AN EFFICIENT LANGUAGE-INDEPENDENT MULTI-FONT OCR FOR ARABIC SCRIPT

Hussein Osman, Karim Zaghw, Mostafa Hazem and Seifeldin Elsehely

Computer Engineering Department, Faculty of Engineering,
Cairo University, Egypt

ABSTRACT

Optical Character Recognition (OCR) is the process of extracting digitized text from images of scanned documents. While OCR systems have already matured in many languages, they still have shortcomings in cursive languages with overlapping letters such as the Arabic language. This paper proposes a complete Arabic OCR system that takes a scanned image of Arabic Naskh script as an input and generates a corresponding digital document. Our Arabic OCR system consists of the following modules: Pre-processing, Word-level Feature Extraction, Character Segmentation, Character Recognition, and Post-processing. This paper also proposes an improved font-independent character segmentation algorithm that outperforms the state-of-the-art segmentation algorithms. Lastly, the paper proposes a neural network model for the character recognition task. The system has experimented on several open Arabic corpora datasets with an average character segmentation accuracy 98.06%, character recognition accuracy 99.89%, and overall system accuracy 97.94% achieving outstanding results compared to the state-of-the-art Arabic OCR systems.

KEYWORDS

Arabic OCR, Word Segmentation, Character Segmentation, Character Recognition, Neural Network

1. INTRODUCTION

The problem of Optical Character Recognition (OCR) has been the scope of research for many years [1]–[3] due to the need for an efficient method to digitize printed documents, prevent their loss and gradual unavoidable wear, as well as increase their accessibility and portability. The challenges that face Arabic OCR systems stem from the cursive and continuous nature of Arabic scripts. The presence of a semi-continuous baseline in Arabic text prevents the use of segmentation techniques proposed for other OCR systems. Moreover, the vertical overlapping of characters caused by ligatures means that segmenting a word along a single horizontal line will not achieve perfect segmentation. Furthermore, the challenges introduced by the nature of the Arabic script do not only affect character segmentation. The recognition of Arabic characters requires a huge training set since each character can have a different shape depending on its position in the word. Also, cases of constantly misclassifying a character as another specific one are frequent due to the presence of characters that are only told apart through the number of dots.
In this paper, we propose a complete OCR pipeline for Arabic text that is language-independent and supports multiple fonts. Our system takes a scanned image of an Arabic document as an input and outputs a digitized text document containing the predicted text. The input image is first preprocessed where binarization, denoising, and deskewing are carried out, followed by line and word segmentation. Character segmentation is then performed based on the extracted word-level features, followed by cut filtration based on the wide rules-set we have defined. The segmented

David C. Wyld et al. (Eds): ACITY, DPPR, VLSI, WeST, DSA, CNDC, IoTE, AIAA, NLPTA - 2020
pp. 57-71, 2020. CS & IT - CSCP 2020 DOI: 10.5121/csit.2020.101506

characters are then fed into our character recognition neural network model which classifies the given character into one of the 29 Arabic characters. Furthermore, post-processing is applied to the predicted characters and conveniently concatenates them into a comprehensible text document.

Our main contribution in this paper lies in the development of an efficient light-weight system that outperforms current state-of-the-art accuracy in the field of Arabic OCR. We achieve outstanding runtime results without trading off our system accuracy through efficient denoising of documents, vectorized implementation of all segmentation stages, and finely tuning our recognition model's complexity. In addition, we maintain our overall system accuracy by improving the character segmentation using the proposed improved cut filtration algorithm. The algorithm is robust against structural, morphological, and topological similarities between letters. We also propose a neural network model to learn the underlying features of input characters and build a character classification model. We finally eliminate the use of any lexical analysis to maintain the language independence of the system.

The rest of the paper is organized as follows: Section II describes the state-of-art related work. Section III discusses in detail the proposed OCR system. Section IV describes the datasets used for training the neural network and for evaluating the overall system performance, then discusses the results with comprehensive comparisons with other related algorithms and methods. Section V discusses limitations in our proposed system and proposals on future work in this area. Finally, Section VI presents the paper's conclusion.

2. RELATED WORK

There has been a variety of techniques proposed in the area of OCR for Arabic text. In this section, we will review the different approaches in the literature for character segmentation, feature extraction, and character recognition in OCR systems.

A significant number of approaches for the Arabic character segmentation task have been proposed in the literature. Mohamed et al. [4] proposed the use of contour extraction to facilitate the character segmentation phase followed by cut index identification based on the contour of a word. Their method achieved significant results in character segmentation; however, its performance degraded in the case of small-sized Arabic fonts or noisy documents. The scale-space method utilized by El Makhfi et al. [5] represents another direction in Arabic character segmentation. This method works well for scanned documents containing high random noise since blobs are retrieved from characters and are then detected to recover the appropriate cut positions in the image. Although this approach has been used in several computer vision applications since its first proposal, its use in Arabic character segmentation is still widely unexplored.

Inspired by NLP applications, Alkhateeb et al. [6] and Radwan et al. [8] proposed the use of a sliding window approach for segmentation. A Convolutional Neural Network (CNN) is used to determine the likelihood of a given sliding window consisting of several segments to be an actual character. Subsequently, the segments that qualify as characters are fed into their recognition model. This segmentation approach has shown very high performance on a single font but failed to maintain this high performance when tested on multiple fonts and font sizes. Lots of efforts [8]–[10] in Arabic character segmentation have been based on histogram analysis of words.
It is important to mention that Qaroush et al. [8] also proposes a very effective word segmentation methodology that achieves state-of-the-art accuracy by implementing cut identification and filtration through gap length. Their proposed method handles multiple fonts and font sizes.

