A Segmentation Free Approach to Arabic and Urdu OCR

Nazly Sabbour[1] and Faisal Shafait[2]

[1]Department of Computer Science, German University in Cairo (GUC), Cairo, Egypt;
[2]German Research Center for Artificial Intelligence (DFKI), Kaiserslautern, Germany

ABSTRACT

In this paper, we present a generic Optical Character Recognition system for Arabic script languages called Nabocr. Nabocr uses OCR approaches specific for Arabic script recognition. Performing recognition on Arabic script text is relatively more difficult than Latin text due to the nature of Arabic script, which is cursive and context sensitive. Moreover, Arabic script has different writing styles that vary in complexity. Nabocr is initially trained to recognize both Urdu Nastaleeq and Arabic Naskh fonts. However, it can be trained by users to be used for other Arabic script languages. We have evaluated our system's performance for both Urdu and Arabic. In order to evaluate Urdu recognition, we have generated a dataset of Urdu text called UPTI (Urdu Printed Text Image Database), which measures different aspects of a recognition system. The performance of our system for Urdu clean text is 91%. For Arabic clean text, the performance is 86%. Moreover, we have compared the performance of our system against Tesseract's newly released Arabic recognition, and the performance of both systems on clean images is almost the same.

Keywords: Character recognition, Arabic script, Urdu Nastaleeq

1. INTRODUCTION

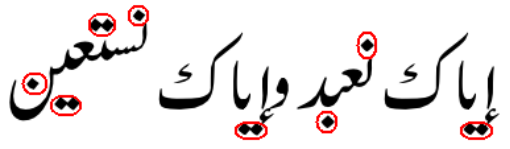
There has been a lot of research in the field of OCR which mainly focuses on character recognition for Latin languages. However, research in Arabic script OCR has started relatively later. A lot of studies have been done in order to investigate the challenges associated with recognition for Arabic script languages, and to suggest approaches that take into account these challenges. Most of the research done in Arabic script OCR is mainly for the Arabic language, such as Badr et al.,[1] Khorsheed,[2] and Cheung et al.[3] However, research for other Arabic script languages such as Persian and Urdu appeared much later, and is even more limited. Research for Urdu language recognition has recently started to grow, e.g., Pal et al.[4] and Hussain et al.[5] As for the available OCR products, a small number of commercial systems provide recognition for some Arabic script languages. On the other hand, free OCR systems mainly provide recognition for Latin languages, such as Ocropus, Ocrad, etc.

In this paper, we present an OCR application for Arabic script languages called Nabocr[*]. The application is initially trained to be used for the recognition of both Urdu and Arabic languages. Moreover, we have generalized the application to handle any other Arabic script language. This is done by allowing the user to train the system from scratch giving it as input sufficient text for this language. In order to evaluate the performance of our recognition system, we have generated a dataset called UPTI (Urdu Printed Text Image Database). UPTI consists of different versions that measure different aspects of the system. It provides versions that measure the accuracy of both line and ligature recognition. Moreover, it contains versions of degraded text images which aim at measuring the robustness of a recognition system against possible image defects, such as, jitter, thresholding, elastic elongation, and sensitivity.

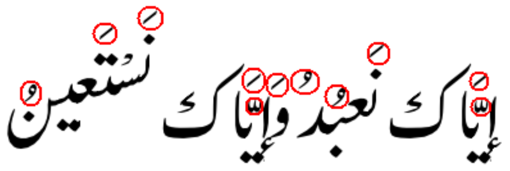
The structure of this paper is organized as follows. In Section 2, we discuss the challenges faced when developing a recognition system for Arabic script languages. In Section 3, we discuss the approaches proposed for implementing our application, Nabocr. In Section 4, experiments for evaluating th performance of our recognition system are presented for both Urdu and Arabic languages. Finally, we provide a conclusion of the work done and recommendations for future work.

Further author information:
Nazly Sabbour: E-mail: nazly.fayek@guc.edu.eg
Dr. Faisal Shafait: E-mail: faisal.shafait@dfki.de

[*]The current version of Nabocr can be found in http://nabocr.com/.



