الجمـــهوريـــة الجـــزائريـــة الديـــمقراطيـــة الشعبيـــة
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة التعـلـيـــم العـالـــي والبـــحث العلـــــمي
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH
جـــامـعـة عمـــار ثليـجـي بالأغـــواط
UNIVERSITY OF AMAR TELIDJI LAGHOUAT

كـلـيـــة العلـــــوم
FACULTY OF SCIENCES
قـســـم الـرياضيـات والإعـــلام الآلـــي
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE

## *Master Thesis*

**Domain :** Mathematics and Computer Science

**Field :** Computer Science

**Option :** Networks, Distributed Systems and Applications

*By:*

**Abdelkarim Ben Sada**

*Topic*

# Machine Learning for Algerian Car Plate Number Recognition

*Defended publicly in front of the jury composed of:*

| | | |
|---|---|---|
| *Mr. Nasreedine Lagraa* | *Professor* | *President* |
| *Mr. Abdallah Lakhdari* | *M.C.(A)* | *Examiner* |
| *Mrs. Fatna Guibadj* | *M.C.(A)* | *Examiner* |
| *Mr. Mohamed Lahcen Bensaad* | *M.C.(A)* | *Supervisor* |

2015/2016

الجمـــهوريـــة الجـــزائريـــة الديـمقـراطيـــــــة الشـــعبيـــــة
REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعـلـيـم الـعـالـــي والـبـحـث الـعـلـمي
MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

جـامـعـة عمّـــار ثـلـيـــجـــي بالأغـــواط
UNIVERSITE AMAR TELIDJI LAGHOUAT

كـــلـيـة الـــــعلـــــوم
FACULTE DES SCIENCES

قـســـــم الـريـاضيـــات والإعـــلام الآلـــي
DEPARTEMENT DE MATHEMATIQUES ET INFORMATIQUE

## *Mémoire de MASTER*

**Domaine :** Mathématiques et Informatique
**Filière :** Informatiques
**Option :** Réseaux, Systèmes et Applications Réparties

*Par:*
**Abdelkarim Ben Sada**

*Thème*

# Machine Learning for Algerian Car Plate Number Recognition

*Soutenu publiquement devant le jury composé de:*

| | | |
|---|---|---|
| *Mr. Nasreedine Lagraa* | *Professeur* | *Président* |
| *Mr. Abdallah Lakhdari* | *M.C.(A)* | *Examinateur* |
| *Mme. Fatna Guibadj* | *M.C.(A)* | *Examinatrice* |
| *Mr. Mohamed Lahcen Bensaad* | *M.C.(A)* | *Encadreur* |

Année Universitaire 2015/2016

*This work is dedicated to my parents,*
*family and friends.*

## *Acknowledgments*

*I would like to express my gratitude to everyone who supported me throughout the course of making this project.*

*I am thankful to my supervisor Dr. M.L Bensaad for his aspiring guidance, invaluably constructive criticism and friendly advice during the project work.*

*A special thanks to my Uncle B. Djendaoui for the French translation of the abstract.*

*The greatest gratitude goes to my parents, for their continuous support.*

*Many thanks to all my friends, M. Hamdini, M. Naoumi and all the ones that I didn't mention.*

*A. Ben Sada*

## ملخص

في الآونة الأخيرة تم استخدام أنظمة التعرف الآلي على لوحات ترقيم السيارات في مناطق كثيرة. هذه الأنظمة لها مجالات تطبيقية عديدة، منها التشغيل الآلي لمواقف السيارات، التحكم في صلاحيات الوصول لمناطق معينة، مراقبة وتعقب المركبات. في هذا المجال، تم اقتراح العديد من الحلول إلا انها غير موجهة خصيصا للسوق الجزائرية.

في هذ المشروع انجزنا نظام تعرف آلي على لوحات الترقيم الخاصة بالمركبات الجزائرية، والذي استخدمنا فيه خوارزميات الرؤية الاصطناعية وتعلم الآلة. بحيث قسّمنا النظام إلى ثلاثة مراحل: استخراج لوحة الترقيم، ثم تقسيمها إلى أرقام منفردة ومن بعد التعرف على كل رقم منها. باستعمال معلومات الحواف والإسقاطات استطعنا تحديد مكان لوحة الترقيم. ثم بإيجاد الفراغات الموجودة فيها قمنا بتقسيمها إلى قطع تحوي أرقام اللوحة. للتعرف على الأرقام المستخرجة علّمنا شبكة عصبونية اصطناعية على قراءة هذه الأرقام.

أظهرت اختبارات هذا النظام على مجموعة من الصور الملتقطة من سيارات جزائرية نتائج جيدة. مع أنه ليس منتجا نهائيا، إلا أنه بالإمكان تحسينه ليكون جاهزا للاستخدام الميداني.



**كلمات مفتاحية:** التعرف الآلي على لوحات ترقيم السيارات، تعلم الآلة، الرؤية الاصطناعية، الشبكات العصبية الاصطناعية.

**Résumé**

Dans les dernières années, la Reconnaissance Automatique de Plaques d'Immatriculation (RAPI) a été déployée dans plusieurs domaines. Parmi lesquels l'automatisation de parking, le contrôle d'accès à des régions spécifiques, la surveillance et la poursuite des véhicules. Différentes implémentations de RAPI existent, mais aucune d'elles n'est destinée pour le marché algérien.

Dans ce travail, on tente de mise en œuvre un système RAPI prévu pour les véhicules algériens. Pour le construire, on a utilisé les algorithmes de la vision artificielle et de l'apprentissage de la machine. Ce système a été divisé en trois étapes : l'extraction, la segmentation et la reconnaissance de la plaque.
D'abord, on utilise les informations des contours et les projections pour localiser la plaque d'immatriculation. Ensuite, par la détection des espaces dans la plaque, on est capable de la segmenter en chiffres. Pour la reconnaître, on a formé un réseau de neurones pour lire les chiffres extraits.

Les tests de notre système sur des images prises des voitures algériennes donnent de bons résultats. Bien qu'il ne soit pas un produit fini, il peut être amélioré pour le déploiement sur le terrain.

**Mots clés:** La reconnaissance automatique de plaques d'immatriculation, RAPI, l'apprentissage de la machine, vision artificielle, les réseaux de neurones.

**Abstract**

In the recent years, Automatic Number Plate Recognition (ANPR) systems have been deployed in many areas. They have many applications including parking automation, access control, monitoring and tracking vehicles. Various implementations of ANPR solutions exist, only none of them target the Algerian market.

In this work we try to implement an ANPR system for Algerian vehicles. To build this system we used machine vision and machine learning algorithms. The system was divided into three steps: license plate extraction, segmentation and recognition. We used edge information and projections for locating the license plate. Then by detecting spaces in the plate we were able to segment it into digits. We trained a neural network on reading Algerian license plate numbers. This network was used to recognize the extracted digits.

Testing the system on a data set of images captured from Algerian cars showed some good results. Although this system is not a finished product, it can be further improved for field deployment.

**Keywords:** Automatic Number Plate Recognition, ANPR, Machine Learning, Machine Vision, Neural Networks.

# Table of Contents

# List of Figures

# List of Tables

# List of Algorithms

# Introduction

There are over 1 billion cars in the world, and over 5 million in Algeria. All of these vehicles have an identification number, it's also known as the license number. It is usually written in a plate mounted onto the car's body[1, 2].

To process and analyze large amounts of data, people resort to computers because they are efficient and extremely fast. It is obvious that the license number is the most important identification data, which a computer system should treat when dealing with vehicles.

Automatic number plate recognition (ANPR) is the automation of inputing a car's identification number into a computer, which requires a considerable amount of manpower if done manually. More formally, an ANPR system is an integrated hardware + software device that reads the vehicle's license plate, and outputs the number in a computer encoded text, to be used by some data processing system.

ANPR systems are deployed in a wide range of applications, including: parking automation and security, controlling access to areas, monitoring and tracking vehicles, law enforcement and many more.

Many countries and organizations have successfully deployed ANPR into their systems, for example the United Kingdom have nearly 8000 cameras capturing between 25 and 30 million ANPR read records every day. Other countries include the United States, Hungary and Ukraine have also introduced such a system[3].

Many implementations of ANPR systems exist, some of them claim to support many country license plates including Algeria, but to the extent of our knowledge, non of them is specific to Algerian license plates. In this work we try to develop an ANPR platform, from the ground up, considering only Algerian vehicles. This gives our platform the advantage of being more suitable to Algerian license plates, what might result to more robustness and accuracy.

Developing an ANPR system involves solving two major problems. The first one consists of giving a computer the ability to locate the license plate in a given car image. The second problem is how to make the computer read numbers from an image. To deal with these problems we have looked to two computer science fields, machine vision and machine learning.

We start this document by briefly introducing some machine vision algorithms in the first chapter, followed by machine learning in the second chapter. And then we mention some methods used by similar works in the third chapter. Finally in the fourth chapter, we explain the methods and techniques we used to build the system, in addition to presenting test results.

# Chapter 1

# Background

Automatic Number Plate Recognition (ANPR) systems are considered part of a discipline called machine vision. The first step in realizing such kind of systems is the extraction of useful information from captured images. In this chapter, we introduce and explain some of the most common methods and algorithms that are used for analyzing and extracting features from images.

## 1.1 Gray-scaling

A gray-scale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

We convert images to gray-scale because processing them is much easier and faster. There are three methods to convert a pixel $P$ from the RGB (Red, Green, Blue) color space to gray-scale. The lightness method averages the most and least prominent colors:

$$P_{intensity} = (max(P_R, P_G, P_B) + min(P_R, P_G, P_B))/2$$

The average method simply averages the values:

$$P_{intensity} = (P_R + P_G + P_B)/3$$

The luminosity method is a more sophisticated version of the average method. It also averages the values, but it forms a weighted average to account for human perception. We are more sensitive to green than other colors, so green is weighted most heavily. The formula for luminosity method is[4]:

$$P_{intensity} = 0.21P_R + 0.72P_G + 0.07P_B$$

## 1.2 Histogram Equalization

Histogram equalization redistributes the pixel intensity values evenly by using cumulative(sum) histogram as a transfer function or as a look-up table. In histogram equalization, the input pixel intensity $x$ is transformed to new value $x'$ by $T$. A transform function $T$ is the product of a cumulative histogram and a scale factor. The scale factor is needed to fit the new intensity value within the range of the intensity values[5].

$$x' = T(x) = \frac{MaxIntensity}{N} \sum_{i=0}^{x} n_i$$

where $n_i$ is the number of pixels at intensity $i$, $N$ is the total number of pixels in the image. A demonstration is shown in Figure 1.1, and the pseudo-code is shown in algorithm 1.

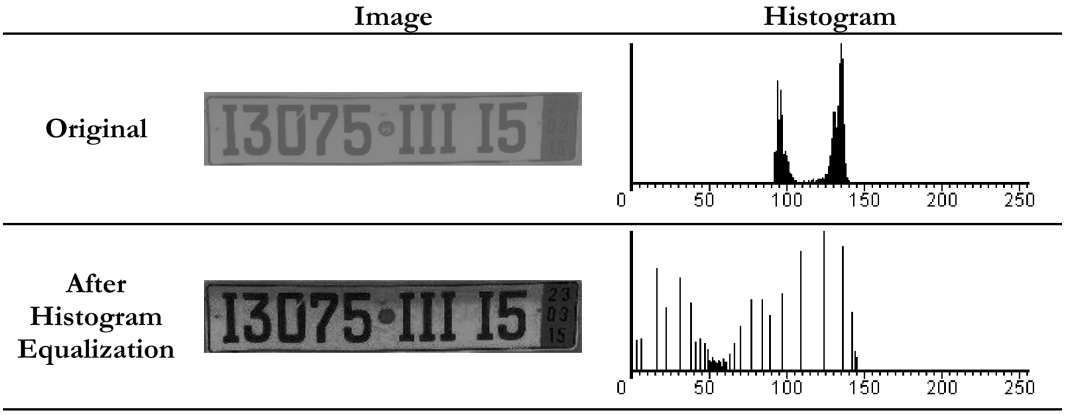



**Figure 1.1:** An example showing the effect of histogram equalization.

**Algorithm 1** Histogram equalization

Suppose a gray-scale image $I$, its histogram $H$, $M$ is the maximum intensity in $H$, *pxCount* is the number of pixels in $I$, $I$' is the histogram equalized $I$.
*cumHist* = $H$
**for** $i = 1$ *to* 255 **do**
  *cumHist*[i] += *cumHist*[i-1]
**end for**
**for each** pixel $(i, j)$ in $I$ **do**
  $I$'[i][j] = $M$ * cumHist[image[i][j]] / *pxCount*
**end for**

## 1.3 Convolution

Convolution is a simple mathematical operation which is fundamental to many common image processing operators. Convolution provides a way of multiplying together two arrays of numbers, generally of different sizes, but of the same dimensionality, to produce a third array of numbers of the same dimensionality. This can be used in image processing to implement operators whose output pixel values are simple linear combinations of certain input pixel values[6].

In the context of image processing, one of the input arrays is normally just a gray-level image. The second array is usually much smaller, and is known as the kernel. If the image has $M$ rows and $N$ columns, and the kernel has $m$ rows and $n$ columns, then the size of the output image will have $M - m + 1$ rows, and $N - n + 1$ columns. Mathematically we can write the convolution as shown in formula 1.1, and in pseudo-code as shown in Algorithm 2.

$$V = \left| \frac{1}{F} \sum_{i=1}^{q} \sum_{j=1}^{q} f_{i,j} * d_{i,j} \right| \tag{1.1}$$

where:

$f_{i,j}$ : the coefficient of a convolution kernel at position i, j in the kernel.
$d_{i,j}$ : the data value that corresponds to $f_{i,j}$
$q$ : the dimension of the kernel, assuming a square kernel.
$F$ : either the sum of the kernel's coefficients, or 1 if the sum equals 0.
$V$ : the output pixel value.

In cases where V is less than 0, V is set to 0.

Kernel convolution usually requires values from pixels outside of the image boundaries. There are a variety of methods for handling image edges:

- **Wrap**: The image is conceptually wrapped (or tiled) and values are taken from the opposite edge or corner.
- **Crop**: Any pixel in the output image which would require values from beyond the edge is skipped. This method can result in the output image being slightly smaller, with the edges having been cropped.

**Algorithm 2** Convolution

```
for each image row in input image do
  for each pixel in image row do
    Set accumulator to zero.
    for each kernel row in kernel do
      for each element in kernel row do
        if element position corresponding* to pixel position then
          Multiply element value corresponding to pixel value.
          Add result to accumulator.
        end if
      end for
    end for
    Set output image pixel to accumulator.
  end for
end for
```

## 1.4 Blurring

Blurring is often used in pre-processing steps, such as removal of small details from an image prior to large object extraction. Below we explain two common blurring filters[7].

### 1.4.1 Mean Filter

Mean filtering is a simple method of smoothing images. It works by reducing the amount of intensity variation between one pixel and the next. It is often used to reduce noise in images[8].

