Typefaces and Ligatures in Printed Arabic Text: A Deep Learning-Based OCR Perspective

Omar Alhubaiti[1] and Irfan Ahmad[1,2](✉)

[1] King Fahd University of Petroleum and Minerals, Dhahran 31261, Saudi Arabia
{g201802660,irfan.ahmad}@kfupm.edu.sa
[2] SDAIA–KFUPM Joint Research Center for AI, Dhahran 31261, Saudi Arabia

Abstract. Arabic script is complex, with multiple shapes for the same characters in different positions. Another challenge of the script, in the context of recognition, is ligatures. A combination of a specific two or more character sequence takes a different shape than what those characters normally look like when they appear in a similar position. Deep learning-based systems are widely used for text recognition these days. In this work, we investigate the performance of deep learning systems for two alternative modeling choices: using *characters* as modeling units and using *character shapes* as modeling units. Moreover, we also investigate the impact on text recognition with mixed typefaces, where the training and test sets have samples from multiple typefaces, and discuss the effect of font families on recognition performance. We extend this by studying the effectiveness of the text recognition system in recognizing text from unseen typefaces, i.e., text in the test set is from a typeface not available in training. Finally, we present a methodology to automatically detect ligatures in printed Arabic text. We conducted experiments on the publicly available APTI dataset of printed Arabic text and report the findings and discuss the results.

Keywords: Printed Arabic Text OCR · Omnifont Text Recognition · Unseen Font Text Recognition · Automatic Ligatures Identification

1 Introduction

Arabic text has a complex cursive script that is written from right to left. Characters in Arabic take on different shapes depending on their position in a word (*beginning*, *middle*, *ending*, and *alone*) [7,9]. Not all characters take on all four shapes in different positions, and some of these shapes are dramatically different for some characters than for others, who would look almost identical. Linking between characters also varies between characters, and is a short (or varying length) horizontal dash, known as *Kashida*. Some characters can only be linked to those before or after them, or both. Ligatures are also present in Arabic script,

M. Coustaty and A. Fornés (Eds.): ICDAR 2023 Workshops, LNCS 14194, pp. 5–18, 2023.
https://doi.org/10.1007/978-3-031-41501-2_1

which are unique shapes of specific sequences of characters that would look different if those characters were not in that sequence. Normally, the characters forming a ligature are connected vertically instead of the normal horizontal connections. The only mandatory ligature is *Lam-Alif*, which is hard to read if it were not a ligature. Other ligatures are optional and vary between different typefaces. According to some studies, there are 478 possible ligatures in Unicode for printed Arabic script [1]. In addition to that, Arabic script employs diacritics to account for missing short vowels, since Arabic is an Abjad script [24]. There are four main diacritics, which can appear singly, doubly (*Tanween*/Nunation), or combined with a stress marker (*Shadda*). There are other diacritics like the tilde ˜_ and *Alif Khanjariyah* ('). It is customary not to write vowels if they are inferable from the context and to only write them when their absence might cause confusion to the reader, with the exception of religious texts and learning materials. All these factors contribute to the difficulty of the text recognition problem in Arabic script.

According to Tahir [23], there are eight main typefaces in the Arabic script: Nashk, Ruqaa, Diwani, Thuluth, Hamayoni, Farsi, Tawqi', and Ijaza. Other typefaces are derived from these and are similar in form to the originals. In modern computer systems, many typefaces can be classified as Naskh variety and come in two forms: Traditional, which is more or less faithful to the rules of writing present in calligraphy books such as [23], and simplified, in which some rules are discarded in favor of readability or simplicity. Other font files such as Diwani or Ruqaa vary very little between implementations and are compliant with calligraphy rules.

In this paper, we present an empirical investigation of the performance of a deep learning-based text recognition system in recognizing Arabic texts in different typefaces. Specifically, we examine the two common modeling choices: using Arabic characters as modeling units and using Arabic character shapes as modeling units to determine which modeling choice is more effective for deep learning-based Arabic OCR. Furthermore, we investigate the impact of mixed typefaces on text recognition, where the training and test sets include samples from multiple typefaces. We also study the effectiveness of the text recognition system in recognizing text from unseen typefaces, i.e., text in the test set is from a typeface not available in the training set, and analyze the effect of font families on recognition performance. Lastly, we introduce a methodology for automatic detection of ligatures in printed Arabic text.

The rest of the paper is organized as follows: In Sect. 2, we discuss the related works in this field. In Sect. 3, we explain the architecture of the deep learning system we employed for Arabic text OCR. In Sect. 4, we present the experiment details and the results. Finally, in Sect. 5 we present the conclusions from our work and some possible future works.

