On the use of a discriminant approach for handwritten word recognition based on bi-character models

Sophea Prum

UNIVERSITÉ DE LA ROCHELLE

ÉCOLE DOCTORALE S2IM

LABORATOIRE : L3i (Informatique, Image et Interaction)

THÈSE présentée par :

Sophea PRUM

soutenue le : **8 novembre 2013**
pour obtenir le grade de : **Docteur de l'Université de La Rochelle**
Discipline : **Informatique et Applications**

Vers une approche discriminante pour la reconnaissance de mots manuscrits en-ligne utilisant des modèles de bi-caractères

[On the Use of a Discriminant Approach for Handwritten Word Recognition Based on Bi-character Models]

JURY :

Nicole VINCENT	Professeur, Université Paris Descartes, Rapporteur
Jean-Philippe DOMENGER	Professeur, Université Bordeaux 1, Rapporteur
Salvatore-Antoine TABBONE	Professeur, Université de Lorraine, Examinateur
Jean-Yves RAMEL	Professeur, Université François Rabelais de Tours, Président
Andreas FISCHER	Docteur, Université Concordia, Examinateur
Jean-Marc OGIER	Professeur, Université La Rochelle, Directeur de thèse
Muriel VISANI	Maître de conférences, Université La Rochelle, Encadrant de thèse

Remerciements

Ce travail de thèse s'est déroulé dans de bonnes conditions grâce à la contribution de nombreuses personnes.

Je tiens tout d'abord à exprimer ma profonde gratitude à mon directeur de thèse Jean-Marc OGIER et mon encadrante scientifique Muriel VISANI pour m'avoir encadré et guidé tout au long de ce travail de thèse avec leurs très grandes qualités scientifiques et humaines. Je les remercie également pour leur disponibilité, leurs relectures minutieuses de ce manuscrit et des articles, leur énergie, leurs encouragements et leur soutien.

Je remercie sincèrement le professeur Jean-Yves RAMEL pour avoir accepté de présider mon jury de thèse. J'exprime également mes remerciements aux professeurs Nicole VINCENT et Jean-Philippe DOMENGER pour avoir examiné et rapporté mon travail de thèse. Je remercie aussi le professeur Salvatore-Antoine TABBONE pour avoir accepté d'être membre de mon jury. Je remercie particulièrement Dr. Andreas FISCHER pour son aide à travers l'échange d'idées scientifiques concernant ce travail thèse ainsi que d'avoir participé au jury.

Mes plus sincères remerciements s'adressent également à tous les membres de l'équipe du projet RecoNomad, les collègues et les amies du L3i pour leur aide, les moments partagés ensemble et qui m'ont été bénéfiques non seulement d'un côté scientifique, ainsi que du côté technique et humain. Ceci m'a permis de réaliser ce travail de recherche dans une ambiance très conviviale.

Un grand merci du fond du cœur à mes parents pour leur soutien et encouragements tout au long de ma vie et plus particulièrement pour tous les efforts qu'ils ont fait pour que je puisse réaliser mes études durant ces si longues années malgré toutes les difficultés. Sans eux ce jour n'aurait jamais été.

J'adresse également mes remerciements à tous mes enseignants et plus particulièrement à ceux du département GIC de l'Institut de Technologie du Cambodge qui m'ont transmis des connaissances en informatique me permettant de surpasser les problèmes techniques durant mon parcours de thèse. C'est grâce à eux que j'ai pu faire mon premier pas vers la France.

Contents

List of Figures

List of Tables

Notations

- C: set of character classes, $C = \{c_1, c_2, \ldots, c_M\}$
- M: number of character classes
- c_m: a character in the set C
- N: number of Top-N hypotheses in each node given by the single character recognition system.
- NP: number of points in a given on-line handwritten word signal
- P_i: a point in a given signal
- L: maximum level of the lattice. It also refers to the (fixed) maximum number of graphemes composing a character.
- $L(c_m)$: the maximum number of graphemes composing the character c_m
- MP: list of local maximum points in a given signal
- mp: list of local minimum points in a given signal
- O: list of nodes in a lattice or list of observations (in the case of Markov Models)
- $o_{(t,t')}$: a nodes in the lattice, where t and t' indicate respectively the starting and the ending graphemes in the node.
- $(o_{(t,t')} \cup o_{(t'+1,t")})$: the sequence of points of a bi-character sample obtained by concatenating the node $o_{(t,t')}$ and $o_{(t'+1,t")}$.
- T: the number of segmented graphemes for an input word
- $b(c_m|o_{(t-t')})$: recognition probability given by the single character recognizer that the node $o_{(t-t')}$ is recognized as character c_m. This node contains the points in the grapheme t to the grapheme t".
- $a(c_n c_m|o_{(t,t')} \cup \ o_{(t'+1,t")})$: probability that the pair of neighboring nodes $o_{(t,t')}$ and $o_{(t'+1,t")}$ is recognized by the bi-character model $B_{c_n c_m}$ (see section 3.6) as the pair of characters $c_n c_m$.
- Δ_{point}: objective distance between two consecutive points
- Δ_{word}: objective corpus-height

Glossary

- ICR: Isolated Character Recognizer. The therms "isolated character" refers to the handwritten character written in pre-defined boxes.
- BLSTM: Bidirectional RNN with Long Short-Term Memory
- CNN: Convolutional Neural Network
- CTC: Connectionist Temporal Classification
- HMM: Hidden Markov Models
- HWR: Handwritten Word Recognition system.
- MS-TDNN: Multi-States Time Delay Neural Networks
- NLP: Natural Language Processing
- NN: Neural Networks
- PDFs: Probability Density Functions
- PSP : Potential Segmentation Points
- RNN: Recurrent Neural Network
- SCR: Single Character Recognizer. The terms "single character" refers to the handwritten characters segmented from handwritten words.
- SFFS: Sequential Floating Forward Selection
- SVM: Support Vector Machines
- TDNN: Time Delay Neural Networks
- TRIE: TRee Information rEtrieval model

Introduction

Writing is the way of expression for describing something by a set of signs, symbols or alphabet which is known as "writing system". There are many writing systems all around the world, for instance Latin, Arabic, Greek, Logo-graphic, etc. These writing systems have been changed from one period to another and have been adapted to each country/language. The first traces of writing are the meaningful historical records which were drawn on stones or on some other supports such as palm leaves or bamboo. Writing soon became one of the most important ways for communication, expression, information conservation etc. and is used in the daily life, as well as in administrative tasks.

Initially, the first documents were produced manually with handwriting. These historical documents written by well known scientists, artists, writers, etc., represent a considerable value but remain very fragile. Direct physical access to these historical documents may damage them, generating an irreversible destruction of our cultural heritage. Later, documents were produced using modern technology by printing text on paper. However, access to these documents remains limited, as long as they are on their original physical support. Thanks to modern technologies, these documents can be digitized and world-wide distributed in digital version using web-based portals. The original version on the physical support, especially the historical documents, can thus be conserved and protected. Generally, digital documents are in the form of an image, resulting from the scanning process. In other words, the digital documents are represented by a matrix of pixels. Unlike a text format document, the content in an image document is not directly understandable by a computer-assisted application. Therefore, many important operations cannot be directly performed on the digital documents. For instance, searching a given keyword in the content of the digital documents, cannot be directly realized as long as these digital documents are in image format. As a consequence, finding a specific page or specific information among a large digital document database becomes a hard task, or even impossible. To solve this problem, a very classical solution can be used. Digital documents can be annotated with meta-data which allow indexing the documents by some predefined and preregistered keywords. However, this human operation is time consuming and costly. In addition, these meta-data cannot represent all the content of the documents, unless they manually re-transcript the whole content of document. This work can be considered as impossible due to the working time and cost. A computer-assisted solution is, therefore, required. In 2004, Google created the Google Book project to provide on-line books browsing services. Numerous printed books were scanned and converted to text using Optical Character Recognition (OCR). Concerning the historical documents, many projects funded by France and

Europe were created: Navidomass[1] (2007-2010), DIGIDOC project [99] (2011-present), DocExplore[2], Europeana[3] and Valconum[4] for instance. In all these projects, different research problems can be identified, among which document structure analysis, graphic indexing, word spotting and off-line handwriting recognition.

Nowadays, there is a considerable evolution of technology offering different ways to produce documents. However, handwriting is still an irreplaceable method for different reasons: 1) habit of the users, 2) in some application context, using handwriting/paper is easier than using some high technology equipment, 3) handwriting can be used for legal proof. Since the last decade, the explosion of various kinds of interactive mobile devices such as Smartphone, Smartpen, electronic tablets, ..., increases the production of handwriting which are used for different purposes such as education[5], medicine and administration tasks, etc. These new electronic devices store the trajectory of handwriting as a sequence of points, where each point is described by its coordinates (x, y), and/or temporal information and/or pen pressure. This kind of information is known as on-line signal. However, similarly to digital documents, the on-line signal also cannot be directly interpreted by a computer-assisted application. Due to this considerably increasing number of on-line handwriting devices, there is a high demand for on-line handwriting recognition systems and on-line handwriting analysis, for different kinds of applications. For instance, these on-line handwriting recognition systems can be applied in the context of automatic administrative form reading, that allows automatic transcription of the content filled in the form. This transcribed content can then be analyzed by a computer-assisted application. The on-line handwriting recognition system can also be used to recognize handwriting on a meeting whiteboard [91] or a teaching whiteboard [6] which facilitates the indexing process. The research activity on on-line handwriting recognition is strongly supported by the IAPR (International Association for Pattern Recognition) research community since more than 30 years. In 1994, the UNIPEN project was proposed as an initiative of the IAPR in order to create a large on-line handwriting database (UNIPEN database). Two famous international conferences (ICDAR[7] and ICFHR [8]) also focus their research subjects on handwriting recognition. Nowadays, the research of on-line handwriting recognition becomes an interesting activity not only in the research community, but also from a commercial point of view. Different international companies such as Vision Object [9], Microsoft and BIC-Education[10] focus their activities on the on-line handwriting recognition problem.

[1] http://navidomass.univ-lr.fr/
[2] http://www.docexplore.eu/
[3] http://www.europeana.eu/
[4] http://valconum.fr/
[5] http://www.edb.utexas.edu/education/news/2013/ntrig/
[6] http://l3i.univ-larochelle.fr/ASPIC-e-education.html
[7] International Conference on Document Analysis and Recognition
[8] International Conference on Frontiers in Handwriting Recognition
[9] http://www.visionobjects.com/
[10] http://www.bic-education.com/

Our research work is a part of the RecoNomad[11] Eureka project, funded by Oséo[12] and in cooperation with DocLedge[13] company. This project aims at offering an automatic administrative forms reading solution. The application of RecoNomad can be described as follows: an administrative form is manually completed by a human handwriting using an electronic tablet and its special pen. The tablet registers the writing trajectories for each field of the form. Based on these writing trajectories, RecoNomad project firstly aims at identifying the type of the completed form among the different types of forms, earlier trained in the system. Secondly, the system consists in recognizing the writing in each field, in order to automatically transcript all this information and send it to an Information System. This project is therefore composed of two main parts: administrative form identification and on-line handwriting recognition. My thesis focuses on the second part *i.e.* on-line handwriting recognition.

The research activities on handwriting recognition started in the 1960's and were re-activated in the 1980's, after a break in the 1970's. Generally speaking, we can classify the handwriting recognition problem in two classes: isolated character recognition (ICR) and handwritten word/text recognition. In the case of the ICR, the systems generally provide satisfying results. Concerning the handwritten word/text recognition, the problem is much more complex. Indeed, a handwritten word/text is composed by handwritten characters, boundaries of which being generally unknown. Considering the huge number of writers, the variation of handwriting is very large and the connection between the characters in handwritten words is also very complex, especially, in the case of unconstrained or cursive handwriting. In order to limit the complexity of the problem, some authors have limited their research scope by imposing writing constraints, for instance, writers have to write in printed mode and they have to rise the pen between characters, helping the character segmentation. In this case, writers must strictly respect the writing constraints. However, imposing writing constraints to the users is not realistic from the RecoNomad application point of view since writers cannot write using their own writing style. Therefore, nowadays, most of the authors are interested in working in the unconstrained handwriting recognition context, which is considered as a complex problem by the scientific community.

Two main approaches for unconstrained or cursive handwriting recognition were considered in the literature: global and analytical approaches. Using global approaches, systems consider the handwritten word image (off-line signal) or the on-line signal as a whole, and try to recognize it using for instance template matching methods. This kind of systems relies on a huge training dataset containing all the words in the lexicon. When the lexicon needs to be changed, a retraining process is generally required. This is the main disadvantage of this kind of approaches. Therefore, they seem to be

[11] http://l3i.univ-larochelle.fr/Reco-Nomad.html
[12] www.oseo.fr
[13] http://www.docledge.eu/

progressively abandoned. On the contrary, using analytical approaches, systems rely on a preliminary step where the input signal (on-line and/or off-line) is segmented into individual characters or sub-parts of character. Then, each segmented part is recognized (in general, independently) and combined with the others to identify the whole word. The analytical approach based systems rely on a training set containing all the possible characters of each language. Therefore, one of their main advantages compared to global approaches is that they can be adapted to different lexicons without any re-training (as long as the alphabet remains the same). This is one of the main reasons of the great interest for analytical approaches recently observed in the literature.

However, analytical approaches based systems require a segmentation method which can be either explicit or implicit. The recognition strategy based on the combination of implicit segmentation, Hidden Markov Models (HMM) and Viterbi algorithm is very frequently applied to solve handwritten word recognition problems [60, 88, 10]. In this kind of approach, input words are first segmented with a sliding window segmentation method. The segments are then combined with characters and words during the recognition stage with respect to the HMM. Attempts towards discriminative handwriting recognition systems include neural network based recognition [33, 43, 65], often under the form of hybrid systems based on a combination with HMM to address the segmentation problem [33, 65]. However, as far as we know, Support Vector Machines (SVM) were rarely used for handwriting word/text recognition despite its has been successfully used for off-line ICR [54].

Motivated by the effectiveness of the SVM classifier, in this thesis we propose a discriminative approach for on-line Handwritten Word Recognition system (HWR) that combines discriminative classifiers with dynamic programming (Viterbi-like). Our system can be described as follows: first, the input handwritten word is explicitly oversegmented into graphemes which are further used to create a lattice of L levels. Each node of the lattice is considered as a character to be recognized by a Single Character Recognition (SCR) which relies on SVM classifier using both on-line and off-line features. The recognition result is further refined with bi-character models which rely on Logistic Regression. These models take into account the graphical context by jointly recognizing the neighboring characters in order to cope with the problem of shared character parts, which refers to the fact that one character class may have a visual appearance similar to a part of other characters classes. This problem will be detailed in the manuscript. Afterwards, the recognition hypotheses are efficiently processed by dynamic programming (similarly to the Viterbi algorithms) to find an optimal sequence of characters for the input handwritten word.

The proposed system can be used together with a large and flexible lexicon; it is adapted to omni writer applications, without any specific requirement concerning the capturing devices. An experimental evaluation is performed on the Unipen-ICROW-03 database. Using a kernel SVM for SCR and a Logistic Regression classifier for bi-character recognition, we demonstrate that the proposed method outperforms a baseline

HMM-based system in terms of performance and computational time.

This thesis is organized as follows:

Chapter 1: Handwriting recognition - general presentation

This chapter presents an overview concerning the handwriting recognition problem. We introduce the two main kinds of handwritten data (off-line and on-line) as well as the different categories of handwriting styles, according to the literature classification. This chapter mainly focuses on the major difficulties when dealing with unconstrained handwriting. Different methods for contextual integration in the literature are also introduced. The objectives and scope of our research are given at the end of the chapter.

Chapter 2: State-of-the-art of handwriting recognition systems

This chapter concentrates on the existing systems in the state-of-the-art. It starts with a short introduction and a discussion on the global and analytical approaches. Then, the presentation focuses on analytical approaches along with a brief introduction of each step related to this kind of approach. Afterwards, different frameworks are presented along with some example systems. The chapter ends with a set of conclusions, based on a table of advantages/drawbacks of each framework.

Chapter 3: Handwritten words recognition system based on two levels analysis: character and bi-characters level

This chapter is dedicated to the presentation of the proposed system. The global view of our method is introduced. Each section of this chapter presents the method(s) by explaining in details the choices made at each step when conceiving the system. The conclusion of this chapter summarizes the contributions of this system compared to existing systems in the literature.

Chapter 4: Experiments and Discussion

This chapter aims at evaluating the effectiveness and efficiency of the proposed system. It presents the experimental protocols/configurations and discusses the experimental results. First, we introduce the different databases that were used in these experiments. Then, we present a series of experiments which are performed with different configurations in order to evaluate

the different methods/strategies for each stage of the system. We also compare the proposed system with a baseline HMM-based system and discuss their recognition results.

Chapter 5: Conclusion and future work

This chapter gives a final conclusion of our research by discussing on the strong and weak points of the proposed system. Finally, we present the perspectives of this work which should allow improving the effectiveness of the system.

1 Handwriting recognition - general presentation

This chapter presents an overview concerning the handwriting recognition problem. We introduce the two main kinds of handwritten data (off-line and on-line) as well as the different categories of handwriting styles, according to the literature classification. This chapter mainly focuses on the major difficulties when dealing with unconstrained handwriting. Different methods for contextual integration in the literature are also introduced. The objectives and scope of our research are given at the end of the chapter.

Contents

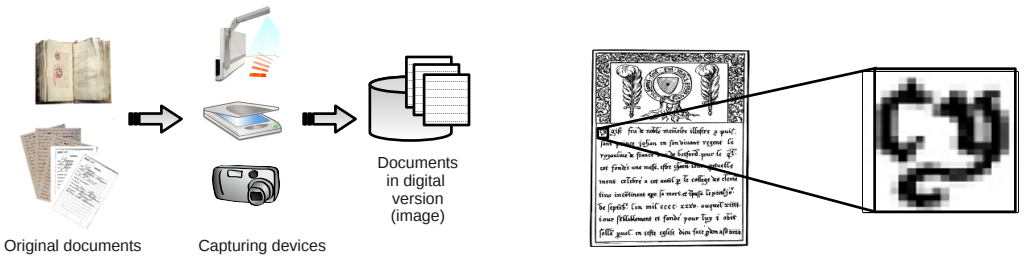
Our research focuses on unconstrained on-line handwritten word recognition which is still known as difficult problem in the research community. The terms "unconstrained" refers to the handwriting that writers can write with their own writing style, without any constraint of writing and without any constraint concerning the acquisition device. In such cases, characters of handwritten words can be more or less connected by ligatures. The boundaries between characters are unknown, which makes the problem more complex.

This chapter gives a general introduction concerning handwriting recognition. It is organized as follows: section 1.1 introduces the two main types of handwritten documents(off-line and on-line) while section 1.2 introduces the different categories of handwriting styles. Section 1.3 illustrates the difficulties when dealing with unconstrained handwriting. Section 1.4 gives a brief introduction of contextual integration that can be used for post-processing. Finally, section 1.5 introduces the scope of our research.

1.1 Off-line and on-line handwritten data

In the domain of handwriting recognition, we generally refer to two categories of handwritten data: off-line and on-line.

Off-line data refers to documents which are obtained by scanning or photographing with a digital camera and which are generally represented in the form of an image. Documents concerned by this process are generally on a physical support (most of the time paper) and can be historical documents, music score, administrative forms, journals and magazines ... The digitization of this kind of documents is illustrated in Figure 1.1(a). The resulting images may be in color, grayscale or binary according to the capturing device and the objective of digitization. Figure 1.1(b) is an illustration of an off-line document of an historical document.

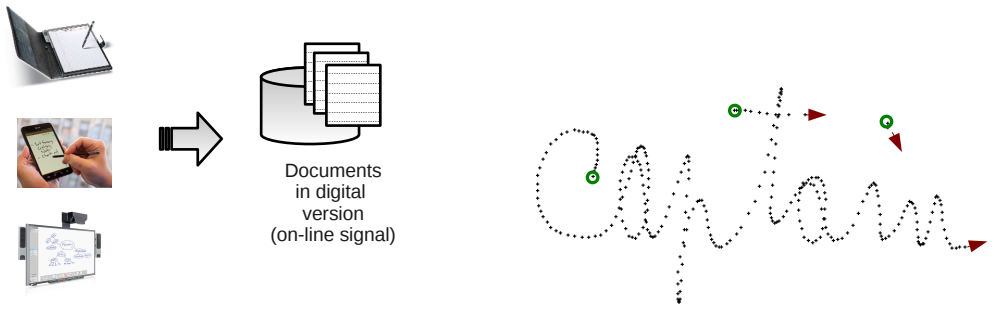


(a) Off-line document: digitization process (b) An example of off-line document

Figure 1.1: Digitization process of off-line documents.

On the other hand, on-line data refers to the data which is acquired through modern devices such as smartphones or tablets, and for which the information is represented by a sequence of points describing the trajectories of handwriting, as illustrated in Figure 1.2.

In the domain of handwriting recognition, the on-line data is generally known as on-line signal. Each point in the on-line signal is characterized by its coordinates (x, y) which describe its spatial position. Some capturing devices may also provide the acquisition time and/or pen pressure and/or pen inclination. Figure 1.2(b) illustrates an example of on-line signal of the word "*captain*". The green circles represent the pen-down (*i.e* pen touches the capturing device) and the red arrows represent the pen-up (*i.e* pen leaves the capturing device). We have to mention that, concerning the sampling resolution, some capturing devices only rely on spatial sampling (DPI[1]). In this case, only the points coordinates are provided and the distance between two consecutive points are homogeneous. Some capturing devices, on the other hand, rely on temporal sampling (DPIS[2]). Each point is represented by its spatial coordinates and its temporal information. As a consequence, distances between two consecutive points are heterogeneous. In this case, the density of points during slow writing is more important than the density of points during rapid writing.

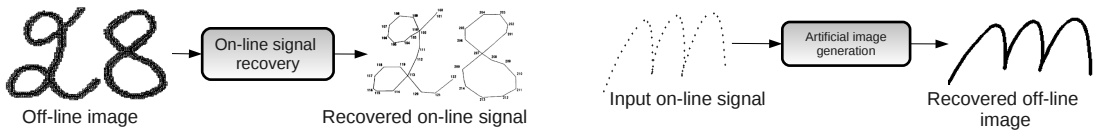


(a) On-line signal capturing process (b) An example of on-line signal

Figure 1.2: On-line signal capturing process.

These two types of data (off-line image and on-line signal) provide different kinds of information. In the literature, the effectiveness of on-line handwriting recognition systems is generally better than the one of off-line handwriting recognition systems, thanks to the possibility to use dynamic information of on-ligne signal. Therefore, many authors have focused on recovering on-line signal from off-line image, as illustrated in Figure 1.3(a) [11, 85, 102]. The recovered on-line signal can be used to improve the effectiveness of the off-line handwriting recognition. However, deducing the on-line signal based on the off-line image is not straightforward because of the necessity to guess/reconstruct some temporal information. On the other hand, creating an off-line image from an on-line signal of handwriting is easier. An artificial image can be simply created by connecting the sequence of points, as illustrated in Figure 1.3(b). Some authors try to use both on-line signal and its recovered off-line image in order to take profit of their complementarity.

[1]Dots Per Inch
[2]Dots Per Inch and Second



(a) On-line signal recovery (b) Off-line signal recovery

Figure 1.3: a) On-line signal recovered from off-line image, extracted from [11]. b) off-line image generated from on-line signal.

Regarding handwriting recognition, these two categories of documents rely on very similar recognition approaches and share some common difficulties. In both cases, isolated character recognition can be considered as an almost solved problem. However, the handwritten word/text recognition is still considered as a complicated problem. This can be explained by the fact that, since a word is composed of multiple characters, the variation in handwritten words may be more important than the variation in isolated characters since shapes of character in handwritten words change according to their surrounding characters. In the context of handwritten word/text recognition, it is hard to detect the connection points between characters and to decompose the handwritten word into characters, especially when dealing with certain categories of handwriting styles, as explained in the next section.

1.2 The different categories of handwriting styles

C.C. Tappert *et al.* [134] classified handwriting styles of English language (and also some other Western languages which rely on the Latin alphabet) into 5 categories, based on their level of difficulty in the recognition process, as illustrated in Figure 1.4.

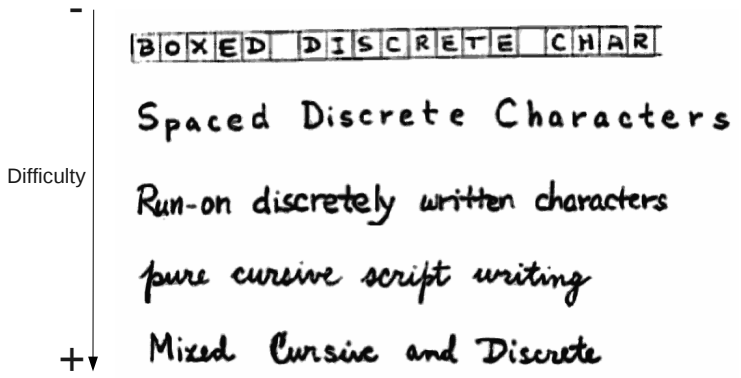


Figure 1.4: Different categories of handwriting styles, extracted from [134].

These 5 categories of handwriting styles are:

1. **Boxed discrete characters:** they are also known as isolated characters since the characters are written in pre-defined boxes. This category is the easiest one compared to other categories since no segmentation process is required. The pixels (for off-line image) or strokes (for on-line signal) contained in each box represent one character. Hence, an ICR can be directly applied.

2. **Spaced discrete characters:** there is no pre-defined box in this case. However, writers have to respect the writing constraint consisting in leaving a blank space between characters. In this case, a simple character segmentation method based on blank space detection or connected component extraction can be applied in order to segment a handwritten word into a sequence of characters. Then, each segmented shape can be directly submitted to an ICR.

3. **Run-on discrete characters:** similar to category 2, writers have to hold-up pen after having written each character with or without leaving blank space between characters. This category of writing provides advantage only for on-line handwriting recognition systems, since pen-up and pen-down information can be used for character segmentation. In this case, no segmentation method is required, since each stroke is supposed to belong to only one character. However, systems have to combine different strokes to form a character, since one character may be composed of multiple strokes.

4. **Pure cursive script writing:** each word has to be written cursively from the beginning to the end of the word. This writing constraint facilitates the segmentation of the input text-line into a sequence of words. Each blank space represents a potential segmentation point between two words. In addition, in the case of on-line signal, the distortion of the signal caused by the delayed strokes (accents, bars on the 't' etc..., further explained in section 2.2.1) is reduced since the delayed strokes can be added only at the end of the word. However, segmenting an input handwritten word into a sequence of characters is much more complicated compared to the three previous categories since segmentation points are generally unknown.

5. **Mixed Cursive and Discrete:** it is also known as unconstrained handwriting and is obviously much more difficult than the other categories. Writers can use their own writing style, without any writing constraint. The variations of possible handwriting styles is huge and the complexity of connections between characters is important.

Since the last two decades, researchers have focused their interest mainly on the unconstrained handwriting recognition problem. However, this problem is still considered as complex and challenging. The complexity of this kind of writings is caused by different factors which are presented in the next section.

1.3 Major difficulties for a handwriting recognition system

The complexity of handwriting depends on different factors such as the variation of writings, quality of handwriting, confusion between characters, etc. In this section, we try to point out the difficulties which generally degrade the effectiveness of handwriting recognition systems.

1.3.1 Variation of writings

The large variations of handwriting is the main cause of recognition errors. In order to deal with this problem, recognition systems must be able to handle the large variety of possible handwriting styles. This large variety of handwriting may be due to 3 important factors: writers, surrounding characters and digitization process.

Writers factor

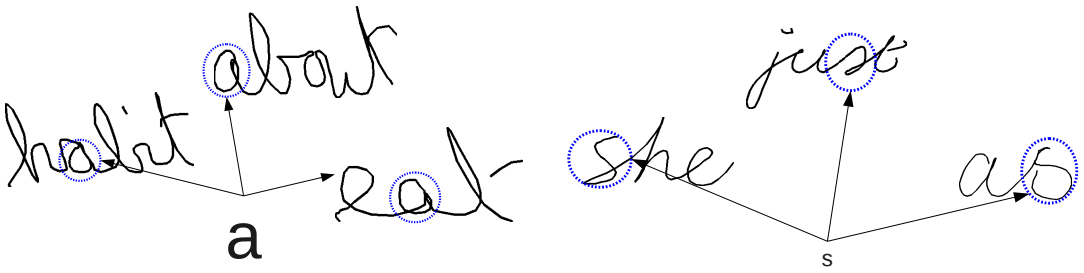
Each writer has his/her own writing style which may change according to his/her nationality, ability, gender, age, left-handed or right-handed writing, health, ... The writing style of each writer is unique. One writer may produce very different shapes for the same letter, even within the same word, as illustrated with the two characters 'i' and 'l' in the words "million" in Figure 1.5. In addition, some writers provide very special shapes for some character classes. Their visual appearances are very different from the standard shape and hard to be recognized even by a human being. For instance, the writings of character '*q*' in red circles illustrated in Figure 1.6.

Figure 1.5: The variety of the characters '*i*' and '*l*' in the writings of the word "*million*" (each word being written by a different writer), extracted from IRONOFF database (see section 4.1.1).

Figure 1.6: Some special shapes of the character '*q*' in the writings of the word "*quatre*", extracted from IRONOFF database (see section 4.1.1).

Surrounding characters

Characters composing handwritten words in the English language and some Western languages which are based on Latin alphabet are more or less connected. In this case, the character shape may be deformed by the ligatures joining neighboring characters. Writings of the same character may also change according to the surrounding characters (*i.e* characters written on its left and right). For instance, in Figure 1.7(a), the shape of the character '*a*' in the word "*habit*", "*about*" and "*eat*" are completely different even though they are written by the same writer. Idem in the case of the character '*s*', as illustrated in the Figure 1.7(b).



(a) writing variation of the character 'a' (b) writing variation of the character 's'

Figure 1.7: Example of writings variation. Writings of the character 'a' and 's' change according to their surrounding characters.

Digitization process

Capturing devices and digitization conditions play a very important role on the quality of the document and on the quality of handwriting. In the case of off-line documents, the digitization hardware and it parameterization have a great effect on the appearance of the resulting image. Some systems are specifically designed for some kinds of documents and some digitization conditions.

For on-line signal, the capturing device has a strong effect on the resulting signal. There is no standard definition for capturing resolution and capturing method. For instance, Figure 1.8 shows an example of the writing of the character '*a*' registered by two different devices which use DPIS capturing methods but with different capturing resolutions. We have to mention that some capturing devices only provide the sequence of points between the pen-down and pen-up while some others also provide an extrapolation of the sequence of points between pen-up and pen-down.

In order to cover all the variation problems, in 1994, the UNIPEN database [49] (see section 4.1.2) have been created, under the initiative of IAPR[3]. This database contains

[3]http://www.iapr.org/

(a) (b)

Figure 1.8: Different writings of the character 'a', given by different capturing devices using different capturing resolutions.

a large set of on-line handwriting characters, words, text-lines written by different nationalities which were collected by different universities and laboratories. The idea is to include a large amount of variations in the training samples, so as to make the models as robust as possible towards handwriting variations. However, we still cannot ensure that the collected data is large enough to cover all the variations of handwriting and can, as a consequence, be used to create a recognition system for any language that relies on the Latin Alphabet. In the case of off-line data, on the other hand, creating a standard benchmark is more complicated. Indeed, the variation of handwriting strongly depends on the documents (for instance, its age, background, etc.), the digitization process, etc.

1.3.2 Quality of handwriting

Beside the variation problems mentioned above, some handwritings are of a very low quality, *i.e.* the character shapes are very different from the usual shapes, and sometime with spelling mistake(s). In such cases, even a human being may have difficulties to recognize them (see Figure 1.9). This explains why the recognition rate of a given system has a strong dependence to the writers [4, 127].

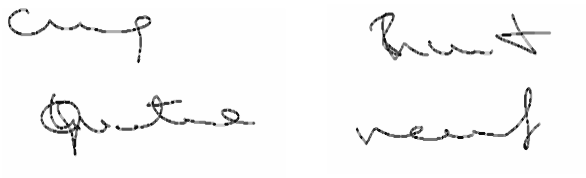


Figure 1.9: Examples of low quality writings.

1.3.3 Similarity between characters

Beside the problems mentioned above, we also notice that some shapes of different character classes may be very similar, as it can be seen in Figure 1.10, between the shape of the characters 'b' and 'f', 'o' and 'v', etc. The similarity problem can also concern some specific classes for which lowercase and uppercase are very similar, *e.g.* the character

‘*o*‘. An example of this kind of ambiguity is illustrated in Figure 1.11.

Figure 1.10: Example of similarity between writings of different character classes, extracted from [4].

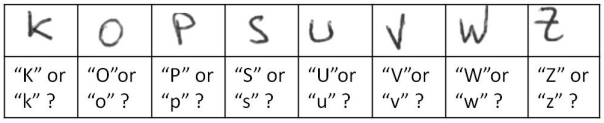


Figure 1.11: Ambiguity between lowercase and uppercase characters.

In the case of unconstrained handwritten words, confusions are more important since the ligatures between character are strongly variable, which generates a high level of potential ambiguities. In the writing of the word ”*loting*” illustrated in Figure 1.12(a), the writing of the character ‘*o*‘ could be confused with the writing of the character ‘*a*‘ and the writing of the character ‘*t*‘ is similar to the writing of the character ‘*d*‘. In the writing of the word ”*jumped*” of the Figure 1.12(b), the visual appearance of the character ‘*m*‘ is very similar to the character ‘*w*‘ while the ligature stroke between the characters ‘*u*‘ and ‘*m*‘ could generate an ambiguity with the character ‘*u*‘.

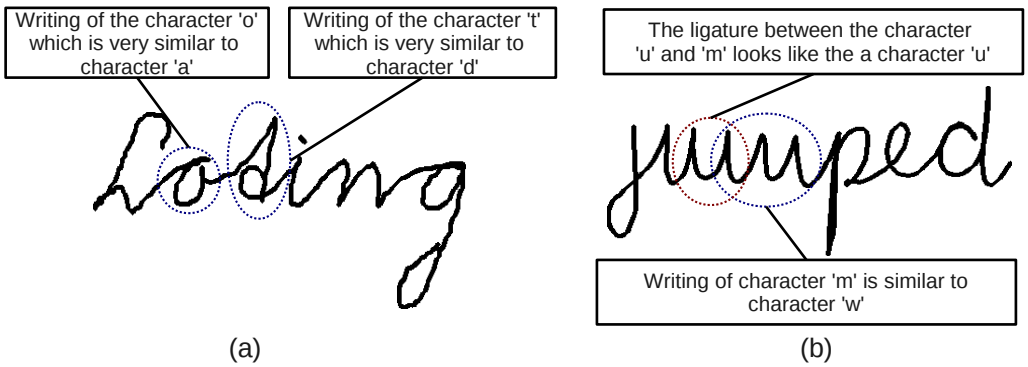


Figure 1.12: Examples of ambiguities between different character classes in handwritten words.

1.3.4 Shared character part

In the context of unconstrained handwritten words, as mentioned above, the character boundaries are not known *a priori*. Segmenting an input handwritten word into a sequence of characters is therefore a hard task. Furthermore, systems may face the "shared character part" problem. It refers to the fact that one character class may have a visual appearance similar to a part of other characters classes. For instance, as one can see in Figure 1.13(a), writing of the character '*a*' can be split into characters '*c*' and '*i*'. In this case, we can say that the characters '*c*' and '*i*' share part in the character '*a*'. Similar problems can be seen in the 2 others examples of Figure 1.13. The characters '*l*' and '*c*' share part in the character '*k*' while the characters '*o*' and '*l*' share part in the character '*d*'.

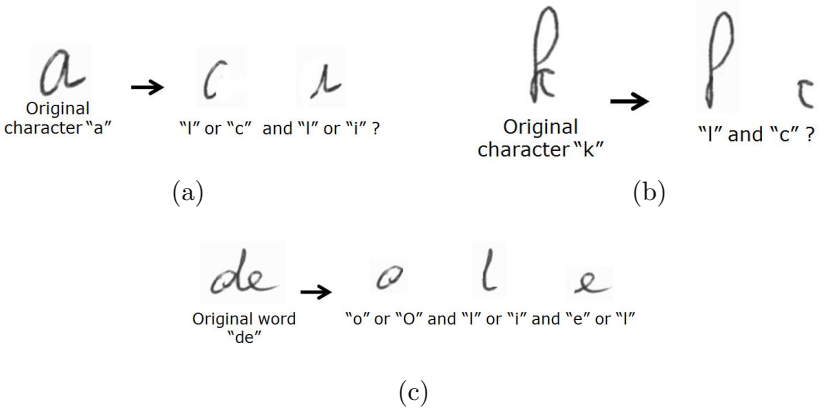


Figure 1.13: Problem of shared character part.

The variation of writings, the possible bad quality of the writing, the similarity between different characters, and the shared character part problem partly explain why unconstrained handwritten word recognition is a difficult problem. Some handwritten words are even difficult to recognize by a human being. The human recognition is sometime possible only because the human brain integrates some contextual information (graphical, lexical, semantic, ...) into the recognition process. In any case, integrating contextual information helps for the human recognition of unconstrained handwritten words (in terms of effectiveness and efficiency), and of course a similar effect can be observed when using computer-assisted recognition systems. Therefore, integrating external context into the handwriting recognition system seems unavoidable. It is the scope of next section.

1.4 Context integration

Due to the large variation of handwriting explained in the previous sections, recognizing handwritten words by a machine learning system is a complex task. In order to deal

with this problem, systems need some additional contextual information. Different levels of contextual information may be used: graphical, lexical, syntactical and semantic context, as illustrated in Figure 1.14.

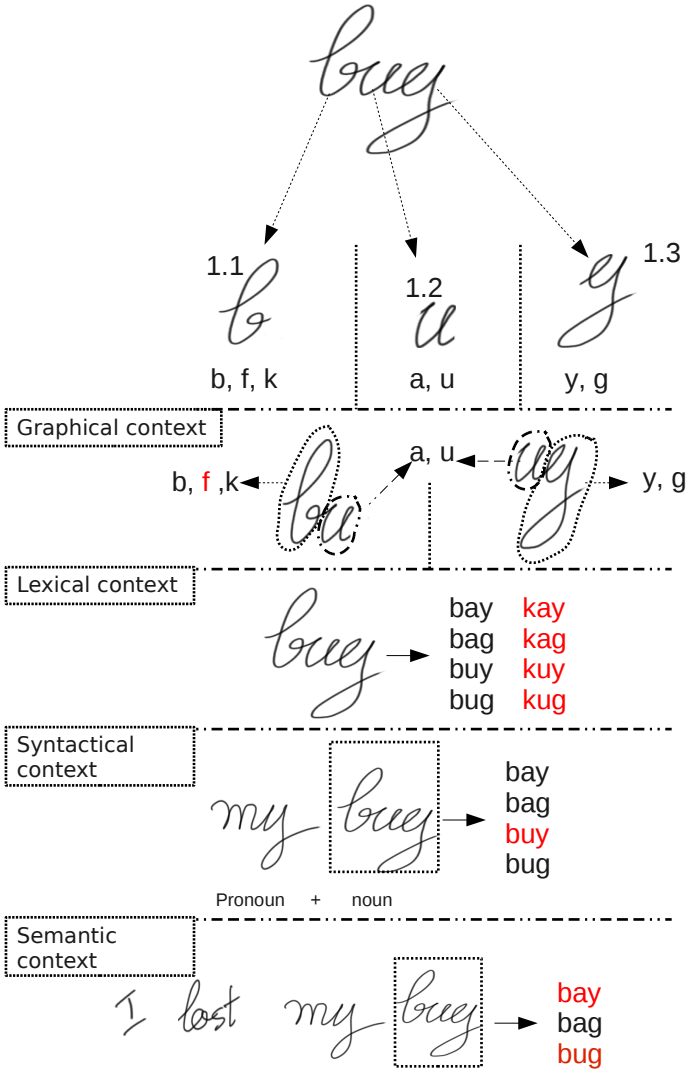


Figure 1.14: Example of contextual integration in a recognition system. At each step, the candidates in red are abandoned.

Taking the example in Figure 1.14, given a handwriting of the word "*bag*" and by considering that the segmentation is correct, some errors may occur while trying to recognize each segmented character. For instance, system may recognize shape 1.1 as characters '*b*', '*f*' *or* '*k*', shape 1.2 as characters '*a*' *or* '*u*' and shape 1.3 as characters '*y*' *or* '*g*'. By integrating different levels of context, errors can be reduced, as explained

below:

- Graphical context: if neighboring characters are taken into account, the character candidate *f* on shape 1.1 may be abandoned, since the lower loop of the character 'f' must normally be located in the lower zone (see Figure 3.3) of the handwritten word. The input signal may be recognized as different words {"bay", "bag", "buy", "bug", "kay", "kag", "kuy", "kug"}.

- Lexical context: by integrating a lexicon, the search space can be limited. This method is very frequently used in the literature. It allows to improve at the same time the effectiveness and the efficiency of the system. The recognized words given by the system must exist in the lexicon. Hence, the words {"kay", "kag", "kuy", "kug"} can be abandoned since they do not exist in the lexicon.

- Syntactic context: this method consists in taking into account the grammar of a given language. In this example, let us consider that the input handwritten word is written after the word "*my*" which is a *pronoun*. If we suppose that the probability that a *verb* follows after a *pronoun* is 0, then the word "*buy*" which is a *verb* can be removed from the list.

- Semantic context: this method takes into account the meaning of the words in its sentence. In this example, in order to select the best word among the three remaining words ({"*bay*","*bag*","*bug*"}) provided by the previous step, the meaning of each word in the phrase is taken into account. In this case, we can say that the word "*bag*" is the most likely one.

Using the graphical context, systems take into account basic information on the shape of the input handwritten word and the shapes of the other characters composing the word. Using the three other types of context information, systems take into account a high level context and external knowledge. The syntactic and semantic contexts can be applied only for text-line recognition problems.

Despite the efforts made by many researchers since haft-century, unconstrained handwriting recognition remains a difficult and challenging problem. Therefore, in this thesis, we focus on certain specific objectives and scopes, as mentioned in the following section.

1.5 Objectives and scopes of this thesis

The objective of our research is to propose an unconstrained on-line handwritten word recognition system which should be able to be applied in the context of omni writer, with a large and flexible lexicon, without any limit concerning the capturing device.

- Omni writer: writers are not known *a priori*. Hence, the system should be trained in general condition, without any specific features related to any writer.

- Not limited to a particular capturing devices: the system should be able to deal with any capturing devices without any re-training process, whatever the characteristic or parameterization of the capturing device.
- Large and flexible lexicon: the system should be able to deal with large lexicons, containing around 20 000 words. No re-training process must be required if the lexicon needs to be changed (*flexible lexicon*). Furthermore, the system should be able to be applied to any language which is based on the Latin alphabet, without any re-training (only changing the lexicon).

In this research, we limit our scope to lowercase and non-accented Latin words. It has to noted that in most cases, in the presence of an accent, the lexicon and integration of context information can help regrouping the good word (as long as the letter supporting the accent has been correctly recognized).

1.6 Conclusion

This chapter gives a general presentation of the difficulties linked to handwriting recognition. In the literature, we consider two types of documents: off-line and on-line. The handwriting styles are classified into five categories based on their difficulty level for recognition. We have also pointed out the major difficulties and problems that researchers frequently encounter. Due to all these difficulties and problems, the unconstrained handwriting is the most difficult and challenging problem in the research domain.

In the next chapter, we introduce and discuss some state-of-the-art HWRs.

It is important to mention that, in this thesis, we classify the handwritten characters into two groups: isolated characters on one hand and single characters on the other hand. Isolated characters refer to the characters which are individually written in predefined boxes, while single characters refer to the ones which are segmented from handwritten words. In general, the shapes of the single characters are more heterogeneous than those of the isolated characters.

2 State-of-the-art of handwriting recognition systems

This chapter concentrates on the existing systems in the state-of-the-art. It starts with a short introduction and a discussion on the global and analytical approaches. Then, the presentation focuses on analytical approaches along with a brief introduction of each step related to this kind of approach. Afterwards, different frameworks are presented along with some example systems. The chapter ends with a set of conclusion, based on a table of advantages/drawbacks of each framework.

Contents

Since the seminal work of P.Marmelstein and M.Eyden [100] in 1964, a large amount of research has been dedicated to handwriting recognition problem for both on-line and off-line data. The research activity related this problem was specifically active in the 1990's. In the case of on-line handwriting recognition, many authors concentrate on the HMM based approach. In this chapter, we study the existing systems in the literature related to on-line and off-line handwritten recognition problems.

2.1 Global approaches *vs* analytical approaches

As mentioned in the introduction, in general, we can classify the recognition systems into two main groups: systems relying on global approaches and those relying on analytical approaches [118, 82, 107].

2.1.1 Global approaches

Systems relying on Global Approaches (also known as Holistic Approaches) consider each word in a given lexicon as one class and try to recognize an input handwritten word as a whole.

Features are computed from the whole shape of the input handwritten word. These features are used to feed a supervised classifier such as Support Vector Machines (SVM), Neural Network (NN) ... or a generative model such as Hidden Markov Model (HMM), where each class corresponds to a word in a given lexicon [42, 149, 93, 51, 36, 70].

This kind of systems needs a huge set of training data because each word in the lexicon needs to be trained with a large number of samples in order to overcome the high variability of data. Thus, if the lexicon needs to be changed, the system has to be re-trained with another updated huge training dataset. For these reasons, global approaches are convenient only when the lexicon is fixed and small (not more than 30-40 words) [93, 82]. For instance, it may be efficiently used for bank checks recognition [62, 76]. In some other cases, such as the recognition of numbers with a varying number of digits, these approaches cannot be applied because of the infinite number of digit combinations.

Analytical approaches have therefore been widely presented in the literature. We propose in the following a state-of-the-art of these approaches.

2.1.2 Analytical approaches

Unlike systems relying on global approaches, systems relying on analytical approaches try to recognize the sequence of characters composing the input handwritten word. Therefore, systems relying on these approaches require a preliminary segmentation step.

The input handwritten word is generally segmented into individual graphemes (i.e. characters or sub-parts of characters) by an explicit or implicit segmentation method. Each grapheme is then analyzed (in general independently) in order to find the character sequence which composes the input handwritten word.

Using these approaches, systems train all the possible graphemes/characters of a given language as basic models. These basic models are then used to recognize any given word in the lexicon. This kind of systems relies on a training dataset containing all the possible graphemes/characters. Retraining process is not required if the lexicon is changed. Therefore, these approaches appear to be more convenient than global approaches with a large and flexible lexicon.

2.1.3 Discussion

Analytical approaches have many advantages compared to global approaches. First, systems based on this kind of approaches can be used in flexible lexicon context without any retraining process. Secondly, only analytical approaches can be used in lexicon-free context, which remains a very difficult problem.

Nowadays, global approaches seem to be progressively abandoned, unless they are combined with analytical approaches [93]. As a consequence, the following sections will be only dedicated to analytical approaches.

2.2 A focus on analytical approaches

The general process of analytical approaches is illustrated in Figure 2.1. Each handwritten word is first normalized (see section 2.2.1) and segmented into graphemes (see section 2.2.2) (*i.e.* characters or sub-parts of character) by using an explicit or implicit segmentation method. Depending on the system, pre-processing may be performed before or/and after segmentation. Then, a feature vector is extracted from each grapheme or group of graphemes (see section 2.2.3) to feed a word recognition system. The word recognition process (see section 2.2.4) is used to find the most likely words relying on classifier(s)/model(s) [60, 65, 81, 71, 73]. Some post-processing methods may be used in order to limit the searching space and improve the effectiveness and the efficiency of the word recognizer (see section 2.2.5). In the following section, we give the overview description of each of these stages.

2.2.1 Pre-processing and normalization

As mentioned in section 1.3, many factors influence the variation and the quality of handwriting: writer personality, writing context, capturing devices (different resolution and/or capturing method)... Hence, a pre-processing step is often necessary to reduce

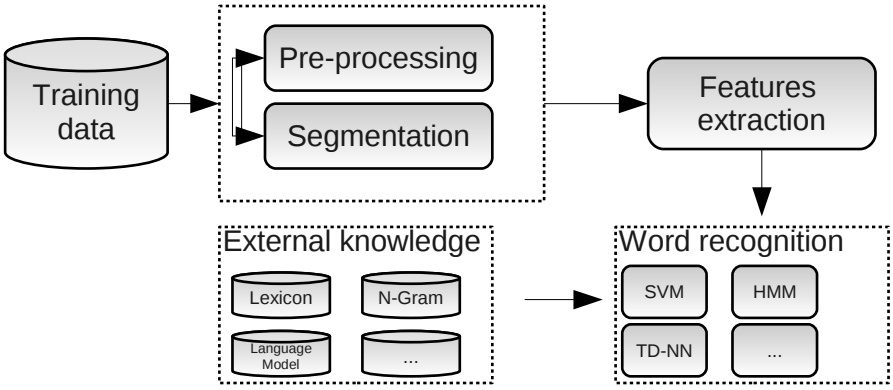


Figure 2.1: General view of the systems relying on analytical approaches

the variability, eventually filter the noise and if necessary, standardize signals from different capturing devices. Pre-processing methods can be divided into two groups: noise removing (filtering, smoothing, ...) and shape normalizing (slope correction, size normalization ...).

For off-line data, three pre-processing methods are usually applied:

- Slope (or skew) correction: consists in correcting text or word misalignment which frequently occurs in handwritten documents.
- Slant correction: consists in correcting the inclination of writing.
- Size normalizing: consists in normalizing the input data in order to ensure that all characters in each handwritten word have approximately the same size.

In the case of on-line data, the important techniques which are commonly used for pre-processing are: interpolating, smoothing and re-sampling [45, 65, 128, 3].

- Interpolating: consists in adding missing points to the on-line signal. Missing points may be due to different factors: capturing device, writing speed, ...
- Smoothing: consists in correcting noisy points in the on-line signal. Such points may be caused by capturing device, writer's shaking hand, ...
- Re-sampling: consists in standardizing the input signals with different resolutions (see Figure 1.8), in order to build a new signal which contains N points where the distance between two consecutive points is equal to a given objective distance Δ.

Beside these three specific pre-processing methods, slope correction and size normalization may be also applied.

Let us also mention the problem related to what is generally called *Delayed Strokes*, when considering on-line data. Indeed, delayed strokes generally correspond to strokes representing dots of characters ('i' or 'j', for instance), crossings in characters such as 't' or accents on special characters such as: 'é', 'è', 'ê' in some specific languages. Delayed strokes are written at different moments and in different orders as illustrated in Figure 2.2, where the crossing of character 't' and dots of character 'i' can be written in many different orders.

- Figure 2.2(a): delayed strokes are added at the end of the word, respectively the bar of the character 't' and then the dot of the character 'i',
- Figure 2.2(b): delayed strokes are added at the end of the word by first adding the dot of the character 'i' and then the bar of the character 't'
- Figure 2.2(c): the bar of the character 't' is added after 'tr' and the dot of the character 'i' is added at the end of the writing.

(a) (b) (c)

Figure 2.2: The delayed strokes (dot of the character 'i' and bar of the character 't') in the writings of the word 'trip' may be written in different ways.

The way the delayed strokes are analyzed has a great impact on the system. Some systems such as [131, 60, 3] keep and analyze them, while some others such as [65] detect and remove them during a pre-processing stage. The latter systems make the assumption that the use of a lexicon will correct all the possible ambiguities (*e.g.* between 'l' and 't'), which is sometimes insufficient and requires a complex language model to correct them (*e.g.* "white" or "while"). In addition, removing delayed strokes may lead to an irreversible loss of information decreasing the effectiveness of the system, especially in some languages such as French, Arabic ... where the delayed strokes (diacritics in Arabic language) are numerous and carry important information [147, 8].

2.2.2 Segmentation

Two main types of segmentation methods are generally used in the literature: explicit and implicit segmentation. A system relying on explicit segmentation tries to find the

Potential Segmentation Points (PSP) to segment the input data into a sequence of characters or graphemes. The system relying on implicit segmentation uses, in general, a sliding window to segment the input data into a sequence of observations. These two segmentation methods are briefly introduced in the next sections.

