne Regular		
6	Helvetica Neue		
7	Helvatica		
8	Karla Regular	3 -> 8	7
9	Montserrat Regular		
10	Open Sans		
11	Roboto		
12	Source Sans Pro Regular		
13	UbuntuMono Regular		
14	Ubuntu Regular		
15	WorkSans Regular		

Table 3.3: Fonts, Sizes, Orientation used in dataset generator

B- Pre-processing the datasets

Data preprocessing is the first (and arguably most important) step toward building a working Deep learning model. It's critical [104].

Data preprocessing is a data mining technique that involves transforming raw data into an understandable format. Real-world data is often incomplete, inconsistent, and/or lacking in certain behaviors or trends, and is likely to contain many errors. Data preprocessing is a proven method of resolving such issues [105].

In Real world data are generally incomplete: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data. Noisy: containing errors or outliers. Inconsistent: containing discrepancies in codes or names.

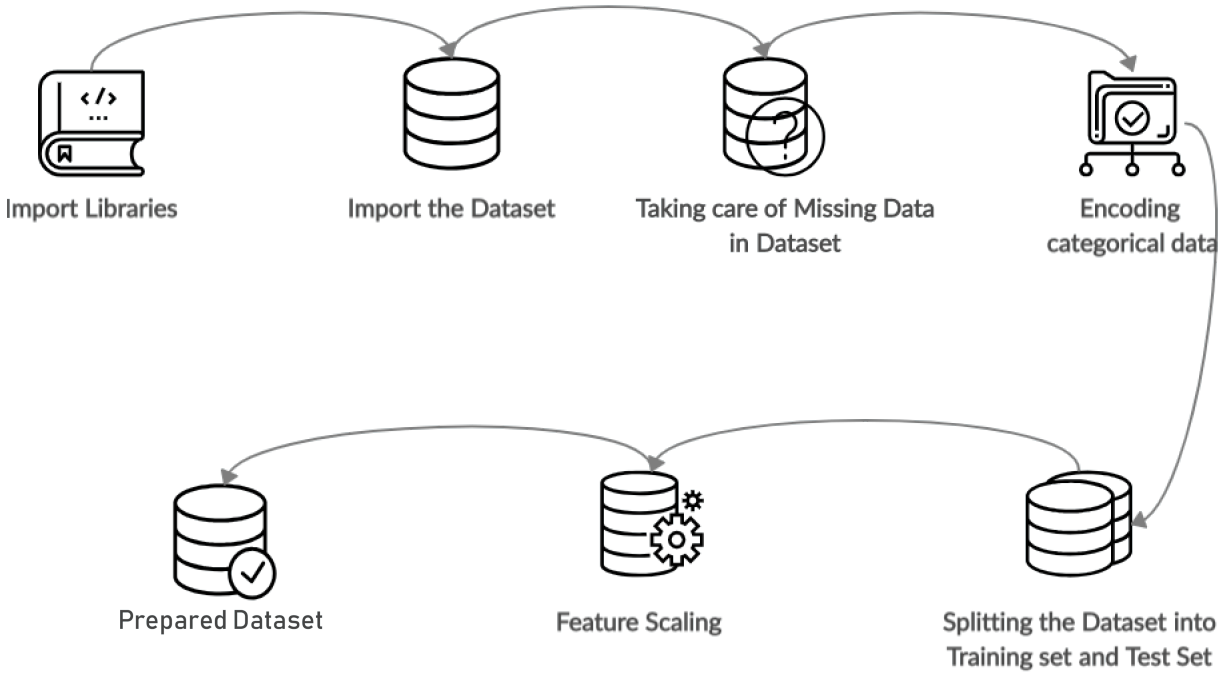


Figure 3.5: Pre-processing Steps.

Figure 3.5 shows the important steps during the pre-processing phase.

Step 1: Import Libraries

First step is usually importing the libraries that will be needed in the program. A library is essentially a collection of modules that can be called and used. A lot of the things in the programming world do not need to be written explicitly ever time they are required. There are functions for them, which can simply be invoked. also the libraries we need to load datasets and work on them.

Step 2: Import the Datasets

A lot of datasets come in different formats. We will need to locate the directory of the datasets files at first (it's more efficient to keep the dataset in the same directory as your system) and read them using a methods which can be found in the libraries into the variables on memory, furthermore used in following pre-processing steps.

Step 3: Taking care of Missing Data in Dataset

Sometimes you may find some data are missing in the dataset. We need to be equipped to handle the problem when we come across them. Obviously you could remove the entire line of data, of course we would not want to do that. One of the most common idea to handle the problem is to take a mean of all the values of the same column and have it to replace the missing data.

Step 4: Encoding categorical data

Sometimes our data is in qualitative form, that is we have texts as our data. We can find categories in text form. Now it gets complicated for machines to understand texts and process them, rather than numbers, since the models are based on mathematical equations and

calculations. Therefore, we have to encode the categorical data.

Difficulty saving money	Counts	Frequencies
Very	231	46%
Somewhat	196	39%
Not very	58	12%
Not at all	14	3%
Not sure	1	0%
Total	500	100%

Table 3.4: Example of categorical data [106].

ID	**Gender**		**ID**	**Male**	**Female**	**Not Specified**
1	Male		1	1	0	0
2	Female	—->	2	0	1	0
3	Not Specified		3	0	0	1
4	Not Specified		4	0	0	1
5	Female		5	0	1	0

Table 3.5: Example of a Dummy encoding [106].

Table 3.4 is an example of categorical data. In the first column, the data is in text form. We can see that there are five categories–Very, Somewhat, Not very, Not at all, Not sure–and hence the name categorical data [106].

Table 3.5 shows that instead of having one column with n number of categories, we will use n number of columns with only 1s and 0s to represent whether the category occurs or not [106].

Step 5: Splitting the Dataset into Training set and Test Set

Now we need to split our dataset into two sets–a Training set and a Test set. We will train our machine learning models on our training set, our deep learning models will try to understand any correlations in our training set and then we will test the models on our test set to check how accurately it can predict. A general rule of the thumb is to allocate 80% of the dataset to training set and the remaining 20% to test set.

Step 6: Feature Scaling

The final step of data preprocessing is to apply the very important feature scaling.

Feature Scaling It is a method used to standardize the range of independent variables or features of data. also to limit the range of variables so that they can be compared on common grounds. Suppose we have this data-set:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	Bi
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Table 3.6: Example of a Dataset with undefined values

In table 3.6 see the Age and Salary column. You can easily noticed Salary and Age variable don't have the same scale and this will cause some issue in your Deep learning model. Because most of the Deep learning models are based on Euclidean Distance.

$$d(A, B) = \sqrt{(x1 - x2)^2 + (y1 - y2)^2} \tag{3.1}$$

Let's say we take two values from Age and Salary column, Age- 40 and 27, Salary- 72000 and 48000.
One can easily compute and see that Salary column will be dominated in Euclidean Distance. And we don't want this thing. So there are several ways of scaling your data [107]..

Re-scaling (min-max normalization)

Also known as min-max scaling or min-max normalization, is the simplest method and consists in re-scaling the range of features to scale the range in [0, 1] or [-1, 1]. Selecting the target range depends on the nature of the data. The general formula is given as:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{3.2}$$

where x is an original value, x' is the normalized value. For example, suppose that we have the students' weight data, and the students' weights span [160 pounds, 200 pounds]. To re-scale this data, we first subtract 160 from each student's weight and divide the result by 40 (the difference between the maximum and minimum weights).

Mean normalization

$$x' = \frac{x - average(x)}{max(x) - min(x)} \tag{3.3}$$

where x is an original value, x' is the normalized value.

Standardization

The general method of calculation is to determine the distribution mean and standard deviation for each feature. Next we subtract the mean from each feature. Then we divide the values (mean is already subtracted) of each feature by its standard deviation.

$$x' = \frac{x - \bar{x}}{\sigma} \tag{3.4}$$

Where x is the original feature vector, x = average(x) x = average(x) is the mean of that feature vector, and sigma is its standard deviation.

C- Training

Its the important phase in our system, which we gonna train a Convolution neural networks model with a specific architecture wheeling to achieve best accuracy result and less data loss, Next can be used in image character recognition, the training phase could be divided into two steps illustrated in the figure 3.6.

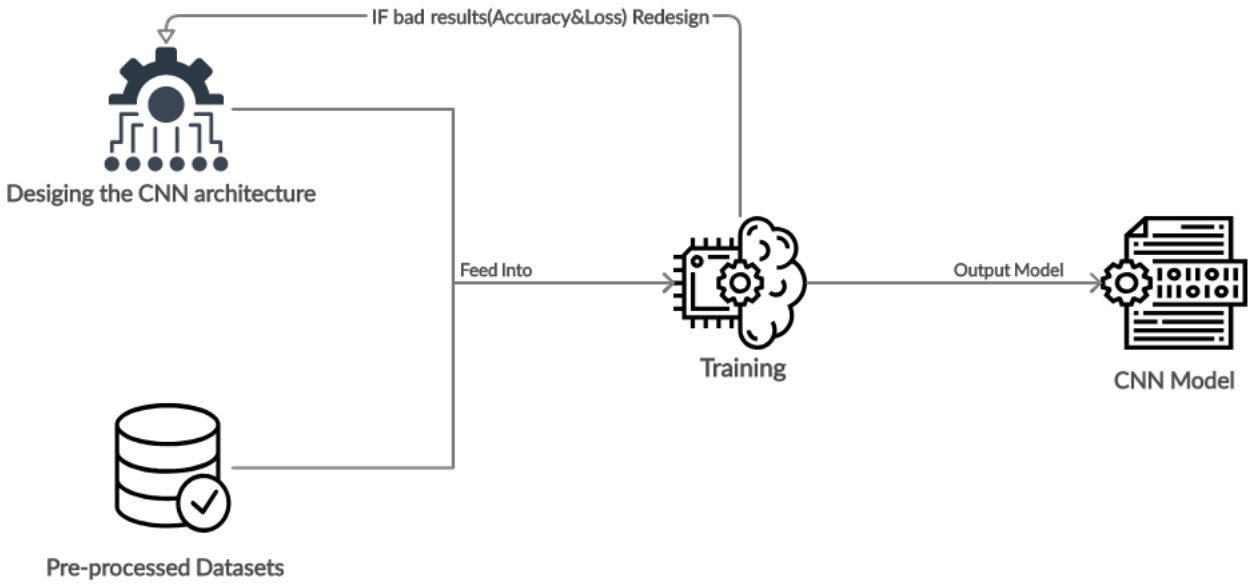


Figure 3.6: Training Phase.