2 Related Work

In this field, recent research employs one of two major classifiers: Hidden Markov Models (HMMs) and deep learning systems based on neural networks. This section first presents the works that use HMMs, followed by those that use deep learning-based systems. Additionally, we describe some of the datasets used in the literature.

In 2008, a study by Prasad et al. [15] utilized HMMs for OCR by detecting the writing line and applying overlapping sliding windows over the line without further segmentation. Hand-crafted features were extracted from the windows. To expand the vocabulary, trimodel units were utilized as HMM models, resulting in a total of 162 HMMs for characters excluding the unigrams. The authors provide detailed information on the HMM construction method. This study was tested on the DAMP dataset and yielded a 15.9% WER for character mode. However, using n-grams improved the performance to 11.5% WER for 5-grams, as there were numerous out-of-vocabulary occurrences during testing.

In [3], Ahmad et al. try reducing HMM models using sub-characters in both printed and hand-written text. The number of HMMs was down to 108 from 178 coming at a slight error cost for handwritten text (e.g. WER of 3.62% up from 3.50%) and slightly better at printed (CER of 5.98% CER down from 6.51%). Authors note that this approach is beneficial when training datasets are imbalanced pertaining to character representation.

Ahmad et al. in [2] performed text recognition using HMMs with an open vocabulary approach. This approach handles multiple typefaces for text-line images. There are three main outputs of this study: a typeface recognizer, a text recognizer for a particular typeface, and unseen typeface recognizer that works by adapting known typefaces. In this work, features are handcrafted (and explained in detailed) and tuned for all the tasks and the recognizer obtains the features using segmented and overlapping sliding windows technique that exploits Arabic scripts' prominent line of writing and trains based on expectation maximization. Results show that splitting typeface and text recognition and adaptation for unseen typefaces provide superior performance. Authors test the framework on their own P-KHATT database which they describe, resulting in 2.89% Character Error Rate (CER) for mono-font, 95% success rate for mixed fonts, and 7.18% CER for unseen fonts. For APTI database, 2.07% CER is the result for mono font recognition. This approach may need testing for all fonts as unseen font in P-KHATT and APTI to further confirm research findings.

An example of work that uses HMMs for printed text recognition while also segmenting text into connected components is [20]. A writing line is established and certain manual features are calculated and HMMs were created without optimizing their number. No results were provided in this work. The dataset for this work is hand-crafted in MS-Word application and is very limited.

An instance of user-friendly systems for OCR is [22] where authors have developed a system for library use to perform OCR on old documents. The system performs image processing (straightening lines etc.), OCR, and translation. This work also describes the interface and API of the system. In terms of OCR, KALDI speech recognition framework is used, which relies on HMMs for language modeling with 165 HMMs used in total. Multiple datasets (ALTEC, IFN/ENIT, KHATT, and HADARA80P) were used for training with special care for ligatures (which are mapped to pronunciation variants in KALDI) and to open the vocabulary. Evaluation is done against three of the four systems mentioned in [6]. The proposed system performed worse than others and authors suggest it is due to difference between training datasets and actual library-setting test (84.6% WER vs FineReader's 100%). This study can benefit greatly from a new dataset that is closer to the target testing domain.

In [13], Jaiem et al. tackle OCR using texture analysis, noting that older works use approaches such as Wavelets and Gabor Filters. This work differentiates between apriori (global) and posterior (context-based) recognition and chooses the former to work on using steerable pyramid (SP) where the image is decomposed recursively to different resolutions by applying filters on it and features extracted at different levels and orientations. A back-propagation Neural Network is used for recognition which is evaluated on APTID/MF yielding results of 99.63% accuracy vs 97.04% using KNN at many typefaces.

When it comes to deep neural networks, many works approach text recognition using RNNs and sometimes CNNs for feature extraction. In [4], Ahmed et al. presents text recognition from scene images. Unconventionally, this work uses CNNs instead of RNNs for this task. For training, input is 50×50 gray-scale images augmented to 5 orientations. The architecture is a fully connected forward network using 3×3 and 5×5 filters, Relu for non-linearity, and max-pooling for features. Using EAST dataset which has 27 classes referring to different scene conditions with 2700 characters, authors tested this approach and optimized learning rate to obtain 14.57% error rate. Authors mentioned that low performance was attributed to small training dataset. The specific network architecture is not provided in this study.