2.2.2.1 Explicit segmentation

There are two kinds of explicit segmentation methods: pure explicit methods and segmentation/recognition methods.

Pure explicit segmentation methods directly decompose the input handwritten word into individual characters. The result of a pure explicit segmentation method is generally a sequence of graphemes, each of which is considered as a potential character. Then, a SCR is used to recognize each segmented character [38, 78, 104, 30].

Systems relying on pure explicit segmentation require a very effective segmentation method. This stage is very important since segmentation inaccuracies may lead to word recognition errors. However, considering the complexity of the problem, there is not any real perfect segmentation algorithm.

In order to solve this problem, explicit segmentation/recognition methods have been presented [82, 107]. Such methods aim at integrating a SCR during the segmentation process in order to reduce errors due to segmentation methods limitation. Many segmentation methods are available in the literature, such as contour analysis based methods [24] (designed for off-line data) and trajectory based methods [4, 3] (designed for on-line signal), etc. The explicit segmentation/recognition methods provide a set of graphemes which is further used to create a lattice at L levels corresponding to the input handwritten word.

2.2.2.2 Implicit segmentation

Systems relying on implicit segmentation do not rely on any segmentation module. They generally use a sliding window parsing the handwriting data. In this case, the obtained result is a set of segmented windows. For each position of the window, a set of features is extracted from the signal in order to feed a dynamic model such as HMM, HMM/NN, etc. which in general has been previously trained for each word in the lexicon.

2.2.2.3 Discussion

Recognition systems relying on explicit segmentation/recognition and systems relying on implicit segmentation methods present some advantages compared to those relying on pure explicit segmentation because the latter may generate word recognition errors due to the limitations of segmentation methods.

In the literature, we observe that systems relying on implicit segmentation methods are more popular for handwriting recognition and generally provide better recognition rates. Surprisingly, explicit segmentation/recognition methods are frequently used for numeral recognition system [55, 129, 156, 153].

2.2.3 Feature extraction

In any classification problem, feature extraction is the fundamental step which conditions the performance of the recognition systems. There are two types of input data: on-line signal and off-line image. As mentioned earlier in section 1.1, on-line signal is acquired through electronic devices such as tablets or smart-phones ..., while off-line image is obtained by scanning handwritten documents. Most of the time, features used for on-line recognition are quite different from those used for off-line recognition. However, an approximation of the off-line data may be computed by interpolating points from the on-line data. This strategy enables to have two points of view on the data and to use both on-line and off-line features in the on-line handwriting recognition system.

2.2.3.1 On-line features

On-line features are generally extracted locally from each point of the signal. The commonly used features in the literature are: direction, curvature, normalized (x, y) coordinates [5, 128]. Beside these local features, Seni *et al.* [44] introduced Zone features by decoding each point of the signal into pre-defined zones (see Figure 3.3): middle, lower, or upper zone. These features may be used to distinguish characters having similar temporal representations such as 'e' and 'l', 'a' and 'd' etc. Some authors [4, 60] also try to represent the input signal by a set of primitives (cusps, crossings, and loops) in order to take profit of the high level information.

2.2.3.2 Off-line features

Off-line features can be locally extracted from some selected pixels (or segments) of the input image, and generally carry information about spatial distribution and/or local directions of the handwriting, enabling to describe the shape of the words. Some famous features can be cited such as Freeman Chain Code [35, 94], zoning, ... Beside these features, the invariant moment-based descriptors such as Hu, Legendre, Zernike's, ... [54, 136] can also be used. This kind of features are extracted from the whole shape of the input image. As far as we know, the invariant moment-base descriptors have never been used for on-line handwriting recognition, while these features are widely used in pattern recognition, especially off-line ICR [54]. SIFT descriptor has been also used in some recent research for off-line digit recognition [141] and word spotting [120].

2.2.3.3 Discussion

The comparison between different families of features is still the object of many discussions. Each category of features (on-line/off-line) has its own set of advantages and

drawbacks. For instance, off-line features are more robust than on-line ones in the presence of variations in the writing trajectories (writing direction, number of strokes, order of strokes, ...), as illustrated in Figure 2.3. This figure shows that, depending on the neighboring characters, there is a large variation in the writing trajectories of the letter 'a'. This has a great impact on the on-line features, while the off-line features remain more stable, since off-line features characterize the shape. In addition, on-line features also depend on the handwriting hand of the writer (*e.g.* left-handed or right-handed writer). Indeed, the writing trajectories of left-handed writer are generally different from those of right-handed writer.

Figure 2.3: Variation in the writing trajectories of the letter 'a' depending on its neighboring characters.

Some authors try to combine these two categories of features to take advantages of their complementarity. Some experiments have shown a remarkable improvement in terms of effectiveness [50, 103, 65, 40, 69, 91, 89, 90, 117].

2.2.4 Word recognition process

As presented in section 2.2.2, the general result of a pure explicit segmentation method is generally a sequence of graphemes each of which is considered as a potential character. On the other side, explicit segmentation/recognition methods provide a lattice of graphemes, combination of graphemes corresponding to the input word. In the case of implicit segmentation methods, the obtained result is a set of segmented windows. Whatever is the segmentation method used, we need a word recognizer, the role of which is to find the "true" sequence of characters, generally relying on character recognizer(s)/model(s).

In the case of explicit segmentation (pure segmentation or segmentation/recognition), an ICR or SCR based on classifiers such as Support Vector Machines (SVM) and/or Neural Network (NN) is generally used [38, 78, 3] to recognize each segmented component (grapheme or character). For implicit segmentation, Hidden Markov Models (HMM) are frequently used [82, 107] to create character and/or word models. Alternatively, the use of Time Delay Neural Networks (TDNN) or Multi-States Time Delay Neural Networks (MS-TDNN) can be seen as a hybrid model of NNs and HMMs [65, 21]. In such cases, Viterbi algorithm is generally used to find the most likely words.

2.2.5 Post-processing

As mentioned in section 1.3, the large variation of handwriting, the complexity of connections between characters, the low quality of handwriting, etc. are major causes of ambiguities and recognition errors. Therefore, context integration is often considered in post-processing for improving the system performance, as explained in section 1.4.

In the context of handwriting recognition, external knowledge is usually used for this post-processing stage [96, 106]. Some methods are commonly used in the literature for representing this knowledge: lexicon-based, n-gram character based, and n-gram word based approaches.

2.2.5.1 Lexicon-based post-processing

In the context of handwriting recognition, a lexicon is usually used for post-processing. The lexicon allows to limit the recognition context and search space. In such a context, recognized words must belong to the lexicon. Different approaches can be found in the literature [4]:

- Verification or validation: every words considered as a possible output must belong to the lexicon.
- Correction: the system computes an edit distance [101] between each recognized word and those in the lexicon. Words having the best matching are considered as the final result of the system.
- Prediction: lexicon is directly integrated during the recognition process. It is used as a predictor, in order to predict possible characters (*i.e.* limit the search space) during the decision.

In order to reduce the complexity of system, TRee Information rEtrieval model (TRIE) may be used. This model is also known as a *prefix tree* in which words having the same prefix share the same branch of the tree. Figure 2.4 illustrates a TRIE model of a given lexicon which contains five words: who, whom, what, when and where.

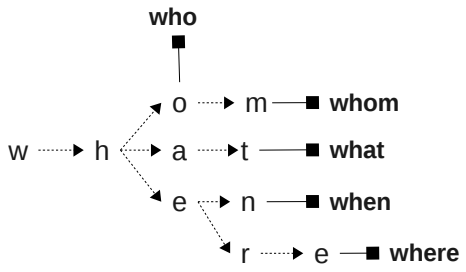


Figure 2.4: TRIE model of the lexicon which contains five words: who, whom, what, when and where.

2.2.5.2 N-gram model at the character level

N-gram model is used at the character level to describe the *a priori* probability of a sequence of N characters. For example, in the case of a bi-gram (2 characters), the character sequence "*qu*" is more frequently used (*i.e.* more probable) than "*qo*". These probabilities can be used as external knowledge to recognize a given signal.

A N-gram model can be applied for both lexicon-based and lexicon-free recognition contexts. A. Brakensiek et *al* [15, 14] applied this kind of strategies for degraded documents and off-line HWR (4 writers only) in a lexicon-free context. The experimental results show the significant improvements brought by the N-gram model. However, the results remain inferior to lexicon-based recognition.

2.2.5.3 Statistical language models

Statistical language models have been largely and successfully used for different applications such as speech recognition, machine translation, spelling correction and handwriting text recognition. Many different methods exist to build language models. In the case of handwriting text recognition, two methods are usually used in the literature: N-gram and N-class model, applied in the context of text/line recognition.

- N-gram model at the word level: similar to the N-gram model at character level, N-gram model at the word level aims at describing the *a posteriori* probability of a sequence of N words
- N-class model: is also known as part of speech model. Each word is classified into its part of speech class (*noun*, *pronoun*, etc.). Similar to N-gram model, the N-class model aims at describing the *a posteriori* probability of a sequence of N classes.

According to different experimental results in the literature [151, 143, 116, 109], the use of statistical language model shows a clear improvement of the effectiveness of the systems. However, the statistical language models can be applied only for text-line recognition problems.

2.2.6 Conclusion

This section introduces an overview of analytical approaches based systems. Each step of an analytical approach has been presented: pre-processing, segmentation, feature extraction, word recognition and post-processing. A discussion and conclusion for each of these steps were given in each sub-section.

Even if we can categorize the approaches according to the analytical/global criterion, it is not sufficient to exhaustively describe the existing systems in the literature. As a consequence, it is necessary to give an overview of some existing systems, in order to understand their functionalities in details. This is the objective of the following section.

2.3 Existing handwriting recognition systems

In the previous section, we presented an overview of the handwriting recognition systems based on analytical approaches. In this section, we present in details the existing recognition systems in the literature. We classify these systems according to their segmentation method: implicit segmentation or explicit segmentation method.

2.3.1 Implicit segmentation based systems

Implicit segmentation is very commonly used for both off-line and on-line data. The main differences are related to the way of applying the sliding window.

- Concerning off-line data, the sliding window is generally moved on the input image along the writing direction. For example, in the case of English language, the sliding window is moved from left-to-right while it is moved from right-to-left for Arabic language.
- Concerning on-line data, the sliding window is generally moved along the writing trajectory.

In both cases (on-line and off-line data), the parameters of the window (window size and overlapping size) are generally *a priori* settled to a pre-fixed value. This value may be fixed manually or by using a validation set. Features extracted from each window are further used to feed a word recognizer, mostly based on HMM models or some kind of hybrid HMM/NN. Since the features extracted from the sliding window are directly used to feed the word recognizer without any explicit segmentation of the input signal into graphemes or characters, these systems are considered as implicit segmentation based.

We propose to classify these systems into two groups: classical HMM-based systems and hybrid HMM/NN-based systems. In this section, we only focus on the functionality of the systems. The recognition results will be illustrated and discussed in section 2.3.3.

2.3.1.1 Classical HMM-based systems

A Hidden Markov Model (HMM) is a model that describes the transitions between a set of states $S = \{s_1, s_2, \dots\}$ for a given set of observations $O = \{o_1, o_2, \dots\}$. The terms "hidden" refers to the fact that there is a set of states S for each observation o_k. A HMM is denoted as $\lambda = \{A, B, \pi\}$ where:

- A: is a matrix of a_{ij} which indicates the transition probabilities from the state s_i to s_j
- B: is a matrix of $b_i(o_k)$ which indicates the emission probability of an observation $o_k \in O$ associated with a state $s_i \in S$
- π: is a vector containing the starting probabilities for each state s_i

In the handwritten word recognition problem, each input handwritten word contains a set of observations, where each observation is composed of the features extracted from a window in the sliding sequence. For every word w_l in the lexicon, a HMM model λ_l is created by a training process. The following paragraph introduces how the HMM model λ_l for each word w_l is trained. Then we will explain how an input handwritten word can be recognized. Finally, two classical HMM-based systems are introduced.

a) Training:

Handwritten recognition systems (for both off-line and on-line data) are generally based on linear or Bakis architecture [109] (see Figure 2.5). For linear architecture, each state is connected to itself and its immediate successor state [60, 88, 8]. The Bakis architecture may be seen as an extension of the linear architecture. The Bakis architecture allows skipping the immediate successor state. Therefore, each state has three connections as illustrated in Figure 2.5. Some existing systems can be cited [110, 10]. They will be introduced after.

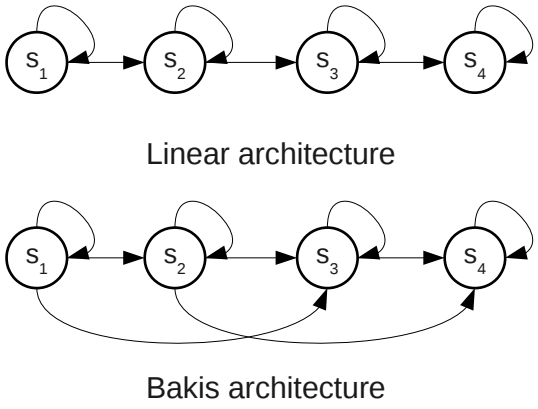


Figure 2.5: HMM linear and Bakis architecture for a 4 states model

A general view of HMM training process is illustrated in Figure 2.6. An initial HMM model (λ_l^0) for each word w_l must be manually initialized before the training process, where the number of states and the number of Gaussians in the mixture of each model are generally fixed using heuristics.

Considering that we have a training dataset consisting in multiple samples of the word w_l where each handwritten word is represented by a sequence of observations O. The training process may take several iterations to update the

parameters in the model. The model λ_l^n constructed at each iteration n must increase the likelihood: $P(O|\lambda_l^n) \geq P(O|\lambda_l^{n-1})$. Based on the initial model λ_l^0, in the first iteration ($n = 1$), the training process aims at re-estimating a new model λ_l^1 that must verify $P(O|\lambda_l^1) \geq P(O|\lambda^0)$, and so on. During the training stage, the model parameters are re-estimated, but neither the model architecture nor the number of states are changed. As an output of the training stage, we obtain one model λ_l for each word w_l in the lexicon.

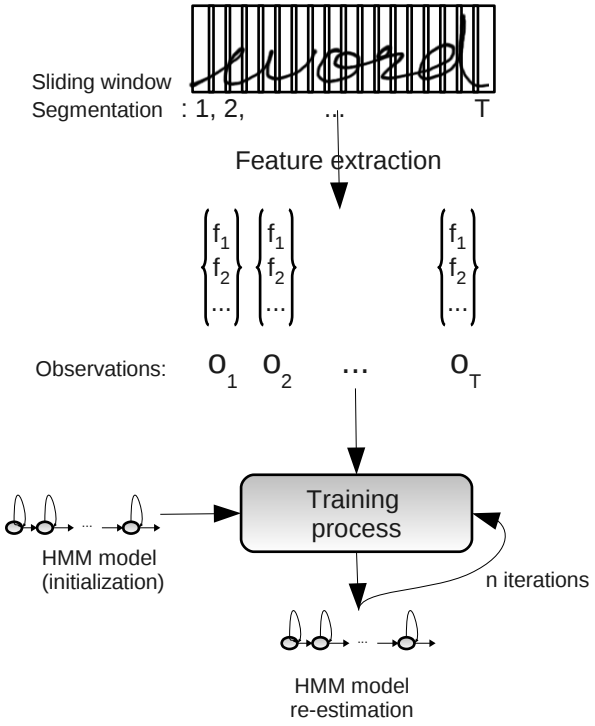


Figure 2.6: An overview of the HMM training process.

Baum-Welch algorithm is usually used to estimate the model. The mixture of Gaussians method is used to compute the emission probability (b_i) of an observation ($o_k \in O$) associated with the state $s_i \in S$ denoted by Equation (2.1).

$$b_i(o_k) \;=\; p(o_k|s_i) \;=\; \sum_{m=1}^{G} c_{im} N(\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im}) \tag{2.1}$$

where G and c_{im} are respectively the number of mixture components (*i.e.* Probability Density Functions (PDFs)) and the weight of the m^{th} component

for the state s_i. $N(\boldsymbol{\mu}_{im}, \boldsymbol{\Sigma}_{im})$ is multivariate Gaussian where $\boldsymbol{\mu}_{im}$ and $\boldsymbol{\Sigma}_{im}$ respectively correspond to the mean vector and covariance matrix of the m^{th} component for the state s_i.

b) Recognition:

During the recognition process, the Viterbi algorithm is generally used to select the most probable sequence of states, given the sequence of observations O. For a given lexicon, we compute the likelihood of each word w_l as $LH(w_l) = P(O|\lambda_l)$, where λ_l is the HMM model of the word w_l. Maximum likelihood strategy is used to select the most likely word for a sequence of observations: $w_l^* = \arg\max_{\lambda_l} P(O|\lambda_l)$, except when linguistic information is integrated in the process.

c) Existing systems:

Hu *et al.* [60] proposed an on-line HWR based on HMM. A vector of seven features is extracted from each point of the signal (which can be considered as a basic sliding window strategy) after normalization and pre-processing. This feature vector contains four on-line local features (tangent slope angle, normalized vertical coordinate, normalized curvature, and ratio of tangents) and three on-line global features (cusps, crossings, and loops). This system relies on a HMM linear architecture with three levels: stroke models, character models, and grammar network. Stroke models are used as the basic models of the system. In a first stage, they are trained on isolated characters; before being secondly trained on whole word samples (without any knowledge about character boundaries), which are then further used to create character models. These character models are then used to create word models of each word in the lexicon.

In this system, each character class is represented by only one HMM character model. One strong limit of such approach is that they do not integrate the "context" of the handwriting, and more specifically the surrounding characters (left and right). Indeed, shapes of one character class may change according to their surrounding characters. Therefore, using only one HMM character model to represent each character class may not be enough to handle this context modeling question.

In order to solve this problem, A.L. Bianne *et al.* proposed a context-dependent system for off-line handwriting recognition [9]. This system relies on HMM models using Bakis architecture. As for every implicit segmentation based system, a sliding window method is applied. For each win-

dow, a vector of features is extracted. This vector contains pixel densities, background/foreground transition, stroke concavities, etc. Character models are trained using a dataset containing whole words without any preliminary character segmentation (*i.e.* word level training). The main difference between this system and other classical HMM-based systems is that, for a given character class, they create a set of character models with different writing contexts. This method is known as *trigraph* method and is inspired from the earlier work presented by A. Kosmala *et al.* [83] for on-line handwriting recognition systems.

Although the classical HMM model is widely used for HWR, it still has several drawbacks. One of these drawbacks is related to the fact that each state is trained without taking into account the other states. Each emission probability is computed by using a Mixture of Gaussians. As a consequence, states are not discriminatively trained. Another well known drawback is related to the fact that the number of Gaussians must be optimally fixed by using some heuristics or optimization strategies [48].

In order to create a HMM model with discriminative training, hybrid systems which combine HMM and Neural Networks (HMM/NN) have been proposed. This hybrid HMM/NN system generally allows replacing the Mixture of Gaussians by Neural Network which is a famous classifier for its discriminative classifical power. We propose an overview of these systems in the following sub-section.

2.3.1.2 Hybrid HMM/NN based system

In the case of hybrid HMM/NN model, the emission probability is computed by neural networks, where all states are discriminated. So, the non-discriminant training problem of the classical HMM model as well as the problem of choosing the optimal number of Gaussians in the mixture can be circumvented.

In such hybrid methods, different types of Neural Networks (NN) are used in the literature: Multi-Layer Perceptron (MLP), Time-Delay Neural Network (TDNN) and Recurrent Neural Network (RNN). Due to the different architectures of NN, different hybrid methods and systems can be found in the literature. In the following paragraphs, we introduce these hybrid methods separately according to the type of NN: HMM/MLP, HMM/TDNN and HMM/RNN.

Hybrid HMM/MLP:

Hybrid HMM/MLP methods have been used early in the literature. Their general principle is illustrated in Figure 2.7. First, a sliding window is passed through the input handwritten word. Features extracted from each window (o_k) are then used to feed the input layer of a MLP. The MLP returns a vector of the *a posteriori* probabilities $p(s_i/o_k)$ which can be converted into emission probabilities $p(o_k/s_i)$ using Bayes' theorem. The

vector of the emission probabilities is then directly integrated in a HMM representation. Recognition process is performed generally by using the Viterbi algorithm as in the case of classical HMM-based system.

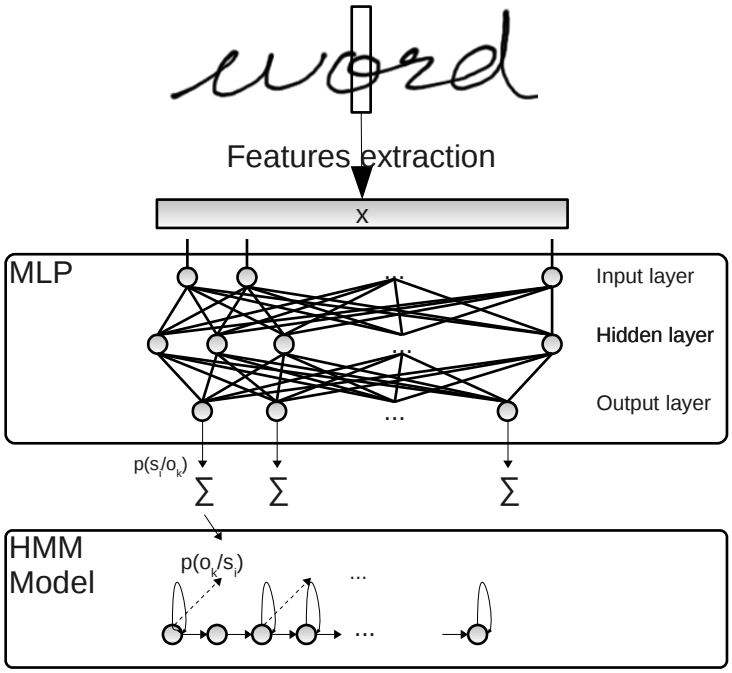


Figure 2.7: General view of hybrid HMM/MLP by considering that the MLP has three layers.

Some other hybrid systems were presented in the-state-of-the-art [39, 97, 124, 33]. In the next paragraphs, we present two different systems introduced by S.España-Boquera *et al.* [33] and J.Schenk and G.Rigoll [124].

S.España-Boquera *et al.* [33] presented a hybrid off-line handwritten text recognition system using hybrid HMM/MLP. The input text line is first normalized by applying several normalization methods: (i) image cleaning, (ii) slope removing, (iii) slant removing, and (iv) size normalizing. The sliding window is used to parse the normalized text line. In each window, a set of 60 features is extracted. In order to use the context of the past and future data, for each window, the authors take into account the features of the 4 preceding windows and 4 succeeding windows. Therefore, in total, a vector of 60*9 features (o_k) is obtained for each sliding window to feed a MLP. The emission probability $p(o_k|s_i)$ is computed by applying the Bayes' theorem.

This hybrid method is the most frequently used in the literature. Although it significantly improves the recognition rates compared to the classical HMM-based system, it still has some disadvantages. Indeed, as explained in [33], the number of units in the output layer of the MLP equals the number of states per character model multiplied by the number of character models. In the case of context-dependent systems, the number of models is significantly increased, which leads to a very high number of units in the output layer. As a consequence, the number of parameters to be estimated by the MLP dramatically increases. In order to train such a large MLP, a huge size of training data is needed. In addition, the training complexity becomes a problem [119].

In order to avoid this kind of problems, J.Schenk and G.Rigoll [124] introduced a hybrid HMM/MLP handwriting recognition system using two different hybrid strategies: "Tied Posteriori" and "Tandem", applied to the on-line handwriting recognition. These strategies were originally presented in [119, 52] for speech recognition. For each point of the signal, they extract 5 on-line local features and 9 off-line local features from the recovered off-line image. Similarly to the system presented in [33], for each point, they add features extracted from preceding and following points to feed the input layer of the MLP. The MLP returns *a posteriori* probabilities ($p(q_j|o_k)$), where the q_j are directly the symbol classes to recognize.

In the case of Tied Posteriori hybrid technique, the outputs of MLP ($p(q_j|o_k)$) are used to replace the mixture of Gaussians traditionally used in the classical HMM. The emission probability of an observation o_k associated to a state s_i in the Equation (2.1) can be denoted by:

$$b_i(o_k) \;=\; p(o_k|s_i) \;=\; \sum_{j=1}^{G} c_{ij}.\frac{p(q_j|o_k)}{p(q_j)} \tag{2.2}$$

where G is the number of symbol classes defined by user. The parameter G corresponds to the number of mixture component in the case of mixture of Gaussians. The parameter c_{ij} is the weight of the j^{th} component for the state s_i.

In the case of Tandem hybrid technique, NN is used as a part of features extraction called Tandem features. The output of NN is converted by applying logarithm:

$$l_i(o_k) = log(p(q_i|o_k)) - \frac{1}{G}\sum_{j=1}^{G} log(p(q_j|o_k)) \quad (2.3)$$

The PCA is then applied on this new set of features before using these features as an input of the classical HMM. Then, the classical HMM training process is performed in order to create a HMM model for each word w_l in the lexicon (see section 2.3.1.1).

The experimental results of both systems ([124, 33]) show significant improvements compared to the classical HMM-based systems. According to [124], the Tandem hybrid technique provides better recognition rates(+1.8%) compared to Tied Posteriori technique. Unfortunately, there is no relevant comparison between these hybrid techniques and the standard hybrid HMM/MLP technique. However, in the literature, we can observe that the classical hybrid technique seems to be more frequently used.

Hybrid HMM/TDNN

Hybrid HMM/TDNN was initially used for speech recognition systems [148, 56]. TDNN refers to Time Delay Neural Networks. In handwriting recognition, the term TDNN is generally used for on-line (temporaly) data while the SDNN (Spatial Delayed Neural Network) is used for off-line (spatial 2D) data. In this part, we focus only on TDNN, since our problem concerns on-line handwriting recognition.

The hybrid HMM/TDNN technique is also known as Multi-State Time Delay Neural Network (MS-TDNN). Similar to HMM/MLP hybrid technique, it aims at replacing Mixture of Gaussians by TDNN.

Figure 2.8 illustrates a general view of the hybrid HMM/TDNN (MS-TDNN) model. First, local features are extracted from each point in the input signal. These features are directly used to feed the input layer of the TDNN. Then, a sliding window is used to integrate a subset of the cells of the input layer to feed the hidden layer. Each subset unit of a given layer is integrated to feed the next layer, and so on. Finally, the output layer of the TDNN will be modeled by a HMM model.

The difference between MS-TDNN and HMM/MLP is principally linked to the different topologies of TDNN and MLP networks. In MLP, each unit of a given layer is linked to every unit of the next layer. Unlike MLP, the main feature of TDNN is time-shift invariant architecture where a subset of units from a given layer is linked to one unit of

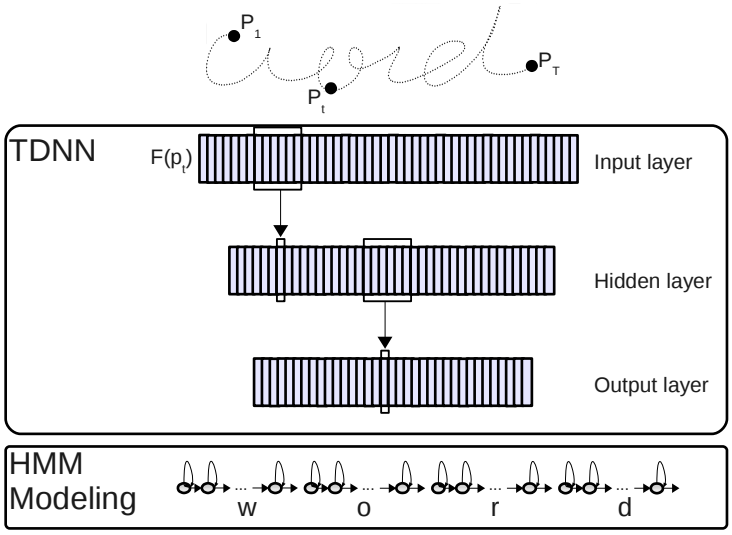


Figure 2.8: Example of hybrid HMM/TDNN (MS-TDNN) with three layers.

the next layer. This architecture allows continuous analysis of data, especially when the inputs are given as a time series. This architecture is very helpful for real time applications such as speech and on-line handwriting recognition systems. It allows recognizing a pattern independently of its position in time.

MS-TDNN based systems may differ from each other according to the way TDNN and HMM are combined and according to the training procedure. In the following paragraphs, we introduce three systems of the literature.

A hybrid HMM/TDNN system was presented by Schenke *et al.* [126] in 1993, initially for on-line hand-printed word recognition system. Then, this approach was applied to on-line cursive word recognition system [125] in 1994. Given an input handwritten word, some pre-processing techniques are applied in order to normalize the information: size normalization, baseline detection, smoothing and resampling. For each point in the normalized signal, a set of 9 on-line local features is extracted. Similar to other implicit segmentation based system, a sliding window segmentation method is used. The size of the sliding window is heuristically estimated on the basis of the average size of a character. The features (o_k) extracted from each sliding window are used to feed a single character TDNN recognition module which returns *a posteriori* probability ($p(c_i/o_k)$) for every character class (c_i). In this system, each state is a character class ($c_i = s_i$). In order to overcome the problem related to character size variation, a HMM model of a character class (c_i) is modeled by several states c_i, called "duration modeling", as illustrated in Figure 2.9.

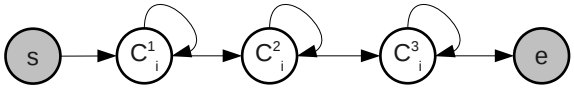


Figure 2.9: A 3 states HMM model of character c_i.

Another similar hybrid HMM/TDNN approach was more recently presented by E.Caillault *et al.* [21] for on-line handwriting recognition. This system can be regarded as a combination of TDNN, MLP and HMM. Unlike the hybrid technique presented in [125], TDNN is not used as a character classifier. In this system, the last hidden layer in the TDNN is used as the input of the MLP. Indeed, TDNN is used as a feature extraction method which projects the features extracted from each sliding window to another feature set, which is considered as more significative (according to author) compared to raw features extracted from each sliding window. The MLP provides estimated probability associated to each character class. Similarly to the system presented in [125], the output of MLP is used to create a HMM model of a character. In this system, the size of the sliding window is heuristically fixed based on the average size of a character.

S. Jaeger *et al.* [65] proposed a hybrid HMM/TDNN system for on-line HWR. First, the input signal is normalized by some standard normalization methods. Then, for each point of the normalized signal, a set of on-line local features and off-line local features is computed to feed the TDNN. This TDNN is composed of two hidden layers. The second hidden layer is considered as a state layer. The state layer is used to create HMM state models each of which is composed of one or more units. Each character class is then modeled by three state models each of which respectively represents first, middle and last part of the character class. These character models are further used to create word models by concatenating character models.

In the hybrid systems presented by Schenke *et al.* [125] and E.Caillault *et al.* [21], the HMM model of a given character class (c_i) is represented by a sequence of states, where each state is associated to a character class c_i. This kind of methods has two main drawbacks. First, TDNN is simply used as a character classifier and the average size of a character (*i.e.* size of sliding window) must be optimally fixed, what is difficult. Second, every HMM character model is represented with the same duration (*i.e.* the same number of states), which is not representative of the reality. These parameters are difficult and almost impossible to tune, since the character sizes differ a lot. For instance, the size of the character 'i' is much smaller than the one of the character 'w'.

In [65], on the other hand, each state of HMM is associated with a sub-part of the character (first, middle and last) instead of the whole character. As mentioned earlier, each HMM character model is modeled by three HMM state models. The length of each state model is optimally fixed during the training process. Therefore, each character class is modeled with an adaptive HMM length. It is an important advantage compared to the previously presented classical HMM-based systems, hybrid HMM/NN based systems and systems presented in [125, 21] for which the length of each character model was preliminary fixed by using a heuristic estimation or validation set.

Hybrid HMM/RNN

Recurrent Neural Network (RNN) is another type of neural networks. Similar to MLP, each unit of the input layer is linked to every unit of the hidden layer. RNN contains only one hidden layer, but allows self-connection on this layer. This self-connection is the main difference with MLP. This mechanism allows the network to use contextual information by integrating past data. In the context of handwriting recognition, the past data is important since the shape of one character is changed according to its surrounding characters (see section 1.3.1). This is a strong advantage compared the hybrid HMM/MLP and the hybrid HMM/TDNN.

A.Graves *et al.* [43] presented a RNN based system which is specifically designed for using the advantages of the RNN. Even though, the author did not present this system as a hybrid of HMM/RNN, it can be seen in this way. A global view of the system presented by A.Graves *et al.* [43] is illustrated in Figure 2.10. For each point, a set of features is extracted. We will not explained in details the normalization and feature extraction steps since they are very similar to the systems presented above. However, we will provide some details concerning the hybrid part.

The extracted features (in time series $t \in \{1, 2, \ldots, T\}$) are used to feed a Bidirectional RNN with Long Short-Term Memory (BLSTM) architecture. In addition, Connectionist Temporal Classification (CTC) is also used. The CTC aims at providing the additional "blank" label (non-character class ε) between two characters on the output layer. At each time t, the BLSTM returns the recognition probability of each character class including the "blank" label. As a result, we get a matrix of $(M + 1)$ x T, where M is the number of character classes, 1 corresponds to the additional blank label and T is the length of time series. Finally, a Token Passing Algorithm is used for word recognition process in order to compute the probability of each word in the given lexicon.

This system provides very promising results with a great improvement of the recognition rate ($\sim$ 12% with a lexicon of 20 000 words) compared to classical HMM-based sys-

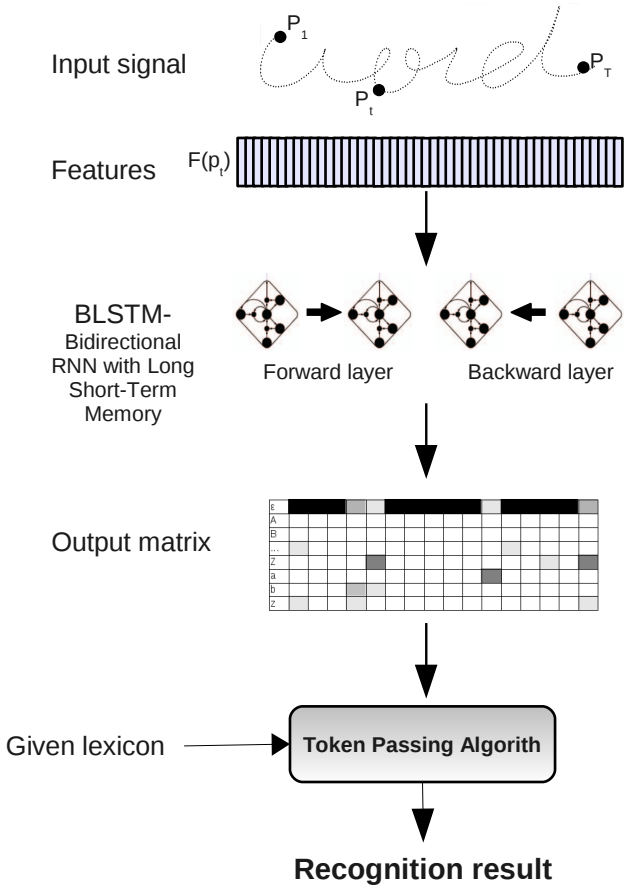


Figure 2.10: The overview of the system presented in [43].

tem. Unfortunately, there is no objective comparison with HMM/MLP or HMM/TDNN based systems. We can notice different interesting ideas in this system. First, the use of Bidirectional RNN allows modeling context in both directions (on the left and the right of the current character). It permits to integrate the variation of character shapes, which strongly depends on left and right neighboring characters. Second, the modeling of the non-character class is another important aspect, which is not considered by more traditional system.

2.3.1.3 Conclusion

We have introduced different handwriting recognition systems that use implicit segmentation strategy such as classical HMM and hybrid HMM/NN based systems. These methods are very frequently used in the literature, and usually give very competitive results. However, they require a huge size of training dataset, especially when using hybrid models [124, 109]. Furthermore, many parameters need to be heuristically estimated.

Another segmentation method called "segmentation/recognition" aims at integrating a recognizer into the segmentation process, in order to avoid segmentation errors. Therefore, this kind of methods shares the same advantage with the implicit segmentation in the sense that the input word is not explicitly segmented into a sequence of characters directly. This kind of systems is presented in the next section.

2.3.2 Explicit segmentation based systems

We classify explicit segmentation based systems into two classes: pure explicit segmentation and segmentation/recognition based systems. Different segmentation methods are presented in the literature: water reservoir concept [104], connected component analysis [95], transformation-based learning (TBL) [17, 32], background and foreground analysis [28] ... A good survey on these segmentation methods are presented in [24, 118].

The explicit segmentation based systems generally suffer of two problems: under-segmentation and over-segmentation.

- *Under-segmentation*: refers to the problem where some segments may not correspond to a single character, because of connections with their neighboring character(s) or some part(s) of their neighboring character(s).
- *Over-segmentation*: at the opposite of the under-segmentation problem, it refers to the problem that one character shape may be broken into two or more segments.

For solving these two problems, heuristic rules and/or verification scheme(s) may be applied, in order to remove noisy segments and/or to group the broken segments. Systems based on pure explicit segmentation specifically require very effective segmentation methods, because errors occurring during segmentation step generally have a direct effect on the overall system. Concerning segmentation/recognition methods, these errors may be fixed during the recognition step.

In this section, we only focus on the functionality of the systems. The recognition results will be given and discussed in section 2.3.3.

2.3.2.1 Pure explicit segmentation methods

Pure explicit segmentation based systems aim at segmenting the input signal into sequences of single characters. Then, a SCR is used to recognize each of them. A global overview of this kind of systems is illustrated in Figure 2.11.

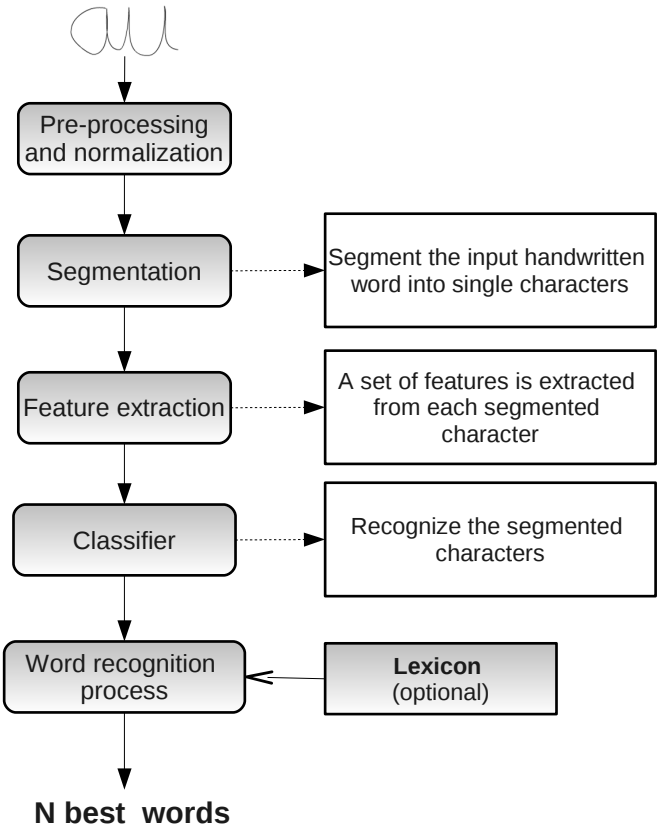


Figure 2.11: The global overview of pure explicit segmentation based systems.

The system presented by S. Ouchtati *et al* [121] focuses on handwritten numeric string recognition. In this system, they do not take into account the problem concerning touching and overlapping digits. A simple segmentation method based on vertical projection is used. The segmented digits are directly submitted to a digit recognition system that relies on MLP. If digits in the string are touching or overlapping, the system faces the under-segmentation problem because touching and overlapping digits cannot be separated by the segmentation method relying on vertical projection. In order to overcome these problems, many methods dedicated to the segmentation of touching and overlapping digits/characters are presented [17, 24, 32, 28, 118]. However, these approaches generally rely on heuristics and it is quite difficult to fix threshold value(s) and heuristic rule(s) to get perfect segments.

To overcome this problem, Dan Ciresan [30] presented an off-line numeral string recognition system by combining single and two-digit recognition sys-

tems. The input signal is segmented by using connected component analysis. Then, some rules are applied to remove noisy components. For example, *components that contain less than four pixels are deleted if their closest component size is more than five pixels.* The remaining connected components are recognized by single and two-digits recognition systems. Both systems use Convolutional Neural Network (CNN) as classifiers. For each component, in order to select the best recognition label given by these two systems, author experiments the system with 3 different methods:

- Method 1: they compare $score1$ and $score2$ which respectively correspond to the recognition probabilities on Top-1 given by the single and the two-digits recognition systems. If $score1 \geq score2$, then Top-1 given by the first system is selected, Top-1 given by second system otherwise.
- Method 2: for each system (single and two-digits), the difference between the recognition score of Top-1 and Top-2 is computed. They obtain dif_1 for the first system (single character) and dif_2 for the second system (two-digits). If $dif_1 \geq dif_2$, then Top-1 given by the first system is selected, Top-1 given by the second system otherwise.
- Method 3: they combine rules presented in methods 1 and 2. If $score1 + dif_1 \geq score2 + dif_2$, then Top-1 given by the first system will be selected and Top-1 given by the second system otherwise.

Experimental results show a great improvement by combining single and two-digit recognition systems with Method 3. However, this system still faces three main problems: 1) if the connected components contain more than two digits, 2) if the connected components contain some part(s) of their neighboring digit(s), 3) if the connected components contain only a part of the digit.

The use of pure-explicit segmentation methods does not require the use of any recognition system during the segmentation step. Therefore, most of the time, errors caused by the segmentation method cannot be corrected. On the other hand, explicit segmentation/recognition based systems allow to integrate simultaneous segmentation and recognition stages in order to avoid segmentation errors. These systems are presented in the next section.

2.3.2.2 Explicit segmentation/recognition methods

The segmentation/recognition-based systems aims at over-segmenting the input signal into graphemes and use a supervised classifier to discover the best groups of graphemes as character candidates.

A global overview of this kind of systems is illustrated in Fig 2.12. First, the input signal is normalized and segmented into a set of graphemes. These graphemes are further used to create a lattice of L levels. Then, from each node of this lattice, a

set of features is extracted in order to feed a discriminative classifier. This classifier returns an *a posteriori* probability for each character class. Finally, during the word recognition process, dynamic programming may be used to compute the probability of every word in the lexicon.Different systems were presented in the literature. In the following paragraphs, we present three different systems.

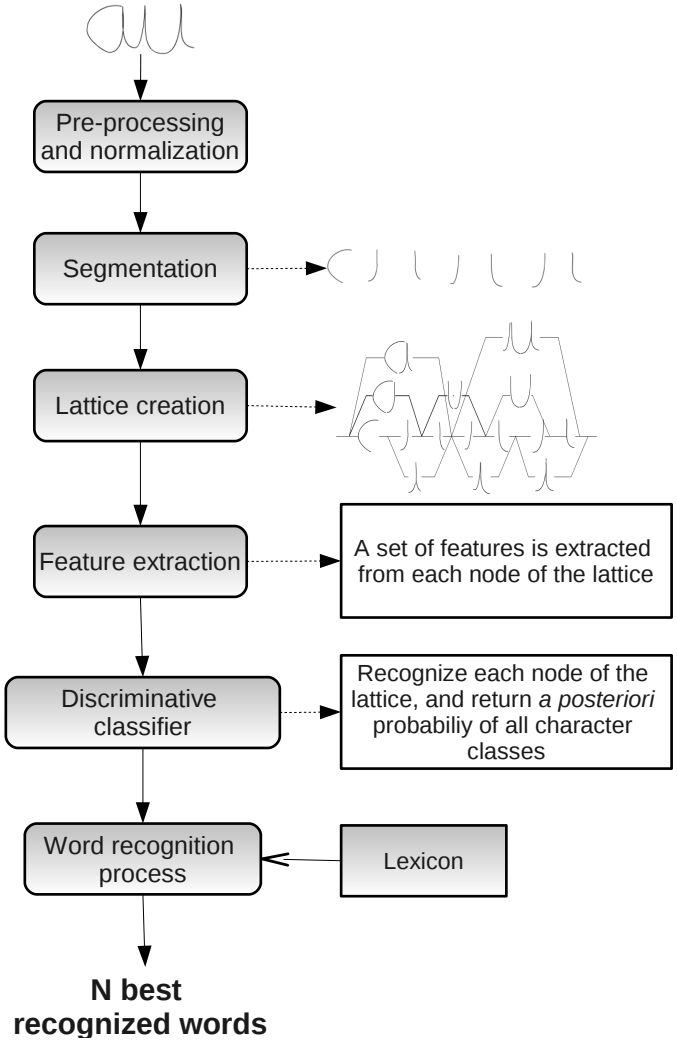


Figure 2.12: Overview of explicit segmentation/recognition based systems.

E. Kavallieratou *et al.* [71] presents an unconstrained off-line handwritten text recognition system in the context of lexicon-free applications. Several normalization and segmentation methods (pre-processing, printed-handwritten text separation, line segmentation, slant correction, and word segmentation) are sequentially applied to segment the input handwritten text into isolated words. The segmented words are submitted to an isolated word recognition system. This system relies on an explicit segmentation/recognition.

Each isolated word is segmented into graphemes using Transformation-Based Learning (TBL) [32, 17]. Then, every grapheme and every pair of graphemes (i.e two-level lattice) are considered as a potential character. For each potential character, a set of features ($\boldsymbol{x}$) is extracted. This set contains features such as: histogram, profile, radial histogram and radial profile. Then, an Euclidean distance is used to compute the distance between the extracted features $\boldsymbol{x}$ and the prototype(s) features $\boldsymbol{x}_c$ of each class. The character class with the shortest distance is considered as the recognized class. Two drawbacks can be identified for this system. The first drawback is related to the segmentation module which is based on the TBL method. This method relies on the machine learning theory in order to learn the segmentation points between characters. However, it may face two problems: 1) TBL requires annotating training data, which is a time-consuming task and 2) because of the limitations of the algorithm or/and the lack of training data, this system may face with segmentation problems. The problem of under segmentation has not been envisaged to be solved. The second drawback concerns the character recognition module which relies on a simple Euclidean distance. This method may not be effective enough to handle the variation problem in SCR. Moreover, it is time consuming recognition strategy. In this context, discriminative classifiers such as SVM or NN are more adapted and more frequently used in the literature.

On the opposite, a system presented by G. Koch *et al.* [80] uses a simple segmentation method which relies on local extreme points of word contours. A discriminant classifier MLP is used to create a SCR.

Similarly to this method, in a system presented by Ahmad *et al.* [3] for on-line handwritten word recognition problem, the segmentation method is based on local extreme points. A grapheme is a group of points located between a local minimum and a local maximum points (on the y axis). A SVM classifier is used as a character recognition module. The outputs of the SVM are then used to feed a HMM model. Finally, the Viterbi algorithm is used to find the best alignment of all words in the lexicon.

2.3.2.3 Conclusion

In this part, we presented the explicit segmentation-based systems. Two main types of methods are presented: pure explicit segmentation and explicit segmentation/recognition based systems.

Systems using pure explicit segmentation require a very powerful and stable segmentation method, since segmentation errors have a direct effect on the overall system. The

main advantage of such methods is the computation time, since the segments given by the segmentation step are directly considered as segmented characters.

On the other hand, for explicit segmentation/recognition system, only a simple over-segmentation method is required. The hybridization of the character recognition system and the segmentation method allows to overcome the segmentation errors. The main drawback of this method is the computation complexity, since a lattice of segmented graphemes has to be created and every node of the lattice has to be recognized by a SCR. Finally, a word decoding method is required in order to explore the lattice to find the most probable sequences of characters.

2.3.3 Discussion and performance comparison

This section presents different types of handwriting recognition system along with their advantages and drawbacks. Table 2.1 shows a list of different systems to summarize this section.

Table 2.1: List of existing systems presented in the state-of-the-art.

Approaches	**Author**	**Nature of data**	**Method**	**Dataset**	**Lexicon**	**R.rate (%)**
Implicit - HMM-based	Hu *et al.* [60]	words	HMM Linear architecture	Unipen, On-line	20 000	76.3
	A.L. Bianne *et al.* [9]	words	HMM Bakis.	Rime, off-line	10 500	67.5 , 74.1
Implicit - HMM/NN-based	J.S and G.R [124]	words	HMM/NN	German database, on-line	2000	84.2 to 95.9
	S.España-Boquera *et al.* [33]	text line	HMM/NN	IAM, off-line	20 000	83.20
	S. Jaeger *et al.* [65]	words	MS-TDNN	UKA, CMU, MIT, on-line	20 000	91 to 97.5
	É. Caillault *et al.* [21]	words	HMM/ TDNN/ MLP	ironoff on-line	197	92
	A.Graves *et al.* [43]	text line	BLSTM, CTC	IAM, on-line	20 000	81.5

Continued on next page

Table 2.1 – continued from previous page

Approaches	Author	Nature of data	Method	Dataset	Lexicon	R.rate (%)
Pure explicit segmentation	S.Ouchtati *et al.* [121]	Numeral string	NN	-	-	-[1]
	D.Ciresan [30]	Numeral string	CNN	NIST SD19 (off-line)	free	93.49
Explicit segmentation/ recognition	A.R. Ahmad *et al.* [3]	words	HMM/ SVM	ironoff on-line	197	64.53
	E. Kavallieratou [71]	text line	Euclidean	NTIS, IAM, GRUHD, off-line	free	65.6 to 82.79[2]
	G.Koch *at al.* [80]	words	MLP	off-line[3]	1000	67.80

According to the recognition results shown in Table 2.1, we can observe that:

- For off-line handwritten word/text, the recognition rates vary from 65% to 83%. The system presented in [33] (relying on hybrid HMM/NN) provides better results for a lexicon of 20000 words. The systems presented in [80] and [9] provide lower recognition rates compared to the system in [33], even if the lexicons are smaller. On the other hand, the system presented in [71] deals with text recognition in the lexicon-free condition. However, the evaluation of the system relies on the character accuracy, not on the words or text line recognition accuracy.
- For on-line data handwritten word/text, the recognition rates vary from 64% to 97%. The system presented in [65] (relying on MS-TDNN) provides better result for a lexicon of 20000 words. The experiments in [21] and [3] were performed with a smaller lexicon (197 words), but taking into account the accented characters.

However, comparing the systems on the basis of their recognition rates is not fair, since authors performed their experiments in different conditions with different lexicons and test databases. Nonetheless, some fair comparison can be extracted from the literature:

- Based on the experiment results in [9], the HMM context-dependent system provides better recognition results (+6.6%) than the classical HMM system.

[1] does not mentioned the recognition result
[2] Evaluation based on character accuracy
[3] Personal database

- J.S and G.R [124] have shown that the hybrid HMM/NN system using "Tied Posteriori" and "Tandem" improves respectively 2.8% and 4.6% of the recognition rate compared to the HMM baseline system.

- A.Graves *et al.* [43] have shown that by combining Bidirectional RNN and CTC methods allows an improvement up to 12.7% compared to the baseline HMM system [88].

According to these comparisons, we can conclude that systems relying on the hybrid HMM/NN provide better results compared to the systems relying on the baseline HMM. Unfortunately, in the literature, as far as we know, there is no a fair comparison between the different types of Neural Networks in the hybrid scheme(HMM/MLP, HMM/TDNN and HMM/RNN).

Based on the publications in the literature, implicit segmentation methods are frequently used for HWR, while explicit segmentation/recognition methods are widely used for numeral string recognition. There is no explanation about this observation, but we can notice two different factors which may influence the analysis: lexicon and number of classes used in each context.

- For handwritten word/text recognition, a lexicon can be predefined and used as recognition context. In the case of numeral string recognition, it is impossible to pre-define any lexicon because the number of possible concatenations of digits is unlimited, except in some special cases such as date or time recognition.

- The number of classes used for numeral string recognition is limited to 10 classes (Arabic numeral string for instance). In the case of word/text recognition, the number of character classes is much higher. In Latin language, the number of classes can vary between 26 (lowercase only) to 72 classes (lowercase + uppercase + some special characters).