Although not being as challenging as character segmentation, feature extraction is one of the keys to boosting the accuracy of any OCR system. Rashad and Semary [11] adopted a simple approach that manually extracts basic features from each character such as height, width, number of black pixels, number of horizontal transitions, number of vertical transitions, and other similar features. Rashid et al. [12] proposed a multi-dimensional Recurrent Neural Network (RNN) for their recognition system that achieved outstanding character recognition rates. However, the effects of using this complex approach on the runtime were not properly investigated. The use of deep learning approaches is highly efficient when developing Arabic OCR systems that operate on unconstrained scene text and video text, not scanned documents [13].

Dahi et al. [14] adopted a similar approach by manually selecting features from a noise-free and pre-segmented character input. They added a font recognition module before the feature extraction, to include the font as a feature for the character recognition, alongside other slightly complex features such as ratios between black pixel count per region and the statistical centre of mass for each character proposed by [15]. The overall system of [14] achieved very high accuracy for Arabic character recognition.

It is worth mentioning that the OCR system of [14] did not include a character segmentation module as it worked only on a pre-segmented character input. Additionally, the OCR architecture of Dahi et al. [14] failed to scale up and recognize Arabic characters in other fonts that were not supported by the font recognition module. A convenient middle ground between ineffective manual extraction [11], [14], and the computationally expensive use of deep learning [12], [13] is presented by the use of Principal Component Analysis (PCA) for automatic feature extraction.
As proposed by Shayegan and Aghabozorgi [16], PCA provides an efficient and effective solution for the problem of feature extraction in recognizing Arabic numerals. However, applying PCA becomes computationally infeasible for the huge datasets needed for training Arabic character recognizers.

As for character recognition, implementing this module without the use of machine learning has been deemed obsolete; because of the outstanding results achieved by machine learning recognition models. Hence, we will only consider the techniques for character recognition that are based on machine learning for review in the subsequent paragraphs. Shahin [17] proposed using linear and ellipse regression to generate codes from the segmented characters. This approach of codebook generation and code matching showed average results for character recognition. Additionally, it suffered from the same problem of not being able to generalize to other Arabic fonts as [14]. The use of the holistic approach in [18] emerged from the difficulty of the segmentation phase as we mentioned. This word-level recognition technique skips all the inaccuracy produced by segmentation errors but creates the need for post-recognition lexical analysis, thus resulting in a language-dependent system that relies on a look-up data base and semantic checks after recognition.

Often paired with the use of a sliding window for segmentation, the use of a Hidden Markov Model (HMM) for character recognition was adopted by [18]–[20]. By using a model that mimicked the architecture of an Automatic Speech Recognition system and an HMM, Rashwan et al. [19] managed to overcome the challenges presented by the presence of ligatures. Many OCR systems use other classical machine learning techniques in their character recognition module; such as random forests [11], [14], K-Nearest Neighbor [11], shallow neural networks [21]. The neural network model used by Al-Jarrah et al. [21] yielded the best results, compared to other classical machine learning techniques [11], [14], [19], [20], and was able to generalize over different fonts.

3. METHOD PROPOSED

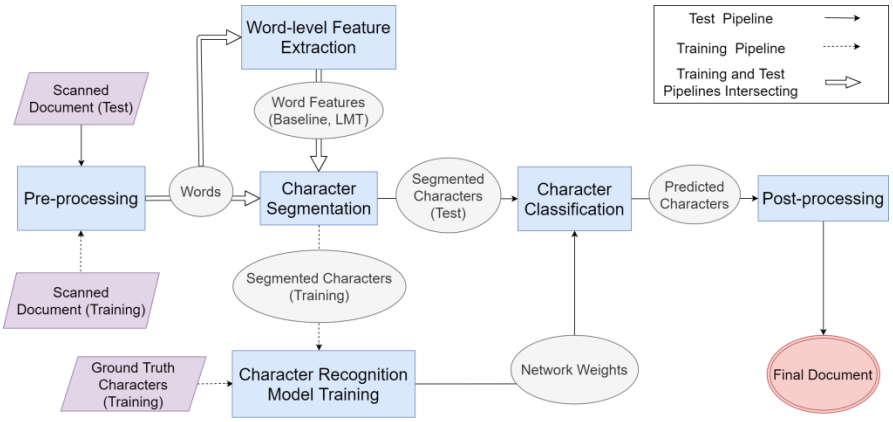
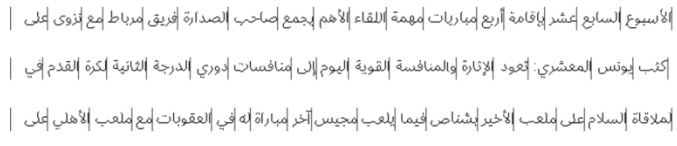


Figure 1. System Architecture: A block diagram representing the system modules and both training and testing (prediction) pipelines.