The idea of mean filtering is simply to replace each pixel value in an image with the mean (average) value of its neighbors, including itself. This has the effect of eliminating pixel values which are unrepresentative of their surroundings. Mean filtering is usually thought of as a convolution filter. Like other convolutions it is based around a kernel, as shown below:

$$K = \begin{pmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{pmatrix}$$

*Corresponding input image pixels are found relative to the kernel's origin.

### 1.4.2 Gaussian Filter

A Gaussian blur is the result of blurring an image by a Gaussian function. It is widely used to reduce image noise and detail. A Gaussian blur effect is typically generated by convolving an image with a kernel of Gaussian values.

A Gaussian kernel can be calculated with a Gaussian function as shown in formula 1.2.

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp(-\frac{x^2 + y^2}{2\sigma^2}) \tag{1.2}$$

where $x$ is the distance from the origin in the horizontal axis, $y$ is the distance from the origin in the vertical axis, and $\sigma$ is the standard deviation of the Gaussian distribution. A demonstration example of Gaussian blur using the following kernel is shown in Figure 1.2.

$$G = \begin{pmatrix} 0.007 & 0.019 & 0.028 & 0.019 & 0.007 \\ 0.019 & 0.055 & 0.078 & 0.055 & 0.019 \\ 0.028 & 0.078 & 0.111 & 0.078 & 0.028 \\ 0.019 & 0.055 & 0.078 & 0.055 & 0.019 \\ 0.007 & 0.019 & 0.028 & 0.019 & 0.007 \end{pmatrix}$$


**Figure 1.2:** An example showing the effect of applying a Gaussian blur.

## 1.5 Edge Detection

Edge detection is a fundamental method for feature extraction. In the general case, the result of applying an edge detection algorithm is an object boundary with connected curves (see Figure 1.3). The difficulty lies in images containing complex scenes, as it might result to boundaries with disconnected curves. Different edge detection operators exist, but here we explain the Sobel edge detector.

The Sobel operator performs a 2-D spatial gradient measurement on an image and emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in an input gray-scale image. The operator consists of a pair of $3 \times 3$ convolution kernels as shown below[9]:

$$Kx = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \quad Ky = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

These kernels are designed to respond maximally to edges running vertically and horizontally relative to the pixel grid, one kernel for each of the two perpendicular orientations. The kernels $Kx$ and $Ky$ can be applied to the input image separately to extract vertical and horizontal edges respectively, and to produce separate measurements of the gradient component $G$ in each orientation ($Gx$ and $Gy$ respectively). These gradients can then be combined together to find the absolute magnitude of the gradient at each point and the orientation of that gradient. The gradient magnitude is given by:

$$|G| = \sqrt{Gx^2 + Gy^2}$$

The gradient orientation is given by:

$$\theta = arctan(Gy/Gx)$$

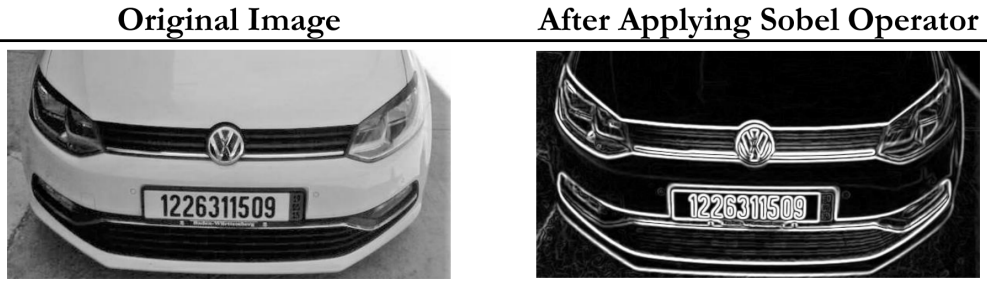



**Figure 1.3:** An example showing the effect of applying the Sobel operator.

## 1.6 Object Extraction

Computer vision has a class of algorithms that are able to find instances of objects within a certain type of shapes. Here we explain the famous Hough transform technique.

The Hough transform is a technique which can be used to isolate features of a particular shape within an image. Because it requires that the desired features be specified in some parametric form, the classical Hough transform is most commonly used for the detection of regular curves such as lines, circles, ellipses, etc. A generalized Hough transform can be employed in applications where a simple analytic description of a feature is not possible. The main advantage of the Hough transform technique is that it is tolerant to gaps in feature boundary descriptions and is relatively unaffected by image noise.

The simplest case of Hough transform is detecting straight lines. In general, the straight line $y = mx + b$ can be represented as a point $(b, m)$ in the parameter space. However, vertical lines pose a problem. They would give rise to unbounded values of the slope parameter $m$. Thus, for computational reasons, Duda and Hart (1972) proposed the use of the Hesse normal form[10]:

$$r = x \cos\theta + y \sin\theta$$

where $r$ is the distance from the origin to the closest point on the straight line, and $\theta$ is the angle between the $x$ axis and the line connecting the origin with that closest point (see Figure 1.4). It is therefore possible to associate with each line of the image a pair $(r, \theta)$.

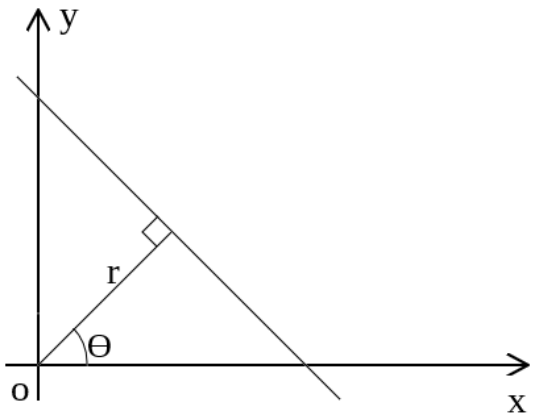



**Figure 1.4:** Line representation in Hesse normal form.

The linear Hough transform algorithm uses a two-dimensional array, called an accumulator, to detect the existence of a line described by $r = x\cos\theta + y\sin\theta$. The dimension of the accumulator equals the number of unknown parameters, i.e., two, considering quantized values of $r$ and $\theta$ in the pair $(r, \theta)$. For each pixel at $(x, y)$ and its neighborhood, the Hough transform algorithm determines if there is enough evidence of a straight line at that pixel. If so, it will calculate the parameters $(r, \theta)$ of that line, and then look for the accumulator's bin that the parameters fall into, and increment the value of that bin. By finding the bins with the highest values, typically by looking for local maxima in the accumulator space, the most likely lines can be extracted. The pseudo-code for the Hough transform is shown in algorithm 3.

**Algorithm 3** Hough transform

```
for all x do
  for all y do
    if edge point at (x, y) then
      for all θ do
        ρ = x cos θ + y sin θ
        Accumulator[θ][ρ] += 1
      end for
    end if
  end for
end for
```

An improvement suggested by O'Gorman and Clowes (1976)[11] can be used to detect lines if one takes into account that the local gradient of the image intensity will necessarily be orthogonal to the edge. Since edge detection generally involves computing the intensity gradient magnitude, the gradient direction is often found as a side effect. If a given point of coordinates $(x, y)$ happens to indeed be on a line, then the local direction of the gradient gives the $\theta$ parameter corresponding to that line, and the $r$ parameter is then immediately obtained. This reduces the computation time and has the interesting effect of reducing the number of useless votes, thus enhancing the visibility of the spikes corresponding to real lines in the image.

## 1.7 Binarization

Image binarization is the process of converting a gray-scale image into a black and white image (see Figure 1.5). The simplest method changes pixels to black if their intensity is less than a threshold, or to white otherwise. The process of finding the appropriate value for the threshold is called thresholding.


**Figure 1.5:** A binarized text image.

There are two kinds of thresholding global and local. The global thresholding calculates and applies a threshold to the whole image. It usually gives poor results in the case of text images, that's due to shadows and noise effects. Finding an optimal threshold with these effects is very difficult, one of its famous methods is Otsu's method[12].

The local thresholding calculates and applies a threshold to a certain area of an image. It gives better results in the case of text images, due to its adaptive technique, what makes it insusceptible to effects like noise and shadows. Two of its famous methods are Niblack's method[13] and Sauvola's method[14]. In this section we explain Sauvola's method.

Considering a gray-scale image in which $g(x, y) \in [0, 255]$ be the intensity of a pixel at location $(x, y)$. In local adaptive thresholding techniques, the aim is to compute a threshold $t(x, y)$ for each pixel such that[15]:

$$o(x,y) = \begin{cases} 0 & \text{if } g(x,y) \leq t(x,y) \\ 255 & otherwise \end{cases} \tag{1.3}$$

In Sauvola's binarization method, the threshold $t(x, y)$ is computed using the mean $m(x, y)$ and standard deviation $s(x, y)$ of the pixel intensities in a $w \times w$ window centered around the pixel $(x, y)$:

$$t(x,y) = m(x,y)\left[1 + k\left(\frac{s(x,y)}{R} - 1\right)\right] \tag{1.4}$$

where $R$ is the maximum value of the standard deviation ($R = 128$ for a gray-scale image), and $k$ is a parameter which takes positive values in the range $[0.2, 0.5]$. The local mean $m(x, y)$ and standard deviation $s(x, y)$ adapt the value of the threshold according to the contrast in the local neighborhood of the pixel.

The statistical constraint in Equation 1.4 gives very good results even for severely degraded images. However in order to compute the threshold $t(x, y)$, local mean and standard deviation have to be computed for each pixel. Computing $m(x, y)$ and $s(x, y)$ in a naive way results in a computational complexity of $O(W^2N^2)$ for an $N \times N$ image. In order to speed up the computation, Sauvola et al. (2000)[16] propose computing a threshold for every $n$th pixel and then using interpolation for the rest of the pixels. This speeds up the computation by some factors at the cost of reduced accuracy of determining the threshold. In addition, the computational complexity is still a quadratic function of the local window dimension. Shafait et al. (2008)[15] proposed an efficient way of computing local means and variances using sum tables (integral images) such that the computational complexity does not depend on the window dimension anymore.

An integral image $I$ of an input image $g$ is defined as the image in which the intensity at a pixel position is equal to the sum of the intensities of all the pixels above and to the left of that position in the original image. So the intensity at position $(x, y)$ can be written as:

$$I(x, y) = \sum_{i=0}^{x} \sum_{j=0}^{y} g(i, j) \tag{1.5}$$

The integral image of any gray-scale image can be efficiently computed in a single pass. Once we have the integral image, the local mean $m(x, y)$ for any window size can be computed simply by using one addition and two subtractions instead of the summation over all pixel values within that window:

$$\begin{aligned} m(x, y) = (I(x + w/2, y + w/2) + I(x - w/2, y - w/2) - \\ I(x + w/2, y - w/2) - I(x - w/2, y + w/2))/w^2 \end{aligned} \tag{1.6}$$

Similarly, the computation of the local variance:

$$s^2(x, y) = \frac{1}{w^2} \sum_{i=x-w/2}^{x+w/2} \sum_{j=y-w/2}^{y+w/2} g^2(x, y) - m^2(x, y) \tag{1.7}$$

The first term in Equation 1.7 can be computed in a similar way as in Equation 1.6 by using an integral image of the squared pixel intensities. Once computed the integral image of the pixel intensities and the square of the pixel intensities, local means and variances can be computed very efficiently, independent of the local window size. Using these formulas, the complexity of equation 1.4 can be reduced to $O(N^2)$. A demonstration of Sauvola's local thresholding is shown in Figure 1.6.

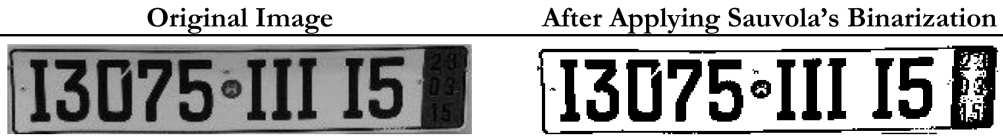



**Figure 1.6:** An example image thresholded using Sauvola's method.

## 1.8 Connected Component Labeling (CCL)

Connected-component labeling is used in computer vision to detect connected regions in binary digital images. There are many algorithms that can be used to extract connected components from images, but here we explain the simplest and fastest method that is called "One component at a time ". This algorithm is part of Vincent and Soille's watershed segmentation algorithm[17], and it works as follows:

Once the first pixel of a connected component is found, all the connected pixels of that component are labeled before going onto the next pixel in the image (see Figure 1.7).

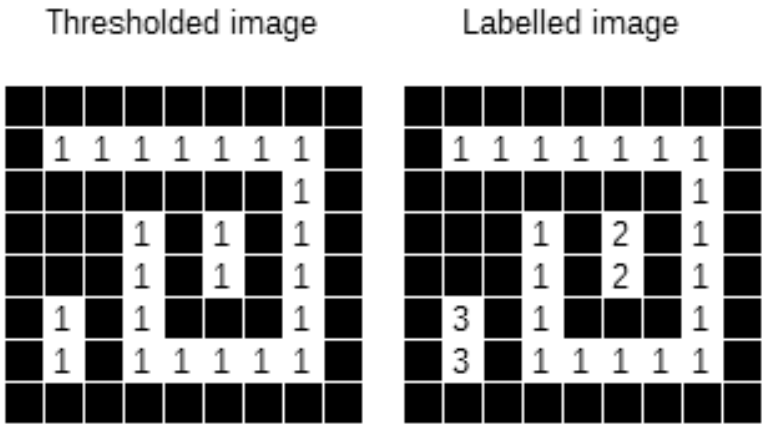



**Figure 1.7:** A binary image before and after CCL.

Assuming that the input image is a binary image, with pixels being either background or foreground and that the connected components in the foreground pixels are desired. The algorithm steps can be written as:

1. Start from the first pixel in the image. Set "curLabel" to 1. Go to (2).
2. If this pixel is a foreground pixel and it is not already labeled, then give it the label "curLabel" and add it as the first element in a queue, then go to (3). If it is a background pixel, then repeat (2) for the next pixel in the image.
3. Pop out an element from the queue, and look at its neighbors. If a neighbor is a foreground pixel and is not already labeled, give it the "curLabel" label and add it to the queue. Repeat (3) until there are no more elements in the queue.
4. Go to (2) for the next pixel in the image and increment "curLabel" by 1.

## 1.9 Conclusion

In this chapter we have seen a number of image processing methods and algorithms that are used to extract information from images. They can be categorized into two main categories. The preprocessing category includes gray-scaling, histogram equalization, blurring and binarization. The feature extraction category includes edge detection, object extraction and CCL. In the next chapters we use these methods to develop a robust ANPR platform for Algerian vehicles.

# Chapter 2

# Machine Learning

Since the invention of computers, humans have wondered whether they can be made to learn and improve automatically with experience. The impact would be dramatic. Over the years scientists have invented many ways that can achieve that goal. The science of developing algorithms and mathematical methods to make computers learn is called Machine Learning.

Machine learning explores the study and construction of algorithms that can learn from data and make predictions based on it. Such algorithms operate by building a model from example inputs in order to make data-driven predictions or decisions, rather than following strictly static program instructions.