Jain et al. in [14] use a CNN(features)-RNN(recognition) architecture for OCR in videos without extracting text from scenes first drawn by the success of such an approach in other similar scripts. The architecture consists of 7 convolutional layers with 4 max-pooling layers in between and 2 batch normalization (CNN), then (RNN) a couple of bidirectional long-short-term-memory layers (blstm) and a soft-max layer, before passing to the CTC layer for sequential recognition. This approach is trained on a hand-crafted dataset from wikipedia then tuned to ACTIV and ALIF datasets. Results vary from (98.17%, 79.67% Character Recognition Rate and Word Recognition Rate, respectively) on some tests to (75.05% and 39.43%), outperforming other systems such as Tesseract in all tests. Accuracy figures are far from the optimum 99% proposed in [5] to approach human-like performance, hence, more wor is needed in this field.

In [17], Rawls et al. were the first to combine deep learning approach for recognition, which uses CNN for feature extraction and LSTM and CTC for recognition with some language modelling in the form of WFST:weighted finite-state transducer using n-gram language models. The architecture composes of 2 convolutional (fully connected + Relu) units (CNN, features are extracted from a sliding window which is hand-optimized), followed by (RNN) (LSTM + fully connected), then a softmax layer, before passing to the CTC objective. This work was tested on the DAMP dataset. Authors specify hardware and parameters, used and report 2.7% and 8.4% CER and WER, respectively, surpassing the benchmark.

A study in 2017 by Alghamdi and Teahan [6] tests four Printed Arabic OCR systems (Automatic Reader, FineReader, Clever Page, and Tesseract) which are first introduced then metrics that measure such systems' performance are discussed, with the authors developing and defining their own suggested metrics (9 in total, including (digit, diacritic, and punctuation), and character (dot, loop, zigzag shaped, and overall) accuracies). Authors discuss several databases settling on KAFD for evaluation purposes. This work then compares the four systems in each of the 9 metrics while giving reasons for each metric's results, where each system excelled at different metrics from others. Authors note low overall accuracy of the systems, while the result being stronger if tested on more datasets. Areas of improvement in the whole field noted by the authors include diacritic and dot performance, and complex typefaces. Qaroush et al. in [16] presented a system for omnifont printed Arabic text recognition. A CNN-based system that relies on explicit character segmentation was developed. The authors report near-perfect segmentation results and high recognition results on APTI dataset of printed Arabic texts.

3 A System for Printed Arabic Text Recognition

The model we use here is adapted from [18]. Figure 1 shows an overview of the model's architecture. This model is a combination of CNN layers for feature detection, and RNN layers for the rest of the tasks which are Recognition and Classification. The CNN part of the model is made of five sets of convolution and max-pooling with the following dimensions: First, we start with 32×128 image with a feature depth of 1, next we have 64×16, 32×8, 32×4, 32×2, and 32×1 layers. The last of which has a feature depth of 256 and is the output of this part of the network.

Next, the RNN input has 32 time steps for a maximum of 32 characters per word and 256 features. Output of this part is a 2D map with 512 hidden cells and 32 time steps and the number of characters is variable (103 when using character shape as a model and 30 when using character as a model). This part is a stacked LSTM (forward and backwards). This is then fed to CTC (Connectionist-Temporal-Classification) to calculate loss for back-propagation. CTC is also used for decoding where we have the number of characters against probabilities for each. Various methods can be used after that like Beam Search or dictionaries but we opt for a simple best path decoding for this work. CTC has the advantages of not needing to segment the image to individual characters.

4 Experiments and Results

4.1 Dataset

We have used the APTI (Arabic Printed Text Image) dataset for conducting teh experiments [21]. APTI is a printed word dataset of more than 110,000 words in 10 different typefaces of which we used seven typefaces. This datasets has

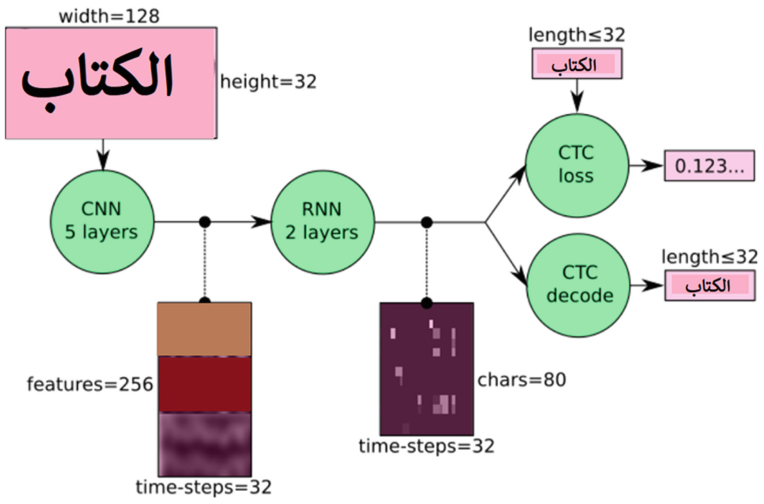


Fig. 1. Arabic text recognition system architecture. (figure adapted from: [19]).