As shown in Figure 1, the input to the OCR system is expected to be a number of scanned images of computerized Arabic documents. In the Pre-processing stage, we apply image preprocessing through the filtering, deskewing, and denoising of the input images, followed by line and word segmentation. In the Word-Level Feature Extraction stage, we generate statistical, structural, and topological features for every word. In the Character Segmentation stage, we apply the Excessive Cut Creation and Improved Cut Filtration algorithms to segment the word into individual characters. These segmented characters, together with the associated ground truth labels, represent the dataset that our Character Recognition Model uses for training. We train an Artificial Neural Network (ANN) to classify each segmented character into one of the 29 possible characters in the Arabic language. Finally, we aggregate the segmented characters into words and generate the output of our OCR system.

3.1. Pre-processing

a)



b)

Figure 2. Examples from Preprocessing Stages: a) cut indices for line segmentation, on document b) cut indices for word segmentation, on lines

The preprocessing module consists of four main steps: raw image filtering, document deskewing, line segmentation, and word segmentation. Initially, we start by converting the input image to

grayscale, then binarizing it by applying Adaptive Gaussian Thresholding. The document deskewing is carried out by rotating the document about its geometric centre with a specific angle calculated through obtaining the orientation of the text's minimum bounding rectangle. This is followed by another round of binary thresholding to set binary values for the pixels that have been interpolated due to the previous rotation. The line segmentation step is performed by blurring the image then applying horizontal histogram projection of black pixels. Local minima of this histogram indicate positions of separations between lines. The goal of blurring is to avoid generating segmented lines containing dots only (e.g. the dots of the 'yaa' letter at the end of the word) or containing special Arabic diacritics only (e.g. 'hamza' or 'shadda'). For word segmentation, we applied thinning to the image instead of blurring. We propose thinning as a solution to enhance the fine details within and between words. This eliminates the overlapping pixels between any two words (if one of them ends with a curved letter) and results in standardized gap lengths between words. We subsequently implemented [8]'s algorithm for cut identification and filtration based on gap lengths.

3.2. Word-level Feature Extraction

The word-level feature extraction algorithm takes the segmented words as an input and generates for each word several geometric features. These features are essential for the character segmentation algorithm to be able to identify individual characters and segment them accordingly. We discuss the generated features for each segmented word in this subsection.

3.2.1. Baseline

The baseline is an imaginary horizontal line that connects all of the letters in an Arabic word [8]. In order to detect the baseline for every word, we search for the row of pixels with the greatest number of black pixels by applying horizontal histogram projection and finding the global maximum.

3.2.2. Line of Maximum Transitions (LMT)

We define a transition as a change in pixel value from 0 (black) to 1 (white) or vice versa. An important feature of the Arabic script is that a transition above the baseline is always due to a character being drawn. The Line of Maximum Transitions (LMT) is the line that cuts through the greatest number of these transitions (i.e. the row of pixels in which the number of transitions from black to white and white to black pixels is greatest) [8]. For better estimation of the baseline and LMT, we propose that both features should be derived from the whole line of text rather than from each word.

3.2.3.Potential Cut Region (PCR)

The LMT's key characteristic is that it passes through all potential characters in an Arabic word and is therefore essential in separating the word into its individual character components. A cut is defined as an imaginary line that separates two characters, and a Potential Cut Region is the area where a cut may exist. Since we cannot at first determine which character-intersections with the LMT belong to the same character and which are the result of a new character being written, we assume that each intersection represents a distinct character and therefore a PCR exists between any two successive intersections.

In order to determine the start and end indices of PCRs, we traverse the LMT from right to left. Each black pixel followed by a white pixel is defined as a start index of a PCR and each white

pixel followed by a black pixel is defined as an end index. Furthermore, the column of pixels that is chosen as the location of a cut is called a cut index.

Figure 3. An Arabic word with the baseline highlighted in red, the LMT highlighted in green, the PCR start indices highlighted in dark blue, and the PCR end indices highlighted in light blue.

3.3. Character Segmentation

To solve the problem of over-segmentation of characters, our character segmentation algorithm consists of two main steps: Excessive Cut Creation (ECC), which generates excessive potential cuts, and Improved Cut Filtration (ICF), which filters the false cuts from these potential cuts and outputs a set of valid cuts only. The Improved Cut Filtration algorithm is considered an improvement over the Cut Filtration algorithm proposed by Qaroush et al. [8].

3.3.1. Excessive Cut Creation Algorithm (ECC)

In the Arabic script, characters are either connected through the baseline or separated by a single space. Therefore, there are two different methods used in the Excessive Cut Creation algorithm: Finding Baseline Cuts and Finding Separation Cuts. The former deals with separating baseline-connected characters and some space-separated characters while the latter addresses the remaining space-separated characters.

I. Baseline Cuts: In order to identify a baseline cut, we inspect each column of pixels in a PCR, starting from the end index to the start index. We count the total number of black pixels above and below the baseline and, for every PCR, we propose the position of the cut index to be the first column where the count is zero, i.e. where the only black pixel allowed is the baseline. This approach is useful as a preliminary step for separating baseline-connected characters and also helps in separating some space-separated characters; e.g. the ‘aleph’, ‘daal’ or ‘thaal’ (ـا، ـد ، ـذ) followed by another letter.

II. Separation Cuts: While baseline cuts succeed in separating some space-separated characters, it will not place a cut whenever a black pixel exists below the baseline. This introduces a real challenge for letters that have curves which dip below the baseline such as ‘reh’ and ‘zeen’ (ـر، ـز) since the entire PCR may contain black pixels below the baseline. To solve this problem, we place a separation cut whenever the pixels at the left and right indices of a PCR are not connected by an uninterrupted path of black pixels. This ensures that a cut will be placed wherever there are two space-separated characters even if one of them happens to dip below the baseline. We choose the separation cut index to be the middle of this PCR.