## 2.1 Applications

Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms is infeasible. Learning algorithms have been successfully deployed in a variety of applications, including:

- Text or document classification, e.g., spam detection.
- Natural language processing, Speech recognition, speaker verification.
- Optical character recognition (OCR).
- Computational biology applications, e.g., protein function or structured prediction.
- Computer vision tasks, e.g., image recognition, face detection.

- Fraud detection (credit card, telephone) and network intrusion.
- Games.
- Unassisted vehicle control (robots, navigation).
- Medical diagnosis.
- Recommendation systems, search engines, information extraction systems.

The list provided above is not comprehensive. Machine Learning have many other applications, and new ones are being discovered everyday.

## 2.2 Types of Machine Learning Tasks

Machine learning applications correspond to a wide variety of learning problems. Some major classes of learning problems are[18]:

- **Classification**: It consists of assigning a category to an item. Some of its examples are: document classification, image classification, text classification, or speech recognition.
- **Regression**: It consists of predicting a real value for an item. Some of its examples are: prediction of stock values or variations of economic variables.
- **Ranking**: It consists of ordering items according to some criterion. One of its famous examples is Web search, returning web pages relevant to a search query.
- **Clustering**: It consists of partitioning items into homogeneous regions. Clustering is often performed to analyze very large data sets. For example, in the context of social network analysis, clustering algorithms attempt to identify "communities" within large groups of people.
- **Dimensionality reduction**: It consists of transforming a representation of items into a lower-dimensional representation of these items, while preserving some properties of the initial representation. A common example involves preprocessing digital images in computer vision tasks.

## 2.3 Terminology

- **Examples**: Items or instances of data used for learning or evaluation.
- **Features**: The set of attributes, often represented as a vector, associated to an example.
- **Labels**: Values or categories assigned to examples. In classification problems, examples are assigned specific categories.
- **Training sample**: Examples used to train a learning algorithm.
- **Validation sample**: Examples used to tune the parameters of a learning algorithm when working with labeled data.
- **Test sample**: Examples used to evaluate the performance of a learning algorithm. The test sample is separate from the training and validation data and is not made available in the learning stage.
- **Loss function**: A function that measures the difference, or loss, between a predicted label and a true label.

## 2.4 Types of Learning

There are many learning types, they differ in the type of training data that is available, the order and method by which training data is received and the test data used to evaluate the learning algorithm[18].

### 2.4.1 Supervised Learning

The learner receives a set of labeled examples as training data and makes predictions for all unseen points. This is the most common scenario associated with classification, regression, and ranking problems.

### 2.4.2 Unsupervised Learning

The learner receives unlabeled training data, and makes predictions for all unseen points. Since in general no labeled example is available in that setting, it can be difficult to quantitatively evaluate the performance of the algorithm. Clustering and dimensionality reduction are examples of unsupervised learning problems.

### 2.4.3 Semi-supervised Learning

The learner receives a training sample consisting of both labeled and unlabeled data, and makes predictions for all unseen points. Semi-supervised learning is common in settings where unlabeled data is easily accessible but labels are expensive to obtain. Various types of problems arising in applications, including classification, regression, or ranking tasks, can be framed as instances of semi-supervised learning.

### 2.4.4 Reinforcement Learning

The training and testing phases are intermixed in reinforcement learning. To collect information, the learner actively interacts with the environment and in some cases affects the environment, and receives an immediate reward for each action. The objective of the learner is to maximize his reward over a course of actions and interactions with the environment.

Many machine learning algorithms and methods have been developed in the last years, each one is suited for some kind of application. In the next section we introduce one of the most common learning methods which is called Artificial Neural Networks. It is inspired from biology, and it is widely used in various types of applications.

## 2.5 Artificial Neural Networks (ANNs)

Neural network learning methods provide a robust approach to approximating real-valued, discrete-valued, and vector-valued target functions. For certain types of problems, such as learning to interpret complex real-world sensor data, artificial neural networks are among the most effective learning methods currently known.

ANNs are typically made of individual units which are interconnected in layers that form a directed cyclic or acyclic graph. they also can be directed or undirected graphs (see Figure 2.1). The most common and practical ANN approaches are based on the Backpropagation algorithm. The Backpropagation algorithm assumes the network is a fixed structure that corresponds to a directed graph, possibly containing cycles. Learning corresponds to choosing a weight value for each edge in the graph. Although certain types of cycles are allowed, the vast majority of practical applications involve acyclic feed-forward networks[19].

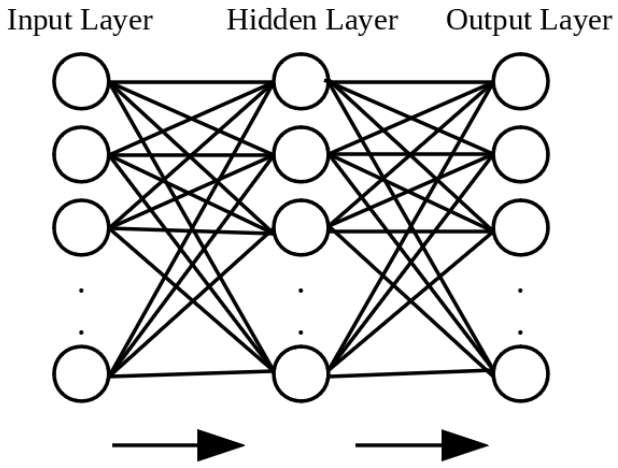



**Figure 2.1:** A two layer feed-forward neural network.

Artificial Neural Networks learning is well-suited to problems in which the training data corresponds to noisy, complex sensor data, such as inputs from cameras and microphones.

### 2.5.1 Inspiration

To simulate humans ability to learn, and make a computer program that can improve with experience, scientists have looked to biology for inspiration. A human brain is made of a large interconnected networks of neurons to process information. Electrical inputs are passed through this network of neurons which result in an output being produced. A neuron (see Figure 2.2) collects inputs using a structure called dendrites, then it effectively sums all of these inputs from the dendrites and if the resulting value is greater than its firing threshold, the neuron fires. When a neuron fires it sends an electrical impulse through its axon to its synaptic terminals. These terminals can then be networked to thousands of other neurons via connections called synapses. There are about one hundred billion ($10^{11}$) neurons inside the human brain each with about one thousand synaptic connections. The Artificial neuron model is a simplified version of a biological neuron[20].

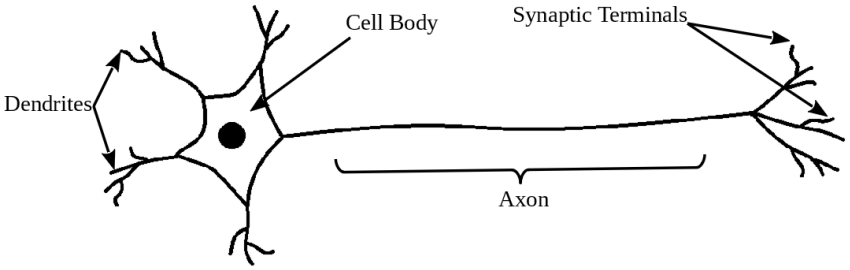



**Figure 2.2:** The schematic of a single biological neuron.

### 2.5.2 Perceptrons

One type of ANN system is based on a unit called a perceptron. A perceptron takes a vector of real-valued inputs, calculates a linear combination of these inputs, then outputs a 1 if the result is greater than some threshold and -1 otherwise (see Figure 2.3). Mathematically, given inputs $x_1$ through $x_n$ the

output $o(x_1, ..., x_n)$ is computed by the perceptron $i$ as follows[19]:

$$o = \sum_{i=0}^{n} w_i x_i \tag{2.1}$$

where each $w_i$ is a real-valued constant, or *weight*, that determines the contribution of input $x_i$ to the perceptron output.

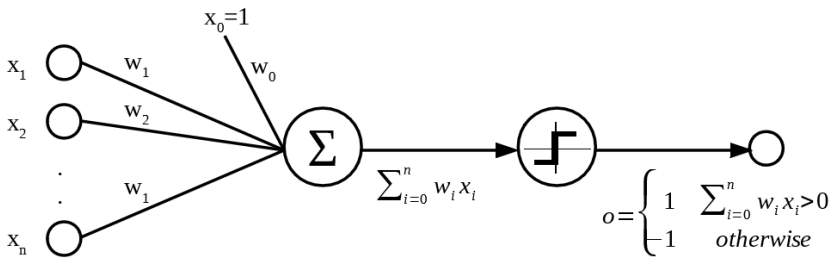



**Figure 2.3:** A perceptron.

Learning a weight vector is done as follows, first we begin with random weights, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly. Weights are modified at each step according to the *perceptron training rule*, which revises the weight $w_i$ associated with input $x_i$ according to the rule:

$$w_i \leftarrow w_i + \Delta w_i \tag{2.2}$$

where

$$\Delta w_i = n(t - o)x_i$$

$t$ is the target output for the current training example, $o$ is the output generated by the perceptron, and $n$ is a positive constant called the *learning rate* which its role is to moderate the degree to which weights are changed at each step.

### 2.5.3 Gradient Descent

Although The *perceptron rule* finds a successful weight vector when the training examples are linearly separable, it can fail to converge if the examples are not linearly separable (see Figure 2.4). A second training rule, called *the delta rule*, is designed to overcome this difficulty. If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept[19].

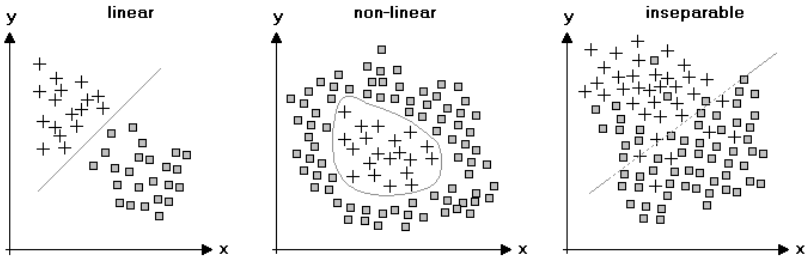



**Figure 2.4:** Training data separability.

The key idea behind the delta rule is to use *gradient descent* to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples. Considering the task of training an *unthresholded perceptron**, we need first to introduce a measure for the *training error* of a hypothesis (weight vector), relative to the training examples:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{2.3}$$

where $D$ is the set of training examples, $t_d$ is the target output for training example $d$, and $o_d$ is the output of the linear unit for training example $d$. The factor $\frac{1}{2}$ is included to cancel the exponent when differentiating.

Gradient descent search determines a weight vector that minimizes $E$ by starting with an arbitrary initial weight vector, then repeatedly modifying it in small steps. At each step, the weight vector is altered in the direction that produces the steepest descent along the error surface depicted in Figure 2.5. This process continues until the global minimum error is reached.

To calculate the direction of steepest descent along the error surface, we compute the derivative of $E$ with respect to each component of the vector $\vec{w}$. This vector derivative is called the gradient of $E$ with respect to $\vec{w}$, written $\nabla E(\vec{w})$:

$$\nabla E(\vec{w}) \equiv \left[ \frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, ..., \frac{\partial E}{\partial w_n} \right] \tag{2.4}$$

Since the gradient specifies the direction of steepest increase of $E$, the training rule for gradient descent is:

$$\vec{w} = \vec{w} + \Delta \vec{w}$$

*A linear unit corresponds to the first stage of a perceptron, without the threshold.

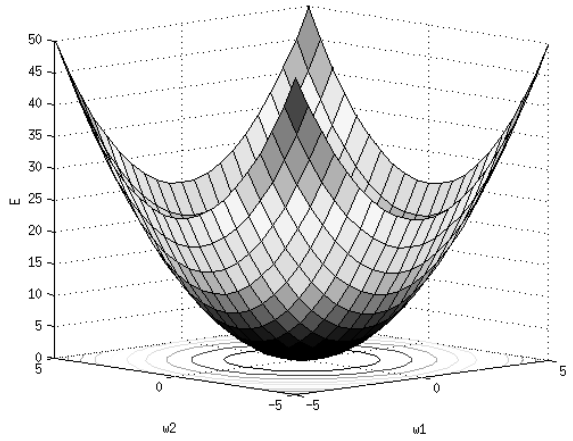



**Figure 2.5:** Error surface of a linear neuron with two input weights.

where

$$\Delta\vec{w} = -n\nabla E(\vec{w}) \tag{2.5}$$

Here $n$ is a positive constant called *the learning rate*, which determines the step size in the gradient descent search. The negative sign is present because we want to move the weight vector in the direction that decreases $E$.

The vector of $\frac{\partial E}{\partial w_i}$ derivatives that form the gradient can be obtained by differentiating $E$. After differentiating $E$ and simplifying the formula we find:

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D}(t_d - o_d)(-x_{id}) \tag{2.6}$$

where $x_{id}$ denotes the single input component $x_i$ for training example $d$. We now have an equation that gives in terms of the linear unit inputs $x_{id}$, outputs $o_d$, and target values $t_d$ associated with the training examples. Substituting Equation 2.5 into Equation 2.6 yields the weight update rule for gradient descent:

$$\Delta w_i = n\sum_{d \in D}(t_d - o_d)x_{id} \tag{2.7}$$

Gradient descent is an important general paradigm for learning. But it has some difficulties:

- Converging to a local minimum can be slow.
- If there are multiple local minima in the error surface, then there is no guarantee that the procedure will find the global minimum.

**Algorithm 4** Gradient descent

**Gradient-Descent**(*training_examples*, $n$)
Each training example is a pair of the form $\langle \vec{x}, t \rangle$, where $\vec{x}$ is the vector of input values, and $t$ is the target output value. $n$ is the learning rate.

- Initialize each $w_i$ to some small random value
- **WHILE NOT** the termination condition is met **DO**
  - Initialize each $\Delta w_i$ to zero
  - **FOR EACH** $\langle \vec{x}, t \rangle$ in *training_examples* **DO**
    - Input the instance $\vec{x}$ to the unit an compute the output $o$
    - **FOR EACH** linear unit weight $w_i$ **DO**

$$\Delta w_i \leftarrow \Delta w_i + n(t - o)x_i \tag{2.8}$$

  - **FOR EACH** linear unit weight $w_i$ **DO**

$$w_i \leftarrow w_i + \Delta w_i \tag{2.9}$$

One common variation of gradient descent that tries to reduce the effect of the above mentioned difficulties is called *stochastic gradient descent*. Whereas the gradient descent training rule computes weight updates after summing over all the training examples in $D$, the idea behind stochastic gradient descent is to approximate this gradient descent search by updating weights incrementally, following the calculation of the error for each individual example. As we iterate through each training example we update the weight according to:

$$\Delta w_i = n(t - o)x_i \tag{2.10}$$

where $t$ is the target value, $o$ unit output and $x_i$ is the $i$th input for the training example in question.