Step 1: Model architecture design

As we already know the deep learning method we are going to work with "Convolution neural networks", We design the model parameters with how much layers, each layer size , each layer type and defining layer function , Also the layers placement as we saw in chapter 2.3.2.4

Step 2: Training

After getting the model architecture ready and designed, And the pre-processed datasets, Now we configure training phase parameters such as : Batches size , Number of epochs, also defining the data gonna use, after we processed the training waiting for good results, in a case for bad results, we back to the Step 1: Model architecture design, and we train again until a good

results accrued.

D- Use of the output model

With the training phase done, Of-course no matter what a best model results have been achieved we save our model for image character recognition, the figure 3.7 shows an example of steps while testing our good results output model with a input image, after that we get the recognition results.

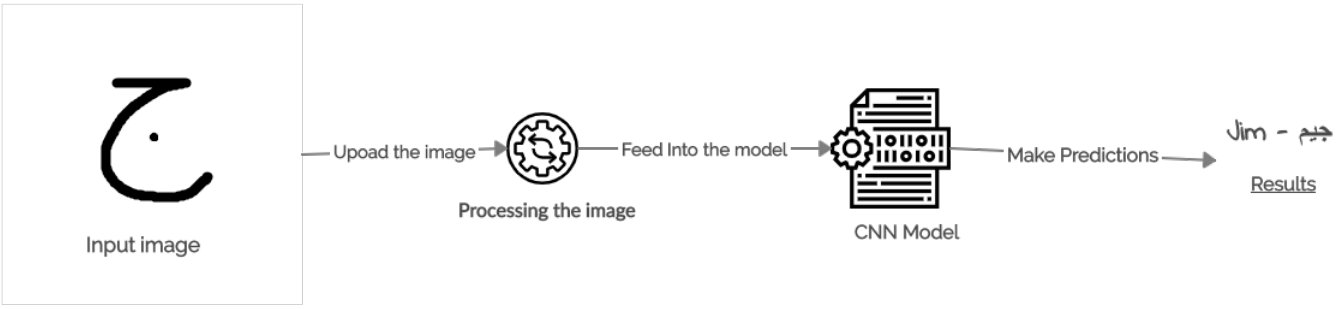


Figure 3.7: Example of testing the deep learning model.

3.2.3.2 Front-end

In our front-end system, The main part for the user-backend interaction, Giving the ability of exchanging data between user-backend, we gonna make simple access design for each part, and each request will interact with the backend related to. The Figure 3.8 will illustrate the whole mechanism.

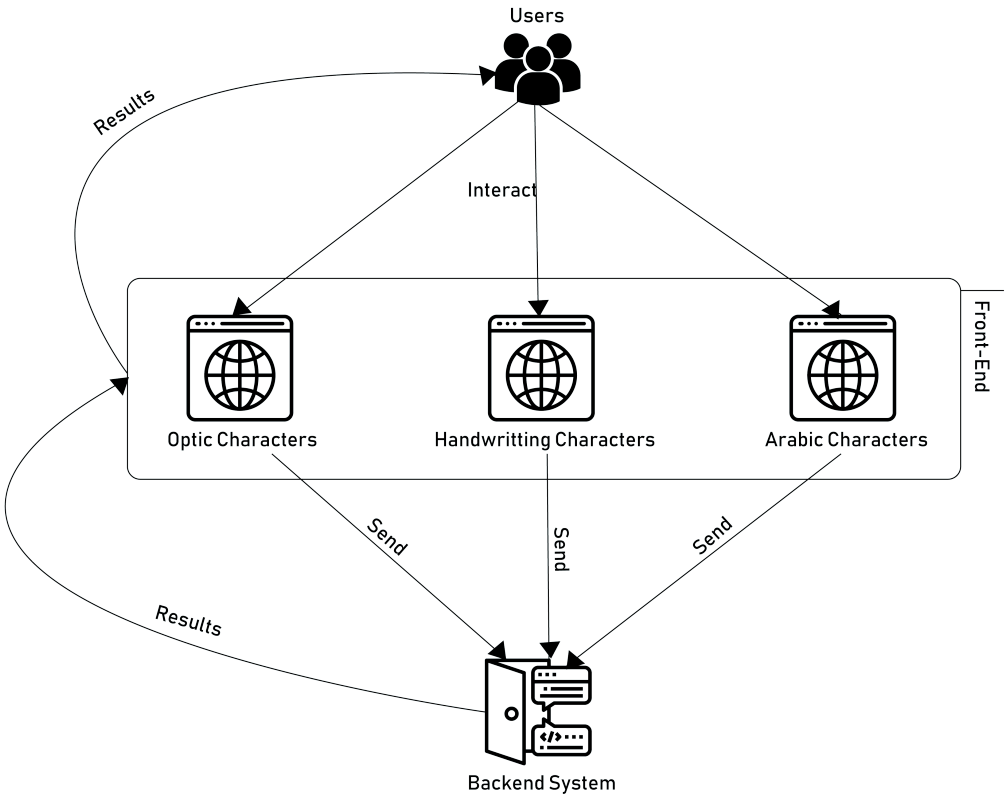


Figure 3.8: Front-End System.

3.3 Implementation

3.3.1 Environments and developing tools

To Develop our system, we are going to use different environments and tools for the backend programming language including API's, libraries, for the front-end also we gonna use programming language, markup language also the style sheet with different libraries and frameworks, working on many editors, IDE etc... .

Python

Figure 3.9: Python Logo.

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale

projects.

JavaScript

Figure 3.10: JavaScript Logo.

JavaScript often abbreviated as JS, is a high-level, interpreted programming language that conforms to the ECMAScript specification. JavaScript has curly-bracket syntax, dynamic typing, prototype-based object-orientation, and first-class functions.

HTML and CSS

Figure 3.11: HTML5 Logo.

Hypertext Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets (CSS) and scripting languages such as JavaScript.

Figure 3.12: CSS Logo.

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.

Tensorflow

Figure 3.13: Tensorflow Logo.

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

Keras

Figure 3.14: Keras Logo.

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible. It was developed as part of the research effort of project ONEIROS (Open-ended Neuro-Electronic Intelligent Robot Operating System), and its primary author and maintainer is François Chollet, a Google engineer. Chollet also is the author of the XCeption deep neural network model.

In 2017, Google's TensorFlow team decided to support Keras in TensorFlow's core library.

Chollet explained that Keras was conceived to be an interface rather than a standalone machine-learning framework. It offers a higher-level, more intuitive set of abstractions that make it easy to develop deep learning models regardless of the computational backend used. Microsoft added a CNTK backend to Keras as well, available as of CNTK v2.0.

Flask (framework)

Figure 3.15: Flask Logo.

Flask is a micro web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program.

Applications that use the Flask framework include Pinterest, LinkedIn, and the community web page for Flask itself.

Anaconda

Figure 3.16: Anaconda Logo.

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda.

Numpy

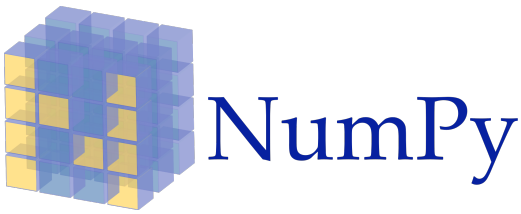


Figure 3.17: Numpy Logo.

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Pandas

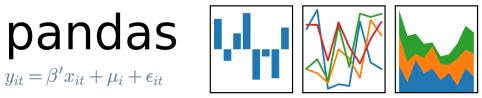


Figure 3.18: Pandas Logo.

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license

JQuery

Figure 3.19: JQuery Logo.

jQuery is a JavaScript library designed to simplify HTML DOM tree traversal and manipulation, as well as event handling, CSS animation, and Ajax. It is free, open-source software using the permissive MIT License. As of May 2019, jQuery is used by 73% of the 10 million most popular websites.

Matplotlib

Figure 3.20: Matplotlib Logo.

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+.

CUDA

Figure 3.21: CUDA Logo.

CUDA is a parallel computing platform and application programming interface model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit for general purpose processing — an approach termed GPGPU.

OpenCV

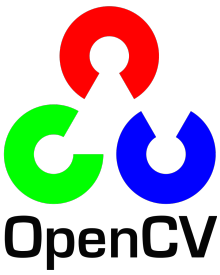


Figure 3.22: OpenCV Logo.

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license.

Ajax

Figure 3.23: Ajax Logo.

Ajax is a set of web development techniques using many web technologies on the client side to create asynchronous web applications. With Ajax, web applications can send and retrieve data from a server asynchronously without interfering with the display and behavior of the existing page.

PyCharm

Figure 3.24: PyCharm Logo.

PyCharm is an integrated development environment used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains.

Brackets

Figure 3.25: Brackets Logo.

Brackets is a source code editor with a primary focus on web development. Created by Adobe Systems, it is free and open-source software licensed under the MIT License, and is currently maintained on GitHub by Adobe and other open-sourced developers. It is written in JavaScript, HTML and CSS.

Web browser

A web browser is a software application for accessing information on the World Wide Web. Each individual web page, image, and video is identified by a distinct Uniform Resource Locator, enabling browsers to retrieve these resources from a web server and display them on a user's device.

3.3.2 Developing the Back-end

In this part of the system we are going to develop the backend system where the training and the processing happen using Python language with some well known API's and libraries working on the deep learning (CNN).