As illustrated in Algorithm 1, we search every PCR for a baseline cut. If we find a baseline cut, then we add this cut to the set of cut Indices. If we fail to find a baseline cut, we search for a

separation cut. If we find a separation cut, then we add this cut to the set of cut Indices. If we find neither a baseline cut nor a separation cut, we claim that this PCR contains a part of a character that should not be cut.

Algorithm 1. ECC Algorithm

```
Input: PCRArray

ExcessiveCutIndices = Φ
for all PCR in PCRArray do
    CutFound = False
    for all PixelColumn in PCR do
        if ProjAboveBaseline = 0 and ProjBelowBaseline = 0 then
            CutFound = True
            ExcessiveCutIndices.Add(PixelColumn.Index)
            break
    if CutFound = False and PCR.IsConnected = False then
        SeparationCutIndex = (PCR.LeftIndex + PCR.RightIndex) / 2
        ExcessiveCutIndices.Add(SeparationCutIndex)

Output: ExcessiveCutIndices
```

3.3.2. Improved Cut Filtration Algorithm (ICF)

After generating a relatively large number of potential cut indices in the ECC stage, we begin inspecting each Potential Character (PC), where a PC is defined as any region that exists between two successive cuts. The goal of the cut filtration stage is to determine which of the cut indices are excessive false cuts. We identify the Arabic letters that usually cause false cuts in the cut filtration algorithm in the paper written by [8] and other character segmentation algorithms and we propose an improved algorithm to detect all of these letters. To the best of our knowledge, there is no single algorithm that can perfectly segment all Arabic letters, and hence we introduce our solution as the new state-of-the-art. We will discuss these challenging cases, and how ICF handles each of them accordingly.

I. **‘Seen’ and ‘Sheen’ case (س، ش):** The most notable causes of excessive false cuts are the letters ‘seen’ and ‘sheen’ (س، ش). These two letters are composed of three successive strokes, with three dots above the second stroke in the case of ‘sheen’. A stroke is a PC that represents a part of a character having a one-pixel thickness. Because each stroke passes through the LMT, the ECC algorithm generates three cut indices, instead of one. In order to filter these cuts, we define a seen-stroke as a stroke with no dots above or below the baseline and no hole. A seen-stroke is also characterized by having a small peak above the baseline (i.e. is relatively short) and a flat structure near the baseline (does not dip below the baseline). Also, we define a sheen-stroke as a seen-stroke with dots above the baseline.

   Based on the above definitions, the false cuts in the ‘seen’ or ‘sheen’ letters in the start and middle of the word (ـسـ، ـشـ) will be filtered by detecting three successive seen-strokes for the ‘seen’ case or two seen-strokes with one sheen-stroke in between for the ‘sheen’ case. Nevertheless, this method filters the false cuts of the two letters in the start and the middle of the word, but it fails to detect false cuts when they appear at the end of the word.
   We propose an improvement over this algorithm by taking into consideration the unique characteristic of the ‘seen’ and ‘sheen’ letters when they exist at the end of the word. This characteristic is the bowl shape seen at the end of these letters (ـس ، ـش). We define a bowl as (1) a PC with no dots above or below the baseline (2) a PC such that its right cut index

must have at least one black pixel and its left cut index must contain no black pixels. This means that the bowl must be directly connected through the baseline to a PC before it and must not be connected to any PC after it. (3) a PC such that there must exist a region where the baseline vanishes within but resurfaces again (i.e. there exists a dip below the baseline). (4) a PC with a relatively small peak above the baseline.

Based on this improved algorithm, the false cuts (the first two cut indices) in the 'seen' letter will be filtered when the IFC algorithm detects three successive seen-strokes or two successive seen-strokes followed by a bowl, whereas the false cuts (the first two cut indices) in the 'sheen' letter will be filtered when the IFC algorithm detects two seen-strokes separated by a sheen-stroke or a seen-stroke followed by a sheen-stroke and a bowl.

II. **'Saad' and 'Daad' case (ص، ض، صـ، ضـ):** The second set of characters that result in an additional false cut is 'saad' and 'daad' (ص، ض، صـ، ضـ). To filter the false cuts in these two letters, we define a hole as a PC containing a rounded area of white pixels (a hole) enclosed by a larger rounded area of black pixels. The two letters consist of a hole followed by a seen-stroke when they appear at the start or the middle of a word, or a hole followed by a bowl when they appear at the end of a word. Initially, we wrote our cut filtration algorithm so that whenever it encountered such a case, it merged the two PCs into one by removing the cut index in-between.

However, an interesting case that generated confusion with this definition was a 'meem' or a 'faa' followed by a 'daal' (فد ، مد). Both cases would always be misinterpreted as a 'saad' or 'daad' and the cut filtration algorithm would falsely merge the 'daal' and the preceding character into one.

Therefore, we defined a saad-stroke to differentiate between the 'daal' stroke and the strokes of 'saad' and 'daad'. The saad-stroke is a seen-stroke with the additional condition of being surrounded by cut indices that have at least one black pixel each. The goal of this extra specification is to ensure that the saad-stroke is connected to the baseline from both sides and is not followed by a space, as is the case with 'daal'. As such, the false cuts in 'saad' and 'daad' can be filtered when the cut filtration algorithm finds a hole followed by a saad-stroke or when it finds a hole followed by a bowl.