To change algorithm 4 to a stochastic descent we remove equation 2.9 and replace equation 2.8 by:

$$w_i \leftarrow w_i + n(t - o)x_i \tag{2.11}$$

### 2.5.4 Multilayer Networks and The Backpropagation Algorithm

Because single perceptrons can only express linear decision surfaces, and multiple layers of cascaded linear units still produce only linear functions, we need a different unit. In order to be able to represent non-linear functions, we choose the sigmoid unit which is based on a smoothed, differentiable threshold function (see Figure 2.6)[19]. The sigmoid unit computes its output $o$ as:

$$o = \sigma(\vec{w}.\vec{x})$$

where

$$\sigma(y) = \frac{1}{1 + e^{-y}} \tag{2.12}$$

and its derivative is:

$$\frac{d\sigma}{dy}(y) = \sigma(y)(1 - \sigma(y)) \tag{2.13}$$

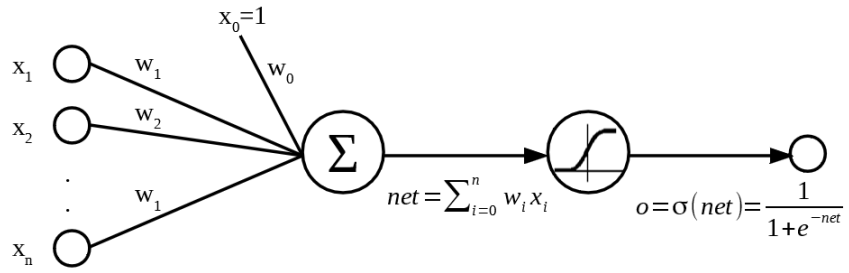



**Figure 2.6:** The sigmoid threshold unit.

Learning weights for a multilayer network is done using the **Backpropagation** algorithm. Given a network with a fixed set of units and interconnections, it employs gradient descent to attempt to minimize the squared error between the network output values and the target values for these outputs.

Stochastic gradient descent involves iterating through the training examples one at a time, for each training example $d$ descending the gradient of the error $E_d$ with respect to this single example. That means, for each training example $d$ every weight $w_{ji}$ is updated by adding to it $\Delta w_{ji}$:

$$\Delta w_{ji} = -n\frac{\partial E_d}{\partial w_{ji}} \tag{2.14}$$

where $E_d$ is the error on training example $d$, summed over all output units in the network.

$$E_d(\vec{w}) = \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

Calculating the partial derivative of the error with respect to a weight $w_{ji}$ is done using the chain rule[†]:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} \frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_{ji} \tag{2.15}$$

Now we need to calculate derivative of $\frac{\partial E_d}{\partial net_j}$ for 2 cases, output and hidden neurons:

- **Case 1: Training rule for output neuron weights**: invoking the chain rule again we can write:

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial net_j} \tag{2.16}$$

  after calculation and simplification we find:

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j) \tag{2.17}$$

  and therefore we write $\Delta w_{ji}$ as:

$$\Delta w_{ji} = -n \frac{\partial E_d}{\partial w_{ji}} = n(t_j - o_j) o_j (1 - o_j) x_{ji} \tag{2.18}$$

  From now on we denote $-\frac{\partial E_d}{\partial net_i}$ as $\delta_i$.

- **Case 2: Training rule for hidden neuron weights:**

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \tag{2.19}$$

  after calculation and simplification we find:

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} -\delta_k w_{kj} o_j (1 - o_j) \tag{2.20}$$

[†]The chain rule is a formula for computing the derivative of the composition of two or more functions.

Rearranging terms and using the $\delta_j$ notation we find:

$$\delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k w_{kj} \tag{2.21}$$

and therefore we write $\Delta w_{ji}$ as:

$$\Delta w_{ji} = n \delta_j x_{ji} \tag{2.22}$$

where:

- $x_{ji}$: the $i$th input to unit $j$.
- $w_{ji}$: the weight associated with the $i$th input to unit $j$.
- $o_j$: the output computed by unit $j$
- $t_i$: the target output for unit $j$
- $Downstream(j)$: all units whose inputs include the output of unit $j$.
- $net_j$: $\sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit $j$).

The complete steps of stochastic gradient descent Backpropagation are shown in algorithm 5.

**Algorithm 5** Stochastic gradient descent Backpropagation

**Backpropagation**(*training_examples*, $n$)
Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

- Initialize each $w_i$ to some small random value
- **WHILE NOT** the termination condition is met **DO**
  - **FOR EACH** $\langle \vec{x}, t \rangle$ in *training_examples* **DO**
    - Input the instance $\vec{x}$ to the unit and compute the output $o_u$ of every unit $u$ in the network
    - **FOR EACH** output unit $k$, calculate its error term $\delta_k$

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{2.23}$$

    - **FOR EACH** hidden unit $h$, calculate its error term $\delta_h$

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{s \in Downstream(h)} w_{sh}\delta_s \tag{2.24}$$

    - Update each network weight $w_{ji}$

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

      where

$$\Delta w_{ji} = n\delta_j x_{ji} \tag{2.25}$$

The gradient descent weight-update rule updates each weight in proportion to the learning rate $n$, the input value $x_{ji}$ to which the weight is applied, and the error in the output of the unit. The error term $\delta_j$ is computed for each network output unit $k$ by multiplying the $(t_k - o_k)$ from the delta rule, and the factor $o_k(1 - o_k)$, which is the derivative of the sigmoid function.

The $\delta_h$ for the hidden units $h$ is calculated in a similar way. However, since training examples provide target values $t_k$ only for network outputs. The error term for hidden unit $h$ is calculated by summing the error terms $\delta_k$ for each output unit influenced by $h$, weighting each of the $\delta_k$'s by $w_{kh}$, the weight from hidden unit $h$ to output unit $k$. This weight characterizes the degree to which $h$ is "responsible for" the error in output unit $k$.

In the case of a network with multiple hidden layers, the error term of the inner layers is calculated in a similar way, but instead of summing the error terms from the output layer, it uses the Downstream layer, which is the layer next to it.

One common variation to the stochastic gradient descent Backpropagation algorithm is to alter the weight-update rule in Equation 2.25 in Algorithm 5 by making the weight update on the $n$th iteration depend partially on the update that occurred during the $(n-1)$th iteration, as follows[19]:

$$\Delta w_{ji}(n) = n\delta_j x_{ji} + \alpha \Delta w_{ji}(n-1) \tag{2.26}$$

where $\alpha$ is a constant $(0 \leq \alpha < 1)$ called the *momentum*, it has the effect of gradually increasing the step size of the search in regions where the gradient is unchanging, thereby speeding convergence.

## 2.6 Conclusion

Machine Learning algorithms are deployed in many aspects of our lives, in our computers, cars, smartphones...etc, and it is still expanding. In this chapter we have took a small peek at this vast world. We have also seen Neural Networks, one of the most common Machine Learning methods. In the next chapters we use this method to build an accurate ANPR system for Algerian cars.

# Chapter 3

# Related Works

Automatic number plate recognition (ANPR) is at the core of every intelligent transportation system (ITS), it has many applications including: parking management, access control, border control, monitoring and tracking vehicles. ANPR algorithms face a number of difficulties that they have to overcome, such as environmental complexities, geometrical changes and runtime limitations. The recent advances in Machine vision, Machine Learning and the increase of computing power, have significantly helped to solve some of this issues.

In the last years, efforts have been made in developing new and innovative approaches to make ANPR systems fast, accurate and robust. In this chapter, we explore some of the most common methods and approaches that are used in todays ANPR systems.

ANPR systems are typically composed of four steps (see Figure 3.1): image acquisition, license plate (LP) extraction, LP segmentation and lastly the LP recognition.

Because the image acquisition step is not in the scope of our work, we focus only on the remaining three steps.

## 3.1 License Plate Extraction

This step takes as an input a car image, and outputs a segment from the image containing the potential LP. The LP's placement in the image is initially unknown. Instead of considering every pixel in the image, which increases the processing time, the LP can be distinguished by its features. The features are derived from the LP format and the characters constituting it. Some of the features are[21]:

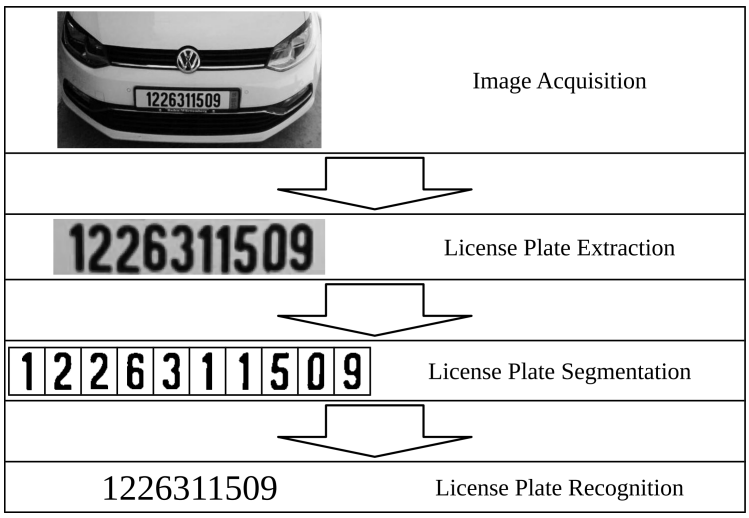



**Figure 3.1:** Steps of a typical ANPR system.

- The LP's color, since some jurisdictions (i.e., countries, states, or provinces) have certain colors for their LPs.
- The rectangular shape of the LP boundary.
- The color change between the characters and the LP's background, known as the texture.
- The existence of the characters can be used as a feature to identify the region of the license plate.

Two or more features can be combined to identify the license plate. The following LP extraction methods are categorized based on the used features.

### 3.1.1 LP Extraction using Boundry and Edge Information

The license plate normally has a rectangular shape with a known width/height ratio. It can be extracted by finding all possible rectangles in the image. Edge detection methods are commonly used to find these rectangles.

Candidate regions are extracted by D. Zheng et al. (2005) using the magnitude of the vertical edges on the LP, this feature is considered very robust. Rectangles that have the same width/height ratio as the LP are considered candidates. This method yielded a result of 96.2% on images under various illumination conditions[22].

A boundary based extraction method that uses Hough Transform (HT) was described by V. Kamat and S. Ganesan (1995). It detects straight lines in the image to locate the LP. HT has the advantage of detecting straight lines with up to 30 degrees inclination. However, it is a time and memory consuming process[23].

### 3.1.2 LP Extraction using Global Image Information

2D cross correlation was used by K. Miyamoto et al. (1991) to find LPs. It was applied with a pre-stored LP template through the entire image to locate the most likely LP area. Extracting LPs using correlation with a template is independent of its position in the image. However, this method is time consuming[24].

### 3.1.3 LP Extraction using Texture Features

This kind of methods depends on the presence of characters in the LP area, which results in significant change of the color between characters and the background. It also results in high edge density area due to color transitions.

Image transformations are widely used in LP extraction. Gabor filters are one of the major tools for texture analysis. This technique has the advantage of analyzing texture in unlimited orientations and scales. The result achieved by F. Kahraman et al. (2003) is 98% when applied to images acquired in a fixed and specific angle. However, this method is time-consuming[25].

Other texture features based methods that are used and not mentioned here include: sliding concentric windows (SCW), vector quantization (VQ), discrete Fourier transform (DFT), wavelet transform (WT) and adaptive boosting (AdaBoost).

### 3.1.4 LP Extraction using Color Features

Some countries have specific colors for their LPs. A number of proposed works involve the extraction of LP by locating its color in the image. The basic idea is that the color combination of a plate and characters is unique, and this combination occurs almost only in a plate region.

Extracting LPs using color information has the advantage of detecting inclined and deformed plates. However, it also has several difficulties.

Detecting the pixel color using the RGB color-space is very difficult, especially in different illumination conditions. The HLS (Hue, Luminosity, Saturation) color-space, which is used as an alternative, is very sensitive to noise. Methods that use color projection often confuse areas with the same color as the plate, as legitimate candidates.

S. Kim et al. (1996) used a Genetic algorithm (GA) as a search method, to identify the LP's color. It was trained using pictures with different lighting conditions, and it was able to determine the upper and lower thresholds for the plate's color. A special function describes the relation between the average brightness and these thresholds. For any input picture the average brightness is determined first, and then from this function the lower and upper thresholds are obtained. Any pixel with a value between these thresholds is labeled. If the labeled region of the connected pixels is rectangular with the same width/height ratio as an LP, the region is considered a plate[26].

### 3.1.5 LP Extraction using Character Features

One of the proposed LP extraction methods is based on locating its characters. This kind of methods examine the image for the presence of characters. If the characters are found, their region is extracted and considered as an LP. J. Matas and K. Zimmermann (2005) used a region based approach. Regions are enumerated and classified using a neural network. If a linear combination of character-like regions is found, the presence of an LP is assumed[27].

H. Hontani and T. Koga (2001) extracted the characters using scale-space analysis. The method extracts large-size blob-type figures that consist of smaller line-type figures and consider them as character candidates[28].

## 3.2 License Plate Segmentation

The segmentation step determines the accuracy of an ANPR system. It consists of dividing the extracted LP into segments, where each segment contains a character. First it has to deal with the cases where the LP might be tilted or has a nonuniform brightness. If the segmentation process fails to segment characters correctly, like merging two characters together, or dividing single ones between segments, the recognition process simply fails to recognize them.

To correct the orientation of a tilted LP, X. Xu et al. (2006) used the bilinear transformation to map the LP to a straight rectangle[29]. M. Pan et al. (2008) used a least square method to treat horizontal and vertical tilt[30].

Binarization is an important process in LP segmentation. Choosing an inappropriate threshold for the binarization of the extracted LP results in joined characters. These characters make the segmentation very difficult. LPs with a surrounding frame are also difficult to segment since after binarization, some characters may be joined with the frame. Enhancing the image quality before binarization helps in choosing the appropriate threshold. Techniques commonly used to enhance the license plate image are noise removal, histogram equalization, and contrast enhancement.

The following LP segmentation methods are categorized based on the used features.

### 3.2.1 LP Segmentation using Pixel Connectivity

Segmentation is performed by K. Miyamoto et al. (1991) and S. Chang et al. (2004) using CCL (explained in "Background" chapter). The labeled groups are analyzed, and those which have the same size and width/height ratio as a character, are considered legitimate segments. This method fails to extract all the characters when there are joined or broken characters[24, 31].

### 3.2.2 LP Segmentation using Projection

Characters and LP background have different colors, S. Zhang et al. (2004); E. Lee et al. (1994) and K. Kim et al. (2000) used this feature by projecting the binary image of the LP vertically to determine the starting and ending points of the characters, and then projecting the segmented characters horizontally to extract each character alone[32, 33, 34]. S. Zhang et al. (2004) used vertical projection along with noise removal and character sequence analysis, to extract the characters. By examining more than 30,000 images, this method reached the accuracy rate of 99.2%[32].

## 3.3 License Plate Recognition

This step tries to identify the characters in the segments. Some of the difficulties that it may face include the inconsistency of The character's size and font, those usually change from car to another. In addition to noise and deformation of characters.

The following recognition methods are categorized on the base of used features.