2.4 Conclusion

There are two main approaches used for handwriting recognition systems: global approaches and analytical approaches. When using global approaches, the segmentation step is not required. This kind of systems can be implemented easily. However, this kind of systems can be applied only for fixed and small lexicons. Furthermore, these approaches cannot be applied for numeral string recognition because of the fact that concatenation of digits is unlimited. Except in some cases, for instance the date-time recognition problem, since the digit strings to be recognized can be pre-defined. On the other hand, systems using analytical approaches are convenient for large and flexible lexicon. Therefore, these approaches are very commonly used. A segmentation method is however required. In the literature, we classify them into two categories: implicit segmentation and explicit segmentation. The combination of the implicit segmentation

method and HMM model is very frequently used, partly due to its success in speech recognition systems.

Explicit segmentation methods are classified into two sub-categories: pure-explicit segmentation and segmentation/recognition method. By using the pure explicit segmentation method, if there is any error occurring during the segmentation step, in most cases, the systems are not able to recognize the input data correctly. On the contrary, when using segmentation/recognition methods, the error(s) occurring during segmentation may be solved by the recognition system.

Comparison between implicit segmentation and explicit segmentation/recognition based systems is a major question in the research domain. Fair comparison based on experimental results is almost impossible. Due to the different architectures of these two methods, features and/or training data used may also be different.

Table 2.2 shows a comparison of the advantages and drawbacks of implicit segmentation and explicit segmentation/recognition-based systems. We notice that each system has its own architecture and may have its own advantages and drawbacks. This table describes the comparison on a global view of these approaches. Discussions on each specify system can be found in section 2.3.

Table 2.2: Comparing of advantages and drawbacks between implicit segmentation and explicit segmentation/recognition-based systems.

System	**Advantages and Drawbacks**
HMM-based systems (implicit)	*Advantages:* • Do not require complex segmentation method. The sliding windows method is generally used. • HMM is a very powerful model for modeling temporal information.
	Continued on next page

Table 2.2 – continued from previous page

System	Advantages and Drawbacks
	Drawbacks: • Parameters of sliding window have to be pre-defined manually, heuristically or by using a validation set. • (*) The number of Gaussians in the mixture has to be fixed with a validation set. • These systems strongly depend on normalization methods, especially, in the case of on-line data where delayed strokes generate additional problems as, explained in section 2.2.1. • (**) HMM is a generative model that is less effective compared to discriminative models [43] • Features are extracted from each sliding window. Therefore, only local features can be used. These systems cannot take the advantages of global features at the character level. • Handwritten words of a lexicon ($L_{training}$) are used to train character models, in general, without annotation of character boundaries. Hence, it is a blind-training strategy in the sense that the exact shapes of the characters are not known. In addition, these character models may be over-fitted to the words that belong to the lexicon $L_{training}$. In order to get a standard character model which can be used to recognize any given handwritten words, a very large training database is required.
HMM/NN-based systems (implicit)	*Advantages:* • Has the same advantage as HMM-based system. • Allows using a discriminative classifier at state and/or character level. Therefore, the system is discriminately trained.

Continued on next page

Table 2.2 – continued from previous page

System	Advantages and Drawbacks
	Drawbacks: • Shares the same drawbacks as classical the HMM-based systems, except points (*) and (**). • The parameters of Neural Networks have to be fixed. In general, they require a large training dataset to obtain an effective system.
Explicit segmentation/ recognition-based systems	*Advantages:* • A simple segmentation method can be used to over-segment the input data into graphemes. Then, a recognition system is used to recognize the sequence of characters and at the same time to solve the over-segmentation issues. • Does not require a huge training dataset. Only a dataset of single characters extracted from handwritten words is required. • Any discriminative classifier can be used to create a SCR. *Drawbacks:* • The SCR must be able to reject invalid graphemes as unknown. • The system complexity can be high, as in the worst case all paths of lattice have to be explored.

3 Handwritten words recognition system based on two levels analysis: character and bi-characters level

This chapter is dedicated to the presentation of the proposed system. The global view of our system is introduced. Each section of this chapter presents the method(s) by explaining in details the choices made at each step when conceiving the system. The conclusion of this chapter summarizes the contributions of this system compared to existing systems in the literature.

Contents

Considering our research objectives mentioned in section 1.5, our work consists in creating an on-line HWR. This HWR must be applied in a context respecting the following features:

- Omni writer
- Large and flexible lexicon
- Capturing devices independence

According to the overview of the existing systems in the literature (see section 2.3), we can observe that implicit segmentation based on Hidden Markov models (HMM) and Viterbi recognition are very frequently used for HWR [60, 88, 10]. For these approaches, input handwritten word is first oversegmented with a sliding window. The segments are then combined to characters and words during the recognition stage with respect to the HMM. Attempts towards discriminative on-line HWR often include neural networks based recognition under the hybrid systems in combination with HMM [33, 43, 65]. As clearly expressed in Table 2.2, using these kinds of methods (HMM or HMM/NN based) lead to important difficulties, especially for setting the parameters of NN and/or HMM.

HMM or hybrid HMM/NN were originally used for speech recognition problems and became popular for handwriting recognition problem. On the contrary, SVM classifier was rarely used for HWR problem, although, it has been widely and successfully used for a large number of applications such as face detection/verification/recognition [64], handwritten character recognition, ... [20]. The experimental results given by some studies [60, 22, 137] have shown the effectiveness of SVM compared to other classifiers (including HMM and TDNN models) for handwritten character recognition.

Our strategy is to implement an on-line HWR based on discriminative classifiers. Our system relies on an explicit segmentation/recognition using SCR and bi-character models. Some main specifications of our system can be drawn.

- Let us mention that, in the literature, the on-line HWRs generally rely on on-line local features (extracted from each point of the input signal) and off-line local features (extracted from each sliding window). The structural/statistical based features and/or invariant moments based features which are successfully used for form recognition are not yet used for on-line HWR. Our system relies on the combination of off-line and on-line features which have been successfully used for form recognition and off-line ICR [54]. Using these two kinds of features (off-line and on-line) allows not only to take profit of their complementarity but also to deal with the problems related to the variation in the writing trajectories (see section 2.2.3.3).
- In the existing systems in the literature, they generally segment the input signal into a sequence of observations using a sliding window segmentation method. Each

observation is analyzed by using Mixture of Gaussians (in the case of HMM) or NN (in the case of hybrid HMM/NN) and directly submits the results to a word decoding process. In our system, we proposed to use two levels of analysis: character and bi-character levels. The analysis at the character level allows finding the possible character candidates, while the analysis at the bi-character level allows taking into account the graphical context of the neighboring characters to put into questions the results given by the character level analysis.

- In the literature, NN is usually used to create HWRs, usually in form of hybrid system with HMM. In our system, motivated by the performance of the SVM, we proposed to use this classifier to create the SCR to be used at the character analysis level.

- In the case of on-line signal, systems need to face the problem related to delayed strokes (see section 2.2.1). In the literature, some authors try to detect the delayed strokes in the input signal and remove them during the pre-processing stage. As a consequence, some important information is lost. In order to deal with the problems related to delayed strokes, we propose a delayed stroke management method which integrates the delayed strokes in the recognition process, what allows the system to consider all the information of handwriting.

The overall presentation of the proposed sysem is given in the following sections.

3.1 Global view of the proposed system

Our research focuses on on-line handwritten word recognition. We consider that the input of the system is the on-line signal of an isolated handwritten word. Therefore, the segmentation of text-line into isolated words is not required. The global view of the proposed system is given in Figure 3.1. First, the input handwritten word is normalized and segmented into a set of graphemes. These segmented graphemes are used to create a lattice of L levels (see section 3.3), where each node (defined as a grapheme or a concatenation of graphemes) is considered as a potential character, and each pair of neighboring nodes is considered as a potential bi-character to be analyzed respectively at the single character and bi-characters levels.

At the character level, a SCR (see section 3.5) is used to emit a list of recognition hypotheses for each node of the lattice (see section 3.3) using a feature vector resulting from the combination of on-line features and artificially generated off-line image based features (from the on-line signal) (see section 3.4). Each node may correspond to any character in the alphabet, or to an unknown pattern (*i.e.* an intermediate information that does not correspond to a character). To deal with this problem, rejection methods are also used (see section 3.5.3).

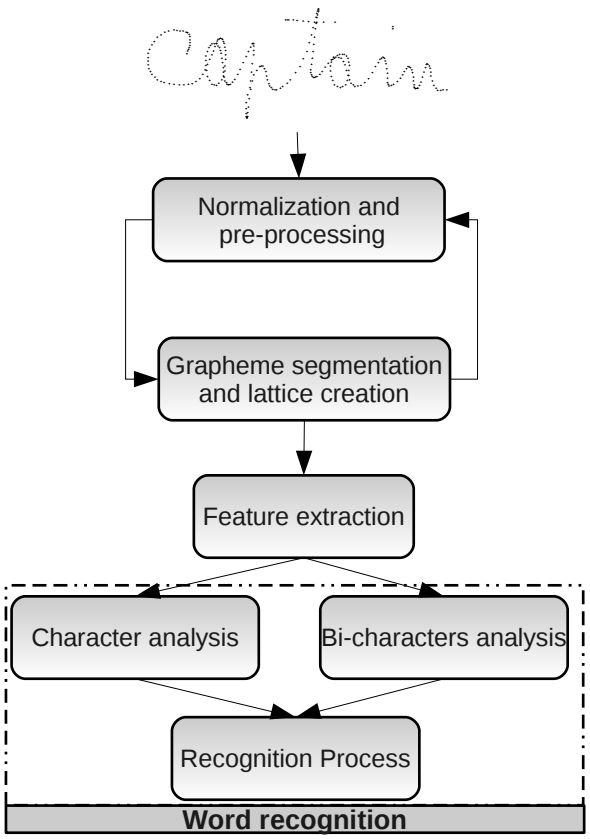


Figure 3.1: Global view of the proposed system.

At the bi-character level, we use bi-characters models so as to validate or invalidate the hypotheses emitted at the single character level, by taking into account two neighboring potential characters (see section 3.6). These bi-character models allow taking into account the graphical context of their surrounding characters, which is very important especially for unconstrained handwriting. Our bi-character models are able to solve different problems:

- Shared character part problem: this kind of problems (see section 1.3.4) occurs when a given character has a visual appearance very similar to parts of other characters. For instance, a character '*o*' may look like the loop in the character '*d*'.
- Unknown pattern: this problem refers to the fact that some nodes of the lattice do not corresponds to any character class considered in the system.
- Similarity between characters: in some cases, the visual appearance of some character classes may be very similar to the visual appearance of some other character classes, especially, when they are recognized individually without taking into account the graphical context of their surrounding characters (see section 1.3.3).

In the last stage of our system, all these sources of information (*i.e.* the results given by the SCR and the bi-characters models) are used in the word decoding process, in order to find the most likely words (see section. 3.7).

3.2 Normalization and pre-processing

On-line signal variations and noise may be due to different factors, such as the characteristic of the capturing device, writing speed, writing context ... These variations have a great impact on the performance of the system, as explained in section 2.2.1. Therefore, pre-processing and normalization methods are required, in order to remove the noise and standardize the input signal. Normalization and pre-processing are carried out at two levels: word and character levels.

3.2.1 Normalization and pre-processing at the word level

Four standard types of pre-processing methods are applied, as illustrated in Figure 3.2: size normalizing, interpolating missing points, smoothing and re-sampling. Moreover, a delayed stroke management method is also proposed, in order to handle the problems caused by this kind of strokes (see section 2.2.1). Since the delayed stroke management method is considered during grapheme segmentation and lattice construction, it will be presented in sections 3.3.1 and 3.3.4.

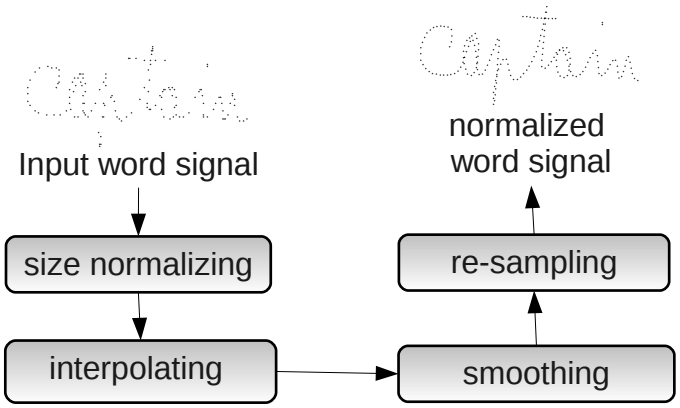


Figure 3.2: Normalization and pre-processing at the word level.

3.2.1.1 Size normalizing

Even with a single writer, one same word may be written in different contexts. The sizes of the handwritten words resulting of these different writing contexts could be different. Furthermore, if they are written using different capturing devices that have different

capturing resolutions, the size of the handwritten words could also vary. Hence, the input signal has to be scaled to a standard size, to ensure that the writings of the same word and character class have approximately the same size.

Before applying any scaling method, the original size of the input word has to be estimated. We have to mention that, in the Latin alphabet, lowercase characters can be classified into four groups according to their position in the writing zone, as illustrated in Figure 3.3. The first group corresponds to the smallest characters ('a', 'c', 'e', 'i', 'm', 'n', 'o', 'r', 's', 'u', 'v', 'w', 'x') which are written only in the middle zone. The second group contains the characters ('b', 'd', 'h', 'k', 'l', 't') which are written in the middle and upper zones. The third group contains the characters ('g', 'j', 'p', 'q', 'y') which are written in the middle and lower zones. Finally, the character 'f' is written in the three zones and is classified into the fourth group. The character 'z' is a special one, it can be classified into the first group or the third group depending on the handwriting style.

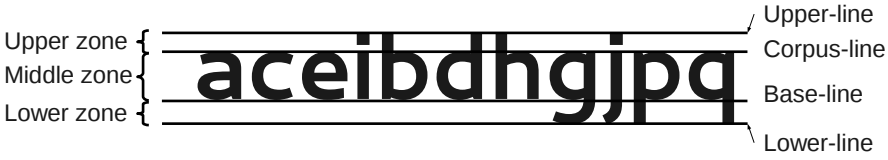


Figure 3.3: An example of the writing positions of different characters.

A word may be composed of characters belonging to one or multiple groups. Therefore, estimating and applying size normalizing based on the global word-height (distance between upper-line and lower-line) may provide non-uniform results between words. For instance, if the writing of the words "*apple*" and "*access*" are normalized to the same size based on their global word-heights, the size of the character '*a*' in the word "*apple*" will be smaller ($\sim$ 3 times) than the size of the character '*a*' in the word "*access*". As a consequence, the variation of these two characters could be very important. For this reason, size normalizing generally relies on the corpus-height of the word which provides the height of the dense zone (*i.e.* middle zone). As illustrated in Figure 3.4, the corpus-height is the distance between the base-line (b) and the corpus-line (c) of the signal.

In some case, these lines (base-line and corpus-line) have to be precisely detected since they will be used as reference for geometrical feature extraction. If these lines are incorrectly detected, unstable features may be generated. In such a context, a method that provides a precise detection such as the Expectation-Maximization algorithm introduced in [7] could be used for instance.

In our case, we only want to estimate the approximate corpus-height of the input handwritten word in order to standardize the size of word signals. Complex method such as Expectation-Maximization algorithm is not necessary because: 1) complexity of the algorithm could be high, 2) it is difficult to set its parameters and 3) this algo-

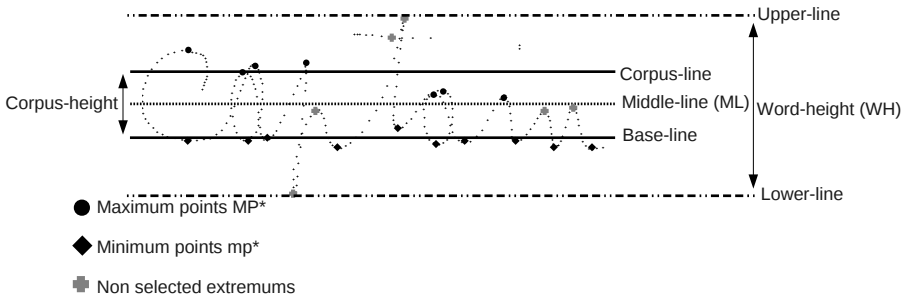


Figure 3.4: An example of corpus-height estimation.

rithm provides the curves of the base-line and the corpus-line that cannot be used to estimated corpus-height directly. To estimate the approximate corpus-height, only the straight lines representing the base-line and the corpus-line are required. For all these reasons, in our system, we propose a simple base-line and corpus-line detection which relies on the average of a set of $|MP|$ local maximum points and the average of a set of $|mp|$ local minimum points on the y axis, as explained hereafter.

Base-line and corpus-line detection: this method is carried out in three steps. During the first step, the word height and middle line are roughly estimated using local maximum points (MP) and local minimum points (mp) on the y axis. At this level, we only need a coarse estimation of these two values (word height and middle line), as it is exhaustively used for removing aberrant local maximum points and local minimum points (outlier points) in a second step. During the third step, in our approach, the baseline and corpus-line are estimated more precisely using only non outlier points. These 3 steps are detailed below:

Step 1: Roughly estimating word-height (WH) and middle-line (ML)

- $WH = \max_{i=1}^{|MP|} MP_i(y) - \min_{j=1}^{|mp|} mp_j(y)$
- $ML = \frac{L_{max} + L_{min}}{2}$,
  where $L_{max} = \frac{1}{|MP|} \sum_{i=1}^{|MP|} MP_i(y)$ and $L_{min} = \frac{1}{|mp|} \sum_{j=1}^{|mp|} mp_j(y)$

Step 2: Detecting and removing outlier points

Some points among the local maximum points (MP_i) and the local minimum points (mp_j) could be outlier points, which could introduce some bias when estimating the corpus-line and the base-line. Hence, they have to be detected and removed using the conditions below:

A given point $P_i \in \{MP \cup mp\}$ is considered as an outlier point if one of these three conditions is validated:

- If the distance between P_i and the middle line ML on the y axis is higher than $\frac{WH}{3}$
- If P_i is a local maximum point located under the middle-line ML
- If P_i is a local minimum point located above the middle-line ML

Step 3: Detecting the base-line (b) and corpus-line (c)

The remaining (non-outlier) local maximum points (MP_i^*) and local minimum points (mp_j^*) are used to compute the coordinates on the y axis of the corpus-line c and the base-line b. These $y-$coordinates are denoted as:

- $c = \frac{1}{|MP^*|} \sum_{i=1}^{|MP^*|} MP_i^*(y)$
- $b = \frac{1}{|mp^*|} \sum_{j=1}^{|mp^*|} mp_j^*(y)$

Estimating the scale factor: once the base-line and the corpus-line are detected, the corpus-height (h_{corpus}) can be estimated as: $h_{corpus} = c - b$. Then, a scale factor (δ_{word}) is used to scale the input signal to the objective corpus-height Δ_{word}. It is computed as follows:

$$\delta_{word} = \frac{\Delta_{word}}{h_{corpus}}$$

3.2.1.2 Interpolating

In the input signal, some points may be missing for different reasons: failure of the capturing device when capturing the sequence of points, writing speed... For instance, in the context of time sampled signal such as Figure 3.5(a), the character 't' has been written with a higher speed than the other characters (such as 'a' for instance). In that case, the density of points stored by the capturing device is lower and some points may be missing.

Our interpolating step consists in adding new points (one or more points depending on the resampling objective and the initial resolution) between two consecutive points P_i and P_{i+1} of the input handwritten word. The new point(s) $P'(x, y)$ to be added after a current point $P_i(x, y)$ is computed by using the Cubic Bezier curve method [65]. This method relies on the current point $P_i(x, y)$ and its three following points $P_{(i+1)}(x, y), P_{(i+2)}(x, y), P_{(i+3)}(x, y)$ in the signal sequence. $P'(x, y)$ can be defined as:

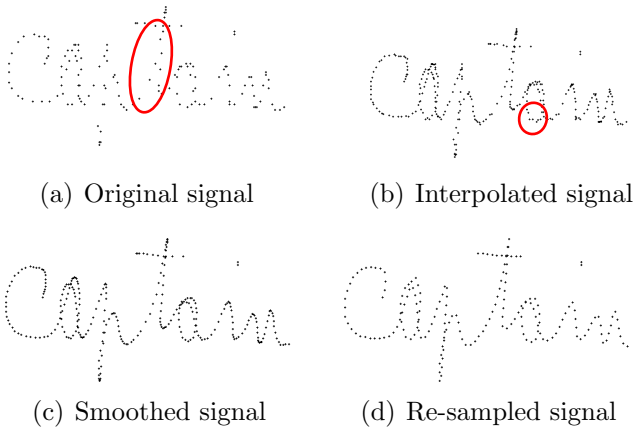
$$P'(x) = (1-t)^3 P_i(x) + 3(1-t)^2 t P_{i+1}(x) + 3(1-t)t^2 P_{i+2}(x) + t^3 P_{i+3}(x)$$

$$P'(y) = (1-t)^3 P_i(y) + 3(1-t)^2 t P_{i+1}(y) + 3(1-t)t^2 P_{i+2}(y) + t^3 P_{i+3}(y)$$

Where $t \in [0,1]$. If the capturing resolution is low, several points should be added between two consecutive points in the input word. In this case, different value of t can be chosen. In [65], this step has a minor impact on the recognition rate. It can be explained by the fact that the capturing device used in that system is good enough to store almost all the points of the writing trajectory. Furthermore, another explanation can be linked to the fact only one type of capturing device is used for creating training and test datasets.

In our research context, the interpolation step is somewhat important. Indeed, different databases (see section 4.1) are used in our work. Writings in these databases have been captured by using different capturing devices, the resolution of which could be very different. In addition, this research work is a part of the RecoNomad[1] project. Therefore, this HWR can be used with different capturing devices. Their capturing resolutions and capturing methods can be different from the capturing resolutions and methods of those used for creating the training databases.

An example of the results obtained by applying this interpolating method is illustrated in Figure 3.5(b).



(a) Original signal (b) Interpolated signal

(c) Smoothed signal (d) Re-sampled signal

Figure 3.5: Normalization and pre-processing at the word level.

[1] http://l3i.univ-larochelle.fr/Reco-Nomad.html

3.2.1.3 Smoothing

Smoothing consists in correcting outlier points of the on-line signal. Such points may be caused by imprecision due to the capturing device, writer's shaking hand, ... These points can be visually spotted in the interpolated signal, as illustrated in Figure 3.5(b).

In order to remove the outlier points, we correct every point P_i in the signal. The corrected version P'_i of the point P_i is the average of its k previous and k following points $(P_{i-k}, \ldots, P_{i+k})$ taking into account the angle α [65] which is composed by the three points P_{i-k}, P_i, P_{i+k} ($\alpha = P_{i-k}\widehat{P_iP_{i+k}}$). The more the angle α is small, the more the point P_i is likely to be an outlier point. Therefore, if the angle α is small, the modification of the point P_i has to be important. Otherwise, the modification of the point P_i is minor. The two coordinates $P'_i(x)$ and $P'_i(y)$ are computed as:

$$P'_i(x) = \frac{P_{i-k}(x)+...+\alpha P_i(x)+...+P_{i+k}(x)}{2*k+\alpha}$$

$$P'_i(y) = \frac{P_{i-k}(y)+...+\alpha P_i(y)+...+P_{i+k}(y)}{2*k+\alpha}$$

Where k is an integer value to be fixed. In many systems in the literature, k is assigned to 2 ($k = 2$). In our system, we use the same value ($k = 2$).

An example of a result obtained by applying this smoothing method is illustrated in Figure 3.5(c).

3.2.1.4 Re-sampling

As explained in the previous sections, the densities of points in the writings are not homogeneous because of different factors: writer, capturing resolution, capturing method, etc. Using the size normalizing, interpolating and smoothing methods allow respectively to standardize the size of the input signal, adding missing points and correcting outlier points. However, in general, the distances between two consecutive points are different after computing these steps. In some application such as writer identification, this variation can be a positive additional information to characterize the writing style of each writer. But, for handwritten word recognition problem, this variation may have a negative impact on the system because writings of one character class or word can be very different, especially when they are provided by different writers. Therefore, in this context, a spatial re-sampling method is required.

The spatial re-sampling consists in standardizing the input signal to a new signal where the distance between every two consecutive points is equal to a given objective distance Δ_{point}. The number N_p of points in the new signal can be estimated by $N_p = (\frac{LS}{\Delta_{point}})$,

where LS is the length of the input signal. The re-sampling algorithm is presented in Algorithm 1.

Algorithm 1 Re-sampling method

- input: Interpolated and smoothed signal S (*i.e.* sequence of points P_i), Objective distance Δ_{point}, Number of points in the new signal N_p
- output: New signal $S^{'}$ (*i.e.* sequence of points $P_n^{'}$) with equal distance Δ_{point} between two consecutive points
- Initialization: $p1 \leftarrow P_1$, $p2 \leftarrow P_2$, $i \leftarrow 2$, $n \leftarrow 1$, $d_{tmp} \leftarrow 0$
- add $p1$ to $S^{'}$: $P_1^{'} \leftarrow p1$
- while $n <= N_p$ do
  - Calculate the distance d between $p1$ and $p2$
    - if $(d_{tmp} < \Delta_{point})$ do
      - $d_{tmp} \leftarrow d_{tmp} + d$
      - move the points $p1$ and $p2$: $p1 \leftarrow p2$, $p2 \leftarrow P_{i+1}$, $i \leftarrow i + 1$
    - else if $(d_{tmp} + d = \Delta_{point})$ do
      - add $p2$ to S': $P_{(n+1)}^{'} \leftarrow p2$
      - $n \leftarrow n + 1$
      - $d_{tmp} \leftarrow 0$
      - $p1 \leftarrow p2$, $p2 \leftarrow P_{i+1}$, $i \leftarrow i + 1$
    - else
      - compute coordinates (x, y) of a new point $P_{(n+1)}^{'}$
        $P_{(n+1)}^{'}(x) \leftarrow p1(x) + (p2(x) - p1(x))\frac{\Delta_{point} - d_{tmp}}{d}$,
        $P_{(n+1)}^{'}(y) \leftarrow p1(y) + (p2(y) - p1(y))\frac{\Delta_{point} - d_{tmp}}{d}$
      - $p1 \leftarrow P_{(n+1)}^{'}$
      - $n \leftarrow n + 1$, $d_{tmp} \leftarrow 0$

3.2.2 Normalization and pre-processing at the character level

As mentioned earlier in section 3.1, our system relies on a SCR. This module is supervisedly trained using a training database of single characters. Hence, normalization and pre-processing at the character level are also required in order to normalize the single characters in the training database. The normalization and pre-processing methods at the character level have to ensure that the characters in the training database and the characters which will be segmented from handwritten words to be recognized are normalized in the same way.

The normalization and pre-processing method used at the character level is very sim-

ilar to the one used at the word level (presented in the previous section). The main difference is the size normalizing. Indeed, each word is composed by the concatenation of different characters. As explained in the previous section, the size of each word has to be normalized based to the corpus-height to make sure that the same character class written in different words has approximately the same size. In the case of isolated characters in the training database, on the other hand, the ground-truth is known in advance. Therefore, the size of each character class can be explicitly settled, as explained hereafter.

Let us remind here that the size of the input word is normalized on the basis on the corpus-height (see section 3.2.1.1 and Figure 3.4). As a consequence, the size of the characters that belong to the group 1 $(a, c, e, \dots)$ approximately equals the size of the corpus-height (Δ_{word}). The size of the characters that belong to the groups 2 and 3 $(b, d, \dots, p, q, \dots)$ approximately equals $2 * \Delta_{word}$ and the group 4 (f) approximately equals $3 * \Delta_{word}$.

Before applying any normalization and pre-processing on each character in the training database, its size has to be normalized by a scale factor δ_{char} according to their group. This scale factor is denoted as:

- $\delta_{char}(group\ 1) = \frac{\Delta_{word}}{h_{char}}$
- $\delta_{char}(group\ 2) = \delta_{char}(group\ 3) = \frac{2*\Delta_{word}}{h_{char}}$
- $\delta_{char}(group\ 4) = \frac{3*\Delta_{word}}{h_{char}}$

where h_{char} is the height of the character (distance between the global maximum and minimum points on the y axis).

Once the size of the characters is normalized, other pre-processing methods (interpolation, smoothing and re-sampling) can be applied in the same way as at the word level (see section 3.2.1).

During word recognition, each node in the lattice does not require any normalization or pre-processing since the word normalization and pre-processing at word level (see section 3.2.1) make sure that the character in the handwritten words to be recognized are normalized roughly in the same way as the characters in the training database. Anyway, we need to use a SCR which is relatively robust towards scale variations.

3.2.3 Conclusion

This section introduces the normalization and pre-processing methods at the word and character levels.

Pre-processings at the word level are applied to the input words to recognize. They remove noise and standardize the input signal. After this normalization and pre-processing, different writings of the same word have approximately the same size, with regular distance between two consecutive points. Hence, the variation in the input signal is reduced and the performance of the system can be ameliorated. Normalization and pre-processing at the character level, on the other hand, allow normalizing characters in the training database of the SCR.

During the recognition stage, the handwritten word signal is submitted to the word normalization and pre-processing steps. Once the input signal is normalized, it will be then submitted to the segmentation and lattice creation step which are introduced in the next section.

3.3 Segmentation, lattice creation and delayed stroke management

This section introduces a segmentation method, lattice creation and delayed stroke management. These three methods are applied as illustrated in Figure 3.6. Given a normalized word signal (after pre-processing, see section 3.2), delayed strokes are first detected (see section 3.3.1). An indicator is assigned to each stroke, to indicate its type: 'main stroke' or 'delayed stroke'. This handwritten word is then over-segmented into a set of graphemes using the explicit segmentation method presented in section 3.3.2. The graphemes that belong to delayed strokes (*i.e.* delayed graphemes) are temporarily removed. The remaining graphemes (*i.e.* belonging to main strokes) are used to create a lattice of L levels (see section 3.3.3). Delayed strokes are then integrated in the lattice using the delayed strokes re-localization method presented in section 3.3.4.

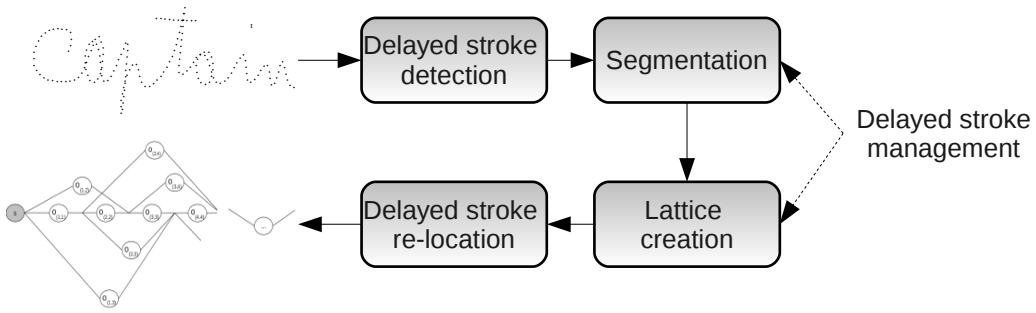


Figure 3.6: Sequential process for segmentation, lattice creation and delayed stroke management.

3.3.1 Delayed stroke detection

The term "Delayed stroke" refers to the strokes that are added after a certain delay. For instance, the dot of the characters 'i' and 'j' and the bar of the character 't' correspond to some delayed strokes. In some languages, accents can be added to some characters (for instance, in French, the accents $\{\acute{}, \grave{}, \hat{}, \ddot{}, \ldots\}$). Those accents constitute delayed strokes as well.

Figure 3.7 shows an example of delayed strokes. This handwritten word contains three strokes: one main stroke and two delayed strokes, which are respectively the dot of the character 'i' and the bar of the character 't'. The impact of this kind of strokes the system was presented earlier (in section 2.2.1).

Figure 3.7: An example of two delayed strokes. This word is composed of three strokes: 1 main stroke and 2 delayed strokes (the bar of character 't' and the dot of character 'i').

We define a method for detecting the delayed strokes relying on their acquisition time and their spatial positions on the x axis. Let us consider that the input signal contains T strokes ($S = \{S_1, S_2, \ldots, S_T\}$). The stroke $S_t \in S$ is considered as a delayed stroke if it is *overlapping* on the x axis with any stroke S_i or *located on the left* of any stroke S_i, where $i \in \{1, 2 \ldots, (t-1)\}$ (when the direction of the language writing is from left to right). Our delayed stroke detection method is presented in Algorithm 2.

Algorithm 2 Delayed stroke detection method.

- input: vectors of minimum values min_x and maximum values Max_x (on the x axis), for every stroke in the stork set S
- output: Indicator of the stroke S_t: indicator (S_t)
- for $t \leftarrow 2$ to T
  - indicator $(S_t) \leftarrow$ 'main stroke'
  - for $i \leftarrow 1$ to $(t-1)$
  - if $min\_x_t < Max\_x_i$ then
    - indicator$(S_t) \leftarrow$ 'delayed stroke'

3.3.2 Segmentation

This step aims at segmenting the input signal into a set of graphemes. Most on-line handwriting recognition systems rely on implicit segmentation methods. Therefore, very few explicit segmentation methods were presented in the literature.

In the case of off-line data, segmentation methods generally rely on local extrema points of word contours [79]. These points are considered as Potential Segmentation Points (PSP), where the connected component between two consecutive PSPs is considered as a grapheme to be segmented. The advantages of this method are its simplicity and stability.

E. Anquetil [4] introduced a segmentation method to segment on-line handwriting into a set of graphemes. In this method, the authors consider re-drawing points, angular points and loop points (see Figure 3.8), as potential segmentation points.

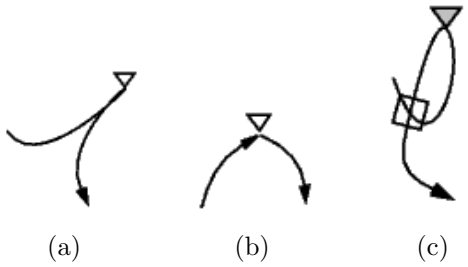


Figure 3.8: Example of segmentation points respectively: re-drawing point, angular point and loop point, extracted from [4].

Ahmad *et al.* [3] presented an explicit segmentation method using a basic assumption. The main idea of this method is very similar to the local extreme points of word contours used for off-line data, introduced above. In this method, the potential segmentation points are local extreme points on the y axis, as illustrated in Figure 3.9.

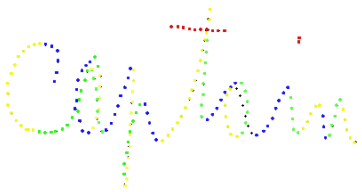


Figure 3.9: Example of graphemes segmented using the segmentation method presented in [3]. Each grapheme is a set of consecutive points displayed in one color.

In both methods (introduced by E. Anquetil [4] and Ahmad *et al.* [3]), the sequence of points located between two consecutive PSPs is considered as one grapheme. In the method introduced by E. Anquetil [4], the PSPs are precisely detected. Each point has to be assigned to its group (re-drawing point, angular point, etc.) since it plays a very important role in character recognition, which relies on fuzzy logic. Some of these potential segmentation points are not necessary in our system. For instance, the loop point is, generally, used to detect the loop primitive in writing and it does not represent a potential segmentation point between characters. In addition, the re-drawing points and angular points can be considered as a local extremum points on the y axis defined by Ahmad *et al.* [3]. For these reasons and for its simplicity and effectiveness, we used the method presented by Ahmad *et al.* [3] in our system.

Although the method presented by Ahmad *et al.* [3] is adapted to on-line data, it still has some difficulties handling the characters {'b', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'p', 'q', 't', 'y', 'z'}. Indeed, these characters are located in at least two writing zones: upper-middle, middle-lower, and/or upper-middle-lower zone. Using this method, another problem occurs when some graphemes contain parts of two neighboring characters, as illustrated in Figure 3.10(a) where the red grapheme contains a part of the character 'g' and a part of the character 'u'.

To handle this problem, we introduce a post-processing method to split the very high graphemes. First, the height h_g of each grapheme is computed by calculating the difference between its local extrema on the y axis. If $h_g \geq \alpha * h_{corpus}$, then this grapheme is split into two equal parts, as illustrated in Figure 3.10(b), where $\alpha \in [1, 2]$ is a fixed value. Of course, this method can over-segment the graphemes of high characters (for instance $\{b, d, p, q, \dots\}$) in the word, but it is not a problem since combinations of graphemes will be considered in the lattice (see section 3.3.3).

(a) Segmentation error (b) Correction

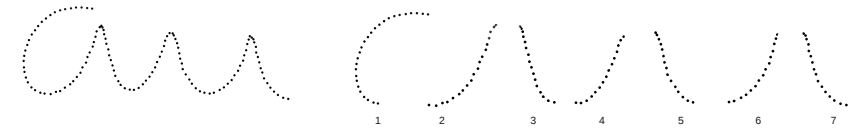
Figure 3.10: a) An example of segmentation error (red grapheme) that occurs when using the method presented in [3]. b) Error correction by applying our post-processing method.

Graphemes that belong to the delayed strokes (see section 3.3.1), are temporarily removed. We will call them *delayed graphemes*. The remaining graphemes, belonging to

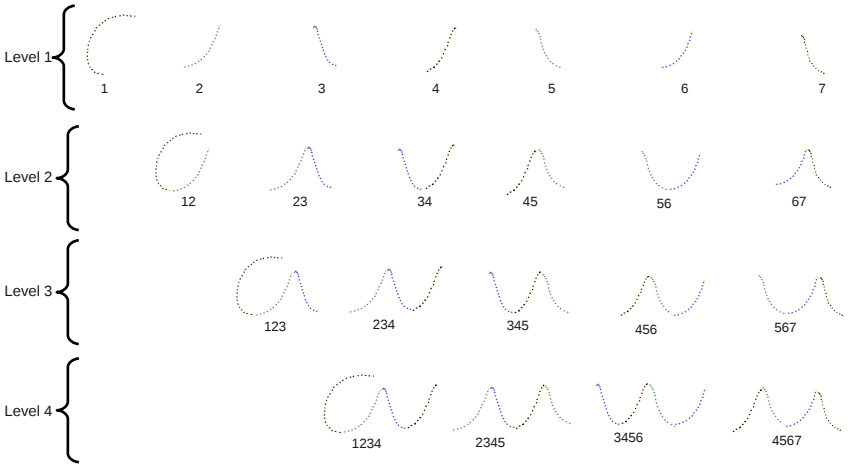
the main strokes (*i.e.* main graphemes), are further used to create a lattice of L levels, as introduced in the following section.

3.3.3 Lattice creation

The main graphemes are grouped in L levels, where L is the maximum number of graphemes used to compose one character, as illustrated in Figure 3.11(c). Because the lattice is created only for the handwritten word to recognize, we obviously do not have any ground-truth, so we consider a fixed value for the maximum number of graphemes L. This parameter is estimated by using the single character training database (see section 4.1.4).



(a) Normalized signal (b) Segmented graphemes



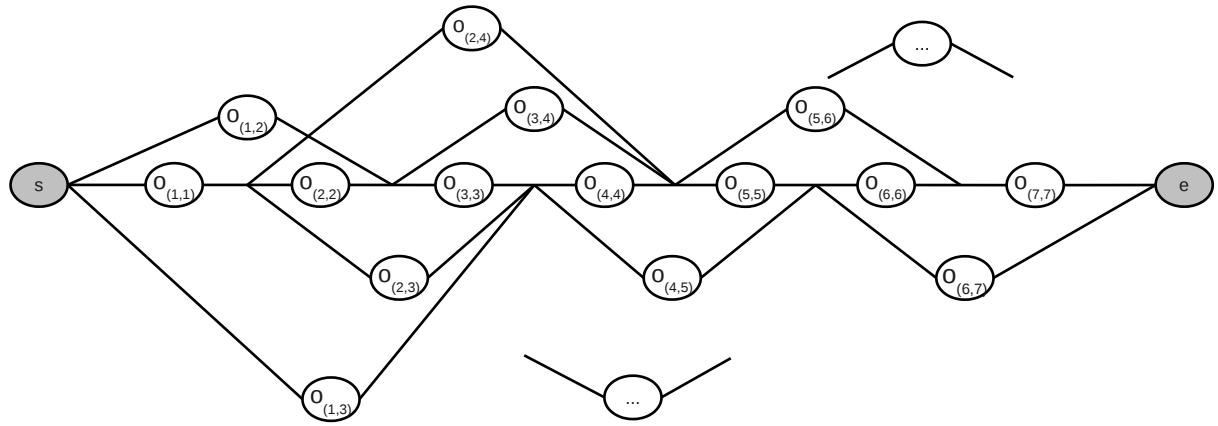
(c) Groups of graphemes

Figure 3.11: a) The normalized signal of word 'au', b) the segmented graphemes, c) the groups of graphemes at 4 levels ($L = 4$).

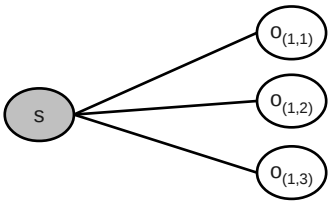
These groups of graphemes are organized into a lattice of L levels. This lattice illustrates all the possible connections between the different groups of graphemes and is further used in the word decoding process (see section 3.7). This lattice is constructed as follows. Suppose that the input signal contains T segmented graphemes ($G = \{g_1, g_2, \ldots, g_T\}$). At level 1, each node contains only one grapheme. Hence, we obtain a list of all segmented graphemes ($G_1 = G = \{g_1, g_2, \ldots, g_T\}$). At level 2, each group of graphemes contains two neighboring graphemes ($G_2 = \{g_{12}, g_{23}, \ldots, g_{(T-1)T}\}$),

and so on.

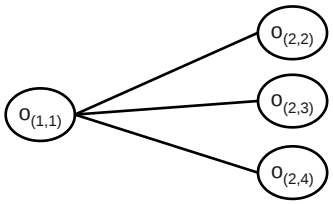
Figure. 3.12(a) illustrates an example of the lattice constructed from the groups of graphemes presented in Figure 3.11. The complete lattice cannot be drawn here since it is large and complex to be observed visually. The numbers in each node correspond to the indexes of the starting and ending graphemes. For instance, the node $o_{(1,3)}$ corresponds to the group 123 (*i.e.* g_{123}) at level 3. From this lattice, the connections can be clearly seen. For instance, nodes $\{o_{(1,1)}, o_{(1,2)}, o_{(1,3)}\}$ (see Figure 3.12(b)) are connected to the starting node of the lattice. Node $o_{(1,1)}$ connects to nodes $\{o_{(2,2)}, o_{(2,3)}, o_{(2,4)}\}$, and so on.



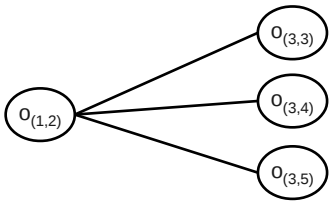
(a) Extract of the lattice of segmented graphemes



(b) forward connections of the starting node



(c) forward connections of the node $o_{(1,1)}$



(d) forward connections of the node $o_{(1,2)}$

Figure 3.12: An example of the lattice created from the groups of graphemes illustrated in Figure 3.11

3.3.4 Delayed stroke re-localizing

As mentioned earlier in section 3.3.2, graphemes that belong to delayed strokes (*i.e.* delayed graphemes) are temporarily removed. Delayed stroke re-localizing aims at adding these delayed graphemes to their corresponding nodes in the lattice and is performed straight after the lattice creation.

Supposing that the input signal contains D delayed graphemes ($Gd = \{gd_1, gd_1, \ldots, gd_D\}$) and P groups of graphemes (*i.e.* nodes), $GG = \{gg_1, gg_2, \ldots, gg_P\}$). The delayed grapheme $gd_d \in Gd$ belongs to a group of graphemes $gg_p \in GG$ if gd_d and gg_p are *overlapping* on the x axis. If this condition is valid then, delayed stroke re-localizing consists in adding gd_d to gg_p ($gg_p = gg_p \cup gd_d$).

3.3.5 Conclusion

This section introduces a segmentation method that over-segments the normalized input signal into a sequence of graphemes. Then, a lattice of L levels of these segmented graphemes is created. Furthermore, in order to solve the problems caused by the delayed strokes (see section 2.2.1), a delayed stroke management method is also presented. This method detects the delayed strokes and assigns them to the corresponding node in the lattice.

Each node in the lattice is further considered as a candidate character in the next process which will be presented in the following sections of this chapter. A set of features is therefore extracted from each node (see section 3.4) to feed a SCR (see section 3.5). Then, each pair of neighboring nodes in the lattice is submitted to the bi-character models (see section 3.6) in order to validate or invalidate the output of the SCR, taking into account the context of neighboring potential characters.

3.4 Feature extraction

This section introduces the features we use for our SCR and for our bi-character models. These features are extracted from the group of graphemes in each node, and from each pair of neighboring nodes of the lattice (see section 3.3.3), as illustrated in Figure 3.13. The features extracted from each node will be used to feed the SCR (see section 3.5) and the features extracted from each pair of neighboring nodes will be used to feed the bi-character models (see section 3.6).

As mentioned in [140], in the field of pattern recognition, choosing a feature extraction method is a very important step. Each feature is designed to represent a specific information. Comparing, choosing, selecting the most relevant features is still an open problem. Usually, experimental evaluations are needed in order to compare their performance for each specific objective. Nonetheless, in the context of handwriting recognition, researchers agree on the idea that a single feature extraction method is insufficient to represent all the possible variations in the handwriting [54]. According to the experimental results in [23, 139], combining different feature extraction methods improves the effectiveness of handwriting recognition systems. In addition, in the case of on-line data, both on-line and off-line features can be extracted (see section 2.2.3). Each category of features (on-line/off-line) has its own advantages and drawbacks. Therefore, in our system, we propose to use both categories of features (on-line and off-line) in order to take

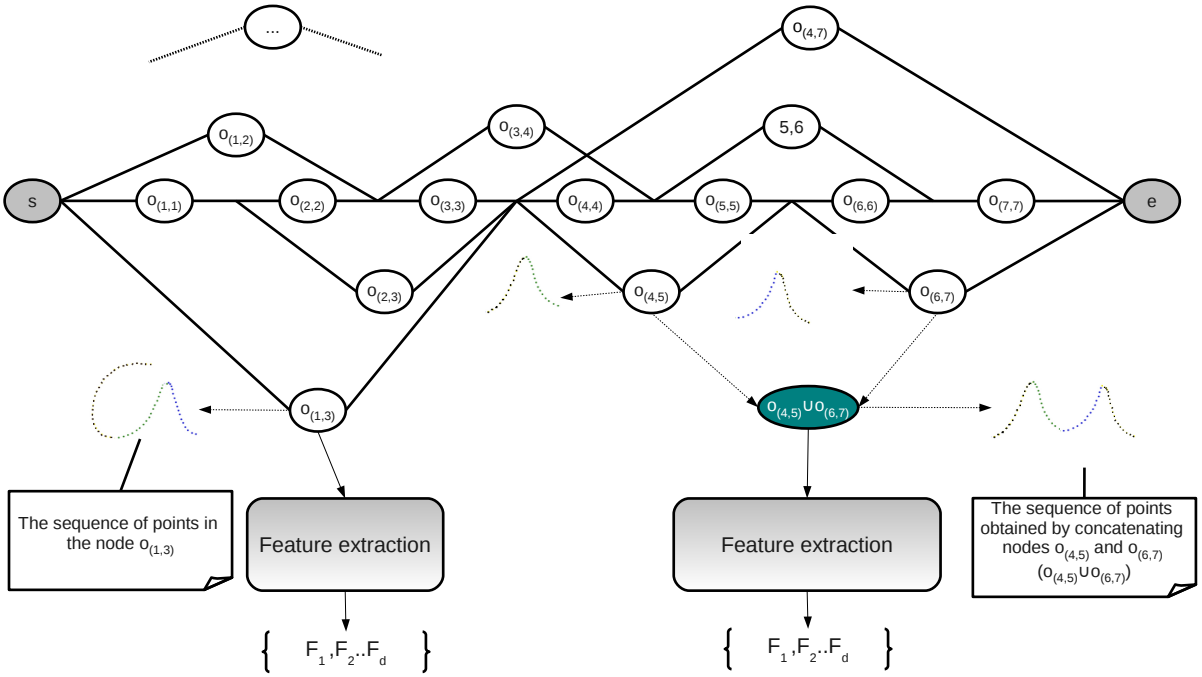


Figure 3.13: Extraction of a set of features from each node and each pair of neighboring nodes.

advantage of their complementarity. In each of these 2 categories, different extraction methods are used. The obtained feature vectors are concatenated to obtain a single feature vector (see Figure 3.14). These features will be detailed in sections 3.4.1 and 3.4.2.

In order to reduce the computational complexity, a feature selection process (see section 3.4.3) will be performed in order to select a sub-set of the features to be used as an input of our SCR and bi-character models.

We have to mention here that feature extraction is not among the contributions of this thesis. We only studied different features in the literature and selected the ones that we thought being the most adapted to our context.

3.4.1 Off-line features

In our system, off-line features are computed from the reconstructed binary image, which is obtained by connecting the consecutive points of the on-line signal and applying dilatation on the resulting skeleton (see section 1.1).

The features set presented in [54] provides high recognition rates in the context of off-line ICR. Therefore, these features will be used in our system. They consist in 3 fam-

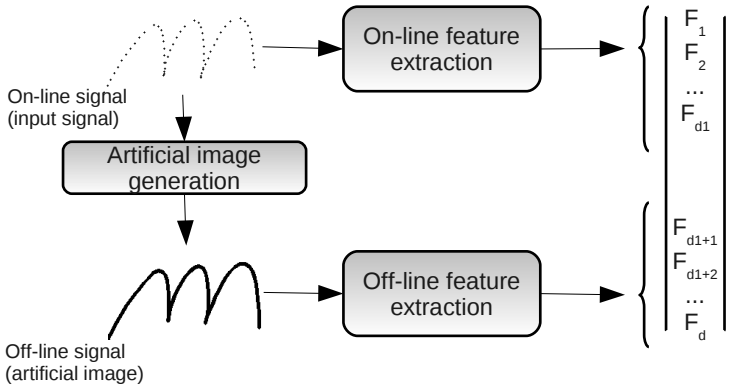


Figure 3.14: Combining off-line and on-line features.

ilies of global features (Hu invariant moments, projection, profile) and 4 families of local features (intersection with straight lines, local extrema, points/junctions and holes and concave arc). These features will be introduced from the sub-sections 3.4.1.1 to 3.4.1.6. In addition to these features, we add the zoning density features which allow computing the distribution of black pixels in the shape (see sub-section 3.4.1.7).

In the context of handwriting, characters are more or less rotated due to the variation problem. Features which are invariant to rotation are therefore generally required, in addition to non rotation invariant features. Indeed, if we used rotation invariant features, the SCR could make confusions between 'p' and 'q' for instance. However, this problem might be solved at the bi-character level. Many invariant descriptor have been proposed in the literature, in order to tackle this rotation invariant question. In [54], only Hu invariant moments is used. However, in the literature, Zernike and Radon moments are often considered as very good descriptors to represent precisely the input image. Some experimental results presented in [72] for sign language recognition system, show a slight superiority of Zernike moments compared to Hu moments. In our context, we decided to use several set of such invariant moments: Hu moment, Zernike moment and Radon moments, considering that feature selection will keep only most useful for our problem.

