



**A Framework Model for  
Arabic Handwritten Text Recognition to Handle Missing  
Text Fragments**

الإطار العملي لنموذج تمييز نصوص العربي بخط اليد للتعامل مع النص المقطوع

**By**

**Baha Abd Al-wahhab Al-Garalleh**

**(400910229)**

**Supervisor**

**Dr. Hussein Hadi Owaied**

**A thesis Submitted in partial Fulfillment  
of the requirements of the Master Degree in Computer Science**

**Faculty of Information Technology**

**Middle East University**

**Amman – Jordan  
June 2013**

**Authorization Statement**

I, Baha Abd-Alwahhab mohssen Al-Garalleh, authorize Middle East University to supply hard and electronic copies of my thesis to libraries, establishments, bodies, and institutions concerned with research and scientific studies upon request, according to university regulations.

Name: Baha Abd-Alwahhab mohssen Al-Garalleh

Date: 10-6-2013

Signature: 

## إقرار تفويض

أنا بهاء عبدالوهاب محسن القرالة الموهن جامعة الشرق الأوسط بتزويد نسخ من رسائلي للمكاتب أو المؤسسات أو الهيئات المعنية بالابحاث و الدراسات العلمية أو الأفراد عند طلبها.

الإسم: بهاء عبدالوهاب محسن القرالة

التاريخ: ١٠/٦/٢٠١٤

التوقيع: 

### Examination Committee Decision

This is to certify that the thesis entitled "A Framework Model for Arabic Handwritten Text Recognition to Handle Missing Text Fragments" was successfully defended and approved on June 10<sup>th</sup> 2013.


#### Examination Committee Members

#### signature

Dr. Hussein Hadi Owaied

Department of Computer Science

Middle East University



Dr. Mudhafar AL-jarrah

Department of Computer Science

Middle East University



Prof. Nael Hirzallah

Faculty of Information Technology

Applied Science University



## **DEDICATION**

This thesis is dedicated to my father's soul, who I was hoping he witness this moment of my life, thanks to him after Allah for what I am today, Allah bless his pure soul. To my mother who stayed up nights and endured so much for this moment, Allah kept her a crown over my head, and to my dear brothers and sisters.

## **ACKNOWLEDGEMENTS**

At first I would like to express my sincere gratitude to my supervisor Dr. Hussein Hadi Owaied for the valuable guidance and advice. He inspired me greatly to work in this thesis, and he contributed a lot in raising my knowledge in all fields and especially the research field. I acknowledge Examination Committee Members. I thank my friends and dear family, who encourage me all the way. I thank my friend Eng. anas soub to his effort of programming the system.

## **Abstract**

In this thesis a framework model for Arabic handwritten text recognition is developed to deal with the missing text fragments. Since there are many Arabic manuscripts ancient of ancient times have been either destroyed or stolen, whether all or some parts or pages, so it's important to complete the text.

The framework model consists of two stages; the first stage consists of four parts which are Input 1 scan text image, recognition stage, extracted feature, and Input 2 Typed. The second stage is the decision stage according to the availability of all Arabic characters set forms. So if all the Arabic characters set are available in the original text the missing fragments will be discovered from the text, otherwise handwritten missing characters are estimated then embedded the fragment in the original document.

The original handwritten text with missing characters, after extracting the handwritten characters' features, then the missing texts will be typed and converted to handwritten text in the same form of the original text.

In this thesis the handwritten text sample of writers was so small, 17 writers, so the recognition process needs to be much efficient and accurate to recognize the characters written by many different users more than the sample used.

The developed framework model can be used in the library, and governmental offices for discovering the missing text in the handwritten documents. The programming language used in the implementation was C#.Net.

## الملخص

في هذه الاطروحة تم تطوير نموذج اطار عملي للتعرف على الكتابة العربية بخط اليد للتعامل مع النصوص المفقودة , و ذلك بسبب وجود الكثير من المخطوطات العربية من العصور القديمة إما انها دمرت او سرقت سواء كانت كلها او اجزاء منها.

النموذج يحتوي على مرحلتين، المرحلة الاولى تتكون من اربعة اجزاء و هي ؛ ادخال الصورة للنص الاصيل، مرحلة التعرف، استخراج الخصائص، ادخال النص المفقود، و المرحلة الثانية هي مرحلة اتخاذ القرار وفقا لتوفر جميع اشكال الحروف العربية، اذا كانت جميع اشكال الحروف موجودة في النص الاصيل فانه يتم تجميع الكلمة المفقودة و دمجها مع النص الاصيل، و اذا لم تكن اشكال الحروف متوفرة في النص الاصيل فانه يتم البحث عن حروف مشابهه في الخصائص لدى كاتب اخر ليتم دمج النص المفقود مع النص الاصيل، حيث يتم ادخال النص المفقود بشكل مطبوع و يتم تحويله الى نص مكتوب بخط اليد بنفس شكل النص الاصيل حسب الحالتين المذكورتين.

في هذه الاطروحة استخدمت عينة صغيرة من الكُتاب، سبعة عشر كاتباً، لذلك فإن عملية التعرف تحتاج لتكون اكثر كفاءة و دقة للتعرف على الحروف المكتوبة بواسطة اكثر من كاتب مختلفين اكثر من العينة المستخدمة.

النموذج الذي تم تطويره يمكن ان يستخدم في المكتبات و الدوائر الحكومية التي تعتمد على الكتابات اليدوية للكشف عن النصوص المفقودة في الوثائق المكتوبة بخط اليد . تم استخدام لغة البرمجة C# لتطبيق هذا النموذج . تم استخدام لغة البرمجة C#.Net لتطبيق هذا النموذج .



## Content

Cover Page .....	I
Authorization Statement .....	II
إقرار تفويض .....	III
Examination committee decision .....	IV
DEDICATION .....	V
ACKNOWLEDGEMENTS .....	VI
Abstract .....	VII
الملخص .....	VIII
Content .....	IX
List of Tables .....	XIII
List of Figures .....	XIV
List of Abbreviations .....	XVI
<b>Chapter ( 1 ) : Introduction.....</b>	<b>1</b>
1.1    Introduction.....	1
1.2    Problem Definition.....	2
1.3    Objectives .....	3
1.4    Problem Significance and Motivation .....	4
1.5    Problem Solution Approach.....	4
1.6    Limitations .....	4
1.7    Goals .....	4
1.8    Thesis Overview .....	5

<b>Chapter ( 2 ) : Termenology</b> .....	6
2.1 Introduction .....	6
2.2 Pattern Recognition .....	6
2.3 Optical Character Recognition (OCR) .....	6
2.4 Handwriting Recognition Categories .....	7
2.4.1 Online Handwriting Recognition.....	8
2.4.2 Offline Handwriting Recognition .....	9
2.5 Application of Offline Handwritten Recognition .....	9
2.6 Arabic language .....	10
<b>Chapter ( 3 ) : Literature Survey</b> .....	16
3.1 Overview .....	16
3.2 Fractal & Multi-Fractal for Arabic Offline Writer Identification.....	16
3.3 Hough Transform Technique.....	17
3.4 Handwritten Character Segmentation using Baseline Approach.....	18
3.5 Recognition of Handwritten Arabic text Using Neural Network .....	18
3.6 Related work .....	19
3.6.1 Arabic Handwriting Recognition System Using Genetic Approach .....	19
3.6.2 Using binary Representation of Character for Feature Extraction.....	19
3.6.3 Arabic Handwritten Word Recognition Using Dots Concepts .....	20
3.6.4 Off-line Arabic Handwriting Recognition Based on Projection Profile....	21
3.6.5 Using Hybrid Hidden Markov (HMM) and (ANN) .....	21
3.6.6 Fuzzy Logic approach to Recognition of Isolated Arabic Characters .....	22

<b>Chapter ( 4 ) : The Structure of Framework .....</b>	<b>23</b>
4.1 Introduction .....	23
4.2 The Model Structure .....	23
4.2.1 Preprocessing .....	25
4.2.1.1 Noise Removal Process .....	26
4.2.1.2 Resize the Image .....	26
4.2.1.3 Binarization.....	27
4.2.1.4 Convert to Grayscale.....	29
4.2.1.4 Segmentation.....	29
4.2.2 Features Extraction .....	34
4.2.2.1 Main Body Features.....	35
4.2.2.2 Secondary Component.....	35
4.2.3 Classification .....	38
4.2.4 Recognition .....	40
4.2.5 If All Form Available .....	41
4.2.6 If there is Any Form not Available.....	41
 <b>Chapter ( 5 ) : Implementation and Results.....</b>	 <b>42</b>
5.1 Introduction .....	42
5.2 Data Collection .....	42
5.3 Implementation .....	43
5.3.1 Load Image .....	44
5.3.2 Preprocessing .....	45

5.3.3 Discussion result .....	54
5.3.3.1 The fragments found in writer database.....	55
5.3.3.2 The fragments not found in writer database .....	56
<b>Chapter ( 6 ) : Conclusion, Dicussion and Future Works .....</b>	<b>58</b>
6.1 Conclusion .....	58
6.2 Discussion .....	59
6.3 Future Work.....	59
<b>Appendix.....</b>	<b>60</b>
1. Test data sample .....	60
2. Programming Code .....	77
<b>References.....</b>	<b>107</b>

## List of Tables

Table 2.1: Example of Arabic words with sub words or PAWs .....	12
Table 2.2: Arabic letters forms .....	15
Table 4.1: Types of the secondary components.....	36
Table 4.2: Possibilities of the secondary components position .....	37
Table 4.3: Example of Features for Lines, Curves, Loops for Arabic Characters.....	38
Table 5.1: Family of characters in basis of writing style.....	50

## List of Figures

Figure 2.1 Categories of character recognition.....	7
Figure 2.2: On-line Handwriting Inputs.....	8
Figure 2.3: Off-line Handwriting Inputs.....	9
Figure 2.4: Meaning of PAW.....	11
Figure 2.5: Example of word in Arabic difficult on segmentation .....	12
Figure 2.6: Typewritten example.....	13
Figure 2.7: Typeset example.....	13
Figure 2.8: Handwritten example .....	13
Figure 2.9: Example of letters connection .....	14
Figure 4.1: A block diagram for the model.....	24
Figure 4.2: Preprocessing stage .....	25
Figure 4.3: The Arabic letter "Noon" Binarization.....	28
Figure 4.4: Segmentation process .....	30
Figure 4.5: Segmentation of printed and handwritten text .....	30
Figure 4.6: Connected pixel.....	32
Figure 4.7: Arabic letter "jeem" .....	36
Figure 4.8: Primitive features .....	38
Figure 5.1: Example of Collected Data.....	43
Figure 5.2: Load image .....	44
Figure 5.3: After loading image, the image appear in the preview area.....	44
Figure 5.4: Resize image Button.....	45
Figure 5.5: Convert to gray button.....	45
Figure 5.6: Input image in gray scale.....	45
Figure 5.7: Gray scale image after digitizing process.....	46
Figure 5.8: Connected Components Labeling Process .....	46
Figure 5.9: Segmented handwritten text .....	47
Figure 5.10: The PAWs Form.....	47
Figure 5.11: Text PAWs process of Arabic statement.....	48
Figure 5.12: Example of PAW consist one character .....	48

Figure 5.13: Example of PAW consist more than one character .....	49
Figure 5.14: Typed and handwritten text prepared to associate .....	51
Figure 5.15: Association process of PAWs .....	51
Figure 5.16: Horizontal and vertical histogram of PAWs .....	52
Figure 5.17: Detect Baseline of Arabic word .....	52
Figure 5.18: Segmentation process .....	53
Figure 5.19: Segmented characters .....	53
Figure 5.20: Calculate and store extracted Features .....	54
Figure 5.21: Test Form .....	55
Figure 5.22: Fragments in same writer style.....	55
Figure 5.23: Search for the fragment word "جامعة" .....	56
Figure 5.24: Fragment result .....	56
Figure 5.25: Fragments Search .....	57
Figure 5.26: Characters form available in writer database .....	57
Figure 5.27: Characters form available in other writers' database.....	57

## List of Abbreviations

ANN	Artificial Neural Networks
NN	Neural Networks
CR	Character Recognition
OCR	Optical Character Recognition
HCR	Handwritten Character Recognition
PAW	Part of Arabic Word
CBSA	Clustering-Based Skeletonization Algorithm
HT	Hough Transforms
MSE	Mean Squared Error
HMM	Hidden Markov Model
UI	User Interface



# Chapter One

## Introduction

### 1.1 Introduction

The main goal of handwriting recognition system is to convert handwritten text documents from digital image format to coded characters format documents in order to be clearly readable and editable using word processing application systems, and represents an attempt to simulate the human reading process. (Hussain F. and Cowell J., 2000).

Handwriting recognition is becoming more important, because it greatly helps in completing office tasks in an easy way and it solved many problems, which leads to reduce the time and reduce the effort to complete these tasks. (Miguel, Po-hsien Wu, 2003).

The handwriting has always been a problem, in the past years a lot of work has been done in the handwriting area, it was clear the use of Artificial Neural Networks (ANN) to solve this problem and also use genetic algorithms but not as much as ANN (Soryani, M. and Rafat, N., 2006).

Comparison between writing in the English language and writing in Arabic noted that the Arabic language is more complex than the English language; Arabic letters have many forms, there is no uppercase and lowercase, but the letters have more than one shape, such as the letter form in the beginning of a word, in the middle, in the end, and isolated shape. Also there is many kinds handwriting, an accurate feature should extract and wide sample should be used for training to have an accurate result. (Zaidan A.A, et al, 2010).

(Abandah G.A. and Khedher M.Z., 2005) said that "Arabic language has many difficulties such as unlimited variation in human writing, similarities of distinct character shapes, character overlaps and interconnections of neighboring characters".

After more than two decades of intensive effort to solving the problems of handwriting recognition, progress in recent years has been very promising and still of great interest and a wide range of research (Bunke M.H, et al, 2009).

The studies in the handwriting recognition field have focused on the issue of conversion from handwritten text to typed text because of the importance of handwriting recognition and the clarity of typed text documents.

The real benefit of this study is in the fields of art and literature, and the ancient manuscripts or any text that lost some of the texts or some of the pages and the desire to complete the text of manuscripts with texts written by the hand of the original author, and to preserve the Arabic script, which is a feature in Arabic language.

## **1.2 Problem Definition**

The basis of the problem is to converting an image with lost words or characters into text and handle the fragment to be included to the text, many people want all texts in the printed form and moving away from handwritten texts, due to the readability of printed text and easy to modify it and the difficulty to read handwritten texts and the lack of clarity in many times, the people have forgotten that the value of many of the writings are the existence in handwritten form, such as old manuscripts.

There are many handwritten texts must remain in handwritten form because of its archeological or religious value, this will need application able to build a new handwritten text from a typed text relaying on original handwritten text.

There are many problems related to the Arabic character recognition, such as the difficulty of writing Arabic characters, the way these characters connect to each other, the different fonts and styles of writing in Arabic, and also that the character in the Arabic language has more than one form of writing according to its position in the word.

Because of these difficulties the following problems have been identified:

1. In the preprocessing stage identify the style of handwritten text and its features.
2. The problems of recognition of character style which are not appearing in the original document.
3. The difficulty of writing Arabic characters and the way these characters connect to each other and the different font and style of writing in Arabic.

### **1.3 Objectives**

The following are the objectives of this study:

1. Selection of appropriate algorithms for processes in the preprocessing stage and other stages in the framework to reach a high percentage of character recognition.
2. Rebuild the damaged manuscripts or any text in whole or in part, such as the loss of characters or words.
3. Design algorithm to build a handwritten text depending on knowing handwritten text to produce a handwritten text similar to the original one.

## **1.4 Problem Significance and Motivation**

The motivation of this research is that a lot of people are not consider the ancient manuscripts as valuable things, in other words that it does not carry any value or that can best to be in printed form, this leads to disposal or stored the ancient manuscripts until it will damage and the fear of the demise of the Arabic language and abandon of Arabic script, which is a feature in Arabic language. Also the recognition of handwritten text in Arabic language is the interesting fields of artificial intelligence and image processing, and to preserve the ancient manuscripts from damage and loss.

## **1.5 Problem Solution Approach**

There are many methodologies for solving problem such as problem reduction. This research work will use the proposed framework model and may be updated according to the future requirements. The difficulty of the Arabic handwriting recognition is that the accuracy of the character recognition affects on the accuracy of the word recognition. The C# .Net programming language used to build the user interface and the used techniques.

## **1.6 Limitation**

The model deals with a handwritten text contain most of the Arabic letters and the most of forms of the letters. In order to embed similar fragments as original text, a database created for the writers should contain a clear text that doesn't contain overlapped characters.

## **1.7 Goals**

The goal is to design a framework model to recognize handwriting and extract features of letters, then the system learning the method used by the author in writing. The system will be able to transfer from printed text to handwritten text

based on learning during the process of recognizing the original handwritten text. The framework model will handle the missing text fragments in the original document, then by using a search process the system will search for other documents have the same font style to complete the text, otherwise if there is no document has the same writing style the search will be in other documents that have a character features similar to the original handwritten text to embedded in it.

## **1.8 Thesis Overview**

The thesis consists of six chapters: the current one is the introduction to the general view of handwriting recognition and has many sections, present the problem definition, and the motivation, significant and solution approach of the problem and the goal of the thesis. Chapter two presents the terminology used in this thesis such as the pattern recognition and the OCR system. Chapter three is an overview of the literature survey and the related works. Chapter four view the structure of the model which consist of two stages each stage consists of many parts, all of the model components have been explained in detail. Chapter five presents the implementation of the developed system and a test case. Chapter six presents the Conclusion, Discussion and Future Works.

## Chapter Two

# Terminology

### 2.1 Introduction

The system of handwriting recognition helps to develop and progress of the automation process, and improve the interaction between human and machines, and helps with many applications including office automation, a large variety of banking, business. Little research has gone into Arabic handwriting recognition due to the difficulty of the task and the language and the lack of researchers interested or work in this field (Amin A., 1997). In the following sections are descriptions of the concepts related and used in the handwriting recognition.

### 2.2 Pattern Recognition

Pattern recognition is one of the most important abilities of human beings. Relying on this ability, human beings extract useful information about their surroundings. Today, as the digital computer technology has been largely used and developed to simulate this unique ability, the stimulation of this unique human ability with automated machines is becoming more realistic (Mike, O., 2006) .

### 2.3 Optical Character Recognition (OCR)

Optical Character Recognition (**OCR**), refers to the branch of computer science that involves reading text from paper and translating the images into full editable form that the computer can manipulate. An OCR system enables you to take a hard copy of book or a magazine article and scan it into an electronic computer file, and then edit the file using word processor applications.

OCR systems have enormous potential because they enable users to of computers to edit the printed documents. OCR is already being used widely in many areas where searches required hours or days can now be accomplished in a few seconds (Bunke, H. & Patrick S.P. Wang, 1997).

OCR is a process to convert a paper document into editable form, which can be used in word processing and other applications as a typed text. The automatic recognition of handwritten text could be applied in many areas, for example 'form-filling' applications such as postal addresses, checks, mail order forms, and many others. All these applications generate handwritten script from a different writers and writing, which must processed later by computers (Newman, R. & Downton, A., Jan 1997).

## 2.4 Handwriting Recognition Categories

Handwriting recognition is divided into two major categories: online handwriting recognition and offline handwriting recognition as seen in figure 2.1.

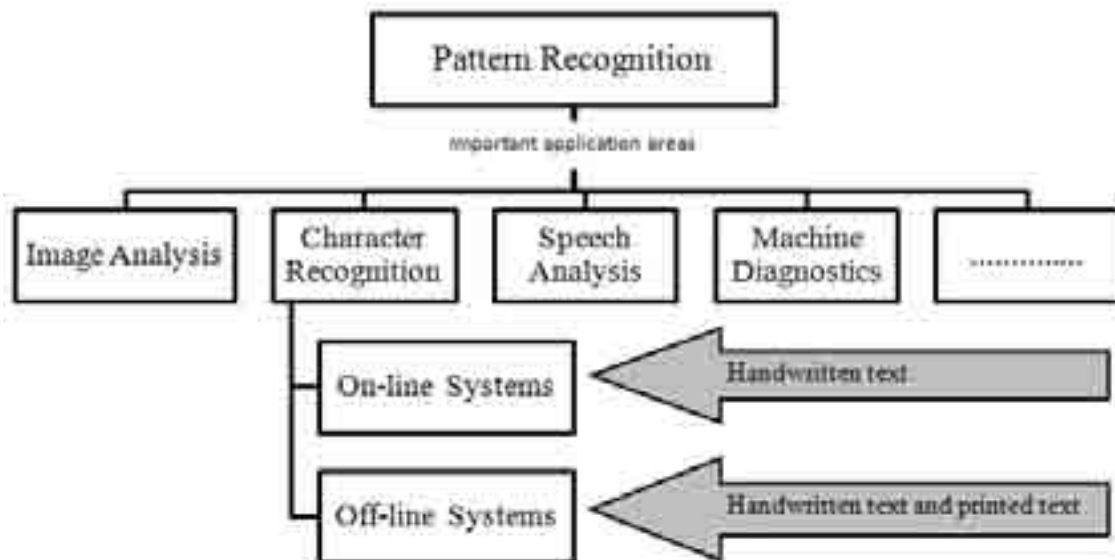


Figure 2.1: Presents the categories of character recognition

### 2.4.1 Online Handwriting Recognition

Online handwriting recognition usually involves a writing pad and an electronic pen. The writing pad samples the pen's movements and translates them into  $x()$  and  $y()$  coordinates. The sampled coordinates are then sent back to the computer in which they are processed (Fujisaki H., et al., 1971).