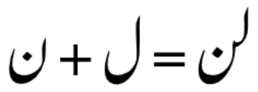
(a) Dots highlighted in sample Urdu text



(b) Diacritics highlighted in sample Urdu text

Figure 1: Dots and Diacritics associated with Arabic letters

Figure 2: Ligature shape composed of two constituent letters

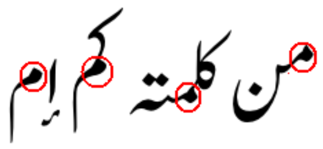


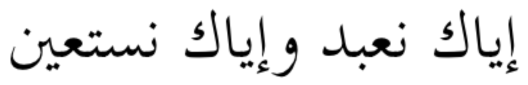
2. CHALLENGES OF ARABIC SCRIPT OCR

Arabic Script is the second most widely used writing script after Latin.[6] It is used by over 400 million people. It is based on an ancient alphabet called Nabataean alphabet. Many languages are based on this script, e.g., Arabic , Persian, Urdu, Kurdish, etc. Arabic letters are written from right to left, while the numbers are written from left to right. A lot of challenges are faced when developing an OCR system for Arabic script because of the nature of Arabic script languages which will be further discussed in this section.

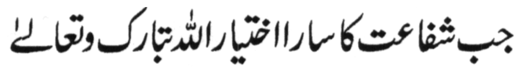
Dots and Diacritics: Some Arabic letters have dots and diacritics associated to them as shown in Figures 1a and 1b. A dot is considered as a main part of the letter. Dots are generally used to differentiate between letters that have the same main body. Diacritics are used to identify a certain sound to be used when pronouncing the letter. Since dots and diacritics are not directly connected to the letter they are associated with, this could possibly cause problems when trying to associate the components that belong to the same letter in order to perform recognition.
Cursive Nature: Arabic script is considered to be a cursive script, where letters sometimes join together in a word to form a single connected shape as shown in Figure 2. The shape formed by the connected letters is called a *Ligature*, which is basically a subword. An Arabic word can be formed of a single ligature or possibly more.
Context Sensitivity: As a result of the cursive nature of the Arabic script languages, a letter can occur in four possible positions (beginning, middle, end, or isolated). According to the context of the letter, each position will have a different shape as shown in Figure 3.
Script Specific Challenges: There are different writing styles for Arabic script languages, such as Naskh, Nastalique, Batol, Jaben, Taleeq, etc. The two most common writing styles in Arabic script languages are Naskh and Nastaleeq. Naskh is widely used in Arabic text books, while Nastaleeq is widely used in Urdu newspapers and novels. Figure 4 shows the difference between the two writing styles. Nastaleeq appears to be more complicated in writing than Naskh. In Nastaleeq font, ligatures can horizontally overlap each other which can lead to problems when segmenting the word into ligatures. Moreover, Nastaleeq has no spacing between the

Figure 3: Different forms of the same letter *Meem*





(a) Naskh



(b) Nastaleeq

Figure 4: Common writing styles for Arabic script

words that can lead to word segmentation errors.

3. ARABIC SCRIPT OCR: NABOCR

The general framework of our approach as shown in Figure 5 consists of three main parts:

- **Training** which takes as input raw Arabic script data as text files. The training part outputs a dataset of ligatures, where each ligature is described by a feature vector.
- **Recognition** which takes as input an image specified by the user. It uses the dataset of ligatures generated from the training part to convert the image into text.
- **User Interface** which allows users to edit the text result of the recognition output. Moreover, users can modify the dataset of ligatures being used by the Recognition part.

3.1 Training Phase

The main goal of the training phase is to prepare the application to be used for text recognition for a certain Arabic script language. Training phase consists of two main steps:

Figure 5: General Framework of Nabocr

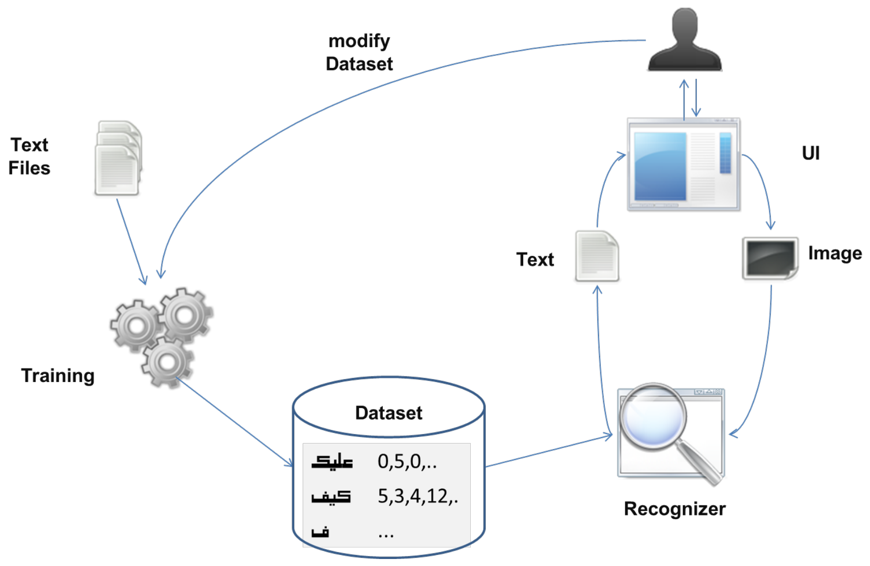
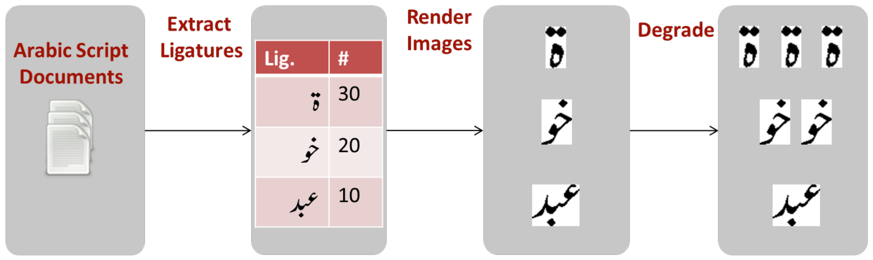