III. **'Baa', 'Taa', 'Thaa' and 'Faa' case:** There is also a difficult case where a 'baa', 'taa', 'thaa' or 'faa' (ب، ت، ث، ف) at the end of a word may cause a false cut. This occurs when the stroke at the end of the aforementioned characters is tall enough to intersect with the LMT. In this case, the ECC algorithm will generate a false cut at this stroke position which will need to be filtered.

In order to filter this extra false cut, we define an end stroke as a 'seen' stroke that has additional restrictions. Firstly, an end stroke must be followed by a cut index that does not intersect the baseline and preceded by a cut index that intersects the baseline. Furthermore, we locate the leftmost and the uppermost black pixels of the PC and we calculate the horizontal distance d between these two pixels. If d is measured to be less than or equal to 2 pixels, the ICF algorithm identifies the PC as an end stroke and removes the preceding cut.

It is worth noting that the extra restriction on the horizontal distance between the top-leftmost and the uppermost black pixels is essential in order not to incorrectly identify the 'daal' letter (ـد) as an end stroke because the stroke in the 'daal' letter has geometrical features that resemble the second stroke in the 'taa' and 'thaa' letters.

Algorithm 2. ICF Algorithm

```
Input: PCArray

FilteredCharacterArray = Φ
for all PC in PCArray do
    if PC is SeenStroke then
        if NextPC is SeenStroke or NextPC is SheenStroke then
            if AfterNextPC is SeenStroke or AfterNextPC is Bowl then
                    PCArray.Merge(PC, NextPC, AfterNextPC)
for all PC in PCArray do
    if PC is Saad/DaadStroke or PC is Bowl then
        PCArray.Merge(PreviousPC, PC)
for all PC in PCArray do
    if PC is EndStroke with D ≤ 2 then
        PCArray.Merge(PreviousPC, PC)
FilteredCharacterArray = PCArray

Output: FilteredCharacterArray
```

Although the previous cut filtration cases may seem largely independent of each other, the filtration order can greatly affect the character segmentation performance. For instance, executing the saad/daad-case before the seen/sheen-case may cause the algorithm to confuse the first stroke of the ‘seen’ letter as a ‘saad’ or a ‘daad’ stroke, and falsely merge it with the preceding character. As shown in Algorithm 2, our ICF algorithm filters all PCs according to the order of filtration cases mentioned in this paper.

It is worth mentioning that our character segmentation algorithm does not depend on any linguistic or statistical patterns in the Arabic language and is well-equipped to segment any sequence of Arabic letters. The previous work of [8] relied on their cut filtration algorithm’s assumption that some letters such as ‘saad’ and ‘daad’ are never followed by ‘seen’ or ‘sheen’. On the other hand, our ICF algorithm is general enough to segment any Arabic word regardless of the arrangement of the letters in the word.

Our character segmentation algorithm provides several improvements over the work of [8]. As opposed to the algorithm proposed by [8], the ECC algorithm does not generate a cut at positions where there are black pixels above or below the baseline, and this improvement reduces the number of false cuts that the cut filtration algorithm has to filter. The second improvement in our ICF is filtering each PC based on clear and specific structural features of Arabic letters such as ‘seen-stroke’, ‘sheen-stroke’, ‘saad-stroke’, ‘end-stroke’, ‘bowl’ and ‘hole’. These features are essential for the ICF algorithm in order to not accidentally remove any valid cut. Previous work in character segmentation [8] did not provide a correct exact definition of a bowl and, as a result, the characters having a bowl-shape in their structure such as ‘noon’, ‘qaaf’, ‘yaa’, ‘raa’, and ‘zaay’ letters (ن، ق، ي، ر، ز) were confused with the bowl part of the ‘saad’ and ‘daad’ letters (ص، ض) and were falsely merged with the preceding character.

The third improvement is filtering the false cuts in the case of the letters ‘baa’, ‘taa’, ‘thaa’, and ‘faa’ based on the leftmost black pixel and the uppermost black pixel instead of the top-leftmost black pixel only. The aforementioned false cuts are detected in [8]’s algorithm if the top-leftmost black pixel has a relatively small height when compared to the highest black pixel in the line. As a result, almost all letters that occur at the end of a word will be falsely merged with the preceding character. Only the ‘alef’ (أ) character will not be falsely merged when written at the end of a word since its top-leftmost black pixel is relatively tall, while every other character in

the Arabic alphabet will have a relatively low top-leftmost black pixel when written at the end of a word.

Finally, our segmentation algorithm solves the challenging case of the ‘seen’ (س) and ‘sheen’ (ش) letters at the end of the word, which was not handled by the algorithm proposed by [8] and resulted in a considerable number of incorrect segmentations. We conclude that our algorithm improves on the one proposed by Qaroush et al. [8] and addresses many of its shortcomings in the character segmentation problem.

3.3.3. Character Recognition Model

We propose feeding the images of the segmented characters to an artificial neural network since this will yield better performance than choosing features manually. The architecture of the ANN used in this research is a multilayered feed-forward network architecture with four layers. This neural network learns highly complex nonlinear features by training on a sufficiently large training set of Arabic characters together with their ground-truth labels.