10 sizes per typeface, of which we chose the largest (24 pts) and 4 styles per typeface (Normal, Bold, Italic, old-Italic) of which we selected the Normal. This is a purely computer-rendered dataset without camera or scanning artifacts. This dataset comes in 5 evenly divided sets that have comparable amount of characters in different shapes and positions which can be separated for use in training, validation, and testing. Ground truth for this dataset initially takes character shape into consideration. For use with character as a model approach, we modified that to discard the character shape information. The seven typefaces we used in our experiments from the APTI dataset are shown in Fig. 2.

4.2 Evaluation Measures

In this section, we present the measures we used to evaluate the text recognition results.

Character Error Rate (CER): CER is defined as follows:

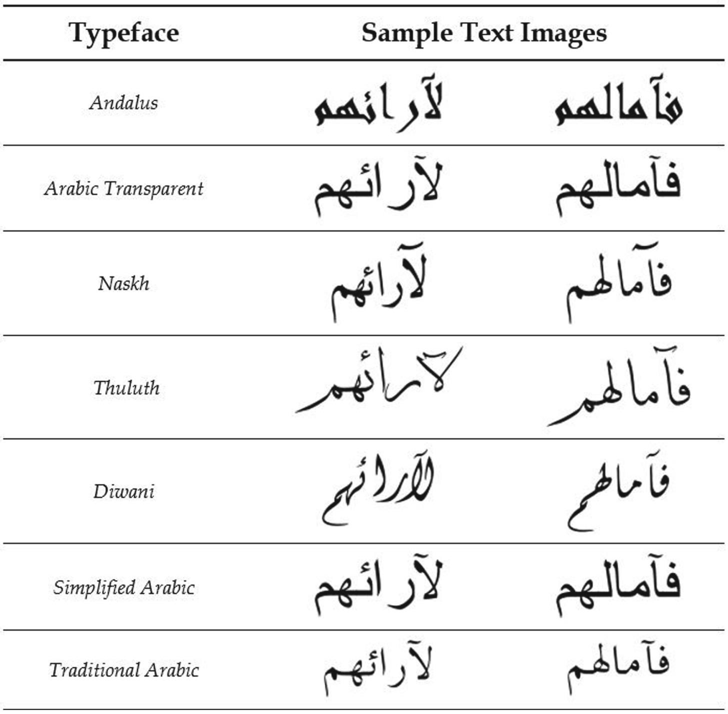
$$CER(\%) = \frac{S + I + D}{N} \times 100 \tag{1}$$

where; S, I, and D are substitution, insertion, and deletion errors respectively, and N is the total number of characters in the evaluation set (cf. [12, p. 419–421]).

Word Error Rate (WER): WER is defined as follows:

$$WER(\%) = \frac{S + I + D}{N} \times 100 \tag{2}$$

where; S, I, and D are substitution, insertion, and deletion errors respectively, and N is the total number of words in the evaluation set.



Typeface	Sample Text Images	
Andalus	لآرائهم	فآمالهم
Arabic Transparent	لآرائهم	فآمالهم
Naskh	لآرائهم	فآمالهم
Thuluth	لآرائهم	فآمالهم
Diwani	لآرائهم	فآمالهم
Simplified Arabic	لآرائهم	فآمالهم
Traditional Arabic	لآرائهم	فآمالهم

Fig. 2. Sample text images from the APTI dataset.

4.3 Experiment Setup

We used a machine with an Intel core-i7-7700HQ processor @2.80 GHz with 24 GB of RAM, and an Nvidia GTX 1060-6 GB graphics card over-clocked to 1911 MHz. The system was implemented using GPU accelerated tensor-flow in Python under Microsoft Windows 10 v1809. We used decaying learning rate starting at 0.01 for the first 10 batches, 0.001 for the next 10,000 batches, and 0.0001 for the rest. We keep training until error rate stops improving for 5 epochs. If accuracy did not improve or has declined, that model is discarded and the previous weights are reused for further training.