Online handwriting recognition provides important information obtained from the pen-up and pen-down events called temporal information. Knowledge of this information is important because they form the basis of recognition. The temporal information such as the stroke sequences, the speed of the pen's tip, and the direction in which the strokes are written is helpful to online recognition as seen in figure 2.2. Whenever there is more information recognition about the written word, it becomes easy to distinguish between characters (I. S. I. Abuhaiba, et al, 1994).



Figure 2.2: On-line handwriting inputs (Klassen, T. 2001).



### 2.4.2 Offline Handwriting Recognition

In offline handwriting recognition there is no need to temporal information, because the process of recognition performed after the user has completed writing as seen in figure 2.3 Offline handwriting recognition is usually used for digitizing old books, manuscripts, and other documents such as a fax image. Offline handwriting recognition involves scanning a text and saving it as a digital image, that used by the recognition software. Despite the fact that offline handwriting recognition loses information about how words were actually written, it has its own uses and applications (M. A. Al-Alaoui, et al, 2009) (Liana M., and V. G. Lorigo, 2006).



Figure 2.3: Off-line handwriting inputs (Klassen, T. 2001).

### 2.5 Application of Offline Handwritten Recognition

Some of the important applications of offline handwriting recognition are (Naveen G. and Karun, 2009):

1. Banks checks Reading: handwriting recognition system is very important for signature verification and for recognition of forms filled by user.

2. Postcode Recognition: Offline handwritten recognition system used for recognition handwritten digits and postal code on letters. Handwriting character recognition system (HCR) can read this code and sort mail automatically.
3. Filled-Form reading: handwriting character recognition (HCR) can be also used for form processing. Forms are used for collecting the public information. The information can be handwritten in the space provided.

## 2.6 Arabic language

Arabic language is used by more than one billion people, either in their daily activities or religion-related activities. Arabic characters are used in several languages such as Arabic, Farsi, and Urdu languages. (Tomai C. I., et al., August 2002)

The recognition techniques for other languages such as Latin, Chinese, and Indian achieved cannot be applied to Arabic handwritten text because of the following characteristics of the Arabic text: the cursive nature, letter shape is context sensitive and writing style variability from person to person (Rath T, et al, 2003).

Arabic handwritten recognition devices have low performance due to the characteristics of the Arabic language such as:

- 1- The Arabic language has 28 letters and each letter can assume two to four different forms depending on its position within the word. For example, the letter "م" reads as meem, has an isolated form which is "م", initial form at the beginning of the word "مـ" , in the middle of the word "مـ " and at the end of the word "م".
- 2- The Arabic writing system is cursive in nature; letters are joined together by one stroke. Consequently, the problem of segmentation arises. No clear demarcations exist

between letters much like the cursive version of the English writing system (Mohammad A. A., June 2011).

3- Words may have one or more connected parts. This adds another difficulty to the recognition process. For example, the word “جامعة”, which means university, consists of two connected parts.

(Aouadi N. and Kacem A., 2005) said that “The system has to go through a training stage to detect words, at this stage; the system generates patterns of parts of words considering different samples. Note that Arabic words consist of one or several parts of word. A Part of Arabic Word (PAW) is a connected component which can refer to a diacritic sign, a single letter or sequence of letters or even whole word”. Figure 2.4 shows the meaning of PAW and the table 2.1 show that the Arabic words may have one or more of PAWs.

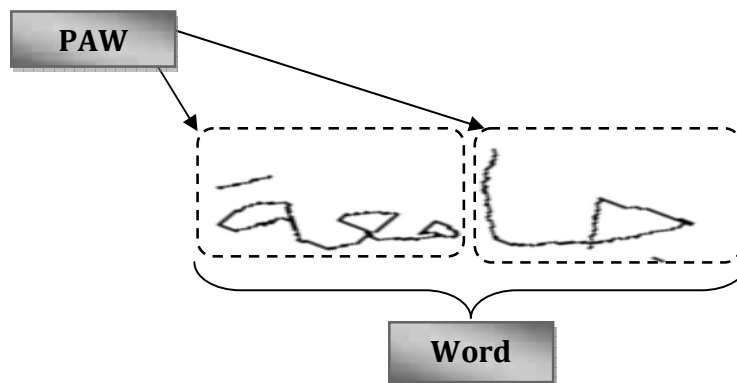


Figure 2.4: Meaning of PAW

Table 2.1: Example of Arabic words with sub words or PAWs

The word image	Number of PAWs or sub words
يعتمد	One
ها مشي	Two
جانزة	Three
الإضافية	Four

4- Some characters may form a new ligature shape in some font of Arabic, which consists two or more characters. For example, figure 2.4 shown the first three letters: "م" and "ج" and "م", in the word "مجموعة", which mean group, are very difficult to segmented as shown in figure 2.5.

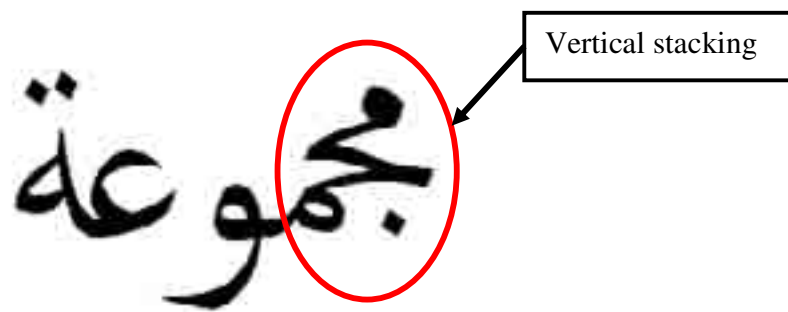


Figure 2.5: example of word in Arabic difficult on segmentation.

5- Some characters have dots on the top such as the letters taa "ت", in the middle, such as the letters jeem "ج", at the bottom such as the letters "ب" baa "ب" (brook & al aghbari, 2008).

Arabic language contains many difficulties in the rules of sentence structure, the drawing of letters, and the connection between the letters. Arabic writing may be classified into three different styles (Khorshed M. S., 2002).

1. Typewritten: This type is the simplest one because the characters are written without ligature or overlaps as shown in figure 2.6.

متأثرا بالظروف الطبيعية

Figure 2.6: Typewritten example.

2. Typeset: This style is more difficult than typewritten because it has many ligatures and overlaps. It is used to write books and newspapers as shown in figure 2.7.

خط الطباعة العربي

Figure 2.7: Typeset example.

3. Handwritten: This is the most difficult style because of the variation of writing the Arabic alphabets from one writer to another as shown in figure 2.8.

اللغة العربية

Figure 2.8: Handwritten example.

Most of the problems faced researchers in the field of Arabic handwriting are the connection of letters; the letters in Arabic have multiple shape or forms depending on the letter's position in the word. The four main letter forms are isolated, beginning, middle, and end of the word.

The letter in the Arabic language can be linked to another letter from its left or its right according to its position in the word, all letters can be connected with another letter on its right , but there are six letters that do not connect to another letter to its left which are “alef” (أ) “daal” (د) “thaal” (ث) “raa” (ر) “zaay” (ز) ”aa” (ع) , an example shown in figure 2.9.

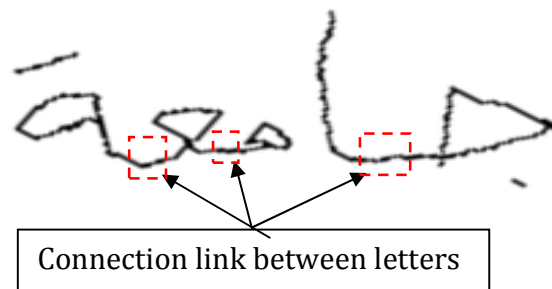


Figure 2.9: Example of letter connection.

In table 2.2 presents the letters of the Arabic language and their forms in writing:

Table 2.2: Arabic letters forms

No	Letter	Main form Isolated	Beginning	Middle	End
1	Alef	أ	-	-	ا
2	Baa	ب	ب	ب	ب
3	Taa	ت	ت	ت	ت
4	Thaa	ث	ث	ث	ث
5	Jeem	ج	ج	ج	ج
6	H'aa	ح	ح	ح	ح
7	Khaa	خ	خ	خ	خ
8	Daal	د	-	-	د
9	Thaal	ذ	-	-	ذ
10	Raa	ر	-	-	ر
11	Zaay	ز	-	-	ز
12	Seen	س	س	س	س
13	Sheen	ش	ش	ش	ش
14	Saad	ص	ص	ص	ص
15	Daad	ض	ض	ض	ض
16	T'aa	ط	ط	ط	ط
17	Dhaa	ظ	ظ	ظ	ظ
18	Ayn	ع	ع	ع	ع
19	Ghayn	غ	غ	غ	غ
20	Faa	ف	ف	ف	ف
21	Qaaf	ق	ق	ق	ق
22	Kaaf	ك	ك	ك	ك
23	Laam	ل	ل	ل	ل
24	Meem	م	م	م	م
25	Noon	ن	ن	ن	ن
26	Haa	ه	ه	ه	ه
27	Waaw	و	و	و	و
28	Yaa	ي	ي	ي	ي

## Chapter Three

# Literature Survey

### 3.1 Overview

The history of character recognition starts as early as 1900, a Russian Scientist “Tyuring” attempted to develop an aid to provide assistance for visually handicapped (Mantas J., 1986). First appeared of handwriting character recognition in the middle fourth decade of the 1940s century with the development of the digital computers (Earnest L. D., 1963).

Compared with research conducted in the field of handwriting recognition in other language there is little researches has been done in the field of Arabic handwriting recognition. (Abandah G. A. and Khedher M. Z., 2005)

Meanwhile in the following sections are the brief descriptions of some available research considering handwriting character recognition. So it is very important researches in this field.

### 3.2 Fractal & Multi-Fractal for Arabic Offline Writer Identification

(Aymen .C, et al. 2010) present a novel method for Arabic text-dependent writer identification based on fractal and multi-fractal features. From the images of Arabic words they calculate the fractal dimensions by using the “Box-counting” method, then they calculate the multi-fractal dimensions by using the method of Diffusion Limited Aggregates (DLA). To evaluate their method, they used 50 writers of the ADAB database, each writer wrote 288 words, with  $\frac{2}{3}$  of words are used for the learning phase and the rest is used for the identification. The results obtained by using nearest neighbor



classifier, demonstrate the effectiveness of their proposed method. The rate of correct identification for some words is more than 90%.

### **3.3 Hough Transform Technique**

(Sofien Touj, et al., 2005) use a Hough transform technique for Arabic optical character recognition. Hough transform (HT) is good for detecting alignment, ascenders and descenders in an image. The basic principle of the (HT) is to define a mapping between image space and parameter space. The idea of this technique is to store all the edge pixels of the target image in a table and defining a reference points for each position, then built the R-Table for each model of the character. Recognition of the characters based on segmentation / recognition approach.

The recorded results show the efficiency of this technique in modeling the different variability of the Arabic script. Different tests show that the Generalized Hough Transform can be easily modified to detect objects in different scales and orientations which may resolve many problems related to the recognition of Arabic printed document without any constraints.

Tests of the system have been made in a set of 166,873 samples of Arabic characters in Arabic Transparent font scanned at 300 dpi. The error analysis has shown confusion in some cases where the character's shape is wholly embedded in another one. A recognition success average rate of 93% is obtained.

The Generalized Hough Transform is also able to detect characters in different fonts. The system have been tested in a set of 234,868 characters in their isolated form in three different fonts, the recognition success average rates obtained are around 97%.

### **3.4 Handwritten Character Segmentation using Baseline Approach**

(Alireza A., et al., 2010) presented an efficient approach to segment Persian off-line handwritten text-line into characters. The proposed algorithm traces the baseline of the input text-line image and straightens it. Subsequently, it over-segments each word or a sub-word using features extracted from histogram analysis and then removes extra segmentation points using some baseline dependent. The baseline straightening method that they used was very helpful for getting more accurate segmentation results. They tested the proposed character segmentation scheme with two different datasets. On a test set of 899 Persian words or subwords created by them, 90.26% of the characters were segmented correctly. From another dataset of 200 handwritten Arabic word images they obtained 93.49% correct segmentation accuracy. In their experiment they got some missing segmentation points because of overlapping two characters or placing one of the characters above the other one.

### **3.5 Recognition of Handwritten Arabic text Using Neural Network**

(Alnsour and Alzoubady, 2006) describe the automation recognition of handwritten Arabic text using a type of the neural network classifier. The system was trained and validated on 600 images and tested on 250 images, consisting of 210,000 Arabic characters written by 300 writers. It is designed for training and testing recognition system for handwritten Arabic characters from several writers whose writing from acceptable to poor in quality. The performance of this system is comparable to existing Arabic character recognition systems.

A two of experiments have been conducted. The experiments use 141 characters classes. Features were extracted from images using the structural feature extraction. The

result achieved were very promising and identification accuracy as high as 90% was obtained, the Neocognitron Artificial neural network classifier have shown good performance.

### **3.6 Related work**

There are many research have been done in the field of handwritten characters recognition. In the following subsections some of researches are most related to this thesis. These researches use different methods and techniques.

#### **3.6.1 Arabic Handwriting Recognition System Using Genetic Approach**

(Hanan .A, et al, 2010) developed a complete system to recognize off-line Arabic handwriting image and Arabic handwriting and printed text database AHPD-UTM that used to implement and test the system. That system consists of preprocessing phases and segmentation phase, the system depend on thinning the image until recognition phase. The genetic algorithm stand on feature extraction algorithm that defined six feature for each segmented peak. The system recognized Arabic handwriting with 87% accuracy, while the confusion and rejection rates are 8.4, those causes for several problems like characters with broken loops and character segmentation problem. The Peak connection solved some of the segmentation problems and helped to provide better accuracy, the conjunction method solve the over segmentation problem. The recognition problem has been solved by genetic algorithm.

#### **3.6.2 Using binary Representation of Character for Feature Extraction**

(Sarhan M. and Helalat, 2007) describe that each typed Arabic character is represented by binary values that are used as input to a simple feature extraction system, whose output is fed to an ANN that consists of two layers. Simulation results are

provided and show that the proposed system always produces a lower Mean Squared Error (MSE) and higher success rates.

The system inputs are the Arabic characters, where each character is represented by a matrix of 7 x 5 binary pixels, producing a 35-element input vector, which is presented to the feature extraction stage. The feature extraction stage obtains the standard deviation of the input vector and produces a 36-element vector with the additional element being the standard deviation of the original 35 values. This output vector is then fed to the ANN. The ANN is composed of two layers. The first layer consists of 10 neurons and the second layer consists of 28 neurons, the number of Arabic characters. Simulation results are provided and show that the proposed system always produces a lower MSE and higher success rates than the current ANN solutions, especially when the contaminating noise level is low.

### **3.6.3 Arabic Handwritten Word Recognition Using Dots Concepts**

(Saeed M., Karim F., 2008) discuss the lexicon reduction for offline Farsi/Arabic handwritten word recognition using dots concepts. The main principle of this technique is to eliminate unlikely candidates by extracting and representing the number and the position of dots with respect to the baseline from the input image. Recognition rate and recognition time are affected by the lexicon size. Recognition speed is the most important criterion if the lexicons are large. On the other hand, recognition accuracy is critical issue for small lexicons. There are a variety of methods can be followed to minimize the size of lexicon such as knowing some information about the application environment characteristics of input pattern, and the clustering of the same lexicon entries. After extracting the dots candidates a classifier is needed to categorize them into classes (single

dot, double dots, and triple dots). A model discriminate discrete HMM is used for recognition because this approach is used for reading 200 city names from postal address fields.

### **3.6.4 Off-line Arabic Handwriting Recognition Based on Projection Profile and Genetic Approach**

(Hanan Aljuaid, 2009) in this research proposed a complete system of off-line Arabic handwriting recognition based on projection profile and genetic approach. Genetic algorithm is computer science technique specially used for optimization and search problems. Preprocessing in this approach is done using a thinning algorithm to thin the image word, and then to extract the vertical and horizontal projection profiles.

In order to define the shape of the characters number of features such as, length and width of the character, loops, and points are needed to be extracted by tracing the boundary of the image from right to left. Feature extraction is followed by recognition phase based on genetic algorithm which depends mainly on fitness function that is calculated for each input vector in order to choose the best fitness one.

### **3.6.5 Using Hybrid Hidden Markov (HMM) and Artificial Neural Network (ANN)**

(S. E. Boquera, et al., 2011) described the off-line recognition of handwritten texts using hybrid hidden Markov (HMM) and artificial neural network (ANN) models. The main principle of this technique is using artificial neural networks (ANNs) in preprocessing stage to remove the slant and slope from text lines and to normalize the size of the images. The slope and the horizontal alignment are estimated using local extreme from a text image.

Normalization is achieved by computing the reference lines of the slope and slant-connected text. Recognition is based on hybrid HMM/ANN where graphemes is modeled using left to right Markov chains and single neural network is used to estimate the emission probabilities.

### **3.6.6 Fuzzy Logic approach to Recognition of Isolated Arabic Characters**

(Majida Ali Abed, et al., 2010) proposed a system does not require segmentation of the Arabic words to characters. The used approach is very suitable for the recognition of a complex data such as Arabic characters. The fuzzy logic implementation uses very little memory, making it possible to provide full functionality. The proposed system work in many steps, entered three different shape for every isolated Arabic character saved in many templates, then applied laws and stages of fuzzy logic, then save the results in different files, then enter four shape for the character they wanted to recognize different from the three shapes were saved, then the last stage is compare the results recognition with the saved results of characters. They used for testing the characters ( Alif, Baa ,Geem ,Daal ,Raa, Seen, Dhad ,Taa, Ain ,Faa ,Kaf , Lam , Meem ,Haa ,Waw ,Yaa ). The proposed system was tested on 96 different shape of characters and a 88% recognition success rate was obtained. The proposed system has been implemented and tested on Matlab R2008b environment. Experiment results showed the effectiveness of the proposed system with isolated Arabic characters. The future works will be improved by integrating other characters based on geometrical measures (example: the size of the input character, ratio height/ width or line of basis of character).

## Chapter Four

# The Structure of Framework

### 4.1 Introduction

In the character recognition systems there are many steps, these steps begin with data capture for the text that will be recognized, the data capture process done by using one of the image input devices such as the digital camera and optical scanners, these devices convert the hard copy of paper to full editable digitized form to be used by the applications on computer, the image saved in computer in one of the image format i.e. (JPEG, bmp, . . etc.).

Preprocessing stage consists of processes that used to prepare the image to be recognized, the number of algorithms used in the processes depends on many factors such as paper quality, resolution of the scanned image, and the amount of skew in the image. The process of converting images to gray scale, noise removal, resize image, binarization and thinning are common processes used in preprocessing stage before the segmentation process, then the features of segmented characters in the feature extraction process used to recognize the character to be classified and used later to handle the text fragments, all of these processed shown in figure 4.1 in the next page.

### 4.2 The Model Structure

Figure 4.1 presents the block diagram for the model structure, which consists of eight stages, these are; Input 1 scan text image, recognition stage, extracted feature, Input 2 Typed Text Fragment, Handwritten text similar to the original text for the fragment, Handwritten Missing Characters in estimated font style, Embedded the Fragment in the

Original Document, Handwritten text with Fragments and Notes and Handwritten text with the Fragments. In the next subsection the detail description of all stages.