### 3.3.1 Character Recognition using Raw Data

Template matching is a simple and straightforward method in recognition. The similarity between a character and the templates is measured, then the template that is the most similar to the character is recognized as the target. Most template matching methods use binary images because of luminosity issues in the gray-scale ones.

K. Miyamoto et al. (1991); E. Lee et al. (1994); P. Comelli et al. (1995); S. Tang and W. Li (2005) used template matching after normalizing the sizes of characters. Several similarity measurements are used, such as the Bayes decision[24], Jaccard value[33], Hausdorff distance[35].

P. Comelli et al. (1995) used the normalized cross correlation method to match the extracted characters with the templates. Each template scans the character column by column to calculate the normalized cross correlation. The template with the maximum value is the most similar one[36].

Template matching is useful for recognizing single font, non-rotated, non-broken and fixed-size characters. If a character is different from the template due to any font change, rotation, or noise, the template matching produces incorrect recognition. T. Naito et al. (2000) dealt with this problem by storing several templates of the same character with different inclination angles[37].

### 3.3.2 Character Recognition using Extracted Features

P. Hu et al. (2002) used the Gabor filter for feature extraction. The character edges whose orientation has the same angle as the filter will have the maximum respond to the filter. This can be used to form a feature vector for each character[38].

J. Jiao et al. (2009) used the pixels' gray-scale values of 11 sub-blocks as the features, after that they were fed into a neural network classifier[39].

S. Chang et al. (2004) used topological features of characters, such as the number of holes, endpoints, three-way nodes and four-way nodes. These features are invariant to spatial transformations[31].

After the vector of features is extracted, many classifiers can be used to recognize characters, such as ANN (Artificial Neural Networks)[39] and SVM (Support Vector Machines)[34].

## 3.4 Conclusion

In this chapter we have seen some of the common and recent ANPR methods used by nowadays systems. These methods have advantages and disadvantages, for example, some of them are accurate and complex, which makes them consume resources. In the other hand there are methods which are simple and fast, but they loose the accuracy. A good ANPR platform would make the right compromise, and implement methods that balance between complexity and accuracy.

# Chapter 4

# Implementation and Test Results

So far we have seen common machine vision algorithms in the background chapter, then we proceeded with introducing the discipline of machine learning, and finally we pointed out some ANPR methods that are used in recent works. In this chapter we try to implement an ANPR system for Algerian vehicles. Following the methodology used by the related works mentioned in the previous chapter, we divided our system into three steps, license plate extraction, segmentation and recognition. In the next sections we explain in detail the algorithms and techniques we used to build the system.

## 4.1 Development Environment

We decided to build the Algerian ANPR system using the programming language Java because it has many advantages, including[40]:

- **Simplicity:** The Java programming language can be programmed without extensive programmer training while being attuned to current software practices. The fundamental concepts of Java are grasped quickly.
- **Robustness:** The Java programming language is designed for creating highly reliable software. It provides extensive compile-time checking, followed by a second level of run-time checking. Language features guide programmers towards reliable programming habits.

- **Security:** Java technology is designed to operate in distributed environments, which means that security is of paramount importance. Security features are designed into the language and run-time system.

- **Portability:** The Java platform is designed to support applications that are deployed into heterogeneous network environments. In such environments, applications must be capable of executing on a variety of hardware architectures that run different operating systems. To accommodate the diversity of operating environments, the Java Compiler generates bytecodes, an architecture neutral intermediate format designed to transport code to multiple hardware and software platforms. The architecture-neutral and portable platform of Java is known as the Java virtual machine (JVM). It's the specification of an abstract machine for which Java programming language compilers can generate code. Specific implementations of the Java virtual machine for specific hardware and software platforms then provide the concrete realization of the virtual machine.

- **Performance:** The Java platform achieves high performance by adopting a scheme by which the interpreter can run at full speed without needing to check the run-time environment. The automatic garbage collector runs as a low-priority background thread, ensuring a high probability that memory is available when required, leading to better performance.

  Another performance feature of Java is The Just-In-Time (JIT) compiler. It is a component of the Java Runtime Environment that improves the performance of Java applications at run time. It is activated when a Java method is called. The JIT compiler compiles the bytecodes of that method into native machine code, compiling it "just in time" to run. When a method has been compiled, the JVM calls the compiled code of that method directly instead of interpreting it, this results to a huge performance boost.

There exist a variety of computer vision and machine learning libraries that would have made the building process easy and fast. But instead we implemented most of the algorithms from scratch, to help us understand how they work, and to have full control over the code. In addition to making the system independent from any libraries, and therefore deployable in most of the platforms that have a Java Virtual Machine (JVM).

## 4.2 Algerian License Plate Format

Using the Algerian regulations published in the official journal[41, 42, 43], we managed to extract the following properties of Algerian license plates.

### 4.2.1 Color

Algerian LPs have a reflective white-gray background for front-plates, and yellow for rear-plates, with black Arabic digits.

### 4.2.2 Composition

The Algerian license plate registration number is composed (from right to left) of:

1. A diagram representing the Wilaya of registration (from 01 to 48).
2. A group of three Arabic digits, separated from previous by an apparent dash. The first two digits represent the release year, and the last one represents the vehicle's category (see table 4.1).

| **Vehicle category** | **Number** |
|---|---|
| Tourism vehicle | 1 |
| Truck | 2 |
| Van | 3 |
| Autocars and Autobus | 4 |
| Road tractor | 5 |
| Other tractors | 6 |
| Special vehicles | 7 |
| Trailer and semi-trailer | 8 |
| Motorcycles | 9 |

**Table 4.1:** Algerian vehicle category numbers.

3. A group of five Arabic digits separated from previous by an apparent dash, representing the chronological order number of registration in its category.

### 4.2.3 Size

Algerian vehicle license plates have a rectangular shape, where the longer side is horizontal. The dimensions of normal plates with one row are shown in table 4.2, and plates with two rows are shown in table 4.3.

| | |
|---|---|
| Width | 455 to 590 mm |
| Height | 100 to 110 mm |
| Digits height | 75 mm |
| Digits width except "1" | 135 mm |
| Width of the digit "1" | 120 mm |
| Gap between digits | 10 mm |
| Gap between digits and edge | 10 mm at minimum |

**Table 4.2:** Dimensions of Algerian one row license plates.

| | |
|---|---|
| Width | 275 mm |
| Height | 200 mm |
| The other dimensions are similar to 1-row plates. | |

**Table 4.3:** Dimensions of Algerian two row license plates.

## 4.3 Plate Extraction

The first step of an ANPR system is the detection of the license plate area. This problematic includes algorithms that are able to detect a rectangular area of the LP in a car image. An LP is described as a small plastic or metal plate attached to a vehicle for official identification purposes. In order for this description to be comprehensible by machines, we need to find an alternative one, that is based on different characteristics[44].

A license plate can be defined as a rectangular area with increased occurrences of horizontal and vertical edges. The high density of edges on such a small area is in many cases caused by the high contrast of the plate's characters, but not always. This definition can also include other objects that fit the description, which will be mistaken as a license plate. Because of this, our platform tries to find several candidates, and then chooses one of them based on a further heuristic analysis.

The extraction process passes through three main operations. First preprocessing the input image. This operation applies a number of filters to prepare it for the next one. Second the statistical analysis. This operation analyses the preprocessed image and extracts a number of candidate LP regions. Finally the heuristic analysis forms a priority ordered list of candidates based on a specified criteria. In the next sections we explain these operations in detail.

### 4.3.1 Preprocessing

This operation prepares the input image to the analysis process, it is also the heaviest in terms of execution time, because it contains CPU intensive image processing filters. It consists of:

#### Size Normalization

Our platform does not specify the hardware to be deployed in, and the imaging sensor to be used, what makes the size of input images undetermined. That size has a direct affect on the preprocessing runtime. To make that runtime constant we need to resize the input image into a new size, which should not be too small so we do not loose edge details, and not too big to make preprocessing very slow.

For the resizing algorithm we used the Java built-in *"getScaledInstance"* function from the class *"Image"*, which takes a width and a height as arguments, in addition to the resizing algorithm type, for that we used the *"SCALE_SMOOTH"* type to reduce the staircase effect.

#### Gray-scaling

We convert the input image to gray-scale, because our plate extraction approach does not rely on colors, and processing gray-scale images is easy and fast. To convert a colored image to gray-scale we used the luminosity method explained in "Background" chapter.

### Edge Extraction

Vertical and horizontal edges are extracted by convolving the input image with the Sobel kernels (explained in "Background" chapter). Because edge extraction algorithms are very sensitive to noise, we used a Gaussian blur to smooth the image and get rid of unwanted edges. A demonstration example is shown in Figure 4.1.


**Figure 4.1:** An example car image after edge extraction.

### 4.3.2 Statistical Analysis

After preprocessing filters are applied to the input image, the license plate area detection is done according to statistical analysis of that image. There are various methods that use statistics. One of them is based on horizontal and vertical projections of the image into axes $x$ and $y$.

The projection of an image according to an axis is a graph, which represents the overall magnitudes of the image on each point of the projection axis (see figure 4.2). Because The projection is calculated after applying an edge extraction filter, the magnitude of a certain point represents the number of edge occurrences. Assuming an image $I$ with a width $w$ and height $h$, the horizontal and vertical projections $P_h$, $P_v$ respectively are defined by[44]:

$$P_h(x) = \sum_{i=0}^{h-1} I(x,i) \quad ; \quad P_v(y) = \sum_{j=0}^{w-1} I(j,y)$$

The LP area extraction is accomplished in two steps, vertical detection, which extracts a band by analysis of the vertical projection, and horizontal detection, which extracts a plate from the band by a horizontal analysis of such band.

#### Vertical Detection

This step extracts a number of band candidates (three in our case) that may contain a plate. After applying a vertical edge extraction filter on the car image, the bands are detected by finding peaks in its vertical projection. Peaks are found by looking for max values in the projection. Considering a vertical projection $P_v$ of an image with height $h$, peak is calculated as follows:

$$peak = \max_{0 \le i < h} P_v(i)$$

The band's starting and ending points are calculated by finding the peak's left and right foots:

$$peak_{leftfoot} = \max_{0 \le i \le peak} \{i | P_v(i) \le c_v * P_v(peak)\}$$

$$peak_{rightfoot} = \min_{peak \le i < h} \{i | P_v(i) \le c_v * P_v(peak)\}$$

where $c_v$ is a constant used to control the peak foot. Its value is calibrated heuristically to 0.57. The pseudo-code for extracting bands is shown in algorithm 6.

**Algorithm 6** Peaks extraction

```
for n = 1 to N do
  peak = indexOf(max(P_v))
  for i = peak downto 0 do
    peak_leftfoot = i
    if P_v(i) ≤ c_v * P_v(peak) then
      break
    end if
  end for
  for i = peak to h − 1 do
    peak_rightfoot = i
    if P_v(i) ≤ c_v * P_v(peak) then
      break
    end if
  end for
  for i = peak_leftfoot to peak_rightfoot do
    P_v(i) = 0
  end for
end for
```

This algorithm extracts $N$ bands by iteratively finding a peak, saving it then overwriting its values by zero. A demonstration of vertical detection is shown in figure 4.2.

### Horizontal Detection

The input to this step is a band candidate, and the output is a number of plate candidates (3 in our case). For this step we apply the same method used in vertical detection, but this time, the band's horizontal projection is calculated after extracting horizontal edges (not the whole image, just the band's image). The constant $c$ is calibrated to $c_h = 0.53$.

A demonstration is shown in figure 4.3.

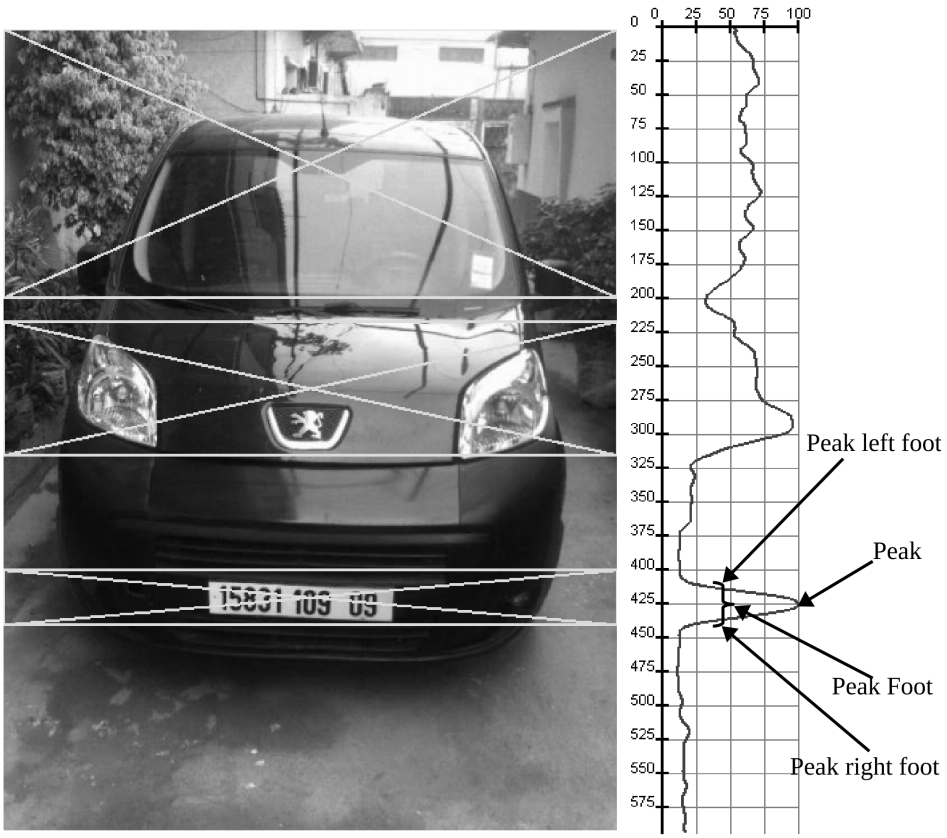



**Figure 4.2:** An example car image with its vertical projection.

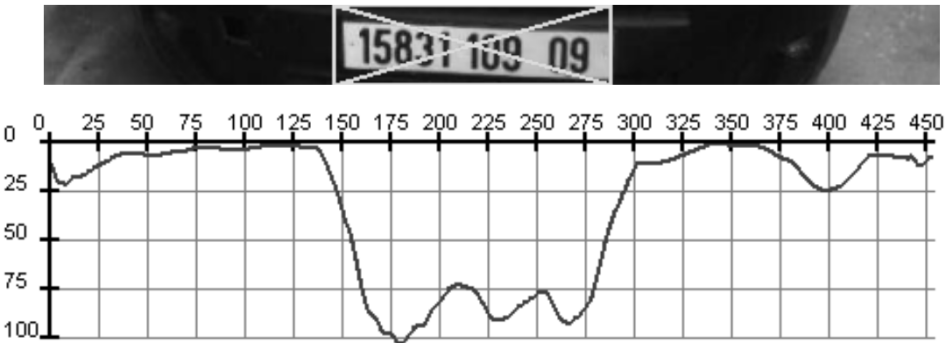


**Figure 4.3:** A band image with its horizontal projection.

### 4.3.3 Skew Correction

The capturing mechanism changes from application to another. The position and distance of the imaging sensor to the car determines whether the plate is straight, rotated or sheared. This phenomenon is caused by the physical fact that the plate is projected from the three-dimensional space into the two-dimensional space.