We will generate our own training set by comparing the number of segmented letters generated from a word-using our proposed algorithms- with the number of letters in the ground truth word. If the numbers match, we associate each letter with its corresponding label. If the numbers do not match, then we skip and discard this word. It is worth noting that this over/under-segmentation problem arises if the scanned document was too noisy or blurry. However, we are cautious about the training set and prefer not to risk accidentally training with false characters.

We begin by resizing all the images generated from the character segmentation algorithm to be 24x24 pixels and then flattening them to be 576-dimensional vectors. In addition, we perform dimensionality reduction on these 576-dimensional vectors using Incremental Principal Component Analysis (IPCA), which -as illustrated in Figure 4- can represent the original vectors using only 200 principal components while retaining 99% of the total variance in the data. These 200-dimensional vectors are then fed to the neural network that consists of two hidden layers and a softmax output layer of sizes 150, 70, and 29 respectively. The softmax output layer assigns a likelihood value for each character of the 29 characters. This value represents the probability that this character is the correct classification for the input image. The inference phase of the model then classifies the input character as the character with the highest likelihood of being the correct prediction.

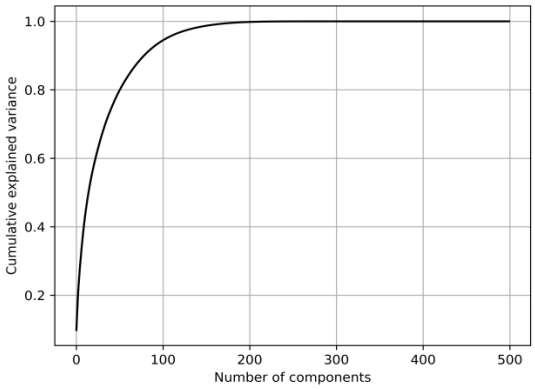


Figure 4. PCA Dimensionality Graph

We trained the model with our generated dataset that consists of 1,200,000 images of perfectly segmented Arabic letters in Naskh script using mini-batch training of batch size 8192 for 80 epochs. Categorical cross-entropy loss is calculated in the output layer and optimized using Adam optimizer [22] with hyper-parameters β1 and β2 being 0.9 and 0.999 respectively. For the hidden layers, the Rectified Linear Unit (ReLU) is used as a non-linear activation function followed by a 10% dropout layer to ensure elimination of overfitting as much as possible. A learning rate of 0.001 with no decay factor is used. The network is initialized using He initialization [23] and shuffled per epoch. We used a validation set of 12,000 images, representing 1% of the training set.

3.3.4. Post-processing

The character recognition model outputs a predicted class for each of the character images. However, a final step of post-processing is necessary to aggregate these characters into words and separate them with spaces to generate a meaningful text document. The previously predicted letters are produced consecutively with no spaces until we encounter an activated End of Word (EOW) flag. Every character has an EOW flag value; values for this flag are obtained in the character segmentation phase by setting the EOW flag to true (activated) for the final segment of the word that is being segmented. In the post-processing step, whenever we encounter a character with an activated EOW flag, indicating the end of a word, we place a space directly after this character. This step ensures the production of a comprehensible document.

4. EXPERIMENTAL RESULTS

4.1. Datasets

There are several datasets for Arabic character recognition [24]. We used the open dataset AlWatan corpus [25] for training our neural network. The dataset includes very large Arabic vocabulary, with different font types, sizes, and styles and is rich with separable characters, overlapping characters, and ligatures. The scanned images in the dataset are 72 dpi resolution images. For training the model, we randomly selected 550 documents/images of plain Arabic Naskh from different topics, approximately 282,000 words, or 1,200,000 characters. For validation and testing, we randomly chose 6 and 10 documents/images of Arabic Naskh script as a validation set and a test set respectively, of sizes 3300 words (12000 characters) and 5500 words (100,500 characters).

We further experimented with the system with different test sets of plain Arabic fonts; Naskh, Transparent Arabic, Simplified Arabic, M. Unicode Sara, Tahoma, Times New Roman, and Arial with different font sizes; 10, 12, 14, and 16. We tested the system on the APTI dataset (Arabic Printed Text Image) [26], which is a standard benchmarking dataset for Arabic OCR tasks. We used Keras [27] for training our model and we ran our experiments on a core i7 5820K 3.3 GHz machine, with 32 GB RAM, Ubuntu OS 16.04, and GPU NVIDIA RTX 2070 with 8 GB memory and 2560 cores.

4.2. Results and Evaluation

This section evaluates our word segmentation algorithm, character segmentation algorithm, character recognition model, as well as the overall system accuracy. We used both Watan-2004 and a subset of the APTI datasets for evaluating our system. We also compare our segmentation algorithms to the work of Anwar et al. [28], Radwan et al. [29], and Mousa et al. [30]. We

compare the overall system performance when our improved segmentation algorithm is used against when the segmentation algorithm of Qaroush et al. [8] is used.

4.2.1. Word Segmentation

We define the word segmentation accuracy as the number of correctly segmented words divided by the total number of actual words in the document. Our method achieves an average word segmentation accuracy of 99.94% for different fonts, as indicated in Table 1, where we also compare our results for each font with Qaroush et al. [8].