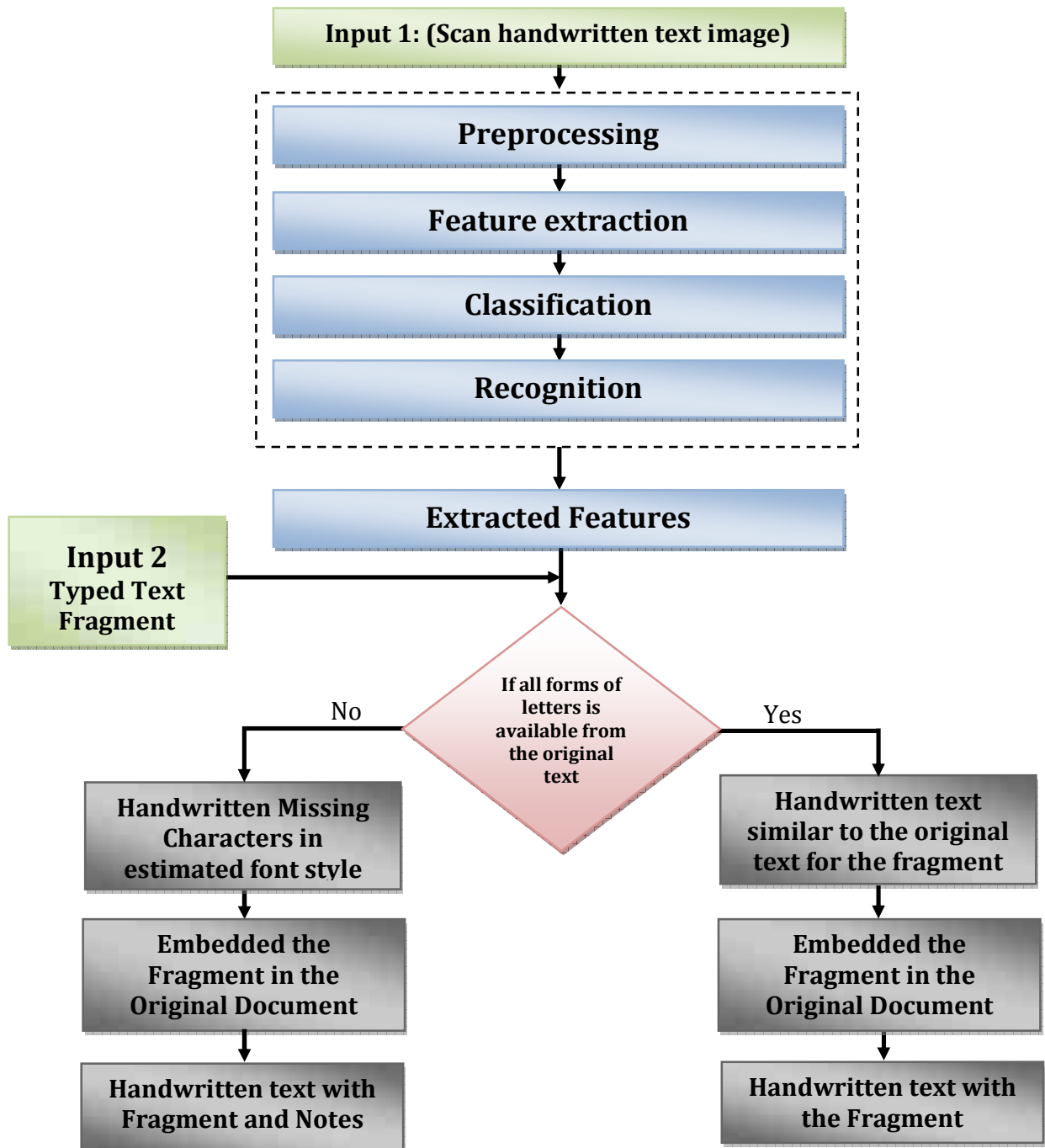


Figure 4.1: a Block Diagram for the Model



### 4.2.1 Preprocessing

The input of the digitizer typically contains noise due to erratic hand movements and inaccuracies in digitization of the actual input. Original documents are often dirty due to smearing and smudging of text and aging.

The documents sometimes are very poor quality because of the seeping of ink from the other side of the page and general degradation of the paper or ink or both. Preprocessing is concerned with the reduction of these noises. The number and type of preprocessing algorithms employ on the scanned image depend on many factors such as paper quality, resolution of the scanned image and the layout of the text. There are many processes performed before the recognition such as: thresholding or binarization, Resize the image, converting a grayscale image into a binary black-white image and thinning as shown in figure 4.2.

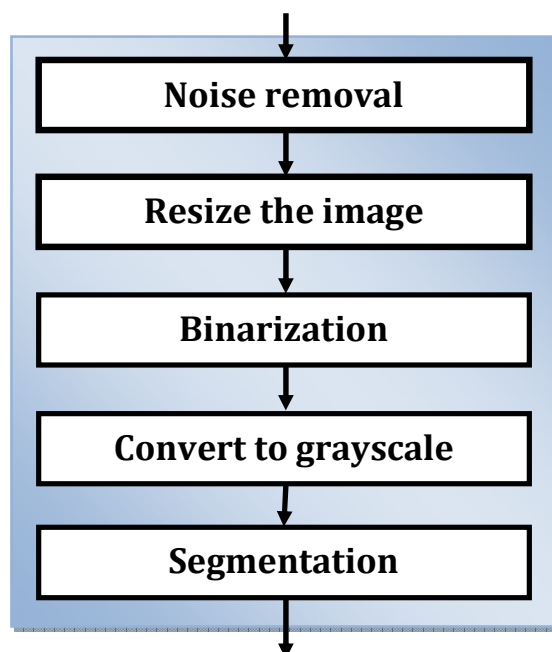


Figure 4.2: Preprocessing stage

#### **4.2.1.1 Noise Removal Process**

Noise removal is one of the common and important operations performed before the recognition process, because there are some noises appear like a dot and it have an effect on text specially in the Arabic language which the difference between some characters just the dot like Arabic character saad “ص” and Arabic character dhad “ض”, and it's important in the extraction of the foreground text (Plamondon R., and Srihari S.N, 2000).

At the beginning of testing the application the AForge library used to remove the noise by using closing filter which applied to binary image, the filter may be used connect or fill objects. Since dilatation is used first, it may fill object areas. Then erosion restores objects. But since dilatation may connect something before, erosion may not remove after that because of the formed connection.

During application testing shows that the samples do not contain a lot of noise and the samples are high-quality, after applying the closing filter which uses to remove noise the result was not satisfactory. But after execution of the digitizing (binarization) process the noise disappeared. The segmentation process starts with detect the lines in the text image before segment the word to PAWs, the line detect ignored the noise that appear out the line, therefore, it is not necessary to use an algorithm to remove noises in this thesis.

#### **4.2.1.2 Resize the Image**

This process used to reduce image size to a size smaller than the original and find the medial axis which defines as a set of pixels  $S$ , these pixels have an equal distance from the boundary pixels around it, the output of this process is skeleton for the handwritten word, this process save the geometry and the connections between the characters and the location of original character..

In this thesis the resolution of sample image was very high and the image was too large to deal with it and cause a slow in the work of the application, image resize process used to resize the image to a size smaller than the original to be shown as A4 paper and to make the application faster.

AForge.NET framework provides set of filters to perform image resize, such as nearest neighbor filter, bicubic interpolation filter, and resize-bilinear filter. The resize-bilinear filter used to resize the image, the code in c# to resize the image is:

```
Resize = new AForge.Imaging.Filters.ResizeBilinear ( newWidth , newHeight );
```

#### **4.2.1.3 Binarization Process**

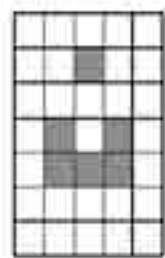
In the image each pixel has a value between 0 and 255. Many researchers choose to work with a binarized the gray values will be converted the image from the grayscale to binary image the pixels will be presented either '0' for white (background) or '1' for black (foreground) (Azizah S., Nasir S.M. and Mohamed O., 2010) .

Thresholding is the method that used to binarize the image. The grayscale images are represented as binary images by picking a threshold value. There are two types of thresholding: global and local thresholding.

In global thresholding, one threshold value is used for the entire document image which is often based on an estimation of the background intensity level, but local thresholding used in the images that have varying levels of intensities, such as satellites or cameras pictures.

In this thesis a global threshold used, it suffices to distinguish the background and the foreground., because the characters are written on a white background, Figure 4.3

displays an image of the Arabic letter 'ن' in its image file format and after it had been binarized.



(a)

0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	1	0	1	0
0	1	1	1	0
0	0	0	0	0
0	0	0	0	0

(b)

Figure 4.3: The Arabic letter "Noon" (a) Bit map image, and (b) Matrix representation (Sarhan M.A., Helalat O., 2007).

The algorithm used to digitize or binarize the image shown in the following steps:

**Step 1:** Calculate the histogram for the grayscale image.

**Step 2:** **B** = max value in histogram for the black pixels.

**W** = max value in histogram for the white pixels.

**Step 3:** Threshold value **T** =  $(\mathbf{W}+\mathbf{B})/2$ .

**Step 4:** For **i =1 To N**, all pixels in image

```
{
  If (grayscale value > T)
    {Pixel color = white}
  Else
    {Pixel color = black}
}
```

#### 4.2.1.4 Convert to Grayscale

Grayscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest (Stephen J., 2006).

The sample images are text image, so any algorithm used will give a satisfactory result, the code of converting the image to grayscale is:

```
// create grayscale filter (BT709)
```

```
Grayscale filter = new Grayscale (0.2125, 0.7154, 0.0721);
```

```
// apply the filter
```

```
Bitmap grayImage = Grayscale.CommonAlgorithms.BT709.Apply( image );
```

The equation of convert the image to grayscale is:

$$\text{Grayscale} = (0.2125 * R) + (0.7154 * G) + (0.0721 * B)$$

Where R is the value of red color, G is the value of green color, and B is the value of blue color in each pixel.

#### 4.2.1.5 Segmentation

Segmentation is a critical and important step in Arabic handwriting recognition systems. After the preprocessing stage, the systems of character recognition perform segmentation operation on the text and end with individual character or stroke before recognition stage (Jumari K. & Mohamed A. Ali, 2002).

The goal of a segmentation process is to partition a word image into regions, each region containing an isolated character, figure 4.4 shown an example of segmentation for an Arabic word "جامعة", the word is clear and don't has an overlapping characters. The

handwritten character segmentation process and recognition process are closely coupled, because it is difficult to segment characters without the support of recognition algorithms, unlike the problem of printed character recognition, figure 4.5 shown the problem of segmentation in handwritten, the segmented characters are not clear and cropped.



Figure 4.4: Segmentation Process

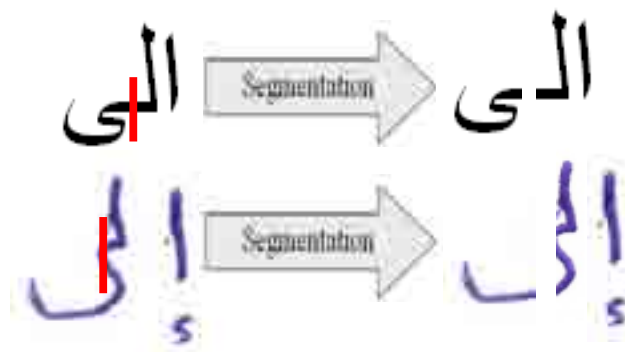


Figure 4.5: Segmentation Process of printed text and handwritten text

There are three approaches for segmentation and other hybrid approaches that are combinations of these three approaches.

1. The classical approach: by using general features this approach segments the image into sequence meaningful sub-images, based on 'character like' properties.
2. Recognition based segmentation; in this approach the system depends on components match classes in its alphabet.

3. Holistic methods (or global approach), in this approach there is no need to segment words into characters, it recognizes the whole words.

The connected components labeling algorithm is used in computer vision to detect connected regions in binary digital images, although color images and data with higher dimensionality can also be processed, it also called region labeling, , or region extraction, it used to extract objects from binary images by assigning a unique label to each connected region of foreground pixels.

Connected component labeling works by scanning an image, pixel-by-pixel (from top to bottom and left to right) in order to identify connected pixel regions, i.e. regions of adjacent pixels which share the same set of intensity values  $V$ . (For a binary image  $V = \{1\}$ ; however, in a gray level image  $V$  will take on a range of values, for example:  $V = \{51, 52, 53, \dots, 77, 78, 79, 80\}$ .)

Finding the connected components in a binary image can be done in several different ways. The simplest method is to iteratively replace each label with the minimum of its 8-connected neighborhood. The algorithm begins with an initial labeling of all 1-pixels and ends when no more replacements can be made.

The graph contains vertices and edges, the vertices contain information required by the comparison heuristic, while the edges indicate connected neighbors. An algorithm traverses the graph, labeling the vertices based on the connectivity and relative values of their neighbors. Connectivity is determined by the medium; image graphs, for example, can be 4-connected or 8-connected as shown in figure 4.6.

There are two versions of connected components labeling: (Shapiro, L., and Stockman, G., 2002)

### 1. One-pass version

The algorithm identifies and marks the connected components in a single pass. The run time of the algorithm depends on the size of the image and the number of connected components (which create an overhead). The run time is comparable to the two pass algorithm if there are a lot of small objects distributed over the entire image such that they cover a significant number of pixels from it. Otherwise the algorithm runs fairly fast.

### 2. Two-pass version

Relatively simple to implement and understand, the two-pass algorithm iterates through 2-dimensional, binary data. The algorithm makes two passes over the image: one pass to record equivalences and assign temporary labels and the second to replace each temporary label by the label of its equivalence class.

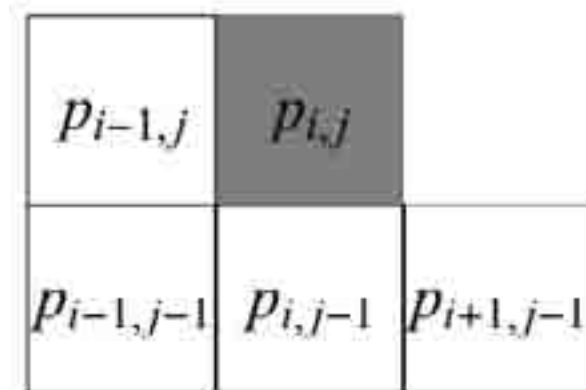


Figure 4.6: Connected pixel (Shapiro, L., and Stockman, G., 2002)

In this thesis the AForge.Net used which use the one-pass version, the code in C# is: `AForge.Imaging.Filters.ConnectedComponentsLabeling (DigitizedImage);`

The segmentation technique used in this thesis depend on the PAW segmentation which done by applying the connected component labeling algorithm on the image text, each PAW contain at least one character, the segmentation point will be detected on the



PAW according to number of characters, for example the PAW that contain one character do not need segmentation, but the PAW that contain two characters need one segmentation point between the two letters, the following algorithm is used to segment the PAW to characters:

**For each PAW in the text image**

**Step 1:** segment the text image to PAWs by using the connected component labeling.

**Step 2:** Associate each paw image with its text.

**Step 3:** Count the characters in each PAW (from text information).

**Step 4:** calculate the horizontal histogram for each PAW and find the maximum value to detect the baseline.

**Step 5:** calculate the vertical histogram for each paw and find the minimum values to detect the segmentation points.

**Step 6:** according to number of characters in each PAW:

**If** (number of character in PAW= 1): No segmentation

**If** (number of character in PAW= 2):

- Find the midpoint.
- Skip-length=20% from the beginning of the PAW and the end of it
- Take-length= skip-length\*3.
- Find the minimum values of vertical histogram after skip-length and within take-length.
- Select the midpoint of the minimum values.

**Else** // more than 2 characters

- Calculate the vertical histogram.
- Calculate the max of the vertical histogram.
- Minimum threshold= 20% of the max of the vertical histogram.
- AverageCharLength = PAWlength / number of characters.

**Loop** fro i=0 To Number of segmentation points

- Interval-begin<sub>i</sub> = I \* AVCL \* 50%.
- Interval-end<sub>i</sub> = (i+1)\*AVCL\*150%.
- Minimum=minima of vertical histogram after interval-begini and within interval-end.

Local minima=all vertical histogram pints < minimum + threshold.

Index=index of the lowest local minima that has a black pixel within the base line.

Segmentapoint[i] = index.

**End loop**

**End loop**

#### **4.2.2 Feature Extraction**

In this part the characteristics of the author of the handwritten document will be extracted and will be used for finding a fragments style similar to the original text to be embedded in the original document.

Feature extraction is the main process and the important one of the character recognition system. The definition of feature extraction is extracted from the raw data the information which is most relevant for classification purposes, in the sense of minimizing within-class pattern variability while enhancing between-class pattern variability.

Features can be extracted from characters or words. Extracted features should provide uniquely relevant identification information of character class without repeat.

The Arabic handwritten characters have features such as the letter's secondary components, main body, skeleton, and boundary. These features are studied and statistically analyzed to reach the targeted characterization (Abandah G.A. and Khedher M.Z., 2005). There are many features for every character in Arabic language, such as main body features and secondary, which are described in the next subsections.

#### **4.2.2.1 Main Body Features**

Main body feature is the letter image after removing the secondary components , there is a letters have secondary components like the letter "jeem" (ج) and there is letters don't have like the Arabic letter "lam" (ل) and the Arabic letter "meem" (م) and there are other letters.

The main body of a letter has many features such as size, area, width, Height, pixel distribution, orientation, roundness, number of loops the researcher in the field of Arabic handwriting recognition chooses some or all of these features as needed to get to a high percentage of the character recognition.

#### **4.2.2.2 Secondary Component**

Most of Arabic letters have a secondary component, mean that the letter consist of two parts the main one and the secondary as shown in figure 4.7, in Arabic the secondary component is the part that not connected the main body like dot in most of Arabic letters, for example the letter (ج), the secondary component in this letter is the dot within its main body.

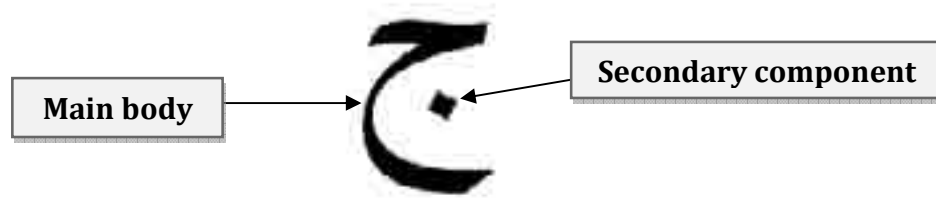


Figure 4.7: Arabic letter "jeem"

By using the connected component labeling techniques we can detect the secondary components of the letter, these techniques are done by segmenting the binary image into its disconnected components.

In Arabic letters the type or position or number of secondary components is very important features. For example, recognizing three dots above the main body are sufficient to recognize the letter “sheen” (ش), in the case of the letter Taa (ت) and the letter Thaa (ث) both have a secondary component above the main body but the different is the number of dots, table 4.1 and table 4.2 shown the type or position or number of secondary components (Abandah G.A. and Khedher M.Z., 2005).

Table 4.1: characters have secondary components

Number	Secondary component type	Examples
1	No secondary	و ه م ل ع ص ر د ح ا ء
2	One Dot	ن ف غ ض ز ذ خ ج ب
3	Two Dots	ي ف ق ة ت
4	Three Dots	ث ش
5	Vertical Bar	ط
6	Vertical Bar and dot	ظ

Table 4.2: Possibilities of the secondary components position

Number	Secondary position	Examples
1	No secondary	وهملعصسردحاء
2	Above	نفاقغضشذرختثة
3	Within	كظج
4	Below	يب

Structural feature extraction used in capturing the essential shape features of characters generally from their skeletons or contours. One approach to feature extraction would be the use of features which are intuitive in the sense that they are directly perceptible to humans, Loops, lines, intersections, and endpoints.

This collection of geometric features gave surprisingly good recognition results in the context of handwritten characters.

Table 4.3: Example of Features for lines, curves, loops for Arabic characters

Feature	Existence	Examples
Line	Vertical	اطظل
	Horizontal	ك
Loop	No loop	ارز
	Loop	فقهه
Curve	Open carves	ن
Intersections	No intersections	ا
	One position	م
	Two positions	ظظ

The feature extraction step is carried out to determine character primitives which may be used for their recognition, many primitives are selected for analysis in software using the dominant point method. These are described in Figure 4.8 (Laheeb M.A. and Ayman J.A., 2006)

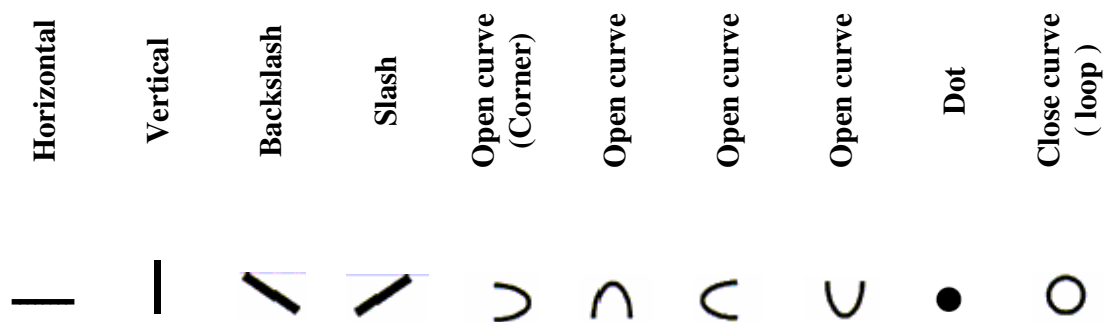


Figure 4.8: Primitive features.

### 4.2.3 Classification

Classification is the decision making stage Based on receiving the output which are the features of characters from the feature extraction process, the classifier recognizes the character or best guess of a character that represents the input features. The classification methods are three types: structural, statistical or neural network classifiers. In this thesis mathematics equations programmed in C# used to extract the feature and classifier the character according to these features.

In this thesis the features calculated for each character are aspect ratio, center of gravity, fullness, loop size, loop completeness, aspect ratio of single dot, two dots count, and three dots count.

The aspect ratio is the ratio of the width of a shape to its height, which calculated using the equation:  $\text{Aspectratio} = \frac{\text{width}}{\text{height}}$ , where width  $\equiv$  character width, height  $\equiv$  character height.