Figure 6: Steps of Dataset Generation



1. Generation of a dataset of images for the possible ligatures of the Arabic script language to be used by the application.
2. Extracting features that describe each ligature in the dataset generated by the previous step.

3.1.1 Ligature Dataset Generation

In order to generate a dataset of ligatures, we use as input sufficient text data written in a given Arabic script language. The following steps are done as shown in Figure 6:

1. Ligatures are identified and extracted from the input text along with their frequency of occurrence in the text. This is done using orthographic rules of Arabic script that determine the delimiters of the ligatures to be extracted.
2. Synthetic images for each extracted ligature are rendered using a rendering engine.
3. Degraded versions are generated for each ligature image, and added to the dataset of synthetic images.

In order for the system to be able to handle input images with noise, the system is trained on degraded images as well as clean images. We have used four degradation models proposed by Baird[7] to generate degraded images from synthetic images. These models are used to describe the following local imaging defects: elastic elongation, jittering, sensitivity, and thresholding. The number of degraded samples generated for each ligature depends on how frequent this ligature is in the language. The more frequent a ligature is, in other words the more probable a ligature is to occur, the more degraded samples it will have in the dataset.

Our system is initially trained to recognize both Urdu Nastaleeq and Arabic Naskh fonts. We have generated datasets for each language using Urdu and Arabic books available online as text files. The number of ligatures extracted for Urdu is over 10,000 ligature having different sizes. As for Arabic ligatures, over 20,000 ligatures are extracted from the Arabic text used. Using the above steps, images are generated for the ligatures extracted in Nastaleeq and Naskh fonts.

3.1.2 Feature Extraction

The second main step of the training phase is to extract a feature vector representing each ligature image included in the generated dataset. For this purpose, the following steps are done as shown in Figure 7:

1. Normalize each image to a fixed width and height. The rescaling process is done in a way that keeps the aspect ratio between the width and the height of the ligature image. Therefore, the shape of the ligature in the output normalized image is not affected.
2. Extract contour (boundary) of the ligatures in each sample image.
3. Describe each ligature shape image using a shape descriptor as a feature vector. The shape descriptor we used is called *Shape Context*, which will be further explained in this section.

Figure 7: Feature Extraction in Nabocr

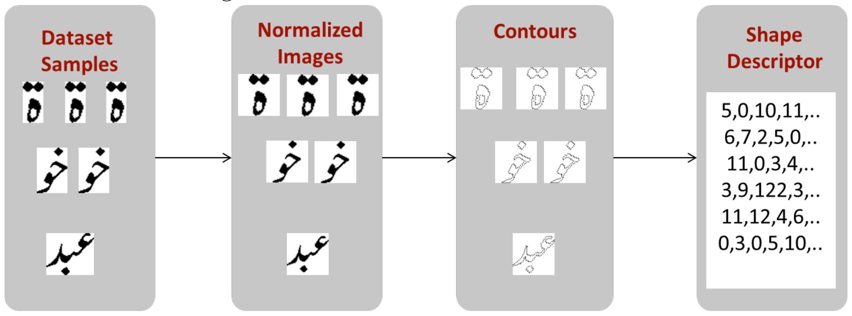
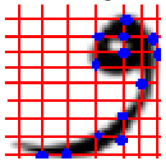


Figure 8: Extracting contour points



Contour Extraction

The contour of a ligature is extracted using an approach proposed by Hassan et al.[8] The approach proposes extracting the contour by first applying a logical grid to the shape image as shown in Figure 8. Secondly, transition points along the grid lines are considered as contour points. Transition points are extracted by detecting points on the grid lines at which the pixels' intensity values transition from black to white, or from white to black. The system takes as input the normalized binary image of a ligature and performs the following steps:

1. Calculate a matrix containing absolute difference between every two pixels in the image horizontally.

$$H_{ij} = |I(i,j) - I(i+1,j)| \tag{1}$$

2. Calculate a matrix containing absolute difference between every two pixels vertically.

$$V_{ij} = |I(i,j) - I(i,j+1)| \tag{2}$$

3. Performing logical OR operation on both matrices.

$$R_{ij} = H_{ij} \text{ OR } V_{ij} \tag{3}$$

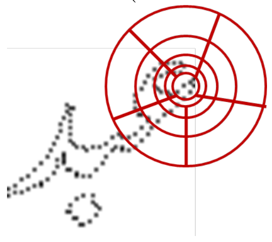
4. A postprocessing step: traversing the contour points after extraction and removing points that are relatively close to each other.

Shape Context

The final step of the training phase is to represent each ligature from the dataset using a feature vector. For this purpose, we use a shape descriptor called *Shape Context*. Shape context is a shape descriptor presented by Belongie et al.,[9] which is used to perform shape matching and object recognition.

In this approach, the object shape is represented as a set of points from its contour. Points can belong to the internal or extrenal contour of the shape being described.

Figure 9: Point Histogram of 16 bins (5 orientation bins and 4 distance bins)



$$P = \{P_i\} \tag{4}$$

Each point P_i is described by a log polar histogram H_i that relates P_i to its surrounding points from the contour of the shape. The origin of a histogram is centered at the point it is describing as shown in Figure 9. It divides the space around it into paritions called *bins*. Each bin is identified by two parameters: distance from the centre point, and the orientation relative to the centre point.

The histogram H_i of a point P_i is considered to be its shape context. It counts the number of surrounding points in each bin using equation 5.

$$H_i(k) = \#[q \neq P_i | q - P_i \in bin(k)] \tag{5}$$

The following steps are done to calculate the shape context of the points describing the contour of an object shape:

- Measuring the Euclidian distance between every two contour points to form a matrix D, where D_{ij} is equal to the distance between points P_i and P_j.
- Distance normalization is done to make the histogram invariant to any scaling resulting in a new matrix $norm(D)$.
- Quantize the normalized distances to a fixed number of distance bins ($rbins$).

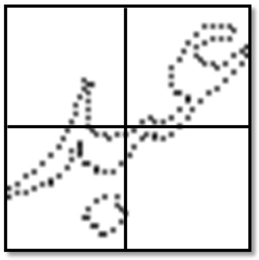
$$quant(D_{ij}) = \sum_{0<k<|rbins|} norm(D_{ij}) < rbin(k) \tag{6}$$

  where given a fixed number of distance bins (rbins), the equation calculates the distance bin at which point P_j occurs relative to P_i.
- Measuring angle between contour points to form a matrix A, where A_{ij} is equal to the angle between points P_i and P_j.
- Angle normalization is done by converting each angle measure to be within the scale from 0 to 2π to form a new matrix $norm(A)$.
- Quantize the resulting normalized angles to a fixed number of orientation bins ($tbins$).

$$quant(A_{ij}) = 1 + floor(norm(A_{ij}) \times |tbins|)/2\pi) \tag{7}$$

  where given a fixed number of orientation bins (tbins), the equation calculates the orientation bin at which point P_j occurs relative to P_i.
- Using both $quant(D)$ and $quant(A)$, we can formulate a histogram for each point using equation 5.

Figure 10: Dividing Ligature into 4 regions



The previous approach calculates the shape context of each point in the contour of a ligature. However, in order to describe the whole ligature, we use an approach proposed by Hassan et al.[10] which is as follows:

- Dividing the ligature into regions as shown in Figure 10.
- Calculating the shape contexts of points in each region separately.
- Summing up the shape contexts of the points in each region to form a histogram describing each region.
- Concatenating the region histograms calculated to form a histogram describing the ligature.

3.2 Recognition Phase

The recognition part takes as input an image which is specified by the user through the user interface. Its main task is to recognize any text that occurs in the input image. The recognized text is presented as an output to the user in an editable format. The recognition of the text in an input image is done using the following steps as shown in Figure 11:

1. Segment page image into lines.
2. Segment each line extracted from the previous step into ligatures.
3. Describing each unkown ligature image using a shape descriptor as explained in the training phase.
4. Classify each unknown ligature to a ligature from the dataset generated in the training phase using k-Nearest Neighbor.
5. Recognized ligatures from the classification step form the output editable text.