3.4.1.1 Hu moments

The Hu invariant moments were presented by M.K Hu in [61]. These features are computed from the global shape of the input image to describe the pixel distribution around its center of gravity. A vector of Hu moment invariant generally contains a set of seven moments $\{\phi(1), \phi(2), \ldots, \phi(7)\}$ (of maximum order 3) which are invariant to translation, scale and also rotation. They are computed as:

- $\phi(1) = \mu_{20} + \mu_{02}$
- $\phi(2) = (\mu_{20} - \mu_{02})^2 + 4\mu_{11}^2$

- $\phi(3) = (\mu_{30} - 3\mu_{12})^2 + (3\mu_{21} - \mu_{03})^2$
- $\phi(4) = (\mu_{30} + \mu_{12})^2 + (\mu_{21} + \mu_{03})^2$
- $\phi(5) = (\mu_{30} - 3\mu_{12})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{12} + \mu_{03})^2] + (3\mu_{21} - 3\mu_{03})(\mu_{21} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu_{03})^2]$
- $\phi(6) = (\mu_{20} - \mu_{02})[(\mu_{30} + \mu_{12})^2 - (\mu_{21} + \mu 03)^2] + 4\mu 11(\mu_{30} + \mu_{12})(\mu_{21} + \mu_{03})$
- $\phi(7) = (3\mu_{21} - \mu_{03})(\mu_{30} + \mu_{12})[(\mu_{30} + \mu_{12})^2 - 3(\mu_{21} + \mu_{03})^2] - (\mu_{30} - 3\mu_{12})(\mu_{12} + \mu_{03})[3(\mu_{30} + \mu_{12})^2 - (\mu_{12} + \mu_{03})^2]$

where

- x_i and y_i are respectively the x and y coordinates of the black pixel P_i in the binary image.
- $\mu_{pq} = \frac{1}{[\sum_{i=1}^{I}(x_i - \overline{x})^2 + \sum_{i=1}^{I}(y_i - \overline{y})^2]^{\frac{(p+q)}{2}+1}} \sum_{i=1}^{I}(x - \overline{x_i})^p(y - \overline{y_i})^q$
- $\overline{x} = \frac{1}{I}\sum_{i=1}^{I} x_i$, $\overline{y} = \frac{1}{I}\sum_{i=1}^{I} y_i$
- I is the total number of black pixels in the image

3.4.1.2 Projection

Projection features [53] permit to locate the dense regions (*i.e.* larger strokes) in the horizontal and vertical directions of a given image based on cumulative histograms (horizontal and vertical).

The cumulative histogram of the vertical projection counts the number of black pixels in each column of the image cumulatively. For instance, let us call $h(1)$, $h(2)$ and $h(3)$ the number of black pixels in the first, second and third columns of the input image. The cumulative histogram can be computed as: $c(1) = h(1)$, $c(2) = c(1) + h(2)$ and $c(3) = c(2) + h(3)$. The cumulative histogram of the horizontal projection can be computed in a similar way, but counting the number of black pixels in each row cumulatively.

For each cumulative histogram, the y axis is divided into equal interval by 10 values $\{y1, y2, \ldots, y10\}$. The projection features are the abscisses corresponding these values in the cumulative histogram ($\{x1,\ x2,\ \ldots,\ x10\}$) (see Figure 3.15). Close values of x_i's indicate dense regions in the direction of the projection (horizontal or vertical).

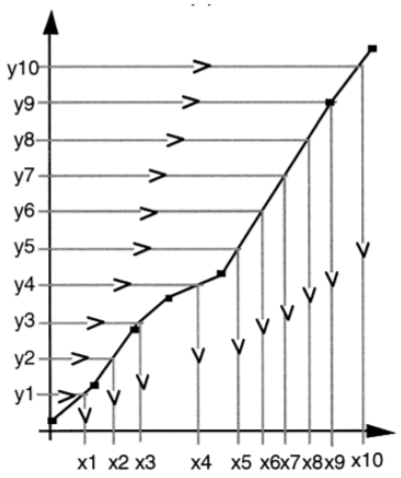
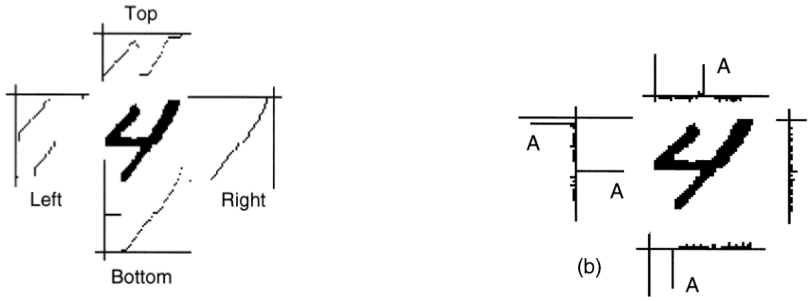


Figure 3.15: Projection features presented in [53].

3.4.1.3 Profile

These features describe the smoothness of a character on the left, right, top and bottom of the image based on its 4 profiles: left, right, top and bottom profiles [53].

A profile of an image refers to the set of first black pixels found while scanning the image in any direction. For example, the left profile is obtained by scanning the image from left to right, and row by row. For each row, only the position of the first pixel encountered when scanning the binary image is kept in the left profile. The top profile is obtained by scanning the image from top to bottom, and column by column, keeping only the position of the first black pixel met, for each column. An example is given in Fig 3.16(a).



(a) Example of profiles

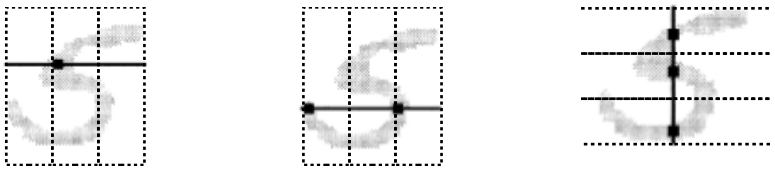
(b) Example of first order derivative of each profile

Figure 3.16: a) Example of four profiles (left, right, top and bottom) presented in [53] and b) their first order derivative results.

For each profile, the first order derivative is computed (see Figure 3.16(b)) in order to extract the maximum amplitude of each profile. The obtained maximum amplitudes of the left and right (and respectively top and bottom) profiles are normalized by the width (and height) of the input image. We obtain 4 features each of which describes the smoothness of the character on that corresponding side (left, right, top or bottom).

3.4.1.4 Intersection with straight lines

This feature extraction method [54] counts the number and the position of intersections between character shapes and some selected straight lines. According to their discussions and experiments, two horizontal straight lines and only one vertical straight line are sufficient (see Figure 3.17). These two horizontal straight lines are located at $\frac{1}{3}$ and $\frac{2}{3}$ of the character height. The vertical straight line passes through the center of gravity of the shape.



(a) Horizontal line, located at $\frac{1}{3}$ of the height

(b) Horizontal line, located at $\frac{2}{3}$ of the height

(c) Vertical line

Figure 3.17: An example of intersections between the character shape and the straight lines: 2 horizontal lines and 1 vertical line. This example is adapted from [54].

In the case of the horizontal straight lines, we divide the image into 3 equal columns. The features are the number of intersections of black pixels with each line, in each column. We obtain 6 features. In the case of the vertical straight line, we divide the image into 3 equal rows. And the features are the number of intersections with the vertical line in each row. We obtain 3 features.

3.4.1.5 Local extrema

This feature extraction method was introduced in [54]. It aims at extracting a set of features concerning the extremum points at the top, bottom, left and right of a given image, as illustrated in Figure 3.18. An extremum point at the top (*i.e.* top extremum) refers to the black pixel which has no connection with other upper or the same level black pixel among its 8 neighboring pixels. They can be extracted by reading the image from top to bottom and by retraining the corresponding black pixels. The bottom, left and right extrema points can be detected in the same way.

To extract the retained features, we divide the input image into $9 = 3 * 3$ zones. For each zone, we count the number of top, bottom, left and right extrema, detected separately. We obtain in total a vector of 36 features (9 features for each type of extremum point).

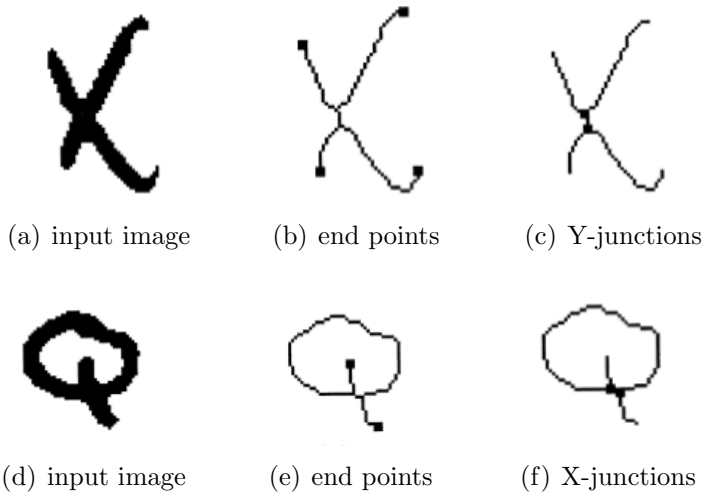
(a) Top (b) Bottom (c) Left (d) Right

Figure 3.18: Examples of local extrema features, extracted from [54].

3.4.1.6 End points and junctions

This feature extraction method [54] aims at extracting the number and the position of end points, Y-junctions and X-junctions from the skeletonized binary image, as illustrated in Figure 3.19.

An end point refers to a black pixel that connects to only one black pixel (in the skeleton) among its 8 neighbor pixels. Y-junctions are detected by applying a set of twelve 3 x 3 templates on each black pixel of the skeleton image (representing each Y-junction possible configuration). In the case of X-junctions, they can be detected using two different methods. The first method uses a set of two templates (representing each X-junction possible configuration), applied in the same way as for Y-junction detection. The second method aims at merging a pair of nearest Y-junctions into a X-junction (see Figure 3.19(f)).



(a) input image (b) end points (c) Y-junctions

(d) input image (e) end points (f) X-junctions

Figure 3.19: An example of end points, X and Y-junctions (extracted from [54]).

3.4.1.7 Zoning density

Zoning density features is a set of ten basic features which contains one global feature and nine local features. The global feature (F_0) is the percentage of black pixels in the whole image. F_0 is denoted by:

$$F_0 = \frac{BP}{W * H} \tag{3.1}$$

where BP is the number of black pixels of the image, W and H are respectively the width and the height of the input image.

The input image is then divided into $3 * 3$ equal zones. For each zone, the percentage of black pixels is also computed. We obtain in total a set of nine local features $\{F_1, F_2, \ldots F_9\}$. These features are computed by:

$$F_i = \frac{BP_i}{w * h} \tag{3.2}$$

where $i \in \{1, 2, \ldots, 9\}$ corresponds to each zone of the image, BP_i is the number of black pixels in the zone i, w and h are respectively the width and height of the zones.

3.4.1.8 Radon transform and R-signature

The Radon transform consists in projecting a given image $f(x, y)$ onto the lines $L(t, \theta)$ taken at different angles $\theta = \theta_1, \theta_2, \ldots$ and $-\infty < t < \infty$, as illustrated in Figure 3.20.

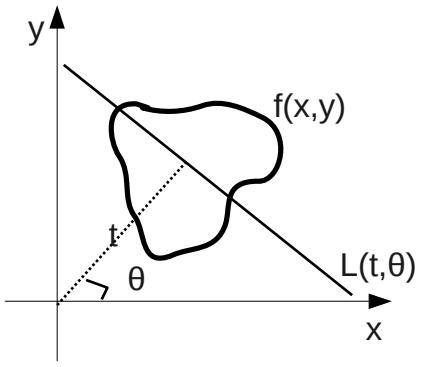


Figure 3.20: Definition of the Radon transform.

The Radon transform matrix $RA(t, \theta)$ of the input image $f(x, y)$ can be directly used as a feature set to represent the image $f(x, y)$. However, the Radon transform is not

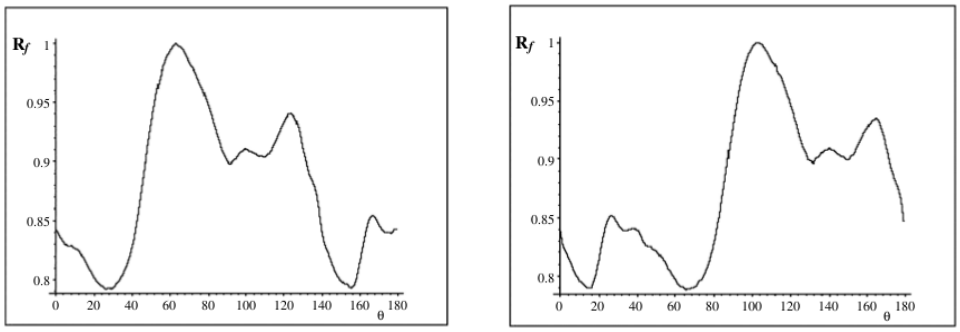
invariant to rotation, translation and scale. In this case, we have to apply some normalization process to make the features to be invariant such as Affine transformation, as introduced in [123].

Instead of using the Radon transform $RA(t, \theta)$ (which can be huge and invariant to rotation, translation and scale) as raw features, we can also extract different families of features from this Radon transform such as Histogram of Radon Transform [132] or the R-signature [130, 133]. In our context, we decided to extract the R-signature family since this feature family is originally designed for binary image shape recognition.

The R-signature $R(\theta)$ can be extracted by summing up the squared of each column of the Radon transform matrix $RA(t, \theta)$, as denoted in the Equation (3.3). Finally, this R-signature $R(\theta)$ can be normalized by its total value.

$$R(\theta) = \int_{-\infty}^{\infty} RA^2(t, \theta)dt \tag{3.3}$$

As explained in [133], the R-signature is invariant to translation and scale. However, a rotation by an angle θ_0 of the input image provides a shift of θ_0 on the R-signature, as illustrated in Figure 3.21. In order to make the R-signature invariant to rotation, some additional method should be used. For instance, in [133], Fourier transform is applied to $R(\theta)$. In our case, we decided to use only the raw R-signature. We are aware that using R-signature invariant to rotation may improve the effectiveness of the system.



(a) Original image (b) Rotated image

Figure 3.21: Shift on the R-signature when the input image is rotated by an angle θ_0, extracted from [133].

3.4.1.9 Zernike invariant

Zernike moments were originally introduced by Michael Reed Teague [135] and have been largely used in many pattern recognition problems [74, 75, 115], including character recognition [27]. They represent one of the most cited set of invariant features when

considering pattern recognition problems.

This approach aims at projecting an input image onto a set of complex Zernike polynomials which are orthogonal to each other on the unit circle $x^2 + y^2 \leq 1$. The Zernike function is defined in the polar coordinates (r, θ) by:

$$V_{nm}(x, y) = V_{nm}(r \sin\theta, r \cos\theta) = R_{nm}(r)e^{im\theta} \tag{3.4}$$

where

$$R_{nm}(r) = \sum_{s=0}^{\frac{(n-|m|)}{2}} \frac{(-1)^s (n-s)!}{s!(\frac{(n+|m|)}{2} - s)!(\frac{(n-|m|)}{2} - s)!} r^{n-2s} \tag{3.5}$$

and

- n is a non-negative integer
- m is an integer
- $n - |m|$ is even
- $|m| \leq n$
- $r = \sqrt{x^2 + y^2}$
- θ is the angle composed by the vector r and the x axis in counterclockwise direction.

The Zernike feature of order n with repetition of m for an image $f(x, y)$ can be computed by projecting this image onto the corresponding Zernike polynomial V^*_{nm}, as follows:

$$Z_{nm} = \frac{n+1}{\pi} \sum_x \sum_y f(x, y) V^*_{nm}(x, y) \Delta x \Delta y \tag{3.6}$$

Where $V^*_{nm}(x, y) = V_{n,-m}(x, y)$

The above original Zernike features are only invariant to rotation. In order to make them invariant to scale and translation, some normalization techniques are generally used [75]. We use this invariant version of Zernike feature in our work [75].

3.4.2 On-line features

These features are extracted from the on-line signal, mainly inspired from [4]. Five sets of features are extracted: normalized length, downward stroke information, start and end point information, histogram of writing direction and derivative, and re-drawing point. Each set of features is detailed in the following sub-sections.

3.4.2.1 Normalized length

Some character classes in the Latin alphabet have the same length and/or width and/or height. For instance, the character classes 'a' and 'b' have the same width but different length and height. The character classes 'g' and 'w' have approximately the same length but different width and height. Given an input on-line singal of a character, three important values can be computed:

1. Length: the length of the input signal is obtained by computing the sum of the distance between two consecutive points.
2. Width: the width of the character shape is obtained by computing the distance between the minimum and the maximum points on the x axis.
3. Height: the height of the character shape is obtained by computing the distance between the minimum and the maximum points on the y axis.

This feature extraction method consists in computing the normalized length of the input on-line signal by using its width and height $(\frac{length}{width}, \frac{length}{height})$.

Taking the example above, given two on-line signals of the characters 'a' and 'b', we obtain: $\frac{length(\text{`}a\text{`})}{width(\text{`}a\text{`})} < \frac{length(\text{`}b\text{`})}{width(\text{`}b\text{`})}$ since $length(\text{`}a\text{`}) < length(\text{`}b\text{`})$ and $width(\text{`}a\text{`}) \simeq width(\text{`}b\text{`})$. As a consequence, writings of characters 'a' and 'b' could be discriminated using this feature.

3.4.2.2 Downward strokes

A downward stroke is the writing trajectory from a local maximum point to the following local minimum point. These strokes are more important than upward strokes, which are generally known as connecting strokes [4]. Indeed, as illustrated in Fig 3.22(c), the downward strokes are generally sufficient to almost correctly represent the input handwritten word.

(a) Original signal (b) Downward (black) and upward (yellow) strokes (c) Signal with only downward strokes

Figure 3.22: An example of upward and downward strokes in an handwriting of the word 'captain'.

The downward strokes contain the number of downward strokes, the length of the longest downward stroke, and the coordinates x, y of the highest point in the downward strokes.

3.4.2.3 Start and end point information

This feature extraction method computes the information on the starting and ending points (P_0, P_t) of the signal. This information contains: x, y coordinates and sine and cosine of writing direction at the starting and ending points, as illustrated in Figure 3.23.

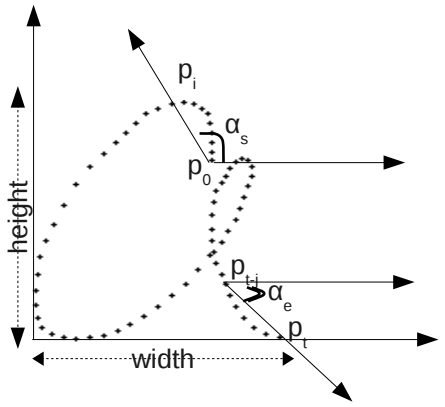


Figure 3.23: Extraction features at the start and end point of the signal.

The coordinates of each point have to be normalized with its height or width of the signal. In the case of starting point (P_0), we obtain two features $x_s = \frac{P_0(x)}{width}$ and $y_s = \frac{P_0(y)}{height}$. The x, y coordinates of the ending point (P_t) can be computed in the same way.

The writing angle at the starting point (α_s) is the angle between the vector $\overrightarrow{P_0P_i}$ and the x axis. The point P_i is the closest following point of P_0, where $|P_0P_i|$ is greater than an objective distance δ. Writing direction at the ending point (α_e) is computed in a similar way. It is the angle between the vector $\overrightarrow{P_{t-j}P_t}$ and the x axis. The point P_{t-j} is the closest previous point, where $|P_{t-j}P_t|$ is greater the an objective distance δ. In our system, $\delta = \frac{length}{10}$, where $length$ is the total length of the signal.

3.4.2.4 Histogram of writing direction

This method computes the histogram of writing directions. First, for each two consecutive P_{i-1} and P_i, a writing direction angle α_i is computed. Then, we associate the angle α_i to one of the eight equal gaps, as illustrated in Figure 3.24. For instance, if $\alpha_i \in]\frac{\pi}{8},\ 3\frac{\pi}{8}]$, its corresponding writing direction is 1. In each direction, the total length of the pairs of points which belongs to this direction is computed. Let us denote the total length in each direction by l_d, where $d \in \{1,\ 2\ \ldots\ 8\}$. l_d is computed by Equation (3.7).

$$l_d = \sum_{i=2}^{NP} |P_{i-1}P_i|,\ such\ that\ \alpha_i \in d. \tag{3.7}$$

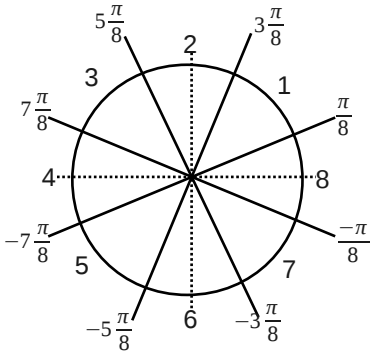


Figure 3.24: Writing direction divided into eight partitions.

Where NP is the total number of points of the input signal.

In total, we obtain eight values $l_1,\ l_2,\ \ldots\ l_7,\ l_8$ where each value corresponds to each direction. These values are normalized by the total length of the input signal and considered as the retained global features.

In order to obtain more detailed information, we segment the input signal into three segments of equal length. These segments represent beginning, middle and ending parts of the character. For each segment, a set of eight histogram features is also extracted. We obtain another 24 local features.

3.4.2.5 Derivative and re-drawing points

The derivative point refers to the point where the writer immediately changes the direction of the writing trajectory. The re-drawing point refers to the derivative point for which the new writing trajectory overlaps on the previous trajectory. Figure 3.25 illustrates an example of the derivative points (indicated by rectangles) and re-drawing points (indicated by circles).

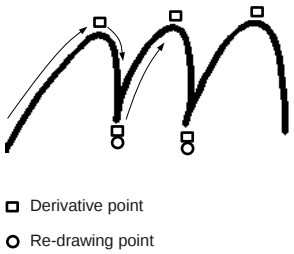


Figure 3.25: An example of the derivative and re-drawing points.

This feature extraction method consists in counting the number of derivative and

re-drawing points. Derivative and re-drawing point detection can be described as below:

- For each point P_i in the input signal, compute the angle α_1 composed by the vector $\overrightarrow{P_{i-1}P_i}$ and the x axis and the angle α_2 composed by the vector $\overrightarrow{P_iP_{i+1}}$ and the x axis (see Figure 3.26(a)).
- Compute the difference δ_{angle} between the angles α_1 and α_2. δ_{angle} describes the changing of writing direction (see Figure 3.26(b)).
  - An important value of δ_{angle} indicates the certitude of derivation. We consider that, if $\delta_{angle} > \frac{\pi}{2}$, then P_i is a derivative point, where $\frac{\pi}{2}$ is a threshold value.
  - In the case where P_i is a derivative point and δ_{angle} gets close to π, then the point P_i will be considered as a re-drawing point. In our system, if $\delta_{angle} \in [\pi - \frac{\pi}{4}, \pi + \frac{\pi}{4}]$, then we consider that P_i is a re-drawing point.

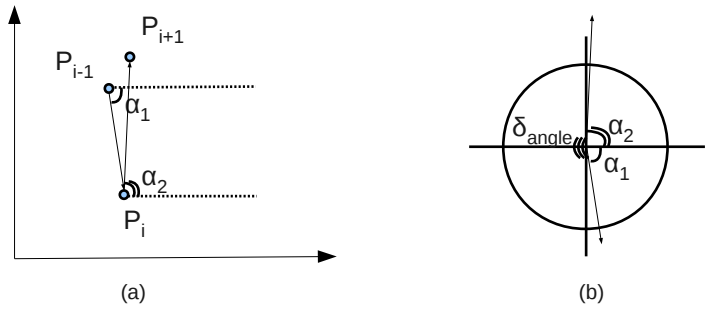


Figure 3.26: An example of derivative and re-drawing point detection.

Taking the example given in Figure 3.26, the point P_i is considered as a derivative point and a re-drawing point at the same time.

3.4.3 Feature selection

The combination of all off-line and on-line features we extract (ses previous section) provides a large set of 254 features. This large set of features may contain some insignificant features, as well as some redundant features. This kind of features may increase the computational time, and sometime degrade the effectiveness of the classification system. Hence, a feature selection method is applied in order to select a subset of significant features.

In the literature, a large amount of research have been dedicated to feature selection problem, which aims at selecting a subset of F^* significant features out of a set of F features (*i.e.* original features set). This new subset of features F^* must not degrade (and if possible improve) the effectiveness of the system.

A.Jain and D.Zongker [66] have classified the feature selection methods as presented in Figure 3.27. At the highest level, all these methods are classified into two categories: Statistical Pattern Recognition based (SPR) and Artificial Neural Network based (ANN). In the ANN-based category, the Node Pruning method is usually used. It consists in pruning the least significant nodes (and therefore removing the corresponding input features) to reduce the complexity of the network. In the SPR-based category, methods are classified into different categories. We can finally identify 5 categories different: an optimal category and 4 sub-optimal categories (deterministic single-solution, stochastic single-solution, deterministic many-solution, stochastic many-solution).

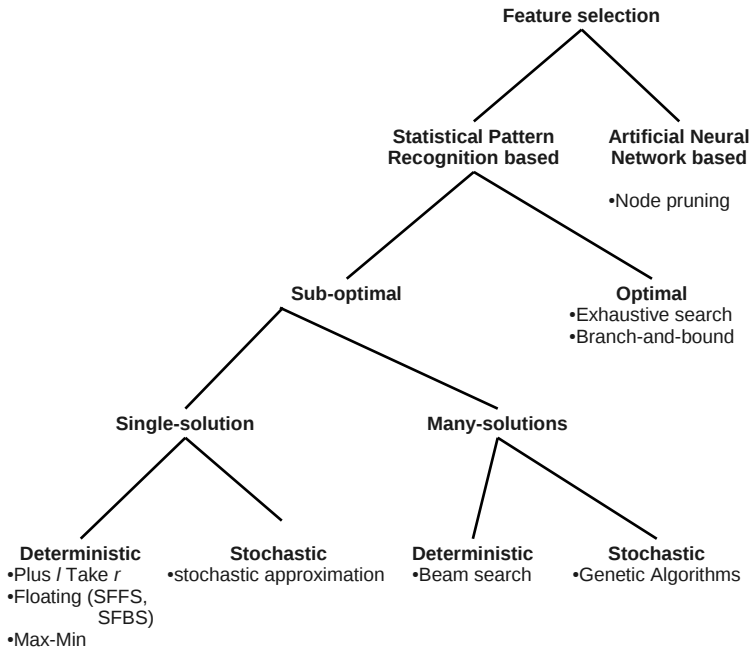


Figure 3.27: The tree of feature selection methods, extracted from [66].

According to the experimental results presented in [34, 66], the Sequential Floating Forward Selection (SFFS) belongs to the deterministic single-solution category, introduced by Pudil *et al.* [114] outperforms all other methods. Hence, we decided to use the SFFS method in our system by using the SVM classifier (see section 3.5.2) as an objective function. The SFFS procedure can be briefly described as below:

Given a set of features F, the SFFS procedure can be presented into 3 steps:

- Initialization: $k = 0$ and $F_k^* = \emptyset$
- **Step1 (Inclusion):** finds the most significant feature $f^+ \in F$ that maximizes the objective function $J(F_k^* + f^+)$. Then, add the feature f^+ into the set F_k^*. Hence, $F_{k+1}^* = F_k^* + f^+$, $k = k + 1$
- **Step2 (Condition exclusion):** finds the least significant feature f^- in F_k^*, where $f^- = \text{argmax}_{f \in F_k^*} J(F_k^* - f)$ and $J(F_k^* - f^-) > J(F_{k-1}^*)$

  - if f^- is the feature just added, keep it and go to step 1.
  - otherwise, remove f^- from F_k^*. Hence, $F_{k-1}^* = F_k^* - f^-$, $k = k - 1$. Then, go to step 3.
- **Step3 (Continuation of conditional exclusion):** continues removing the least significant features in set F_k^* as in Step2. If $k = 2$, go to step 1. Otherwise, repeat step3.

We will see in the following parts that in the SCR (see section 3.5) and bi-character models (see section 3.6), only a set of selected features (*i.e.* significant features F^*) obtained by applying this technique is used. However, this set of features has been selected only for classification of single character. We are aware that considering a different set of features for bi-character models (or adapted to each bi-character model) might improve the recognition rates, however, it would also increase the computational complexity, since we could not re-use the same features for each node and combination of nodes in the lattice (see section 3.6.2).

Let us precise here that we do not consider that this feature selection stage is a contribution of our research.

3.4.4 Conclusion

This section introduces the feature extraction methods we use in our system. They are classified into two groups: off-line and on-line features.

The set of off-line features contains nine families of features (local and global). These features are widely used in the context of pattern recognition as well as for off-line ICR. The set of on-line global features allows using the information in the on-line signal, such as the information about the downward stroke and the writing direction. The combination of these two sets of features allows taking profit of their complementarities. However, considering the fact that some of these features may contain some noise and be redundant, the SFFS feature selection method is used to select the relevant ones. It allows selecting only the useful feature set F^* to be used in the system.

These selected feature set F^* will be used in the SCR (see section 3.5) and the bi-characters models (see section 3.6).

3.5 Single character analysis

3.5.1 Objective

As mentioned earlier in section 3.1, our system relies on an explicit segmentation/recognition method. Therefore, the Single Character Recognition (SCR) system is obviously a crucial step since it is used to define character candidates by recognizing groups of

graphemes for each node of the lattice (see section 3.3), as illustrated in Figure 3.28. This SCR relies on a set of selected features given by the feature selection step (see section 3.27).

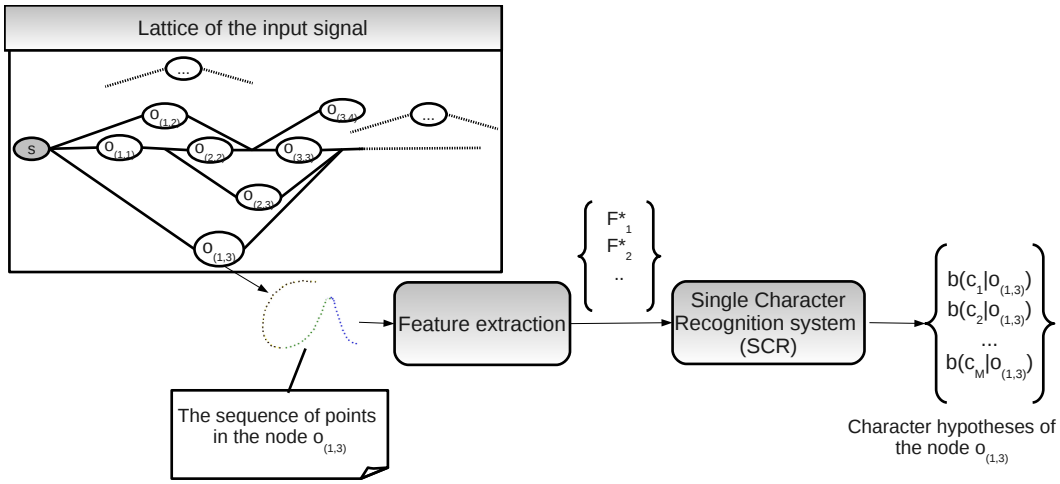


Figure 3.28: The application of the SCR in the proposed HWR.

In the example of Figure 3.28, this set of features is extracted from the node $o_{(1,3)}$ to feed a SCR. This system returns the probability estimate $b(c_m|o_{(1,3)})$ of each character class c_m for the node $o_{(1.3)}$. In the scope of our research (see section 1.5), we focus on lowercase and non-accented words. Hence, the character class set corresponds to the Latin alphabet ($c_m \in C, C = \{a, b, \ldots, z\}$), which contains 26 classes ($M = 26$). Of course, we plan to extend the number of classes by considering uppercase letters in future work.

The SCR has to be able to:

1. Recognize the characters segmented from handwritten words, by integrating the fact that the shapes of one character class can vary depending on its surrounding characters (see section 1.3.1). Indeed, the variation of the segmented characters may be more important than those of isolated characters (which are written in predefined boxes). Therefore, a system trained with an isolated character database could not cover all the possible variations in segmented characters. Hence, it has to be trained with characters segmented from handwritten words. This training database will be presented in section 4.1.4.

2. Reject unknown patterns, since some nodes of the lattice contain the concatenation of multiple neighboring graphemes (see section 3.3) which do not systematically correspond to characters. This kind of node has to be classified as an unknown pattern, *i.e.* to be rejected. In order to deal with this problem, a recognition system with rejection has to be applied (see section 3.5.3).

3.5.2 Single character recognition system

According to A.K. Jain *et al.* [67], recognition approaches are classified into four categories: template matching, structural matching, statistical classification and neural networks.

1. Template matching: This type of approaches consists in matching the input data against predefined template(s) (*i.e.* prototype(s)) of every predefined class. This approach has been successfully used for various applications [31, 68, 63]. However, in the problem of SCR, the intra-class variation can be very important (see section 1.3.1). Hence, it is very difficult to define the template(s) for each class. This method is therefore not adapted to this context.

2. Structural and syntactical matching: Using this type of approaches, each class is represented in terms of the inter-relationships between a set of predefined primitives (*i.e.* sub-classes) based on a pre-defined grammar or graphs. Few handwriting recognition systems based on structural matching have been presented in the literature [4, 25]. Indeed, defining the primitives and a grammar which allow covering all the variations in handwriting is a hard task. Despite the fact that this type of approach seems to be very promising, it is still rarely used for handwriting recognition in the literature.

3. Neural networks: A neural network is an interconnection of a large number of artificial neurons. It can be seen as a weighted directed graph, in which nodes correspond to neurons and edges corresponds to weights. This interconnected architecture allows connecting the neurons from an input layer to the neurons of an output layer like in MultiLayer Perception (MLP) for instance. The input layer is fed by a vector of d features which represents the objects to recognize and the output layer usually corresponds to the classes to be recognized. This approach is able to learn non-linear classification problems. Hence, it is able to deal with the variations in handwriting. However, it still has some drawbacks. The main drawback of this approach is the difficulty to adjust its parameters. For instance, in MLPs and TDNNs which are the most frequently used for handwriting recognition, the number of layers and the number of neurons per layer have to be fixed, without any standard method to adjust these parameters. Most authors select the best values of those parameters heuristically, using their own training database.

4. Statistical classification: In this type of approach, each sample is represented by d features. The samples in the training database can be viewed as a distribution of points in a $d-$dimensional space. Statistical approaches aim at finding the boundaries to separate samples belonging to different classes and to group samples belonging to the same class. Support Vector Machines (SVM) is a statistical classifier that is very frequently used in the literature. SVM has provided successful results for isolated character/digit recognition systems [20, 6, 122, 87, 22, 12, 137].

In addition, the experimental results given in [1] have shown that SVM provides better results, compared to neural networks on different handwriting databases (see Table 3.1). Because of the effectiveness of SVM and because it is easier to tune its parameters (compared to NN), we decided to use this classifier in our system. In the following paragraphs, we give a brief introduction of SVM.

Table 3.1: Comparison between SVM and NN (MLP and TDNN) classifiers for on-line character recognition, given in [1]. These recognizers rely on the local on-line features extracted from each points of the on-line character signal (210 features).

Type of data	IRONOFF			UNIPEN			IRONOFF-UNIPEN		
	MLP	TDNN	SVM	MLP	TDNN	SVM	MLP	TDNN	SVM
Digit	98.2	98.4	**98.83**	97.5	97.9	**98.33**	97.9	98.4	**98.68**
Lowercase	90.2	90.7	**92.47**	92.0	92.8	**94.03**	91.3	92.7	**93.76**
Uppercase	93.6	94.2	**95.46**	92.8	93.5	**94.81**	93.0	94.5	**95.13**

SVM has been presented by Vapnik *et al.* [13, 19], originally to solve a bi-class classification problem. The main idea of the original (bi-class and linear) SVM is to find a decision hyperplane in the feature space, which is able to separate the samples belonging to the positive class from the samples belonging to the negative class. This hyperplane has to ensure the smallest error, by maximizing the margin between these two classes, as illustrated in Figure 3.29.

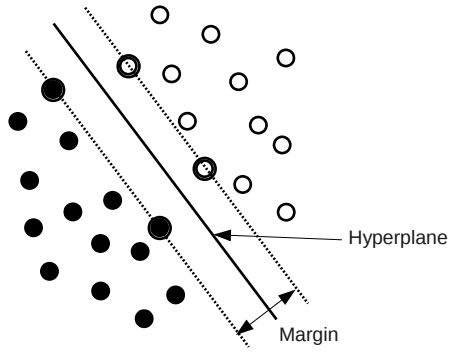


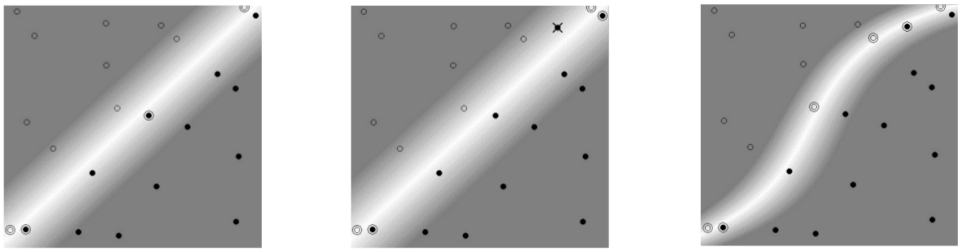
Figure 3.29: The main idea of SVM. The black samples belong to the negative class while the white samples belongs to the positive class.

There are actually two types of SVMs: linear and non-linear.

- Linear SVM seeks for a hyperplane to separate the classes. This method can be applied only to linearly separable data (see Figure 3.30(a)). In real applications, most of the time, the classes are not linearly separable, as illustrated in Figure 3.30(b).

In this example, we can see that a black point (in cross sign) belonging to the black class is located in the white class side. In order to deal with this problem, a slack variable is used to accept some errors (as few as possible) during training. The linear SVM is not frequently used in the literature. However, there are some recent works focusing on the linear SVM, due to its efficiency in the context of very large-scale data [57].

- Non-linear SVM is an extension of the linear SVM. This method uses a non-linear kernel, by mapping the features of the input data onto a higher dimensional feature space (possibly of infinite dimensions). As illustrated in Figure 3.30(c), the non-linear separable data in Figure 3.30(b) are separable by a degree 3 polynomial kernel function. Different kernel functions have been presented in the literature [19]: linear, polynomial, radial basis function, Sigmoid, etc.



(a) Linearly separable data

(b) Non-linearly separable data

(c) Degree 3 polynomial kernel

Figure 3.30: An example of data distribution extracted from [19]. a) the classes are linearly separable. b) the classes are non-linearly separable. c) the non-linearly separable class in (b) are separable by a degree 3 polynomial kernel.

Non-linear SVM is the most frequently used in the literature, compared to linear SVM. Choosing a kernel function adapted to the recognition problem is a hard task. However, in the literature, SVM relying on the Radial Basis Function (RBF) kernel [58] is the most frequently used for different recognition problems. In general, it provides good recognition results. Therefore, we decided to use an RBF SVM in our system.

We have to mention that, SVM classifier is originally designed to solve bi-class classification problems. In order to deal with a multi-class classification problems (for instance $M - class$ classification problem for our SCR), different methods have been proposed in the literature [150, 16, 84, 98, 77, 108]. Chih-Wei Hsu and Chih-Jen Lin [59] proposed a comparison of 4 different strategies: all-together, one-against-all, pairwise, DAGSVM. A brief description of these 4 methods is given below:

- All-together [150]: considers all the M classes together and tries to find M optimal hyperplanes to separate all these classes by using only one decision function. Figure 3.31 illustrates an example (presented [150]) for a 4 classes classification problem by applying this strategy.

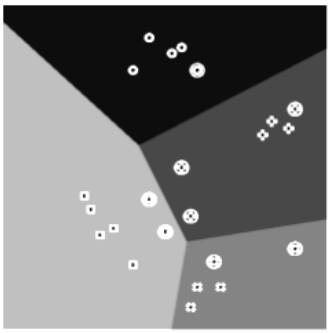


Figure 3.31: An example of all-together strategies to classify 4 classes of data, extracted from [150].

- One-against-all [16]: relies on a similar idea as the all-together method. Unlike the all-together strategy, this strategy tries to find M optimal hyperplanes to separate the M classes, but by solving M problems (*i.e.* training M binary SVM models) separately. The m^{th} SVM is trained by considering the samples belonging to class c_m as a positive class and the samples belonging to all other classes as a negative class.

- Pairwise [84]: is also known as one-against-one method. It aims at creating $\frac{M(M-1)}{2}$ $bi-class$ SVM models where each SVM model is trained using the samples belonging to the corresponding two classes. Considering a $4-class$ problems, 6 $bi-class$ SVM models are created: c_1 *vs.* c_2, c_1 *vs.* c_3, c_1 *vs.* c_4, c_2 *vs.* c_3, c_2 *vs.* c_4 and c_3 *vs.* c_4. During the recognition step, the input sample is submitted to all the SVM models. For the final decision, different strategies can be used, as for instance, voting strategy, where the input sample belongs to the class that gets the maximum number of votes.

- Directed Acyclic Graph SVM (DAGSVM) [108]: relies on the same training step as the one-against-one method. $\frac{M(M-1)}{2}$ $bi-class$ SVM models are created. During the recognition step, these $\frac{M(M-1)}{2}$ SVM models are organized as a binary directed acyclic graph of $\frac{M(M-1)}{2}$ nodes and M leaves where each leaf corresponds to each class label, as illustrated in Figure 3.32. The input sample is submitted to the root node. It is then submitted to the SVM model in the following upper or lower node, depending on the output provided by the SVM at the root node. This process continues until it reaches a leaf of the graph, which indicates its predicted label class. The main advantage of the DAGSVM method compared to the Pairwise method is related to the computational time during the recognition step. Indeed, the input sample is not submitted to all the $bi-class$ SVM since the use of DAG allows pruning the "unnecessary" nodes (bi-class SVM models) in the graph. However, the main drawback of this method is related to the fact that the effectiveness of the system may depend on the structure of the Graph, considering

the classification error of the $bi-class$ SVM model in each node. Error in any node of the Graph directly generates classification error in the overall classifier.

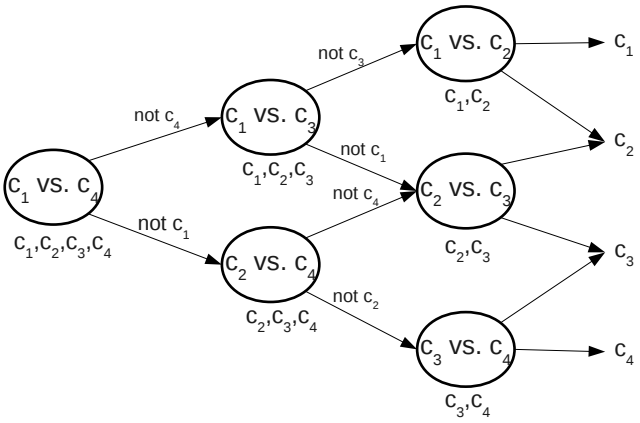


Figure 3.32: Example of the DAGSVM for a $4-class$ classification problem.

According to the experimental results in [59], the one-against-one method provides better results, compared to other methods for different recognition problems. For this reason, the one-against-one method is used in our system. In a near future, we will investigate different kernel types and multi-class strategies.

In addition, SVM was originally designed to predict only the class label of the input data. The decision value given by SVM does not allow to describe or directly predict posterior probability. But, in real applications, the posterior probability is often required for post-processing or to feed the next step of the system. For instance, in our system, SVM classifier is used to recognize each node in the lattice. The posterior probabilities given by the SVM have to be estimated, since they are used to feed the word decoding process (see section 3.7). In the literature, a large amount of research has been dedicated to probability estimation for bi-class and multi-class SVM. In our system, we use the probability estimation method presented by T.F. Wu *et al.* [154].

3.5.3 Single character recognition system with rejection

As mentioned earlier in this section, a group of graphemes contained in a given lattice node can correspond to an unknown pattern. In many cases, unknown patterns are associated with the lowest SVM output values. However, the range of the SVM output values depends on data distribution inside the classes and, even after estimating class probabilities from the SVM outputs [26, 154], it is difficult to settle a threshold for precisely detecting unknown patterns. In order to deal with this problem, we experimented two methods. The first method consists in adding a garbage class to the SVM, and the second method consists in using a rejection system based on a cascade of Adaboost classifiers to refine the outputs given by the SVM.

3.5.3.1 Garbage class in the SVM

Let us consider that we have a classification problem with M classes. This method consists in adding an additional class (*i.e.* garbage class) to represent unknown patterns. Hence, the SVM classifier will deal with $M+1$ classes instead of M classes. The training data of the garbage class has to be chosen carefully because it has to be representative of all the possible unknown shapes. Therefore, the garbage class has to contain very numerous samples. As detailed in section 4.1.6, we use diverse unknown shapes segmented from handwritten words for the training data of this class.

3.5.3.2 Rejection system based on a cascade of Adaboost classifiers

Given a classification problem of M classes ($C = \{c_1, c_2, \ldots, c_M\}$), this method aims at creating a set of M rejectors $R = \{r_1, r_2, \ldots, r_M\}$, inspired from the method introduced by Viola and Jones which was originally used for face detection [144]. Each rejector $r_m \in R$ is a specific rejector for the class $c_m \in C$ and is used to refine the outputs of the SVM, as illustrated in Figure 3.33.

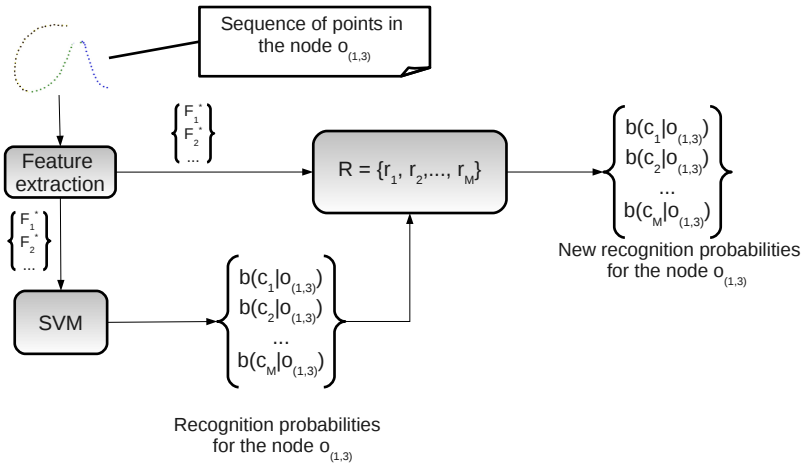


Figure 3.33: The use of the rejection system to refine the recognition probabilities provided by SVM.

Each rejector r_m relies on a cascade of Adaboost classifiers. At the opposite of many existing algorithms [29, 37, 155, 46], this method does not rely on *a posteriori* probabilities (*i.e.* the outputs of the classifier). Alternatively, it aims at sequentially rejecting the negative samples by analyzing the input data, independently of the *a posteriori* probabilities provided by the classifier.

A rejector r_m for the character class c_m is a cascade of specific Adaboost classifiers $\{\phi_m^1, \phi_m^2, \ldots, \phi_m^K\}$, where K is the number of Adaboost classifiers in the cascade. The

training process and application of a rejector r_m are presented in the following paragraphs.

a) Training process

Each Adaboost classifier ϕ_m^k as well as the rejector r_m are trained by considering the samples belonging to the class c_m as the positive samples (*i.e.* positive class) and the samples belonging to all other classes and the unknown class as the negative samples (*i.e.* negative class). Let us denote

- PS the set of training samples of the positive class
- NS the set of training samples of the negative class

The training process of a rejector r_m can be described as follows (see Figure 3.34): the first Adaboost ϕ_m^1 is trained by using all the training samples ($PS \cup NS$). The second Adaboost of the cascade ϕ_m^2 is trained using all the positive samples and the non rejected negative samples ($PS \cup \{NS - ns^1\}$) from the Adaboost ϕ_m^1, and so on, where ns^1 represents the set of negative samples correctly rejected by the Adaboost classifier ϕ_m^1 and so on. Each Adaboost ϕ_m^k is a weak classifier which relies on few features selected from the feature set provided by the feature selection stage (see section 3.4.3). New rejectors ϕ_m^k are sequentially added until the rejection rate reaches an objective false positive rate γ or no additive negative sample can be rejected.

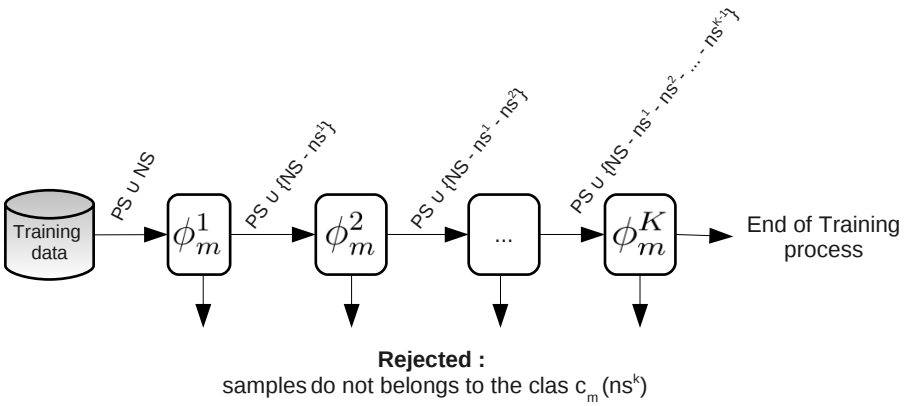


Figure 3.34: Training process of a rejector r_m.

b) Application

The rejector r_m is used as follows during the recognition stage (see Figure 3.35):

Given a new node $o_{(t,t')}$, its feature set is submitted to the rejector r_m of the character c_m; it is sequentially presented to all the Adaboost classifiers $\phi_m^{k=1...K}$ in the cascade until one of them classifies it as negative class (*i.e.* rejected). If none of these Adaboost rejects $o_{(t,t')}$, then $o_{(t,t')}$ belongs to the class c_m. Otherwise, $o_{(t,t')}$ does not belong to the class c_m.

- If $o_{(t,t')}$ does not belong to the class c_m, then the output probability $b(c_m|o_{(t,t')})$ given by the classifier (here SVM) is decreased as follows: $b(c_m|o_{(t,t')}) = \frac{b(c_m|o_{(t,t')})}{e^{Pr_m}}$ where Pr_m is the rejection score given by the rejector r_m.
- If $o_{(t,t')}$ is not rejected from the class c_m (belonging to the class c_m), then the output probability $b(c_m|o_{(t,t')})$ is not modified.

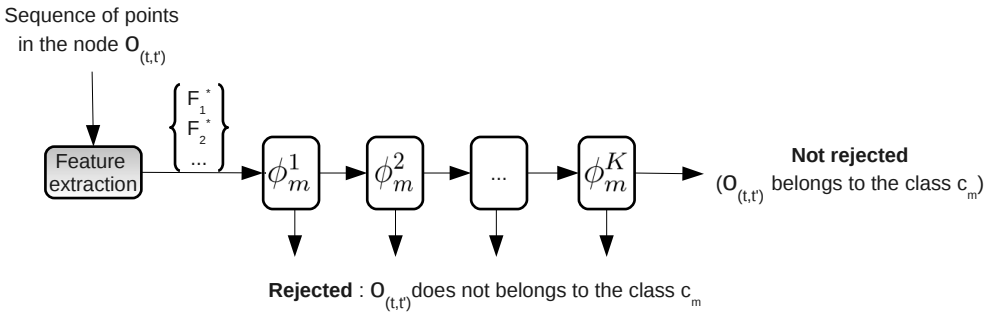


Figure 3.35: A specific rejector r_m based on a cascade of Adaboost classifiers $\{\phi_m^1, \phi_m^2, \ldots, \phi_m^K\}$ for the character class c_m.

It is important to mention that the effectiveness of each rejector r_m depends on 3 important parameters given by the user:

- The objective false positive rate f_k: the value of f_k indicates the percentage of negative samples accepted by the k^{th} Adaboost classifier ϕ_m^k in the cascade of the rejector r_m. For instance, in the case where $f_k = 20\%$, the Adaboost classifier ϕ_m^k accepts up to 20% of negative samples classified as positive class. In other word, ϕ_m^k has to be able to reject at least 80% of the negative samples.
- The objective true positive rate d_k: the value of d_k indicates the percentage of positive samples that has to be correctly recognized as positive class. There is of course a strong dependence between the parameters f_k and d_k.
- The objective false positive rate γ of the overall rejection system: the value of γ indicates the percentage of negative samples accepted by each rejector r_m. For instance, if $\gamma = 1\%$, it means that the rejector r_m is able to reject at least 99% of the negative samples.

An experimental comparison of these two strategies for rejection (SVM with garbage class and rejection system based on cascade of Adaboost classifiers) is given in section 4.2.3.1.