The center of gravity (cg) of a distribution of mass in space is the unique point where the weighted relative position of the distributed mass sums to zero. The following equations used to calculate this feature:

$$X_{\text{cg}} = \frac{\sum_{i=1}^N P_i X_i}{M}, \text{ For the X axis.}$$

$$Y_{\text{cg}} = \frac{\sum_{i=1}^N P_i Y_i}{M}, \text{ For the Y axis.}$$

Where N is the total number of pixels in the image,  $P_i = \begin{cases} 1 & , \text{ black pixel} \\ 0 & , \text{ white pixel} \end{cases}$ ,  $X_i$  is the x-coordinate of  $P_i$ ,  $Y_i$  is the y-coordinate of  $P_i$ , and M is the total number of black pixels.

Feature of fullness or blob's (object or character) fullness, calculated as  $M/N$ , where N is the number of black pixels, and M is the total number of pixels in the image. If it equals to 1, then it means that entire blob's rectangle is filled by blob's pixel (no blank areas). If it equals to 0.5, for example, then it means that only half of the bounding rectangle is filled by blob's pixels.

The other features as, the loop size feature which calculated using the equation of the area of the loop as width\*height, the loop completeness feature calculated just for the characters which have a loop, Aspectratio of single dot count calculated as width\height,

the two dots count and three dots count features which used to count the dots by using the AForge to count the blob's.

Euclidean geometry was used to calculate the difference between the characters features of the writer and other writers to find the nearest percentage of match between the characters; this will be used in the last stage in search process of characters fragments.

#### **4.2.4 Recognition**

The recognition process uses the extracted feature set of the handwriting image as input to determine which model class has the best similarity for the input (Feliachi A., et al., 2002). The four best-known approaches for pattern recognition are:

1. Template matching: comprised of measuring the similarity between input image, typically a 2-dimension shape, and a group of prototypes or templates.
2. Statistical: uses a decision function based on the feature set of the input image and the representation of feature space of different classes.
3. Syntactic: views a picture as a language description and a class as sentences that belong to the language. A specific class can be derived according to a grammar.
4. Neural network: is a massively parallel computing scheme having some organized structure to learn non-linear input and output relationship.

In this thesis the statistical approach used to recognize the characters by comparing the features of characters to find the appropriate and similar characters to the original text.



#### **4.2.5 In case If there is any form not available**

The Missing Characters will be written in estimated font style from another writer similar in characters features and then embedded in the original document; the result is handwritten text with Fragment and Notes, the notes are the fragments are not written by the original writer and it found in text to another writer and the features of characters are similar or close to the original characters.

#### **4.2.6 In case if all forms available**

The Missing Characters will be written in the same style of the same writer then will be embedded in the original document; the result is handwritten text with Fragment without notes.

## Chapter Five

# Implementation and Results

### 5.1 Introduction

In this thesis the C# .Net programming language used to develop the system. C# programming language used to build the User Interface (UI), and all preprocessing processes such as: loading the image from the computer, convert the image to grayscale, binarization, and thinning, connected component labeling, segmentation process, and feature extraction process. The process of classifying the character has been done according to the features that calculated using mathematic equations. Aforge.Net library used to help in preprocessing stage. Microsoft SQL server 2008 used to create the database.

### 5.2 Data Collection

The starting point of the project was the creation database with all Arabic character forms images. Character images are handwritten digitized images, characters are written by 17 people in different handwriting styles and in different fonts. This means that characters on paper have different sizes and different resolutions; the sample contains 118 words, 400 characters of all forms of characters. Sample image for one of the writers shown in figure 5.1.



Figure 5.1: example of collected data

### 5.3 Implementation

In this thesis the C# programming language is used for the implementation of the framework model. The program was dealing with data sets of text image quickly and the developed application to handle the fragments. In the following subsections are the detail descriptions of the implementation.

### 5.3.1 Load Image

In this step the input digital image used to recognize characters. Figure 5.2 represents the step of loading image in preprocessing stage and how it appears after loading shown in figure 5.3.

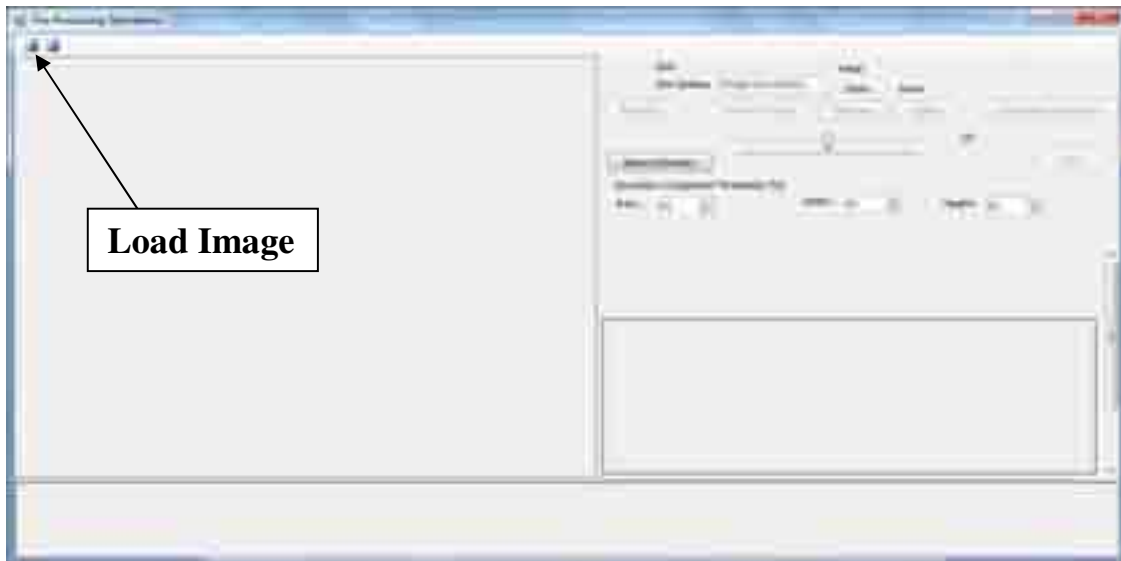


Figure 5.2: Load Image.



Figure 5.3: After loading image, the image appear in the preview area

### 5.3.2 Preprocessing

After loading the character image, the image is preprocessed by many steps, which are mentioned in chapter four, the following are these steps of preprocessing.

1. Resize the image, this process resize the image to be smaller than the original one to complete the other processes by click the button 'resize to A4' as shown in figure 5.4.

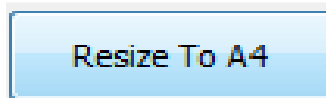


Figure 5.4: resize button

2. The RGB image is converted into a Gray scale image as shown in figure 5.6 by clicking the button "convert to gray" in figure 5.5. The image converted to grayscale.

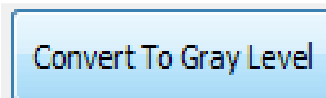


Figure 5.5: convert to gray button.



Figure 5.6: input image in grayscale

3. Digitizing, convert the grayscale image to binary representation, as shown in figure 5.7.

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف  
 الإضافية وترتيبها الحجابي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش  
 ص ض ط ظ ع غ ف ق ك ل م ن  
 ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بحيث يعتمد على وصل  
 او دمج الحروف ببعضها لتكون مقاطع .  
 ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من  
 الابدعية العربية الاختيارية غير ماسبق مثل الضم والفتح والكسر .  
 هناك خطوط واشكال متنوعة للكتابة من هذه الخطوط مثلاً:  
 النسخ والرقعة والتلث .

Figure 5.7: gray scale image after digitizing process

4. The next step is the connected component labeling process, in this process the connected component, line, main body, secondary component detected as shown in figure 5.8.

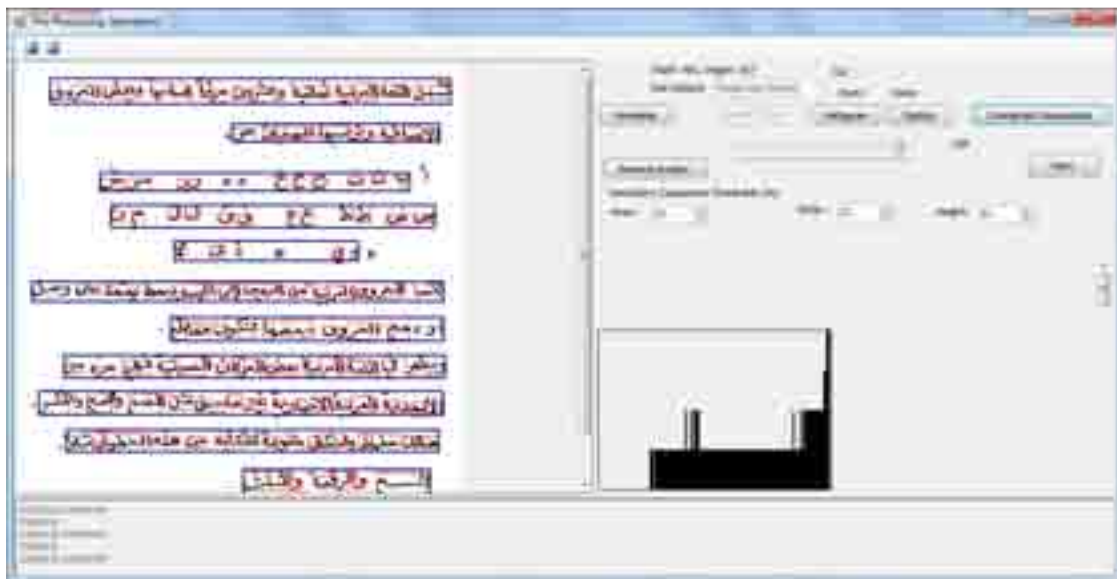


Figure 5.8: Connected Components Labeling Process

Then the text components appear segmented as primary and secondary component, the main body of PAW selected in the red rectangles, the secondary component selected in blue rectangles, and the line selected in a dark blue rectangle as shown in figure 5.9.



Figure 5.9: Segmented handwritten text.

The connected component labeling process segment the text to PAWs, each PAW in the text contains one character or many characters connected together, the preprocessing form in figure 5.10 present the typed text prepared to be segmented to PAWs and characters.



Figure 5.10: The PAWs Form.

After clicking on the process button, the text will be segmented to PAWs as shown in figure 5.11; the Arabic statement “اللغة العربية” consists of five PAWs listed in the figure on the left side.

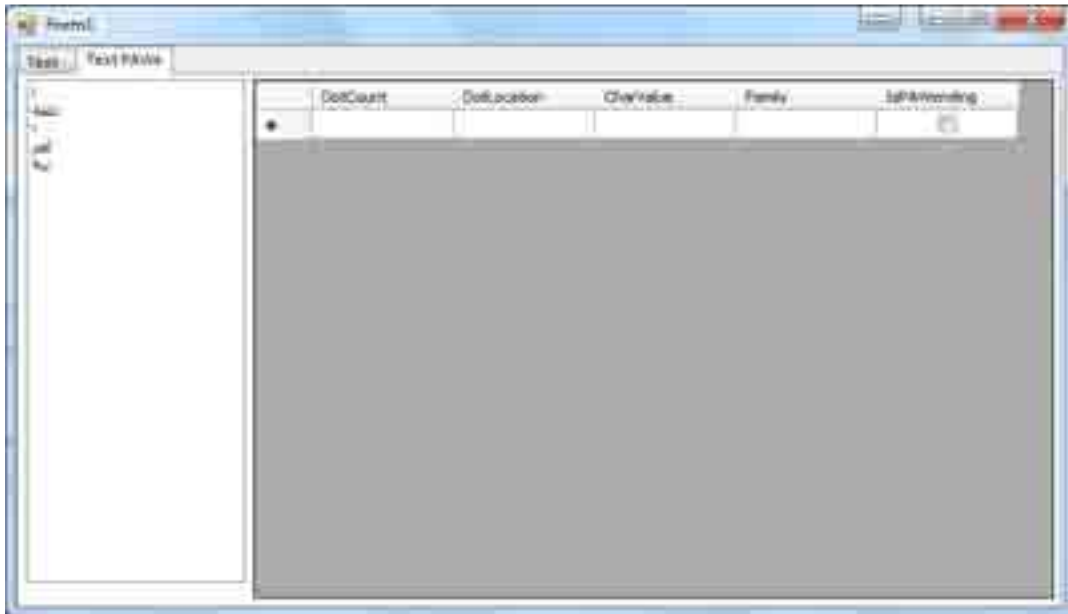


Figure 5.11: Text PAWs process of Arabic statement.

The right list as shown in the figure 5.12 presents of the character on each PAW just when single click on any PAW from the left list.

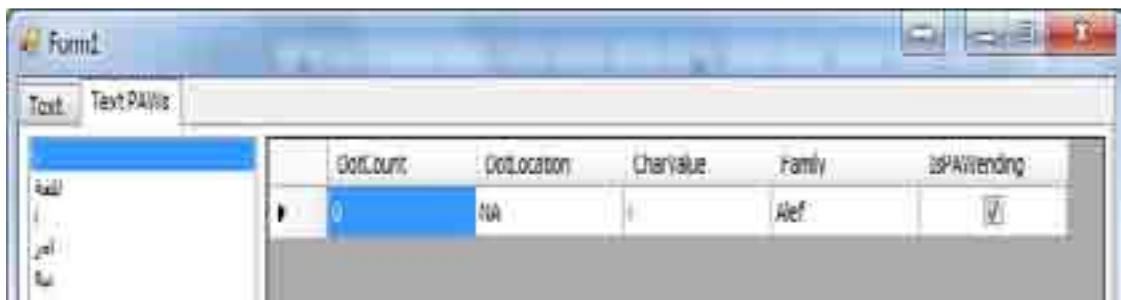


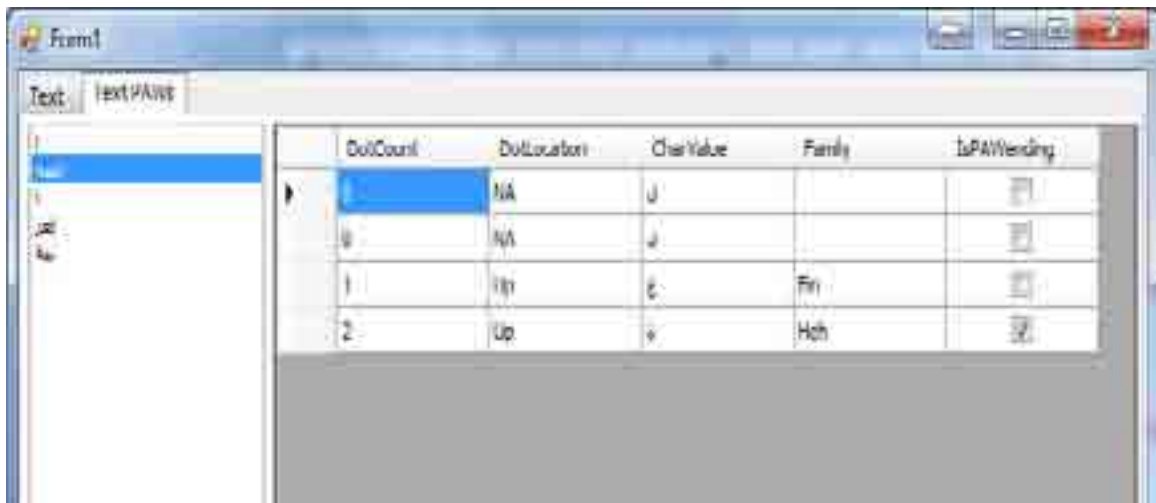
Figure 5.12: Example of PAW consists of one character

There is a description for each character one each PAWs presented in the right list consists of the four columns:



1. Dot Count: present the number of dots with the character, which can be one or two or three dots or without dots.
2. Dot Location: present the location of the dot of each character, which can be up or down or within the character.
3. Character Value: present the character writing style in Arabic.
4. Family: there are characters grouped in one family, this family created on the basis of the shape of writing style as shown in table 5.1.
5. Is PAW ending: present the character that the PAW ends with it.

The figure 5.13 presents the description of the PAW “للغة” which is consist more than one character.



DotCount	DotLocation	CharValue	Family	IsPAWending
0	NA	ا		<input type="checkbox"/>
0	NA	آ		<input type="checkbox"/>
1	Up	أ	Fin	<input type="checkbox"/>
2	Up	آ	Hch	<input type="checkbox"/>

Figure 5.13: Example of PAW consist more than one character.

Table 5.1: Family of characters in basis of writing style

Family name	Family value	Characters of family
Alef	ا	أ إ آ ء
Baa	ب	ب ت ث
Jeem	ج	ج ح خ
Daal	د	د ذ
Raa	ر	ر ز
Seen	س	س ش
Saad	ص	ص ض
TTa	ط	ط ظ
Ein	ع	ع غ
Faa	ف	ف ق
Kaaf	ك	ك
Laam	ل	ل
Meem	م	م
Noon	ن	ن
Heh	ه	ه
Yaa	ي	ى ي
Lam-Alef	لا	لا

Then after the typed text segmented to PAWs, the handwritten image also segmented to PAWs and associated to the printed to classify the PAWs in the text as shown in figure 5.14.



Figure 5.14: typed and handwritten text prepared to associate

Then in figure 5.15 the process of associating between the PAWs from the typed text and handwritten text, the process of associate and classifying start by clicking the process button on the down side of the form.

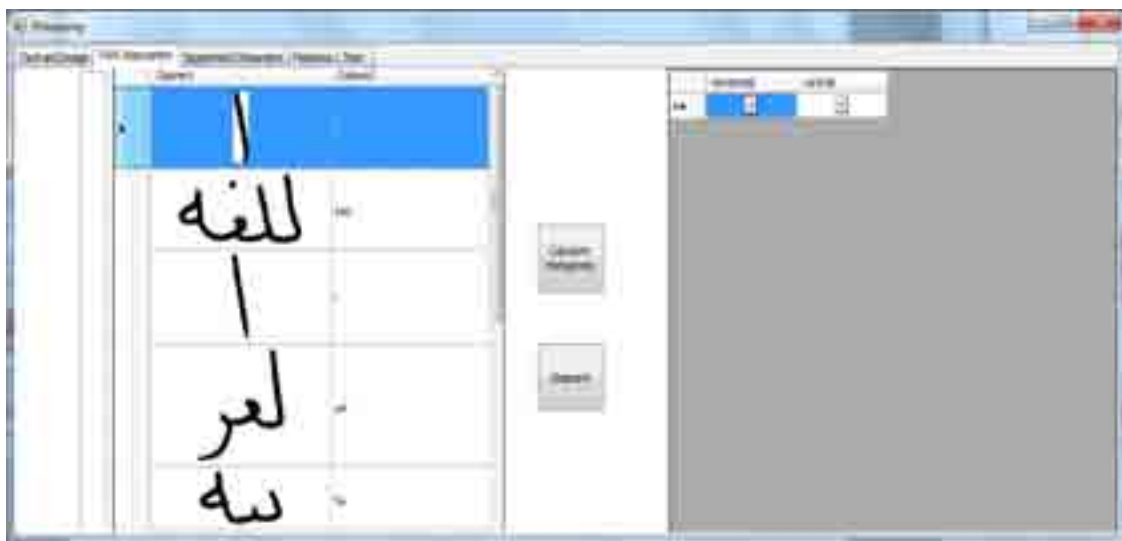


Figure 5.15: Association process of PAWs

To complete the segmentation process the horizontal histogram is needed to detect the base line of handwritten text and the vertical histogram is needed to detect the point of segmentation between the characters, figure 5.16 shown the horizontal and vertical

histogram on the right side of the form for each PAW, by click the calculate histogram button.

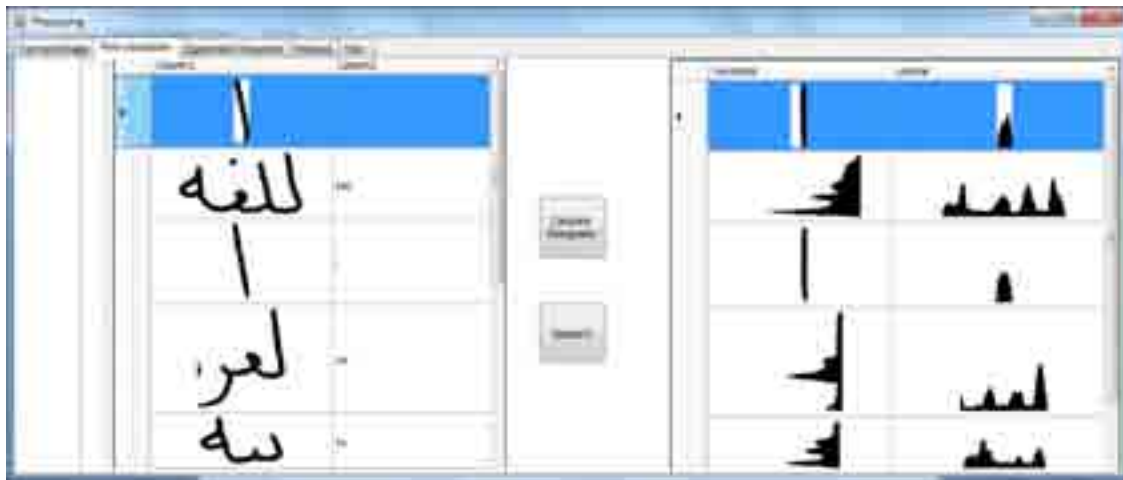


Figure 5.16: Horizontal and vertical histogram of PAWs

Baseline is useful to detect the line which the letters connect to each other, the base line can be detected by calculate the horizontal histogram as shown in figure 5.17.