Sheared and rotated plates (see Figure 4.4) are extremely difficult to extract. In order to alleviate this issue we need to implement a mechanism that detects and corrects the LP's skew.

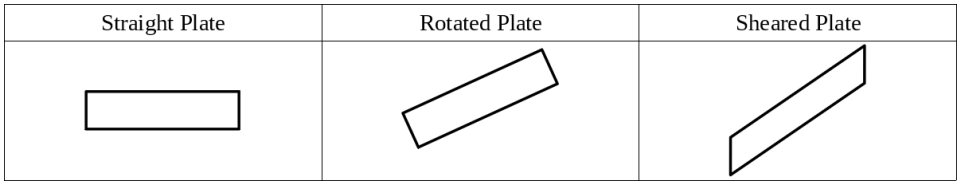



**Figure 4.4:** Possible geometrical states of a plate.

#### Detection

One way to detect the LP's orientation angle is to use the Hough transform (see "Background" chapter). It is used to find the top and bottom borders of the plate. Before using Hough transform to detect lines, the borders (i.e. edges) have to be emphasized by an edge extraction filter, in our case we used the Sobel operator[44].

The plate contains many lines other than top and bottom borders, but they usually are the thickest lines. A mechanism that selects only one angle that corresponds to a thick border line is needed.

We used a voting mechanism in which all detected lines vote for their angles, and the winning angle is chosen to be the LP's orientation. This method has shown to be effective because the Sobel filter preserves the thickness of edges, and the Hough transform will find many lines inside these edges, which will eventually vote for the same angle.

### Correction

In most of the cases, plates are sheared vertically rather than rotated. To decide what kind of transform is needed to fix the plate's skew, a threshold angle $\phi$ is chosen heuristically ($\phi = 10$ in our case). Assuming the detected skew angle is $\theta$, the type of transform is decided as follows:

$$\begin{cases} \text{if} & \theta > \phi & \text{Rotation} \\ \text{if} & \theta \leq \phi & \text{Shearing} \end{cases}$$

Suppose that we are transforming an image $I$ with width $w$ and height $h$ into a new image $I'$ with width $w'$ and height $h'$ by an angle $\theta$, we define:

- **Rotation:** considering $I$'s top right corner $c1$, bottom right corner $c2$ and bottom left corner $c3$. Their new positions after rotation $c1'$, $c2'$ and $c3'$ respectively are calculated as follows:

$$\begin{cases} c'_x = & c_x \cos(\theta) - c_y \sin(\theta) \\ c'_y = & c_y \cos(\theta) + c_x \sin(\theta) \end{cases}$$

  Using the new corner coordinates of $I'$ we can calculate its width and height:

$$\begin{aligned} w' = & \quad x_{max} - x_{min} \\ h' = & \quad y_{max} - y_{min} \end{aligned}$$

  where

$$\begin{cases} x_{min} = & \min(0, c1'_x, c2'_x, c3'_x) \\ x_{max} = & \max(c1'_x, c2'_x, c3'_x) \\ y_{min} = & \min(0, c1'_y, c2'_y, c3'_y) \\ y_{max} = & \max(c1'_y, c2'_y, c3'_y) \end{cases}$$

  The pseudo-code for rotation is shown in algorithm 7.

**Algorithm 7** Rotation

```
for x = x_min to x_max − 1 do
  for y = y_min to y_max − 1 do
    x_src = round(x cos(θ) + y sin(θ)))
    y_src = round(y cos(θ) − x sin(θ))
    I'(x, y) = I(x_src, y_src)
  end for
end for
```

- **Vertical Shearing:** first we need to calculate the shearing factor $m$:

$$m = \tan(\theta)$$

The width and height of $I'$ are calculated as follows:

$$\begin{aligned} w' &= w \\ h' &= h + \Delta h \end{aligned}$$

where

$$\Delta h = ceil(abs(m) * w)$$

The pseudo-code for shearing is shown in algorithm 8.

**Algorithm 8** Vertical shearing

```
for each pixel P at position (x, y) in I do
  y' = y + x * m
  if m < 0 then
    y' = y' + Δh
  end if
  if 0 ≤ y' < h' then
    I'(x, y') = I(x, y)
  end if
end for
```

A demonstration example is shown in figure 4.5.

| | Rotated | Sheared |
|---|---|---|
| Original plate |  |  |
| Transformed plate |  |  |

**Figure 4.5:** Examples showing plates before and after skew correction.

### 4.3.4 Exact Clipping

The result of the previous steps is a wide area that contains the potential license plate. In this step we try to detect a smaller area, which contains only the plate's digits.

After correcting the plate's skew, a known issue that often arises is the plate's border being segmented and recognized as digits. An approach we followed to solve this issue is by analyzing the Connected Components (see "Background" chapter) in the plate, then by a simple heuristic analysis we remove pieces that might be part of the plate's border (see Figure 4.6).

The piece's size was used as heuristic to remove ones that are close in size to the plate's. This method has definitely alleviated the issue, however, it has a drawback, which is removing numbers that are touching the border due to considering them part of it. This happens only in cases of deformed or steeply inclined plates.

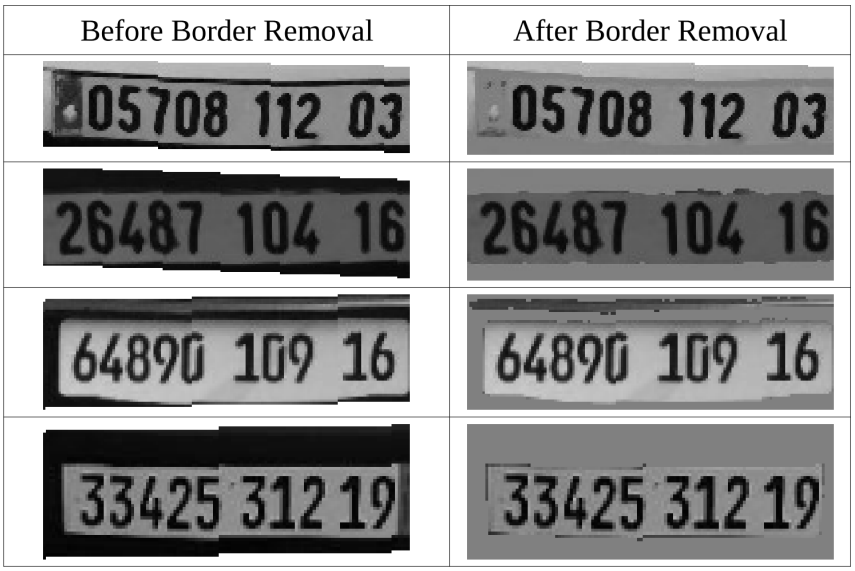



**Figure 4.6:** Examples showing plates before and after border removal.

Another affect that happens after the skew correction is the change in the plate's height. It is caused by the transform (rotation or shearing) used by skew correction. To get rid of the additional height and obtain the exact area of the plate, we used vertical detection again but this time we set the constant $c_v$ to 0.46.

### 4.3.5 Heuristic Analysis

The resulting list of plate candidates is processed by a two level heuristic analysis. The first level calculates a cost based on the plate's properties for each candidate, then the list is sorted according to that cost. The second level processes the sorted list top to bottom (the high priority first) by a deeper analysis which is time consuming. The second analysis consists of segmenting the plate into individual digits, which allows it to make a definitive decision, either to accept the candidate, or to reject it and move to the next one.

#### Priority Based Heuristic Analysis

A number of properties are considered in the calculation of a plate's cost, those properties are decided heuristically. Together they recommend the candidates that are likely to be plates, and defer the ones that are unlikely to be legitimate plates. This differentiation optimizes the runtime of the platform, where only candidates with a high likelihood are processed by the more time consuming deeper analysis[44].

The properties include:

- **Height ($H$):** the plates with a lower height are preferred.
- **Width/Height Ratio ($R$):** typical Algerian LPs have a width/height ratio of 5, the closer the value of this property to 5 the more the candidate is preferred.
- **Band's Peak ($P$):** plates extracted from bands with higher peak value are preferred.
- **Band's Peak Foot Surface ($S$):** plates extracted from bands with higher peak foot surface ($\sum_{i=peak_{leftfoot}}^{peak_{righfoot}} P_y(i)$) are preferred.

The candidate's cost equals to the weighted sum of its properties. The weights are added to control the contribution magnitude of the property to the total cost. Candidates with lower costs are preferred.

$$cost = 0.03 * H + 0.25\frac{1}{P} + 0.4\frac{1}{S} + 0.4 * abs(R - 5)$$

**Final Heuristic Analysis**

The final heuristic analysis determines if a candidate is a valid plate or not. The candidates must be segmented into individual digits. Assuming $N$ is the number of segments and $w_i$ is the width of segment $i$, a candidate is considered valid if it satisfies two conditions:

- Typical Algerian LPs contain a fixed 5 digits part, in addition to a variable size identification part that may contain up to 6 digits. After taking account the extra pieces (dirt, logos, border pieces...etc) that might exist in the plate, this feature is used to reject candidates that do not satisfy the following condition:

$$6 \leq N \leq 15$$

- LPs usually have digit segments with uniform widths. The standard deviation $S$ of their widths $w_i$ can be used as a heuristic. If $S$ is less than a heuristically determined threshold, this condition is satisfied.

$$S = \sqrt{\frac{1}{N}\sum_{i=0}^{N-1}(w_i - \bar{w})^2}$$

where $\bar{w}$ is the average of segments widths $\bar{w} = \frac{1}{N}\sum_{i=0}^{N-1} w_i$ .

## 4.4 Plate Segmentation

In order for the plate to be recognized, it needs to be segmented into individual digits. This step deals with the problematic of dividing the LP into segments that contain digits. Some of the difficulties that face this step include dividing a single digit between two segments, or merging two digits into one segment. The errors that happen in this step significantly effect the recognition accuracy.

Considering only plates with one row of numbers, which are most common in Algeria, the approach we followed to segment them is by finding the empty spaces between digits, which can be detected by analyzing the horizontal projection of the plate. This method is robust and fast, but it fails to segment overlapped digits, which are not very common in standard plates.

### 4.4.1 Preprocessing

Before analyzing the plate's horizontal projection, it needs to be thresholded (Binarized). As we mentioned in "Background" chapter, there exist two thresholding methods, local (Adaptive) and global. In this case plates usually contain noise and shadows, what makes the local thresholding best suitable for this situation (see Figure 4.7). A blurring filter can be applied before thresholding to remove noise.

| | |
|---|---|
| Original plate |  |
| Global thresholding |  |
| Adaptive thresholding |  |

**Figure 4.7:** A plate example after global and local thresholding.

### 4.4.2 Segmentation

After thresholding, the plate's background is white and its digits are black. By detecting peaks (corresponding to white spaces) in the plate's horizontal projection we can find digits boundaries (see Figure 4.8).

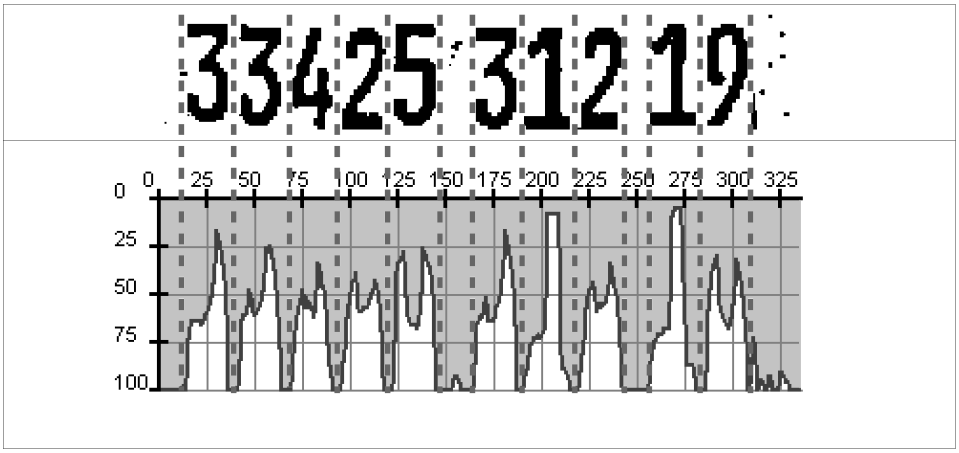



**Figure 4.8:** A binary plate image with its horizontal projection.

The segmentation algorithm iteratively finds peaks in the horizontal projection $P_h$. They are considered spaces if they satisfy a condition that takes into account their magnitude. This process is repeated until no spaces are found. The steps are shown in algorithm 9.

**Algorithm 9** Plate segmentation

Let $M$ be the maximum value of $P_h$: $M = \max\{P_h(i)\}$

1. Find peaks's index: $peak = indexOf(\max_{0 \leq i < w}\{P_h(i)\})$
2. Find left and right peak foots:

$$peak_{leftfoot} = \max_{0 \leq i < peak}\{i | P_h(i) \leq C_h * P_h(peak)\}$$

$$peak_{rightfoot} = \min_{peak \leq i < w}\{i | P_h(i) \leq C_h * P_h(peak)\}$$

3. Overwrite *peak*'s values with zeros.
4. If *peak* is not corresponding to a space: i.e. $P_h(peak) < C_k * M$, go to 7.
5. Divide plate horizontally at point *peak*.
6. Go to 1.
7. End.

Where $C_h$ is a constant that controls the peak foots, and $C_k$ is a constant that determines if a peak is considered a space. They are calibrated heuristically to 0.7 and 0.87 respectively. If $C_k$'s value is too big digits will be merged together, and if its value is too small digits are divided between segments.

### 4.4.3 Digit Piece Extraction

A digit segment may contain pieces other than the digit itself, like a border, screw or a dirt piece. Those pieces have to be removed before the recognition step. We analyzed the segments using Connected Component Labeling (see "Background" chapter), and then by a simple heuristic analysis we removed non digit pieces (see Figure 4.9). Each piece $p$ is enclosed in a rectangle which

is defined by two points:

$$x_0 = min\{x|(x,y) \in p\} \quad ; \quad y_0 = min\{y|(x,y) \in p\}$$

$$x_1 = max\{x|(x,y) \in p\} \quad ; \quad y_1 = max\{y|(x,y) \in p\}$$

A cost based on the following two properties is calculated for each piece:

- **Piece's surface:** A digit's piece is usually the biggest in the segment, pieces with bigger surfaces are preferred.

$$surface = (x_1 - x_0) * (y_1 - y_0)$$

- **Number of white pixels:** Digits in most fonts have shapes that contain closed loops, curves and corners. This causes them to contain white areas. We used this property to differentiate them from other pieces.

$$whitePixelsCount = surafe - ||p||$$

The cost is calculated using a weighted sum of the above mentioned properties:

$$cost = 0.6 * whitePixelsCount + 0.8 * surface$$

The highest cost piece is selected.