4.4 Character vs Character Shape as Modeling Unit for Printed Arabic OCR

First we trained the model on two of the seven typefaces, one simple (Arabic Transparent) and one complex (Diwani), to determine whether character as a model (Beginning, middle, and end shapes of a character are treated as one single model) was better or worse than character shape as a model (Beginning, middle, and end are treated as separate models). We did these experiments using 90% of the set-1 of the APTI dataset for training, and the remaining

10% for testing, using 25,000 random images for each epoch. The model only accepts single characters as symbols for shapes. For example, we can represent Alif-Beginning as ‘A’, Baa-Beginning as ‘B’, and so on. This poses a problem when having many character shapes as is the case when using character shape as a model. We got around this issue by using additional Unicode characters for extra shapes. We also use this when encoding ligatures as a single unit. When the model validates, it treats all shapes of the same character as one symbol, since when printing this output to a document, the font-system converts a character to its appropriate shape again and thus it makes no sense to penalize the model over this type of mistakes. The results of these experiments are summarized in Table 1. From the results, we can see that using character as a model performs better than using character shape as a model. This is in contrast to what was reported in [10] for handwritten Arabic text recognition using deep learning systems. A possible explanation for this phenomenon is that the character and their different position dependent shapes are quite regular in printed Arabic texts but highly variable in handwritten texts. This, in addition to the fact that we can have more samples per class if we used character as a model than otherwise, and the network is able to select the best features to generalize and use as discriminants for that character.

4.5 Single Typeface, Mixed Typeface, and Unseen Typeface Text Recognition

Next, we ran a set of experiments without involving ligatures at all. These consist of 8 total training: one for each of the seven typefaces and the eighth experiment for all the seven typefaces combined. We trained the model on sets 1–3 of each typeface of size 24-pts with normal style and validated using the set–4. For each epoch, the system selects 30,000 random images from the training set split into 600 batches) except for the last experiment involving all typefaces which has 150,000 random training images per epoch using a 75%–25% training/validation-testing split. Results are present in Table 2. We can see that the model performs quite well even for complex typefaces with lots of overlap. The best results are reported for Simplified Arabic typeface and the worst results are for Naskh typeface. It is also interesting to note that mixed typeface recognition results are as good as single typeface recognition, which are in contrast to what is reported for some HMM-based systems (e.g., [2]). This could possibly be attributed to the high capacity of deep learning models having high number of trainable parameters.

Table 1. Summary of the results for different modeling units on text from two typefaces from the APTI dataset.

Modeling Unit	Typeface	CER	WER
Character	Arabic Transparent	0.82	4.33
	Diwani	0.27	1.56
Character Shape	Arabic Transparent	1.01	5.00
	Diwani	1.54	8.11

Table 2. Summary of the OCR results for different typefaces from the APTI dataset.

Typeface	CER	WER
Andalus	0.26	1.33
Arabic Transparent	0.15	0.73
Naskh	0.28	1.34
Thuluth	0.26	1.28
Diwani	0.24	1.26
Simplified Arabic	0.14	0.67
Traditional Arabic	0.15	0.76
Mixed (All combined)	0.20	1.03

Next, to investigate the performance of the system on unseen typeface and to test whether typeface similarity has an effect on model accuracy, we designed a set of leave-one-out experiments. We train the system on six typefaces and test on the remaining typeface, alternating. Results are summarized in Table 3.

From these results, we can clearly see that for fonts that visually differ from each other, the model was unable to generalize and achieved poor results. For Simplified Arabic and Arabic Transparent typefaces, however, and to a lesser extend, Traditional Arabic, the model was able to generalize very well. The explanation for this is that all these typefaces belong to the same family (Naskh Family). Decotype Naskh was visually more archaic so the model was not able to generalize sufficiently in this case. So we can have the intuition that we can use one typeface from the same family (out of several different Naskh implementations, or several different Andalus implementations, for instance) to have a more general model for that family.

Table 3. Summary of the OCR results for unseen typefaces from the APTI dataset.

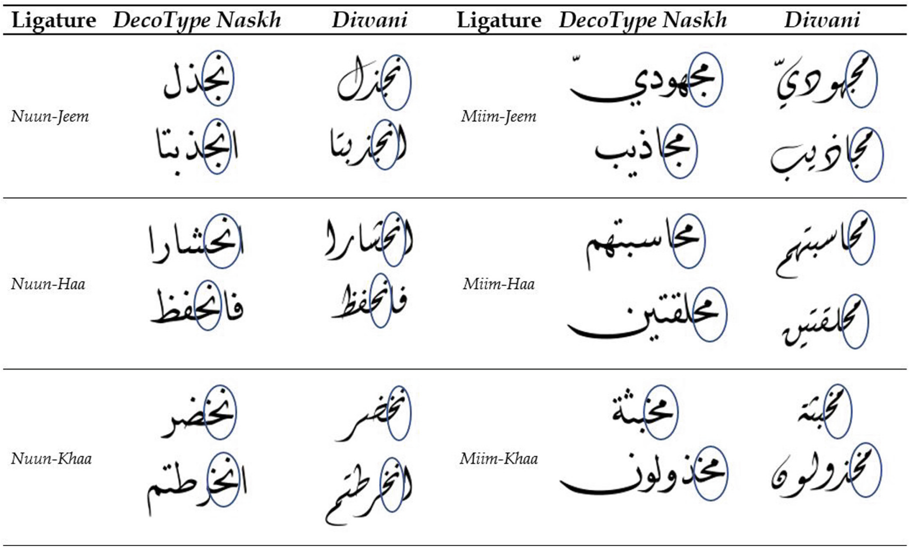
Unseen Typeface	CER	WER
Andalus	56.83	98.09
Arabic Transparent	0.18	0.89
Naskh	10.94	39.00
Thuluth	20.00	66.03
Diwani	56.21	98.56
Simplified Arabic	0.29	1.42
Traditional Arabic	5.30	25.73