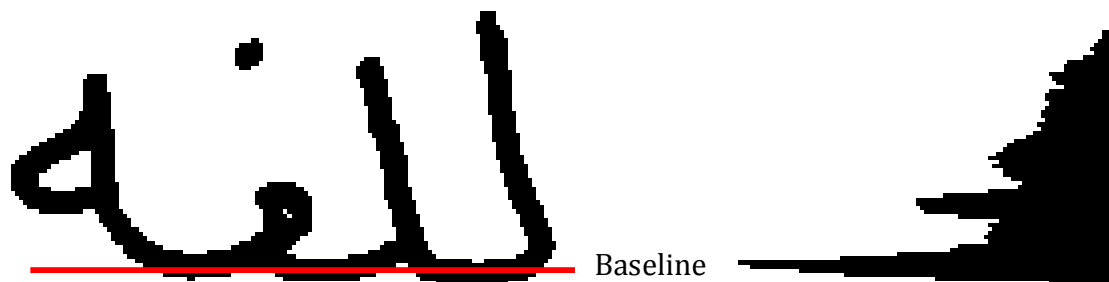


Figure 5.17: Detect the base line of Arabic word

The next process is the character segmentation when click the segment button the PAWs in the left side of the form segmented to characters by drawing a red line at the

point of segmentation as shown in figure 5.18, and shown individual characters in figure 5.19.

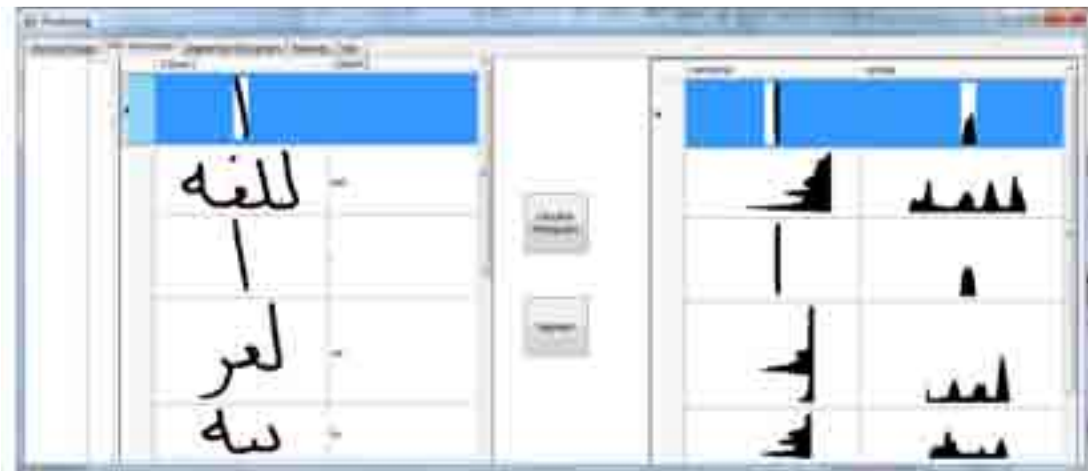


Figure 5.18: Segmentation process

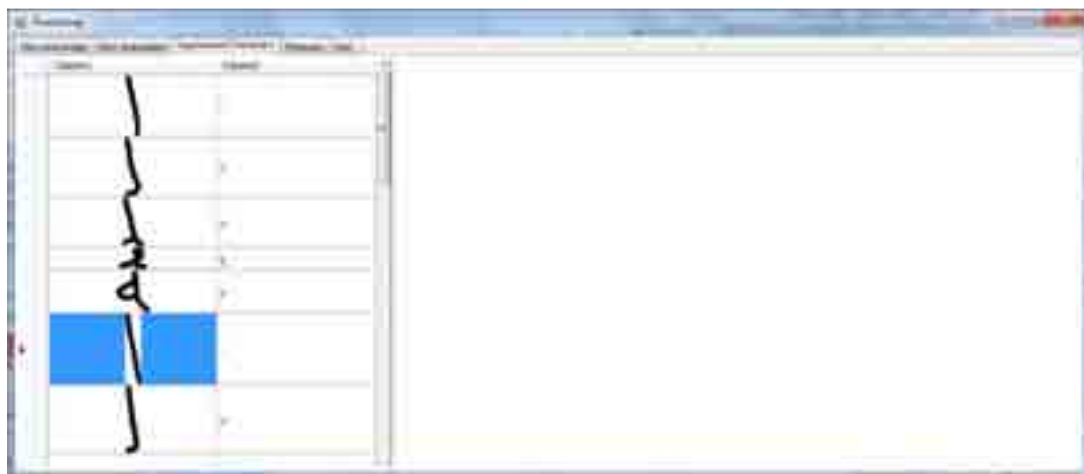


Figure 5.19: Segmented characters

After the segmentation process and the extraction features process for each character, the features for each character stored in database with the writer ID as shown in figure 5.20, now every character is associated to a writer in database which means that the characters are classified for each writer.

ID	DocID	DocLen	Page	Year	DocName	Page	Author	...	...	...	...
910	0	1	0	2000	11111111	1	11111111	...	...	...	...
911	0	1	0	2000	11111111	1	11111111	...	...	...	...
912	0	1	0	2000	11111111	1	11111111	...	...	...	...
913	0	1	0	2000	11111111	1	11111111	...	...	...	...
914	0	1	0	2000	11111111	1	11111111	...	...	...	...
915	0	1	0	2000	11111111	1	11111111	...	...	...	...
916	0	1	0	2000	11111111	1	11111111	...	...	...	...
917	0	1	0	2000	11111111	1	11111111	...	...	...	...
918	0	1	0	2000	11111111	1	11111111	...	...	...	...
919	0	1	0	2000	11111111	1	11111111	...	...	...	...
920	0	1	0	2000	11111111	1	11111111	...	...	...	...
921	0	1	0	2000	11111111	1	11111111	...	...	...	...
922	0	1	0	2000	11111111	1	11111111	...	...	...	...
923	0	1	0	2000	11111111	1	11111111	...	...	...	...
924	0	1	0	2000	11111111	1	11111111	...	...	...	...
925	0	1	0	2000	11111111	1	11111111	...	...	...	...
926	0	1	0	2000	11111111	1	11111111	...	...	...	...
927	0	1	0	2000	11111111	1	11111111	...	...	...	...
928	0	1	0	2000	11111111	1	11111111	...	...	...	...
929	0	1	0	2000	11111111	1	11111111	...	...	...	...
930	0	1	0	2000	11111111	1	11111111	...	...	...	...

Figure 5.20: calculate and store extracted Features

### 5.3.3 Discussion of results

The features of the characters are now known and each character related to its writer; in figure 5.20 the stage of testing shown a field to enter the writer ID and a field to enter the fragments, search process starts the search of fragments in the same writer characters forms database, there are two cases of search result:

### 5.3.3.1 : The fragments found in writer database:

First case, if the fragments are available in the database for the same writer the result will be fragments text written in the same style of writing for the same writer as shown in figure 5.21, the fragments “لغة” exists in its form in the original handwritten text as shown in figure 5.22.

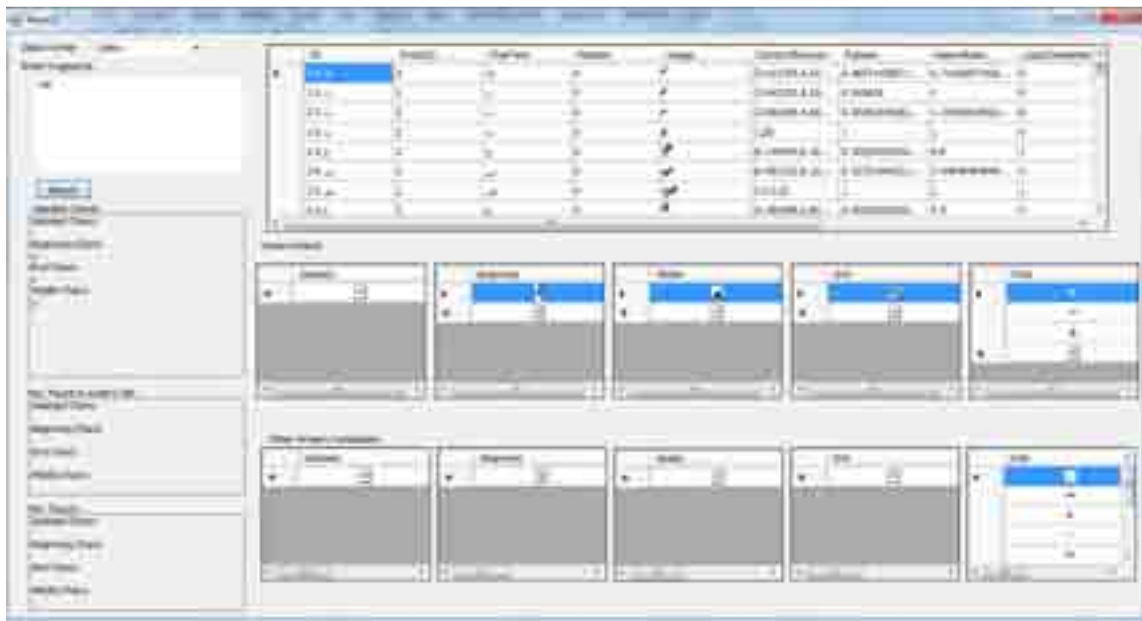


Figure 5.21: Test Form



Figure 5.22: Fragments in same writer style

### 5.3.3.2 : The fragments not found in writer database:

Second case, if the fragments not found in writer database, the search will be according to match the value of features and compare it with other characters features to other writers and the result will be a fragment's text written in a writing font style approximately similar to the original text in the features of characters, figure 5.23 an example to search the word “جامعة” which not all of its characters form are available in same writer database, but have been found in another writer database, the result shown in figure 5.24.

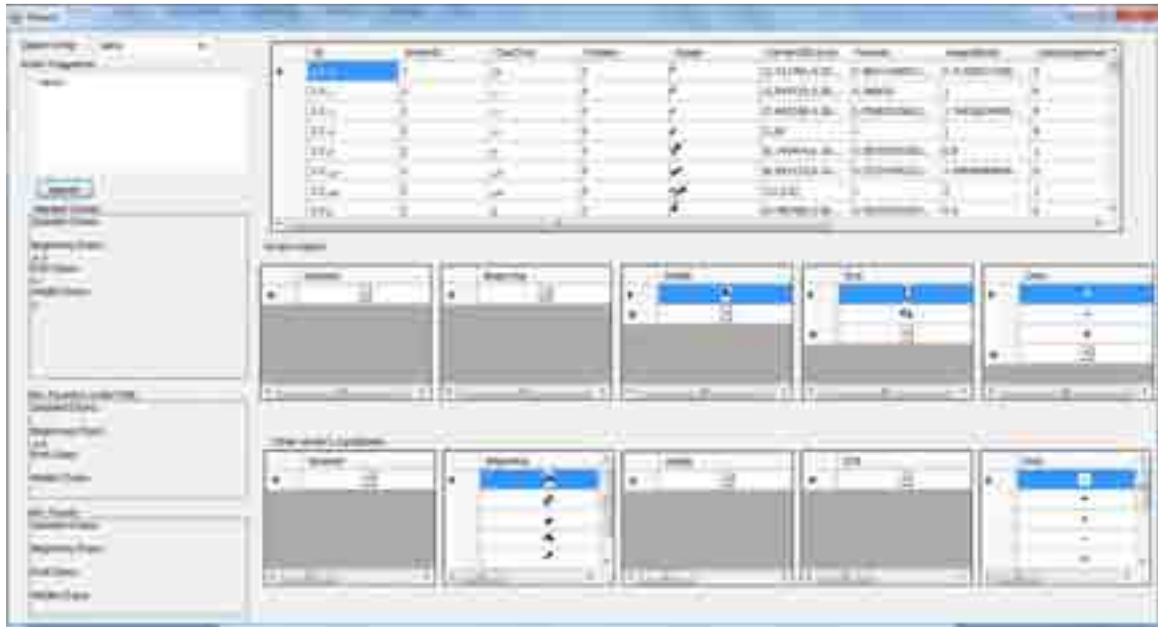


Figure 5.23: Fragments in other writer style

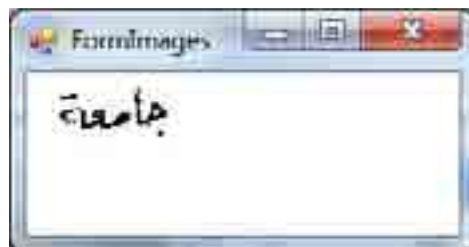


Figure 5.24: Fragments found in another writer database



In the test form the user choose a writer and enter the fragments in text field then press the search button as shown in figure 5.25.



Figure 5.25: fragments search

In test form the characters define as isolated, beginning, middle, and end form in writer database and in other writer's database as shown in figure 5.26 and figure 5.27.



Figure 5.26: characters form available in writer database.



Figure 5.27: characters form available in other writer's database.

## Chapter six

# Conclusion, Discussion and Future Work

### 6.1 Conclusion

Most of the existing application software cannot give a 100% of recognition, since there are variations of human handwriting, various shapes and size of writing, the overlapped and connected characters. This is a problem because there is no method segment the Arabic text 100%. The recognition process needs to be much efficient and accurate to recognize the characters written by different users. In this thesis, there are reasons that create the problem in Arabic handwriting recognition:

1. There are characters not clear in writing so it's hard to recognize it and it cause a problem with the handle fragments stage.
2. There are characters similar to another character in shape so it's hard to recognize.
3. Sometimes characters are overlapped, so it's difficult to segment.
4. The same user can write differently at different times.
5. The character can be written at different shape at the time and in different fonts.

Meanwhile in this thesis the recognition rate of PAWs was 98%, where the application has been tested on samples containing 200 PAWs where the error rate was 2%.

## **6.2 Discussion**

In this thesis the problem of overlapped characters appears in the segmentation process due to the difficulty of the similarity of Arabic language characters in the handwritten, if there are overlapped characters the segmentation process will not work well.

Also the problem of the connection of the characters to each other and the multiplicity of forms of characters According to its position in the word, so if the characters not written correctly that will cause a problem with the word reconstruction stage, the problem will appear in the work where the characters not connected correctly.

In the algorithm that designed to detect the PAWs, the features area, width, and height used to detect which is PAW and which is secondary component, when these feature values is smallest then there is secondary component will be detected as individual PAW, in the application this problem solved by allow the user to change these values to get appropriate detecting of PAWs.

## **6.3 Future Works**

1. The framework model can be implemented for recognition of other languages which are not complicated languages like Arabic language.
2. If there are missing characters shaped in the original document can predict the writing style of it without resorting to compensate the character from another document written by another writer.

## Appendix

### 1. Test data

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الاضمانية وترتيبها الرباعي هو

أ ب ت ث ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ق ك ل م ن  
ه و ي ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بنظم يعتمد على وهل

أو دمج الحروف بعضها لتكون مقاطع

ويظهر في اللغة العربية بعض الحركات الصوتية مربي جزء من

الابجدية العربية الاختيارية غير ما سبق مثل الصم والفتح

والكسر

هناك فطووم وأشكال متنوعة للكتابة من هذه الفطووم مثلاً

النسخ والرقعة والثلث

بلغ العدد الف حرفاً على هامش جائزة بيبس الشمس لتحييت

القرآن

### Test sample 1

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الاضفائية وترتيبها الهمجائي هو

أ ب ت ث ن ح خ ع ح ز ر ز س هـ  
 ص ض ط ظ ع غ ف ق ك ل م ن  
 هـ و ي د ذ ي لا

تكتب الحروف العربية من اليمين إلى اليسار بنظم يعتمد على وصل

أو دمج الحروف ببعضها لتكون مقاطع

ويظهر في اللغة العربية بعض الحركات الصوتية فهي مجرد من

الابجدية العربية الاختيارية فير ما سبق مثل الضم والفتح والكسر

هناك فطوالم وأشكال متنوعة للكتابة من هذه الطوالم مثل

النسخ والرقعة والثلث

بلغ العدد الف طرفي على هامش جاشرة بصيغ الشمس لتعطي القرآن

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الاضافية وترتيبها الراجحي هو

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين الى اليسار بخط يعتمد على ومهل  
أو دمج الحروف ببعضها لتكون مقاطع

ويظهر في اللغة العربية بعض الحركات المبتوتة وهي جزء من  
الابجدية العربية الاختيارية غير ما سبق مثل الضم والفتح والكسر

هنالك فلولم وأشكال متنوعة للكتابة من هذه الخطوط مثلا

النسخ والرقعة والثلث

بلغ العدد الف لفرق على هامش جائزة بصيص الشمس لتعظيم القرآن

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً ويعتبر الحروف

الإصغية وترتيبها الهمجائي هو

أ ب ت ث ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي ء ه ي لا

تكتب الحروف العربية من اليمين إلى اليسار بنظم يقصد على وجه

أو دمج الحروف بعضها لتكون متاهج

ويظهر من اللغة العربية بعض التراكيب الصوتية فهي مزج من

الابجدية العربية الاختيارية غير ما سبق مثل الضم والفتح والكسر

صنالك موهوم وأشكال متنوعة للكتابة من هذه الموهوم مثلاً

النسخ والرقعة والسند

يلغ العدد ألف مرة على هامش جائزة بعض الشهور لتخصي القرآن

تتضمن اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف  
الاضدادية وترتيبها اللفظي هو

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي س ة ي لا

تتكون الحروف العربية من الياءين إلى اليسار ينطق بقصد على وجه  
أو دمج الحروف بعضها لتكون مقاطع

ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من  
الإبجدية العربية اللفظية غير عاصقة مثل الضم والفتح والكسر  
هناك خطوط وأشكال متشعبة للكتابة من هذه الخطوط مثل  
النسخ والرقعة والشُّكُ

بلغ العدد الفعلي للحروف على هامش جائزة بيبس الشهر لتعريف القرآن



تُشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض  
الحروف الإضافية وترتيبها الهجائي هو:  
أ ب ت ث ج ح ح ذ ز ر ز س ش ص ض  
ط ظ ع غ ف ق ك ل م ن ه و ي ء  
ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بسط يعتمد على  
وصل أو دمج الحروف ببعضها لتكون مقاطع  
ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من  
الابجدية العربية الاختيارية غير ماضية مثل الضم والفتح والكسر  
صالك خطوط وأشكال متنوعة للكتابة من هذه الخطوط مثلاً:  
المسح والرقعة والثلث  
يلج العديد الفاطري على هامش جائزة بصيص الشمس لتحيط القرآن

تشتمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف  
الاصطناعية وثنائية الارجاء هي حروف

أ ب ت ث ج ح خ د ذ ر ز س ص ش  
ض ظ ط ق ك م ن  
ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين الى اليسار ويتم تقدم على اليمين  
او دمج الحروف بعضها لتكون مقاطع

ويظهر في اللغة العربية بعد الحركات الصوتية ضيق مزج من  
التجويد العربية الالهيانية غير ما سبقا مثل الضم والفتح والكسر  
هناك فطووم و أشكال متنوعة للكتابة من هذه الفطووم مثل  
المعج والرمعة والثلث

بلغ العدد الفاطووم على هامش جائزة يصير الشمس الفطووم القرآني

تشتمل اللغة العربية بغاية وعشرون حرفاً أساسياً وبعض الحروف  
الإضافية وتربيعها الهجائي هو:

أ ب ت ث ج ح خ د ذ ر ز س س ش  
ص ض ط ظ ع غ ف ق ك ل م ن

تكتب الحروف العربية من اليمين إلى اليسار بنقطة يعتمد على وهل

أو دمج الحروف ببعضها لتكون مقاطع .

ويظهر في اللغة العربية بعض الحركات الصوتية مهم جزء من

الأبجدية العربية الاختيارية غير ماسح مثل الضم والفتح والكسر.

هناك خلوص وأشكال متنوعة للكتابة من هذه الخلفاء مثلا:

النسخ والرتعة والثلث.

بلغ القدر الف حرف عن هاشم جائزة بعضهم الشمس لتخط القرآن.

تشكل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الإضافية وترتيبها الهجائي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع ع ف ق ك ل م ن  
ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار فقط يعتمد على ذلك  
أو دمج الحروف ببعضها لتكون مقاطع.

ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من  
الوحدة العربية الاختيارية غير ما سبق مثل الغم والفتح والكسر  
هناك خطوط وأشكال متنوعة للكتابة من هذه الخطوط مثلاً:

المنسخ و الرقعة والتلث

بلغ العدد الف حرف على هامش جائزته بعض النسخ لتجويد القرآن

تشكل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الاضافية وقرئها الهجائي هو:

أ ب ت ث ج ح د ذ ر ز س ش

هـ و ي ك ط ظ ع غ ف ق ي ل آل م ن

هـ و ي ك ط ظ ع غ ف ق ي ل آل

تكتب الحروف العربية من اليمين إلى اليسار وخط يعتمد على وصل

أو دمج الحروف ببعضها لتكونة للفاعل

ويظهر في اللغة العربية بعض الحركات الصوتية من جزر من

الأبجدية العربية الاختيارية غير ماسطة مثل التميم والتويريكسر

هذالك خطوط وأشكال متنوعة للكتابة من هذه الخطوط مثل:

النسخ والرقعة والثلث

بلغ العدد الف حرف على ما نحن جائرة ببعض الشئ لتفصيل القرآن

تسجل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الإضافية وترتيبها المعاني هو :

أ ب ت ث ج ح 22 د د ر ر س ش  
ص ض ط ظ ع ح ف ق ك ل م ن  
ه و ي ع ه ي لا

تكتب الحروف العربية من اليمين إلى اليسار بنقط معتدلة على وصل

أو دمج الحروف بعضهم لتكون مصاع

وتظهر في اللغة العربية بعض الحركات المبهمة التي لم تكن

في لغات أخرى العربية إلا صارت في العربية من أصلها في اللغة العربية

هناك خطوط وأشكال كثيرة للاكتفاء منها هذه الخطوط مثل

النسخ والرقعة والثلث

يلعب العرب العاقل على أوضاع حائرة يصعب إتقان إتقان القرآن

تُشكل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الإضافية وترتيبها الحجابي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش

ص ض ط ظ ع ف ق ك ل م ن

ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بحيث يعتمد على وصل

أو دمج الحروف ببعضها لتكون مقاطع.

ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من

الأبجدية العربية الاختيارية غير ماضية مثل الضم والفتح والكسر.

هناك خطوط وأشكال متنوعة للكتابة من هذه الخطوط مثلاً:

السح والرقعة والتلث

بلغ البلاد الف طرفي على هامس جائرة يصيغ الشمس لتحفيظ القرآن.

تشتمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف الإضافية وترتيبها الهجائي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش ص ض ط ظ  
ع ف ق ك ل م ن ه و ي آ ة إ ي ح

تكتب الحروف العربية من اليمين إلى اليسار بخط مستدعاً  
وعلى أوجه الحروف بعضها لتكون مقاطع.

ويظهر في اللغة العربية بعض الحركات الصوتية فهي جزئية  
التي تجديء العربية الاختيارية غير ما سبق مثل الضم والفتح والكسر.

هنالك ملحوظ وأشكال متنوعة للكتابة من هذه الملحوظات:

الفتح والرقعة والثلاث

بلغ العدد الف الحرف على هامش جائزه لبعض العلماء لتخيل القرآن



تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف  
الاضافية وترتيبها الطباعي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي ة ي ل

تلت الحروف العربية من اليمين إلى اليسار فلو تعلّم على وصل  
أو وضع الحروف بعضها لتكون مقاطع .

ويظهر في اللغة العربية بعض المراتب الصوتية فهي جزء من  
الأبجدية العربية الاقترابية غير ما سبق مثل الضم والفتح والهمزة  
هناك خطوط وأسلاك متنوعة للكتابة من هذه الخطوط مثل:  
النسخ والرفعة والتلخيص .

بلغ العدد الف حرف على هامش جائزة وصي السحر فخطبة القرآن .

تشمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف  
الامتدادية وترتيبها الهجائي هو :

أ ب ت ث ج ح د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بخط يعتمد  
على وسط أو دمج الحروف ببعضها لتكون مقاطع

و يظهر في اللغة العربية بعض الحركات الصوتية فهي جزء من  
الأصوات العربية الاختيارية غير ما سيف مثل الهم والنخ والكسر  
هذه الحركات وأشكال متنوعة للكتابة من هذه الحركات مثلا :

التنخيد والرفعة والتثنية.

بلغ العدد الف حرف على هامش جائزة بيبي الشمس لتحفيز القرآن

تعمل اللغة العربية تمايزاً ومتميزاً حتماً أساسياً وهذه الحروف  
الاضاعية وتربطها التوافقية هـ ا.

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي م ع ي لا

تكتب الحروف العربية من اليمين إلى اليسار فيتميز مشترك على  
أوضح الحروف بعضها لتكون مقالع .

ويظهر في اللغة العربية بعض الحركات الصوتية فهي ح من  
الأجديده العربية الاستيعابية غير ما سبق مثل الضم والفتح والكسر  
هـ ذلك شروطاً وأشكالاً متنوعة الكتابية من هذه الحروف مثل:  
الفتح والرقعة والتلث

يلج العرب إلى طرق على هامش جانبه بصرف عن التمسك بالحرف التقليدي

تشتمل اللغة العربية ثمانية وعشرون حرفاً أساسياً وبعض الحروف

الإضافية وترتيبها الهجائي هو:

أ ب ت ث ج ح خ د ذ ر ز س ش  
ص ض ط ظ ع غ ف ق ك ل م ن  
ه و ي ء ة ي لا

تكتب الحروف العربية من اليمين إلى اليسار بخط يعقد على وصل

أو دمج الحروف بعضها لتكون مقاطع

ويظهر في اللغة العربية بعض الحركات الصوتية وهي حركات من  
الذكورية العربية التقليدية غير ما سبق مثل الصم والفتح والأمر

كصالح خطه وأمثال شوه للكتابة بهذه الخطوط مثلا:

النسخ و الرقة والثلث

بلغ عدد الفنون التي جازتها بعض الشعوب لتحرير القرآن

## 2. Programming code in C#.net

### // load image

```
private void toolStripOpenImage_Click ( object sender , EventArgs e )
    {
        if ( openFileDialog.ShowDialog ( ) == DialogResult.OK )
            {
                TextImage = new Bitmap ( openFileDialog.FileName );
                pictureBox1.Image = TextImage;

                //grayImage = new ImagingClasses.Image ( TextImage );
                status = PreProcessingStatus.RawImage;
                Log = "Image Loaded";

                ShowImage ( );
                Log = "Image Shown";
            }
        btnGrayLevel.Enabled = true;
        btnNormalize.Enabled = true;
    }
```

### // Resize Image

```
private void btnNormalize_Click ( object sender , EventArgs e )
    {
        if ( TextImage.Height > 1200 )
            TextImage = PixelImage.Normalize1200 ( TextImage );
        ShowImage ( );
        btnGrayLevel.Enabled = true;
    }

public static Bitmap Normalize1200 ( Bitmap image )
    {
        Rectangle rect= new Rectangle ( new Point ( 0 , 0 ) ,
        image.Size );
        int maxRectHeight= rect.Height;
        double multiplier=( double ) 1200 / ( double )
maxRectHeight;
```

```

        int newWidth=( int ) ( image.Width * multiplier );
        int newHeight=( int ) ( image.Height * multiplier );
        AForge.Imaging.Filters.ResizeBilinear resize=new
AForge.Imaging.Filters.ResizeBilinear ( newWidth ,
newHeight );
        return resize.Apply ( image );
    }

```

### **// Convert image to gray level**

```

private void btnGrayLevel_Click ( object sender , EventArgs e )
{
    Log = "Converting to Gray ";
    TextImage = ApplyGrayLevelFilter ( TextImage );
    Log = "Conversion Complete..";
    status = PreProcessingStatus.GrayImage;
    ShowImage ( );
    btnGrayLevel.Enabled = false;
    btnHistogram.Enabled = true;
}

```

```

private Bitmap ApplyGrayLevelFilter ( Bitmap img )
{
    return
AForge.Imaging.Filters.Grayscale.CommonAlgorithms.BT709.Apply (
img );
}

```

### **// Calculate histogram**

```

private void btnHistogram_Click ( object sender , EventArgs e )
{
    AForge.Imaging.ImageStatistics st= new
AForge.Imaging.ImageStatistics ( TextImage );
AForge.Math.Histogram his=new AForge.Math.Histogram (
st.Gray.Values.Select ( v => ( int ) Math.Log10 ( v + 1 )
).ToArray ( ) );
GrayImageHistogram h= new GrayImageHistogram ( st.Gray.Values ,
TextImage.Width , TextImage.Height );
pictureBoxHistogram.SizeMode = PictureBoxSizeMode.AutoSize;
pictureBoxHistogram.Image = h.DrawHistogram ( );
    btnDigitize.Enabled = true;
}

```

```

        histogram1.Values = his.Values;
        trackBarThreshold.Value = threshold = h.AutoThreshold + 25;

    }
// Digitizing or binarization the image

private void btnDigitize_Click ( object sender , EventArgs e )
    {
        Log = "Digitizing..";
        ShownImage = DigitizeImage ( TextImage ).ShowImage;
        status = PreProcessingStatus.DigitizedImage;
        Log = "Digitizing Completed";
        btnConnectedComponents.Enabled = true;
    }

private PixelImage DigitizeImage ( Bitmap img )
    {

        DigitizedImage = new PixelImage ( img , threshold );

        return DigitizedImage;
    }

public PixelImage ( Bitmap grayImage , int threshold = 127 )
    {
        Width = grayImage.Width;
        Height = grayImage.Height;
        Pixels = PixelsFromImage ( grayImage , threshold );
        ZoomValue = 1;
    }

public static Pixel [ , ] PixelsFromImage ( Bitmap grayImage ,
int threshold = 127 )
    {
        int Width = grayImage.Width;
        int Height = grayImage.Height;
        Pixel[,] Pixels = new Pixel [ Width , Height ];
        BitmapData data=grayImage.LockBits ( new Rectangle (
0 , 0 , Width , Height ) , ImageLockMode.ReadOnly ,
grayImage.PixelFormat );

        try
        {
            AForge.Imaging.UnmanagedImage unmangedGrayImg=new
AForge.Imaging.UnmanagedImage ( data );

```

```

        for ( int x=0 ; x < Width ; x++ )
        {
            for ( int y=0 ; y < Height ; y++ )
            {
                if ( unmangedGrayImg.GetPixel ( x , y ).R > threshold )
                {
                    Pixels [ x , y ] = new WhitePixel ( x , y );
                }
                else
                {
                    Pixels [ x , y ] = new BlackPixel ( x , y );
                }
            }
        }
    }
    finally
    {
        grayImage.UnlockBits ( data );
    }
    return Pixels;
}

```

### **// connected components labeling**

```

private void btnConnectedComponents_Click ( object sender ,
EventArgs e )
{
    ShownImage = ApplyConnectedCompnentAlgorithm ( DigitizedImage );
    Rectangle[] smallObjectRects=
    DigitizedImage.SmallObjectsCandidatesRects.OrderBy( r =>
    r.Height*r.Width).ToArray();
    FormSmallObjects frm=new
    FormSmallObjects(smallObjectRects,DigitizedImage.ShowImage,Writer
    );
        frm.ShowDialog();
        btnNext.Enabled = true;
    }

private Bitmap ApplyConnectedCompnentAlgorithm (PixelImege pimge)
{
    decimal[] thresholds=new decimal [ 3 ];
    thresholds [ 0 ] = nudArea.Value; thresholds [ 1 ] =
    nudHeight.Value; thresholds [ 2 ] = nudWidth.Value;
}

```



```

        status = PreProcessingStatus.connectedComponent;
        return pimg.ConnectedComponents ( thresholds );
    }

public Bitmap ConnectedComponents ( decimal [ ] thresholds )
    {
        AForge.Imaging.Filters.ConnectedComponentsLabeling
filter=new AForge.Imaging.Filters.ConnectedComponentsLabeling ();
        Bitmap image=filter.Apply ( ShowInversedImage );
        BlobRectangles =
filter.BlobCounter.GetObjectsRectangles ( );
        minRectArea = BlobRectangles.Min ( r => r.Height * r.Width );
        minRectHeight = BlobRectangles.Min ( r => r.Height );
        minRectWidth = BlobRectangles.Min ( r => r.Width );
        maxRectArea = BlobRectangles.Max ( r => r.Height * r.Width );
        medianArea = BlobRectangles.OrderBy ( r => r.Area ( ) ).Skip (
        BlobRectangles.Length / 2 - 1 ).ToArray ( ) [ 0 ].Area ( );
        medianHeight = BlobRectangles.OrderBy ( r => r.Height ).Skip (
        BlobRectangles.Length / 2 - 1 ).ToArray ( ) [ 0 ].Height;
        medianWidth = BlobRectangles.OrderBy ( r => r.Width ).Skip (
        BlobRectangles.Length / 2 - 1 ).ToArray ( ) [ 0 ].Width;
        maxRectHeight = BlobRectangles.Max ( r => r.Width );
        maxRectWidth = BlobRectangles.Max ( r => r.Width );
        threshodArea = ( int ) ( ( minRectArea + maxRectArea ) *
        thresholds [ 0 ] / 100 );
        thrsholdHeight = ( int ) ( ( minRectHeight + maxRectHeight ) *
        thresholds [ 1 ] / 100 );
        thrsholdWidth = ( int ) ( ( minRectWidth + maxRectWidth ) *
        thresholds [ 2 ] / 100 );
        FuzzyThreshold.ThreshodArea = new FuzzyThreshold ( threshodArea ,
        minRectArea );
        FuzzyThreshold.ThreshodHeight = new FuzzyThreshold (
        thrsholdHeight , minRectHeight );
        FuzzyThreshold.ThreshodWidth = new FuzzyThreshold ( thrsholdWidth
        , minRectWidth );
        Rectangle[] mainObjects=BlobRectangles.Where ( r => r.IsMain ( )
        ).ToArray ( );
        Rectangle[] smallObjectsRects = BlobRectangles.Except (
        mainObjects ).ToArray ( );// ( r => r.Width * r.Height <
        threshodArea ).ToArray ( );
        Rectangle[] midObjects=smallObjectsRects.Where ( r => r.IsMid ( )
        ).ToArray ( );
        smallObjectsRects = smallObjectsRects.Except ( midObjects
        ).ToArray ( );
        AForge.Imaging.Blob[] blobs= filter.BlobCounter.GetObjects (
        image , false );
    }

```

```

Bitmap newImage=new Bitmap ( Width , Height );
Lines = Line.ExtractImageLines ( filter , image , thresholds );
Rectangle[] lineRects=Lines.Select (l => l.MainRect).ToArray ();
Rectangle [] baseLineRects=Lines.Select ( l => l.CenterLine
).ToArray ( );
    foreach ( Rectangle rect in BlobRectangles )
    {
        rect.Inflate ( new Size ( 5 , 5 ) );
    }
    using ( Graphics g =Graphics.FromImage ( newImage ) )
    {
        g.Clear ( Color.White );
        foreach ( AForge.Imaging.Blob blob in blobs )
        {
            g.DrawImageUnscaled ( ReverseBackGround ( blob.Image ) ,
            blob.Rectangle.Location );
        }
        Pen penLine=new Pen ( Color.Green , 1 );
        Pen penMain=new Pen ( Color.DarkBlue , 1 );
        Pen penMid=new Pen ( Color.Red , 1 );
        penMid.DashPattern = new float [ ] { 1 , 1 };
        if ( mainObjects.Length != 0 )
            g.DrawRectangles ( penMain , mainObjects );
        if ( smallObjectsRects.Length != 0 )
            g.DrawRectangles ( Pens.Yellow , smallObjectsRects );
        if ( midObjects.Length != 0 )
            g.DrawRectangles ( penMid , midObjects );
        if ( lineRects.Length != 0 )
            g.DrawRectangles ( penLine , lineRects );
        if ( baseLineRects.Length != 0 )
            g.DrawRectangles ( Pens.Blue , baseLineRects );
        penLine.Dispose ( );
        penMain.Dispose ( );
        penMid.Dispose ( );

    }
    SmallObjectsCandidatesRects=new
List<Rectangle>(smallObjectsRects);
    SmallObjectsCandidatesRects.AddRange(midObjects);
    return newImage;
}

public Bitmap ConnectedComponents ( )
{
    // create filter

```

```

        BlobsFiltering filter = new BlobsFiltering ( );
        // configure filter
        filter.CoupledSizeFiltering = true;
        filter.MinWidth = Width;
        filter.MinHeight = Height;
        // apply the filter
        try
        {
            return filter.Apply ( ShowInversedImage );
        }
    }
}
// associate image and text

private void btnProcess_Click ( object sender , EventArgs e )
{
    dataGridView1.Rows.Clear ( );
    InsertPAWImagesToGrid ( );
    ExtractPawsFromText ( );
    tabControl1.SelectedIndex = 1;
}

private void InsertPAWImagesToGrid ( )
{
    int linesCount=0;
    foreach ( Line line in
        Lines/*.OrderByDescending(l=>l.MainRect.Y)*/)
    {
        linesCount++;
        int maxWidth=40;
        int sequence=0;
        foreach ( Rectangle rect in
            line.MainLineRects.OrderByDescending ( r => r.X ) )
        {
            ImagePAWCount++;
            sequence++;
            DataGridViewRow row= new DataGridViewRow ( );
            DataGridViewImageCell imageCell=new DataGridViewImageCell ( );
            DataGridViewTextBoxCell lineCell=new DataGridViewTextBoxCell ( );
            DataGridViewTextBoxCell sequencCell=new DataGridViewTextBoxCell
            ( );
            DataGridViewTextBoxCell txtCell=new DataGridViewTextBoxCell ( );
            DataGridViewTextBoxCell LTRBCell=new DataGridViewTextBoxCell ( );
            imageCell.Value = GetPartOfImage ( TextImage.ShowImage , rect );
            txtCell.Value = "";
            lineCell.Value = linesCount;
            sequencCell.Value = sequence;
        }
    }
}

```

```

row.Height = rect.Height + 10;
maxWidth = ( maxWidth > rect.Width ) ? maxWidth : rect.Width;
row.Cells.Add ( imageCell ); row.Cells.Add ( txtCell );
row.Cells.Add ( lineCell ); row.Cells.Add ( sequencCell );
StringBuilder sb=new StringBuilder ( );
sb.AppendFormat ( "{0},{1},{2},{3}" , rect.Left , rect.Top ,
rect.Right , rect.Bottom );
        LTRBCell.Value = sb.ToString ( );
        row.Cells.Add ( LTRBCell );
        dataGridView1.Rows.Add ( row );
    }
}

private void ExtractPawsFromText ( )
{
ArabicPAW[] paws= ArabicPAW.ExtractPAWs ( textBox1.Text );
TextPAWCount = paws.Length;
if ( paws.Length <= dataGridView1.Rows.Count )
{
    //text Paws are less than or equal to image paws
    for ( int i = 0 ; i < paws.Length ; i++ )
    {
        dataGridView1.Rows [ i ].Cells [ 1 ].Value = paws [ i ].Text;
    }
}
else
//we have more text paws than image paws
{
    int rowCount=dataGridView1.Rows.Count;
    for ( int i = 0 ; i < rowCount ; i++ )
    {
        dataGridView1.Rows [ i ].Cells [ 1 ].Value = paws [ i ].Text;
    }
    for ( int i = rowCount ; i < paws.Length ; i++ )
    {
        DataGridViewRow row= new DataGridViewRow ( );
        DataGridViewImageCell imCell=new DataGridViewImageCell ( );
        DataGridViewTextBoxCell txtCell=new DataGridViewTextBoxCell ( );
        row.Cells.Add ( imCell );
        txtCell.Value = paws [ i ].Text;
        row.Cells.Add ( txtCell );
        dataGridView1.Rows.Add ( row );
    }
}
}
}

```

**// show PAWs in picture box to be ready for segmentation process**

```

private void dataGridView4_CellDoubleClick ( object sender ,
DataGridViewCellEventArgs e )
    {
        InsertIntoGridView1 ( e.RowIndex );
    }
    List<PAW> PAWs=new List<PAW> ( );
    List<PictureBox> pbs=new List<PictureBox> ( );
Dictionary<PictureBox,PAW> pp=new Dictionary<PictureBox, PAW>();
private void btnPawProcess_Click ( object sender , EventArgs e )
    {
        int j=0, j1=0;
        for (int pci=0 ; pci < panelPAWImages.Controls.Count ; pci++)
            {
                panelPAWImages.Controls.RemoveAt ( pci );
            }
        for ( int i = 0 ; i < dataGridView1.Rows.Count - 1 ; i++ )
            {
                DataGridViewRow item =dataGridView1.Rows [ i ];
                if ( item.Cells [ 0 ] != null && item.Cells [ 1 ]
                != null && item.Cells [ 2 ] != null && item.Cells [ 3 ]
                != null && item.Cells [ 4 ] != null &&
                !string.IsNullOrEmpty ( item.Cells [ 1
                ].Value.ToString ( ) ) )
                    {
                        PAW newPaw=new PAW ( item );
                        PAWs.Add ( newPaw );
                        PictureBox pb=new PictureBox ( );
                        pb.Size = new System.Drawing.Size ( 110 , 110 );
                        if ( j > 5 ) { j = 0; j1++; }
                        pb.Location = new Point ( 110 * j++ , 110 * j1 );
                        pb.BorderStyle = BorderStyle.Fixed3D;
                        pb.SizeMode = PictureBoxSizeMode.Normal;
                        panelPAWImages.Controls.Add ( pb );
                        pb.Click += pb_Click;
                        pb.MouseMove += pictureBoxMouseMove;
                        pb.Image = newPaw.GetInfo ( );
                        pp.Add ( pb , newPaw );
                    }
            }
        }
        tabControl1.SelectedIndex = 2;
    }