3.3.2.1 Loading datasets

Handwriting Characters

After searching and downloading the benchmark datasets, we want to load them into the memory so can be pre-processed and the code for that:

```
mndata = MNIST('data2',return_type='numpy',gz=True)
dataset = mndata.select_emnist('balanced')
X_train , y_train = mndata.load_training()
X_test , y_test = mndata.load_testing()
```

Listing 3.1: Loading Handwriting Characters datasets

We define the repository of the datasets and and then we call a method to read the data in gz and it will give us separated variables in numpy type.

Arabic Characters

After getting the Arabic characters datasets as CVS files, we organize them in repositories, using the pandas library we try to load them into the memory furthermore been reprocessed, the datasets contain 8 files : 4 files for Arabic Digits , 4 files for Arabic letters the code below we show the load session

```
#Loading Arabic Letters Dataset
training_letters_images = pd.read_csv(letters_training_images_file_path,
    compression='zip', header=None)
training_letters_labels = pd.read_csv(letters_training_labels_file_path,
    compression='zip', header=None)
testing_letters_images = pd.read_csv(letters_testing_images_file_path,
    compression='zip', header=None)
testing_letters_labels = pd.read_csv(letters_testing_labels_file_path,
    compression='zip', header=None)

#Loading Arabic Digits Dataset
training_digits_images = pd.read_csv(digits_training_images_file_path,
    compression='zip', header=None)
training_digits_labels = pd.read_csv(digits_training_labels_file_path,
    compression='zip', header=None)
testing_digits_images = pd.read_csv(digits_testing_images_file_path, compression
    ='zip', header=None)
testing_digits_labels = pd.read_csv(digits_testing_labels_file_path, compression
    ='zip', header=None)
```

Listing 3.2: Loading Arabic Characters datasets

OCR Characters

Concerning the Optical characters datasets we are going to use unusual technique called "data generator" mentioned in chapter 3.2.3.1, As we gonna define the type of fonts we want our data sets to have, The characters, sizes, orientations, etc..., The function details is described in code below :

```
font_folder = '/data'
font_list = os.listdir(font_folder) #list of all fonts

digit = '0 1 2 3 4 5 6 7 8 9 ' #all digits
lc = 'a b c d e f g h i j k l m n o p q r s t u v w x y z ' #all lower case
all_uc = 'A B C D E F G H I J K L M N O P Q R S T U V W X Y Z ' #all upper case
sel_uc = 'A B D E F G H J K L M N Q R T Y ' #selected upper case
all_sign = '''! @ # % ^ & ? / ( ) { } [ ] < > * - + = \ : ; ' . ''' #all useful
    signs
sel_sign = '''. ( ) % ''' #selected signs

digi_perc = digit + '% ' #digits with percentage

digi_lc = digit + lc #digits and all lower case
digi_uc = digit + all_uc #digitand all upper case
all_letters = lc + all_uc #all letters
digi_all_letters = digi_lc + all_letters #all digits and all letters
digi_all_letters_all_sign = digi_all_letters + all_sign #all digits, all letters
    and all useful signs

lc_sel_uc = lc + sel_uc
digi_lc_sel_uc = digit + lc + sel_uc
digi_all_letters_sel_sign = digi_all_letters + sel_sign # all digit, all letters
    and selected signs
digi_lc_sel_uc_sel_sign = digi_lc + sel_uc + sel_sign # all digits, all lower
    case, selected upper case and selected signs
digi_lc_sel_uc_all_sign = digi_lc + sel_uc + all_sign # all digits, all lower
    case, selected upper case and all signs

#calling the function
data, labels = data_gen(font_folder, fl,
                        data_str = ds,
                        img_size = ims, e_kernel_size = eks, d_kernel_size = dks,
                        w_min = wmi, w_max = wma, h_min = hmi, h_max = hma,
                        ang_lb = alb, ang_ub = aub, ang_off = aoff,
                        area_thres = at,
                        dial_ub = dub, dial_off = doff,
                        count_start = cs, show_bool = 0)
```

Listing 3.3: Optic Characters datasets generator

The function shows us that, We define the characters we want to generate and defining the parameters that we need to input into the generation function.

3.3.2.2 Pre-processing the loaded datasets

Handwriting Characters

After we've loaded the datasets into the memory in numpy type, we preprocesse these variable so can be used for the training session in mention steps:

- Convert data to numpy arrays and normalize images to the interval [0, 1].
- Reshaping all images into 28*28 for pre-processing.
- Prepare the input for the model.

And the code for that:

```
X_train = np.array(X_train) /255
y_train = np.array(y_train)
X_test = np.array(X_test) /255
y_test = np.array(y_test)

X_train = X_train.reshape(X_train.shape[0], 28, 28)
X_test = X_test.reshape(X_test.shape[0], 28, 28)

X_train = X_train.reshape(X_train.shape[0], 784,1)
X_test = X_test.reshape(X_test.shape[0], 784,1)

train_images = X_train.astype('float32')
test_images = X_test.astype('float32')

train_images = hf.resh(train_images)
test_images = hf.resh(test_images)


train_labels = np_utils.to_categorical(y_train, 47)
test_labels = np_utils.to_categorical(y_test, 47)
```

Listing 3.4: Pre-processing the loaded Handwriting Characters datasets

Arabic Characters

For Arabic characters datasets the preprocessing step its need regarding the loaded data so following some steps to prepare the data for training:

1. Image Normalization : We rescale the images by dividing every pixel in the image by 255 to make them into range [0, 1].
2. Encoding Categorical Labels : From the labels CSV files we can see that labels are categorical values and it is a multi-class classification problem. Our outputs are in the form of: Digits from 0 to 9 have categories numbers from 0 to 9, Letters from alef to yeh have categories numbers from 10 to 37.

3. Reshaping Input Images to 64x64x1 : So we will reshape the input images to a 4D tensor with shape (nb-samples, 64, 64 ,1) as we use grayscale images of 64x64 pixels.

Giving the code code trying to implements the steps above:

```
training_digits_images_scaled = training_digits_images.values.astype('float32')
    /255
training_digits_labels = training_digits_labels.values.astype('int32')
testing_digits_images_scaled = testing_digits_images.values.astype('float32')
    /255
testing_digits_labels = testing_digits_labels.values.astype('int32')

training_letters_images_scaled = training_letters_images.values.astype('float32'
    )/255
training_letters_labels = training_letters_labels.values.astype('int32')
testing_letters_images_scaled = testing_letters_images.values.astype('float32')
    /255
testing_letters_labels = testing_letters_labels.values.astype('int32')

# number of classes = 10 (digits classes) + 28 (arabic alphabet classes)
number_of_classes = 38
training_letters_labels_encoded = to_categorical(training_letters_labels,
    num_classes=number_of_classes)
testing_letters_labels_encoded = to_categorical(testing_letters_labels,
    num_classes=number_of_classes)
training_digits_labels_encoded = to_categorical(training_digits_labels,
    num_classes=number_of_classes)
testing_digits_labels_encoded = to_categorical(testing_digits_labels,
    num_classes=number_of_classes)

# reshape input digit images to 64x64x1
training_digits_images_scaled = training_digits_images_scaled.reshape([-1, 64,
    64, 1])
testing_digits_images_scaled = testing_digits_images_scaled.reshape([-1, 64, 64,
     1])

# reshape input letter images to 64x64x1
training_letters_images_scaled = training_letters_images_scaled.reshape([-1, 64,
     64, 1])
testing_letters_images_scaled = testing_letters_images_scaled.reshape([-1, 64,
    64, 1])
```

Listing 3.5: Pre-processing the loaded Arabic Characters datasets

Also we need to concatenate the digits and letters in one input using method from the Numpy library.

OCR Characters

Because we've used a data generator function so, us we manipulate the generation of the datasets, while we calling the generation function and before we save the datasets so we controle

each output image from datasets, and then after saving it the outcome is a pre-processed generated characters datasets.

3.3.2.3 Training the model

Handwriting Characters

After we've prepared the datasets, First we design our architecture for the chosen deep learning method (convolution neural networks), using tensorflow and the code for that:

```
model = Sequential()
model.add(Reshape((28,28,1), input_shape=(784,)))
model.add(Convolution2D(32, (5,5),activation='relu'))
model.add(Convolution2D(64, (5,5),activation='relu'))
model.add(Convolution2D(128, (5,5),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(1024, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(Dropout(0.5))
model.add(Dense(47, activation='softmax'))
```

Listing 3.6: Designing the Handwriting Characters model

After we getting everything setup, Pre-processed datasets and the model architecture designed, now we need to launch the train session with certain chosen parameters:

```
model.compile(loss='categorical_crossentropy', optimizer=Adadelta() , metrics=['
    accuracy'])

callbacks = [EarlyStopping(monitor='val_loss', patience=10)]
print(model.summary())
history = model.fit(train_images,train_labels,validation_data=(test_images,
    test_labels), batch_size=128, epochs=60 , verbose=1)
```

Listing 3.7: Training the Handwriting Characters model

Using the Adadelta optimizer which the best option for Handwritten character we launch the training on 60 epochs and with batch size at 128 per epochs. Next the giving output model will be used later on testing and the prediction **Arabic Characters**

When we get to this session, Now it time to design the model architecture and trying to find a one that fits us and give good out results, after many manipulation we get to this:

```
  model.add(Conv2D(filters=16, kernel_size=3, padding='same', input_shape=(64,
    64, 1), kernel_initializer=kernel_initializer, activation=activation))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=2))
  model.add(Dropout(0.2))
  model.add(Conv2D(filters=32, kernel_size=3, padding='same', kernel_initializer
    =kernel_initializer, activation=activation))
  model.add(BatchNormalization())
```

```
  model.add(MaxPooling2D(pool_size=2))
  model.add(Dropout(0.2))
  model.add(Conv2D(filters=64, kernel_size=3, padding='same', kernel_initializer
   =kernel_initializer, activation=activation))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=2))
  model.add(Dropout(0.2))
  model.add(Conv2D(filters=128, kernel_size=3, padding='same',
   kernel_initializer=kernel_initializer, activation=activation))
  model.add(BatchNormalization())
  model.add(MaxPooling2D(pool_size=2))
  model.add(Dropout(0.2))
  model.add(GlobalAveragePooling2D())
  #Fully connected final layer
  model.add(Dense(38, activation='softmax'))
```