Table 1. Word Segmentation Comparison with Qaroush et al. [8]

Font	Method Proposed		Qaroush et al. [8]	
	Input Words	Accuracy	Input Words	Accuracy
Tahoma	25,928	99.96%	2,319	99.4%
Naskh	29,169	99.95%	2,921	96.1%
Simplified Arabic	22,687	99.94%	2,884	98.8%
Transparent Arabic	21,055	99.90%	2,860	99.1%

4.2.2. Character Segmentation

We first segment each word and compare the number of segmented characters with the number of actual characters in the word. Then we count their absolute difference as incorrectly segmented characters. We finally define the character segmentation accuracy as the total number of correctly segmented characters divided by the total number of actual characters. Our method achieves an average character segmentation accuracy of 98.23% for different fonts, as indicated in Table 2, where we also compare our results for each font with [8]. Comparing our results with the results of [28] and [30], as shown in Table 3, we note that our system outperforms all other character segmentation algorithms in the character segmentation task.

Table 2. Character Segmentation Comparison with Qaroush et al. [8]

Font	Method Proposed		Qaroush et al. [8]	
	Input Characters	Accuracy	Input Characters	Accuracy
Tahoma	114,080	97.80%	12,262	97.00%
Naskh	131,260	98.66%	12,585	94.52%
Simplified Arabic	100,957	99.06%	13,572	96.10%
Transparent Arabic	89,483	97.40%	13,120	96.26%

Table 3. Character Segmentation Results

Method Tested	Dataset	Font Sizes	Font Styles	Segmentation Accuracy
Anwar et al. [28]	Self-Generated	70pt	Traditional Arabic	97.55%
Mousa et al. [30]	Self-Generated	Not reported	Not reported	98.00%
Qaroush et al. [8]	[26]: 100,000 characters	10, 12, 14, 16, 18, and 24	Font Set A*	97.50%
Method Proposed	[25] + [26]: 1,000,000+characters	10, 12, 14, and 16	Font Set B†	98.23%

*Font Set A: Naskh, Transparent Arabic, Simplified Arabic, M Unicode Sara, Tahoma, and Advertising Bold

†Font Set B: Font Set A, Andalus, Diwani, Thuluth

4.2.3. Overall System Performance

The neural network achieved average recognition accuracy of 99.89%. The overall system accuracy is measured by calculating the Levenshtein edit distance between the generated document and the actual document. We show that our system achieves an overall accuracy of 97.94%. Therefore, and to the best of our knowledge, we propose that our Arabic OCR system is superior to all other segmentation based Arabic OCR in terms of accuracy and running time. It is also worth mentioning that our proposed system was developed and trained using a very large dataset, which is rich with Arabic text in different font types and sizes. Table 4 shows the evaluation of our system in comparison to [31]'s system and [8]'s segmentation method followed by an ANN for recognition.

Table 4. Overall System Evaluation

Method Tested	Dataset	Number of Words	System Accuracy	Avg. Run Time / 550 Words (sec)
Qaroush et al. [8] + ANN	[25]	3300	94.95%	3.42
Touj et al. [31]	1500 words	1500	97.00%	NA
Method Proposed	[25] + [26]	3300	97.94%	1.49

5. FUTURE WORK

Much like other OCR systems, our system's performance decreases when operating on documents with high noise. This creates the demand to work on more elaborate preprocessing methods in the future without sabotaging our system's remarkable runtime. Also, our recognition model is currently limited to the fonts that it inferred from the training set; as a result of that, we aim to enrich the training set to contain more fonts and possibly all fonts without ligatures.
Apart from the low-level additions to our system, potential work on this system would include integrating it into larger applications such as image-speech systems. Such applications -where instant results are crucial- will be a perfect fit for our method because of the very low runtime we provide.

6. CONCLUSION

Arabic Optical Character Recognition introduces many challenges in the character segmentation and recognition phase. This paper proposes a complete language-independent Arabic OCR pipeline with an improved character segmentation algorithm based on word-level features and a bio-inspired character recognition model based on neural networks. We proposed a highly accurate and efficient system. The overall system architecture consists of: a simple yet effective pre-processing module, an enhanced reliable module for character segmentation, an artificial neural network for the recognition of segmented characters, and finally a post-processing module that formulates the output of our system into a digitized document. We evaluated our system on different datasets with high variability in font types and sizes. The experimental results show that our system outperforms the current state-of-the-art algorithms for word segmentation, character segmentation, and character recognition. We evaluated the overall performance of the system and concluded that our system achieves outstanding results in accuracy and running time.

REFERENCES

[1] M. Namysl and I. Konya, "Efficient, lexicon-free ocr using deep learning," arXiv preprint arXiv:1906.01969, 2019.

[2] S. Dixit, M. Bharath, Y. Amith, M. Goutham, K. Ayappa, and D. Harshitha, "Optical recognition of digital characters using machine learning," International Journal of Research Studies in Computer Science and Engineering, vol. 5, no. 1, pp. 9–16, 2018.

[3] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multidigit number recognition from street view imagery using deep convolutional neural networks," arXiv preprint arXiv:1312.6082, 2013.

[4] K. Mohammad, A. Qaroush, M. Ayesh, M. Washha, A. Alsadeh, and S. Agaian, "Contour-based character segmentation for printed arabic text with diacritics," Journal of Electronic Imaging, vol. 28, no. 4, p. 043030, 2019.

[5] N. El Makhfi and O. El Bannay, "Scale-space approach for character segmentation in scanned images of arabic documents," Journal of Theoretical and Applied Information Technology, vol. 94, no. 2, p. 444, 2016.