```

```

public PictureBox CurrentPictureBox { get; set; }
public Point CurrentPoint=new Point ( 0 , 0 );
void pb_Click ( object sender , EventArgs e )
    {
        CurrentPictureBox = ( ( PictureBox ) sender );
        PAW currentPAW=pp [ CurrentPictureBox ];
        pictureBox1.Image = currentPAW.RawImage;
        currentPAW.ReplaceSegmentationPoint (CurrentPoint.X);
        CurrentPictureBox.Image = currentPAW.GetInfo ( );
    }

private void pictureBoxMouseMove ( object sender , MouseEventArgs
e )
    {
        PictureBox CurrentPictureBox= ( ( PictureBox ) sender );
        if ( CurrentPictureBox.Image == null ) return;
        PAW currentPAW=pp [ CurrentPictureBox ];
        System.Drawing.Point
        mouseLocation=CurrentPictureBox.PointToClient (
        Control.MousePosition );
//the point should be less than pb and panel//////////
        Rectangle
        pbScreenCords=CurrentPictureBox.RectangleToScreen (
        new Rectangle ( new System.Drawing.Point ( 0 , 0 ) ,
        pictureBox1.Size ) );
        Rectangle
        panellScreenCords=panelPAWImages.RectangleToScreen ( new
        Rectangle ( new System.Drawing.Point ( 0 , 0 ) , panel1.Size ) );
//pbScreenCords.Intersect ( panellScreenCords );
        Rectangle clippingRect=CurrentPictureBox.RectangleToClient (
        pbScreenCords );
        if ( clippingRect.Contains ( mouseLocation ) )
            {
                int x=mouseLocation.X; int y=mouseLocation.Y;
                CurrentPoint.X = x; CurrentPoint.Y = y;
                lblX.Text = x.ToString ( ); lblY.Text = y.ToString ( );
                pictureBox1.Image = currentPAW.GetInfo ( x , y );
            }
    }
}

```

**// segmentation process**

```

private void btnSegment_Click ( object sender , EventArgs e )
{
    Character.AddDataGridColumns ( dgvDB );
    Features.AddDataGridColumns ( dgvDB );
    dgvDB.Rows.Clear ( );
    int writerID=Writer.ID;
using ( Database1Entities db=new Database1Entities ( ) )
    {
        foreach ( PAW paw in PAWs )
        {
            if ( paw.PAWText.Contains ( "ﻻ" ) )
                continue;
            charSegment[] segments= paw.Segment1 ( );
            int i=-1;
            for ( int j=0 ; j < segments.Length ; j++ )
            {
                charSegment c = segments [ j ];
                if ( c == null ) continue;
                i = ( segments.Length == 1 ) ? -1 : i + 1;
                if ( segments.Length == 1 )
                {
                    i = -1;//single char
                }
                else if ( j == segments.Length - 1 )
                {
                    i = -2;//end char
                }
                else
                {
                    i = j;//first=0 , positive =>mid char
                }
            }
            ArabicChar ac=ArabicChar.CreateInstance ( c.CharText [ 0 ] );
            Features newFeatures=new Features (ac, i, c.CharImage , Writer);
            Character ch=new Character (c.CharText, i, Writer.ID, newFeatures
            , c.CharImage );
            CharImage newEntry=new CharImage ( c );
            DataGridViewRow row1=ch.GetDataGridRow ( );
            dgvDB.Rows.Add ( row1 );
            newEntry.WriterID = ( int ) numericUpDown1.Value;
            db.CharImages.Add ( newEntry );
            DataGridViewRow row=newEntry.GetDataGridRow ( );
            row.Cells [ 1 ].Value = c.CharImage;
            dataGridView5.Rows.Add ( row );
        }
    }
}

```

```

ch.Save ( );
}
}
db.SaveChanges ( );
}
tabControl1.SelectedIndex += 2; ;
}

public charSegment [ ] Segment1 ( )
{
    charSegment[] cs=new charSegment [ NumberOfChares ];
    int lenght=SegmentationPoints.Length;
    switch ( NumberOfChares )
    {
        case 1:
            return new charSegment [ ] { new charSegment (
this.RawImage , this.PAWText , this.Rect ) };
        default:
            SegmentationPoints [ lenght - 1 ] = (
RawImage.Width - SegmentationPoints [ lenght - 1
] < 4 ) ? SegmentationPoints [ lenght - 1 ] - 4 :
SegmentationPoints [ lenght - 1 ];
            SegmentationPoints [ 0 ] = ( SegmentationPoints [
0 ] < 4 ) ? 5 : SegmentationPoints [ 0 ];
            for ( int i = 0 ; i < lenght + 1 ; i++ )
            {
                int startp=( i == 0 ) ? 0 :
SegmentationPoints [ i - 1 ];
                int endp=( i == lenght ) ?
RawImage.Width : SegmentationPoints [ i ];

                try
                {
                    Bitmap img=RawImage.GetPartOfImage ( startp , endp );
                    string chartxt=PAWText [ PAWText.Length - i - 1 ].ToString ( );
                    cs [ i ] = new charSegment ( img , chartxt , Rect.GetPartOfRect (
startp , endp ) );
                }
                catch ( Exception )
                {
                }
            }
            return cs.Reverse ( ).ToArray ( );
    }
}
}
}

```



```

public void Segment ( )
    {
        if ( NumberOfChares == 1 ) return;
        if ( NumberOfChares == 2 )
            {
                FindOneSegmentationPoint ( );
            }
        else
            {
                FindSegmentationPoints ( );
            }
    }

public void FindOneSegmentationPoint ( )
    {
        List<int> minIndecis=new List<int> ( );
        int index=0;
        int midPoint=Histograms.vArray.Length / 2;
        int skipLenght=Histograms.vArray.Length / 5;
        int takeLenght=skipLenght * 3;
        int minimum=Histograms.vArray.Skip ( skipLenght ).Min ( );
        for ( int x = skipLenght ; x < takeLenght ; x++ )
            {
                if ( Histograms.vArray [ x ] == minimum )
                    {
                        int lp=FindeLowstPointAtBaseline ( x );
                        if ( lp != 0 )
                            {
                                minIndecis.Add ( x );
                            }
                    }
                minIndecis.Sort ( );
                if ( minIndecis.Count > 0 )
                    {
                        index = minIndecis [ minIndecis.Count / 2 ];
                    }
            }
        if ( index == 0 )
            index = midPoint;
        SegmentationPoints [ 0 ] = index;
    }

public void FindSegmentationPoints ( )
    {
        int[] hist=Histograms.vArray;
        int max=hist.Max ( );
        int minthresold=max / 5;
        int averageCharLenght=hist.Length / NumberOfChares;
    }

```

```

int[] minimas=new int [ NumberOfSegmentationPoints ];
int[] indices=new int [ NumberOfSegmentationPoints ];

minimas.Initialize ( );
for ( int i = 0 ; i < minimas.Length ; i++ )
    {
        int intervalBegin=i * ( int ) ( averageCharLenght
* .5 ) + 1;
        int intervalEnd=( i + 1 ) * ( int ) (
averageCharLenght * 1.5 );
        minimas [ i ] = hist.Skip ( intervalBegin ).Take
( intervalEnd ).Min ( );
        indices [ i ] = intervalBegin;
        for ( int x = intervalBegin ; x < intervalEnd &&
x < hist.Length ; x++ )
            {
                if ( hist [ x ] < minimas [ i ] + minthresold )
                    {
                        if ( FindeLowstPointAtBaseLine ( x ) != 0 )
                            {
                                indices [ i ] = x;
                                if ( x == 0 ) indices [ i ] = intervalBegin;
                            }
                    }
            }
        SegmentationPoints [ i ] = indices [ i ];
    }
}

public bool IsAtBaseLine ( int x )
    {
        for ( int y = BaseLine.Top ; y < BaseLine.Bottom ; y++ )
            {
                if ( PAWPixeles.Pixels [ x , y ].IsBlack ) return true;
            }
        return false;
    }

public int FindeLowstPointAt ( int x )
    {
        for ( int y = PAWPixeles.Pixels.GetUpperBound ( 1
) ; y > 0 ; y-- )
            {
                if ( PAWPixeles.Pixels [ x , y ].IsBlack )
                    return y;
            }
        return 0;
    }
}

```

```

        public int FindLowstPointAtBaseLine ( int x )
        {
            for ( int y = BaseLine.Bottom - 1 ; y > BaseLine.Top ; y-- )
            {
                if ( x > PAWPixeles.Pixels.GetLowerBound ( 0 ) &&
                    x < PAWPixeles.Pixels.GetUpperBound ( 0 ) &&
                    y > PAWPixeles.Pixels.GetLowerBound ( 1 ) &&
                    y < PAWPixeles.Pixels.GetUpperBound ( 1 ) )
                {
                    if ( PAWPixeles.Pixels [ x , y ].IsBlack ) return y;
                }
                else
                {
                    return ( BaseLine.Bottom + BaseLine.Top ) / 2;
                }
            }
            return 0;
        }

        public void ReplaceSegmentationPoint ( int x )
        {
            if ( NumberOfSegmentationPoints == 0 ) return;
            int closesteSegmentationPointIndex=0;
            int mindistance=int.MaxValue;
            for ( int i = 0 ; i < SegmentationPoints.Length ; i++ )
            {
                int distance=Math.Abs ( x - SegmentationPoints [ i ] );
                if ( distance < mindistance )
                {
                    closesteSegmentationPointIndex = i;
                    mindistance = distance;
                }
            }
            SegmentationPoints [ closesteSegmentationPointIndex ] = x;
        }

```

### **// cleaning the characters images**

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Text;

```

```

using System.Threading.Tasks;
using System.Windows.Forms;
using TextImaging;

namespace DatabaseCleaning
{

public partial class Character
    {
        public Bitmap ModifiedImage { get; set; }
        public Bitmap OriginalImage { get; set; }
        public Character ( )
            {
            }
        public bool Clean1 ( )
            {
                OriginalImage = Image.ImageFromBytes ( );
                Bitmap tempImage= OriginalImage.AddWhiteFrame ( 2 );
                ModifiedImage = tempImage.InversedDigitizedImage ( );
                AForge.Imaging.RecursiveBlobCounter bc=new
                AForge.Imaging.RecursiveBlobCounter ( );
                bc.ObjectsOrder = AForge.Imaging.ObjectsOrder.Area;
                bc.ProcessImage ( tempImage );
                AForge.Imaging.Blob[] blobs= bc.GetObjects (
                tempImage , true );
                ModifiedImage = blobs [ 0 ].Image.ToManagedImage ( );
                return true;
            }

public bool Clean ( )
    {
        bool edited=false;
        OriginalImage = Image.ImageFromBytes ( );
        Bitmap tempImage= OriginalImage.AddWhiteFrame ( 2 );
        ModifiedImage = new Bitmap ( tempImage );
        tempImage = tempImage.InversedDigitizedImage ( );
        AForge.Imaging.Filters.Closing cf= new
        AForge.Imaging.Filters.Closing ( );
        cf.ApplyInPlace ( tempImage );
        AForge.Imaging.Filters.ConnectedComponentsLabeling
        bc=new
        AForge.Imaging.Filters.ConnectedComponentsLabeling (
        );
        bc.Apply ( tempImage );
    }
}

```

```

        Rectangle[] rects= bc.BlobCounter.GetObjectsRectangles
        ( ).OrderBy ( r => r.Width * r.Height ).ToArray ( );
        for ( int i = 0 ; i < rects.Length - 1 ; i++ )
            {
ModifiedImage = ModifiedImage.ClearPartOfImage ( rects [ i ] );
edited = true;
            }

        ModifiedImage = ModifiedImage.RemoveFrame ( 2 );
        return edited;
    }

public static void AddCharacterColumns ( DataGridView dgv )
    {
        dgv.Columns.Add ( "clmCharacter" , "Character" );
        dgv.Columns.Add ( "clmImg1" , "original Image" );
        dgv.Columns.Add ( "clmImg2" , "Proposed Image" );
        dgv.Columns.Add ( "clmImg1" , "Accept" );
        dgv.Columns.Add ( "clmID" , "ID" );
        dgv.Columns.Add ( "clmRemove" , " " );
    }

public DataGridViewRow GetCharacterRow ( )
    {
        DataGridViewTextBoxCell IdCell=new DataGridViewTextBoxCell ( );
        DataGridViewTextBoxCell charCell=new DataGridViewTextBoxCell ( );
        DataGridViewImageCell imCell=new DataGridViewImageCell ( );
        DataGridViewImageCell imCell2=new DataGridViewImageCell ( );
        DataGridViewCheckBoxCell chkCell=new DataGridViewCheckBoxCell ( );
        DataGridViewButtonCell btnRemoveCell=new DataGridViewButtonCell
        ( );

        IdCell.Value = ID;
        charCell.Value = CharText;
        imCell.Value = OriginalImage;
        imCell2.Value = ModifiedImage;
        chkCell.Value = true;
        btnRemoveCell.Value="Remove";
        DataGridViewRow row =new DataGridViewRow ( );
        row.Cells.Add ( charCell );
        row.Cells.Add ( imCell );
        row.Cells.Add ( imCell2 );
        row.Cells.Add ( chkCell );
        row.Cells.Add ( IdCell );
        row.Cells.Add(btnRemoveCell);
        return row;
    }

```

```

    }
}

```

**// search process for all fragment characters form in the current writer  
and in other writers database**

```

private void btnSearch_Click ( object sender , EventArgs e )
{
    Initialize ( );
    dgvDots.Rows.AddRange ( CurrentWriter.GetTheWriterDots
( ).ToArray ( ) );
    var otherWriters=writers.Where ( w => w.ID !=
CurrentWriter.ID );
    foreach ( var otherWriter in otherWriters )
    {
        foreach ( var row in
CurrentWriter.GetOnotherWriterDots ( otherWriter ).ToArray (
) )
        {
            if ( row.Cells[0].Value !=null )
            {
                dgvOthersDot.Rows.Add(row);
            }
        }
    }
    ArabicPAW[] paws= ArabicPAW.ExtractPAWs (
txtSearch.Text );
    foreach ( ArabicPAW paw in paws )
    {
        switch ( paw.Text.Length )
        {
            case 0:
                return;
            case 1:
                if ( !neededIsolatedChars.Contains ( paw.Text [ 0 ] ) )
                {
                    neededIsolatedChars.Add ( paw.Text [ 0 ] );
                }
                break;
            case 2:
                if ( !neededBeginningChars.Contains ( paw.Text [ 0 ] ) )
                {
                    neededBeginningChars.Add ( paw.Text [ 0 ] );
                }
        }
    }
}

```

```

if ( !neededEndChars.Contains ( paw.Text [ 1 ] ) )
    {
        neededEndChars.Add ( paw.Text [ 1 ] );
    }
    break;
default:
if ( !neededBeginningChars.Contains ( paw.Text [ 0 ] ) )
    {
        neededBeginningChars.Add ( paw.Text [ 0 ] );
    }
if ( !neededEndChars.Contains ( paw.Text [ paw.Text.Length - 1 ]
) )
    {
        neededEndChars.Add ( paw.Text [ paw.Text.Length - 1 ] );
    }
    foreach ( var item in paw.Text.ToCharArray ( 1 ,
paw.Text.Length - 2 ) )
        {
            if ( !neededMidChars.Contains ( item ) )
                {
                    neededMidChars.Add ( item );
                }
        }
        break;
    }
    }
    DisplayAllNeeded ( );
    SearchCurrentWriter ( );
    SearchOtherWriters ( );
    DisplayNotFoundInCurrentWriter ( );
    DisplayNotFound ( );
}

private void SearchOtherWriters ( )
    {
foreach ( var item in GetOthersIsolatedChars ( ).ToList ( ) )
    {
        DataGridViewImageCell imCell=new
DataGridViewImageCell ( );
        DataGridViewTextBoxCell writerCell=new
DataGridViewTextBoxCell ( );
        DataGridViewTextBoxCell distanceCell=new
DataGridViewTextBoxCell ( );
        DataGridViewRow nrow= new DataGridViewRow ( );
        imCell.Value = item.Image.ImageFromBytes ( );
        writerCell.Value = item.WriterID.ToString ( );
    }
}

```

```

        distanceCell.Value =
        ArabicTextRecovery.Writer.GetWriterByID (
        item.WriterID ).Distance ( CurrentWriter ).ToString (
        );
nrow.Cells.Add ( imCell ); nrow.Cells.Add ( writerCell );
nrow.Cells.Add ( distanceCell );
        dgvOthersIsolated.Rows.Add ( nrow );
    }
foreach ( var item in GetOthersBeginningChars ( ).ToList ( ) )
    {
        DataGridViewImageCell imCell=new
        DataGridViewImageCell ( );
        DataGridViewTextBoxCell writerCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewTextBoxCell distanceCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewRow nrow= new DataGridViewRow ( );
        imCell.Value = item.Image.ImageFromBytes ( );
        writerCell.Value = item.WriterID.ToString ( );
        distanceCell.Value =
        ArabicTextRecovery.Writer.GetWriterByID (
        item.WriterID ).Distance ( CurrentWriter
        ).ToString ( );
        nrow.Cells.Add ( imCell ); nrow.Cells.Add (
        writerCell ); nrow.Cells.Add ( distanceCell );
        dgvOthersBeginning.Rows.Add ( nrow );
    }
foreach ( var item in GetOthersMiddleChars ( ).ToList ( ) )
    {
        DataGridViewImageCell imCell=new
        DataGridViewImageCell ( );
        DataGridViewTextBoxCell writerCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewTextBoxCell distanceCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewRow nrow= new DataGridViewRow ( );
        imCell.Value = item.Image.ImageFromBytes ( );
        writerCell.Value = item.WriterID.ToString ( );
        distanceCell.Value =
        ArabicTextRecovery.Writer.GetWriterByID (
        item.WriterID ).Distance ( CurrentWriter
        ).ToString ( );
        nrow.Cells.Add ( imCell ); nrow.Cells.Add (
        writerCell ); nrow.Cells.Add ( distanceCell );
        dgvOthersMiddle.Rows.Add ( nrow );
    }

```



```

foreach ( var item in GetOthersEndChars ( ).ToList ( ) )
    {
        DataGridViewImageCell imCell=new
        DataGridViewImageCell ( );
        DataGridViewTextBoxCell writerCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewTextBoxCell distanceCell=new
        DataGridViewTextBoxCell ( );
        DataGridViewRow nrow= new DataGridViewRow ( );
        imCell.Value = item.Image.ImageFromBytes ( );
        writerCell.Value = item.WriterID.ToString ( );
        distanceCell.Value =
        ArabicTextRecovery.Writer.GetWriterByID (
        item.WriterID ).Distance ( CurrentWriter ).ToString (
        );
        nrow.Cells.Add ( imCell ); nrow.Cells.Add (
        writerCell ); nrow.Cells.Add ( distanceCell );
        dgvOthersEnd.Rows.Add ( nrow );
    }
}

private void SearchCurrentWriter ( )
    {
        foreach ( byte[] item in GetAvailableIsolatedChars (
        ).Select ( ch => ch.Image ).ToList ( ) )
            {
                DataGridViewImageCell imCell=new
                DataGridViewImageCell ( );
                DataGridViewRow nrow= new DataGridViewRow ( );
                imCell.Value = item.ImageFromBytes ( );
                nrow.Cells.Add ( imCell );
                dgvIsolated.Rows.Add ( nrow );
            }
        foreach ( byte[] item in
        GetAvailableBeginningchars ( ).Select ( ch => ch.Image
        ).ToList ( ) )
            {
                DataGridViewImageCell imCell=new
                DataGridViewImageCell ( );
                DataGridViewRow nrow= new DataGridViewRow ( );
                imCell.Value = item.ImageFromBytes ( );
                nrow.Cells.Add ( imCell );
                dgvBeginning.Rows.Add ( nrow );
            }
    }
}

```

```

        foreach ( byte[] item in GetAvailableMiddleChars (
).Select ( ch => ch.Image ).ToList ( ) )
        {
            DataGridViewImageCell imCell=new
DataGridViewImageCell ( );
            DataGridViewRow nrow= new DataGridViewRow ( );
            imCell.Value = item.ImageFromBytes ( );
            nrow.Cells.Add ( imCell );
            dgvMiddle.Rows.Add ( nrow );
        }
        foreach ( byte[] item in GetAvailableEndChars (
).Select ( ch => ch.Image ).ToList ( ) )
        {
            DataGridViewImageCell imCell=new
DataGridViewImageCell ( );
            DataGridViewRow nrow= new DataGridViewRow ( );
            imCell.Value = item.ImageFromBytes ( );
            nrow.Cells.Add ( imCell );
            dgvEnd.Rows.Add ( nrow );
        }
    }

private void DisplayAllNeeded ( )
{
    StringBuilder sb=new StringBuilder ( );
    sb.AppendLine ( "Isolated Chars:" );
    foreach ( char c in neededIsolatedChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
    sb.AppendLine ( Environment.NewLine + "Beginning
Chars:" );
    foreach ( char c in neededBeginningChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
    sb.AppendLine ( Environment.NewLine + "End Chars:" );
    foreach ( char c in neededEndChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
    sb.AppendLine ( Environment.NewLine + "Middle Chars:" );
    foreach ( char c in neededMidChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
}

```