4.6 An Approach to Automatically Identify Ligatures in Printed Arabic Texts

We performed a pilot study to determine whether our approach will be able to distinguish between a ligature and a non-ligature in a typeface where we do not fully know the ligature (either we do not have the font file or the font file does not contain ligature information, like .ttf fonts). We chose Naskh typeface as a relatively simpler one and Diwani as the complex one. We could not choose a simpler typeface such as Simplified Arabic or Arabic Transparent because these typefaces do not have much ligatures, if at all.

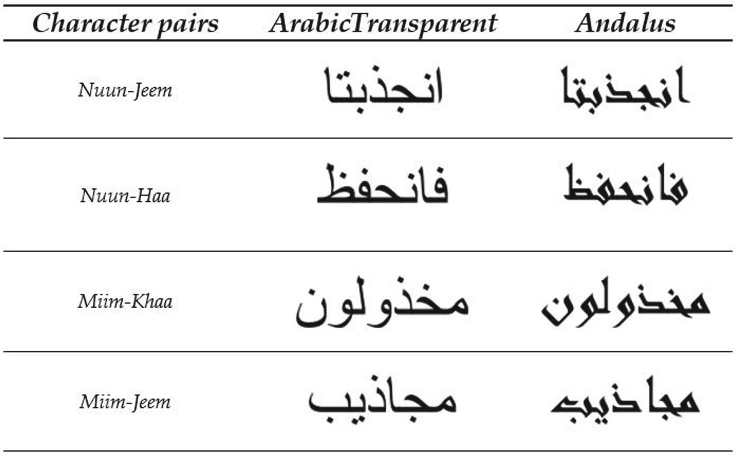
We selected six bigram combinations that are present as ligatures in both typefaces: (*Nuun* or *Miim* in the beginning position with either *Jeem*, *Haa*, or *Khaa* in the middle position). They are the most common ligatures in Arabic (cf. [8,9]). Additionally, We have selected six similar bigram combinations that are non-ligatures (*Saad* or *Daad* in the beginning position either *Jeem*, *Haa*, or *Khaa* in the middle position). Sample ligatures from the APTI dataset is shown in Fig. 3. It should be noted that same character pairs may not form ligatures in other typefaces as illustrated in Fig. 4.

To develop a system to automatically identify ligatures, we train our text recognition system twice. Once treating these 12 character bigrams as ligatures thereby modeling each of them as a single ligature model. This is done by changing the training set annotations. For the second round of training, we treat the 12 character bigrams as individual characters and not as ligatures. Our final ligature identification system then decodes the unseen test sets with the possibility of treating the 12 character bigrams pairs as either ligatures or non-ligatures depending on which has the higher probability.



Ligature	DecoType Naskh	Diwani	Ligature	DecoType Naskh	Diwani
Nuun-Jeem	نجذل انجذبتا	نجذل انجذبتا	Miim-Jeem	مجهوديّ مجاذيب	مجهوديّ مجاذيب
Nuun-Haa	انحشارا فانحفظ	انحشارا فانحفظ	Miim-Haa	محاسبتهم محلقتين	محاسبتهم محلقتين
Nuun-Khaa	نخضر انخرطتم	نخضر انخرطتم	Miim-Khaa	مخبثة مخذولون	مخبثة مخذولون

Fig. 3. Sample ligatures (circumscribed within a blue oval) from the APTI dataset. (Color figure online)