Listing 3.8: Designing the Arabic Characters model

Using Adam optimizer, we launch the training on 30 epochs with 20 batch size on each epoch

```
model = create_model(optimizer='Adam', kernel_initializer='uniform', activation=
    'relu')

history = model.fit(training_data_images, training_data_labels,
                    validation_data=(testing_data_images, testing_data_labels),
                    epochs=10, batch_size=20, verbose=1, callbacks=[checkpointer
    ])
```

Listing 3.9: Training the Arabic Characters model

OCR Characters

Data generator function gave us pre-processed datasets ready for the training phase before that as we use to do we design the following model :

```
  conv1 = conv_block(1,image_batch,weights[i], conv_op = conv_op[i],
   conv_padding='SAME', dropout=dropout[i],weight_decay=wd)
  i=i+1
  pool1=tf.nn.max_pool(conv1, ksize=[1, 4, 4, 1], strides=[1,4,4,1],padding='
   SAME', name='pool1') #32x32

  conv2 = conv_block(2,pool1,weights[i], conv_op = conv_op[i], conv_padding='
   SAME', dropout=dropout[i],weight_decay=wd)
  i=i+1
  pool2=tf.nn.max_pool(conv2, ksize=[1, 4, 4, 1], strides=[1,4,4,1],padding='
   SAME', name='pool2') #8x8

  conv3 = conv_block(3,pool2,weights[i], conv_op = conv_op[i], conv_padding='
   SAME', dropout=dropout[i],weight_decay=wd)
  i=i+1
  pool3=tf.nn.max_pool(conv3, ksize=[1, 4, 4, 1], strides=[1,4,4,1],padding='
   SAME', name='pool3') #2x2

```

```
13  conv4 = conv_block(4,pool3,weights[i], conv_op = conv_op[i], conv_padding='
     SAME', dropout=dropout[i],weight_decay=wd)
14  i=i+1
15  pool4=tf.nn.max_pool(conv4, ksize=[1, 2, 2, 1], strides=[1,2,2,1],padding='
     SAME', name='pool4')#1x1
```

Listing 3.10: Designing the Optic Characters model

After that we processed in the training, on 10 epochs for 32 batch for each epoch, and with callup training function.

```
1 train(folder_path, train_filename, test_filename,
2       train_data_count, file_count,
3       weights, dropout, wd,
4       img_size, max_char, class_count,
5       batch_size=batch_size, learning_rate=lr, epochs=epochs,
6       restore=False, var_lr=[None,None])
```

Listing 3.11: Training the Optic Characters model

3.3.3 Developing the Front-end

In the front-end part of the system, we gonna create a multi-pule user interface pages, we want to create a simple design that can every user with different gender and ages to use it, with some simple colors, and structure. The figure 3.26 shows the architecture of the front-end pages

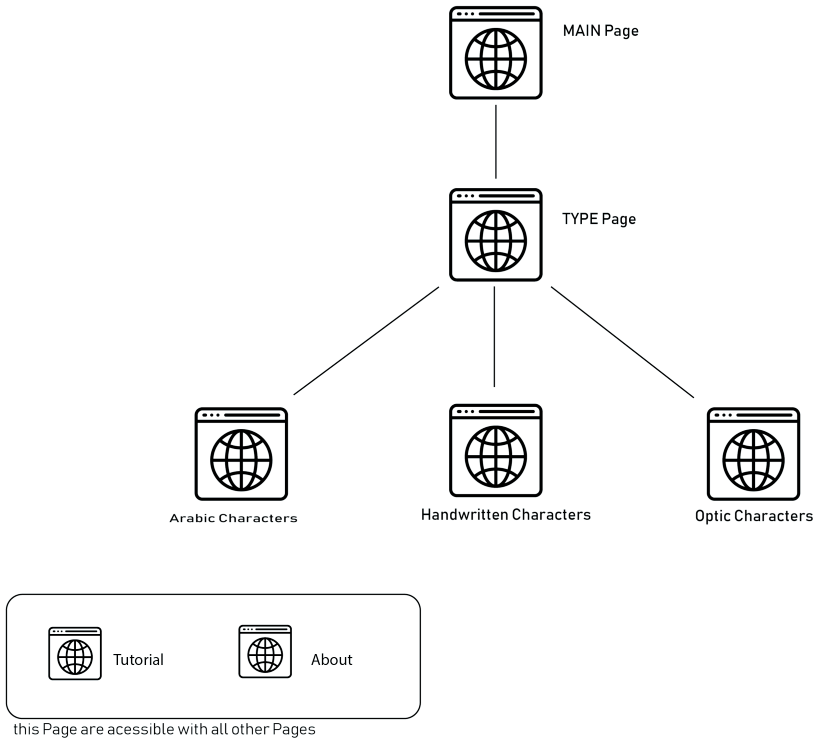


Figure 3.26: The architecture of the front-end pages.

Working on HTML for the content and the structure of the things that we want to put in out pages, With the use of the CSS which give us big advantage on designing and styling our page for better looking and well organizing, also the JavaScript the one that makes our pages moving and working giving some benefits on animations, button functions, pages function, uploading, printing, etc..., the figures below shows the content of each created web page.

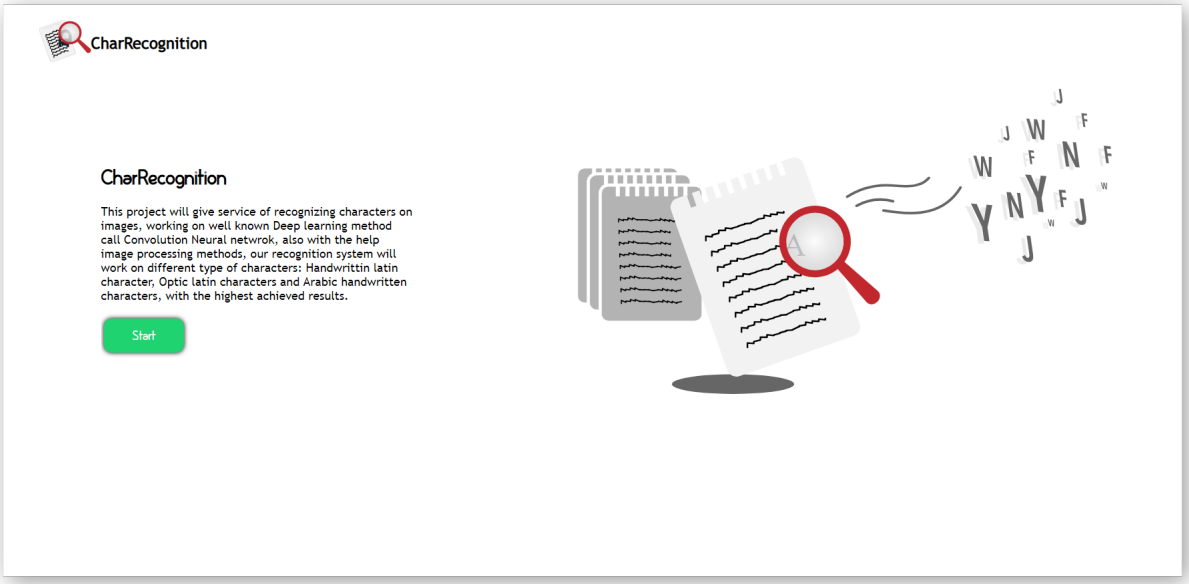


Figure 3.27: First Page.

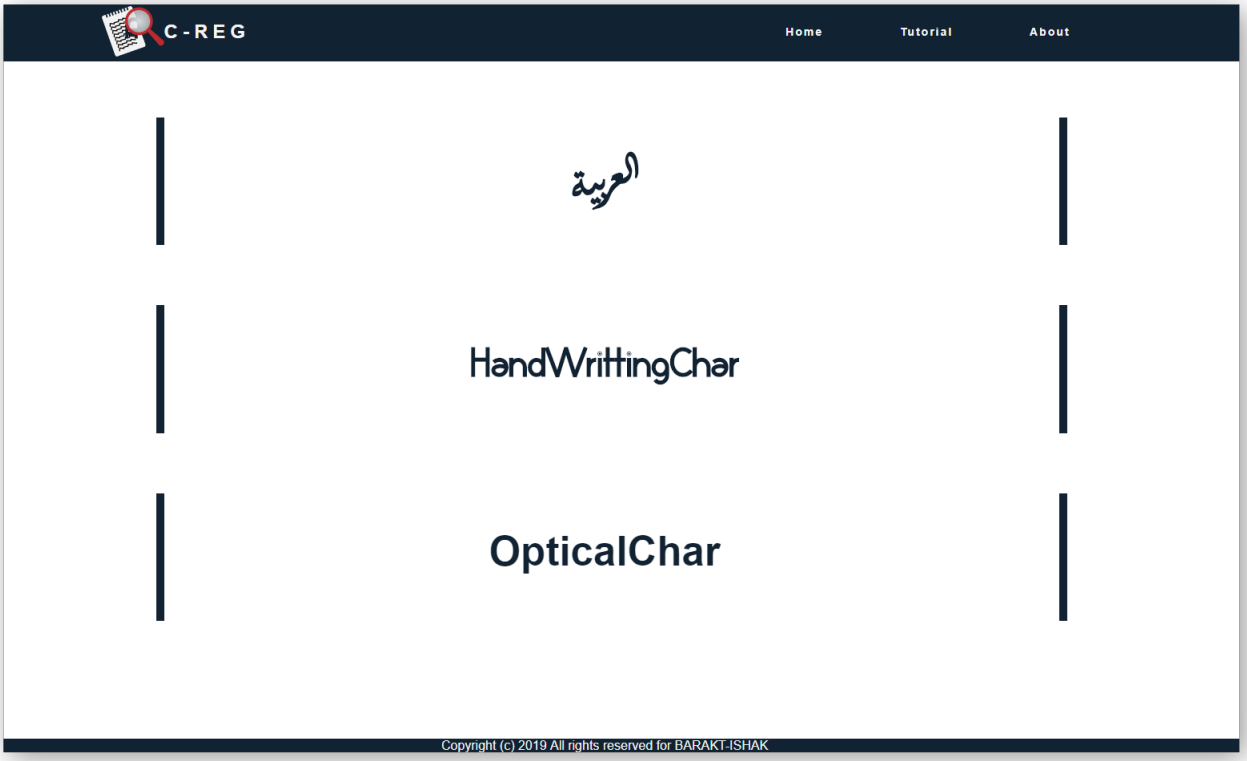


Figure 3.28: Second Page.