3.5.4 Conclusion

This section introduces a SCR relying on a RBF-based SVM classifier and the features selected from 14 families of both on-line and off-line features. This system is used to generate character hypotheses for each node in the lattice, and feed the word decoding process presented section 3.7.

In order to deal with lattice nodes corresponding to unknown patterns, we implemented a SCR with rejection. Two methods are presented. The first method consists in creating a RBF-based SVM classifier with rejection by adding a garbage class. The second method uses specific rejectors to refine the outputs given by the SVM classifier. Some experiments presented in section 4.2.1.2 compare the effectiveness of these two methods.

This SCR relies on a SVM which is known as a discriminative classifier. In order to reduce the computational time of the system, only the N candidates with highest probabilities given by the SCR are considered in the next step. The lowest recognition probabilities are considered as fruitless candidates. The value of N will be experimentally fixed according to the experimental results of the SCR (see section 4.2.1.2) to ensure that the effectiveness of the system is not degraded.

3.6 Bi-character analysis

3.6.1 Objective

Characters composing unconstrained handwritten words are naturally more or less connected. These connections have a great impact on the shapes of the characters. If these characters are analyzed independently, the system may face some confusing problems. Indeed, the shape of one character can change depending on the characters written on its left and right, as earlier explained in section 1.3.1. In addition, some characters may share parts with some other characters (see section 1.3.4). To solve this problem, we propose the idea of the bi-character analysis, which aims at taking into account the graphical context by jointly analyzing each pair of neighboring nodes in the lattice.

In the literature, we can notice some similar ideas. For instance, A.L. Bianne *et al.* [9] have introduced a context-dependent recognition system relying on HMM models. Each character class of the alphabet is modeled by different HMM character models, according to its surrounding characters (*i.e.* characters written on its left and right, see section 2.3). This system is able to deal with variations in the character shape. However,

this method does not allow to take into account the graphical context (see section 1.4) of the neighboring characters. Hence, it may not be able to overcome the problem of "shared character part" (see section 1.3.4).

Dan Ciresan [30] introduced an off-line numeral string recognition system based on single and two-digit classifiers. This method relies on a pure explicit segmentation method. Each segmented component is considered as one digit and two-digit candidates which are submitted respectively to a single digit classifier and a two-digit classifier. Then, the system compares the outputs given by both systems and selects the best one according to some rules. The overall system was earlier explained in section 2.3.2.1. The two-digit classifier introduced in [30] allows to overcome the under-segmentation problem (*i.e.* when some segmented components contain more than one character/digit) which is usually happening when using pure explicit segmentation methods. Since this system relies on a pure explicit segmentation method, it still faces the problems due to over or under segmentation. For instance, 1) if the segmented components contain only a part of digit, 2) if the segmented components contain one digit plus a part of another digit, 3) if the segmented components contain more than two digits, etc.

In our system, we introduce bi-character models for on-line HWR. Unlike [30], our system relies on an explicit segmentation/recognition method. As a consequence, the bi-character models are designed to deal with the problems happening when the input word signal/image is over-segmented (*i.e.* one character may be segmented into many graphemes), which occurs particularly in the context of the character shared path problem. In addition, the number of bi-character classes (676 classes $= 26 * 26$) is much more important than the number of two-digit classes (100 classes) presented in [30]. Hence, the bi-character models used in our system has to be able to deal with a classification problem with a large number of classes.

In our system, the bi-character models allow recognizing pairs of neighboring characters by jointly considering every pair of neighboring nodes in the lattice in order to validate/invalidate the hypotheses given by the SCR (see section 3.5), during the word recognition process (see section 3.7). We have to mention here that two additional problems may be also handled by using bi-character models: the problem of similarity between characters (see section 1.3.3) and the problem of unknown patterns (in some nodes of the lattice). The following sub-sections explain how these three problems can be handled by the bi-character models. Then, we explain how the bi-character models can be integrated in the handwriting word recognition system. Finally, we introduce the method used to build the bi-character models.

3.6.1.1 Character shared part problem

Let us reconsider the example presented earlier in section 1.3.4. Given a handwritten signal of the word "*de*", the word decoding process provides different recognition paths where each path corresponds to a word in the lexicon. Suppose that there are only two

best paths, as illustrated in Figure 3.36.

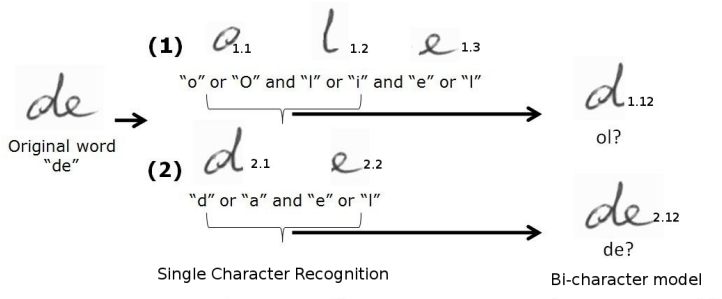


Figure 3.36: Bi-character models allow solving the character shared part problem.

In the path (1), which is the most likely according to the SCR, the input signal could be recognized as the sequence of characters "*ole*" or "*oll*" or "*oil*" and so on. Indeed, the shapes 1.1 and 1.2 are parts of the character '*d*'. If they were observed individually, the visual appearance of the shape 1.1 is very close to character '*o*' or '*O*'. Hence, it may be recognized by the SCR as character '*o*' or '*O*' with a high probability. Idem for the shape 1.2, which may be recognized as character '*i*' or '*l*' with a high probability.

We can consider that this problem can be handled by using a lexicon. But in that case, the effectiveness of the system would strongly depend on the lexicon. In addition, the lexicon cannot cover all the problems, as for instance between the words "*while*" and "*white*". A complex language model(s) is therefore required to solve this kind of problems, but could not anyway solve all the possible ambiguities.

By using bi-character models, we concatenate the shapes 1.1 and 1.2, and obtain a new shape 1.12. This new shape is submitted to the bi-character models of "*ol*", "*oi*", "*Ol*" and *Oi*". It should be rejected by these models, since its visual appearance is very different from the visual appearance of "*ol*","*oi*", "*Ol*" and "*Oi*". As a consequence, the path (2) will become more likely, since the concatenation of shapes 2.1 and 2.2 (shape 2.12) is very close to the visual appearance of "*de*". Finally, the input signal will be recognized correctly as "*de*".

3.6.1.2 Similarity between different character classes

As explained earlier in section 1.3.3, in the Latin alphabet, some characters are very similar when we try to recognize them individually, *e.g.* characters '*e*' and '*l*'. They are very similar in visual appearance and writing trajectory. Sometimes, they are hard to be distinguished even by a human being. Furthermore, in some cases, the writings in lowercase and uppercase are very similar, as *e.g.* the character '*o*'.

Taking the example illustrated in Figure 3.37, the visual appearance of the shape 1 (respectively shape 2) is very similar to the character '*e*' or '*l*'. Hence, the SCR would give high output values for the shape 1 as character '*l*', and for the shape 2 as character '*e*'. Indeed, the shape 1 corresponds to the character '*e*' and the shape 2 corresponds to the character '*l*'.

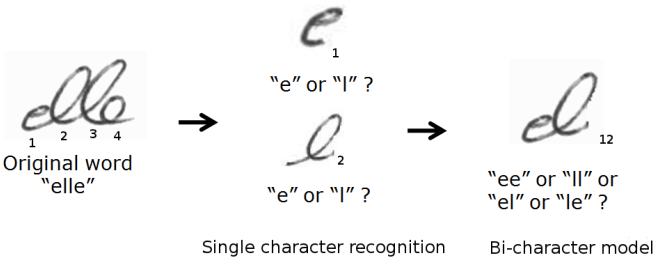


Figure 3.37: An example of the ambiguity between the characters '*e*' and '*l*'.

By concatenating these two shapes, we obtain a new shape 12. Its visual appearance is very different from the visual appearances of the pairs of characters "*ee*", "*ll*" and "*le*". Hence, there is a high probability that they will be rejected by these bi-character models. As a consequence, our system based on bi-character models may recognize the shape 12 as "*el*".

3.6.1.3 Unknown pattern

Beside the two problems explained in the previous sections, the bi-character models also permit to solve the problem concerning unknown patterns. As mentioned in section 3.5.3, every node in the lattice contains a combination of graphemes which are segmented from the input handwritten word. As a consequence, some nodes may correspond to unknown patterns since they do not belong to any character class in the alphabet. To deal with this kind of problems, we have proposed to use a SCR with rejection option at the character analysis level. However, by individually considering each node of the lattice, some unknown patterns may not be handled at the character analysis level. Therefore, using the bi-character models is another opportunity to handle the unknown pattern problem, since the models will jointly analyze each pair of neighboring nodes.

3.6.2 Bi-character models

As earlier mentioned in section 3.3, each node $o_{(t,t')}$ in the lattice is considered as a potential character and submitted separately to a SCR. This system provides a list of N potential character candidates, each of which is described by the estimated *a posteriori* probability of recognition as a class c_m $(b(c_m|o_{(t,t')}))$. In order to validate/invalidate these results, the graphemes in the pair of neighboring nodes are combined into only

one signal to be submitted to the corresponding bi-character models. To illustrate this description, an example is given hereafter.

Example: let us take the example of the neighboring nodes $o_{(1,2)}$ and $o_{(3,4)}$ in the lattice created in section 3.3.3 (see Figure 3.38). Considering that the 2 potential character candidates ($N = 2$) given by our SCR for node $o_{(1,2)}$ are: $b(c_m = \text{`}o\text{`}|o_{(1,2)})$, $b(c_m = \text{`}a\text{`}|o_{(1,2)})$ and for node $o_{(3,4)}$ are: $b(c_m = \text{`}u\text{`}|o_{(3,4)})$, $b(c_m = \text{`}v\text{`}|o_{(3,4)})$. At the level of bi-character analysis, these two nodes are concatenated. We obtain a new group of graphemes $(o_{(1,2)} \cup o_{(3,4)})$. This one is submitted to the bi-character models $(B_{(\text{"}ou\text{"})}, B_{(\text{"}ov\text{"})}, B_{(\text{"}au\text{"})}, B_{(\text{"}av\text{"})})$ that return recognition probabilities/scores that the group of graphemes $(o_{(1,2)} \cup o_{(3,4)})$ is recognized as the pairs of characters "ou", "ov", "au" and "av".

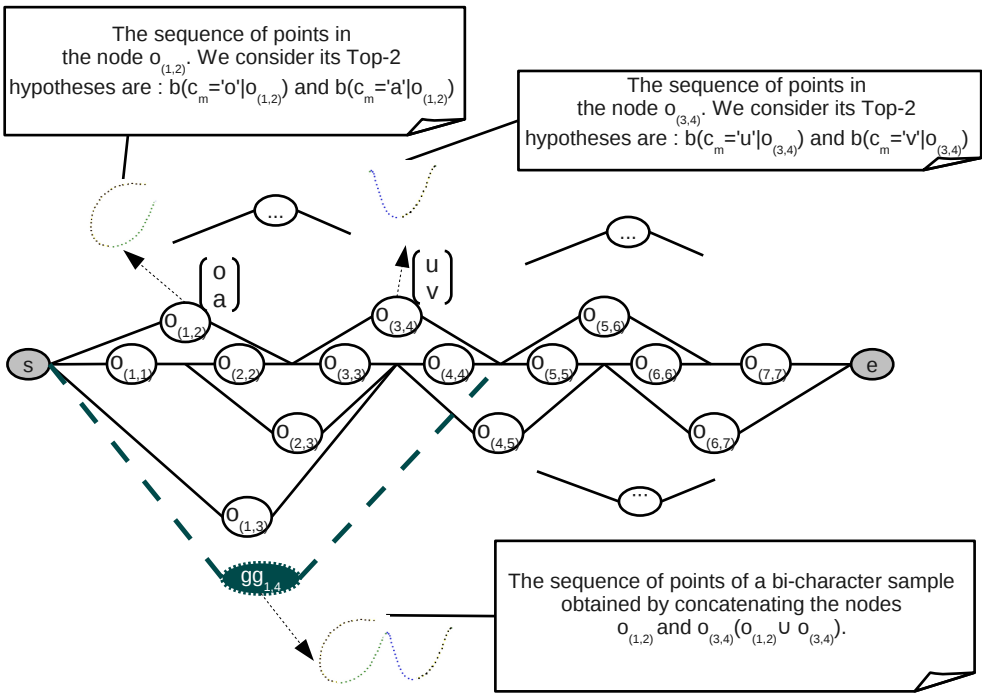


Figure 3.38: Inputs of bi-character models (from the lattice).

We have to mention two important points:

1. To have an ideal model, we should use an adapted set of features by applying a feature selection algorithm for each bi-character model using the feature selection method explained in section 3.4.3. However, technically, a bi-character sample may be obtained by concatenating different pairs of neighboring nodes. For instance, in Figure 3.38, the group of graphemes $gg_{1,4}$ represents the concatenations $(o_{(1,2)} \cup o_{(3,4)})$, $(o_{(1,1)} \cup o_{(2,4)})$ and $(o_{(1,3)} \cup o_{(4,4)})$. The group of graphemes

$gg_{1,4}$ will be used to validate or invalidate the pair of nodes: ($o_{(1,2)}$ *and* $o_{(3,4)}$), ($o_{(1,1)}$ *and* $o_{(2,4)}$) or ($o_{(1,3)}$ *and* $o_{(4,4)}$). The bi-character models used to verify each pair of nodes are selected according to the N potential character candidates (hypotheses given by SCR) in each node, as explained in the example above.

In order to avoid re-computing the set of features for each bi-character model, we decided to use the same set of features for all the models. Therefore, the feature set computed from the group of graphemes $gg_{1,4}$ can be used to validate/invalidate all the hypotheses for the neighboring nodes: ($o_{(1,2)} \cup\ o_{(3,4)}$), ($o_{(1,1)} \cup\ o_{(2,4)}$) and ($o_{(1,3)} \cup\ o_{(4,4)}$).

2. Furthermore, if the node contains more than two graphemes, its group of graphemes is also considered as a bi-character shape. This shape will be used to validate or invalidate a pair of neighboring nodes at a lower level. For instance, the group of graphemes $gg_{1,3}$ in the node $o_{(1,3)}$ is considered as a bi-character shape to validate or invalidate the pairs of nodes ($o_{(1,1)}$ *and* $o_{(2,3)}$) and ($o_{(1,2)}$ *and* $o_{(3,3)}$). We decided to create the bi-character models using the same features set as the SCR (see sections 3.4 and 3.4.3) as so, the features computed and used in the SCR can be directly reused in the bi-character models which saves computational time.

In the context of bi-character models, the number of classes to be recognized is very important compared to the SCR, since there are $26 * 26 = 676$ bi-character classes in the case of Latin alphabet (lowercase and without accent character classes). As explained above, the bi-character models are used for verification (*i.e.* authentication) to validate or invalidate the outputs of the SCR. For these reason, only the one-against-all strategy is needed. The models $B_{c_ic_j}$ of a bi-character class c_ic_j is created by considering that samples belonging to the class c_ic_j are the positive samples (*i.e.* positive class, $y = 1$) and the samples belonging to all other classes are the negative samples (*i.e.* negative class, $y = 0$), where $c_i, c_j \in \{a, b, c, \ldots, z\}^2$. This problem becomes a binary classification problem. Therefore, only bi-class classifier is needed to build these bi-character models.

As mentioned earlier in section 3.5.2, SVM is among the best statistical classifiers and was originally designed for bi-class classification problem. In general, there are two types of SVMs (see section 3.5.2). Non-linear SVM using a kernel function usually provides very promising results in various classification problems including isolated character/digit recognition problems [20]. However, this method is not adapted for our bi-character models since the storage of one model using kernel SVM require 15Mo storing space. Therefore, in order to store all the 676 models, up to 10Go of storing space would be required. Furthermore, for loading such a huge amount of data, the system requires a big size of Random-Access Memory. Due to this resource requirement, such a system cannot be used on an ordinary computer nowadays (and therefore cannot be easily embedded in a tablet). To circumvent this problem, logistic regression is used to create these bi-character models since the logistic regression is specifically designed to solve a bi-class classification problem in the case of large-scale data. In addition, the

logistic regression has been successfully used in different fields such as medical, banking.

We give a brief introduction of Logistic Regression in the following paragraph. Then, we explain how to use the Logistic Regression to build the bi-character models.

Logistic Regression Given an input data $\boldsymbol{X}$ to be recognized, the probability can be estimated by a logistic function, as in Equation (3.8).

$$P(y = \pm 1|\boldsymbol{X}, \boldsymbol{\beta}) \;=\; \frac{1}{1 + \exp^{-y(\beta^T \boldsymbol{X} + b)}} \tag{3.8}$$

where y is the class label and b is the bias, which is often solved by adding an additional dimension to the data: $X \leftarrow [X, 1]$ and $\boldsymbol{\beta} \leftarrow [\boldsymbol{\beta}, b]$. The Equation (3.8) can be denoted as:

$$P(y = \pm 1|\boldsymbol{X}, \boldsymbol{\beta}) \;=\; \frac{1}{1 + \exp^{-y(\boldsymbol{\beta}^T \boldsymbol{X})}} \tag{3.9}$$

where $\boldsymbol{\beta}$ is the vector of regression coefficients which is obtained by a training process. This training process can be briefly explained as below:

The logistic regression assumes the training data as a binomial distribution. Given a training database of K samples: $(\boldsymbol{X}_1, y_1), (\boldsymbol{X}_2, y_2), \ldots, (\boldsymbol{X}_K, y_K)$, where $\boldsymbol{X}_k = \{x_k^1, x_k^2, \ldots, x_k^d\}$, which consists in a set of d features that represents the sample k and $y_k = \{1, 0\}$ represents the label of the sample k (1: positive class, 0 negative class). The training process aims at estimating the optimal vector of the regression coefficients $\boldsymbol{\beta}$ by minimizing the negative log-likelihood, as denoted in Equation (3.10) [86].

$$arg\min_{\boldsymbol{\beta^*}} \Big[- \textstyle\sum_{k=1}^{K} log \frac{1}{1+\exp^{(-y_k \boldsymbol{\beta}^T \boldsymbol{X}_k)}} \Big] \tag{3.10}$$

$$= arg\min_{\boldsymbol{\beta^*}} \big[- \textstyle\sum_{k=1}^{K} [log(1) - log(1 + \exp^{(-y_k \boldsymbol{\beta}^T \boldsymbol{X}_k)})] \big]$$

$$= arg\min_{\boldsymbol{\beta^*}} \big[\textstyle\sum_{k=1}^{K} log(1 + \exp^{(-y_k \boldsymbol{\beta}^T \boldsymbol{X}_k)}) \big] \tag{3.11}$$

To get a smooth training, the regularization variable $\frac{1}{2}\boldsymbol{\beta}^T \boldsymbol{\beta}$ can be added. Equation (3.11) can be denoted as:

$$arg\min_{\boldsymbol{\beta^*}} \big[\tfrac{1}{2} \boldsymbol{\beta}^T \boldsymbol{\beta} + C \textstyle\sum_{k=1}^{K} log(1 + \exp^{(-y_k \boldsymbol{\beta}^T \boldsymbol{X}_k)}) \big] \tag{3.12}$$

where $C > 0$ is a parameter given by the user to balance the two parts of the Equation (3.12).

Bi-character models using Logistic Regression

In our context, the model of each bi-character class $c_i c_j$ ($B_{c_i c_j}$) is trained using the optimization function presented in equation 3.12. We obtain the corresponding regression coefficient vector $\boldsymbol{\beta}_{c_i c_j}$. The verification stage works as below:

Let us consider the example in Figure 3.38, where $\boldsymbol{X}$ is the concatenation of the neighboring nodes $o_{(1,2)}$ and $o_{(3,4)}$ ($\boldsymbol{X} = (o_{(1,2)} \cup o_{(3,4)})$). The label to be verified is "au". Hence, the recognition score $a(\text{"}au\text{"}|o_{(1,2)} \cup o_{(3,4)})$, denoted as:

$$a(\text{"}au\text{"}|o_{(1,2)} \cup o_{(3,4)}) = P(\text{"}au\text{"}|\boldsymbol{X}, \boldsymbol{\beta}_{\text{"}au\text{"}}) \;=\; \frac{1}{1 + \exp^{\delta_{\text{"}au\text{"}}(\boldsymbol{X})(\boldsymbol{\beta}^T_{\text{"}au\text{"}}\boldsymbol{X})}} \quad (3.13)$$

where $\delta_{\text{"}au\text{"}}(\boldsymbol{X}) = 1$ if $\boldsymbol{X}$ belongs to the class "au", and 0 otherwise. Here, the ground-truth is known as we use the bi-character models to validate/invalidate the recognition results given by the SCR.

3.6.3 Conclusion

This section introduces the original idea of bi-character models. These models allow recognizing pairs of characters by jointly considering every pair of neighboring nodes in the lattice. The objective is to solve three problems: shared character part, similarity between different character classes, and unknown patterns. In our system, the bi-character models are built using logistic regression.

In order to find the most probable words associated with an input signal, we must combine the SCR and the bi-character models during the word decoding process detailed in the next section.

3.7 Word decoding process

As mentioned earlier in section 3.1, each node in the lattice (see section 3.3.3) is analyzed by a SCR (see section 3.5). The results at the character level are further verified by the bi-character models (see section 3.6), by taking into account pairs of neighboring nodes.

In the word decoding step, a word decoding method is applied to the lattice, to find the most probable words in a given lexicon for each lattice. In our work, two word decoding strategies are used. The first strategy relies on a directed graph search method (see section 3.7.2), while the second strategy relies on dynamic programming (see section 3.7.3).

As with other systems in the literature, external knowledge is required in order to improve the effectiveness and the efficiency of the system (see section 2.2.5). In this experiment, we use a lexicon which is classical for HWR problem. Before introducing our two decoding methods, the lexicon structure is introduced in the following section.

3.7.1 Lexicon TRIE model

A lexicon is generally used for HWR, in order to enhance their effectiveness and efficiency, by integrating knowledge about the language in the recognition process. To integrate a lexicon in the system, different methods can be used (see section 2.2.5).

In our context, we use a prediction method to guide the decisions during propagation in the lattice. The lexicon is represented by a TRIE model (also known as prefix tree). Each node at step l points to the list of all possible characters at step $l+1$. Figure 3.39 illustrates an example of a TRIE that represents a lexicon containing 6 words ("*au*","*en*","*cinq*","*ou*","*une*","*unis*").

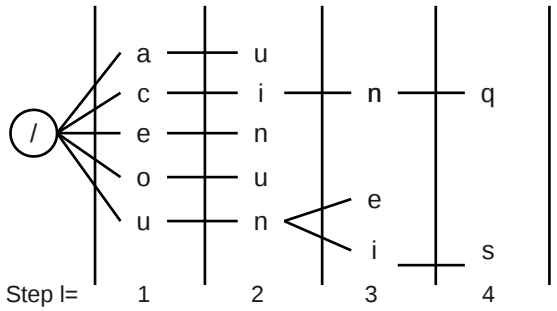


Figure 3.39: TRIE model representing a lexicon that contains 6 words ("au", "en", "cinq", "ou", "une", "unis")

Let us denote:

- D: lexicon tree
- D_l: set of candidate characters at step l
- d_l: a candidate character at step l, ($d_l \in D_l$)

The prediction is performed as follows:

- All the possible characters at step 1 ($l = 1$) is: $D_1(\emptyset) = \{`a`, `c`, `e`, `o`, `u`\}$
- All the possible characters at step 2 ($l = 2$) where $d_1 = `u`$ is: $D_2(`u`) = \{`n`\}$
- All the possible characters at step 3 ($l = 3$) where $d_1 = `u`, d_2 = `n`$ is: $D_3("un") = \{`e`, `i`\}$.

During the word decoding process, at step l, the search space at step $l + 1$ will be restricted to the character candidates given by the lexicon at step $l + 1$ ($D_{(l+1)}$).

Due to a lack of time, the TRIE model will be integrated only with our first method, directed graph search (see section 3.7.2). In our second method (see section 3.7.3), we use a flat search strategy where the decoding process is applied word by word for every word in the lexicon. In both methods, only the paths corresponding to words in the lexicon are decoded, and therefore, our system is lexicon dependent whatever the word decoding method used.

3.7.2 Directed graph search method

This method aims at searching the most probable paths in the lattice (based on the outputs of the SCR, as well as the bi-character models) associated to each word w_i in the lexicon. This search method relies on a graph search similar to Beam-Search algorithm where only E best candidates are considered at each stage of propagation in the graph. Graph corresponds to the lattice in our system. The terms "directed" refers to the fact that the propagation in the graph is performed in only one direction (from left-to-right for the left-to-right handwriting). A TRIE model D of the lexicon (see section 3.7.1) is used to predict a list of possible candidate characters during the search process. Each path from the start node to the end node corresponds to a word in the lexicon.

Let us denote the lattice as below:

- T: number of graphemes
- L: maximum level of the lattice. It also refers to the maximum number of graphemes composing a character.
- $o_{(t,t')}$: a node in the lattice, where t and t' indicate respectively the starting and the ending indexes of the graphemes in the node and $t \leq t'$ and $1 \leq t, t' \leq T$.
- $o_{(1,t)}$: is the starting nodes of the lattice (starting grapheme = 1), where $t = 1$ to L.
- Nodes following the node $o_{(t,t')}$ are $o_{(t'+1,t'+l)}$, where $l = 1$ to L and $t' + l \leq T$;
- $b(c_m|o_{(t,t')})$: estimated probability that the node $o_{(t,t')}$ is recognized as character class c_m. This estimated probability is given by the SCR (see section 3.5).

- N: number of Top-N hypotheses in each node, given by the SCR.
- $a(c_m c_n | o_{(t,t')} \cup o_{(t'+1,t")})$: estimated probability that the pair of neighboring nodes $o_{(t,t')}$ and $o_{(t'+1,t")}$ is recognized by the bi-character model $B_{c_m c_n}$ (see section 3.6) as the pair of character $c_m c_n$.

The word search process is in two steps. To illustrate this searching strategy, let us start with an introduction example. A more formal description will be provided after.

Example:

Let us consider that we have an input signal containing 7 segmented graphemes ($T = 7$). The corresponding 3 levels ($L = 3$) lattice is illustrated in Figure 3.40. At each node of the lattice, let us select only 3 potential character candidates ($N = 3$), as illustrated in Figure 3.41(a). The starting nodes in this lattice are $\{o_{(1,1)}, o_{(1,2)}, o_{(1,3)}\}$.

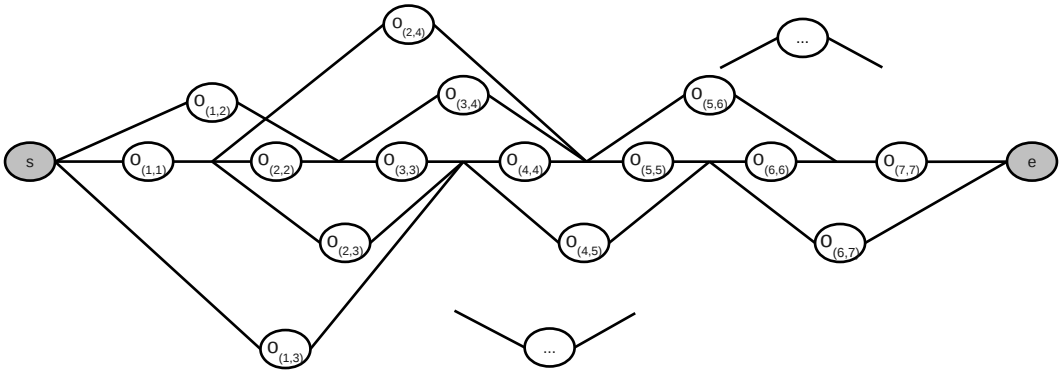
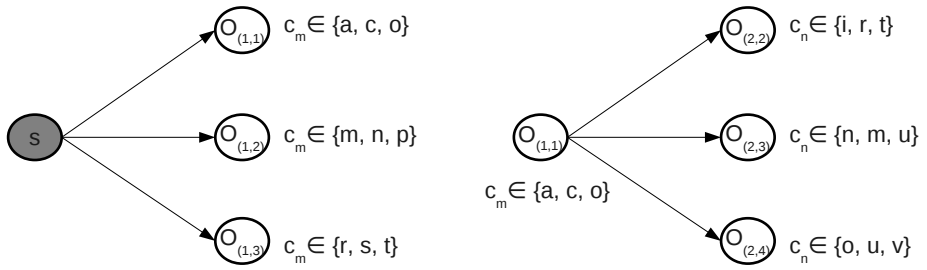


Figure 3.40: A lattice of $T = 7$ graphemes and $L = 3$ levels.

Suppose that we have a TRIE model of a lexicon as illustrated in Figure 3.39, the results of the search process are below:

Step 1: Let us take the starting node $o_{(1,1)}$ ($t = 1$) as an example. Nodes following the node $o_{(1,1)}$ are $\{o_{(2,2)}, o_{(2,3)}, o_{(2,4)}\}$. We suppose the $N = 3$ potential character candidates in each node as given in Figure 3.41(b). Some pairs of characters will not be considered since they do not satisfy the TRIE model (see condition 1 hereafter). For instance, this is the case of the pair of characters $c_m c_n =$ "ai", since there is no character $c_n =$ 'i' following the character $c_m =$ 'a' in the lexicon ($c_n =$ 'i' $\notin D_2(c_m =$ 'a') $= \{$'u'$\}$). The list WL_1, PL_1, NL_1 containing respectively the pairs of characters, their scores and the last visited nodes (supposing that they are ordered by score PL_1) are:



(a) Starting nodes (b) Nodes following the node $o_{(1,1)}$

Figure 3.41: The nodes extracted from the lattice in Figure 3.40. a) Example of starting nodes and their $N = 3$ potential character candidates. b) Nodes following the node $o_{(1,1)}$ and their $N = 3$ potential character candidates.

WL_1: list of sub-words	PL_1 : list of scores	NL_1: list of the last nodes visited
$c_m c_n =$ "au"	$b(c_m = \text{`}a\text{`}\|o_{(1,1)}) * b(c_n = \text{`}u\text{`}\|o_{(2,3)}) * a(c_m c_n =$ "au"$\|o_{(1,1)} \cup\ o_{(2,3)})$	$o_{(2,3)}$
$c_m c_n =$ "au"	$b(c_m = \text{`}a\text{`}\|o_{(1,1)}) * b(c_n = \text{`}u\text{`}\|o_{(2,4)}) * a(c_m c_n =$ "au"$\|o_{(1,1)} \cup\ o_{(2,4)})$	$o_{(2,4)}$
$c_m c_n =$ "ci"	$b(c_m = \text{`}c\text{`}\|o_{(1,1)}) * b(c_n = \text{`}i\text{`}\|o_{(2,2)}) * a(c_m c_n =$ "ci"$\|o_{(1,1)} \cup\ o_{(2,2)})$	$o_{(2,2)}$
$c_m c_n =$ "ou"	$b(c_m = \text{`}o\text{`}\|o_{(1,1)}) * b(c_n = \text{`}u\text{`}\|o_{(2,3)}) * a(c_m c_n =$ "ou"$\|o_{(1,1)} \cup\ o_{(2,3)})$	$o_{(2,3)}$
$c_m c_n =$ "ou"	$b(c_m = \text{`}o\text{`}\|o_{(1,1)}) * b(c_n = \text{`}u\text{`}\|o_{(2,4)}) * a(c_m c_n =$ "ou"$\|o_{(1,1)} \cup\ o_{(2,4)})$	$o_{(2,4)}$

Now, $c_m c_n =$ "au" and $c_m c_n =$ "ou" are full words in the lexicon. However, these found words ("au" and "ou") contain respectively only 3 and 4 graphemes out of the 7 graphemes in the input handwritten word. Therefore, they will be not added into the final list WL, PL, NL of possible words.

The list of $E = 3$ highest candidates with highest scores (WL_1^*, PL_1^*, NL_1^*) (considering that they are ordered by score) are:

WL_1^*: list of sub-words	PL_1^*: list of scores	NL_1^*: list of the last nodes visited
$c_m c_n =$ "au"	$b(c_m =$ 'a'$\|o_{(1,1)}) * b(c_n =$ 'u'$\|o_{(2,3)}) * a(c_m c_n =$ "au"$\|o_{(1,1)} \cup\ o_{(2,3)})$	$o_{(2,3)}$
$c_m c_n =$ "au"	$b(c_m =$ 'a'$\|o_{(1,1)}) * b(c_n =$ 'u'$\|o_{(2,4)}) * a(c_m c_n =$ "au"$\|o_{(1,1)} \cup\ o_{(2,4)})$	$o_{(2,4)}$
$c_m c_n =$ "ci"	$b(c_m =$ 'c'$\|o_{(1,1)}) * b(c_n =$ 'i'$\|o_{(2,2)}) * a(c_m c_n =$ "ci"$\|o_{(1,1)} \cup\ o_{(2,2)})$	$o_{(2,2)}$

For each sub-word $c_m c_n \in WL_1^*$, we continue to step 2. In the case of $c_m c_n =$ "au", the continuation is impossible since in the lexicon, there is not any character following after the sequence of characters "au" ($D_3(c_m c_n =$ "au"$) = \emptyset$, see condition 2, hereafter). Let us consider the case that $c_m c_n =$ "ci".

Step 2: By considering the case that $c_m c_n =$ "ci", we obtain its recognition score $P($"ci"$)$, its last character $c_n =$ 'i' and its last node visited $o_{(2,2)}$ given by the step 1. The nodes following the node $o_{(2,2)}$ are: $o_{(3,3)}, o_{(3,4)}$ and $o_{(3,5)}$. We consider their character candidates given in Figure 3.42.

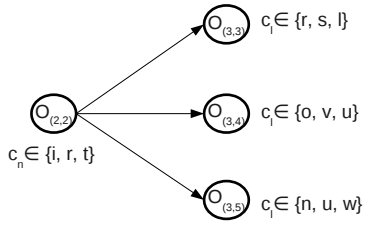


Figure 3.42: Nodes following the nodes $o_{(2,2)}$ and their candidate characters.

By taking into account the TRIE model condition (see condition 3 hereafter), there is only one character ('n') following the sequence of character "ci" ($D_3(c_m c_n =$ "ci"$) = \{$'n'$\}$). Therefore, the possible sub-words given by this stage (WL_2, PL_2 and NL_2) are:

WL_2: list of sub-words	PL_2: list of scores	NL_2: list of the last nodes visited
"cin"	$P(c_m c_n) * b(c_k =$ 'n'$\|o_{(3,5)}) * a(c_n c_k =$ "in"$\|o_{(2,2)} \cup\ o_{(3,5)})$	$o_{(3,5)}$

By taking the $E = 3$ best candidates with highest scores, we obtain $WL_2^* = WL_2, PL_2^* = PL_2$ and $NL_2^* = NL_2$. For each candidate in WL_2^*, we continue with step 2 recursively until the end of the graph (lattice).

Description of directed graph search method

Step 1: This step searches for a list of first pairs of characters WL_1 (and list PL_1 of their computed scores) from the starting nodes of the lattice. A list NL_1 is used to store the last visited nodes which allows indicating the positions of the propagation in the graph. This searching process is detailed hereafter. The results provided in this searching step will be further submitted to **Step 2**.

For each starting node $o_{(1,t)}$ ($t = 1$ to L) and their following nodes $\{o_{(t+1,t+l)}\}$ (where $l = 1$ to L and $t + l < T$), we search a list WL_1 of pairs of characters $c_m c_n$, where c_m and c_n are respectively the recognized characters at nodes $o_{(1,t)}$ and $o_{(t+1,t+l)}$. The pair of characters $c_m c_n$ have to exist in the TRIE model D of the lexicon. We verify the below condition:

condition 1:

- $c_m \in D_1(\emptyset)$: to verify if there is any word in the lexicon starting with the character c_m
- $c_n \in D_2(c_m)$: to verify if there is any word in the lexicon starting with the pair of characters $c_m c_n$

If the pair of characters $c_m c_n$ does not verify the condition 1, it will not be considered during the next propagation step. If $c_m c_n$ is a full word in the lexicon, and contains all the graphemes in the lattice, it will be added to the final list of words WL, and its recognition score is added to the list PL. Otherwise, if the $c_m c_n$ is a full word in the lexicon, but does not contain all the graphemes in the lattice (too short compared to the input handwritten word signal), it will not be added in the final list of word.

The recognition score $P(c_m c_n)$ can be computed from the SCR and the bi-character models outputs using Equation (3.14) and can be added to the scores list PL_1. The last visited nodes $o_{(t+1,t+l)}$ are stored in the list NL_1.

$$\begin{aligned} P(c_m c_n) = b(c_m|o_{(1,t)}) * \\ b(c_n|o_{(t+1,t+l)}) * \\ a(c_m c_n|o_{(1,t)} \cup \ o_{(t+1,t+l)}) \end{aligned} \tag{3.14}$$

To reduce the computational complexity, only the $E-best$ (giving the highest recognition scores) pairs of characters in the list WL_1 (and their corresponding elements in the list PL_1 and NL_1) are selected for the next step. We

obtain a new list WL_1^*, PL_1^* and NL_1^* of E elements. The value of the parameter E is experimentally settled.

For each pair of characters $c_m c_n \in WL_1^*$, we continue to step 2, if this pair of characters $c_m c_n$ verify the condition 2, as explained below.

condition 2: this condition is to verify is there any possible character(s) following the pair of characters $c_m c_n$ in the given lexicon. This condition is denoted as: $D_3(c_m c_n)! = \emptyset$.

Step 2 : from each element $c_m c_n \in WL_1^*$, we search the most probable paths (where each path is associated to one word in the lexicon) and their recognition scores. Each pair of characters $c_m c_n \in WL_1^*$ has a score $P(c_m c_n) \in PL_1^*$ and a last visited node $o_{(t',t")} \in NL_1^*$ (where $t' = t + 1$ and $t" = t + l$).

For all nodes $o_{(t"+1,t"+l)}$ (where $l = 1$ to L) following the node $o_{(t',t")}$ and all characters c_k in the candidate characters of the node $o_{(t"+1,t"+l)}$, we search a list of another E consecutive nodes and characters giving highest scores taking into account the lexicon (condition 3), as explained hereafter. From this new list of selected nodes and characters, we repeat step 2 recursively by taking into account the lexicon (condition 4), until the ending node of the lattice is reached.

condition 3: this condition aims at verifying if in the lexicon, there is at least a word where the character c_k follows the sequence of characters $c_m c_n$. This condition can be denoted as $c_k \in D_3(c_m c_n)$.

condition 4: similar to the condition 2, it aims at verifying if there is any possible character(s) following the sequence of characters $c_m c_n c_k$ in the lexicon. This condition is denoted as: $D_4(c_m c_n c_k)! = \emptyset$.

For each c_k in the candidate characters of the node $o_{(t"+1,t"+l)}$ and verifying the conditions 3 and 4, we compute the score $P(c_m c_n c_k)$ as follows:

$$\begin{aligned} P(c_m c_n c_k) = P(c_m c_n) * \\ b(c_k | o_{(t"+1,t"+l)}) * \\ a(c_n c_k | o_{(t',t")} \cup \ o_{(t"+1,t"+l)}) \end{aligned} \tag{3.15}$$

where $b(c_k|o_{(t"+1,t"+l)})$ and $a(c_n c_k|o_{(t',t")} \cup \ o_{(t"+1,t"+l)})$ are respectively the outputs of the SCR and bi-character models. The multiplication result of these values can be a

very small value that computer cannot store in the memory ("arithmetic underflow"). Therefore, in practice, sum of logarithms is used instead of multiplication. Hence, the Equations (3.14) and (3.15) can be respectively replaced by:

$$\begin{aligned} log(P(c_m c_n)) = log(b(c_m|o_{(1,t)}))+ \\ log(b(c_n|o_{(t+1,t+l)}))+ \\ log(a(c_m c_n|o_{(1,t)} \cup \ o_{(t+1,t+l)})) \end{aligned} \tag{3.16}$$

$$\begin{aligned} log(P(c_m c_n c_k)) = log(P(c_m c_n))+ \\ log(b(c_k|o_{(t"+1,t"+l)}))+ \\ log(a(c_n c_k|o_{(t',t")} \cup \ o_{(t"+1,t"+l)})) \end{aligned} \tag{3.17}$$

After the step 2, the system provides a list WL of words w_i and their associated recognition scores in PL. A word w_i in the lexicon may be found in different paths in the lattice, with different scores. Finally, for each word w_i, only the path giving the highest score and reach the ending node of the lattice is selected as the final result. The recognition scores of each word w_i is then normalized by its length (number of characters). The $W - best$ words with highest scores are considered as the outputs of the system.

Conclusion

Our research work is a part of a commercial project in collaboration with a private company. Although this searching strategy provides very encouraging results [112, 111] and allowed us to provide the first result to the company, it has some drawbacks. Indeed, it relies on the best-first search algorithm. Firstly, the recognition paths are not optimally chosen and it can provide a local optimum solution. Secondly, this method is very dependent on the pruning parameter E, which is difficult to settle. If E is too small, then the system risks to miss the correct path. On the other hand, if E is high, the computational time may be very high.

In order circumvent these drawbacks, we proposed another word decoding method. It relies on dynamic programming, similar to Viterbi algorithm [146].

3.7.3 Dynamic programming

Given a lattice G of graphemes corresponding to a word to recognize, this method consists in searching an optimal path for each word w_i in a given lexicon, and calculating its recognition score. This method relies on dynamic programming, introduced hereafter. We have to mention that, in our preliminary experiments, a flat search strategy (*i.e.* word by word search) is used. A speedup could be achieved with a TRIE representation

of the lexicon instead of decoding separately each word in the lexicon.

3.7.3.1 Description of the method

Let us consider the input handwritten word as a sequence of nodes $o_{(t,t')}$ in the corresponding lattice, where each node is indexed by its starting grapheme t and its ending grapheme t'. Let us call T the total number of graphemes in the word, and L the maximum level of the lattice (*i.e.* the maximum number of graphemes in a character). Given a lattice G, the recognition score $P_{w_i}(G)$ of the word $w_i = \{c_1, c_2, \ldots, c_K\}$ is computed recursively using Equation (3.18) and $P_{w_i}(G) = P(T, c_K|\{c_1, \ldots c_{K-1}\})$. At each grapheme t and each character c_k in the word w_i, only the best path (optimal path) is kept. An example of this decoding method is given in Figure 3.43.

$$\begin{aligned} P(t, c_k|\{c_1, \ldots c_{k-1}\}) = \max_{l=1..L}[&P(t-l, c_{k-1}|\{c_1, \ldots c_{k-2}\}) \\ &b(c_k|o_{(t-l+1,t)}) \\ &a(c_{k-1}c_k|s(t-l, c_{k-1}) \cup o_{(t-l+1,t)})] \end{aligned} \tag{3.18}$$

In practice, the sum of logarithms is used instead of multiplication (because of arithmetic underflow). The Equation (3.18) can be denoted as:

$$\begin{aligned} log(P(t, c_k|\{c_1, \ldots c_{k-1}\})) = \max_{l=1..L}[&log(P(t-l, c_{k-1}|\{c_1, \ldots c_{k-2}\}))+ \\ &log(b(c_k|o_{(t-l+1,t)}))+ \\ &log(a(c_{k-1}c_k|s(t-l, c_{k-1}) \cup o_{(t-l+1,t)}))] \end{aligned} \tag{3.19}$$

Where

- $t \in \{1, 2, \ldots, T\}$
- $s(t-l, c_{k-1})$ is the node selected for the character c_{k-1} at earlier steps.
- $b(c_k|o_{(t-l+1,t)})$ is the estimated probability that node $o_{(t-l+1,t)}$ is recognized as the character c_k. This estimated probability is given by the SCR (see section 3.5).
- $a(c_{k-1}c_k|s(t-l, c_{k-1}) \cup o_{(t-l+1,t)})$ is the output probability provided by the bi-character model B_{c_{k-l},c_k} for the pair of neighboring nodes $s(t-l, c_{k-1})$ and $o_{(t-l+1,t)}$ (see section 3.6).

Initialization:

- if $k = 1$ and $t \leq L$ then, $P(t, c_1) = b(c_1|o_{(1,t)})$ and $s_{(t,c_1)} = o_{(1,t)}$
- if $k = 1$ and $t > L$ then $P(t, c_1) = null$, since $o_{(1,t)}$ cannot be a node (as it would be at a higher level than the maximum level L of the lattice).

An example of such a case is given in Figure 3.43 (the cross for character 'c' at $t = 4$)

- if $t < k$ then $P(t, c_k|\{c_1, c_2, \ldots c_{k-1}\}) = null$, as computation is impossible as one grapheme cannot stand for more than one character (see Figure 3.43, the cross for letter 'i' at $t = 1$). This hypothesis is realistic because our segmentation method tends to over-segment the characters, not to under-segment them.

The recognition score of each word w_i is then normalized by its length (number of characters). Words with the highest scores are considered as the outputs of the system. An example is given hereafter to illustrate this method.

Example:

Considering the lattice given in Figure3.40, page 120 of 7 graphemes concatenated at 3 levels ($T = 7$, $L = 3$), the decoding process of the word $w_i =$ "*cinq*" ($c_1 =$ '*c*', $c_2 =$ '*i*', $c_3 =$ '*n*' and $c_4 =$ '*q*') is illustrated in Figure 3.43.

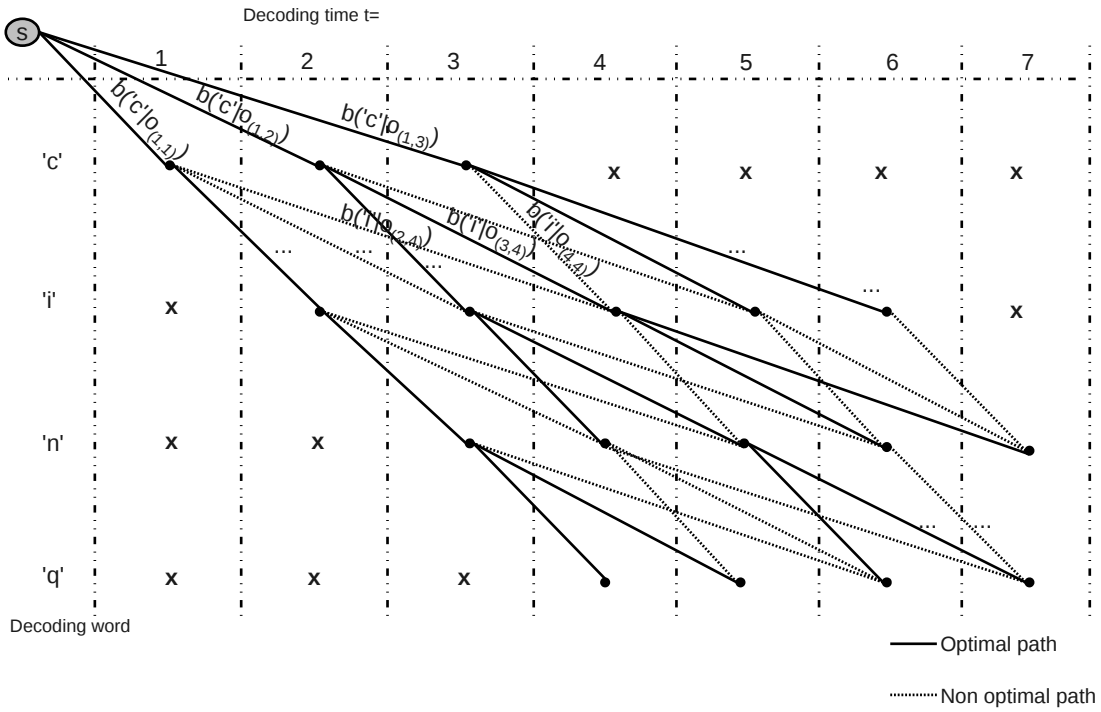


Figure 3.43: An example of the decoding process of word "*cinq*" for the lattice given in Figure 3.40 using our dynamic programming method. The solid lines represent the optimal paths.

The decoding results can be described as follows:

t=1

k=1: $c_k = `c`$

$$P(1, `c`) = b(`c`|o_{(1,1)})$$

$$s(1, `c`) = o_{(1,1)}$$

k={2,3,4}

$$P(1, `i`|\{`c`\}) = P(1, `n`|\{`c`, `i`\}) = P(1, `q`|\{`c`, `i`, `n`\}) = null$$

t=2

k=1: $c_k = `c`$

$$P(2, `c`) = b(`c`|o_{(1,2)})$$

$$s(2, `c`) = o_{(1,2)}$$

k=2: $c_k = `i`$, $c_{k-1} = `c`$

$$P(2, `i`|\{`c`\}) = \max[P(1, `c`) * b(`i`|o_{(2,2)}) * a("ci"|s(1, `c`) \cup o_{(2,2)})]$$

$$s(2, `i`) = o_{(2,2)}$$

k={3,4}

$$P(2, `n`|\{`c`, `i`\}) = P(2, `q`|\{`c`, `i`, `n`\}) = null$$

t=3

k=1: $c_k = `c`$

$$P(3, `c`) = b(`c`|o_{(1,3)})$$

$$s(3, `c`) = o_{(1,3)}$$

k=2: $c_k = `i`$, $c_{k-1} = `c`$

$$P(3, `i`|\{`c`\}) = \max[P(1, `c`) * b(`i`|o_{(2,3)}) * a("ci"|s(1, `c`) \cup o_{(2,3)}),$$
$$P(2, `c`) * b(`i`|o_{(3,3)}) * a("ci"|s(2, `c`) \cup o_{(3,3)})]$$

suppose that $P(2,\text{`}c\text{`}) * b(\text{`}i\text{`}|o_{(3,3)}) * a(\text{"}ci\text{"}|s(2,\text{`}c\text{`}) \cup o_{(3,3)})$ is maximum, then the retained node is $s(3,\text{`}i\text{`}) = o_{(3,3)}$, otherwise $s(3,\text{`}i\text{`}) = o_{(2,3)}$

k=3: $c_k = \text{`}n\text{`},\ c_{k-1} = \text{`}i\text{`}$

$$P(3,\text{`}n\text{`}|\{\text{`}c\text{`},\text{`}i\text{`}\}) = \max[P(2,\text{`}i\text{`}|\{\text{`}c\text{`}\}) * b(\text{`}n\text{`}|o_{(3,3)}) * a(\text{"}in\text{"}|s(2,\text{`}i\text{`}) \cup o_{(3,3)})]$$

$$s(3,\text{`}n\text{`}) = o_{(3,3)}$$

k=4

$$P(4,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\}) = null$$

t=4

k=1

$$P(4,\text{`}c\text{`}) = null$$

k=2

$$P(4,\text{`}i\text{`}|\{\text{`}c\text{`}\}) = \max[P(1,\text{`}c\text{`}) * b(\text{`}i\text{`}|o_{(2,4)}) * a(\text{"}ci\text{"}|s(1,\text{`}c\text{`}) \cup o_{(2,4)}),$$
$$P(2,\text{`}c\text{`}) * b(\text{`}i\text{`}|o_{(3,4)}) * a(\text{"}ci\text{"}|s(2,\text{`}c\text{`}) \cup o_{(3,4)}),$$
$$P(3,\text{`}c\text{`}) * b(\text{`}i\text{`}|o_{(4,4)}) * a(\text{"}ci\text{"}|s(3,\text{`}c\text{`}) \cup o_{(4,4)})]$$

suppose that $P(2,\text{`}c\text{`}) * b(\text{`}i\text{`}|o_{(3,4)}) * a(\text{"}ci\text{"}|s(2,\text{`}c\text{`}) \cup o_{(3,4)})$ is the maximum score, then the retained node is $s(4,\text{`}i\text{`}) = o_{(3,4)}$, otherwise, it would be $o_{(2,4)}$ or $o_{(4,4)}$ (depending on which therm is maximum)

k=3

$$P(4,\text{`}n\text{`}|\{\text{`}c\text{`},\text{`}i\text{`}\}) = \max[P(2,\text{`}i\text{`}|\{\text{`}c\text{`}\}) * b(\text{`}n\text{`}|o_{(3,4)}) * a(\text{"}in\text{"}|s(2,\text{`}i\text{`}) \cup o_{(2,4)}),$$
$$P(3,\text{`}i\text{`}|\{\text{`}c\text{`}\}) * b(\text{`}n\text{`}|o_{(4,4)}) * a(\text{"}in\text{"}|s(2,\text{`}i\text{`}) \cup o_{(4,4)})]$$

suppose that $P(3,\text{`}i\text{`}|\{\text{`}c\text{`}\}) * b(\text{`}n\text{`}|o_{(4,4)}) * a(\text{"}in\text{"}|s(2,\text{`}i\text{`}) \cup o_{(4,4)}))$ is the maximum score, then the retained node is $s(4,\text{`}n\text{`}) = o_{(4,4)}$, otherwise it is $o_{(2,4)}$

k=4

$$P(4,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\}) = \max[P(3,\text{`}n\text{`}|\{\text{`}c\text{`},\text{`}i\text{`}\}) * b(\text{`}q\text{`}|o_{(4,4)}) * a(\text{"}nq\text{"}|s(3,\text{`}n\text{`}) \cup o_{(2,4)}))]$$
$$s(4,\text{`}q\text{`}) = o_{(4,4)}$$

In the case of $t = \{5, 6, 7\}$, the decoding process can be described in the same way.