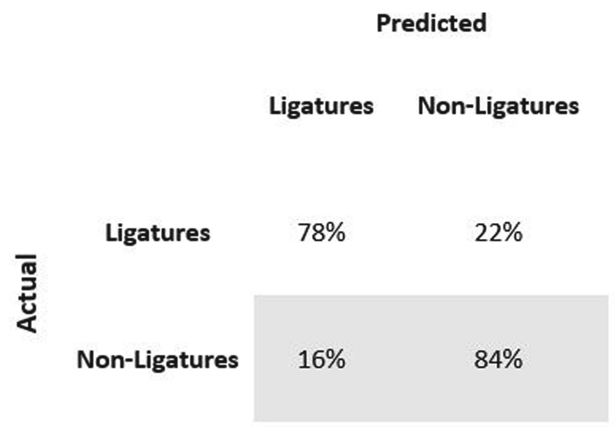


Character pairs	ArabicTransparent	Andalus
Nuun-Jeem	انجذبتا	انجذبتا
Nuun-Haa	فانحفظ	فانحفظ
Miim-Khaa	مخذولون	مخذولون
Miim-Jeem	مجاذيب	مجاذيب

Fig. 4. Sample text images from the APTI dataset where character combinations do not form ligatures.

If the model detects a ligature correctly, we consider this as a true positive. Additionally, if the model classified a non-ligature pair correctly by outputting individual characters instead of ligatures, this is regarded as a true negative. On the other hand, if a ligature was classified as a non-ligature by outputting individual characters instead of ligatures, this is treated as a false negative. Similarly, if a non-ligature was classified as a ligature by outputting individual characters instead of ligatures by outputting ligatures instead of individual characters, we consider this as a false positive. Based on this setup, we ran our experiments on the unseen test sets of the two selected typefaces. The resulting confusion matrix is presented in Fig. 5.

We can see that the system was able to differentiate between ligatures and non-ligatures effectively. The possible explanation is that a bigram ligature looks visually different from how the original characters would look like in other contexts. Thus the system learns the individual characters from other contexts and ligatures from explicit training on these bigram pairs. On the other hand, non-ligature bigrams would look the same as they look in other contexts. Moreover, we have many more samples in the dataset where the two constituent characters of this non-ligature bigram appear as separate characters. So, the system will lean more towards recognizing the non-ligature bigrams as individual characters instead of as ligatures.



		Predicted	
		Ligatures	Non-Ligatures
Actual	Ligatures	78%	22%
	Non-Ligatures	16%	84%

Fig. 5. Confusion matrix for Ligature Detection.

5 Conclusions and Future Work

In this paper, we present our work related to printed Arabic text recognition using deep-learning. We investigated two alternative modeling choices and conclude that character as a model perform better than character shape as a model. Further we conducted experiments to investigate the system's performance for single typeface text recognition to observe if different typefaces lead to different recognition performance due to their visual complexity. We observed that the system was able to effectively recognize text in all the seven typefaces which ranges from visual simple to visually complex typefaces. In fact, the system's performance was at times better for visually complex typefaces as compared to some of the simpler ones. Moreover, the system's performance of mixed typefaces text recognition was as good as single typeface text recognition. We also saw that models trained for different typefaces in the same family can generalize to other unseen typefaces from the same family but not to typefaces from different families. Last but not the least, we presented an approach to develop a system to automatically identify ligatures in unseen text. The system, although not perfect, performed reasonably well.

The experiments were conducted on only one dataset and for only seven different typefaces. This clearly is a limitation of the work. Further experiments need to be carried out to confirm the generalizability of the presented conclusions. Furthermore, only six bigram ligatures were experimented. There are many more ligatures, so further work is needed to confirm the effectiveness of the presented approach for automatic identification of ligatures. Accordingly, possible follow-ups for this work include expanding the case study for ligature detection into more ligatures and typefaces, and trying the scheme for additional datasets. Investigating other deep learning models, such as transformer models can also be an interesting future work. This work might also be applicable to handwritten text although the factor that each writer might create their own ligatures can limit the usefulness of this which needs to be investigated. Moreover, deeper investigation into the font-family concept is warranted to determine whether models can generalize within the classical categorization of typefaces as seen in calligraphy books. Last but not the least, generating rare occurring ligatures using systems such as GANs (e.g., [11]) in order to robustly train a text recognition system can be an interesting future work.

Acknowledgment. The authors would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for supporting this work. Irfan Ahmad would like to additionally thank Saudi Data and AI Authority (SDAIA) and KFUPM for supporting him through SDAIA-KFUPM Joint Research Center for Artificial Intelligence grant number JRC–AI–RFP–06.