3.2.1 Page Segmentation

In our application, we have used a simple approach called *horizontal projection* assuming one column pages. In this approach, we count the number of black pixels in each row in the page, and then we cut at parts having low horizontal projection. However, this may cause some problems when applying it to Arabic script documents. As shown in Figure 12, dots and diacritics in Arabic text may occur slightly above the text line, which may lead to cutting into the line itself and having false lines detected. A possible solution to this problem is to perform an extra step after the extraction of lines using horizontal projection in order to detect any possible false lines. To detect false lines, the heights of the lines extracted are checked. If a line has a relatively small height compared to the other lines, then it is considered as a false line and it is then merged with the nearest line detected.

Figure 11: Recognition steps in Nabocr

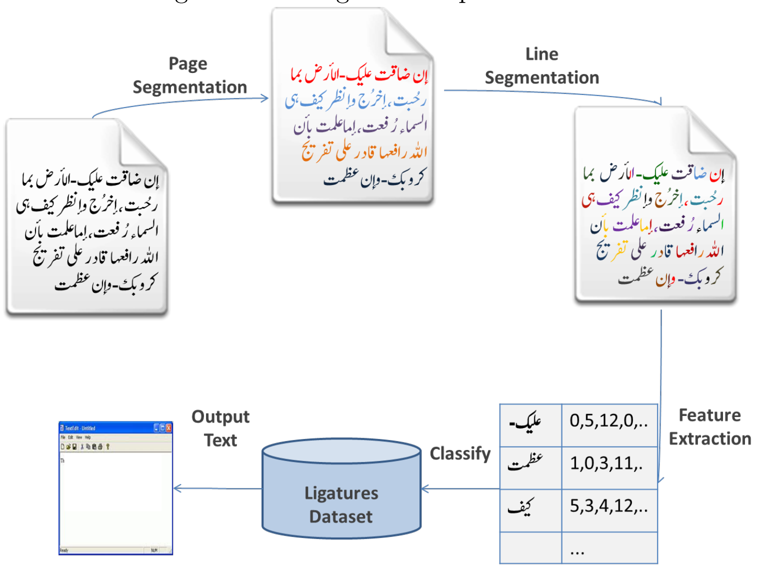
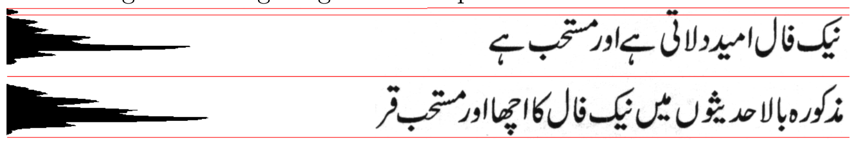


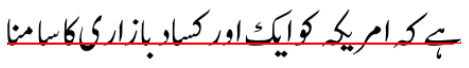
Figure 12: Page Segmentation problems due to diacritics



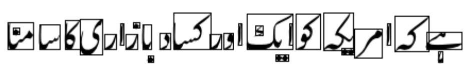
3.3 Line Segmentation

In order to segment each line into its constituent ligatures, we use an algorithm proposed by Javed et al.[11] Javed et al. proposed this algorithm specifically for Arabic script Nastaleeq lines, where given as input an image of a line, the algorithm performs the following steps:

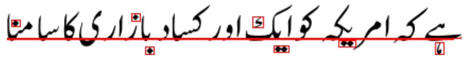
1. Estimating the baseline position by calculating the row which has the maximum horizontal projection as shown in Figure 13a.
2. Extract connected components in the text line as shown in Figure 13b.
3. Identify the diacritics and dots from the main bodies in the text line, where the dots or diacritics are considered as the components occuring above or below the baseline as illustrated in Figure 13c.
4. Associate each dot and diacritic to its corresponding ligature by calculating the horizontal span of each detected dot and diacritic onto the baseline.



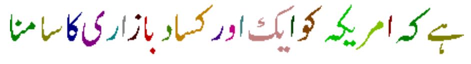
(a) Baseline Detection



(b) Extraction of Connected Components

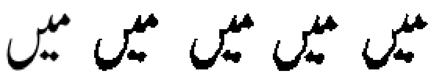


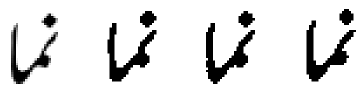
(c) Dots Detection

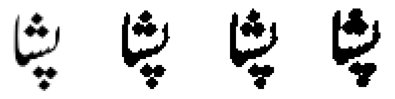


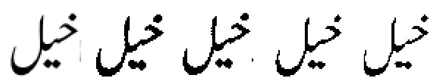
(d) Output of Line Segmentation

Figure 13: Line Segmentation Steps


(a) Elastic Elongation


(b) Jitter


(c) Sensitivity


(d) Thresholding
Figure 14: Degradation parameters applied to UPTI

After performing the above steps on the input line, the output of the line segmentation algorithm will be as demonstrated in Figure 13d, where each ligature is detected separately along with its dots and diacritics.