In this example (see Figure 3.43), we obtain 4 optimum paths of the word "*cinq*":

1. $\{o_{(1,1)}, o_{(2,2)}, o_{(3,3)}, o_{(4,4)}\}$ with recognition score $P(4,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\})$
2. $\{o_{(1,1)}, o_{(2,2)}, o_{(3,3)}, o_{(4,5)}\}$ with recognition score $P(5,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\})$
3. $\{o_{(1,2)}, o_{(3,3)}, o_{(4,5)}, o_{(5,6)}\}$ with recognition score $P(6,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\})$
4. $\{o_{(1,2)}, o_{(3,3)}, o_{(4,5)}, o_{(5,7)}\}$ with recognition score $P(7,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\})$

The first 3 paths are not retrained since the word "*cinq*" is found before the ending node is reached. Hence, the recognition score of word *cinq* for the lattice G is $P_{\text{"}cinq\text{"}}(G) = P(7,\text{`}q\text{`}|\{\text{`}c\text{`},\text{`}i\text{`},\text{`}n\text{`}\})$ in the 4^{th} path.

3.7.3.2 Adaptation to the character size

The method presented above assumes that every character class has the same maximum number of graphemes (*i.e* size of character), despite their differences. It is however obvious that the number of graphemes in the character '*c*' is lower than the number of graphemes in the characters $\{\text{`}n\text{`},\text{`}m\text{`},\dots\}$. Let us suppose that the maximum number of graphemes of the character '*c*' (respectively '*i*', '*n*', '*q*') is equal to 2 (and 3). In such a case, it is useless to decode the character $c_k = \text{`}c\text{`}$ at $t = 3$ and its following paths $(\{o_{(1,3)}, o_{(4,4)}, o_{(5,7)}\}, \{o_{(1,3)}, o_{(4,5)}, o_{(6,7)}\}, \dots)$, as illustrated in Figure 3.44.

Our objective is to take into account the maximum number of graphemes adapted to each character class during the word decoding process. Using this trick, the computational time can be reduced, since useless paths can be avoided. Furthermore, the ambiguities with unknown pattern can be also reduced. Equation (3.18) can be replaced by Equation (3.20).

$$\begin{aligned} P(t, c_k|\{c_1, \dots c_{k-1}\}) = \max_{m=1..L(c_k)} [&P(t-m, c_{k-1}|\{c_1, \dots c_{k-2}\}) * \\ &b(c_k|o_{(t-m+1,t)}) * \\ &a(c_{k-1}c_k|s(t-m, c_{k-1}) \cup o_{(t-m+1,t)}] \end{aligned} \tag{3.20}$$

Where $L(c_k)$ is the maximum number of graphemes for the character c_k.

Under the following conditions:

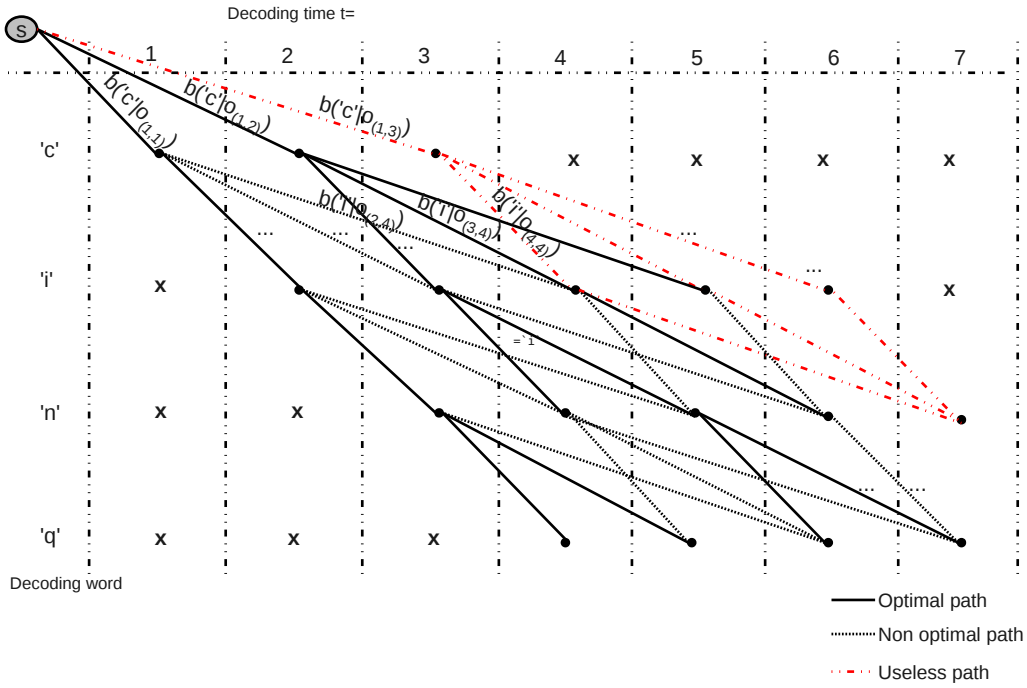


Figure 3.44: Example of the decoding process of word "*cinq*" using dynamic programming: adaptation to the size of character. The red-dashed lines represent the fruitless paths.

- if $k = 1$ and $t \leq L(c_k)$, then $P(t, c_1) = b(c_1|o_{(1,t)})$ and $s_{(t,c_1)} = o_{(1,t)}$
- if $k = 1$ and $t > L(c_k)$, then $P(t, 1) = null$, since it is considered as a fruitless path (for instance in Figure 3.44, character 'c' and $t = 3$), or cannot be a node.

In order to estimate the number of graphemes per character class, a single character database (see section 4.1.4) is used. The samples of each character class c_k are segmented into graphemes using the same segmentation method as in handwritten word segmentation (see section 3.3). A accumulative histogram of the number of graphemes for each character c_k is created by counting the number of character samples that contain different number of graphemes $i = \{1, 2, 3, \ldots, 10\}$, denoted as $HG_{c_k}(i)$. We consider that the maximum number of graphemes $L(c_k)$ for each character class c_k that will be retained must be able to represent at least 99% of the total samples in the class c_k. It can be described as: $\sum_{i=1}^{L(c_k)} \frac{HG_{c_k}(i)}{NC(c_k)} > 99\%$, where $NC(c_k)$ is the total number of character samples of the class c_k.

Table 3.2 gives an example of the cumulative graphemes histogram for three different characters, $c_k = \{'a', 'c', 'w'\}$, and the number of graphemes varied from 1 to 10. According to this histogram, the number of maximum graphemes for the character classes

‘a‘, ‘c‘ and ‘w‘ are respectively: $L(\text{‘}a\text{‘}) = 5$, $L(\text{‘}c\text{‘}) = 3$ and $L(\text{‘}w\text{‘}) = 6$.

Table 3.2: The cumulative histogram of graphemes of characters ‘a‘, ‘c‘ and ‘w‘.

character class $c_k = \text{‘}a\text{‘}$			character class $c_k = \text{‘}c\text{‘}$			character class $c_k = \text{‘}w\text{‘}$		
i	$HG_{\text{‘}a\text{‘}}(i)$	$\sum_{i=1}^{10} \frac{HG_{\text{‘}a\text{‘}}(i)}{NC(\text{‘}a\text{‘})}$	i	$HG_{\text{‘}c\text{‘}}(i)$	$\sum_{i=1}^{10} \frac{HG_{\text{‘}c\text{‘}}(i)}{NC(\text{‘}c\text{‘})}$	i	$HG_{\text{‘}w\text{‘}}(i)$	$\sum_{i=1}^{10} \frac{HG_{\text{‘}w\text{‘}}(i)}{NC(\text{‘}w\text{‘})}$
1	9	0.60%	1	356	23.73%	1	0	0.00%
2	82	6.07%	2	751	73.80%	2	56	3.73%
3	837	61.87%	**3**	381	**99.20%**	3	154	14.00%
4	515	96.20%	4	12	100.00%	4	1200	94.00%
5	49	**99.47%**	5	0	100.00%	5	71	98.73%
6	7	99.93%	6	0	100.00%	**6**	19	**100.00%**
7	1	100.00%	7	0	100.00%	7	0	100.00%
8	0	100.00%	8	0	100.00%	8	0	100.00%
9	0	100.00%	9	0	100.00%	9	0	100.00%
10	0	100.00%	10	0	100.00%	10	0	100.00%

3.7.4 Conclusion

This section introduces two word decoding methods which are applied to the lattice of graphemes (see section 3.3) in order to find the best path associated with each word w_i in a given lexicon. These methods rely on the information provided by a SCR (see section 3.5) and the bi-character models (see section 3.6). The words with the highest recognition scores are considered as the recognized words of the input writing.

The first method is a directed graph search method based on a pruning strategy. This search method is similar to Beam-Search algorithm. At each step, only the E-best candidates are considered. The fruitless candidates (*i.e.* candidates with lowest probability) are abandoned. The parameter E has to be optimally fixed, otherwise, the correct path may be lost.

To handle this difficulty, a second method is proposed. This method finds an optimal path for each word w_i in the lattice. It relies on dynamic programming, similarly to Viterbi algorithm, and takes into account the size of each character.

3.8 Conclusion

This chapter introduces a new discriminative approach to create an on-line handwritten word recognition system. This proposed system relies on three mains aspects: explicit grapheme segmentation/recognition, SCR and bi-character models. In this system:

- Both on-line and off-line (local and global, statistical and structural) features are used in order to take profit of their complementarity, and avoid some problems linked to the instability of the on-line features (see section 2.2.3.3 and Fig. 2.3).

- A delayed stroke management method is proposed (see section 2.2.1).
- A SCR with rejection is used. Two solutions have been proposed for rejecting unknown patterns. The first method consists in adding a garbage class in the SVM classifier, while the second method relies on a cascade of Adaboost classifiers to refine the outputs given by the SVM classifier. Both solutions satisfy the requirement of the explicit segmentation/recognition method (see table. 2.2).
- At each node of the lattice, only the N potential characters returned by the SCR (with highest scores) are used (see section 3.5.4). Using such method, computational time is reduced without degrading the effectiveness of the system. Different experimental results are given in section 4.2.
- Contextual information is integrated in the recognition process by jointly recognizing pairs of neighboring characters using bi-character models. These bi-character models allow verifying recognition results returned by the SCR, in order to avoid some ambiguities that may appear during word decoding. The bi-character models are built using logistic regression.
- Two word decoding methods are introduced: directed graph search and dynamic programming.

In the next chapter, a series of experiments are presented, in order to evaluate our system and different solutions introduced at the: single character, bi-character and word decoding levels. A comparison between the proposed system and a baseline HMM-based system is given and discussed.

4 Experiments and Discussion

This chapter aims at evaluating the effectiveness and efficiency of the proposed system. It presents the experimental protocols/configurations and discusses the experimental results. First, we introduce the different databases that were used in these experiments. Then, we present a series of experiments which are performed with different configurations in order to evaluate the different methods/strategies for each stage of the system. We also compare the proposed system with a baseline HMM-based system and discuss their recognition results.

Contents

Our proposed on-line HWR relies on an explicit segmentation/recognition approach using a two levels analysis: character (see section 3.5) and bi-character (see section 3.6) levels. The three main stages in this proposed system are:

- At the character level, a SCR is used (see section 3.5). It relies on a SVM classifier using a combination of on-line and off-line features. In order to deal with the problem concerning the unknown patterns in some nodes in the lattice, two different rejection strategies are proposed, respectively based on SVM with garbage class and rejection systems (relying on a cascade of Adaboost classifiers).
- At the bi-character level, the bi-character models are used to refine the outputs provided by the SCR. These bi-character models are created using Logistic Regression (see section 3.6).
- Two strategies are presented for the word decoding process. The first one relies on directed graph search, while the second relies on dynamic programming similar to Viterbi algorithm (see section 3.7).

This chapter provides experimental results and discussions on the effectiveness and efficiency of our system. The experiments introduced in section 4.2 aim at evaluating the different methods/strategies which are used in these three main stages. Before introducing these experiments, in section 4.1, we introduce the different databases which are used for our evaluation process. Section 4.3 is dedicated to a comparison between our system and a baseline HMM-based system. Finally, a conclusion is provided as a summary of all the obtained results.

4.1 Databases

Three standard databases of the literature are used in our experiments: IRONOFF, UNIPEN and Unipen-ICROW-03. In addition to these three databases, a single character database and a bi-character database are created and are used to respectively train the SCR and the bi-character models.

Let us remind that the isolated characters refer to the characters which are individually written in predefined boxes, while single characters refer to the ones which are segmented from handwritten words.

4.1.1 IRONOFF

The IRONOFF database has been collected by Nantes University [142]. It contains writings of isolated characters and isolated words in both types of data: on-line and off-line. The off-line data corresponds to real scanned image. The isolated characters are classified into three groups (digit, uppercase and lowercase). The isolated words are classified into three other groups (bank cheque, accented and English words), as shown in Table 4.1.

Table 4.1: Detailed information about the IRONOFF database.

	Group of data	Number of classes/lexicon size	Number of samples
Character	Digit	10	4086
	Lowercase	26	10685
	Uppercase	26	10679
Word	Cheque word	30	11934
	Accented word	171	28657
	English word	26	2689

Note:

- The group of "Cheque words" contains the writings of 30 French words which are used in the French Cheque system. Each word is composed of only lowercase and non accented characters.
- The group of "Accented words" contains the writings of 171 French words. Each word can be composed of lowercase, uppercase and/or accented characters.
- The group of "English words" contains the writings of 26 English words. Each word can be composed of lowercase and/or uppercase characters.

4.1.2 UNIPEN

The UNIPEN database contains on-line handwriting of isolated characters, single characters extracted from handwritten words, isolated words and text-lines of various languages (including Latin and Chinese)[49]. The writings in this database were collected by different companies and universities at the initiative of the International Association of Pattern Recognition[1] (Technical Committee 11). In the case of isolated words and text-lines, they contain different styles of writing: hand-printed style, cursive style and mixed styles.

The UNIPEN database contains two sets of data: Train_r01_v07 and DevTest_R01_v02. The first set (Train_r01_v07) is publicly available while the second set (DevTest_R01_v02) is not publicly available. The Train_r01_v07 set contains 9 categories of data, as illustrated in Table 4.2.

[1] http://www.iapr.org

Table 4.2: Detailed information of Train_r01_v07 set in the UNIPEN database, extracted from the official web site (http://unipen.nici.kun.nl/) of the UNIPEN database.

Category	Number of characters/ words/ lines	Number of files	Description
1a	15953	634	isolated digits
1b	28069	1423	isolated uppercase characters
1c	61351	2145	isolated lowercase character
1d	17286	1222	isolated symbols (punctuation etc.)
2	122628	2735	isolated characters, full character set (lowercase characters, uppercase characters, digit and symbols)
3	67352	1949	single characters in the context of words or texts
6	75529	3298	cursive and unconstrained words (without digits and symbols)
7	85213	3393	words, any style, full character set
8	14544	4563	text: (minimally two words of) free text, full character set (mixed case and digit)

4.1.3 Unipen-ICROW-03

This database has been used for the "On-line handwritten word recognition competition" organized during ICDAR-2003[2]. This database is particularly adapted for evaluating the unconstrained on-line HWR, since it contains a set of freestyle handwritten words (handprinted, unconstrained and cursive) written by 72 writers of different nationalities (Dutch, Irish, Italian, + mixed). Different kinds of capturing devices were used to produced this data. In this database, the writings are classified by writer, and the number of handwritten words per writer varies from 46 to 300, with a total of 13119 handwritten words and a lexicon of 879 words.

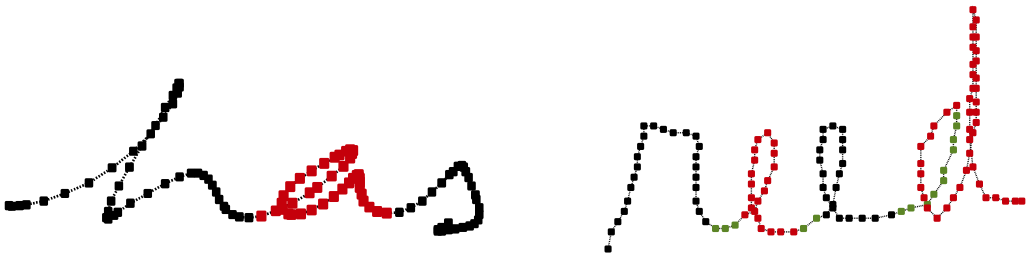
Since our research scope focuses on handwritten words containing lowercase characters, only 12440 handwritten words (out of 13119 handwritten words) of 67 writers (out of 72 writers) are considered in our experiments, with a lexicon size of 818 words.

4.1.4 Single character database

As mentioned in section 3.5, in order to recognize segmented characters from handwritten words (*i.e.* each node of the lattice), the SCR has to be trained with a database containing characters segmented from handwritten words.

[2]http://www.ai.rug.nl/~lambert/unipen/icdar-03-competition/_README

Unfortunately, segmentation points between characters in the IRONOFF and Unipen-ICROW-03 databases are not given. The UNIPEN database provides a set of data (in category 3) which contains single characters extracted from handwritten words. We do not have any detailed explanation about the collection process on this category. We do not know whether these characters are segmented from handwritten words/texts, or individually written. Based on our observations of the data, we noticed that some characters in this category are individually written (isolated characters) while, some other characters are segmented from handwritten words/texts (single characters). Since the data in the UNIPEN database are collected by different organizations, character segmentation methods are also different. We can notice that there are two different methods. In the first method, they consider ligatures as a part of the characters, as illustrated in Figure 4.1(a). In the second method, they do not consider ligatures as a part of the characters (see Figure 4.1(b)). In this case, the information concerning ligatures (sequences of green points in Figure 4.1(b)) is lost. Therefore, the segmented characters provided by this second method cannot be used in our system, since we consider ligatures as a part of the characters. Finally, there are a lot of incorrectly segmented characters that may be caused by segmentation errors. For all these reasons, the isolated character in the category "3" of the UNIPEN database cannot be used to train our SCR.



(a) (b)

Figure 4.1: Segmented characters extracted from the category "3" in the UNIPEN database. The sequences of black/red points represent the segmented characters, while the sequences of green points represent the ligatures removed by the second segmentation method.

Creating our own single character database was an unavoidable task. However, manually doing such a task requires huge of time and effort, since the segmentation of training characters must be as precise as possible. In order to avoid such manual work, we proposed a semi-automatic segmentation method to create training data, as illustrated in Figure 4.2. Each handwritten word of the training database is submitted to a basic version of our HWR[3]. Indeed, this system aims at finding the best segmentation points

[3]Using ICR (see section 4.2.1.1) which is trained with the isolated characters selected from the IRONOFF and UNIPEN databases. The bi-character models are not yet integrated, and the word decoding process is based on graph search.

between characters which compose the input handwritten word (the ground-truth is known, as we are working with the training data). The sequence of points between two consecutive segmentation points is considered as a segmented character. Each segmented character is then submitted to a ICR (see section 4.2.1.1). If the estimated recognition probability is higher than a given threshold, the segmented character is sent to the manual verification stage to remove the incorrect samples. Otherwise, the segmented character is automatically removed.

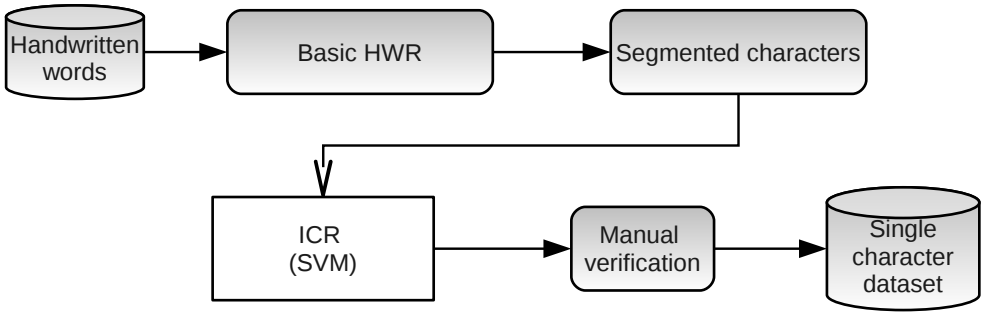


Figure 4.2: Single character segmentation method using a semi-automatic process.

In this single character database version, only the writings of cheque words in the IRONOFF database are segmented. The number of segmented characters for each character class is reported in Table. 4.3, column *set 1*. We additionally use another single character database generated by Nantes University [3]. The samples in this database are segmented from all the handwritten words in the IRONOFF database using a commercial HWR. The number of these segmented characters is reported in Table. 4.3, column *set 2*.

4.1.5 Bi-character database

As far as we know, since the idea of using bi-character models for on-line HWR was new at the beginning of our study, there was no bi-character database providing samples of on-line bi-characters. Therefore, a bi-character database had to be created in order to train the bi-character models.

In our context, creating real training data for all bi-character classes is a difficult task, since these bi-character samples have to be segmented from handwritten words. A method similar to the single character segmentation introduced earlier in section 4.1.4 could have been used. However, trying to apply such approach would have raised many problems. First of all, there is a very large set of 26*26 classes to be manually verified, in order to check the consistency of the data. Secondly, considering the very large number of classes to be used to train the models and considering the distribution of co-occurrence of characters in the handwritten words database, training samples of some bi-character

Table 4.3: Detailed information about our single character databases (our own database and the database generated by Nantes university).

Character class	set 1	set 2	set 1 + set 2	Character class	set 1	set 2	set 1 + set 2
a	1688	3955	5643	n	1673	5969	7642
b	0	921	921	o	1744	4286	6030
c	1630	2386	4016	p	366	1295	1661
d	1163	1485	2648	q	1863	1838	3701
e	1596	9564	11160	r	1451	4828	6279
f	1080	1318	2398	s	1719	4096	5815
g	389	1219	1608	t	1352	6211	7563
h	402	982	1384	u	1654	4578	6232
i	760	6430	7190	v	315	860	1175
j	0	147	147	w	0	152	152
k	0	216	216	x	1497	1328	2825
l	1017	2226	3243	y	0	1002	1002
m	1116	1401	2517	z	1877	2296	4173

classes may be missing or may not contain enough samples to be representative.

In order to circumvent these problems, we artificially generated a bi-characters database by concatenating single characters segmented from handwritten words. Considering the large variation of writings produced by different writers, generating bi-character samples with a pair of single characters provided by different writers may produce unlikely bi-character shapes. Therefore, we decided that bi-character samples had to be concatenated by using pairs of single characters issuing from the same writer.

Our constraint for this bi-character generation method was to provide samples as similar as possible to real bi-character samples. As mentioned above, artificial bi-character samples were artificially generated by using the single character database introduced in section 4.1.4. The management of the character size and alignment is an important question, since it conditions the realistic representation of artificially constructed bi-characters. In particular, when generating a bi-character sample, the height of the characters composing this bi-character sample has to be coherent and as similar as possible to the real data. For instance, in the samples of bi-character "ab", the height of the character 'a' must be almost two times smaller than the height of the character 'b'. For this reason, the single characters are first normalized by using the character normalization method introduced in section 3.2.2. This method ensures that character signals belonging to the same character class have approximately the same size. Then, the bi-character generation method explained hereafter is applied. This artificial bi-character generation method aims at creating a bi-character sample by concatenating two character signals.

Artificial bi-character generation method:

Let us suppose that P^1 and P^2 respectively correspond to the on-line signals of characters 1 and 2. In order to generate a bi-character sample composed by these two characters, first, we have to extract their alignment lines which indicate their alignment positions on the y axis. Considering our strategy for generating bi-character samples from two single characters, these single characters can be issued from different words written at different spatial positions on the capturing device. The concatenation of these two characters is not straight forward. Indeed, we need to re-locate the character 2 according to the position of the character 1 by applying horizontal and vertical alignment based on their alignment lines, as presented in the following paragraphs.

Detecting the alignment line (AL)

For each character signal, its alignment line is detected based on its original writing position in the handwritten word. The alignment line of P^1 ($AL(P^1)$) can be estimated as:

- If P^1 belongs to groups 1 ($\{$'a', 'c', ...$\}$) or 2 $\{$'b', 'd', ...$\}$ (see section 3.2.2), its alignment line is located at the minimum point on the y axis of the character signal.

  $AL(P^1) = \min_{i=1}^{|P^1|}(P_i^1(y))$

- Otherwise, if P_i^1 belongs to Groups 3 ($\{$'g', 'j', ...$\}$) or 4 ($\{$'f'$\}$), the alignment line is located at the middle of the character signal.

  $AL(P^1) = \min_{i=1}^{|P^1|}(P_i^1(y)) + [\max_{i=1}^{|P^1|}(P_i^1(y)) - \min_{i=1}^{|P^1|}(P_i^1(y))]/2$

The alignment line of P^2 ($AL(P^2)$) can be estimated by the same way.

Horizontal and vertical alignment

This step aims at concatenating the character signals P^1 and P^2 to create a bi-character sample by applying horizontal and vertical alignment, as illustrated in Figure 4.3. Each point P_j^2 of the signal P^2 is modified as below:

$$\begin{aligned} P_j^2(x) &= P_j^2(x) + dx \\ P_j^2(y) &= P_j^2(y) + dy \end{aligned} \tag{4.1}$$

where:

$$dx = \max_{i=1}^{|P^1|}(P_i^1(x)) - \min_{j=1}^{|P^2|}(P_j^2(x)) \text{ and } dy = AL(P^1) - AL(P^2)$$

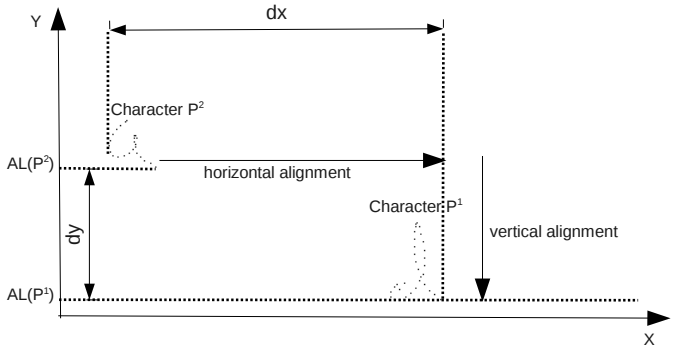


Figure 4.3: Horizontal and vertical alignments to create the bi-character samples.

The character 'z' is a special case since the writings of this character may belong to the Group 1 or 3. In our system, we classified the character 'z' into Group 3 and we assume that some incorrect bi-characters would be generated. We are aware that the generated bi-character samples may contain some noise. However, this noise is not necessarily a problem as long as the number of generated samples is high enough, as it provides more variability in the training stage. This noise can be removed if we consider post-processing stage of manual verification as for single characters. In this experiment, we consider all the generated bi-characters. Figure 4.4 illustrates some examples of bi-character samples generated by our method.

4.1.6 Unknown pattern database

Unknown pattern samples refer to the samples that do not belong to any character class, nor any bi-character class in the system. Let us remind here that, in our system, some nodes in the lattice may contain such patterns, because the nodes are obtained by concatenating different segmented graphemes.

Unknown patterns samples are required when creating a rejection option in the SCR, as explained in section 3.5.3. In our context, the samples of unknown patterns are generated as follows: given a training handwritten word, the corresponding lattice is created (see section 3.3). The concatenations of graphemes in the nodes of the lattice which does not belong to any character class (based on our manual verification) are retained as samples of unknown patterns. In our system, we have generated 5800 samples of unknown patterns. Figure 4.5 illustrates some examples of unknown pattern samples.

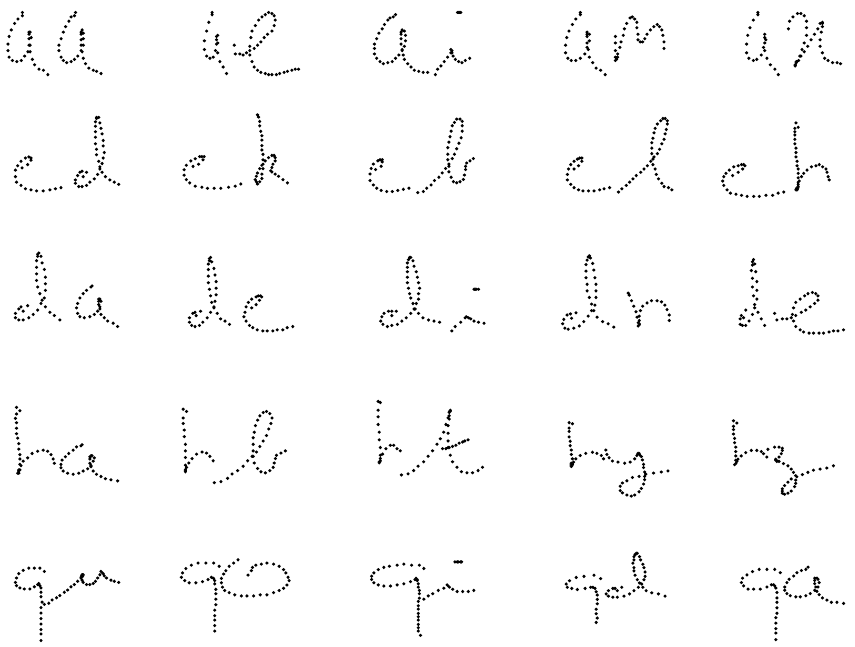


Figure 4.4: Bi-character samples artificially generated by our strategy.

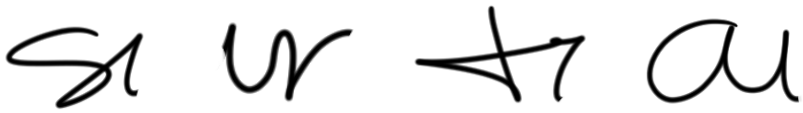


Figure 4.5: Some unknown pattern samples.

4.2 Experiments and discussions

In order to evaluate the effectiveness and efficiency of each method/strategy used at each stage of our system, we perform different series of experiments for respectively evaluating the ICR, the SCR, the bi-character models and the HWR. They are presented and discussed in the following sub-sections.

4.2.1 Character recognition evaluation

4.2.1.1 Evaluation in the case of isolated characters

This experiment (*Exp.*1) evaluates the character recognizer introduced in section 3.5 by testing this recognizer with isolated characters. The recognition results given by this

recognizer are then compared with those of the state-of-the-art systems. In this experiment, 26 lowercase character classes in the Latin alphabet are considered.

The experiments were performed using isolated characters randomly selected from two isolated character databases: IRONOFF and UNIPEN databases (see sections 4.1.1 and 4.1.2). The training set contains 1600 characters per class, the test set contains 400 characters per class, while the validation set contains 400 characters per class. This system relies on the combination of the off-line and the on-line features presented in sections 3.4.1 and 3.4.2 respectively. The 45 features (31 off-line features and 14 on-line features) given by the feature selection step (see section 3.27) are used.

Table. 4.4 shows a comparison of the recognition rates of our ICR and the ICR presented in [2] which relies on local on-line features[4] and SVM classifier. This comparison was published in [138] by the RecoNomad research team. Our ICR provides very competitive results in terms of recognition rates. In addition, our ICR performs much faster in terms of computational time for both training and testing processes. According to Table .4.5, the computational time of our ICR is 5 times faster than the ICR presented in [2], using the same database size. This speedup can be explained by the fact that, in our ICR, only 45 features are used while the ICR presented in [2] uses up to 210 features.

Table 4.4: Comparison of the recognition rates of our ICR and the ICR presented in [2], extracted from [138].

	Number of samples per class		Recognition rate (%)	
Nature of data	Training	Test	*Exp.*1	System in [2]
Digit	1600	400	**98.7**	98.6
Uppercase	1600	400	**95.6**	95.1
Lowercase	1600	400	93.3	**93.7**

Table 4.5: Comparison of the computational time between our ICR and ICR presented in [2] in the case of Digit recognition (10 classes). This comparison was published in [138].

	Our system	System in [2]	Speedup
Training (1600/class)	**64105 ms**	358156 ms	5.6 times
Test (400/class)	**13656 ms**	74063 ms	5.4 times

As for the HWR, this recognizer has to deal with the characters segmented from handwritten words. In order to observe its effectiveness when dealing with the segmented characters, the next experiments are performed on the database of single characters segmented from handwritten words.

[4]features are extracted from each point in the on-line signal.

4.2.1.2 Evaluation in the case of single characters

These experiments evaluate the effectiveness of the SCR (see section 3.5) when dealing with single characters segmented from handwritten words. In our system, as said earlier, 26 lowercase character classes are considered. In this context, we performed three kinds of experiments: SCR without garbage class, SCR with garbage class, and specific rejection systems. Let us remind that, in our HWR, the delayed strokes are considered. Therefore, when creating the SCR which will be used for HWR, it is important to consider the dot of character 'i' and 'j' and the bar of the character 't'.

SCR without garbage class (*Exp.*2): For this experiment, the SVM classifier is trained without any garbage class. Each class in the training, test and validation sets respectively contains 1600, 400 and 400 samples. Therefore, each character class requires 2400 samples selected from the single character database, as introduced earlier in section 4.1.4. However, since some character classes do not have enough samples (see Table 4.3), isolated characters of IRONOFF and UNIPEN databases were used to complete the training/test and validation set. In this experiment, 71 features (43 off-line features and 28 on-line features) are retained after the feature selection stage (see section 3.27). This number of features is more important than the number of features selected for ICR (see *Exp.*1). This can be explained by the fact that the shapes of characters segmented from handwritten words are more heterogeneous than those of isolated characters, therefore requiring more information to describe them.

Figure 4.6 illustrates the cumulative $Top-N$ recognition rates given by the system in *Exp.*2 (without garbage class). We notice that, when considering $Top-N=1$, the recognition rate is 88.23%, which is lower than the recognition rate obtained in *Exp.*1 (93.3%) on isolated characters. This can be explained by the higher variability in the single character shapes. In the case of $Top-N=7$, recognition rate raises up to 99.02%. When considering $Top-N \geq 8$, the recognition rates remain stable.

We also observe that the average recall and precision values over all the character classes are 88.23% and 88.50% respectively. However, for some character classes, the values of recall and/or precision are relatively weak, as illustrated in Figure 4.7. For instance, the recall value for character '*i*' is 70.8%. As a result, 29.2% of its samples are incorrectly classified. In addition, the precision value of the character class '*i*' is 67.82%. This means 32.18% of the samples which are classified as the character '*i*' do not belong to the class '*i*'. Similar observations can be made for some other character classes, for instance {'*l*', '*n*', '*r*', '*t*', '*u*'}. When studying more in details, the confusion matrix given in Table A.1, Appendix A, we can state that this is due to the high similarity between those character classes. For instance,

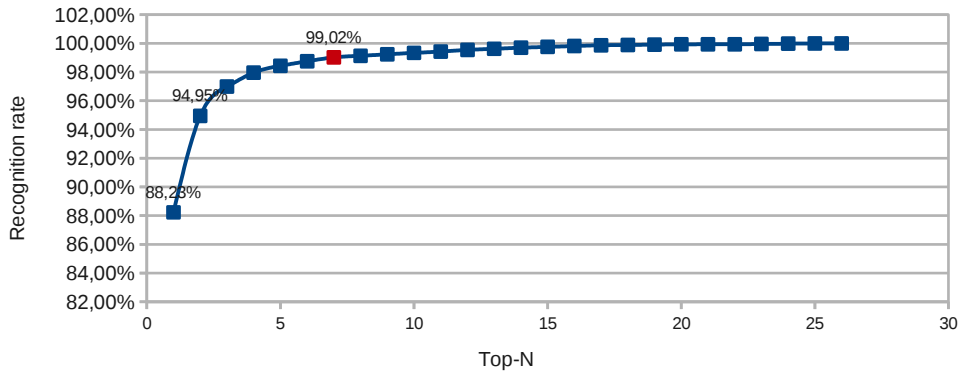


Figure 4.6: The cumulative $Top-N$ recognition rates given by the SCR without garbage class ($Exp.2$).

the shape of the character class 'i' is similar to the shapes of the character classes {'l','t','r','v'}. However, this is not really an issue when using this recognizer in our HWR. Indeed, in our system, for each node of the lattice, the N potential character candidates are considered, allowing to take into account other potential character candidates. In addition, the bi-character models are used to refine the outputs provided by SCR.

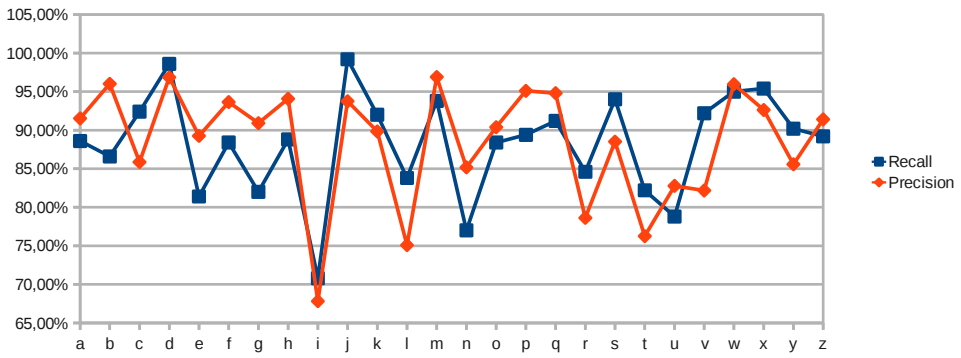


Figure 4.7: Recall and precision of each character class given by the SCR without garbage class ($Exp.2$).

SCR with garbage class ($Exp.3$): this experiment evaluates the effectiveness of the SCR when adding a garbage class in the SVM classifier, as introduced earlier in 3.5.3.1. We use the same training, test and validation sets as for $Exp.2$. The 3000 samples used in the garbage class correspond

to various unknown samples segmented from the handwritten words, as explained earlier in section 4.1.6.

The cumulative $Top - N$ recognition rates are shown in Figure 4.8. Similar observations as in $Exp.2$ can be made. For the cases $Top - N = 1$ and $Top - N = 7$, the recognition rates are 87.69% and 98.94% respectively. This recognition rate is slightly lower than the recognition rates in the $Exp.2$. This difference can be explained by two reasons: 1) the SVM classifier deals with a problem of 27 classes instead of 26 only in the $Exp.2$ and 2) the heterogeneity of the garbage class makes the classification more difficult for the SVM.

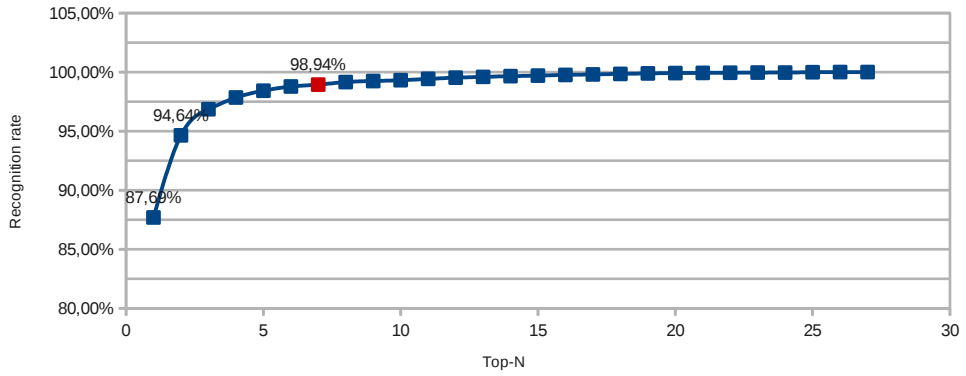


Figure 4.8: Cumulative $Top - N$ recognition rates given by the SCR with garbage class ($Exp.3$).

The average values of the recall and precision over all the character classes are 87.55% and 88.07% respectively. Similar to the $Exp.2$, the values of the recall and/or precision for some character classes are low, as illustrated in Figure 4.9. In this experiment, we focus on the garbage class only. The values of the recall and precision in the case of garbage class are 91.3% and 82.9% respectively. The high value of the recall for the garbage class shows the effectiveness of the rejection option. In other words, 91.3% of garbage class samples are correctly classified in the garbage class. However, the lower value of the precision indicates a relatively high rejection error rate. In other words, 17.1% of the samples belonging to the alphabet classes ('a', 'b', ... 'z') are classified in the garbage class. In our case, these values do not represent a main issue when using this recognizer in our HWR. Indeed, we do not explicitly reject the candidates that are classified into the garbage class, since SVM provides all the class labels in the outputs and each class label is

characterized by an estimated recognition probability. Therefore, in the case of rejection error, the correct label of the input sample is still provided by the SVM, but in this case, with lower estimated recognition probability.

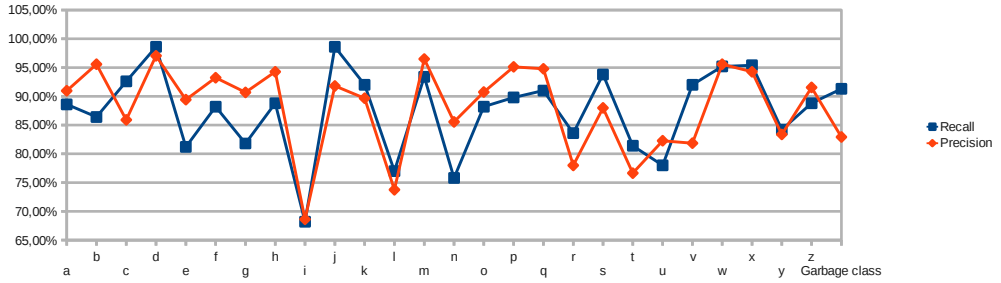


Figure 4.9: Recall and precision of each class given by the SCR with garbage class (*Exp.*3).

Rejection system (*Exp.*4): This experiment aims at evaluating the rejection systems which rely on the cascade of Adaboost classifiers, as introduced in section 3.5.3.2. In this experiment, we do not consider the SVM classifier. The effectiveness of the rejection systems when using together with SVM classifier can be observed only when they are used for HWR, as presented after in paragraph-B section 4.2.3.1.

This experiment is based on the same training, test and validation set as in *Exp.*3. A set of specific rejection systems for every character classes is created ($R = \{r_{`a`}, r_{`b`}, \ldots, r_{`z`}\}$) where each specific rejection system r_{c_k} of a character class c_k is trained by supposing that the samples belonging to the character class c_k are the positive samples and the samples belonging to all other classes (including the samples of unknown patterns) are negative samples. The test and validation sets are considered in the same way.

During the test step, we perform the experiments for each rejector independently. As mentioned in section 3.5.3.2, the effectiveness of each rejector r_{c_k} depends on 3 parameters: 1) the objective false positive rate f_k of the Adaboost classifiers in the cascade, 2) the objective true positive rate d_k of the Adaboost classifiers in the cascade and 3) the objective false positive rate γ of the overall rejection system (r_{c_k}). After performing the experiments, we observe that in many cases, the rejection system cannot reach the objective false positive rates f_k and γ as no additional negative samples can be rejected. Hence, a stopping condition has been added, in order to avoid useless computational costs.

Based on the observations of the experimental results, the value of f_k can be roughly fixed to 30%. This means that each Adaboost in the cascade accepts 30% of false positive. In other words, each Adaboost in the cascade has to be able to reject at least 70% of negative samples, unless the stopping condition is valid. The value of γ is fixed to 0.1%. This means that, the overall rejection system has to be able to reject up to 99.9% of the negative samples, unless the stop condition is valid.

In this experiment, we mainly focus on the value of the objective true positive rate parameter (d_k). By varying the values of d_k from 90% to 99% with a gap of 1%, we obtain a ROC curve (Receiver Operating Characteristic) illustrated in Figure 4.10.

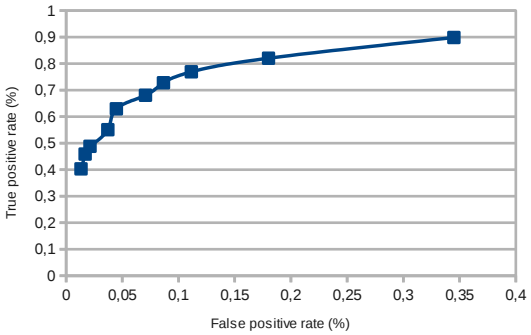


Figure 4.10: ROC curve given by the *Exp.*4.

This ROC curve shows that when using a high value of d_k, we obtain a rejection system with a high true positive rate (*i.e.* the rejection error rate is low) but providing a high false positive rate (*i.e.* the acceptation error rate is high). Otherwise, using a low value of d_k, we obtain a rejection system with a low true positive rate (*i.e.* the rejection error rate is high) but providing a low false positive rate (*i.e.* acceptation error rate is low). When using these rejection systems in our handwritten words recognition system, the value of d_k has to be optimally settled, which is not an easy task in our context. Moreover, adding these Adaboost rejectors in the overall system makes the training complexity higher (while the increase in rejection rate is very low).

4.2.2 Bi-character models evaluation

This section presents the evaluation of the bi-character models (*Exp.*5) which are built using logistic regression (see section 3.6). For each class, we consider 3000 samples for

training and 1000 samples for testing, randomly selected from the bi-character database (see section 4.1.5). As explained earlier in section 3.6.2, in order to optimize the computational time, the proposed bi-character models use the same 72 features as the SCR. The recognition rates of these models vary from ∼91% to ∼99%.

Since there is no bi-character models in the literature, the experimental results given by our bi-character models cannot be directly compared or discussed unless they are integrated in our HWR which is presented in the next section.

4.2.3 Handwritten word recognition system evaluation

This section presents the experiments conducted on the overall on-line HWR. In order to assess the performance of this system and to measure its robustness towards an increasing size of lexicon, we perform a series of experiments for which the size of the lexicon (containing no accented words) is progressively increased as follows: 818, 5000, 10000 and up to 20000 words. The lexicon of 818 words contains only the lowercase words of the Unipen-ICROW-03 database (see section 4.1.3). To obtain larger lexicons (5000, 10000 and 20000 words), we add English words, which are randomly selected from an English dictionary.

The experiments are performed with 11 different configurations using a computer with 3GHz of CPU, as illustrated in Table 4.6. The column "config." indicates the configuration number while the column "SCR" indicates the different SCRs (with and without rejection) used in the experiments. The column "Bi-char models (Log. Reg.)" indicates the use of the bi-character models in the experiments. The column "Word decoding" indicates the use of the two word decoding strategies in the experiments. The column "parameters" indicates the two important parameters of our system: N (the number of best potential characters considered in each node of the lattice), E (the pruning parameter used when the directed graph search strategy is used). For each configuration, the accuracy (acc) and the average computational time per word to recognize (CT in second) are reported and discussed.

The first 8 configurations evaluate different methods and strategies which are used in each stage of the system: character analysis (see section 3.5), bi-character models (see section 3.6) and word decoding process (see section 3.7). The number of levels in the lattice (*i.e.* size of character) is fixed to 7 ($L = 7$). This value is roughly estimated on the single character database (see section 4.1.4).

The configuration 9 aims at evaluating the system when using the maximum number of graphemes adapted to each character class. We keep on using $L = 7$. However, we additionally use the maximum number of graphemes adapted to each character class during the word decoding process (see section 3.7.3.2).

In the first 9 configurations, the delayed stroke management method (see section 3.3)

Table 4.6: Different configurations used to perform the experiments of our HWR. Note: (*) using a fixed maximum number of graphemes (7) for each character class. (**) using the maximum number of graphemes adapted to each character class estimated on a single character database (see section 3.7.3.2). (***) the delayed stroke management method (see section 3.3) is not considered.

Config.	SCR			Bi-char models (Log. Reg.)	Word decoding		Parameters		Test data	Note
	SVM	SVM + gar.cls.	SVM + Rej. Sys.		Graph search	Dyn. prog	N	E		
Config.1	x				x		7	3,5,7,9,11	1000	
Config.2	x				x		3, 5, 7,9	7	1000	*
Config.3		x			x		5	7	1000	
Config.4	x		x		x		5	7	1000	
Config.5	x					x	5,7,9,11, all	no	1000	
Config.6		x				x	7	no	1000	
Config.7	x			x		x	7	no	1000	
Config.8		x		x		x	7	no	1000	
Config.9		x		x		x	7	no	1000	**
Config.10		x		x		x	7	no	1000	***
Config.11		x		x		x	7	no	12440	*

is used. In configuration 10, we aim at observing the effectiveness of this delayed stroke management method. Therefore, in this configuration, we perform experiments without taking into account the delayed stroke management method, what allows comparing the recognition results.

In the first 10 configurations, only 1000 handwritten words which are randomly selected from the Unipen-ICROW-03 database are used. In the configuration 11, all the handwritten words in the Unipen-ICROW-03 database are used. Experiments using this last configuration allow observing the effectiveness of the system by writers.

It is important to mention that in some configurations (1 to 6), the bi-character models are not yet integrated. In this case, the estimated probability that the pair of neighboring nodes $o_{(t,t')}$ and $o_{(t'+1,t")}$ is recognized by the bi-character model $B_{c_m c_n}$ is initialized to 1 ($a(c_m c_n | o_{(t,t')} \cup o_{(t'+1,t")}) = 1$).

We present all these experiments in the following 6 sub-sections. A summary of the experimental results are given in the conclusion at the end of this section.

4.2.3.1 Using the directed graph search strategy

The objective of these experiments (configurations 1 to 4) is to observe the effectiveness and efficiency of the directed graph search strategy (see section 3.7.2) which is used during the word decoding process. In these experiments, only the SCR is used. The experiments are performed using 4 different configurations, classified into two parts. In the first part (configurations 1 and 2), we observe the impact of the different parameters, while in the second part (configurations 3 and 4) we compare the effectiveness of our system when using the SCR with and without rejection option.

A) Effect of the parameters E and N

In this paragraph, we want to observe the impact of the number of paths to be considered during the propagation in the graph (pruning parameter E) and the number of potential character candidates to be considered in each node of the lattice (N). The experiments are performed with two different configurations, as explained hereafter.

> **Configuration 1**: in this configuration, the SCR without rejection option (see *Exp.*2) is used. During the word decoding process, the directed graph search strategy is applied. The pruning parameter E has a great impact on the effectiveness and efficiency of the system. To optimally select the value of this parameter, we perform different experiments with 5 different values of E ($E = \{3, 5, 7, 9, 11\}$).
>
> According to the experimental results of the SCR (see Table 4.6 in *Exp.*2), when $Top-N = 7$, the cumulative recognition rate rises up to 99% while the cumulative recognition rates when $Top-N \geq 8$ remains stable. Therefore, in each node of the lattice, only the 7 potential character candidates provided by the SCR are considered ($N = 7$).

Table 4.7: The experimental results (accuracy and computational times) of the proposed HWR using configuration 1. The directed graph search strategy is used with different values of the pruning parameter ($E = \{3, 5, 7, 9, 11\}$).

Lexicon	E=3		E=5		E=7		E=9		E=11	
	acc(%)	CT(s)	acc(%)	CT(s)	acc(%)	CT(s)	acc(%)	CT(s)	acc(%)	CT(s)
818	66.6	3	69.9	4	71.9	7	**72.2**	10	**72.2**	14
5000	51.3	11	56.5	17	58.3	30	**59**	50	**59**	81
10000	45.9	23	51.3	26	52.6	54	53.9	98	**54.5**	169
20000	40.3	32	47.2	63	48.6	87	49.6	161	**50.4**	292

Based on the experimental results provided in Table 4.7, the following observations can be made:

- Increasing the value of the parameter E improves the recognition rates of the system, but it also increases the computational times. More specifically:

- Considering a small lexicon of 818 words, when the value of E is increased, the recognition rate and the computational time are also linearly increased.
- In the case of a large lexicon size (20 000 words), when $E > 7$, the recognition rate is slightly improved. However, the computational time is dramatically increased.