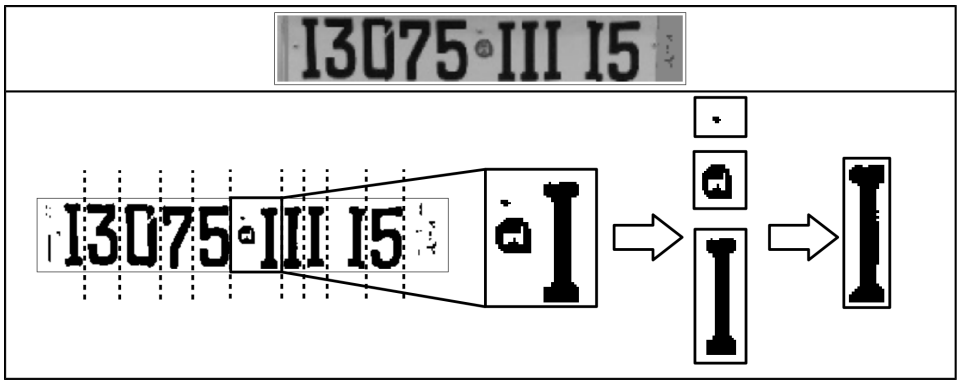


**Figure 4.9:** An example showing the extraction of a digit piece.

### 4.4.4 Segments Heuristic Analysis

License plates sometimes contain symbols on the left, right or center, in addition to imperfect clipping resulting to pieces from the plate's border. Those non-digit segments must be eliminated to not get confused as digits by the recognizer. One way to detect them is by analyzing a number of selected segment properties, and based on these properties decide whether that segment represents a digit.

We used two kinds of properties that are based on size and color of the segment. Size properties include height ($P_{height}$), width ($P_{width}$) and width/height ratio ($P_{width/height}$). Color properties include hue ($P_{hue}$), saturation ($P_{saturation}$), brightness ($P_{brightness}$) and contrast ($P_{contrast}$). A color property $P_{prop}$ of segment $i$ with width $w$ and height $h$ is calculated by averaging all pixels values[44]:

$$P^i_{prop} = \frac{1}{w * h} \sum_{i=0}^{w} \sum_{j=0}^{h} prop(i, j)$$

Considering a plate with $n$ segments, the average of a property over all segments $A_{prop}$ is calculated as follows:

$$A_{prop} = \frac{1}{n} \sum_{i=0}^{n} P^i_{prop}$$

The property difference from average $D_{prop}$ is calculate as follows:

$$D_{prop} = \frac{|P^i_{prop} - A_{prop}|}{A_{prop}}$$

In table 4.4 we show the conditions which a segment must completely fulfill to be accepted as a digit segment.

| Type | Property | Condition |
|---|---|---|
| Color | Hue | $D_{hue} < 2$ |
| | Saturation | $D_{saturation} < 2$ |
| | Brightness | $D_{brightness} < 0.37$ |
| | Contrast | $D_{contrast} < 0.8$ |
| Size | Height | $D_{height} < 0.37$ |
| | Width | $D_{width} < 0.75$ |
| | Width/Height Ratio | $0.1 < \frac{Width}{Height} < 0.92$ |

**Table 4.4:** Segments heuristic analysis conditions.

## 4.5 Plate Recognition

In machine learning, recognizing objects in images (digits in our case) is considered a classification problem. Because the input data (images) comes from a sensor (camera), and may contain noise, the most suitable learning method in this case is the Artificial Neural Network (ANN)[19].

In this step we decided to use a feed-forward neural network (see "Machine Learning" chapter) to recognize digits. In the next sections we explain the choices that were made in designing and training the network.

### 4.5.1 Input Encoding

Since the ANN input is to be some representation of the digit image, one way to encode it is by extracting a number of features from it, and inputting them into the network, but instead of that, we fed the complete digit image into the network. The input binary image of the digit is first resized to $10 \times 16$ pixels, that makes the input layer of the network have 160 input neurons, where each pixel corresponds to a neuron.

This choice is robust and very simple to implement, but it has its downsides, for example the network will not be able to recognize inclined digits. This issue is solved by Skew correction in the Extraction step. It also cannot recognize digits with new fonts that it was not trained for. This is not a real issue in our case because standard license plates have a unified font. In case of different fonts, the issue can be solved by extending the training data set to include these new fonts.

Although choosing digit features as an input to the network has many advantages, it is an unnecessary complexity, because our recognizer needs only to recognize ten types of images, which correspond to ten digits (0-9).

### 4.5.2 Output Encoding

The Neural Network must output a number that corresponds to the digit in the input image. We could encode this ten-way (0-9) classification with a single output neuron, assigning outputs of, 0, 0.1, 0.2...0.9 to each number. But instead we used ten distinct output neurons, each representing one of the ten possible numbers, with the highest-valued output taken as the network prediction. This is called the *1-of-n* output encoding and it has two advantages[19]:

- It provides more degrees of freedom to the network for representing the target function, there are $n$ times as many weights available in the output layer.
- The difference between the highest-valued output and the second-highest can be used as a measure of the confidence in the network prediction.

The target values are represented by a vector, in which the index of a position with value 1 represents the output number. Other values are set to 0. For example, 0 is represented by $\langle 1, 0, 0, 0, 0, 0, 0, 0, 0, 0\rangle$, and 1 is represented by $\langle 0, 1, 0, 0, 0, 0, 0, 0, 0, 0\rangle$ and so on.

### 4.5.3 Network Structure and Training

The most common network structure is a layered network with feed-forward connections from every unit in one layer to every unit in the next. We used this standard structure with two layers of sigmoid units (one hidden layer and one output layer). Because there is no specific rule to how many neurons are needed in the hidden layer, we started by testing some numbers. We found that under 50 neurons, the training never converges to a network with an acceptable error over the training set. By increasing the number of neurons, we arrived at 250 which resulted to a minimum acceptable error. The training was done using 1296 digit images, and it took about 15 minutes on a dual-core Intel i5 clocked at 2.5 GHz. In those training experiments the learning rate $n$ was set to 0.1, and the momentum $\alpha$ was set to 0.7.

The final network structure is shown in Figure 4.10.

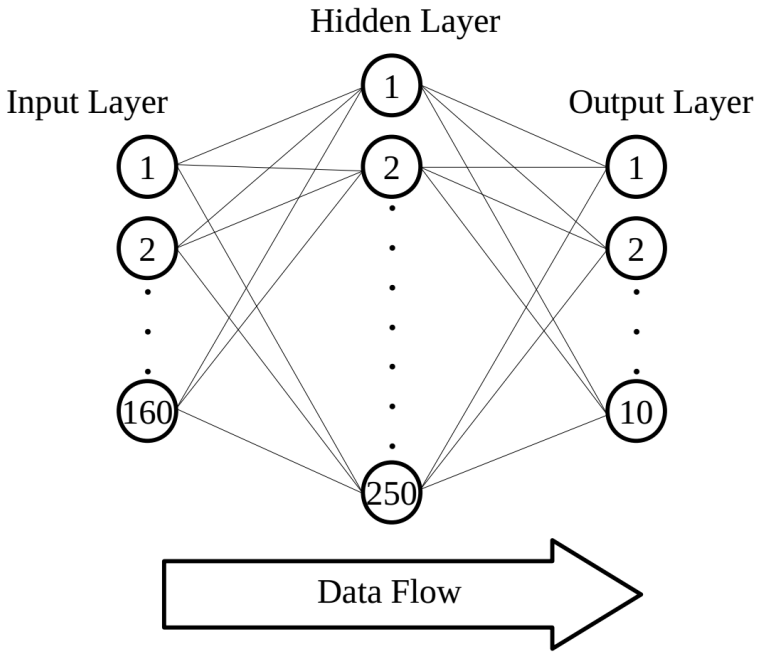



**Figure 4.10:** The final structure of the neural network.

## 4.6 Code Structure

The code is mainly composed of two packages. The "*imageprocessing*" package which contains most of the image processing algorithms and filters. Its classes consist of: *Photo, Car, Band, Plate* and *Segment*. The "*machinelearning*" package which contains the recognition algorithms. Its classes include *DataGenerator* and *NeuralNetwork*. The class diagrams of these packages are provided in appendix A.

## 4.7 Results and Discussion

We tested the Algerian ANPR system implemented in the previous sections, and here we present the results. We divided the tests into three parts, plate extraction, segmentation and recognition. This division is justified, because the error rate from a step may affect rates of the following steps.

### 4.7.1 Testing Data

We used two data sets. The first one for testing the plate extraction and segmentation. It contained 392 car images. The second data set (different than the training data set) was used for testing the recognition. It contained 200 car images, and we extracted from it 1137 binary digit images. The images in these data sets were captured by different people, using various types of cameras, in many kinds of environments and illumination conditions. They were obtained from the Algerian website "Ouedkniss".

### 4.7.2 Plate Extraction Results

We consider a successful extraction, if the system extracts the exact area of the plate with tolerating extra pieces to the plate's left and right, because they are handled later by a heuristic analysis. The testing data set was divided to two parts. First part contained images of cars with inclined plates, and the other part contained images of cars with straight plates. The results are shown in table 4.5.

| | Number of Images | Number of Errors | Success Rate (%) |
|---|---|---|---|
| Inclined Plates | 201 | 14 | 93.03 |
| Straight Plates | 191 | 8 | 95.81 |
| Total | 392 | 22 | 94.39 |

**Table 4.5:** Plate extraction results.

Most of the extraction errors happened in car images with complex backgrounds, this is due to the increase of edges that made detecting the plate very difficult. It is very important to note that the used data set is not uniform. It contained images that are captured from different angles, with mostly low resolutions because of compression, which significantly effects the extraction's accuracy.

### 4.7.3 Plate Segmentation Results

Using the same testing data set used for evaluating the extraction step, excluding only images where the extraction failed to find a correct plate, we evaluated the segmentation step. We consider a successful segmentation, if no digits were split between segments, or were merged into one segment. The results are shown in table 4.6.

| | Number of Images | Number of Errors | Success Rate (%) |
|---|---|---|---|
| Inclined Plates | 125 | 38 | 69.6 |
| Straight Plates | 219 | 38 | 82.65 |
| Total | 344 | 76 | 77.91 |

**Table 4.6:** Plate segmentation results.

The segmentation often fails to correctly segment an inclined plate. Even though its skew was corrected, the digits remain skewed. The issue of splitting and merging digits only happens in the case of plates with low contrast. The most split digit is zero, because it contains a large space inside, which is often interpreted as an inter-digit space.

### 4.7.4 Digits Recognition Results

The recognition step was tested by more than a thousand digit images. Because we extracted them from a data set with real Algerian cars, we did not control the distribution of digits. Some of the common numbers like one and zero have bigger frequency than others. The recognition results are shown in table 4.7.

| | Number of Images | Number of Errors | Success Rate (%) |
|---|---|---|---|
| Digit "0" | 150 | 1 | 99.33 |
| Digit "1" | 374 | 9 | 97.59 |
| Digit "2" | 102 | 3 | 97.06 |
| Digit "3" | 102 | 2 | 98.04 |
| Digit "4" | 70 | 2 | 97.14 |
| Digit "5" | 73 | 6 | 91.78 |
| Digit "6" | 92 | 5 | 94.57 |
| Digit "7" | 48 | 0 | 100 |
| Digit "8" | 51 | 5 | 90.2 |
| Digit "9" | 75 | 5 | 93.33 |
| Total | 1137 | 38 | 96.66 |

**Table 4.7:** Digits recognition results.

From the results shown above, we can see that the numbers five and eight are among the hardest to recognize, with the lowest success rates. This issue can be dealt with by increasing their frequency in the training data set. Some of the situations that may cause the recognizer to fail include, tilt or deformation of digits, in addition to font variation. In general, the recognition has a high success rate, and it can even be increased by using a bigger training data set.

### 4.7.5 Evaluating the Complete System

We were not able to estimate the success rate of the complete Algerian ANPR system, because of issues concerning the segments heuristic analysis. We did not have enough data and time to calibrate the analysis to reject non digit segments, causing the recognizer to mistake them as numbers, what resulted to wrong readings. This issue does not happen in case of high resolution images with clear license plates.

## 4.8 Conclusion

In this chapter we explained the algorithms and methods we used to implement the Algerian ANPR system. During the implementation process we focused only on functionality because of the limited time period.

We also presented the results of testing the implemented system, where the extraction and recognition steps has shown high success rates of about 95%, but the segmentation step has shown a success rate lower than 80%. In general we can say that the results were acceptable.

# Conclusion

Although many implementations of ANPR systems exist, the aim of this work was to build one that is specific to Algerian vehicles. Our system was implemented in three main steps. The first step extracts candidate plates from the car's image using edge information. This was done by statistically analyzing the vertical and horizontal projections of the car's image, to locate the plate vertically and horizontally. After fixing the tilted candidate plates, they were sorted on base of priority.

The second step segments candidate plates into individual digits. The non plate candidates were excluded using a heuristic analysis based on Algerian plates properties. The plate's segments were also analyzed to remove non-digit ones.

The final step uses machine learning to recognize the extracted segments. A feed-forward neural network was trained on a set of real Algerian cars. By feeding segments into this network it was able to recognize digits.

Our testing results showed that the extraction and recognition steps have high success rates, where the segmentation step achieved relatively low success rates.