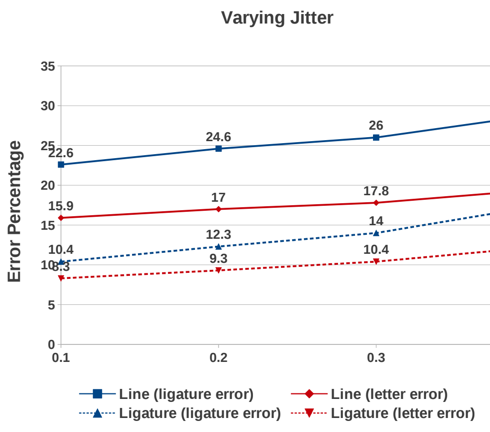
4. TESTS AND RESULTS

4.1 Urdu Evaluation

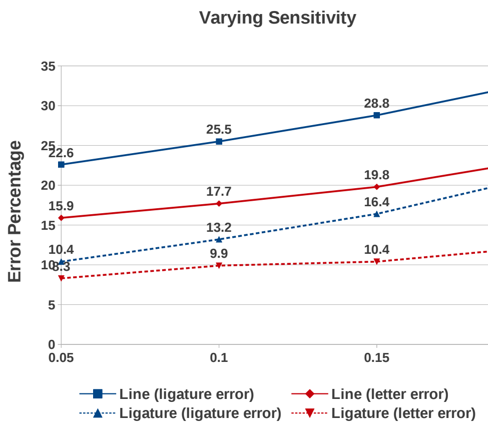
We have generated a dataset called UPTI (Urdu Printed Text Image Database) in analogy to APTI dataset, which is an online database for synthetic word images. APTI[12] dataset aims to measure the performance of different recognition systems for the Arabic language. UPTI dataset contains more than 10,000 synthetic images of Urdu text rendered in Nastaleeq font. The UPTI dataset consists of different versions, where each version measures different aspects of the recognition system. The different versions are as follows:

1. **Line level UPTI** which contains synthetic images for text lines.
2. **Ligature level UPTI** which contains synthetic images for text lines, where each ligature in the line is colored with a different color.
3. **Degraded UPTI** which contains degraded versions of both line and ligature level datasets. The degradation is done using four different degradation parameters, which are elastic elongation, sensitivity, jittering, and thresholding. Each degradation parameter is changed gradually while keeping the other degradation parameters at their default values as shown in Figure 14. The main aim of this process is to generate different degraded versions of UPTI and to measure the performance of the OCR recognition system against the variance of each degradation parameter.

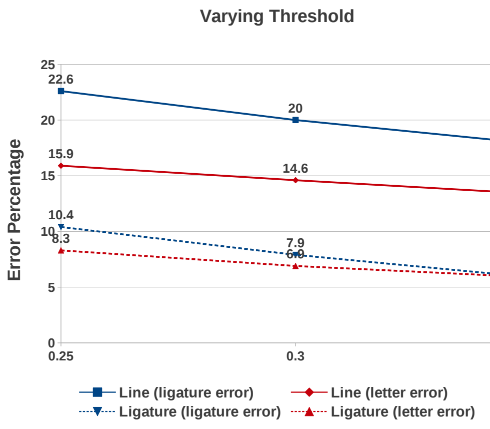
Performance Evaluation



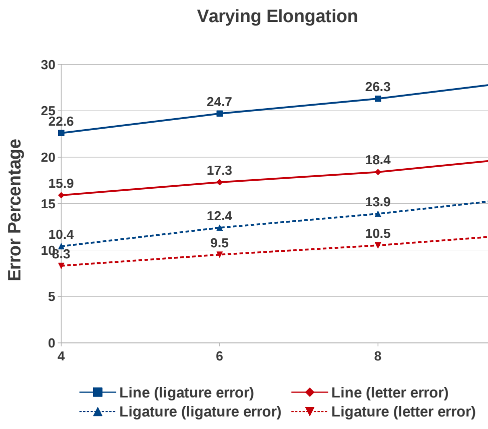
(a) Jitter



(b) Sensitivity



(c) Threshold



(d) Elastic Elongation

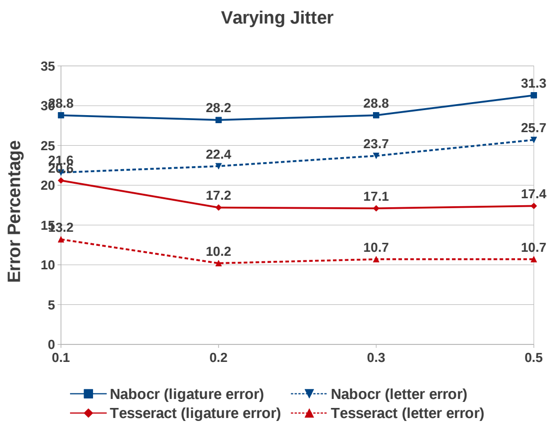
Figure 15: UPTI Evaluation Results for Line and Ligature level UPTI