Conclusion: according to these experimental results, we noticed that when $E > 7$, there is a minor impact on the recognition rates. However, increasing the value of this parameter can dramatically increase the computational time, especially when using a large lexicon. As a compromise between the recognition rate and the computational time, we recommend using $E = 7$. Nonetheless, in the context of small lexicon, a higher value of E can be used. In the following experiments, $E = 7$ is considered.

Configuration 2: This configuration is used to observe the impact of the parameter N (the number of potential character candidates to be considered in each node, see section 3.5.4) of the system. In this configuration, we perform the experiments with different values of N ($N = \{3, 5, 7, 9\}$) and using $E = 7$, based on the experimental results given in the configuration 1. The experimental results are shown in Table 4.8.

Table 4.8: The experimental results of our HWR when using the directed graph search strategy with different values of parameter N ($N = \{3, 5, 7, 9\}$) and by considering $E = 7$ (configuration 2).

Lexicon	N=3		N=5		N=7		N=9	
	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)
818	60.30	2	70.4	3	**71.9**	7	71.8	33
5000	52.70	6	**58.5**	10	58.3	30	57.3	189
10000	48.50	11	**53.2**	16	52.6	54	52	318
20000	44.30	18	48.4	28	**48.6**	87	47.9	463

Based on the experimental results illustrated in Table 4.8, we can observe that:

- Normally, increasing the value of the parameter N improves the recognition rate. Surprisingly, the recognition rates slightly decrease when $N = 9$. It can be explained by the fact that when the number of potential character candidates in each node is increased, the number of possible candidates to be taken into account during the pruning process is also increased. As a consequence, the number of incorrect candidates is increased which raises the confusions/errors due to the pruning process (which can select a local optimum path, as explained in section 3.7.2).

- The system provides the highest recognition rates when $N = 5$ or 7, depending on the lexicon.

Conclusion: considering a lexicon of 818 (and 20000) words, the system provides the highest recognition rates when $N = 7$. However, there is a very small improvement compared to the case where $N = 5$. But, when considering $N = 7$, the computational time increases twice which makes the system two times slower. Therefore, in the following experiments, we decided to use $N = 5$ for directed graph search strategy.

The recognition results in configurations 1 and 2 allow us to observe the impact of the parameters E and N on our HWR. In the following experiments, we take into account the SCR with rejection option. This allows us to observe the effectiveness of our HWR when using the SCR with and without rejection option.

B) Comparison of the two rejection methods for the SCR.

As explained earlier in section 3.5, in order to deal with the problems related to the unknown patterns in some nodes of the lattice, we propose two different strategies to build a SCR with rejection option. The first strategy adds a garbage class in the SVM classifier, while the second strategy relies on rejection systems based on cascades of Adaboost classifiers.

The objective of these experiments is to evaluate the effectiveness of these two strategies when they are used in our HWR. We perform the experiments with the two different configurations presented hereafter.

Configuration 3: in this configuration, the SCR based on SVM with a garbage class (see *Exp* 3) is used. The values of E and N are initialized to 7 and 5 respectively, based on the previous experimental results.

The experimental results are given in the column "Configuration 3" of the Table 4.9. The column "Configuration 2" illustrates the recognition results when using the SCR without garbage class (see experiments in the configuration 2, using the same value of E and N). According to these recognition results, we observe that:

- Compared to the configuration 2, the system which relies on a SCR with a garbage class provides an improvement of 3.9% to 6.8% on the recognition rates, depending on the size of the lexicon.
- In the case for which the size of the lexicon is increased, the number of possible ambiguities caused by unknown patterns in the lattice is also increased. Using the SCR with garbage class allows handling these

ambiguities. Therefore, the larger the lexicon size is, the more the improvement on the recognition rate is important.

Table 4.9: Experimental results of the HWR using a SCR with and without garbage class, and graph search word decoding strategy (configuration 3).

Lexicon	Configuration 3 (with garbage class)		Configuration 2 (without garbage class)	
	acc(%)	CT (s)	acc(%)	CT (s)
818	**74.30**	3	70.4	3
5000	**64.20**	10	58.5	10
10000	**59.10**	17	53.2	16
20000	**55.20**	30	48.4	28

Configuration 4: this configuration is similar to the configuration 3. However, the rejection systems strategy (see section 3.5.3.2) is used to refine the outputs provided by the SVM without garbage class.

The experimental results are illustrated in the column "Configuration 4" of Table 4.10. The column "Configuration 2" illustrates the recognition results where the SCR is built without rejection option. The column "Configuration 3" shows the recognition results when using the SCR with garbage class.

Table 4.10: Experimental results of the HWR using a SCR with rejection systems, and a graph search word decoding strategy.

Lexicon	Configuration 4 (with rejection systems)		Configuration 2 (without rejection option)		Configuration 3 (with garbage class)	
	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)
818	71.1	5	70.4	3	**74.30**	3
5000	59.2	19	58.5	10	**64.20**	10
10000	54	24	53.2	16	**59.10**	17
20000	49	41	48.4	28	**55.20**	30

Based on these experimental results, we observe that:

- Using the rejection systems to create a rejection option in the SCR slightly improves the recognition rates. However, this strategy increases the computational time compared to the case where SVM with garbage class is used like in configuration 3.
- The recognition rates are also much lower than the system using the configuration 3 (using the SCR with garbage class).

Conclusion: from configurations 1 to 4, the directed graph search strategy is considered during the word decoding process. This search strategy provides encouraging results in

terms of recognition rates [112, 111]. However, the computational time considerably increases if the value of the pruning parameter (E) is increased. Concerning the parameter N, the same problem concerning computational time can be identified. We choose to use $E = 7$ and $N = 5$ as default values, since they provide a good trade-off result between the recognition rate and the computational time.

According to these experimental results, using the SCR with a rejection option (for both strategies) improves the recognition rates of the system. However, the garbage class strategy outperforms the rejection systems strategy in terms of recognition rate and computational time. Therefore, in the following experiments, only the garbage class strategy is considered at the single character level. In addition, dynamic programming is considered instead of the directed graph search strategy, what allows us to compare the effectiveness and efficiency of these two strategies.

4.2.3.2 Using dynamic Programming

In these experiments, we use dynamic programming (see section 3.7.3) in the word decoding process. The impact of the parameter N and the SCR with garbage class were already studied when using a directed graph search strategy in the sub-section 4.2.3.1. In this sub-section, we aim at observing their impacts on the system when the dynamic programming is used. Then, we compare the effectiveness of the system when using the SCR with and without rejection option. According to the previous experimental results, in the case of the SCR with rejection option, the garbage class strategy outperforms the rejection systems strategy. Therefore, in these experiments, we consider only the garbage class strategy. Finally, we compared these recognition results with those of the directed graph search strategy. The experiments and discussions are divided into the three following paragraphs.

A) Observing the impact of the parameter N

To observe the impact of the parameter N on the system, in this configuration (**Configuration 5**), the experiments are performed with 5 different values of N ($N = \{5, 7, 9, 11, all\}$). $N = all$ means that all the character candidates provided by the SCR are considered, for each node of the lattice. The SCR without rejection option (see $Exp2$) is used. The recognition results are given in Table 4.11.

Based on these experimental results, we observe that:

- The recognition rates are improved when increasing the value of the parameter N. However, when $N \geq 7$, the improvement remains very slight.
- Concerning the complexity of the system, increasing the value of the parameter N increases the computational time. In the case of $N = all$, its computational time can be tripled compared to the case with $N = 7$.

Table 4.11: Experimental results of the HWR with different value of the parameter N and by using dynamic programming during word decoding process (configuration 5).

Lexicon	N=5		N=7		N=9		N=11		N=all	
	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT(s)
818	49.7	1	72.8	1	72.9	2	73.1	2	73.3	3
5000	42.6	1	63.7	1	64	2	64	2	64.3	4
10000	39.6	1	58.7	1	58.8	2	58.9	2	59	5
20000	35.7	2	53.2	2	53.4	2	53.8	3	53.9	6

- By comparing the cases where $N = 7$ and $N = 9$, we observe that the differences of recognition rates are small (0.1% to 0.3% only), while the computational time can be doubled with small lexicon.

Based on the observations mentioned above, we decided to use $N = 7$ as a default value in the following experiments. Although, $N = all$ provides better recognition rates, the computational time when considering bi-character models will be much higher than considering $N = 7$. In the case $N = all$, $26 * 26$ bi-character candidates have to be verified instead of $7 * 7$ when using $N = 7$. In addition, the SCR with garbage class is taken into account.

B) Observing the effect of rejection in the SCR

The experimental results presented in sub-section 4.2.3.1 have shown that using garbage class strategy to build a SCR with rejection option allows to obtain significant improvements of the recognition rate when graph strategy is used for word decoding. In this paragraph, we also aim at observing the effectiveness of the system when using the garbage class strategy, but this time, by applying dynamic programming for word decoding. The experiments are performed using the configuration described below:

Configuration 6: in this configuration, the SCR with a garbage class (see *Exp.*3) is used. We consider $N = 7$, based on the previous experimental results. The experimental results of this configuration are shown in the column "Configuration 6", Table 4.12.

By comparing the recognition results in the columns "Configuration 6" and "Configuration 5" (without garbage class in the SCR), we observe that using a garbage class in the SCR improves the recognition rate of 3.5% to 4.4%, depending on the lexicon. In the following paragraph, we compare the effectiveness and efficiency of the directed graph search strategy and the dynamic programming for word decoding.

Table 4.12: Comparison of the recognition rates when the HWR uses the SCR with garbage class (configuration 6) with those of the HWR using the SCR without garbage class (configuration 5).

Lexicon	Configuration 6 (SCR with garbage class)		Configuration 5 (SCR without garbage class)	
	acc (%)	CT (s)	acc (%)	CT (s)
818	**76,3**	1	72,8	1
5000	**68,1**	1	63,7	1
10000	**62,5**	1	58,7	1
20000	**56,9**	2	53,2	2

C) Directed graph search strategy *vs.* dynamic programming

This paragraph compares the effectiveness and efficiency of the directed graph search strategy on the one hand and dynamic programming strategy on the other hand. Both strategies can be used in the word decoding process. The recognition results are given in Table 4.13. The columns "Configuration 2" and "Configuration 3" illustrate the recognition results when the HWR uses the directed graph search strategy, while the columns "Configuration 5" and "Configuration 6" illustrate the recognition results when the HWR uses the dynamic programming strategy.

Table 4.13: Comparison the recognition results given when the HWR uses directed graph search strategy and when the HWR uses the dynamic programming.

Lexicon	SCR without garbage class				SCR with garbage class			
	Configuration 5		Configuration 2		Configuration 6		Configuration 3	
	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)	acc(%)	CT (s)
818	**72.8**	1	70.4	3	**76.3**	1	74.30	3
5000	**63.7**	1	58.5	10	**68.1**	1	64.20	10
10000	**58.7**	1	53.2	16	**62.5**	1	59.10	17
20000	**53.2**	2	48.4	28	**56.9**	2	55.20	30

Based on this comparison, we observe that:

- The dynamic programming strategy outperforms the directed graph search strategy in terms of recognition rate, and computational time. This can be explained by the fact that dynamic programming provides an optimal path associated to each word in the lexicon while the directed graph search strategy may provide only local optimum; and therefore, the correct path may be lost due to pruning process.
- In the case for which the garbage class is not considered in the SCR, the HWR using dynamic programming provides 5% better recognition rates than when the HWR uses the directed graph search strategy. If the garbage class is considered, the difference is 4%.

Conclusion: After performing a series of experiments using 6 different configurations (from configuration 1 to 6), two important observations can be made:

- Using a SCR with rejection can improve the recognition rate of the system. We notice that the garbage class strategy provides better recognition results (both in terms of recognition rate and computational time) compared to the rejection systems (based on cascades of Adaboost classifiers) strategy.
- Dynamic programming clearly outperforms the directed graph search strategy in terms of recognition rates and computational time even though TRIE model strategy (which would speed-up the process) is not considered when using the dynamic programming (due to limitation of time at the end of this thesis).

In the following experiments, the bi-character models will be integrated in our system. Based on the experimental results given in this section, only dynamic programming is considered for word decoding.

4.2.3.3 Integration of bi-character models

In this sub-section, the experiments are performed by taking into account the bi-character models. These models are created by using logistic regression (see *Exp.*5). First, we evaluate the effectiveness of the bi-character models by using the SCR without rejection option. Then, we perform another experiments by taking into account the SCR with garbage class to observe the performance of the system when simultaneously using the garbage class and the bi-character models. In both cases, we consider only 7 potential character candidates in each node in the lattice ($N = 7$), based on the previous experimental results and by using dynamic programming for word decoding.

A) Using SCR without garbage class

In this configuration (configuration 7:), we use SCR without garbage class (*Exp.*2) and by integrating the bi-character models. The experimental results are provided in the column "Configuration 7" of Table 4.14.

By comparing with the recognition results without integrating the bi-character models (see column "Configuration 5", Table 4.14), we observe that:

- The bi-character models allow improving the recognition rates of 2.6% to 3.5%, depending to the lexicon size.
- However, using the bi-character models increases the computational time.

In order to select the optimal configuration, we decide to perform the experiments by using the SCR with garbage class and by considering bi-character models, as introduced in the following paragraph.

Table 4.14: The recognition results when using the SCR without garbage class and by integrating bi-character models.

Lexicon	Configuration 7 (Logistic Regression)		Configuration 5 (without bi-character models)	
	acc (%)	CT (s)	acc (%)	CT (s)
818	**75.4**	6	72.8	1
5000	**66.4**	7	63.7	1
10000	**62.3**	8	58.7	1
20000	**56.7**	9	53.2	2

B) Including a garbage class in the SCR

Based on the experimental results presented in paragraph-B of section 4.2.3.1 and paragraph-B of section 4.2.3.2, we observed that using the SCR with a garbage class significantly improves the recognition rates. Furthermore, according to the experimental results in paragraph A of section 4.2.3.3, we observe that the bi-character models also improve the effectiveness of the proposed system. In this configuration (**configuration 8**), we evaluate the effectiveness of the garbage class strategy (using SCR in *Exp.*3) and of the bi-character models (*Exp.*5) when both of them are simultaneously considered in our HWR. The experimental results are provided in the column "Configuration 8" of Table 4.15.

Table 4.15: Comparison of the recognition results when our system uses bi-character models and SCR with garbage class simultaneously.

Lexicon	With bi-character models (Logistic Regression)				Without bi-character models			
	Configuration 8 (with garbage class)		Configuration 7 (without garbage class)		Configuration 6 (with garbage class)		Configuration 5 (without garbage clas)	
	acc (%)	CT (s)	acc (%)	CT (s)	acc (%)	CT (s)	acc (%)	CT (s)
818	**77.6**	6	75.4	6	76.3	1	72.8	1
5000	**69.4**	7	66.4	7	68.1	1	63.7	1
10000	**65.7**	8	62.3	8	62.5	1	58.7	1
20000	**59.4**	9	56.7	9	56.9	2	53.2	2

The column "Configuration 7" illustrates the recognition results when the HWR takes into account the bi-character models, but without garbage class in the SCR. The column "Configuration 6", on the other hand, illustrates the recognition rate when the HWR does not integrate the bi-character models, but when the SCR with garbage class is considered. The column "Configuration 5" presents the recognition results of our basic HWR, when the garbage

class and the bi-character models are not considered.

Based on these recognition results, we observe that:

- When the bi-character models are considered, the garbage class allows an improvement of 2.2% to 3.4% of the recognition rates, depending on the lexicon (configuration 7 *vs.* configuration 8). However, let us notice that the improvement is smaller compared to the one obtained when the system does not take into account any bi-character models (3.5% to 4.4%), as explained earlier in paragraph B, sub-section 4.2.3.2.
- By comparing the recognition results of the column "Configuration 8" and those of the column "Configuration 6", we can observe that when considering the SCR with garbage class, the bi-character models allow an improvement of 1.3% to 3.2% of recognition rates, depending on the lexicon size. This improvement is however smaller than the one obtained in the case where the system does not take into account the garbage class in the SCR (2.6% to 3.6%), as explained in paragraph-A of this section.
- By comparing configuration 8 with our basic HWR (column "Configuration 5"), we can notice that using simultaneously the garbage class in the SCR and the bi-character models allows an improvement of 4.8% to 7% of the recognition rates, depending on the lexicon size.

Conclusion: This section evaluates the effectiveness of our HWR when integrating the bi-character models. Based these experimental results, we can conclude that using the bi-character models significantly improves the recognition rates of our HWR. The level of improvement varies according to the configurations and lexicons. In the case that the garbage class strategy is not considered, the bi-character models provide an improvement of 2.6% to 3.6% of the recognition rates. However, in the case where the garbage class strategy is used, the bi-character models provide an improvement of 1.3% to 3.2% of the recognition rates. The improvement is smaller, probably because the garbage class and the bi-character models may solve some common ambiguities/errors. Nonetheless, simultaneously considering the garbage class and the bi-character models allows an improvement of 4.8% to 7% of the recognition rates. The "Configuration 8" (where the SCR based on SVM with garbage class, bi-character models, and the dynamic programing word decoding procedure are used) is considered as the best configuration of our system. However, it is important to mention that using the bi-character models increases the computational time. We identified several possibilities for improving the global complexity and the speed of the system. These opportunities will be discussed in the conclusion and perspectives chapter of this thesis.

In all the experiments presented above, we considered that all the characters had the same maximum number of graphemes ($L = 7$) despite their differences. In the following

experiments, we take into account the maximum number of graphemes adapted to each character class (see section 3.7.3.2).

4.2.3.4 Effect of the adaptation of the maximum number of graphemes to the character size

As explained earlier in section 3.7.3.2, each character class may have a different maximum number of graphemes. In this section, we evaluate the effectiveness of the system when using the maximum number of graphemes adapted to each character class (see Table A.3 in Appendix A).

Indeed, some nodes in the lattice may contain a set of concatenated graphemes such that the number of graphemes in the node is higher than the number of graphemes constituting the character to recognize. These nodes are potentially related to unknown patterns. Anyway, they do not correspond to the character to recognize. When using the maximum number of graphemes adapted to each character class, these nodes will not be considered during the word decoding process. Therefore, the problems related to unknown patterns may be reduced. In other words, this strategy also helps handling unknown patterns problem.

In this experiment (**configuration 9**), we decided to keep on using SCR with garbage class, bi-character models and dynamic programming in the word decoding process, since this configuration (see configuration 8) outperforms all others configurations. The recognition results of this system without and with adaptation to the character size are provided in the columns "Configuration 8" and "Configuration 9" of Table 4.16 respectively.

Table 4.16: Recognition rates when using a maximum number of graphemes adapted to each character class (configuration 9) or a fixed number $L = 7$ (configuration 8).

Lexicon	Configuration 9 (adaptation to character size)		Configuration 8 (without adaption to character size)	
	acc(%)	CT (s)	acc(%)	CT (s)
818	76.7	6	**77.6**	6
5000	**69.6**	7	69.4	7
10000	**65.9**	8	65.7	8
20000	**60**	9	59.4	9

Based on the comparison of the recognition results given in Table 4.16, using the maximum number of graphemes adapted to character class slightly improves the recognition rates only when using a lexicon of large size. This can be explained by the fact that almost of the unknown patterns in some nodes of the lattice are already handled by

the garbage class strategy (in the SCR) and bi-character models. Concerning the computational time, we do not see any difference. In order to observe the improvement in terms of computational time, we need to record the computational time in millisecond. Therefore, in our system, we will not consider this strategy.

As mentioned in section 2.2.1, the delayed strokes can cause the additional variation in on-line handwriting. In order to deal with the problem caused by the delayed strokes, we have presented a delayed stroke management method, as explained earlier in section 3.3. This method consists in detecting and re-localizing the delayed strokes in the lattice based on its spatial position on the x axis. The experiment on this method is performed in the following section.

4.2.3.5 Observing the effectiveness of the delayed strokes management method

In this section, we aim at observing the effectiveness of the delayed strokes management method (see section 3.3) by performing the experiments on our system with another configuration. This configuration (**Configuration 10**) is the same as the configuration 8 where the SCR with garbage class and bi-character models (based on logistic regression) are used. However, in configuration 10, we do not consider the delayed stroke management method.

The recognition results are illustrated in the column "Configuration 10" of Table 4.17. By comparing with those in the column "Configuration 8" where the delayed stroke management method is considered, we can notice a average degradation of 3% of the recognition rate. In other words, using our delayed stroke management method significantly increases the effectiveness of the system. We can also conclude that the delayed strokes play a very important role for on-line HWR. Therefore, these kind of strokes must not be ignored during the recognition process.

Table 4.17: Comparison of the recognition rates given by our system when the delayed stroke management method is not considered and when the delayed stroke management method is considered.

Lexicon	Configuration 10 (without delayed stroke management method)		Configuration 8 (with delayed stroke management method)	
	acc (%)	CT (s)	acc (%)	CT (s)
818	74.3	6	**77.6**	6
5000	65.9	7	**69.4**	7
10000	62.5	8	**65.7**	8
20000	57.2	9	**59.4**	9

It is important to observe the recognition rate for each writer, since different writers may have different writing styles. Their writings are more or less complex to be read/recognized, even by a human being. An automatic handwritten words recognition system can meet the same problem. The following experiments aim at reporting the recognition rate of different writers.

4.2.3.6 Effect of the writing styles on the system performances

In the previous experiments (from experiment 1 to 10), we evaluated the effectiveness of the different methods proposing for each stage of our system: pre-processing, character analysis, bi-character analysis and word decoding process. In all these experiments, we tested our system with only 1000 handwritten words, randomly selected from the Unipen-ICROW-03 (see section 4.1.3). In this section, we aim at observing the recognition rates of different writers. Therefore, we performed the experiments with all the handwritten words in the Unipen-ICROW-03. Let us remind here that our research focuses on the handwritten words composed of lowercase alphabet. Therefore, only 12440 handwritten words (out of 13119 handwritten words) of 67 writers (out of 72 writers) are selected and considered in this experiment. The selection process is automatically performed based on the ground-truth of each handwriting sample. In this configuration (**Configuration 11**), the SCR with garbage class and the bi-character models are considered, since this configuration provides better recognition results. The observations on the experimental results are presented in two following paragraphs.

A) Recognition rate as a function of the writer

In this paragraph, we focus our observation on the recognition rates of each writer. Figure 4.11 illustrates the recognition rate by writer when using lexicons of 818 words and 20000 words. Based on these recognition results, we observe that:

- Considering a small lexicon of 818 words, the recognition rates vary from 9.09% (for the writer "NIC-Pc95-koen.dat") to 98.33% (for the writer "NIC-Oli92-pietro.dat"), highlighting the dependence to the writer.
- In the case of a large lexicon of 20000 words, the recognition rates vary from 5.74% (for the writer "NIC-Pc95-koen.dat") to 95.92% (for the writer "NIC-Lo93b-stephani.dat").

The low values of recognition rates provided for some writers can be explained by two different reasons:

- The database (Unipen-ICROW-03) contains some errors, especially in the ground-truth. For instance, the writer "NIC-Pc95-koen.dat" wrote some words which were composed of uppercase characters, as illustrated in Figure 4.12(a). However, in the ground-truth, the corresponding labels are in lowercase characters. As mentioned above, our HWR focuses only on handwritten words which are composed of lowercase characters. In these experiments, we selected the test samples based on the ground-truth. Therefore, the writings provided by "NIC-Pc95-koen.dat" were also selected while they were erroneous. This kind of error may also happen for some other writers. We did not manually remove this kind of errors because we wanted to experiment our system with the raw data, so that, other authors can use these experimental results as a

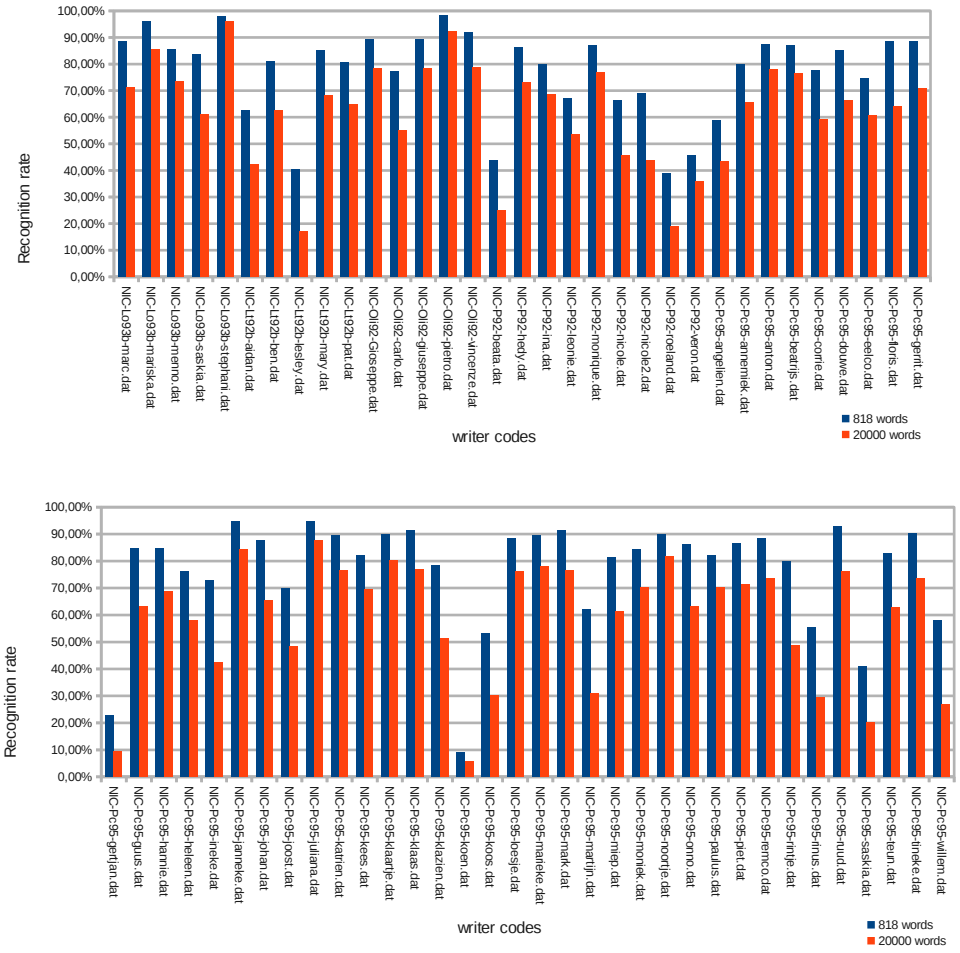


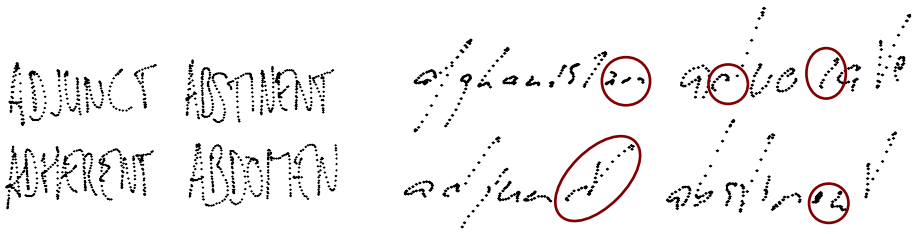
Figure 4.11: Handwriting recognition rates by writer for all the writer of the Unipen-ICROW-03 database (displayed on two rows for space reasons).

reference for comparison with their own system. Doing like this, the comparison is somewhat fair in terms of test data.

- The other reason for this low recognition rates is related to very bad quality of the handwriting. For instance, in the writings provided by the writer "NIC-Pc95-gertjan.dat", some characters are incorrectly written (see the red circles, Figure 4.12(b)). Sometimes, it is almost impossible to recognize, even by a human being. The associated errors can therefore be considered as acceptable.

We can also observe the number of writers in the different ranges of recognition rates (see Figure 4.13). We notice that:

- Considering a lexicon of 818 words, the proposed HWR provides recognition rate higher than 75% for 48 writers (out of 67 writers). For 12 writers, the system provides recognition rates between 50% to 75%.



(a) writer "NIC-Pc95-koen.dat" (b) writer "NIC-Pc95-gertjan.dat"

Figure 4.12: Handwriting samples provided by two different writers: NIC-Pc95-koen.dat and NIC-Pc95-gertjan.dat, extracted from Unipen-ICROW-03 database.

And for the remaining 7 writers, our system provides recognition rates below 50%.

- Considering a lexicon of 20000 words, only 19 writers obtain a recognition rate higher than 75% while the 30 other writers obtain a recognition rate between 50% and 75%. 18 writers obtain a recognition rate below 50%.

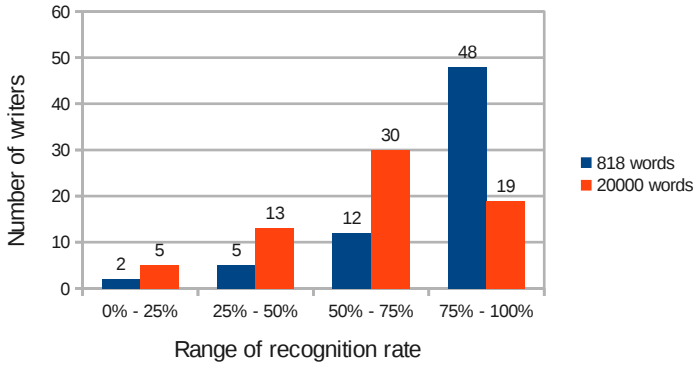


Figure 4.13: Number of writers for each range of recognition rates, depending on the lexicon size.

B) Effect of the writer on the recognition rates, depending on the lexicon size

Figure 4.11 shows the recognition rates as a function of the writer with two different lexicons (818 words, and 20000 words). In this paragraph, we aim at observing the recognition rates of each writer when the size of lexicon is increased.

First, let us observe the average recognition rates (acc) and Computational Time (CT) over all the writers when the size of lexicon is increased from 818 to 20000 words, as illustrated in Table 4.18. We can see an average decrease of 16.5% of the recognition rates while the computational time is more than doubled. This is somewhat normal for an automatic handwriting recognition system in such a context.

Table 4.18: Average recognition rates over all writers, depending on the size of the lexicon.

Lexicon	Configuration 11	
	acc (%)	CT (s)
818	77,72	4
20000	61,23	9

If we observe more in details the recognition rates by writer for different lexicon sizes (818 words and 20000 words), as illustrated in Figure 4.11, we can see that:

- For some writers, the recognition rates are slightly decreased, when the lexicon size is increased from 818 words 20000 words ($\sim$ 24 times larger). For instance, the recognition rates of the writers "NIC-Lo93b-stephani.dat" and "NIC-Oli92-pietro.dat" are respectively decreased from 97.96% to 95.92% ($\sim$ 2%) and from 98.33% to 92.31% ($\sim$ 6%).
- For some other writers, on the other hand, the recognition rates are dramatically decreased. For instance, in the case of writer "NIC-Pc95-rintje.dat", the recognition rate is decreased from 80% to 48.89% ($\sim$ 31%). This phenomenon might be explained by the writer's handwriting style (which may be different from the other writers' styles in the database) and the increase of the possible number of ambiguities when increasing the lexicon size.

Conclusion

The recognition rates of the proposed HWR vary a lot according to the writer. For some writers, our system provides recognition rates higher than 75%, which can be considered as a good performance in terms of effectiveness. For some writers, the recognition rates can even be higher than 90%. At the opposite, our system may provide some recognition rates below 50% for some writers. This important heterogeneity in the results can be explained by the high variation of handwriting styles. As a consequence, some handwritten words provided by some writers are more complex and difficult to be analyzed by our system, especially when the lexicon size is increased.

In order to improve the recognition rates for this group of writers, a HWR adapted to each writer may be required, as explained in the perspective of this thesis.

In the following section, we summarize all the experimental results provided by the series of experiments conducted on our HWR and draw some general conclusion.

4.2.3.7 Summary and conclusion

In the previous sections, we presented a series of experiments concerning our system, performed with 11 different configurations. The experimental results when using the configuration 1 to 10 are illustrated in Figure 4.14.

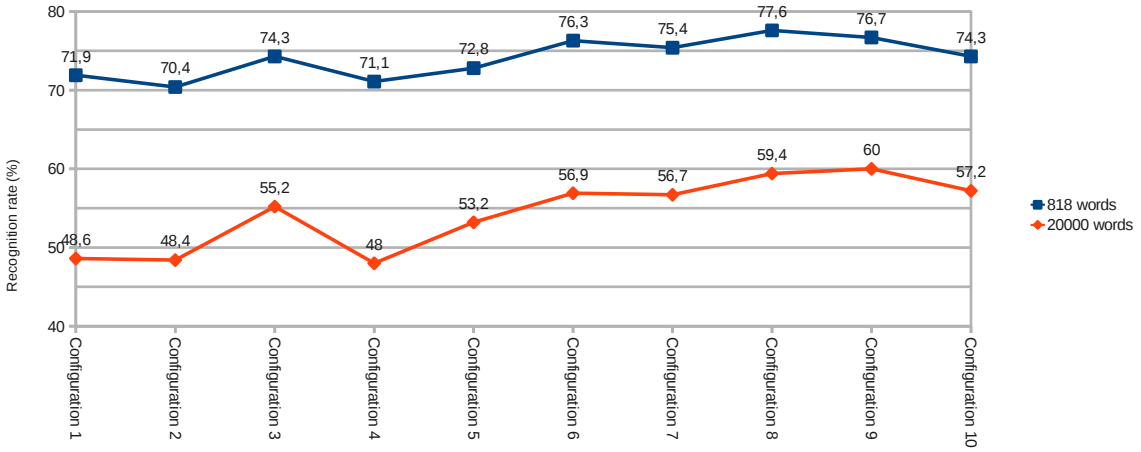


Figure 4.14: Experimental results given by our system over all the different configurations.

From configuration 1 to configuration 4, the directed graph search strategy is used during the word decoding process. Only the SCR is considered. The bi-character models are not integrated yet. Configurations 1 and 2 allow observing the impact of the pruning parameter E and the parameter N (number of potential characters considered in each node of the lattice) respectively. In the configurations 3 and 4, we evaluate the effectiveness and efficiency of the garbage class strategy and the rejection systems strategy, which are used to create rejection option in the SCR. Based on all these experimental results, we observe that using the garbage class strategy significantly improves the recognition rate.

In the remaining configurations, we use dynamic programming for word decoding. The configurations 5 and 6 allow comparing the effectiveness of our system when using the SCR without garbage class and the SCR with garbage class. The garbage class strategy improves the recognition rates. In addition, dynamic programming outperforms the directed graph search strategy both in terms of recognition rates and computational

times.

In configurations 7 and 8, we integrate the bi-character models (based on logistic regression) in our systems. In the configuration 7, we use the SCR without garbage class while in the configuration 8, we use the SCR with garbage class. Based on the recognition results, we can conclude that using together the SCR with garbage class and the bi-character models (configuration 8) allows significant improving of the recognition rates. When integrating the bi-character models in HWR, the computational times increases. However, we can speedup the process if the TRIE model is used to represent the lexicon, as explained in section 5.2.1 (the future work related to this thesis).

In configuration 9, we use dynamic programming adapted to character size. Instead of considering that all the character classes have the same size, we adapt the character size (number of maximum graphemes composing a character) to each character class. This strategy allows slightly to improve the recognition rates ($\leq 0.6\%$) when using a large lexicon size. This can be explained by the fact that the ambiguities related to the unknown pattern are solved by the garbage class strategy in the SCR and bi-character models. Therefore, in the following, we use a fixed number of graphemes per character in the word decoding process.

In the configuration 10, we remove the delayed stroke management method from our system to prove its efficiency. When the delayed stroke management method is not used, the effectiveness of the system is degraded of an average of 3%, what shows the effectiveness of our delayed stroke management method. We can also conclude that delayed strokes contain important information about the words and that this information has to be considered in the handwriting recognition systems.

Finally, we observe the recognition rates of different writers (configuration 11). We notice large differences between the performances, depending on the writer. The low recognition rates for some writers can be explained by two main reasons: 1) labeling errors in the ground-truth and 2) the high complexity in their writings.

In the next section, in order to be as exhaustive as possible, we propose to compare the effectiveness and efficiency of our system with a baseline HMM-based system.

4.3 Comparison with a baseline HMM-based system

Based on our survey on handwritten word recognition systems (see chapter 2), we can see that Hidden Markov Model (HMM) is very frequently used to build handwritten word/text recognition systems, especially in the case of on-line signal. As a consequence, HMM-based systems are frequently used as a reference system in the literature [124, 43, 9]. Therefore, we decided to use a baseline HMM-based system as a reference system for performance comparison.

This section aims at comparing the effectiveness and efficiency of our system with a baseline HMM-based system. A paper related to this work is presented in [113]. First, we briefly introduce this HMM-based system. Then, we introduce the experimental protocol and analyze the experimental results.

4.3.1 Presentation of the baseline HMM-based system

A brief introduction to HMM is presented earlier in section 2.3.1.1. Let us remind here that a HMM is a model that describes the transitions between a set of states $S = \{s_1, s_2, \dots\}$ for a given set of observations $O = \{o_1, o_2, \dots\}$. A HMM is denoted by a model $\lambda = \{A, B, \pi\}$. In this part, we present the architecture and parameters of the HMM-based system that we use as a baseline here. Then, we introduce the training process of this system, which allows us to optimize the values of some important parameters, such as the number of Gaussians in the mixture and the number of states for each character model. The settings and training method we use here are inspired from the system presented in [47].

Architecture and definitions of HMM-based system

- The baseline HMM-based system relies on a linear architecture, which allows self-transition and transition to its immediate successor state (see Figure 2.5 in section 2.3.1.1).
- The input signal is normalized using the same word normalization and pre-processing method as in our system (see section 3.2.1).
- Three families of local on-line features which are usually used in the literature [65] for HMM-based systems, are extracted from each point of the normalized signal. These features are: normalized x, y coordinates, sine and cosine of the curvature angle as well as sine and cosine of the direction angle.
- The size of the sliding window is settled to 4 points and the overlapping width is settled to 2 points.
- The average length (*i.e.* average number of sliding window) of each character model ($Ns(c_k)$) is estimated from the training database (see Appendix B). Table B.1, in Appendix B gives the resulting average length of each character class. These values are then used to estimate the number of states for each character model during the training process.
- The number of Gaussians in the mixture and the number of states for each character model is optimally settled using a validation database, as explained in the description of the training process presented in the following paragraph.

Training process

The reference system is trained using 72028 writings (in lowercase characters) selected from the UNIPEN database [49]. The number of Gaussians in the mixture and the numbers of states for each character model are optimally fixed by using a validation set which contains 800 writings, randomly selected from the Unipen-ICROW-03 database. The training and validation process are described below:

Based on the estimated average length of each character class ($Ns(c_k)$), we want to find the optimal number of states of the HMM model for the character c_k. Indeed, this HMM-based system relies on linear architecture which allows self-transition. In other words, it allows repeating several transitions in the same state. As a consequence, the number of states for the HMM model corresponding to the character c_k is generally lower than its average length $Ns(c_k)$. In this reference system, we define the number of states $N_{state}(c_k) = \eta * Ns(c_k)$, where η is the coefficient value ranging in $\{0.1, 0.2, \ldots, 1\}$. For each coefficient η, a complete training process is performed. The training process consists in three stages:

- Stage 1: for each character class c_k, an initial HMM model $\lambda^0_{c_k}$ is created. The number of states is fixed to $N_{state}(c_k) = \eta * Ns(c_k)$ while the number of Gaussians in the Mixture is fixed to 1.
- Stage 2: the system is trained in 4 iterations using the handwritten words in the training database. After these 4 iterations of training, we perform an experiment on the validation dataset. The recognition rate value is recorded. Then, we increase the number of Gaussians in the mixture by 1.
- Stage 3: We re-perform the stage 2 recursively, until the the recognition rate on the validation database remains stable.

Table 4.19 illustrates the recognition rate on the validation set when increasing the number of Gaussians in the mixture, and with different values for the coefficient η. Based on these results, we observe that when the number of Gaussians in the mixture equals 10 and the coefficient η equals 0.5, the HMM-based system provides the best recognition rate. When using the coefficient $\eta = 0.5$, the number of states per character varies from 9 to 14, depending on the character class. These values are considered during the test stage, which is introduced in the following section.

Table 4.19: Recognition rates on the validation database.

Number of Gaussians in the mixture	Coefficient of number of states (η)									
	0,1	0,2	0,3	0,4	**0,5**	0,6	0,7	0,8	0,9	1
1	0.25	3	10.5	18.75	24.5	25.62	17.5	10.02	7.38	3.5
2	0	6.88	27.25	41.12	44.38	40.88	31	21.15	13.64	6.76
3	0.25	9.5	34	47.75	50.62	47.25	36.75	25.66	15.64	8.64
4	0.5	15.38	43.12	53	57.62	50.25	38.75	26.66	16.77	9.14
5	0.62	15.55	46.12	58.88	60.62	52.38	40.5	28.54	19.02	9.64
6	0.75	20.12	49.75	61.12	64	53.12	41.75	30.04	19.02	9.76
7	1	23.62	51.62	62.75	65.38	55.5	44.5	30.41	19.77	10.14
8	1.25	27	52.62	65.12	68.12	57	44.75	31.54	20.18	10.14
9	1.25	27.62	52.5	65.62	68.5	58.12	45.25	32.79	20.78	10.14
10	2.25	27.88	52.75	66.25	**69.12**	58.12	45.75	32.54	21.03	10.26
11	2.25	27.25	53.25	66.75	69	57.88	45.5	32.42	21.03	10.76
12	2	28	53.5	66.5	69	57.75	46	32.92	21.4	10.64
13	2.38	28.12	54.75	66.25	68.62	57.38	46.12	33.04	21.53	10.76
14	2.75	28.38	55.25	66.38	68.25	57.62	46.5	33.42	2053	10.76

4.3.2 The proposed system *vs.* the baseline HMM-based system

In order to compare the effectiveness and efficiency of our system with the baseline HMM-based system, we perform the experiments on both systems using the same test database and lexicons. This test database contains 3614 writings (in lowercase characters) selected from the Unipen-ICROW-03 database. Concerning our system, we use the configuration where the SCR with garbage class and the bi-character models are considered. The recognition results are provided in Table 4.20.

Table 4.20: Experimental results of the proposed system and the HMM-based system: accuracy (recognition rate) and Computational Times (in seconds).

Lexicon	Our proposed system		HMM-based system	
	acc(%)	CT (s)	acc(%)	CT (s)
5000	**70,84**	6	65,15	9
10000	**67,02**	7	60,19	22
20000	**62,59**	9	54,51	40

Based on these recognition results, we can observe that using the lexicon of 5000, 10000 and 20000 words, the recognition rates of our proposed system are respectively 5.69%, 6.83% and 8.08% higher than the recognition rates of the HMM-based system. We can notice that the improvement becomes larger when increasing the lexicon size. This can be explained by different reasons:

- Features: our system relies on the combination of off-line and on-line features, which allows taking profit of their complementarity. Unlike our system, the baseline HMM-based system relies only on a set of local features.

- Segmentation: our system relies on explicit segmentation/recognition, whereas the baseline HMM system relies on implicit segmentation. Therefore, more information is given to our system during the training stage (we use a database of segmented characters) compared to the HMM, the training of which is "blind" since the character boundaries are unknown.

- Single character recognizer: the SCR used in our system relies on SVM, which is famous for its effectiveness for discriminant classification. In general, it is more effective for classification than HMM, which is a generative model.

- Bi-character models: the bi-character models used in our system allow integrating the graphical context by taking into account the neighboring characters which is not the case for the baseline HMM.

Concerning computational times, our system performs 4 times faster than the baseline HMM-based system when considering a lexicon of 20000 words. This is due to the fact that, in our system, for each node of the lattice, only 7 potential character candidates ($N = 7$) are considered. This strategy renders the word decoding process much faster, since some paths are pre-filtered. This strategy is not used in the baseline HMM-based system. As far as we know, this strategy was not used in any existing HMM-based system.

4.4 Conclusion

In this chapter, we present and analyze many experiments respectively for: i) Isolated Character Recognizer (ICR), ii) Single Character Recognizer (SCR), iii) bi-character models and iv) Handwritten Word Recognition (HWR). Detailed analysis and discussions were presented in the corresponding sections and sub-sections. In this conclusion, we point out some main conclusions.

Our ICR was trained for isolated character recognition using isolated characters randomly selected from the IRONOFF and UNIPEN databases. It uses SVM and 45 features resulted from feature selection (see section 3.4.3). These features consist in both on-line and off-line features. Based on the experimental results, our ICR provides very competitive recognition results in terms of recognition rate as well as computational times compared to a state-of-the-art recognizer. This first experiment allows assessing the effectiveness and efficiency of our recognizer compared to the state-of-the-art. We can therefore apply it to single characters (extracted from handwritten words).

The SCR applied to single characters was trained mainly with the single character samples segmented from handwritten words. This recognizer relies on 72 features resulting from feature selection. In general, the recognition rates obtained on single characters is lower than the recognition rates obtained on isolated characters. This is because the shapes of the characters segmented from handwritten words are more heterogeneous and

more complex than those of the characters written in isolated boxes. The inclusion of the garbage class in the SCR does no have a great impact at the recognition rates at the character level.

Concerning our HWR, we performed a series of experiments using 11 different configurations, in order to evaluate the effectiveness and efficiency of the methods/strategies used at each stage of the system and to observe the recognition rates for each writer. Based on the experimental results, we can notice that simultaneously using the SCR with garbage class and the bi-character models based on logistic regression provides an improvement of 4.8% to 7% of the recognition rates, depending on the lexicon size. However, the effectiveness of the system strongly depends on the writers. In this case, a HWR adapted to each writer may be required [18, 145].

We have also compared the effectiveness and efficiency of our system with a baseline HMM-based system by testing both systems with the same handwritten words and lexicons. Our system provides better results in terms of recognition rates, especially in the context of large lexicons (up to 8% improvement for a lexicon of 20000 words) and computational time ($\sim$ 4 times faster for a lexicon of 20000 words). However, the HMM-based system used for this comparison is a baseline system. The effectiveness and efficiency of this HMM-based system could certainly be improved. Nonetheless, we have shown that an explicit segmentation/recognition based system using a discriminative classifier can provide a very competitive results, compared to an implicit segmentation-based system relying on generative models.

5 Conclusion and future work

This chapter gives a final conclusion of our research by discussing on the strong and weak points of the proposed system. Finally, we present the perspectives of this work which will allow improving the effectiveness of the system.

Contents

Handwriting is an irreplaceable way for communication. The explosion of interactive mobile devices such as smartphones, smartpens, electronic tablets, etc., offers alternative ways to produce handwriting. These devices store the trajectory of handwriting as sequences of points, generally called on-line handwriting. Nowadays, these kinds of devices are used for different purposes, for instance education, medical and administrative tasks, which increases the amount of on-line handwriting produced every day. The research activities related to on-line handwriting recognition raised a great interest not only in the research community, but also from a commercial point of view.

The intrinsic difficulties in handwriting recognition problems are mainly due to the large variability of writing styles all over the world. In Latin script, a handwritten word is composed of a sequence of characters which are more or less connected, depending on handwriting style. The character boundaries are difficult to identify, especially in the context of unconstrained handwriting. Therefore, unconstrained handwritten word recognition is still considered as a difficult problem in the research community, despite the efforts brought by many researchers since half a century.

In this research work, we propose an on-line HWR relying on explicit segmentation/recognition and discriminative classifiers at the single character level and also at the bi-character level so as to integrate context information about the neighborhood of single character. The proposed system can be applied to a large and flexible lexicon without any specific requirement concerning to the capturing device. In this chapter, we summarize our main contributions during this PhD thesis. We also try to objectively discuss on the limitations of the proposed system and try to suggest future work both at a short term and a long term perspectives.

5.1 Summary

Motivated by the effectiveness of the SVM classifier on many pattern recognition problem, our first approach was to develop a first version of on-line HWR relying on an explicit segmentation/recognition method and a SCR using SVM with RBF kernel. The input handwritten word is normalized in order to standardize the signal and to remove noise. Then, this normalized signal is segmented into a sequence of graphemes to be further used to create a L level lattice. Each node of the lattice is considered as a character to be submitted to a SCR. The feature set used for this SCR was established on the basis of the combination of on-line and off-line features, to take profit of their complementarity. The off-line features were extracted from the image artificially generated from the original (on-line) signal. Finally, for the word decoding stage, we have implemented a directed graph search strategy, searching in the lattice to find the best path for each word in the lexicon.

This first system provides encouraging results, but its performances remain below the baseline HMM-based system results. After studying in details the framework of this first

system, we identified five problems related to: 1) the unknown patterns, since the signal in each node of the lattice is obtained by concatenating different segmented graphemes 2) the problem related to the delayed strokes, 3) the similarity between different character classes, for instance between the characters 'e' and 'l', 4) the shared character parts problem, where the shape of a part of some character classes may be similar to the shape of other character classes, 5) the fact that the directed graph search strategy can find a local optimum and therefore fail to provide an optimal path associated to each word in the lexicon. In addition, the pruning parameter E has to be optimally fixed to obtain a trade-off between computational times and recognition rates.

In order to deal with the problems related to unknown patterns, we have proposed to use two different strategies to build a SCR with rejection option, as explained in section 3.5.3. The first strategy aims at adding a garbage class in the SVM classifier while the second strategy aims at using a set of specific rejection systems (cascade of Adaboost classifiers) to post-process the outputs given by the SVM classifier (without garbage class in that case). We observed that the garbage class strategy outperforms the specific rejection systems strategy in terms of recognition rates and computational times (see part B, section 4.2.3.1). The garbage class strategy provides an improvement of 3.9% to 6.8% of the recognition rate compared to using a SCR without rejection option, depending on the lexicon size. The improvements brought by the garbage class strategy makes our system more competitive than the baseline HMM-based system, at least in terms of recognition rates.

In the particular case of on-line handwriting, delayed strokes may cause additional variations, or distortions in the writing. In order to deal with this problem, we have introduced a delayed strokes management method (see section 3.3). This method aims at detecting the delayed strokes and re-localizing them based on their spatial position on the x axis. This method allows to improve the recognition rate up to 3%, depending on the lexicon size (see section 4.2.3.5).

In order to tackle the local optimum and computational time problems, we have proposed another word decoding strategy relying on Dynamic Programming to find the optimal path associated to each word in the lexicon. In the case of large lexicon (20000 words), using dynamic programming in the word decoding process allows to improve the recognition rate up to 4% compared to the recognition rate provided by the graph search strategy (see part C, section 4.2.3.2). In terms of computational time, dynamic programming performs 15 times faster than the graph search strategy, even thought in this new word decoding method, the TRIE model strategy is not yet used to represent the lexicon which would speed-up the process.

In order to solve the problems related to the similarity between different character classes and the problems related to the shared character part, we have proposed the idea of bi-character models (see section 3.6). These bi-character models allow taking into account the graphical context each of single character candidate by jointly recog-

nizing two neighboring characters in the lattice. In addition, the bi-character models (based on logistic regression) can also solve the problem related to the unknown patterns which have not been handled by the garbage class in the SCR. Using bi-character models provides an improvement up to 3.6% in the case of large lexicons (20000 words). In the case that the garbage class strategy and the bi-character models are jointly used, the recognition rate is improved up to 7% compared to our baseline system (without garbage class nor bi-character models), as detailed in section 4.2.3.3. Our system provides a better recognition rate up to 8% and is 4 times faster than a baseline HMM-based system (see section 4.3).

In summery, we have proposed a discriminative approach to create an on-line HWR that combines explicit segmentation/recognition with discriminative classifiers (SVM and logistic regression) and dynamic programming. Our system analyzes the input handwritten word at two levels (character and bi-character levels), what allows this system to outperform a baseline HMM-based system in terms of recognition rates and computational times. Our research work open some new directions for on-line handwriting recognition problems.

Although our system provides some satisfying result compared to a baseline HMM-based system, it still has some limitations and weak-points that may be improved in the future.