References

1. Arabic ligature: Graphemica search. https://graphemica.com/search?q=arabic+ligature. Accessed 20 Apr 2023
2. Ahmad, I., Mahmoud, S.A., Fink, G.A.: Open-vocabulary recognition of machine-printed Arabic text using hidden Markov models. Pattern Recogn. **51**, 97–111 (2016)
3. Ahmad, I., Rothacker, L., Fink, G.A., Mahmoud, S.A.: Novel sub-character hmm models for Arabic text recognition. In: 2013 12th International Conference on Document Analysis and Recognition, pp. 658–662. IEEE (2013)
4. Ahmed, S.B., Naz, S., Razzak, M.I., Yousaf, R.: Deep learning based isolated Arabic scene character recognition. In: 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR), pp. 46–51. IEEE (2017)
5. Al-Badr, B., Mahmoud, S.A.: Survey and bibliography of Arabic optical text recognition. Signal Process. **41**(1), 49–77 (1995)
6. Alghamdi, M., Teahan, W.: Experimental evaluation of Arabic OCR systems. PSU Res. Rev. **1**(3), 229–241 (2017)

7. Amara, N.: On the problematic and orientations in recognition of the Arabic writing. In: CIFED 2002, pp. 1–10 (2002)
8. Elarian, Y., Ahmad, I., Awaida, S., Al-Khatib, W., Zidouri, A.: Arabic ligatures: analysis and application in text recognition. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 896–900. IEEE (2015)
9. Elarian, Y., Ahmad, I., Zidouri, A., Al-Khatib, W.G.: Lucidah ligative and unligative characters in a dataset for Arabic handwriting. Int. J. Adv. Comput. Sci. Appl. **10**(8) (2019)
10. Eltay, M., Zidouri, A., Ahmad, I.: Exploring deep learning approaches to recognize handwritten Arabic texts. IEEE Access **8**, 89882–89898 (2020)
11. Eltay, M., Zidouri, A., Ahmad, I., Elarian, Y.: Generative adversarial network based adaptive data augmentation for handwritten Arabic text recognition. PeerJ Comput. Sci. **8**, e861 (2022)
12. Huang, X., Acero, A., Hon, H.W., Reddy, R.: Spoken Language Processing: A Guide to Theory, Algorithm, and System Development, vol. 1. Prentice Hall PTR, Upper Saddle River (2001)
13. Jaiem, F.K., Kanoun, S., Eglin, V.: Arabic font recognition based on a texture analysis. In: 2014 14th International Conference on Frontiers in Handwriting Recognition, pp. 673–677. IEEE (2014)
14. Jain, M., Mathew, M., Jawahar, C.: Unconstrained scene text and video text recognition for Arabic script. In: 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR), pp. 26–30. IEEE (2017)
15. Prasad, R., Saleem, S., Kamali, M., Meermeier, R., Natarajan, P.: Improvements in hidden Markov model based Arabic OCR. In: 2008 19th International Conference on Pattern Recognition, pp. 1–4. IEEE (2008)
16. Qaroush, A., Awad, A., Modallal, M., Ziq, M.: Segmentation-based, omnifont printed Arabic character recognition without font identification. J. King Saud Univ.-Comput. Inf. Sci. **34**(6), 3025–3039 (2022)
17. Rawls, S., Cao, H., Sabir, E., Natarajan, P.: Combining deep learning and language modeling for segmentation-free OCR from raw pixels. In: 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR), pp. 119–123. IEEE (2017)
18. Scheidl, H.: Handwritten text recognition in historical documents. In: Tuwien Faculty of Informatics, Technische Universität Wien, Diplomarbeit. Tuwien (2018)
19. Scheidl, H.: Build a handwritten text recognition system using tensorflow. Medium, 09-Aug-2020 (2019)
20. Shaker, A.S.: Recognition of off-line printed Arabic text using hidden Markov models. Ibn AL- Haitham J. Pure Appl. Sci. **31**(2), 230–238 (2018). https://doi.org/10.30526/31.2.1952. http://jih.uobaghdad.edu.iq/index.php/j/article/view/1952
21. Slimane, F., Ingold, R., Kanoun, S., Alimi, A.M., Hennebert, J.: A new arabic printed text image database and evaluation protocols. In: 2009 10th International Conference on Document Analysis and Recognition, pp. 946–950. IEEE (2009)
22. Stahlberg, F., Vogel, S.: QATIP-an optical character recognition system for Arabic heritage collections in libraries. In: 2016 12th IAPR Workshop on Document Analysis Systems (DAS), pp. 168–173. IEEE (2016)
23. Tahir, K.M.: Tarikh al-khatt al-Arabi wa-adabuh: huwa kitāb tārīkhī ijtimāī adabī muzayyan bi-al-uwar al-khaīyah wa-al-rusāum al-fūtughrāfīyah. al-Matbaah al-Tijariyah al-Hadīthah (1982)
24. Zitouni, I.: Natural Language Processing of Semitic Languages. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-45358-8