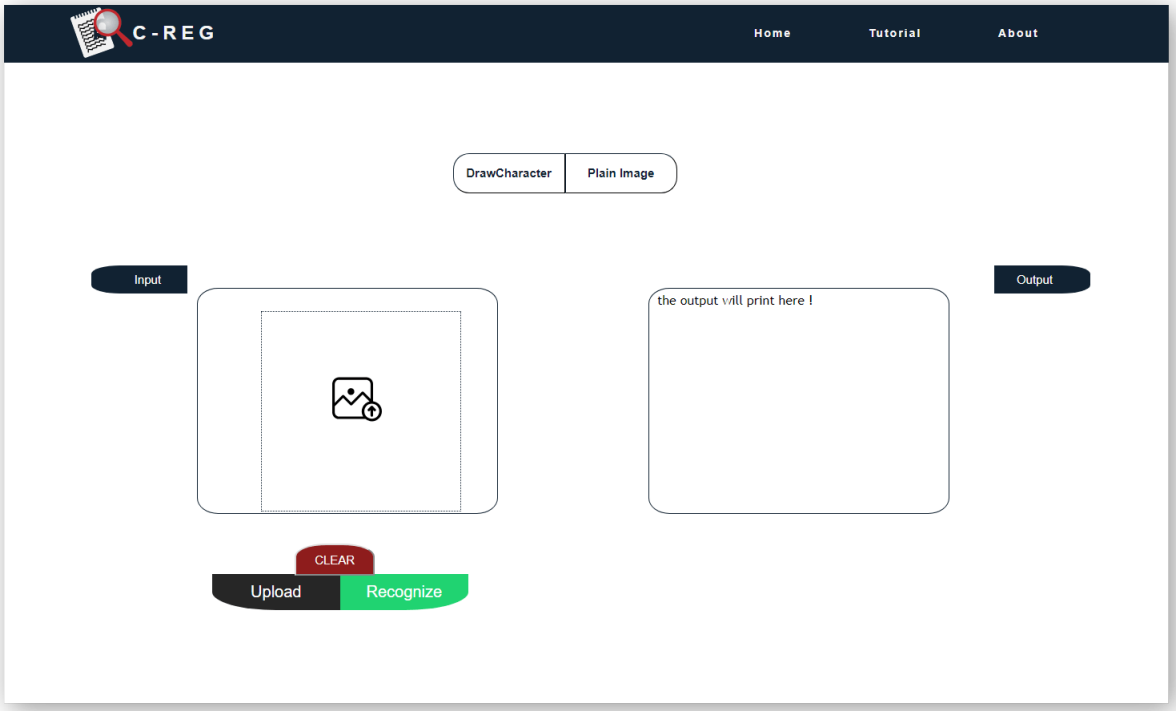


Figure 3.29: Third Page.

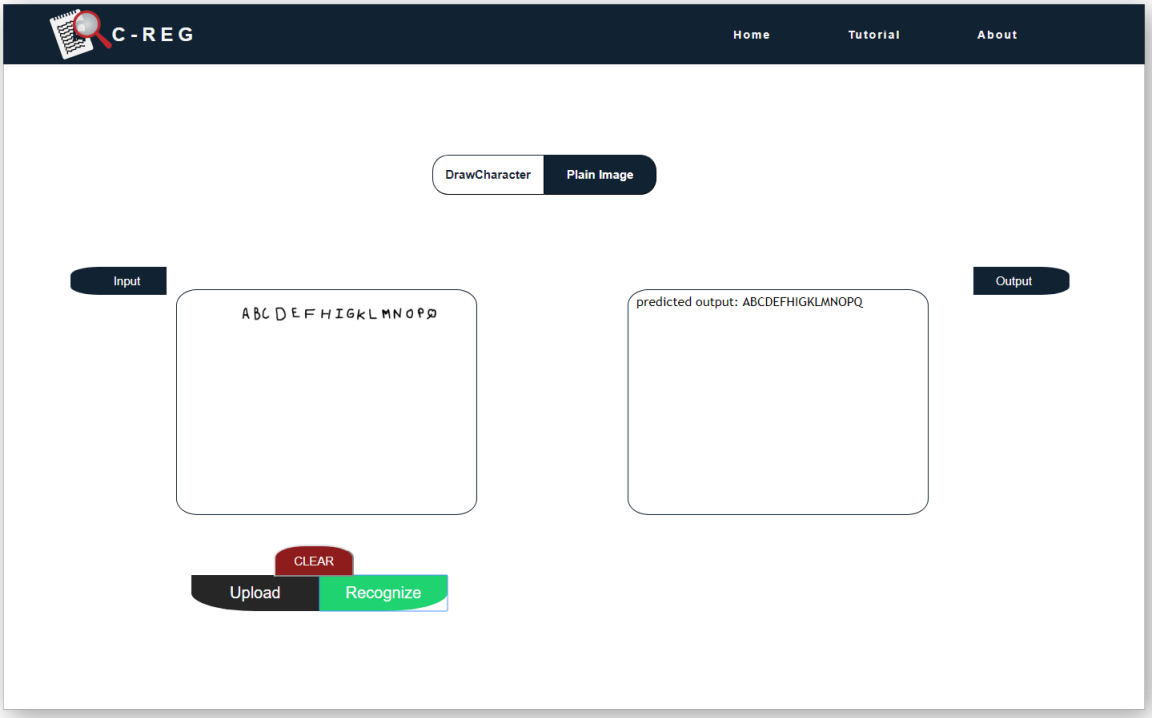


Figure 3.30: Fourth Page.

3.4 Results

Working on specific hardware and software combination, for our configuration: i5 3K CPU and 8 GB of ram memory, windows 10 and Nvidia 770 GTX used for the training phase, taking a benefits of the paralleling computing for fast training gaining time and resources, we obtain very much good results on the different models we gonna train .

3.4.1 Arabic handwritten character Results

The table 3.7, illustrate a different manipulation of parameters and CNN architecture designing wheeling to get the best output model we can get for the prediction accuracy.

No.	CNN architecture	Optimizer	Training Time (GPU)	loss value %	Accuracy value
Test 1	Conv2D,Conv2D,Maxpool,Dropout,Conv2D,Conv2D,Maxpool,Dropout,FC,FC,FC	RMSprop	32 min	14%	90%
Test 2	Conv2D,Maxpool,Conv2D,Maxpool,Conv2D,Maxpool,Dropout,FC,FC,FC	RMSprop	18 min	30%	94%
Test 3	Conv2D,Conv2D,Maxpool,FC,FC	Adam	5 min	63%	88%

Table 3.7: Arabic handwritten character tests

Test 1: We have implemented a given architecture composed of 11 layers (4 convolution, 2 max-pool, 2 dropout and 3 fully-connected layers) also we used the RMSprop optimizer to calculate the loss values for the training, the training to about 32 min using GPU paralyzing which as we can see gives us good accuracy value as 90% and 14% loss.

Test 2: In this test we gonna change the CNN architecture composed of 10 layers (3 convolution layers, 3 max-pool, 1 dropout and 3 fully-connected layers) leaving the same optimizer as RMSprop, we gain less execution time of 18 min less 14 min from test 1 also we gain some accuracy value of 94% however the loss value have increased which is bad sign of bad predictions.

Test 3: Trying to lower to complexity of our CNN Architecture and also changing the the optimizer from RMSprop to Adam we achieved big step in training time with 13 min from test 2 and 17 from test 1, however we had big loss in the accuracy value with 88% and the loss value had increased to 63%.

With many previous tests as we proposed CNN architecture that can gives us best outcome results shown in figure 3.31 composed of 12 layers (4 convolution layers, 3 max-pooling and 3 dropout also 1 fully-connected which is a group of the classes we want to predict).

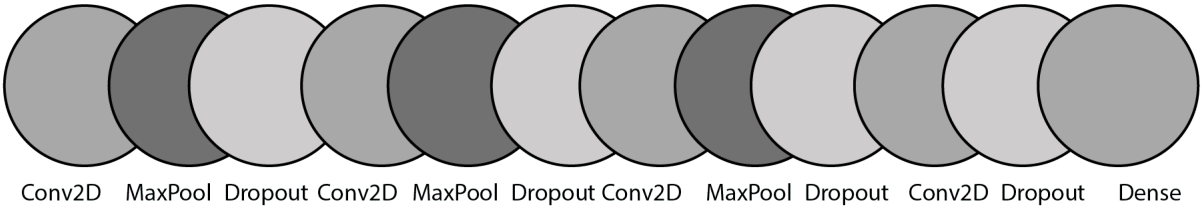
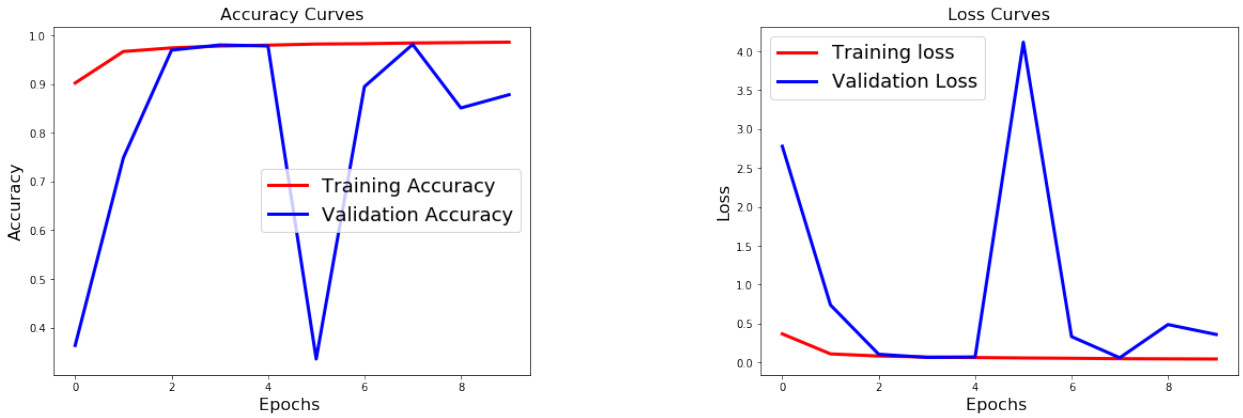


Figure 3.31: Proposed CNN architecture for Arabic characters.