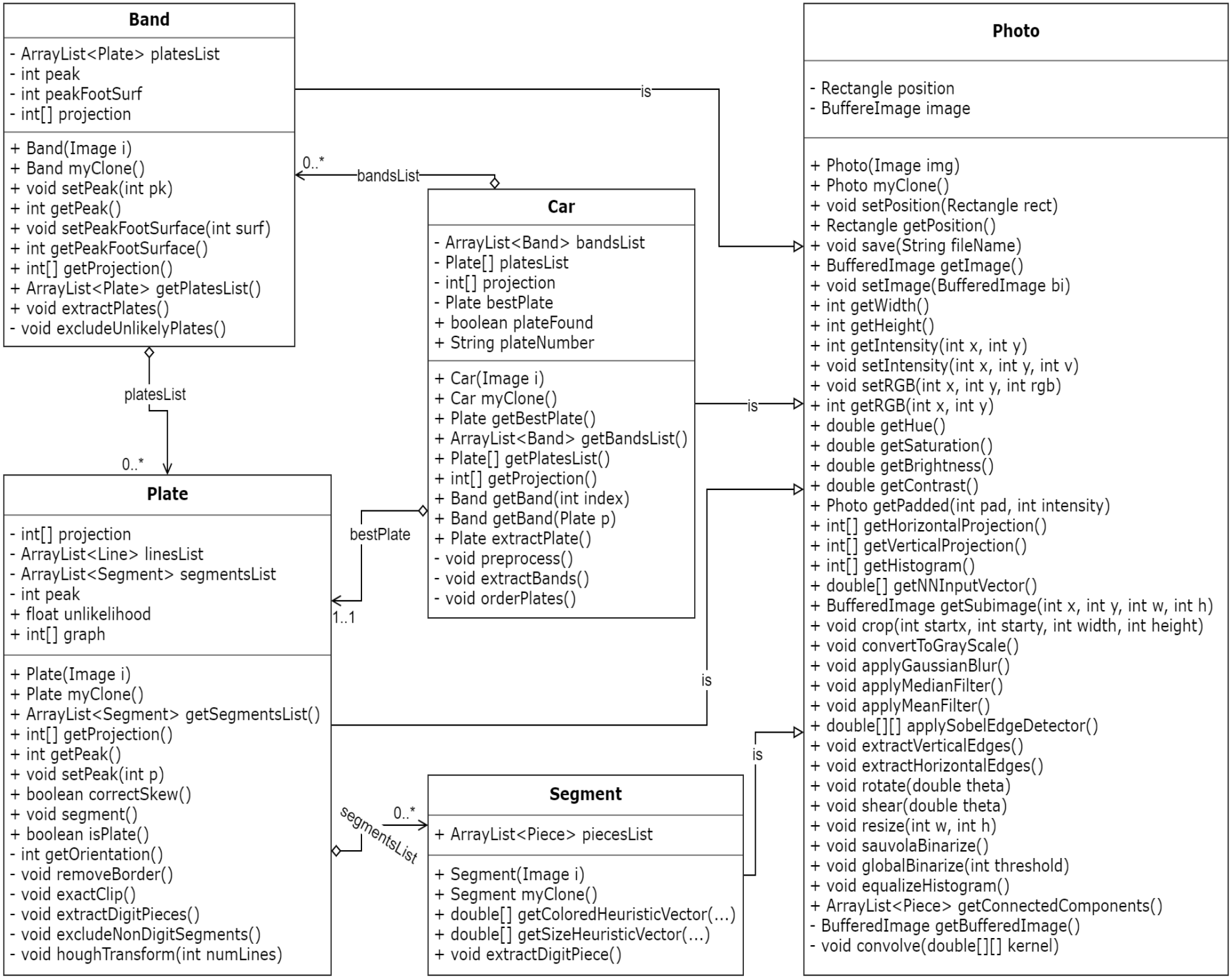
The implemented ANPR system was able to recognize correctly plates in high resolution clear images, but it failed to read other unclear images. This is mostly due to the segments heuristic analysis, which sometimes failed to exclude non-digit segments. We consider it the weakest part in our system.

Our system can be improved in many ways. Optimizing it to be suitable for real-time application is one way. Another way would be increasing the size of the training data set to improve the recognition accuracy. The segmentation step also can be improved to form a robust complete package.

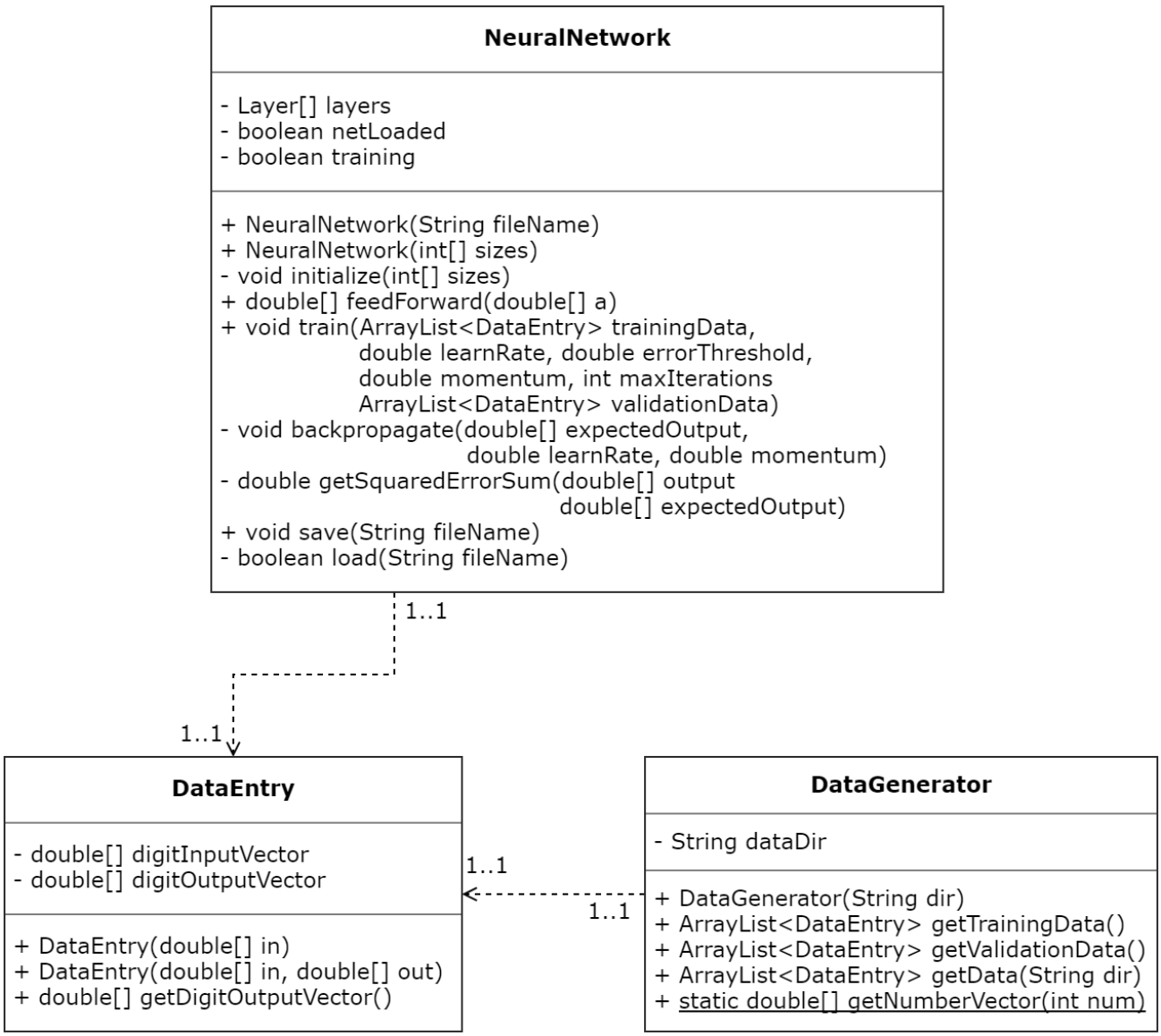
# Appendices

# Appendix A

# Class Diagrams




Class diagram of the package "*imageprocessing*".




Class diagram of the package “*machinelearning*”.

# References


[1] J. Sousanis, "World Vehicle Population Tops 1 Billion Units," *Wards Auto*, Aug. 2011.

[2] Office national des statistiques, "Situation annuelle du parc automobile." `http://www.ons.dz/-Au-31-12-2013-.html`, Dec. 2013. [Online; accessed 16-May-2016].

[3] National police chiefs' council UK, "Use of ANPR by police forces and other law enforcement agencies (LEA)." `http://www.npcc.police.uk/FreedomofInformation/ANPR.aspx`. [Online; accessed 16-May-2016].

[4] J. D. Cook, "Converting color to grayscale." `http://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/`. [Online; accessed 24-April-2016].

[5] A. Song Ho, "Histogram Equalization." `http://www.songho.ca/dsp/histogram/histogram.html`, 2006. [Online; accessed 24-April-2016].

[6] R. Fisher, S. Perkins, A. Walker, and E. Walfart, "Image Convolution." `http://homepages.inf.ed.ac.uk/rbf/HIPR2/convolve.htm`, 2003. [Online; accessed 24-April-2016].

[7] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2nd ed., 2001.

[8] R. Fisher, S. Perkins, A. Walker, and E. Walfart, "Spatial Filters - Mean Filter." `http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm`, 2003.

[9] I. Sobel, "History and definition of the sobel operator," *Presentation at Stanford A.I. Project 1968*, 2014.

[10] R. O. Duda and P. E. Hart, "Use of the hough transformation to detect lines and curves in pictures," *Commun. ACM*, vol. 15, pp. 11–15, Jan. 1972.

[11] F. O'Gorman and M. B. Clowes, "Finding picture edges through collinearity of feature points.," *IEEE Trans. Computers*, vol. 25, no. 4, pp. 449–456, 1976.

[12] N. Otsu, "A Threshold Selection Method from Gray-level Histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[13] W. Niblack, *An Introduction to Digital Image Processing*, pp. 115–116. Birkeroed, Denmark: Strandberg Publishing Company, 1985.

[14] J. Sauvola, T. Seppanen, S. Haapakoski, and M. Pietikainen, "Adaptive document binarization," in *Proceedings of the Fourth International Conference on Document Analysis and Recognition, 1997*, vol. 1, pp. 147–152, Aug 1997.

[15] F. Shafait, D. Keysers, and T. Breuel, "Efficient implementation of local adaptive thresholding techniques using integral images," vol. 6815, SPIE, Jan. 2008.

[16] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognition*, vol. 33, pp. 225–236, 2000.

[17] L. Vincent and P. Soille, "Watersheds in digital spaces: an efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, pp. 583–598, Jun 1991.

[18] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of Machine Learning*, pp. 3–8. The MIT Press, 2012.

[19] T. Mitchell, "Machine Learning," McGraw-Hill International Editions, pp. 81–127, McGraw-Hill, 1 ed., 1997.

[20] P. S. Churchland and T. J. Sejnowski, *The computational brain.* Computational neuroscience series, Cambridge ( Mass.): MIT Press London, 1992. Nouveau tirage: 1999.

[21] S. Du, M. Ibrahim, M. Shehata, and W. Badawy, "Automatic License Plate Recognition (ALPR): A State-of-the-Art Review," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 23, pp. 311–325, Feb. 2013.

[22] D. Zheng, Y. Zhao, and J. Wang, "An efficient method of license plate location," *Pattern Recognition Letters*, vol. 26, no. 15, pp. 2431 – 2438, 2005.

[23] V. Kamat and S. Ganesan, "An efficient implementation of the Hough transform for detecting vehicle license plates using DSP'S," in *Real-Time Technology and Applications Symposium, 1995. Proceedings*, pp. 58–59, May 1995.

[24] K. Miyamoto, K. Nagano, M. Tamagawa, I. Fujita, and M. Yamamoto, "Vehicle license-plate recognition by image analysis," in *International Conference on Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON*, pp. 1734–1738, Oct. 1991.

[25] F. Kahraman, B. Kurt, and M. Gökmen, "License Plate Character Segmentation Based on the Gabor Transform and Vector Quantization," in *License Plate Character Segmentation Based on the Gabor Transform and Vector Quantization*, pp. 381–388, 2003.

[26] S. K. Kim, D. W. Kim, and H. J. Kim, "A recognition of vehicle license plate using a genetic algorithm based segmentation," in *, International Conference on Image Processing, 1996. Proceedings*, vol. 1, pp. 661–664, Sept. 1996.

[27] J. Matas and K. Zimmermann, "Unconstrained licence plate and text localization and recognition," in *IEEE Intelligent Transportation Systems, Proceedings 2005.*, pp. 225–230, Sept. 2005.

[28] H. Hontani and T. Koga, "Character extraction method without prior knowledge on size and position information," in *Vehicle Electronics Conference, 2001. IVEC 2001. Proceedings of the IEEE International*, pp. 67–72, 2001.

[29] X. Xu, Z. Wang, Y. Zhang, and Y. Liang, "A method of multi-view vehicle license plates location based on rectangle features," in *2006 8th international Conference on Signal Processing*, vol. 3, 2006.

[30] M.-S. Pan, J.-B. Yan, and Z.-H. Xiao, "Vehicle license plate character segmentation," *International Journal of Automation and Computing*, vol. 5, pp. 425–432, Oct. 2008.

[31] S.-L. Chang, L.-S. Chen, Y.-C. Chung, and S.-W. Chen, "Automatic license plate recognition," *IEEE Transactions on Intelligent Transportation Systems*, vol. 5, pp. 42–53, Mar. 2004.

[32] Z. Sanyuan, Z. Mingli, and Y. Xiuzi, "Car plate character extraction under complicated environment," in *2004 IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4722–4726, 2004.

[33] E. R. Lee, P. K. Kim, and H. J. Kim, "Automatic recognition of a car license plate using color image processing," in *Image Processing, 1994. Proceedings. ICIP-94., IEEE International Conference*, vol. 2, pp. 301–305, Nov. 1994.

[34] K. K. Kim, K. I. Kim, J. B. Kim, and H. J. Kim, "Learning-based approach for license plate recognition," in *Neural Networks for Signal Processing X, 2000. Proceedings of the 2000 IEEE Signal Processing Society Workshop*, vol. 2, pp. 614–623, 2000.

[35] T. Shuang-tong and L. Wen-ju, "Number and Letter Character Recognition of Vehicle License Plate Based on Edge Hausdorff Distance," in *Sixth International Conference on Parallel and Distributed Computing Applications and Technologies (PDCAT'05)*, pp. 850–852, Dec. 2005.

[36] P. Comelli, P. Ferragina, M. N. Granieri, and F. Stabile, "Optical recognition of motor vehicle license plates," *IEEE Transactions on Vehicular Technology*, vol. 44, pp. 790–799, Nov. 1995.

[37] T. Naito, T. Tsukada, K. Yamada, K. Kozuka, and S. Yamamoto, "Robust license-plate recognition method for passing vehicles under outside environment," *IEEE Transactions on Vehicular Technology*, vol. 49, pp. 2309–2319, Nov. 2000.

[38] P. Hu, Y. Zhao, Z. Yang, and J. Wang, "Recognition of gray character using gabor filters," in *Proceedings of the Fifth International Conference on Information Fusion, 2002*, vol. 1, pp. 419–424, July 2002.

[39] J. Jiao, Q. Ye, and Q. Huang, "A configurable method for multi-style license plate recognition," *Pattern Recognition*, vol. 42, pp. 358–369, Mar. 2009.

[40] Sun Microsystems Inc, "The Java Language Environment." `http://www.oracle.com/technetwork/java/intro-141325.html`, 1997. [Online; accessed 12-May-2016].

[41] Ministere d'etat chargé des transports, "Arrêté relatif à l'immatriculation et à la réimmatriculation des véhicules automobiles," *Le Journal officiel de la République algérienne démocratique et populaire*, vol. 14, pp. 786–788, June 1975.

[42] Ministere d'etat chargé des transports, "Arrêté fixant les règles administratives relatives au numéro d'immatriculation des véhicules automobiles," *Le Journal officiel de la République algérienne démocratique et populaire*, vol. 27, pp. 904–909, May 1988.

[43] Ministere d'etat chargé des transports, "Arrêté modifiant l'arrêté du 5 mai 1988 fixant les règles administratives relatives au numéro d'immatriculation des véhicules automobiles," *Le Journal officiel de la République algérienne démocratique et populaire*, vol. 32, no. 53, pp. 15–16, 1993.

[44] O. Martinsky, "Algorithmic and mathematical principales of automatic number plate recognition systems," *BRNO University of technology*, 2007.