```

    }
    lblNeeded.Text = sb.ToString ( );
}
private void DisplayNotFoundInCurrentWriter ( )
{
    StringBuilder sb=new StringBuilder ( );
    sb.AppendLine ( "Isolated Chars:" );
foreach ( char c in NonAvailableFromWriterIsolatedChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
sb.AppendLine ( Environment.NewLine + "Beginning Chars:" );
foreach ( char c in NonAvailableFromWriterBeginningChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
sb.AppendLine ( Environment.NewLine + "End Chars:" );
foreach ( char c in NonAvailableFromWriterEndChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
sb.AppendLine ( Environment.NewLine + "Middle Chars:" );

    foreach ( char c in NonAvailableFromWriterMidChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
    lblNotFoundInWriterDB.Text = sb.ToString ( );
}

private void DisplayNotFound ( )
{
    StringBuilder sb=new StringBuilder ( );
    sb.AppendLine ( "Isolated Chars:" );
foreach ( char c in NonAvailableFromOthersIsolatedChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
sb.AppendLine ( Environment.NewLine + "Beginning Chars:" );
    foreach ( char c in NonAvailableFromOthersBeginningChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
sb.AppendLine ( Environment.NewLine + "End Chars:" );
    foreach ( char c in NonAvailableFromOthersEndChars )

```

```

        {
            sb.AppendFormat ( "{0} " , c );
        }
sb.AppendLine ( Environment.NewLine + "Middle Chars:" );
foreach ( char c in NonAvailableFromOthersMidChars )
    {
        sb.AppendFormat ( "{0} " , c );
    }
lblNotFound.Text = sb.ToString ( );
}

private List<ArabicTextRecovery.Character>
GetAvailableIsolatedChars ( )
{
    foreach ( var item in neededIsolatedChars )
    {
        List<ArabicTextRecovery.Character> bc=
WriterCharacters.Where ( c => c.Position == -1 &&
c.CharText == item.ToString ( ) ).ToList ( );
        if ( bc.Count == 0 )
        {
            NonAvailableFromWriterIsolatedChars.Add ( item );
        }
        else
        {
            AvailableneededIsolatedChars.AddRange ( bc );
        }
    }
    return AvailableneededIsolatedChars;
}

private List<ArabicTextRecovery.Character>
GetAvailableBeginninghars ( )
{
    foreach ( var item in neededBeginningChars )
    {
List<ArabicTextRecovery.Character> bc= WriterCharacters.Where (
c => c.Position == 0 && c.CharText == item.ToString ( ) ).ToList
( );
        if ( bc.Count == 0 )
        {
            NonAvailableFromWriterBeginningChars.Add ( item );
        }
        else
        {
            AvailableneededBeginningChars.AddRange ( bc );
        }
    }
}

```

```

    }
    return AvailableneededBeginningChars;
}
    private List<ArabicTextRecovery.Character>
GetAvailableMiddleChars ( )
{
    foreach ( var item in neededMidChars )
    {
        List<ArabicTextRecovery.Character> bc=
WriterCharacters.Where ( c => c.Position > 0 &&
c.CharText == item.ToString ( ) ).ToList ( );
        if ( bc.Count == 0 )
        {
            NonAvailableFromWriterMidChars.Add ( item );
        }
        else
        {
            AvailableneededMidChars.AddRange ( bc );
        }
    }
    return AvailableneededMidChars;
}
    private List<ArabicTextRecovery.Character>
GetAvailableEndChars ( )
{
    foreach ( var item in neededEndChars )
    {
        List<ArabicTextRecovery.Character> bc=
WriterCharacters.Where ( c => c.Position == -2 &&
c.CharText == item.ToString ( ) ).ToList ( );
        if ( bc.Count == 0 )
        {
            NonAvailableFromWriterEndChars.Add ( item );
        }
        else
        {
            AvailableneededEndChars.AddRange ( bc );
        }
    }
    return AvailableneededEndChars;
}

    private List<ArabicTextRecovery.Character>
GetOthersIsolatedChars ( )

```

```

    {
        foreach ( var item in
NonAvailableFromWriterIsolatedChars )
            {
                List<ArabicTextRecovery.Character> bc=
OthersCharacters.Where ( c => c.Position == -1 &&
c.CharText == item.ToString ( ) ).ToList ( );
                if ( bc.Count == 0 )
                    {
                        NonAvailableFromOthersIsolatedChars.Add ( item );
                    }
                else
                    {
                        AvailabFromOtherslneededIsolatedChars.AddRange ( bc );
                    }
            }
        return AvailabFromOtherslneededIsolatedChars;
    }
    private List<ArabicTextRecovery.Character>
GetOthersBeginningChars ( )
    {
        foreach ( var item in NonAvailableFromWriterBeginningChars )
            {
                List<ArabicTextRecovery.Character> bc=
OthersCharacters.Where ( c => c.Position == 0 &&
c.CharText == item.ToString ( ) ).ToList ( );
                if ( bc.Count == 0 )
                    {
                        NonAvailableFromOthersBeginningChars.Add ( item );
                    }
                else
                    {
                        AvailabFromOtherslneededBeginningChars.AddRange ( bc );
                    }
            }
        return AvailabFromOtherslneededBeginningChars;
    }
    private List<ArabicTextRecovery.Character>
GetOthersMiddleChars ( )
    {
        foreach ( var item in NonAvailableFromWriterMidChars )
            {
                List<ArabicTextRecovery.Character> bc=
OthersCharacters.Where ( c => c.Position > 0 && c.CharText ==
item.ToString ( ) ).ToList ( );

```

```

        if ( bc.Count == 0 )
        {
            NonAvailableFromOthersMidChars.Add ( item );
        }
        else
        {
            AvailabFromOtherslneededMidChars.AddRange ( bc );
        }
    }
    return AvailabFromOtherslneededMidChars;
}
private List<ArabicTextRecovery.Character>
GetOthersEndChars ( )
{
    foreach ( var item in NonAvailableFromWriterEndChars )
    {
        List<ArabicTextRecovery.Character> bc=
        OthersCharacters.Where ( c => c.Position == -2 &&
        c.CharText == item.ToString ( ) ).ToList ( );
        if ( bc.Count == 0 )
        {
            NonAvailableFromOthersEndChars.Add ( item );
        }
        else
        {
            AvailabFromOtherslneededEndChars.AddRange ( bc );
        }
    }
    return AvailabFromOtherslneededEndChars;
}
private void dgv_CellContentDoubleClick ( object sender ,
DataGridViewCellEventArgs e )
{
    if ( frmImage == null || frmImage.IsDisposed )
    {
        frmImage = new FormImages ( this );
    }
    frmImage.Show ( );
}
}
}
}

```

**//show the recovered text**

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RecoverText
{
    public partial class FormImages : Form
    {
        public List<Bitmap> Images { get; set; }
        public Form1 MyParent { get; set; }
        public List<DataGridView> dgvs;
        private Point MouseDownLocation;
        private bool Normalize=false;
        private Point CurentPoint;
        public FormImages ( Form1 parent )
        {
            Images = new List<Bitmap> ( );

            MyParent = parent;
            dgvs = MyParent.dgvs;
            InitializeComponent ( );
            foreach ( var item in dgvs )
            {
                item.CellContentDoubleClick += CellContentDoubleClick;
            }
            CurentPoint = new Point ( this.Width - 100 , 100 );
        }
        PictureBox lastPb;
        void CellContentDoubleClick ( object sender ,
            DataGridViewCellEventArgs e )
        {
            DataGridView dgv=sender as DataGridView;
            if ( dgv == null || e.RowIndex < 0 ) return;
            Bitmap image= dgv.Rows [ e.RowIndex ].Cells [ 0
                ].Value as Bitmap;
            Images.Add ( image );
            PictureBox pb= new PictureBox ( );
        }
    }
}

```



```

if ( Normalize )
    {
        pb.Height = 50;
        pb.Width = ( int ) ( 50.0 / ( double )
image.Height ) * image.Width;
    }
else
    {
        pb.Height = image.Height;
        pb.Width = image.Width;
    }
GetCurrentPoint ( sender as DataGridView , pb );
pb.Location = CurentPoint;//new Point ( pb.Width *
Images.Count + 10 * Images.Count , pb.Height );
SetCurrentPoint ( sender as DataGridView , pb );
pb.SizeMode = PictureBoxSizeMode.StretchImage;
pb.Image = image;
pb.MouseDown += pb_MouseDown;
pb.MouseMove += pb_MouseMove;
this.Controls.Add ( pb );
}
private void SetCurrentPoint ( DataGridView sender ,
PictureBox pb )
    {
        if ( sender.Name.Contains ( "Isolated" ) ||
sender.Name.Contains ( "End" ) )
            {
                CurentPoint.X = CurentPoint.X - pb.Width * 2;
                lastPb = pb;
            }
        else if ( sender.Name.Contains ( "Dot" ) )
            {
                CurentPoint.Y = CurentPoint.Y + pb.Height * 2;
                if ( lastPb != null )
                    {
                        CurentPoint.X = CurentPoint.X - lastPb.Width;
                    }
            }
        else
            {
                CurentPoint.X = CurentPoint.X - pb.Width;
                lastPb = pb;
            }
    }
}

```

```
private void GetCurrentPoint ( DataGridView sender ,  
PictureBox pb )  
{  
  
    if ( sender.Name.Contains ( "Dot" ) )  
    {  
  
        if ( lastPb != null )  
        {  
CurentPoint.X = CurentPoint.X + lastPb.Width;  
CurentPoint.Y = CurentPoint.Y - lastPb.Height + 1;  
        }  
    }  
}
```

## References

- Abandah G.A. & Khedher M.Z. (2005). Analysis of Handwritten Arabic Letters Using Selected Feature Extraction Techniques. *International Journal of Computer Processing of Languages*, 22(1), 49-73.
- Al-ammam, M., al-majed, R. & Aboalsamh, H. (2005). Online Handwriting Recognition for the Arabic Letter Set. Thesis Master, CIT'11 Proceedings of the 5th World Scientific and Engineering Academy and Society (WSEAS). International conference on Communications and information technology, Wisconsin, USA.
- Al-Jawfi R. (July 2009). Handwriting Arabic Character Recognition LetNet Using Neural Network. *The International Arab Journal of Information Technology*, 6(12), p 304-309.
- Alireza Alaei, P. Nagabhushan, Umapada Pal, 2010, A Baseline Dependent Approach for Persian Handwritten Character Segmentation, *International Conference on Pattern Recognition, IEEE computer society*, DOI 10.1109/ICPR, pp 1977- 1980.
- Alnsour A.J. and Alzoubady L.M. (2006). Arabic Handwritten Characters Recognized by Neocognitron Artificial Neural Network. University of Sharjah. *Journal of Pure & Applied Sciences*, 3(2).  
Retrieved online from:  
[https://www.sharjah.ac.ae/English/About\\_UOS/UOSPublications/Appliedsciences/Issues/Documents/3\\_2/ AlNsour\\_1\\_e.pdf](https://www.sharjah.ac.ae/English/About_UOS/UOSPublications/Appliedsciences/Issues/Documents/3_2/ AlNsour_1_e.pdf).
- Amin A. (1997). Arabic character recognition, *Handbook of Character Recognition and Document Image Analysis. World Scientific Publishing Company*, pp. 397-420.
- Aouadi N. and Kacem A. (2005). A new system of Word Spotting for manuscript retrieval based on Generalized Hough Transform. UTIC: research Unit of Technologies of Information and communication ESSTT: High School of Sciences and Techniques of Tunis, University of Tunis, Tunisia,  
Retrieved from:  
[http://grec2011.cau.ac.kr/GREC2011Proceedings/Session1MapandancientDocuments/grec2011\\_submission\\_20.pdf](http://grec2011.cau.ac.kr/GREC2011Proceedings/Session1MapandancientDocuments/grec2011_submission_20.pdf).

- Aymen C., Houcine B., Haikal El Abed , 2010, Fractal and Multi-Fractal for Arabic Offline Writer Identification, *International Conference on Pattern Recognition, IEEE computer society*, DOI 10.1109/ICPR, p 3793- 3796.
- Azizah S., Nasir S.M. & Mohamed O. (2010). Chain Coding and Preprocessing Stages of Handwritten Character Image File. *Electronic Journal of Computer Science and Information Technology (eJCSIT)*, 2(1), 6-13.
- Brook S. and Zaher Al-Ghbari. (2008). Classification of Personal Arabic Handwritten Documents. *Information science & applications*, 5(6), ISSN: 1790-0832.
- Bunke M.H, Jianying Hu. , Fumitaka K. and Ching Y.S. (2009). New Frontiers in Handwriting Recognition. *Pattern Recognition*, 42(12), 3129-3130.
- Bunke, H. & Patrick S.P. Wang, 1997. Handbook of Character Recognition and Document Image Analysis.
- Burrow P., (2004). Arabic Handwriting Recognition. Master Thesis. School of Informatics, University of Edinburgh, UK. Retrieved from <http://www.inf.ed.ac.uk/publications/thesis/online/IM040241.pdf>
- Earnest L. D., 1963. Machine Reading of Cursive Script, in Proc. *IFIP Congress*, Amsterdam, pp.462-466.
- Feliachi Ali, Ronald Klein, Norman Lass, Roy Nutter, Powsiri Klinkhachorn. 2002. Off-line Thai Handwriting Recognition In Legal Amount. Phd thesis, Morgantown, West Virginia.
- Freeman, J.A. & Skapura D.M. (1991). Neural Networks-Algorithms, Applications, and Programming Techniques. *Addison Wesley. Reading, MA*, ISBN 0-201-51376-5.
- Fujisaki H., Nagai S., and Hidaka N., 1971, On-line recognition of handwritten numerals, Annual Report, Faculty Engineering, University of Tokyo, Japan, Volume 30, pp. 103-110.
- Hanan Aljuaid, Zulkifli M. and Muhammad Sarfraz, 2010, A Tool to Develop Arabic Handwriting Recognition System Using Genetic Approach, *Journal of Computer Science* 6 (5), ISSN 1549-3636, Science Publications, pp 490-495.
- Hussain, F. & Cowell, J. (2000). Character Recognition of Arabic and Latin Scripts. *IEEE International Conference on Information Visualization*, pp. 51–56.

- I. S. Abuhaiba, S. A. Mahmoud, and R. J. Green, 1994, Recognition of handwritten cursive Arabic characters, *IEEE Transactions of Pattern Analysis and Machine Intelligence*, Vol. 16, No. 6.
- Jumari K. & Mohamed A. Ali, Jun 2002, a survey and comparative evaluation of selected off-line arabic handwritten character recognition systems, University of technology, Malaysia, *Journal Technology*, 36(E): pp. 1–18
- Khorsheed, M.S. and Clocksin, W.F. (2005). Structural Features of Cursive Arabic Script, Computer Laboratory. *10th British Machine Vision Conference*, vol.2, pp. 422-431.
- Khorsheed. M. S., 2002, Off-line arabic character recognition - a review, *Pattern Analysis & Applications*, Vol 5, pp.31–45.
- Klassen, T. ,(2001), Towards neural network recognition of handwritten Arabic letters. Master thesis, Dalhousie University, Halifax, USA.
- Laheeb M. Alzoubady and Ayman J. Alnsou, June 2006, Arabic Handwritten Characters Recognized by Neocognitron Artificial Neural Network University of Sharjah, *Journal of Pure & Applied Sciences*, Vol. 3, No. 2.
- Lazzerini Beatrice and Marcelloni Francesco , 2000, A linguistic fuzzy recogniser of off-line handwritten characters, University of Pisa, Italy, *Pattern Recognition Letters*, 2, pp.319-32.
- Leane. M.M, Cumming.I, Corrigan.O.I, 2003, The use of artificial neural networks for the selection of the most appropriate formulation and processing variables in order to predict the in vitro dissolution of sustained release minitablets. *AAPS PharmSciTech*; 4 (2).
- Lorigo, L.M. & Venu, G. (2006). Off-line Arabic handwritten recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(40), 1-40.
- M. A. Al-Alaoui, M. A. Abou Harb, Z. A., Chahine, and E. Yaacoub, 2009, A New Approach for Offline Handwriting Recognition. *IEEE MEEM* 4(3), pp. 89-97.
- Majida Ali A., Hamid Ali Abed AL-Asadi, Zainab S. B., Ahmad N. I., 2010, Fuzzy Logic approach to Recognition of Isolated Arabic Characters, *International Journal of Computer Theory and Engineering*, Vol. 2, No. 1, p 119-124.

- M. Liana, and V. G. Lorigo, 2006, Off-line Arabic Handwriting Recognition: A Survey, *IEEE Transactions on pattern analysis and machine intelligence*, 28, pp. 712-724.
- Mantas J., 1986, An Overview of Character Recognition Methodologies, *Pattern Recognition*, vol.19, no.6, pp. 425-430.
- Mohammad Amin Abou-harb, June 2011, feature based Arabic handwriting recognition for teaching illiterates, thesis, American university of Beirut, Beirut, Lebanon.
- Muhammad Sarfraz, Syed Nazim Nawaz, Abdulaziz Al-Khuraidly, 2003, Offline Arabic Text Recognition System. In proceeding of: Geometric Modeling and Graphics, ISBN: 0-7695-1985-7, pp. 30-35.
- Naveen G. and Karun, June-2009, Handwritten Gurumukhi Character Recognition Using Neural Networks, Thesis, Computer Science and Engineering Department , Thapar University.
- Newman, R. and Downton, A., Jan 1997, An Offline Script and Character Recognition Toolset. (OSCAR), On-line reference retrieved from: [http://www.essex.ac.uk/ease/research/mma\\_lab/newoscar/people/oscar.html](http://www.essex.ac.uk/ease/research/mma_lab/newoscar/people/oscar.html).
- Plamondon R., and Srihari S.N, (2000). Online and Offline handwriting recognition: A comprehensive survey, *Pattern Analysis and Machine Intelligence*, IEEE Transactions, 22(1), 63-84.
- Rath. T., Lavrenko, V. and Manmatha, R., 2003, Retrieving Historical Manuscripts using Shape, CIIR Technical Report.
- Saeed, K. (1995). Experimental Algorithm for Testing the Realization of Transfer Functions. Proc. of 14th IASTED Conf. on *Modeling, Identification and Control*, pp. 114-116.
- Salvador España Boquera, María José Castro Bleda, Jorge Gorbe Moya, Francisco Martínez, 2011, Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models, *IEEE Trans. Pattern Anal. Mach. Intell.* pp.14-21.
- Sarhan, M.A. & Helalat O. (2007, May). Arabic Character Recognition using ANN Networks and Statistical Analysis, World Academy of Science, Engineering and Technology NYIT-New York Institute of Technology. Retrieved from <http://www.waset.org/journals/waset/ v27/v27-6.pdf>

- Shapiro, L., and Stockman, G. (2002). Computer Vision. Prentice Hall. pp. 69–73.
- Singh S. and A. Amin, 1998, Neural network recognition and analysis of hand-printed characters, Proc. *IEEE International Joint Conference on Neural Networks IJCNN'98, IEEE World Congress on Computational Intelligence*, Anchorage, Alaska, Vol.3, pp. 1743-1747.
- Singh, Y.P. Yadav, V.S. Gupta, A. & Khare A. (2009). Bi Directional Associative Memory Neural Network Method in the Character Recognition. *Journal of Theoretical and Applied Information Technology*. 5(4), 382-386.
- Sofien Touj, Najoua Essoukri Ben Amara, Hamid Amiri, 2005, Generalized Hough Transform for Arabic Optical Character Recognition. *The International Arab Journal of Information Technology*, Vol. 2, No. 4, pp 326 – 333.
- Somaya Al-Ma'adeed A. S., 2004, Recognition of Off-line Handwritten Arabic Words, thesis, University of Nottingham.
- Somaya Alma'adeed, Colin Higgins, and Dave Elliman, 2004, Off-line recognition of handwritten Arabic words using multiple hidden Markov models, *Knowledge-Based Systems*, Volume 17, Issues 2–4, pp. 75-79.
- Soryani, M. & Rafat, N. (2006, December). Application of Genetic Algorithms to Feature Subset Selection in a Farsi OCR. *International Journal of Engineering and Applied Sciences*, 3(2), 71-74.
- Salvador España Boquera, María José Castro Bleda, Jorge Gorbe-Moya, Francisco Zamora-Martínez, 2011, Improving Offline Handwritten Text Recognition with Hybrid HMM/ANN Models. *IEEE Trans. Pattern Anal. Mach. Intell.* 33(4): 767-779.
- Tomai C. I., Zhang B. and Govindaraju V., August 2002, Transcript Mapping for Historic Handwritten Document Images. In: Proc. *Of the 8th Int'l Workshop on Frontiers in Handwriting Recognition*, pp. 413- 418.
- Zaidan, A.A., Zaidan B.B., Jalab H.A., Alanazi H.O. and Alnaqeib Rami, (May 2010), Offline Arabic Handwriting Recognition Using Artificial Neural Network, *journal of computer science and engineering*, 1(1), 47-67.