We launch the architecture at total 30 epochs in 57 min with 20 batch per epoch, so the model get deep training and so we increase the accuracy, the graphs 3.32 shows the values of the loss, and the accuracy during the training phase. We got very high accuracy of 98.86% and with a value of 9% of loss.



(a) Accuracy graph

(b) Loss graph

Figure 3.32: Handwritten Arabic characters training graphs

3.4.2 Handwritten Latin characters Results

The table 3.8 shows the tests we were making during the training with of parameters manipulations.

No.	CNN architecture	Optimizer	Training Time (GPU)	loss value %	Accuracy value
Test 1	Conv2D,Conv2D,Maxpool,FC,FC,FC	Adam	48 min	50%	84%
Test 2	Conv2D,Maxpool,Conv2D,Maxpoot,FC,FC,FC,FC	Adam	1h30min	41%	85%
Test 3	Conv2D,Conv2D,FC	Adalta	40 min	82%	86%
Test 4	Conv2D,Conv2D,FC	Adam	34 min	69%	84%
Test 5	Conv2D,Maxpool,Conv2D,Maxpool,FC,FC,FC,FC	adalta	1h47min	30%	86%
Test 6	Conv2D,Conv2D,Conv2D,FC	adalta	46 min	15%	87%
Test 7	Conv2D,Conv2D,Conv2D,FC	adam	51 min	12%	88%

Table 3.8: Handwritten Latin characters tests

Test 1: Trying a simple CNN architecture composed of (2 convolution layers, and max-pool layer, and 3 fullyconnected) and using the Adam optimizer to calculate the loss values, we had a model trained in about 48 min with 84% accuracy value and 50% loss value which seems like the model will give lot of prediction errors.

Test 2: This test we made we've tried to increase the complexity of the layers add 1 max-pool layer and 1 fullyconnected layer to the test 1, the training to long time about 1h30min and not big progress in the loss value

Test 3: In this Test we are going to create very simple CNN model with (2 Convolution layers and 1 fullyconnected layer) also with adalta optimizer the training time took about 40min with 86% however we've got higher loss value up to 82% means bad results.

Test 4: We kept the same CNN architecture, but we made change in the optimizer as we used Adam, the training time had less then test 3 with about 34min also we gain some loss values. be-still not the result we want.

Test 5: The architecture its the same as test 2, we include some optimizer changes as we used the adalta the training took bit longer then test 2, and the loss value have come 30% which means its big progress in the, the accuracy got to 86%.

Test 6: In this test we did a CNN architecture composed only of Convolutions layer and Fully-Connected and we chose the Adalta optimizer so at 46 min of training we've got quite good results comparing to the previous one, the loss values we made it down to 15% and the accuracy up to 87%.

Test 7: Working on the same architecture from the past test 6 only changing the optimizer to Adam, bit of training time increase also the accuracy to 88%, the loss values had come down to 12%.

Continuing from the test 7 which is the best results we've got comparing to the other test, with few augmentation and architecture amelioration, we come to make the architecture with best results so far, the one proposed in figure 3.33 composed of 6 layers (3 convolution layers, 1 maxpooling and 1 dropout and 1 fully-connected that contains the output classes, now we're ready for more learning and training so we can increase our model accuracy and predictions error.

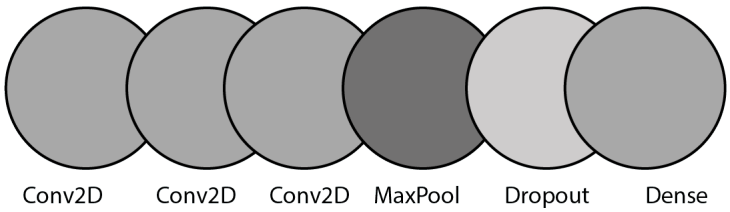
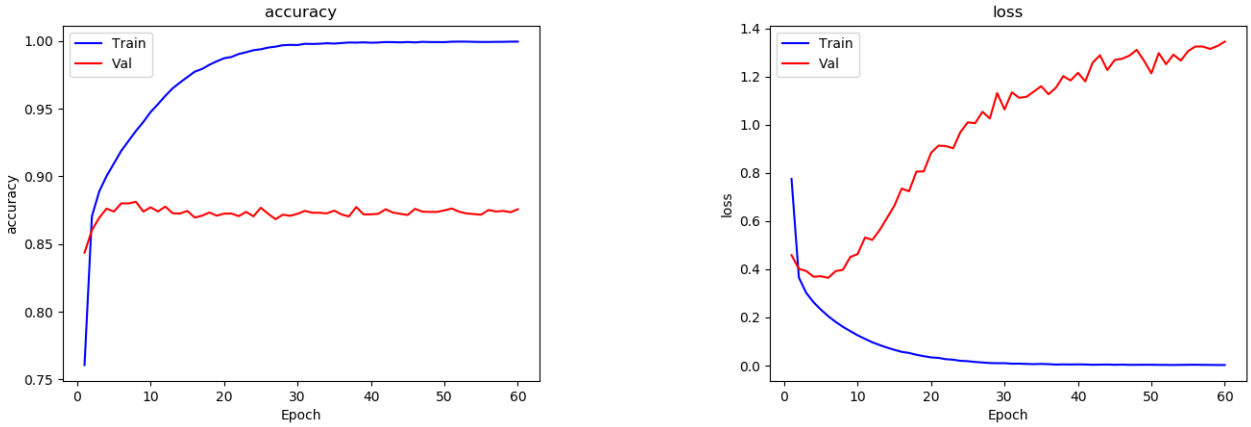


Figure 3.33: Proposed CNN architecture for Handwritten Latin characters.

We launch the training at 60 epochs and 128 batch per epoch, the training took about 2hours and 15 min so we got a good model results with less predictions error of 12% and a accuracy value of 88% of all classes. The graphs 3.34 shows the values of the loss, and the accuracy during the training phase.



(a) Accuracy graph.

(b) Loss graph.

Figure 3.34: Handwritten Latin characters training graphs.

3.4.3 Optic Latin characters results

A proposed CNN architecture in purpose of building a CNN model for the Optic Latin characters recognition, figure 3.35 shows that our architecture composed of 9 layers (4 convolution layers, 4 max-pooling layers and 1 fully connected layer that contain the whole classes output).

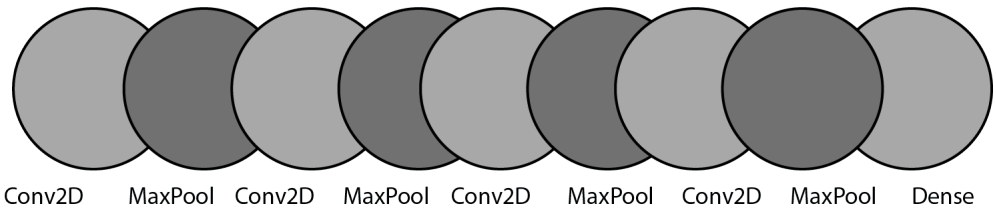


Figure 3.35: Proposed CNN architecture for Optic Latin characters.

We launch the training with the proposed architecture, at 10 epochs for 1120 batch per epoch, the obtained model results is for 96% accuracy and of 15% of loss value.

Name	Accuracy
Arabic handwritten characters	
Ahmed El-Sawy, Mohamed Loey and Hazem EL-Bakry	94.9%
Baseline (vanilla)	32.37%
Our Model	**98.86%**
Latin handwritten characters	
Gregory C, Saeed A, Jonathan T and Andre S	80%
Our Model	**88%**
Optic Latin characters	
Haochen Z, Dong L, Zhiwei X	78.10%
Our Model	**96%**

Table 3.9: Results Comparison

3.5 Discussion of results and comparison

We been able to build a Characters recognition system using deep learning technique convolution neural networks, We made three types of characters (Handwritten Arabic characters, Handwritten Latin characters, Optic Latin characters). with many tests we made we've gained the best results we could.

Our Arabic handwritten CNN model which can classify the Arabic handwritten images into digits and letters. We tested the model on more than 13000 image with all possible classes and got very high accuracy of 98.86%, which seems pretty much convince, Comparing to an exist work made by Ahmed El-Sawy, Mohamed Loey from Benha University and Hazem EL-Bakry from Mansoura University [101]. They achieved 94.9% accuracy which is less with about 4% from our CNN model, also camparing to Benchmarks CNN model (vanilla) with accuracy of 32.37% from the baseline Model (vanilla), our CNN model had made a big advantage accuracy score with about 68% difference.

Our Latin handwritten CNN model which can classify the Latin handwritten images into digits, uppercase letters and lowercase letters. Tested on more than images with all possible classes and we've got a good accuracy of 88%, comparing to an existent work on the EMINST balanced Datasets, by Gregory Cohen, Saeed Afshar, Jonathan Tapson, and Andre van Schaik from Western Sydney University [102]. And with the results they had about 80% accuracy, Our CNN model had a big accuracy win, comparing to the same datasets creators, which gives us more prediction results and less Miss-classification with about 8% difference then the other model.