In order to evaluate the recognition of Urdu text, we have initially run the recognition algorithm on the undegraded versions of the UPTI datasets. The results for the line level dataset gave a ligature error rate of 13.3%, and a letter error rate of 11.2%. However, if we ignored the unknown foreign symbols, such as the punctuation marks, and foreign numbers, which are not considered by the recognition system, the ligature error rate drops to 9.1%, where the letter error rate becomes 8.5%. When increasing the degradation effects on line level dataset images, recognition error rate changes as shown in Figure 15.

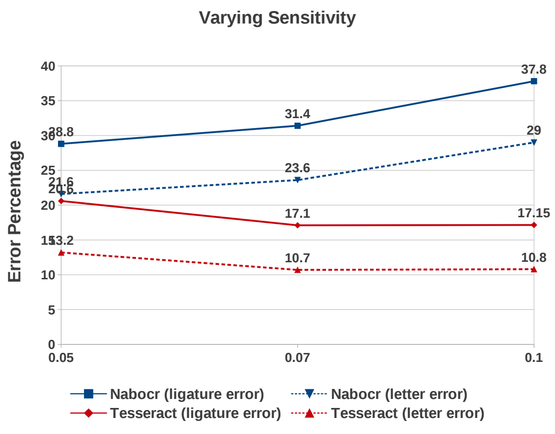
When measuring the error rate for the ligature level dataset , the error rate dropped significantly from 9.13% to 0.4% for the ligature error. Similarly, the letter error rate decreased from 8.48% to 2.6%. The decrease in the error rate from the line to ligature level is expected. For the line level dataset, we first use the line segmentation algorithm in order to know where the ligatures are, and then we perform recognition. However, the ligatures are already colored in the ligature level dataset. Therefore, we do not perform line segmentation and we only perform recognition on the ligatures. Therefore, the ligature level error rate does not include line segmentation errors, which reduce the error rate significantly. When increasing the degradation effects on ligature level dataset images, recognition error rate varies as shown in Figure 15.

4.2 Arabic Evaluation

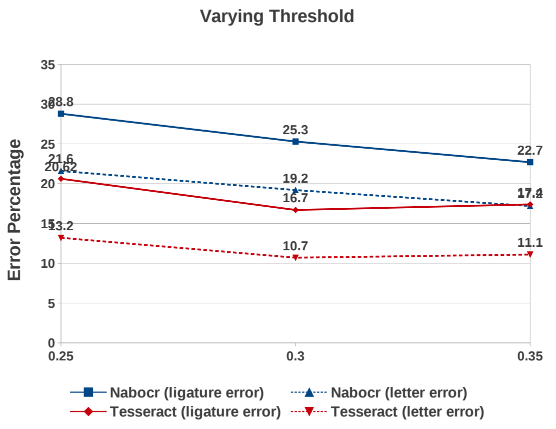
Our recognition system, Nabocr can be used to recognize any Arabic script language. Therefore, in order to evaluate the performance of the system, it is important to perform evaluation on more than one Arabic script



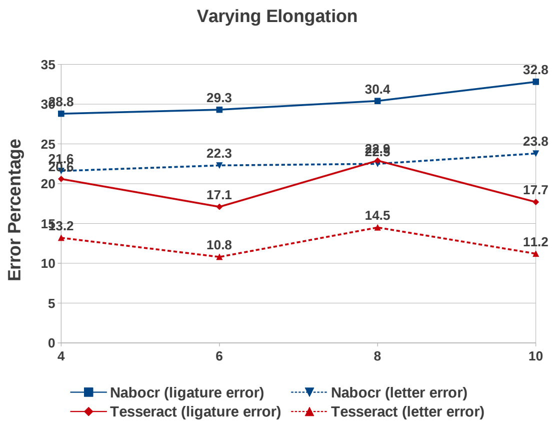
(a) Jitter



(b) Sensitivity



(c) Threshold



(d) Elastic Elongation

Figure 16: Evaluation results for Tesseract and Nabocr

language. We have used an online electronic book written in Arabic to measure the performance of Nabocr. The e-book used consists of 47 single column pages. In order to measure how robust the Arabic recognition is against the different degradation effects, we have generated degraded versions of the book by applying elastic elongation, sensitivity, jittering, and thersholding.