5.2 Limitations of the proposed system and possible improvements

We classify these limitations and future work into two categories: short term and long term perspectives.

5.2.1 Short term perspectives

In the short term perspective, four main points can be improved:

Improving the training databases

As mentioned in section 4.2.1.2, some character classes do not have enough segmented characters. Therefore, the training database may not contain enough data to cover all the variation of characters composing handwritten word. In order to train the SCR, we therefore need to add more samples of these classes in the single character training database. In this case, the single character segmentation method presented in section 4.1.4 can be used to extract segmented character from handwritten words in some large database such as UNIPEN.

Additionally, the bi-character database contains only the samples which are artificially generated from single characters (see section 4.1.5), which may produce some incorrect samples in the training stage. This bi-character database can be improved by two methods: 1) cleaning the database by semi-automatically removing the incorrect samples and 2) add real bi-character samples segmented from handwritten words in the bi-character database. For this second method, a semi-automatic bi-character segmentation method similar to the single character segmentation method presented in section 4.1.4 can be used. We consider that using the current version of our HWR to segment bi-character samples from training handwritten words could be a good strategy. Indeed, the segmentation method can provide correct samples without requiring strong effort to do the manual verification.

Using TRIE model to speedup word decoding process

During the word decoding process, in the case where dynamic programming is used, we find the optimal path for each word in the lexicon by exploring the lattice word by word (*i.e.* flat search strategy). In order to reduce the computational time, we could easily use the TRIE model to represent the lexicon. In this case, words that share the same prefix share also the same branch in the TRIE model. The optimal path of this prefix is searched only one time and can be used for different words that share this prefix. Let us take an example of the words "when" and "where". These two words share the same prefix "whe". The optimal path of this prefix can be used to find the optimal path of the word "when" and "where". The recognition results remain of course unchanged when the TRIE model is used, as in any case, only the words in the lexicon are used.

Recognizing words containing uppercase characters

In order to recognize the handwritten words which contain mixed-case characters (lowercase and uppercase), the SCR has to be able to recognize at the same time the lowercase and uppercase characters and the bi-character models have to be able to deal with a pair of character composed of lowercase and/or uppercase characters.

In the case of SCR, adding the uppercase character classes in the SVM classifier would increase the number of classes in the SVM, which would certainly decrease the performances of the whole HWR. In this case, this SCR could be improved by experimenting different solutions. For instance, it could be possible to study further features, as well as to combine different classifiers at different levels [152] or even to use multiple-kernel SVM [41] to improve the performances of the recognizer.

In the case of bi-character models, directly adding uppercase character classes would significantly increase the number of bi-character classes up to 2704 classes (52*52 classes). As a consequence, 2704 bi-character models would have to be created. The performances of the proposed system would certainly dramatically decrease, mainly due to the high number of classes. However, in real applications, there is only few characters written

in uppercase, since the uppercase characters are written only at the beginning of sentences and only in some nouns and acronyms. For instance, in the Unipen-ICROW-03 database (see section 4.1.3), only 5% of words contain uppercase characters. In addition, the number of uppercase characters is less than 1% of the total characters in the database. Therefore, we can continue using only the bi-character models of lowercase bi-characters (676 bi-character classes), by applying these models only in the case where the two neighboring candidate characters are written in lowercase. In the case where at least one of the two character candidates given by the SCR is an upperase character, we would consider that the bi-character model returns the value 1 ($a(c_m c_n | o_{(t,t')} \cup \ o_{(t'+1,t")}) = 1$). This means that no verification is applied after SCR in such infrequent cases.

Recognizing words containing accented characters

In order to recognize a handwritten word containing accented characters, the SCR and bi-character models have to be able to deal with accented characters and accented bi-characters. Similar to the problem mentioned above, considering accented characters would increase the number of character classes and bi-character classes, which would certainly decrease the effectiveness of the system.

In order to deal with this problem, a method was proposed by our team [138]. Given an input signal, this method tries, in the first step, to detect the accent and the main character parts. Then, the accented part and the main character are respectively submitted to an accented recognizer and a SCR. Finally, the recognition results given by these two recognizers are combined to obtain the final recognition result. In the case of bi-character models, a similar method can be used.

5.2.2 Long term perspectives

For the long term perspectives, three main possible directions could be considered in the continuity of our work:

Handwritten line/text recognition

It is possible to use our proposed HWR to recognize handwritten line/text. Indeed, a handwritten line is composed of handwritten words. In order to apply our system in this context, two possible solutions can be used. 1) estimating the segmentation points between handwritten words. Then, submitting each segmented word to the HWR. 2) Adapting the dynamic programming method that we used for word decoding to line decoding, based on a language model.

This handwritten line recognition system could be then used to recognize handwritten text. First, the handwritten text has to be segmented into a set of handwritten lines. Then, the handwritten line recognition system could be applied. As far as we know, there is no line segmentation method for on-line signal in the literature. However,

many methods have been introduced for line segmentation in the context of off-line documents [92, 105], which may provide some initial directions for line segmentation in the context of on-line signal or could be used directly on the reconstructed off-line signal, as a first attempt.

Using Natural Language Processing (NLP)

Once our HWR would be applied for handwritten line/text recognition, high level context (syntactic and semantic context) can be used as external knowledge (see section 1.4), in order to improve the performances of the system.

In the case of handwritten line/text recognition, authors usually use language models in the form of N-gram models as external knowledge to improve the performance of the system. However, NLP has been widely and successfully used in different research problems such as speech recognition, machine translation, text-to-speech, etc. while it is not yet largely applied and discussed in the context of handwritten line/text recognition research problem. Therefore, NLP is another direction that could be considered. In addition, using NLP would also handle the problems related to accented characters explained above.

Adaptation to each writer

Based on the recognition results illustrated in section 4.2.3.6, the recognition rate of our system varies according to the writer. This is due to the large variation of writing produced by different writers. In order to improve the recognition rate, we could create a recognition system adapted to each writer. In this adaptation context, if the writer is not known in advance, two sub-problems have to be solved: writer identification and handwritten word recognition adapted to each writer. Two solutions for these two sub-problems have been proposed and implemented by two Master 2 internship students, realized under our direction. These solutions are presented in the following paragraphs.

Concerning writer identification, our strategy [18, 145] relies on the information at the grapheme or character levels. This strategy ensures a writer identification rate up to 90% when applied in the context of 70 writers on the IRONOFF database (see section 4.1.1).

In order to make our HWR adapted to each writer, we have to create a SCR and bi-character models adapted to each writer. We have already proposed a solution to these problems. This solution is to create training databases personalized to each writer by taking into account not only the samples given by each writer, but also the samples provided by other writers. The idea is to avoid the problems linked to an insufficient number of samples provided by each writer. We experimented this strategy in the context of ICR by using the isolated characters in the UNIPEN and IRONOFF databases (as a standard database) and our own database written by 10 writers. More specifically, the personalized isolated character database for the writer wr_i contains all the training

samples in the standard database (in our case UNIPEN and IRONOFF) and his/her character samples. Then, during the training process, we simply increase the weight of his/her samples and the samples that have the same writing style as his/her writing style. When considering that the writer is correctly identified, this adaptation strategy provides an improvement from 3% to 19% of the recognition rate, depending on the writer.

Due to limitation of time at the end of this thesis, we did not use these two solutions together (writer identification, and ICR adaptation to writer) to create a HWR adapted to each writer. We hope that these two solutions will improve the performance of the HWR.

Appendices

A Experimental results

Table A.1: The confusion matrix provided by the SCR without garbage class (*exp.*2)

Actual class	Predicted class																									
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	443	0	11	5	4	0	0	0	1	0	0	0	0	1	13	0	2	0	6	0	5	0	3	5	0	1
b	2	433	0	2	1	3	0	0	1	3	2	7	0	0	7	3	0	2	15	15	0	0	0	0	3	1
c	6	0	462	1	14	0	0	0	6	0	0	4	0	1	2	1	0	0	0	1	0	0	0	1	1	0
d	2	0	0	493	0	0	0	0	1	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0
e	3	2	29	0	407	1	0	0	11	0	2	26	0	0	11	0	0	1	3	0	0	0	0	4	0	0
f	0	2	3	0	1	442	2	0	0	7	0	7	0	0	1	9	2	6	4	11	0	0	0	1	0	2
g	1	0	0	0	1	8	410	0	0	5	0	0	0	0	1	0	12	0	9	3	0	0	0	3	31	16
h	0	4	0	1	0	0	0	444	0	0	35	1	0	9	0	0	0	0	0	2	1	0	0	2	1	0
i	0	0	10	0	5	0	0	0	354	0	0	35	0	4	0	0	0	41	0	22	8	18	1	1	1	0
j	0	0	0	0	0	0	0	0	0	496	0	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0
k	0	2	0	1	0	1	0	15	0	1	460	1	0	0	0	3	1	0	0	7	1	0	2	5	0	0
l	0	1	1	0	11	0	0	0	26	0	2	419	0	0	0	0	0	2	0	37	0	0	0	0	1	0
m	1	0	0	0	0	0	0	2	0	0	0	0	469	9	0	0	0	0	0	0	14	0	3	1	1	0
n	1	0	1	0	0	0	0	6	3	0	3	0	8	385	0	0	0	34	1	0	45	8	4	0	1	0
o	8	1	13	2	5	0	0	0	6	0	0	0	0	0	442	1	0	0	8	1	3	8	1	0	0	1
p	0	1	0	1	0	8	2	0	1	0	1	1	0	2	0	447	1	5	3	13	0	0	0	1	13	0
q	10	0	1	1	0	6	9	1	0	0	0	3	0	1	2	2	456	0	3	0	0	0	0	0	4	1
r	0	0	2	1	1	0	0	0	39	1	0	6	0	7	1	0	0	423	2	3	0	10	0	0	0	4
s	1	2	1	0	0	2	2	0	2	3	0	0	0	0	1	0	0	4	470	4	0	0	0	1	0	7
t	0	0	2	0	0	0	1	2	27	6	1	39	0	1	0	1	0	5	3	411	0	0	0	0	0	1
u	2	0	0	0	0	0	0	1	22	0	0	0	2	25	0	0	0	3	0	0	394	43	5	2	1	0
v	0	0	0	0	0	0	0	0	17	0	1	0	1	0	4	1	0	4	0	1	3	461	1	1	5	0
w	1	0	0	0	0	0	0	0	3	0	0	0	3	2	0	2	0	0	0	0	2	12	475	0	0	0
x	1	0	2	0	5	0	0	0	1	0	2	1	1	5	0	0	0	1	0	0	0	0	0	477	2	2
y	0	1	0	0	0	0	14	1	0	4	3	1	0	0	0	0	7	4	1	0	0	1	0	6	451	6
z	2	2	0	1	1	1	11	0	1	3	0	1	0	0	3	0	0	3	3	7	0	0	0	4	11	446

Table A.2: The confusion matrix provided by the SCR with garbage class (*exp.*3)

Actual class	Predicted class																												
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	garbage class
a	443	0	11	5	4	0	0	0	1	0	0	0	0	0	13	0	2	1	6	0	5	0	3	5	0	1	0
b	2	432	0	2	1	3	0	0	1	2	1	4	0	0	7	3	0	0	15	14	0	0	0	0	3	1	9
c	6	0	463	1	13	0	0	0	5	0	0	3	0	1	2	1	0	0	0	1	0	0	0	0	1	0	3
d	2	0	0	493	0	0	0	0	1	0	0	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
e	4	1	29	0	406	1	0	0	11	0	1	28	0	0	9	0	0	2	3	0	0	0	0	3	0	0	2
f	0	3	3	0	1	441	2	0	0	6	0	6	0	0	1	10	2	6	3	12	0	0	0	1	0	3	0
g	1	0	0	0	1	8	409	0	0	5	0	0	0	0	1	0	12	0	9	3	0	0	0	2	31	15	3
h	0	4	0	1	0	1	0	444	0	0	32	1	0	9	0	0	0	0	0	3	1	0	0	2	1	0	1
i	0	0	10	0	6	0	0	0	341	0	0	24	1	3	0	0	0	37	0	19	8	18	1	1	1	0	30
j	0	0	0	0	0	0	0	0	0	493	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	5
k	0	2	0	1	0	1	0	14	0	1	460	1	0	0	0	3	1	0	0	4	1	0	2	4	0	0	5
l	0	1	1	0	11	0	0	0	19	0	0	385	0	0	0	0	0	2	0	34	0	0	0	0	1	0	46
m	1	0	0	0	0	0	0	2	0	0	0	0	467	10	0	0	0	0	0	0	15	0	3	1	1	0	0
n	1	0	1	0	0	0	0	6	4	0	3	0	8	379	0	0	0	33	1	0	46	8	5	0	1	0	4
o	9	1	13	2	4	0	0	0	6	0	0	0	0	0	441	1	0	1	7	1	3	8	1	0	0	1	1
p	0	1	0	1	0	7	2	0	0	0	1	1	0	2	0	449	1	5	3	13	0	0	0	0	12	0	2
q	11	0	1	1	0	7	10	1	0	0	0	2	0	0	2	1	455	0	3	0	0	0	0	0	4	1	1
r	0	0	1	1	1	0	0	0	38	1	0	4	0	5	1	0	0	418	2	4	0	10	0	0	0	4	10
s	1	2	1	0	0	2	3	0	1	4	0	0	0	0	1	0	0	5	469	3	0	0	0	0	0	5	3
t	0	0	2	0	0	0	1	2	19	6	1	36	0	1	0	2	0	4	3	407	0	0	0	0	0	1	15
u	2	0	0	0	0	0	0	1	20	0	0	0	2	26	0	0	0	3	0	0	390	46	5	2	1	0	2
v	0	0	0	0	0	0	0	0	16	0	1	0	2	0	3	0	0	4	0	1	3	460	1	1	4	0	4
w	1	0	0	0	0	0	0	0	3	0	0	0	3	1	0	2	0	0	0	0	2	11	476	0	0	0	1
x	1	0	2	0	5	0	0	0	0	0	2	1	1	5	0	0	0	1	0	0	0	0	0	477	2	2	1
y	0	1	0	0	0	0	13	1	0	4	2	0	0	0	0	0	7	0	1	0	0	1	0	4	421	6	39
z	2	2	0	0	1	1	11	0	1	2	0	1	0	0	3	0	0	4	4	8	0	0	0	3	13	444	0
garbage class	0	2	1	0	0	1	0	0	10	13	9	22	0	1	1	0	0	10	4	3	0	0	1	0	8	1	913

Table A.3: Maximum number of graphemes for each character class.

Char.label	Nb.states	Char.label	Nb.states	Char.label	Nb.states
a	5	j	5	s	4
b	6	k	6	t	6
c	3	l	6	u	5
d	6	m	6	v	4
e	4	n	5	w	6
f	7	o	4	x	5
g	7	p	6	y	6
h	6	q	6	z	4
i	4	r	4		

B Baseline HMM-based system: training parameters

As mentioned earlier in section 4.3, the average length (*i.e* number of sliding windows) of each character class ($Ns(c_k)$) have to be estimated. In our baseline HMM-based system, this parameter $Ns(c_k)$ is estimated using the following Equation:

$$Ns(c_k) = \frac{1}{|c_k|} \sum_{w \in word_{c_k}} \frac{L(w)|w_{c_k}|}{|w|} \tag{B.1}$$

where

- $word_{c_k}$: is the set of words in the training database containing the character c_k
- $Ns(c_k)$: average length (*i.e* number of sliding windows) of the character c_k
- $L(w)$: word length (number of sliding windows)
- $|w|$: Word length (number of characters)
- $|w_{c_k}|$: Number of characters c_k in word w
- $|c_k|$: Total number of characters c_k in the dataset training.

Table B.1 illustrates the average length of all the character classes given by our estimated method. The number of states for each character model can be then optimally fixed during the training process basing on the value of $Ns(c_k)$, as explained in section 4.3.

Table B.1: Maximum number of states for each character model.

Char.label	Nb.states	Char.label	Nb.states	Char.label	Nb.states
a	25	j	19	s	22
b	25	k	25	t	21
c	22	l	22	u	23
d	24	m	26	v	21
e	22	n	23	w	24
f	20	o	23	x	23
g	24	p	22	y	24
h	24	q	22	z	24
i	20	r	22		

Bibliography

[1] A.R Ahmad, M. Khalid, and C. Viard-Gaudin. Comparison of Support Vector Machine and Neural Network in Character Level Discriminant Training for Online Word Recognition. In *Proceedings of the Uniten Students Conference on Research and Development*, Malaysia, 2004.

[2] A.R. Ahmad, M.Khalia, C. Viard-Gaudin, and E.Poisson. On-line Handwriting Recognition System using Support Vector Machine. In *Proceedings of the TENCON, IEEE Region 10 International Conference*, pages 311–314, Chiangmai, Thailand, 2004.

[3] A.R. Ahmad, C. Viard-Gaudin, and M. Khalid. Lexicon-Based Word Recognition Using Support Vector Machine and Hidden Markov Model. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 161–165, Washington DC, USA, 2009.

[4] E. Anquetil. *Modélisation et Reconnaissance par la Logique Floue : Application à la Lecture Automatique En-ligne de l'Ecriture Manuscrite Omni-scripteur*. PhD thesis, Université de Rennes I, France, 1997.

[5] C. Bahlmann. Directional Features in Online Handwriting Recognition. *Pattern Recognition*, 39(1):115–125, 2006.

[6] C. Bahlmann, B. Haasdonk, and H. Burkhardt. On-Line Handwriting Recognition with Support Vector Machines : A Kernel Approach. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition*, pages 49–54, Washington, DC, USA, 2002.

[7] Y. Bengio and Y.L. Cun. Word Normalization for On-Line Handwritten Word Recognition. In *Proceedings of the International Conference on Pattern Recognition*, pages 409–413, Copenhagen, Denmark, 1994.

[8] F. Biadsy, J. El-sana, and N. Habash. Online Arabic Handwriting Recognition using Hidden Markov Models. In *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*, pages 85–90, La Baule, France, 2006.

[9] A.L. Bianne-Bernard, C. Kermorvant, and L. Likforman-Sulem. Context-dependent HMM Modeling using Tree-based Clustering for the Recognition of Handwritten Words. In *Document Recognition and Retrieval*, California, USA, 2010.

[10] A.L. Bianne-Bernard, F. Menasri, R.A.H.Mohamad, C.Mokbel, C.Kermorvant, and L. Likforman-Sulem. Dynamic and Contextual Information in HMM Modeling for Handwritten Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(10):2066 –2080, 2011.

[11] G. Boccignone, A. Chianese, L.P. Cordella, and A. Marcelli. Recovering Dynamic Information from Static Handwriting. *Pattern Recognition*, 26(3):409 – 418, 1993.

[12] F. Bortolozzi, A.S. Brito, L.S. Oliveira, and M. Morita. Recent Advances in Handwritten Recognition. *U. Pal, S.K. Parui, B.B. Chaudhuri (Eds.) Document Analysis*, pages 1–30, 2008.

[13] B.E. Boser, I.M. Guyon, and V.N. Vapnik. A Training Algorithm for Optimal Margin Classifiers. In *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, pages 144–152, Pittsburgh, PA, USA, 1992.

[14] A. Brakensiek and G. Rigoll. A Comparison of Character N-Grams and Dictionaries Used for Script Recognition. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 241–245, Seattle, USA, 2001.

[15] A. Brakensiek, J. Rottland, A. Kosmala, and G. Rigoll. Off-line Handwriting Recognition using Various Hybrid Modeling Techniques and Character N-grams. In *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition*, pages 343–352, Amsterdam, Netherlands, 2000.

[16] E.J. Bredensteiner and K.P. Bennett. Multicategory Classification by Support Vector Machines. *Computational Optimizations and Applications*, 12:53–79, 1999.

[17] E. Brill. Transformation-Based Error-Driven Learning and Natural Language Processing: A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21:543–565, 1995.

[18] Q.A. Bui, M. Visani, S. Prum, and J.M. Ogier. Writer Identification Using TF-IDF for Cursive Handwritten Word Recognition. In *Proceedings of the 11th International Conference on Document Analysis and Recognition*, pages 844–848, Beijing, China, 2011.

[19] C.J.C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[20] H. Byun and S.W. Lee. Applications of Support Vector Machines for Pattern Recognition: A Survey. In Lee Seong-Whan and Verri Alessandro, editors, *Pattern Recognition with Support Vector Machines*, Lecture Notes in Computer Science, pages 571–591. Springer Berlin / Heidelberg, 2002.

[21] E. Caillault, C. Viard-gaudin, and A.R. Ahmad. MS-TDNN with Global Discriminant Trainings. In *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pages 856–861, Seoul, South Korea, 2005.

[22] E. Caillaut. *Architecture et Apprentissage d'un Système Hybride Neuro-Markovien pour la Reconnaissance de l'Ecriture Manuscrite En-Ligne*. PhD thesis, Université de Nantes, France, 2005.

[23] J. Cao, M. Ahmadi, and M. Shridhar. Recognition of Handwritten Numerals with Multiple Feature and Multistage Classifier. *Pattern Recognition*, 28:153–160, 1995.

[24] R.G. Casey and E. Lecolinet. A Survey of Methods and Strategies in Character Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.

[25] K.F. Chan and D.Y. Yeung. Recognizing On-line Handwritten Alphanumeric Characters Through Flexible Structural Matching. *Pattern Recognition*, 32(7):1099–1114, 1999.

[26] C.C Chang and C.J Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):1–27, 2011.

[27] K. Chao and D.S. Mandyam. Invariant Character Recognition with Zernike and Orthogonal Fourier-Mellin Moments. *Pattern Recognition*, 35(1):143–154, 2002.

[28] Y.K. Chen and J.F. Wang. Segmentation of Single- or Multiple-Touching Handwritten Numeral String Using Background and Foreground Analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1304–1317, 2000.

[29] C. Chow. On Optimum Recognition Error and Reject Trade-Off. *IEEE Transactions on Information Theory*, 16(1):41–46, 1970.

[30] D. Ciresan. Avoiding Segmentation in Multi-Digit Numeral String Recognition by Combining Single and Two-Digit Classifiers Trained without Negative Examples. In *Proceedings of 10th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 225–230, Timisoara, Romania, 2008.

[31] L. Cole, D. Austin, and L. Cole. Visual Object Recognition using Template Matching. In *Proceedings of Australian Conference on Robotics and Automation*, Canberra, Australia, 2004.

[32] S. Efstathios, N. Fakotakis, and G. Kokkinakis. Automatic Extraction of Rules for Sentence Boundary Disambiguation. In *Proceedings of the Workshop on Machine Learning in Human Language Technology*, pages 82–88, Chania, Greece, 1999.

[33] S. España-Boquera, M.J. Castro-Bleda, J. Gorbe-Moya, and F.Zamora-Martinez. Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):767–779, 2011.

[34] F.J. Ferri, P. Pudil, M. Hatef, and J. Kittler. Comparative Study of Techniques for Large-Scale Feature Selection. *E.S. Gelsema, L.N. Kanal (Eds.), Pattern Recognition in Practice IV, Elsevier Science*, pages 403–413, 1994.

[35] H. Freeman. On the Encoding of Arbitrary Geometric Configurations. *IRE Transactions on Electronic Computers*, EC-10(2):260–268, 1961.

[36] Cinthia O.A. Freitas. Handwritten Isolated Word Recognition: An Approach Based on Mutual Information for Feature Set Validation. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 665–669, Seattle, USA, 2001.

[37] G. Fumera and F. Roli. Analysis of Error-Reject Trade-Off in Linearly Combined Multiple Classifiers. *Pattern Recognition*, 37(6):1245–1265, 2004.

[38] P.D. Gader, M.A. Mohamed, and J.H. Chiang. Handwritten Word Recognition With Character and Inter-character Neural Networks. *IEEE Transactions on Systems, Man, and Cybernetics*, 27(1):158–164, 1997.

[39] S. Garcia-Salicetti, B. Doizzi, P. Gallinari, A. Mellouk, and D. Fanchon. A Hidden Markov Model Extension of a Neural Predictive System for On-line Character Recognition. In *Proceedings of the 3rd International Conference on Document Analysis and Recognition*, pages 50–53, Washington, DC, USA, 1995.

[40] N. Gauthier, T. Artières, P. Gallinari, and B. Dorizzi. Strategies for Combining On-line and Off-line Information in an On-line Handwriting Recognition System. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 412–416, Seattle, WA, USA, 2001.

[41] M. Gönen and E. Alpayd. Multiple Kernel Learning Algorithms. *Journal of Machine Learning Research*, 12:2211–2268, 2011.

[42] V. Govindaraju and R.K. Krishnamurthy. Holistic Handwritten Word Recognition using Temporal Features Derived from Off-line Images. *Pattern Recognition Letters*, 17(5):537–540, 1996.

[43] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. A Novel Connectionist System for Unconstrained Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31:855–868, 2009.

[44] G.Seni, N. Nasrabadi, and S. Rohini. An On-line Cursive Word Recognition System. In *Proceedings of the Computer Vision and Pattern Recognition*, pages 404–410, Seattle, USA, 1994.

[45] W. Guerfali and R. Plamondon. Normalizing and Restoring On-line Handwriting. *Pattern Recognition*, 26(3):419 – 431, 1993.

[46] L. Guichard, A.H. Toselli, and B. Coüasnon. Handwriting Word Verification by SVM-based Hypotheses Re-scoring and Multiple Threshold Rejection. In *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 57–62, Kolkata, India, 2010.

[47] S. Gunter and H. Bunke. Optimizing the Number of States, Training Iterations and Gaussians in an HMM-based Handwritten Word Recognizer. In *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pages 472–476, Scotland, UK, 2003.

[48] S. Günter and H. Bunke. HMM-based Handwritten Word Recognition: on the Optimization of the Number of States, Training Iterations and Gaussian Components. *Pattern Recognition*, 37(10):2069–2079, 2004.

[49] I. Guyon, L. Schomaker, R. Plamondon, M. Liberman, and S. Janet. UNIPEN project of on-line data exchange and recognizer benchmarks. In *Proceedings of the 12th International Conference on Pattern Recognition*, pages 29–33 vol.2, 1994.

[50] M. Hamanaka, K. Yamada, and J. Tsukumo. On-line Japanese Character Recognition Experiments by an Off-line Method Based on Normalization-Cooperated Feature Extraction. In *Proceedings of the 2nd International Conference on Document Analysis and Recognition*, pages 204–207, Tsukuba, Japan, 1993.

[51] A. Hennig and N. Sherkat. Cursive Script Recognition using Wildcards and Multiple Experts. *Pattern Analysis and Applications*, 4(1):51–60, 2001.

[52] H. Hermansky, D.P. Ellis, and S. Sharma. Tandem Connectionist Feature Extraction for Conventional HMM Systems. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pages 1635–1638, 2000.

[53] L. Heutte. *Reconnaissance de Caractères Manuscrits : Application à la Lecture Automatique des Chèques et des Enveloppes Postales.* PhD thesis, Université de Rouen, France, 1994.

[54] L. Heutte, T. Paquet, J.V. Moreau, Y. Lecourtier, and C. Olivier. A Structural/Statistical Feature Based Vector for Handwritten Character Recognition. *Pattern Recognition Letters*, 19(7):629–641, 1998.

[55] L. Heutte, P. Pereira, O. Bougeois, J.V. Moreau, B. Plessis, P. Courtellemont, and Y. Lecourtier. Multi-Bank Check Recognition System: Consideration on the Numeral Amount Recognition Module. *Pattern Recognition and Artificial Intelligence*, 11:595–618, 1997.

[56] H. Hild and A. Waibel. Speaker-Independent Connected Letter Recognition With A Multi-State Time Delay Neural Network. In *Proceedings of the 3rd European Conference on Speech, Communication and Technology*, pages 1481–1484, 1993.

[57] C.J. Hsieh, K.W. Chang, C.J. Lin, S.S. Keerthi, and S. Sundararajan. A Dual Coordinate Descent Method for Large-Scale Linear SVM. In *Proceedings of the 25th International Conference on Machine Learning*, pages 408–415, New York, USA, 2008.

[58] C.W. Hsu, C.C. Chang, and C.J. Lin. A Practical Guide to Support Vector Classification . Technical report, Department of Computer Science, National Taiwan University, 2010.

[59] C.W. Hsu and C.J. Lin. A Comparison of Methods for Multiclass Support Vector Machines. *IEEE Transactions on Neural Networks*, 13(2):415–425, 2002.

[60] J. Hu, S.G. Lim, and M.K. Brown. Writer Independent On-line Handwriting Recognition using an HMM Approach. *Pattern Recognition*, 33(1):133–147, 2000.

[61] M.K. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, 1962.

[62] S. Impedovo, G. Pirloa, and A. Salzo. Automatic Bank Check Processing: A new engineered system. *Pattern Recognition and Artificial Intelligence*, 11(4):5–42, 1997.

[63] M. Iwamura, T. Tsuji, and K. Kise. Memory-based Recognition of Camera-Captured Characters. In *Proceedings of the 9th International Workshop on Document Analysis Systems*, pages 89–96, Boston, MA, USA, 2010.

[64] K.N. Plataniotis J. Lu and A.N. Ventesanopoulos. Face Recognition using Feature Optimization and V-Support Vector Machine. *IEEE Neural Networks for Signal Processing XI*, 2:373–382, 2001.

[65] S. Jaeger, S. Manke, J. Reichert, and A. Waibel. Online handwriting recognition: the NPen++ recognizer. *International Journal on Document Analysis and Recognition*, 3(3):169–180, 2001.

[66] A. Jain and D. Zongker. Feature Selection: Evaluation, Application, and Small Sample Performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153 –158, 1997.

[67] Anil K. Jain, Robert P. W. Duin, and J. Mao. Statistical Pattern Recognition: A review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):4–37, 2000.

[68] Z. Jin, Z. Lou, J. Yang, and Q. Sun. Face Detection using Template Matching and Skin-color Information. *Neurocomputing*, 70(4-6):794–800, 2007.

[69] K.Alahari, S.L Putrevu, and C.V. Jawahar. Learning Mixtures of Offline and Online features for Handwritten Stroke Recognition. In *Proceedings of the 18th International Conference on Pattern Recognition*, pages 379–382, Hong Kong, China, 2006.

[70] M.N. Kapp, C.O.D.A. Freitas, and R. Sabourin. Handwritten Brazilian Month Recognition: An Analysis of Two NN Architectures and a Rejection Mechanism. In *Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition*, pages 209–214, Tokyo, Japan, 2004.

[71] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. An Unconstrained Handwriting Recognition System. *International Journal on Document Analysis and Recognition*, 4:226–242, 2002.

[72] D. Menotti K.C. Otiniano-Rodríguez, G. Cámara-Chávez. Hu and Zernike Moments for Sign Language Recognition. In *Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition*, 2012.

[73] M. Kherallah, L. Haddad, and A.M. Alimi. A new Approach for Online Arabic Handwriting Recognition. In *Proceedings of the 2nd International Conference on Arabic Language Resources and Tools*, Cairo, Egypt, 2009.

[74] A. Khotanzad and Y.H. Hong. Invariant Image Recognition by Zernike Moments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(5):489–497, 1990.

[75] A. Khotanzad and J.H. Lu. Classification of Invariant Image Representations using a Neural Network. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(6):1028–1038, 1990.

[76] J.H. Kim, K.K Kim, and C.Y. Suen. Hybrid schemes of homogeneous and heterogeneous classifiers for cursive word recognition. In *Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition*, pages 433–442, Amsterdam, Netherlands, 2000.

[77] J. Kindermann, E. Leopold, and G. Paass. Multi-class Classification with Error Correcting Codes. Technical report, GMD German National Research Center for Information Technology, 2000.

[78] S. Knerr, V. Asimov, O. Baret, N. Gorsky, and J.C. Simon. The A2iA Intercheque System: Courtesy Amount and Legal Amount Recognition for French Checks. *International Journal of Pattern Recognition and Artificial Intelligence*, 11:505–548, 1997.

[79] G. Koch. *Catégorisation Automatique de Documents Manuscrits : Application aux Courriers Entrants*. PhD thesis, Université de Rouen, France, 2006.

[80] G. Koch, T. Paquet, and L. Heutte. Combination of Contextual Information for Handwritten Word Recognition. In *Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition*, pages 468 – 473, Tokyo, Japan, 2004.

[81] A.L. Koerich, Y. Leydier, R. Sabourin, and C.Y. Suen. A Hybrid Large Vocabulary Handwritten Word Recognition System using Neural Networks with Hidden Markov Models. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition*, pages 99–104, Ontario, Canada, 2002.

[82] A.L. Koerich, R. Sabourin, and C.Y. Suen. Large Vocabulary Off-line Handwriting Recognition: A Survey. *Pattern Analysis and Applications*, 6(2):97–121, 2003.

[83] A. Kosmala, J. Rottland, and G. Rigoll. Improved On-Line Handwriting Recognition Using Context Dependent Hidden Markov Models. In *Proceedings of the 4nd International Conference on Document Analysis and Recognition*, pages 641–644, Ulm, Germany, 1997.

[84] U.H.G. KreBel. Advances in kernel methods. chapter Pairwise Classification and Support Vector Machines, pages 255–268. MIT Press, Cambridge, MA, USA, 1999.

[85] P.M. Lallican, C. Viard-gaudin, and S. Knerr. From Off-Line To On-Line Handwriting Recognition. In *In Proceedings of the 7th International Workshop on Frontiers in Handwriting Recognition*, pages 303–312, Amsterdam, 2000.

[86] C.J. Lin, R.C. Weng, and S.S. Keerthi. Trust Region Newton Methods for Large-scale Logistic Regression. In *Proceedings of the 24th International Conference on Machine Learning*, pages 561–568, Oregon, USA, 2007.

[87] C.L Liu, H. Sako, and H. Fujisawa. Effects of Classifier Structures and Training Regimes on Integrated Segmentation and Recognition of Handwritten Numeral Strings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1395–1407, 2004.

[88] M. Liwicki and H. Bunke. HMM-Based On-Line Recognition of Handwritten Whiteboard Notes. In *Proceedings of the 10th International Workshop Frontiers in Handwriting Recognition*, pages 595–599, La Baule, France, 2006.

[89] M. Liwicki and H. Bunke. Combining On-Line and Off-Line Systems for Handwriting Recognition. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, pages 372–376, Curitiba, Paraná, Brazil, 2007.

[90] M. Liwicki, H. Bunke, and Neubrckstrasse. Writer-Dependent Recognition of Handwritten Whiteboard Notes in Smart meeting Room Enviroments. In *Proceedings of the 8th International Workshop on Document Analysis Systems*, Nara, Japan, 2008.

[91] M. Liwicki, A. Graves, H. Bunke, and J. Schmidhuber. A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, pages 367–371, 2007.

[92] G. Louloudis, B. Gatos, I. Pratikakis, and C. Halatsis. Text Line and Word Segmentation of Handwritten Documents. *Pattern Recognition*, 42(12):3169–3183, 2009.

[93] S. Madhvanath and V. Govindaraju. The Role of Holistic Paradigms in Handwritten Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(2):149–164, 2001.

[94] S. Madhvanath, G. Kim, and V. Govindaraju. Chaincode Contour Processing for Handwritten Word Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(9):928–932, 1999.

[95] U.V. Marti and H. Bunke. Text Line Segmentation and Word Recognition in a System for General Writer Independent Handwriting Recognition. *Proceedings of the International Conference on Document Analysis and Recognition*, 0:159–163, 2001.

[96] U.V. Marti and H. Bunke. Hidden Markov Models. chapter Using a Statistical Language Model to Improve the Performance of an HMM-based Cursive Handwriting Recognition Systems, pages 65–90. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2002.

[97] S. Marukatat, T. Artieres, R. Gallinari, and B. Dorizzi. Sentence Recognition Through Hybrid Neuro-Markovian Modeling. In *Proceedings of the 6th International Conference on Document Analysis and Recognition*, pages 731 –735, 2001.

[98] E. Mayoraz and E. Alpaydin. Support Vector Machine for Multiclass Classification. In *Proceedings of International Workshop on Artificial Neural Networks*, Alicante, Spain, 1999.

[99] M. Mehri, P. Gomez-Krämer, P. Héroux, and R. Mullot. Old Document Image Segmentation using the Autocorrelation Function and Multiresolution Analysis. In *Proceedings of the 20th Document Recognition and Retrieval*, 2013.

[100] P. Mermelstein and M. Eyden. A System for Automatic Recognition of Handwritten Words. In *Proceedings of the Fall Joint Computer Conference, Part I*, pages 333–342, New York, USA, 1964.

[101] R.A Wagner M.J and Fischer. The String-to-String Correction Problem. *Journal of ACM*, 21(1):168–173, 1974.

[102] V. Nguyen and M. Blumenstein. Techniques for Static Handwriting Trajectory Recovery: A Survey. In *Proceedings of the 9th International Workshop on Document Analysis Systems*, pages 463–470, Boston, MA, USA, 2010.

[103] H. Nishida. An Approach to Integration of Off-line and On-line Recognition of Handwriting. *Pattern Recognition Letters*, 16(11):1213–1219, 1995.

[104] U. Pal, A. Belaïd, and C. Choisy. Touching Numeral Segmentation using Water Reservoir Concept. *Pattern Recognition Letters*, 24(1-3):261–272, 2003.

[105] V. Papavassiliou, T. Stafylakis, V. Katsouros, and G. Carayannis. Handwritten Document Image Segmentation into text Lines and Words. *Pattern Recognition*, 43(1):369–377, 2010.

[106] F. Perraud, C. Viard-Gaudin, E. Morin, and P.M. Lallican. N-gram and N-class models for on line handwriting recognition. In *Proceedings of the 7th International Conference on Document Analysis and Recognition*, pages 1053–1057, Scottland, UK, 2003.

[107] R. Plamondon and S.N. Srihari. On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[108] J.C. Platt, N. Cristianini, and J. Shawe-taylor. Advances in Neural Information Processing Systems. chapter Large Margin DAGs for Multiclass Classification, pages 547–553. MIT Press, 2000.

[109] T. Plötz and G.A. Fink. Markov Models for Offline Handwriting Recognition: A Survey. *International Journal on Document Analysis and Recognition*, 12(4):269–298, 2009.

[110] T. Plötz, C. Thurau, and G.A. Fink. Camera-based Whiteboard Reading: New Approaches to a Challenging Task. In *Proceedings of the 11th International Conference on Frontiers in Handwriting Recognition*, pages 385–390, Montreal Canada, 2008.

[111] S. Prum, M. Visani, and J.M. Ogier. Cursive On-line Handwriting Word Recognition Using a Bi-character Model for Large Lexicon Applications. In *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 194–199, Kolkata, India, 2010.

[112] S. Prum, M. Visani, and J.M. Ogier. On-Line Handwriting Word Recognition Using a Bi-character Model. In *Proceedings of the 20th International Conference on Pattern Recognition*, pages 2700–2703, Istanbul, Turkey, 2010.

[113] S. Prum, M. Visani, and J.M. Ogier. A Discriminative Approach to On-Line Handwriting Recognition Using Bi-Character models. In *Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 364–368, Washington, DC, USA, 2013.

[114] P. Pudil, J. NovoviAovA, and J. Kittler. Floating search methods in feature selection. *Pattern Recognition Letters*, 15(11):1119–1125, 1994.

[115] H.A. Qader, A.R. Ramli, and S. Al-Haddad. Fingerprint Recognition Using Zernike Moments. *International Arab Journal of Information Technology*, 4(4):372–376, 2007.

[116] S. Quiniou, E. Anquetil, and S. Carbonnel. Statistical Language Models for On-line Handwritten Sentence Recognition. In *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pages 516–520, Seoul, South Korea, 2005.

[117] M.I Razzak, S.A. Hussain, M. Sher, and Z. S.Khan. Combining Off-line and Online Preprocessing for Online Urdu Character Recognition. In *Proceedings of the International MultiConference of Engineers and Computer Scientists*, Hong Kong, 2009.

[118] A. Rehman and T. Saba. Off-line Cursive Script Recognition: Current Advances, Comparisons and Remaining problems. *Artificial Intelligence Review*, 37(4):261–288, 2012.

[119] J. Rottland and G. Rigoll. Tied posteriors: an Approach for Effective Introduction of Context Dependency in Hybrid NN/HMM. In *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1241–1244 vol.3, 2000.

[120] M. Rusinol, D. Aldavert, R. Toledo, and J. Llados. Browsing Heterogeneous Document Collections by a Segmentation-Free Word Spotting Method. In *Proceedings of the 11th International Conference on Document Analysis and Recognition*, pages 63–67, Beijing, China, 2011.

[121] B. Mouldi S. Ouchtati and A. Lachouri. Segmentation and Recognition of Handwritten Numeric Chains. *Journal of Computer Science*, 4(3):242–248, 2007.

[122] J. Sadri, C.Y. Suen, and T.D. Bui. Application of Support Vector Machines for Recognition of Handwritten Arabic/Persian Digits. In *Proceedings of the 2nd Conference on Machine Vision and Image Processing*, pages 300–307, 2003.

[123] K. C. Santosh, Bart Lamiroy, and Laurent Wendling. DTW for Matching Radon Features: a Pattern Recognition and Retrieval Method. In *Proceedings of the 13th international conference on Advanced concepts for intelligent vision systems*, ACIVS'11, pages 249–260, Berlin, Heidelberg, 2011. Springer-Verlag.

[124] J. Schenk and G. Rigoll. Novel Hybrid NN/HMM Modelling Techniques for On-line Handwriting Recognition. In Guy Lorette, editor, *Proceedings of the 10th International Workshop on Frontiers in Handwriting Recognition*, La Baule, France, 2006. Université de Rennes 1.

[125] M. Schenkel, I. Guyon, and D. Henderson. On-line cursive script recognition using time delay neural networks and Hidden Markov Models. In *Proceedings of the*

International Conference on Acoustics, Speech, and Signal Processing, volume 2, pages 637–640, 1994.

[126] M. Schenkel, H. Weissman, I. Guyon, C. Nohl, and D. Henderson. Recognition-based Segmentation of On-line Hand-printed Words. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 723–730, 1993.

[127] L. Schomaker. From Handwriting Analysis to Pen-Computer Applications. *IEE Electronics Communication Engineering Journal*, 10:93–102, 1998.

[128] A. Sharma. *Online Handwritten Gurmukhi Character Recognition*. PhD thesis, Mathematics and Computer Applications Thapar University, India, 2009.

[129] Z. Shi, S.N. Srihari, Y.C. Shiu, and V. Ramanaprasad. A System for Segmentation and Recognition of Totally Unconstrained Handwritten Numeral Strings. In *Proceedings of the 4nd International Conference on Document Analysis and Recognition*, pages 455–458, Ulm, Germany, 1997.

[130] S.Tabbone and L. Wendling. Technical symbols recognition using the two-dimensional radon transform. In *Proceedings of the 16th International Conference on Pattern Recognition*, pages 200–203 vol.3, 2002.

[131] T. Starner, J. Makhoul, R. Schwartz, and G. Chou. On-line Cursive Handwriting Recognition using Speech Recognition Methods. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume v, pages 125–128, 1994.

[132] S. Tabbone, O.R. Terrades, and S. Barrat. Histogram of radon transform. A useful Descriptor For Shape Retrieval. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4, Tampa, FL, USA, 2008.

[133] S. Tabbone, L. Wendling, and J.P. Salmon. A New Shape Descriptor Defined on the Radon Transform. *Computer Vision and Image Understanding*, 102(1):42–51, 2006.

[134] C.C. Tappert, C.Y. Suen, and T. Wakahara. State of the Art in On-Line Handwriting Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[135] M.R. Teague. Image Analysis via the General Theory of Moments. *Journal of the Optical Society of America*, 70(8):920–930, 1980.

[136] O.R. Terrades, S. Tabbone, and E. Valveny. A Review of Shape Descriptors for Document Analysis. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, pages 227–231, Curitiba, Brazil, 2007.

[137] N.C. Tewari and A.M. Namboodiri. Learning and Adaptation for Improving Handwritten Character Recognizers. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 86–90, Barcelona, Spain, 2009.

[138] D.C. Tran, P. Franco, and J.M. Ogier. Accented Handwritten Character Recognition Using SVM - Application to French. In *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 65–71, Kolkata, India, 2010.

[139] D. Trier, A.K. Jain, and T. Taxt. Feature Extraction Methods for Character Recognition - A Survey. *Pattern Recognition*, 29(4):641–662, 1996.

[140] O.D. Trier, A.K. Jain, and T. Taxt. Feature extraction methods for character recognition : A Survey. *Pattern Recognition*, 29(4):641–662, 1996.

[141] S. Uchida and M. Liwicki. Analysis of Local Features for Handwritten Character Recognition. In *Proceedings of the 20th International Conference on Pattern Recognition*, pages 1945–1948, Istanbul, Turkey, 2010.

[142] C. Viard-Gaudin, P.M Lallican, P. Binter, and S. Knerr. The IRESTE On/Off (IRONOFF) Dual Handwriting Database. In *Proceedings of the 5th International Conference on Document Analysis and Recognition*, pages 455–458, Washington DC, USA, 1999.

[143] A. Vinciarelli, S. Bengio, and H. Bunke. Offline Recognition of Unconstrained Handwritten Texts Using HMMs and Statistical Language Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):709–720, 2004.

[144] P. Viola and M. Jones. Robust Real-time Object Detection. Technical report, Cambridge Research Laboratory, 2001.

[145] M. Visani, Q.A. Bui, and S. Prum. On-line Cursive Handwriting Characterization using TF-IDF Scores of Graphemes. In *10th International Workshop on Document Analysis Systems*, Queensland, Australia, 2012.

[146] A.J. Viterbi. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[147] E. Vural, H. Erdogan, K. Oflazer, and B. Yanikoglu. An Online Handwriting Recognition System for Turkish. In *Proceedings of the IEEE 12th Signal Processing and Communications Applications Conference*, pages 607 – 610, 2004.

[148] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K.J. Lang. Phoneme recognition using Time-Delay Neural Networks. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 37(3):328–339, 1989.

[149] X. Wang, V. Govindaraju, and S. Srihari. Holistic Recognition of Handwritten Character Pairs. *Pattern Recognition*, 33(12):1967 – 1973, 2000.

[150] J. WESTON and C. Watkins. Multi-class Support Vector Machines. Technical report, CSD-TR-98-04, Department of Computer Science, Egham, Surrey, England, 1998.

[151] M. Wienecke, G.A. Fink, and G. Sagerer. Experiments in Unconstrained Off-line Handwritten Text Recognition. In *Proceedings of the 8th International Workshop on Frontiers in Handwriting Recognition*, Ontario, Canada, 2002.

[152] M. Woźniak, M. Graña, and E. Corchado. A Survey of Multiple Classifier Systems as Hybrid Systems. *to appear in Information Fusion*, 2013.

[153] D. Wu, C.Y. Suen, and A. Krzyzak. A New Courtesy Amount Recognition Module of a Check Reading System. In *Proceedings of the 19th International Conference on Pattern Recognition*, pages 1–4, USA, 2008.

[154] T.F. Wu, C.J. Lin, and R.C. Weng. Probability Estimates for Multi-class Classification by Pairwise Coupling. *Journal of Machine Learning Research*, 5:975–1005, 2004.

[155] X.C. Yin, H.W. Hao, Y.F Tang, J. Sun, and S. Naoi. Rejection Strategies with Multiple Classifiers for Handwritten Character Recognition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 1126–1130, Barcelona, Spain, 2009.

[156] L. Yun, CS. Liu, X.Q. Ding, and F. Qiang. A Recognition Based System for Segmentation of Touching Handwritten Numeral Strings. In *Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition*, pages 294 – 299, 2004.

List of Publications

International Conference Papers

[1] **S. Prum**, M. Visani, A. Fischer ,and J.M. Ogier. A Discriminative Approach to On-Line Handwriting Recognition Using Bi-Character models. *In Proceedings of the 12th International Conference on Document Analysis and Recognition*, pages 364-368, Washington, DC, USA, 2013.

[2] Q.A. Bui, M. Visani, **S. Prum**, and J.M. Ogier. Writer Identification Using TF-IDF for Cursive Handwritten Word Recognition. *In Proceedings of the 11th International Conference on Document Analysis and Recognition*, pages 844-848, Beijing, China, 2011.

[3] **S. Prum**, M. Visani, and J.M. Ogier. Cursive On-line Handwriting Word Recognition Using a Bi-character Model for Large Lexicon Applications. *In Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 194-199, Kolkata, India, 2010.

[4] **S. Prum**, M. Visani, and J.M. Ogier. On-Line Handwriting Word Recognition Using a Bi-character Model. *In Proceedings of the 20th International Conference on Pattern Recognition*, pages 2700-2703, Istanbul, Turkey, 2010.

International Workshop Paper

[1] M. Visani, Q.A. Bui, and **S. Prum**. On-line Cursive Handwriting Characterization using TF-IDF Scores of Graphemes. *In Proceedings of the 10th International Workshop on Document Analysis Systems*, Queensland, Australia, 2012.

THÈSE présentée par **Sophea PRUM**

Titre : Vers une approche discriminante pour la reconnaissance de mots manuscrits en-ligne utilisant des modèles de bi-caractères

Résumé : Avec l'avènement des dispositifs nomades tels que les smartphones et les tablettes, la reconnaissance automatique de l'écriture manuscrite cursive à partir d'un signal en ligne est devenue durant les dernières décennies un besoin réel de la vie quotidienne à l'ère numérique. Dans le cadre de cette thèse, nous proposons de nouvelles stratégies pour un système de reconnaissance de mots manuscrits en-ligne. Ce système se base sur une méthode collaborative segmentation/reconnaissance et en utilisant des analyses à deux niveaux: caractère et bi-caractères. Plus précisément, notre système repose sur une segmentation de mots manuscrits en graphèmes afin de créer un treillis à L niveaux. Chaque nœud de ce treillis est considéré comme un caractère potentiel envoyé à un moteur de Reconnaissance de Caractères Isolés (RCI) basé sur un SVM. Pour chaque nœud, ce dernier renvoie une liste de caractères associés à une liste d'estimations de probabilités de reconnaissance. Du fait de la grande diversité des informations résultant de la segmentation en graphèmes en particulier à cause de la présence de morceaux de caractères et de ligatures, l'injection de chacun des nœuds du treillis dans le RCI engendre de potentielles ambiguïtés au niveau du caractère. Nous proposons de lever ces ambiguïtés en utilisant des modèles de bi-caractères, basés sur une régression logistique et dont l'objectif est de vérifier la cohérence des informations à un niveau de reconnaissance plus élevé. Finalement, les résultats renvoyés par le RCI et l'analyse des modèles de bi-caractères sont utilisés dans la phase de décodage pour parcourir le treillis dans le but de trouver le chemin optimal associé à chaque mot dans le lexique. Deux méthodes de décodage sont proposées (recherche heuristique et programmation dynamique), la plus efficace étant basée sur de la programmation dynamique.

Mots clés : reconnaissance de mots manuscrits en-ligne; modèle de bi-caractères; programmation dynamique; Séparateurs à Vaste Marge; combinaison de caractéristiques en-ligne et hors-ligne;

Abstract : With the advent of mobile devices such as tablets and smartphones over the last decades, on-line handwriting recognition has become a very highly demanded service for daily life activities and professional applications. This thesis presents a new approach for on-line handwriting recognition. This approach is based on explicit segmentation/recognition integrated in a two level analysis system: character and bi-character. More specifically, our system segments a handwritten word in a sequence of graphemes to be then used to create a L-levels lattice of graphemes. Each node of the lattice is considered as a character to be submitted to a SVM based Isolated Character Recognizer (ICR). The ICR returns a list of potential character candidates, each of which is associated with an estimated recognition probability. However, each node of the lattice is a combination of various segmented graphemes. As a consequence, a node may contain some ambiguous information that cannot be handled by the ICR at character level analysis. We propose to solve this problem using "bi-character" models based on Logistic Regression, in order to verify the consistency of the information at a higher level of analysis. Finally, the recognition results provided by the ICR and the bi-character models are used in the word decoding stage, whose role is to find the optimal path in the lattice associated to each word in the lexicon. Two methods are presented for word decoding (heuristic search and dynamic programming), and dynamic programming is found to be the most effective.

Keywords : On-line Handwriting Recognition; Bi-Character models; Dynamic Programming; Support Vector Machine; Combining On-line and Off-line Features