Our Optic Latin CNN model, can classify the Optic images into digits and letters, and some extra shapes as : comma, point, etc..., Tested on more then 1000 image with an 96% accuracy,

comparing to an exist work by Haochen Zhang, Dong Liu, Zhiwei Xiong from University of Science and Technology of China-Hefei [108]. Their work had achieved 78.10% of accuracy as notice its away less about 18% from our CNN model.

3.6 Limitation

With the good results we had and achieved using the Deep learning technique (Convolution neural networks) to recognize characters in images, and during testing the performance of our system few Limitation that we had to discover:

1. In the Arabic handwritten Characters, the system can recognize the characters on a one character level, So it won't be able to recognize the Word and sentence and paragraph level.

2. Miss-classification of some images that contain bad written characters, or the complex one.

3. For the Latin handwritten characters, we can recognize the characters on word-level and one characters-level, Which we miss the sentence and paragraph level.

4. Our system miss the classification of some symbols.

5. The system won't recognize on any type of images.

3.7 Conclusion

The objective of this chapter was to present the general and detailed design of our character recognition system using deep learning as well as to implement and test the effectiveness of new architectures in the convolutional neural network method (CNN).

According to the different tests performed and the results obtained, we could deduce that our character recognition system is efficient and more accurate.

General Conclusion

The technological developments of the last twenty years have allowed digital systems to invade our daily lives. There is more to demonstrate that digital occupies a place of choice in the world today. Among the major components of digital systems, great importance is given to the image. The representation and processing of digital images is the subject of very active research at present. The processing of images is a very vast field that has known, and still knows, an important development for a few decades.

Artificial intelligence has come a long way in recent years, it has applications everywhere. Deep learning is one of the most popular learning techniques today. Deep learning has absolutely dominated the field of processing and thus recognition of images. Computer vision (image recognition) is one of the hottest topics in artificial intelligence. Its power is so revolutionary that almost every day we are seeing technical advances in image processing services. Today, image recognition techniques based on Deep Learning are increasingly used in industry with for example applications in medical imaging and security. One of the classes of deep learning that represents an interesting method for image processing and recognition is that of convolutional neural networks.

Among the technologies of image recognition, character recognition has become a very interesting and challenging field of study. Character recognition is a computer process that makes it possible to recognize, in an image, the letters composing a text. This makes it possible to transform an image file into a text file. The main interest of this technique is to be able to then search in a text, as well as to select words or sentences of the same text. When we talk about character recognition in images, we need a reliable and accurate system that really extracts accurate results in a timely manner.

In this master project, we focus on character recognition using deep learning in order to improve the accuracy and training time of this kind of character recognition. To do this, we propose new architectures in the CNN method based on certain types of character recognition; handwritten characters in Arabic and Latin, as another type one has optical character recognition.

Our memory and organized in 3 chapters with a general introduction and a conclusion. In the first chapter, we present the field of digital image recognition. Through the second chapter, we provide an introduction to the field of deep learning. The third chapter deals with the design of our solution and the implementation of our new architectures and the results obtained.

About our perspective for the future works, our idea and objective will be improving the whole system of character recognition with some important add-on:

- Trying to add another languages, also characters for the handwritten Arabic and Latin.
- Make our system able to recognize from very difficult images.
- Add the word recognition for the Arabic handwritten characters, also Line and text recognition for the Latin handwritten characters.
- Improving the Optic datasets with another fonts and sizes, and trying to make fonts recognition only for the optic characters.
- Add more functionalities to the front-end mobile version.

Bibliography

1. Szeliski, R. *Computer Vision: Algorithms and Applications* doi:10.1007/978-1-84882-935-0 (Jan. 2011).
2. *the human eye* https://courses.lumenlearning.com/boundless-physics/chapter/the-human-eye/. Visited: 20-12-2018.
3. Wyszecki, G. & S. Stiles, W. Color Science: Concepts and Methods, Quantitative Data and Formulae, 2nd Edition. *Color Science: Concepts and Methods, Quantitative Data and Formulae, 2nd Edition, by Gunther Wyszecki, W. S. Stiles, pp. 968. ISBN 0-471-39918-3. Wiley-VCH , July 2000* (July 2000).
4. Hunt, R. *The Reproduction of Colour (6th ed)* (Chichester UK: Wiley–ISandT Series in Imaging Science and Technology, 2004).
5. *Color* https://en.wikipedia.org/wiki/ColorDevelopment_of_theories_of_color_vision. Visited: 20-12-2018.
6. Von Helmholtz, H. *Physiological Optics – The Sensations of Vision* (Cambridge: MIT Press, 1866).
7. Palmer, S. *Vision Science: From Photons to Phenomenology* (Jan. 1999).
8. Economic and Social Research Council – Science in the Dock, Art in the Stocks (2007).
9. *Spectral color* https://en.wikipedia.org/wiki/Spectral_color. Visited: 20-12-2018.
10. *Feature Extraction* https://deepai.org/machine-learning-glossary-and-terms/feature-extraction. Visited: 20/12/2018.
11. Ayodele, T. Introduction to Machine Learning. doi:10.5772/9394 (Feb. 2010).
12. *Edge detection* https://en.wikipedia.org/wiki/Edge_detection. Visited: 21-12-2018.
13. *Corner detection* https://en.wikipedia.org/wiki/Corner_detection. Visited: 21-12-2018.
14. *Blob detection* https://en.wikipedia.org/wiki/Blob_detection. Visited: 22-12-2018.
15. *Ridge detection* https://en.wikipedia.org/wiki/Ridge_detection. Visited: 22-12-2018.

16. *Scale-invariant feature transform* https://en.wikipedia.org/wiki/Scale-invariant_feature_transform. Visited: 22-12-2018.

17. *Autocorrelation* https://en.wikipedia.org/wiki/Autocorrelation. Visited: 22-12-2018.

18. Stockman, G. & Shapiro, L. G. *Computer Vision* 1st. ISBN: 0130307963 (Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001).

19. Barghout, L. & Lee, L. W. *Perceptual information processing system* (Paravue Inc. U.S. Patent Application, 2003).

20. *Image Recognition* https://www.techopedia.com/definition/33499/image-recognition. Visited: 18-01-2019.

21. Lewis, M. B. & Ellis, H. D. How we detect a face: A survey of psychological evidence. *International Journal of Imaging Systems and Technology* **13,** 3–7 (2003).

22. *Facial motion capture* https://en.wikipedia.org/wiki/Facial_motion_capture. Visited: 18-01-2019.

23. *Facial motion system.* https://en.wikipedia.org/wiki/Facial_recognition_system. Visited: 18-01-2019.

24. *Powershot s5 review.* https://www.dcresource.com/reviews/canon/powershot_s5-review/index.shtml. Visited: 18-01-2019.

25. *tesco face detection sparks needless surveillance panic facebook fails with teens do.* https://www.theguardian.com/technology/2013/nov/11/tesco-face-detection-sparks-needless-surveillance-panic-facebook-fails-with-teens-do. Visited: 18-01-2019.

26. *the privacy issue of facial recognition.* https://www.amarvelfox.com/ibm-has-to-deal-with-the-privacy-issue-of-facial-recognition.html. Visited: 18-01-2019.

27. *Object Detection.* https://towardsdatascience.com/object-detection-with-10-lines-of-code-d6cb4d86f606. Visited: 18-01-2019.

28. Donahue, J. *et al.* Long-term recurrent convolutional networks for visual recognition and description, 2625–2634 (June 2015).

29. *Automatic target recognition* https://en.wikipedia.org/wiki/Automatic_target_recognition. Visited: 29-01-2019.

30. Se, S., Lowe, D. & Little, J. Vision-Based global localization and mapping for mobile robots. *Robotics, IEEE Transactions on* **21,** 364–375 (July 2005).

31. Heikkilä, J. & Silvén, O. A real-time system for monitoring of cyclists and pedestrians. *Image and Vision Computing* **22,** 563–570 (July 2004).

32. Esteva, A. *et al.* Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **542.** doi:10.1038/nature21056 (Jan. 2017).

33. Filho, P., Soares de Oliveira, L., Nisgoski, S. & S. Britto Jr, A. Forest species recognition using macroscopic images. *Machine Vision and Applications* **25.** doi:10.1007/s00138-014-0592-7 (May 2014).

34. Lameski, P. Plant Species Recognition Based on Machine Learning and Image Processing (Nov. 2017).

35. Strachan, N., Nesvadba, P. & Allen, A. Fish species recognition by shape analysis of images. *Pattern Recognition* **23,** 539–544. ISSN: 0031-3203 (1990).

36. Yu, X. *et al.* Automated identification of animal species in camera trap images. *EURASIP Journal on Image and Video Processing* **2013,** 52. ISSN: 1687-5281 (Sept. 2013).

37. Huang, T. & Aizawa, K. Image processing: Some challenging problems. English (US). *Proceedings of the National Academy of Sciences of the United States of America* **90,** 9766–9769. ISSN: 0027-8424 (Nov. 1993).

38. *Image enhancement* https://www.mathworks.com/discovery/image-enhancement.html. Visited: 05-02-2019.

39. *OCR Document* https://dev.havenondemand.com/apis/ocrdocumentoverview. Visited: 05-02-2019.

40. *OCR Document* https://www.collinsdictionary.com/dictionary/english/character-recognition. Visited: 05/02/2019.

41. Tappert, C. C., Suen, C. Y. & Wakahara, T. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12,** 787–808. ISSN: 0162-8828 (Sept. 1990).

42. *Intelligent word recognition* https://en.wikipedia.org/wiki/Intelligent_word_recognition. Visited: 09-02-2019.

43. *Intelligent character recognition* https://en.wikipedia.org/wiki/Intelligent_character_recognition. Visited: 09-02-2019.

44. *Optical Character Recognition (OCR) – How it works* https://www.nicomsoft.com/optical-character-recognition-ocr-how-it-works/. Visited: 10-02-2019.