Performance Evaluation

One of the systems that has recently provided support for Arabic language recognition is Tesseract.[13] As a part of the evaluation process, we have run Tesseract's recognition engine on the Arabic e-book versions that were used for evaluating Arabic Nabocr, and we have compared the results of both systems. For undegraded versions of the book, both systems achieved nearly the same performance. In comparing the ligature error rate, Tesseract's error rate is 16.2%, while our system scored 16.1 %. When comparing the letter error rate, Tesseract's error rate is 10.1%, while Nabocr's error rate is 13.9%. On the other hand, when we test Tesseract on the different degraded versions of the e-book, Tesseract is relatively more robust than Nabocr with different degradation levels as shown in Figure 16.

5. CONCLUSION

In this paper, we have presented an OCR system for Arabic script recognition, called Nabocr. First, we have analyzed the different challenges faced when performing recognition for Arabic script languages. Secondly, we have explained the different approaches used in our system to overcome these challenges. The system is initially trained to recognize Urdu Nastaleeq script and Arabic Naskh script. Moreover, it can be further generalized to recognize other Arabic scripts. We have evaluated the performance of our application for both Urdu and Arabic recognition. In order to evaluate our system for Urdu recognition, we have generated a dataset called UPTI (Urdu Printed Text Image Database), which contains different versions that measure different aspects of a recognition system. For clean Urdu text images, the error rate for line recognition is 11.2%. The error rate for ligature recognition is 2.6%. Moreover, we have evaluated the effect of different degradation parameters, such as jittering, sensitivity, etc. on the performance of our system. As for Arabic recognition, we have evaluated the performance of our system by performing recognition on an e-book. Our system scored an error rate of 13.9%. Moreover, we have compared the performance of our Arabic recognition against the newly released version by Google's Tesseract. The error rate of both systems is nearly the same for clean images.

As for future work, we recommend several ways to enhance the approaches used by our system. In order to improve the recognition accurracy of Nabocr, we propose using a language model for each Arabic script language to take into consideration the probability of the sequence of ligatures being recognized. Improving line segmentation is highly recommended because of the gap between the error rates of both the line and ligature recognition of our system. Possible ways to improve line segmentation is to add a separate mechanism for dots/diacritics detection and detecting merged ligatures.

REFERENCES

[1] Al-Badr, B. and Haralick, R., "Segmentation-free word recognition with application to Arabic," in [*Proceedings of the Third International Conference on Document Analysis and Recognition*], 355–359 (1995).

[2] Khorsheed, M. S. M., "Automatic Recognition Of Words In Arabic Manuscripts," tech. rep. (2000).

[3] Cheung, M. B. A. and Bergmann, N. W., "An Arabic Optical Character Recognition System Using Recognition-Based Segmentation," *Pattern Recognition* (34), 215–233 (2001).

[4] Pal, U. and Sarkar, A., "Recognition of Printed Urdu Script," in [*Proceedings of 7th Int. Conf., on Document Analysis and Recognition*], 1183–1187 (2003).

[5] Husain, S. A. and Amin, S. H., "A Multi-tier Holistic Approach for Urdu Nastaliq recognition," in [*In IEEE Int. Multi-topic Conference, Karachi*], (2002).

[6] "Encyclopaedia Britannica online." http://www.britannica.com/EBchecked/topic/31666/Arabic-alphabet. Accessed: 25/05/2012.

[7] Baird, H. S., "Document image analysis," ch. Document image defect models, 315–325, IEEE Computer Society Press, Los Alamitos, CA, USA (1995).

[8] Hassan, E., Chaudhury, S., and Gopal, M., "Shape Descriptor Based Document Image Indexing and Symbol Recognition," in [*10th International Conference on Document Analysis and Recognition*], 206–210, IEEE Computer Society (2009).

[9] Belongie, S., Malik, J., and Puzicha, J., "Shape Matching and Object Recognition Using Shape Contexts.," *IEEE Trans. Pattern Anal. Mach. Intell.* **24**(4), 509–522 (2002).

[10] Hassan, E., Chaudhury, S., and Gopal, M., "Word shape descriptor-based document image indexing: a new DBH-based approach," *International Journal on Document Analysis and Recognition (IJDAR)* , 1–20 (2012).

[11] Javed, S.T., H. S., "Improving Nastalique Specific Pre-Recognition Process for Urdu OCR," in [*13th IEEE International Multitopic Conference 2009 (INMIC 2009)*], (2009).

[12] "APTI." http://diuf.unifr.ch/diva/APTI/competition.html. Accessed: 17/05/2012.

[13] "Tesseract." http://code.google.com/p/tesseract-ocr/. Accessed: 17/05/2012.