[6] J. H. AlKhateeb, J. Ren, J. Jiang, and H. Al-Muhtaseb, "Offline handwritten arabic cursive text recognition using hidden markov models and re-ranking," Pattern Recognition Letters, vol. 32, no. 8, pp. 1081– 1088, 2011.

[7] M. A. Radwan, M. I. Khalil, and H. M. Abbas, "Neural networks pipeline for offline machine printed arabic ocr," Neural Processing Letters, vol. 48, no. 2, pp. 769–787, 2018.

[8] A. Qaroush, B. Jaber, K. Mohammad, M. Washaha, E. Maali, and N. Nayef, "An efficient, font independent word and character segmentation algorithm for printed arabic text," Journal of King Saud University- Computer and Information Sciences, 2019.

[9] L. Zheng, A. H. Hassin, and X. Tang, "A new algorithm for machine printed arabic character segmentation," Pattern Recognition Letters, vol. 25, no. 15, pp. 1723–1729, 2004.

[10] M. Amara, K. Zidi, K. Ghedira, and S. Zidi, "New rules to enhance the performances of histogram projection for segmenting small-sized Arabic words," in International Conference on Hybrid Intelligent Systems. Springer, 2016, pp. 167–176.

[11] M. Rashad and N. A. Semary, "Isolated printed arabic character recognition using knn and random forest tree classifiers," in International Conference on Advanced Machine Learning Technologies and Applications. Springer, 2014, pp. 11–17.

[12] S. F. Rashid, M.-P. Schambach, J. Rottland, and S. von der N¨ull, "Low resolution Arabic recognition with multidimensional recurrent neural networks," in Proceedings of the 4th International Workshop on Multilingual OCR, 2013, pp. 1–5.

[13] M. Jain, M. Mathew, and C. Jawahar, "Unconstrained scene text and video text recognition for arabic script," in 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR). IEEE, 2017, pp. 26–30.

[14] M. Dahi, N. A. Semary, and M. M. Hadhoud, "Primitive printed arabic optical character recognition using statistical features," in 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS). IEEE, 2015, pp. 567–571.

[15] A. Rosenberg and N. Dershowitz, Using SIFT descriptors for OCR of printed Arabic. Tel Aviv University, 2012.

[16] M. A. Shayegan and S. Aghabozorgi, "A new dataset size reduction approach for pca-based classification in ocr application," Mathematical Problems in Engineering, vol. 2014, 2014.

[17] A. A. Shahin, "Printed arabic text recognition using linear and nonlinear regression," arXiv preprint arXiv:1702.01444, 2017.

[18] F. Nashwan, M. A. Rashwan, H. M. Al-Barhamtoshy, S. M. Abdou, and A. M. Moussa, "A holistic technique for an arabic ocr system," Journal of Imaging, vol. 4, no. 1, p. 6, 2018.

[19] M. Rashwan, M. Fakhr, M. Attia, and M. El-Mahallawy, "Arabic ocr system analogous to hmm-based asr systems; implementation and evaluation," Journal of Engineering and Applied Science-Cairo, vol. 54, no. 6, p. 653, 2007.

[20] N. B. Amor and N. E. B. Amara, "Multifont arabic characters recognition using houghtransform and hmm/ann classification." Journal of multimedia, vol. 1, no. 2, pp. 50–54, 2006.

[21] O. Al-Jarrah, S. Al-Kiswany, B. Al-Gharaibeh, M. Fraiwan, and H. Khasawneh, "A new algorithm for arabic optical character recognition." WSEAS Transactions on Information Science and Applications, vol. 3, no. 4, pp. 832–845, 2006.

[22] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," arXiv preprint arXiv:1412.6980, 2014.
[23] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 1026–1034.
[24] A. Lawgali, M. Angelova, and A. Bouridane, "Hacdb: Handwritten Arabic characters database for automatic character recognition," in European Workshop on Visual Information Processing (EUVIP). IEEE, 2013, pp. 255–259.
[25] M. Abbas, "Alwatan," https://sites.google.com/site/mouradabbas9/corpora, 2004.
[26] F. Slimane, R. Ingold, S. Kanoun, A. M. Alimi, and J. Hennebert, "A new arabic printed text image database and evaluation protocols," in 2009 10th International Conference on Document Analysis and Recognition. IEEE, 2009, pp. 946–950.
[27] F. Chollet et al., "Keras," https://keras.io, 2015.
[28] K. Anwar, H. Nugroho et al., "A segmentation scheme of arabic words with harakat," in 2015 IEEE International Conference on Communication, Networks and Satellite (COMNESTAT). IEEE, 2015, pp. 111–114.
[29] M. A. Radwan, M. I. Khalil, and H. M. Abbas, "Predictive segmentation using multichannel neural networks in arabic ocr system," in IAPR Workshop on Artificial Neural Networks in Pattern Recognition. Springer, 2016, pp. 233–245.
[30] M. A. Mousa, M. S. Sayed, and M. I. Abdalla, "Arabic character segmentation using projection based approach with profile's amplitude filter," arXiv preprint arXiv:1707.00800, 2017.
[31] S. Touj, N. E. B. Amara, and H. Amiri, "Generalized hough transform for arabic printed optical character recognition." Int. Arab J. Inf. Technol., vol. 2, no. 4, pp. 326–333, 2005.