45. Sezgin, M. & Sankur, B. Survey over image thresholding techniques and quantitative performance evaluation. *Journal of Electronic Imaging* **13,** 146–168 (Jan. 2004).

46. Trier Oeivind Due; Jain, A. K. Goal-directed evaluation of binarisation methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17,** 1191–120 (Feb. 2015).

47. Milyaev, S., Barinova, O., Novikova, T., Kohli, P. & Lempitsky, V. Image Binarization for End-to-End Text Understanding in Natural Images f8. *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR,* 128–132 (Aug. 2013).

48. *Image Optimization* https://www.bisok.com/grooper-data-capture-method-features/image-processing/. Visited: 10-02-2019.

49. Pati PB Ramakrishnan, A. Word Level Multi-script Identification. *Pattern Recognition Letters* **29,** 1218–1229 (Sept. 1987).

50. *Basic OCR in OpenCV* http://blog.damiles.com/2008/11/20/basic-ocr-in-opencv.html. Visited: 11-02-2019.

51. Smith, R. An Overview of the Tesseract OCR Engine. **2,** 629–633. ISSN: 1520-5363 (Sept. 2007).

52. *What's OCR?* http://www.dataid.com/aboutocr.htm. Visited: 11-02-2019.

53. *How-ocr-software-works?* http://ocrwizard.com/ocr-software/how-ocr-software-works.html. Visited: 11-02-2019.

54. *The basic patter recognition and classification with openCV* http://blog.damiles.com/2008/11/14/the-basic-patter-recognition-and-classification-with-opencv.html. Visited: 11-02-2019.

55. *How ocr works* https://www.explainthatstuff.com/how-ocr-works.html. Visited: 14-02-2019.

56. *Javascript Using OCR and Entity Extraction for LinkedIn Company* https://community.havenondemand.com/t5/Blog/javascript-Using-OCR-and-Entity-Extraction-for-LinkedIn-Company/ba-p/460. Visited: 14-02-2019.

57. *how to crack captchas* https://www.andrewt.net/blog/how-to-crack-captchas/. Visited: 15-02-2019.

58. *Breaking a Visual CAPTCHA* https://www2.cs.sfu.ca/~mori/research/gimpy/. Visited: 16-02-2019.

59. *OCR and Neural Nets in JavaScript* https://johnresig.com/blog/ocr-and-neural-nets-in-javascript/. Visited: 16-02-2019.

60. Patterson, J. & Gibson, A. *Deep Learning: A Practitioner's Approach* (" O'Reilly Media, Inc.", 2017).

61. Subarna, D. http://www.engineeringenotes.com/artificial-intelligence-2/neural-network-artificial-intelligence-2/structure-of-neural-network-artificial-intelligence/35410. Visited: 25/02/2019.

62. Haykin, S. *Neural Networks and Learning Machine* (Jan. 2008).

63. Mitchell, T. M. *Machine Learning* (1997).

64. *Reinforcement learning* https://en.wikipedia.org/wiki/Reinforcement_learning. Visited: 25-02-2019.

65. Blum, A. & Mitchell, T. Combining Labeled and Unlabeled Data with Co-Training. *Proceedings of the Annual ACM Conference on Computational Learning Theory.* doi:10.1145/279943.279962 (Oct. 2000).

66. Belkin, M., Niyogi, P. & Sindhwani, V. Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples. *Journal of Machine Learning Research* **7,** 2399–2434 (Nov. 2006).

67. Goodfellow, I., Bengio, Y. & Courville, A. *Deep Learning* http://www.deeplearningbook.org Visited: 25/02/2019.

68. Baldi, P. Autoencoders, unsupervised learning, and deep architectures. *Journal of Machine Learning Research, Workshop and Conference Proceedings* **27,** 37–50 (Jan. 2012).

69. Khan, A., Zameer, A., Jamal, T. & Raza, A. Deep Belief Networks Based Feature Generation and Regression for Predicting Wind Power Architectures. *ArXiv e-prints* (2018).

70. Creswell, A. *et al.* Generative Adversarial Networks: An Overview. *ArXiv e-prints* (2017).

71. Gersey, B. Generative Adversarial Networks. *degree of Master of Advanced Study in Mathematics* (2018).

72. R. Medsker, L. & C. Jain, L. Recurrent Neural Networks: Design and Applications (Jan. 1999).

73. Du, K.-L. & Swamy, M. in, 337–353 (Dec. 2014). ISBN: 978-1-4471-5570-6. doi:10.1007/978-1-4471-5571-3_11.

74. Chinea Manrique de Lara, A. Understanding the Principles of Recursive Neural Networks: A Generative Approach to Tackle Model Complexity. doi:10.1007/978-3-642-04274-4_98 (Nov. 2009).

75. R. Bowman, S., Potts, C. & Manning, C. Recursive Neural Networks Can Learn Logical Semantics. doi:10.18653/v1/W15-4002 (June 2014).

76. *Recursive neural network* https://en.wikipedia.org/wiki/Recursive_neural_network. Visited 01-03-2019.

77. *Réseau neuronal convolutif* https://fr.wikipedia.org/wiki/RÃľseau_neuronal_convolutif. Visited 01-03-2019.

78. Deshpande, A. *A Beginner's Guide To Understanding Convolutional Neural Networks* https://adeshpande3.github.io/A-Beginner27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/.Visited: 03-03-2019.

79. Deeplizard. *Machine Learning & Deep Learning Fundamentals* http://deeplizard.com/learn/video/qSTv_m-KFk0. Visited: 03-03-2019.

80. Prabhu. *Understanding of Convolutional Neural Network (CNN) Deep Learning* https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. Visited: 08-03-2019.

81. Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE* **86,** 2278–2324 (Dec. 1998).

82. Krizhevsky, A., Sutskever, I. & E. Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems* **25.** doi:10.1145/3065386 (Jan. 2012).

83. Zeiler, M. & Fergus, R. Visualizing and understanding convolutional networks. *European Conference on Computer Vision(ECCV)* **8689,** 818–833 (Jan. 2013).

84. Szegedy, C. *et al.* Going deeper with convolutions. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 1–9 (June 2015).

85. Simonyan, K. & Zisserman, A. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556* (Sept. 2014).

86. He, K., Zhang, X., Ren, S. & Sun, J. Deep Residual Learning for Image Recognition, 770–778 (June 2016).

87. Hogeweg, P. The Roots of Bioinformatics in Theoretical Biology. *PLoS computational biology* **7,** e1002021 (Mar. 2011).

88. Attwood T.K. Gisel A., E. N.-E. & E, B.-R. Bioinformatics - Trends and Methodologies. *InTech* (2011).

89. Baldi, P. & Brunak, S. Bioinformatics: The Machine Learning Approach. *MIT Press* (Jan. 2001).

90. "Deep learning" : les dessous d'une technologie de rupture. *Futurible.*

91. *Qu'est-ce que l'apprentissage profond?* https://www.netapp.com/fr/info/what-is-deep-learning.aspx. Visited: 20-03-2019.

92. Cireşan, D., Meier, U., Masci, J. & Schmidhuber, J. Multi-Column Deep Neural Network for Traffic Sign Classification. *Neural networks : the official journal of the International Neural Network Society* **32,** 333–8 (Feb. 2012).

93. Cai, M., Shi, Y. & Liu, J. Deep maxout neural networks for speech recognition. *2013 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2013 - Proceedings,* 291–296 (Dec. 2013).

94. Luke. *Image Colorization with Convolutional Neural Networks* https://lukemelas.github.io/image-colorization.html. Visited: 25-03-2019.

95. Nguyen, T., Mori, K. & Thawonmas, R. Image Colorization Using a Deep Convolutional Neural Network (Apr. 2016).

96. Farabet Clement, e. a. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence,* 1915–1929. (2013).

97. Guyon, I., Albrecht, P., Lecun, Y., Denker, J. & E. Hubbard, W. Design of a neural network character recognizer for a touch terminal. *Pattern Recognition* **24,** 105–119 (Dec. 1991).

98. Bengio, Y., Lecun, Y. & Henderson, D. Globally Trained Handwritten Word Recognizer Using Spatial Representation, Convolutional Neural Networks, and Hidden Markov Models. 937–944 (Jan. 1993).

99. Bhatt, P. & Patel, I. Optical Character Recognition using Deep learning – A Technical Review. *National Journal of System and Information Technology (NJSIT)* **11** (June 2018).

100. Mikolov Tomas, e. a. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781 (* (2013).

101. Elsawy, A., Loey, M. & El-Bakry, H. Arabic Handwritten Characters Recognition using Convolutional Neural Network. *WSEAS TRANSACTIONS on COMPUTER RESEARCH* **5,** 11–19 (Jan. 2017).

102. Van Schaik, G. C. S. A. J. T. A. *The EMNIST Dataset* `https://www.nist.gov/node/1298471/emnist-dataset`. Visited: 16-04-2019.

103. Tapson, G. C. S. A. J. & van Schaik, A. EMNIST: an extension of MNIST to handwritten letters. *arXiv:1702.05373v1,* 10 (Feb. 2017).

104. Bonner, A. *The complete beginner's guide to data cleaning and preprocessing* `https://towardsdatascience.com/the-complete-beginners-guide-to-data-cleaning-and-preprocessing-2070b7d4c6d`. Visited: 01-05-2019.

105. Sharma, M. *What Steps should one take while doing Data Preprocessing?* `https://hackernoon.com/what-steps-should-one-take-while-doing-data-preprocessing-502c993e1caa`. Visited: 01-05-2019.

106. Dey, A. *Data Preprocessing for Machine Learning* `https://medium.com/datadriveninvestor/data-preprocessing-for-machine-learning-188e9eef1d2c`. Visited: 23-05-2019.

107. *Feature scaling* `https://en.wikipedia.org/wiki/Feature_scaling`. Visited: 25-05-2019.

108. Zhang, H., Liu, D. & Xiong, Z. CNN-based text image super-resolution tailored for OCR, 1–4 (Dec. 2017).

Listings