

Classifying Arabic handwriting using convolutional neural network

Anas Quteishat^{1,2}, Ashraf Al Sharah², Ahmed Qtaishat³, Tareq Ahmad Alawneh²

¹Department of Communication and Computer Engineering, Faculty of Engineering, Al-Ahliyya Amman University, Amman, Jordan

²Department of Electrical Engineering, Faculty of Engineering Technology, Al-Balqa Applied University, Al-Salt, Jordan

³Foundation Program, Sohar University, Sohar, Oman

Article Info

Article history:

Received Sep 22, 2025

Revised Jul 26, 2026

Accepted Aug 12, 2026

Keywords:

Arabic handwriting

Artificial neural network

Convolutional neural networks

Deep learning

Optical character recognition

ABSTRACT

This paper presents a classification of Arabic handwritten letters using convolutional neural networks (CNNs). The paper aims to address the unique challenges of recognizing Arabic letters due to their special characteristics such as dots and similarities. Leveraging deep learning, particularly CNNs, the project employs an architecture specifically designed for image classification tasks, which is well-suited for optical character recognition (OCR) of Arabic handwritten letters. The dataset used consists of images collected from the "Arabic handwritten alphabets, words, and paragraphs" (AHAWP) and "Arabic handwritten character database" (AHCDB), which have been carefully processed to ensure uniformity and diversity. The model was built using deep network designer, a tool in MATLAB, and trained over three epochs to prevent overfitting, achieving an accuracy of 89.2%.

This is an open access article under the [CC BY-SA](https://creativecommons.org/licenses/by-sa/4.0/) license.



Corresponding Author:

Anas Quteishat

Department of Communication and Computer Engineering, Faculty of Engineering

Al-Ahliyya Amman University

Amman, Jordan

Email: a.quteishat@ammanu.edu.jo

1. INTRODUCTION

Throughout the last 100 years, people have strived to instill reading abilities into machines through various methods which produced different levels of success. Machine recognition of handwriting remains an elusive task due to its demanding algorithms which makes it one of the most advanced image recognition problems. The primary goal of this process involves machine reading ability, but the initial letter classification systems served as the foundation to develop advanced reading capabilities. The process known as optical character recognition (OCR) enables the automatic conversion of printed or handwritten text found in images to digital forms that permit users to search, modify, store compactly, present online, and execute machine-based processing through cognitive computing systems. OCR transforms text data management since it allows quick and precise conversion of various document types into digital formats through which users can achieve applications like document word search, automated postal mail sorting, and document editing from printed form. Modern industry applications alongside everyday systems heavily rely on OCR technology because artificial intelligence and machine learning continue to develop their capabilities [1]. OCR enables us to translate text from one language to another using translation tools, enables traffic systems to recognize and record vehicle license plates, and enables employees to save time and effort wasted on data entry processes [2]–[5]. While OCR has many applications and benefits, it also faces several challenges and problems. Text in documents with complex layouts, fonts, or poor-quality images is difficult to be

recognized. Handling multiple languages, various font styles and handwritten text can be challenging. Also, OCR can be affected by background noise or clutter, which can interfere with the text recognition process. Moreover, implementing OCR can be expensive and complex, requiring specialized hardware, software, and expertise.

This paper focuses on building a system to recognize handwritten basic Arabic letters, which could be very useful to develop an Arabic writing tool as an alternative to the keyboard, as some people still prefer to write by hand. Handwritten Arabic text has unique technical challenges, which have been more recently addressed. Numerous image types have been the subject of different proposed and applied methods. There are 28 letters in the Arabic alphabet, and 15 of them have varied shapes. The shape of some characters is similar, but the difference arises with the position and number of dots, whether above or below the characters. Additionally, a group of two points can be written as one or two connected components and a group of three points can be written as one, two, or three connected components, depending on the style of writing. It can be considered that recognition process includes three general stages. Firstly, pre-recognition, which involves image processing to enhance the quality of the image to make it suitable for OCR. It includes noise reduction, binarization, skew correction, and normalization. Secondly, recognition itself, the core of OCR where the actual recognition of characters takes place. This can be done using one of two primary methods: pattern recognition, where the system compares the text in the image with a database of known patterns, and feature extraction, where the system decomposes the characters into features (like lines, loops, and intersections) and matches them to known character features. In the final stage, the errors are corrected, this step involves understanding the context and text is formatted into readable text [6]. Faizullah *et al.* [4] presented the first omni font OCR; this system could recognize standard fonts. When dealing with Arabic OCR, the system needs to recognize cursive scripts, which is a more complex problem, where dealing with letter shaper is more advanced. Kasem *et al.* [7] was applied the OCR to Arabic newspapers headlines; this is done by analyzing sub-words. Al-Yousefi and Udpa [8] used Bayesian classification to isolate characters through moments projection.

With the advances in deep learning Arabic OCR was applied using these methods, hence the handling of contextual complexity is managed. Li *et al.* [9] proposed an OCR built on end to end framework. This method used vision transformers for handwriting recognition, this shows the benefits of sequencing without segmenting the words. Mortadi *et al.* [10] introduced ALNASIKH, this method relayed on a transformer-based Arabic OCR system the system uses the transformer-based optical character recognition (TrOCR) architecture for Arabic text. Recent methods used Arabic-nougat extended vision TrOCR to markdown extraction, showing the importance of document-level OCR systems for Arabic text understanding. Recently, end-to-end architectures combining optical and contextual recognition significantly improved the performance of OCR in Arabic handwriting [11]. The integration of large language models (LLMs) improved contextual understanding in OCR [12].

The main contribution of this paper is to build a basic Arabic letters dataset and prepare it for training a convolutional neural network (CNN) [13]. The presented work will create an Arabic letter dataset containing 14,000 Arabic letters obtained from two different datasets to achieve high diversity. This process requires additional effort in the process of matching the different types of data from different sources. MATLAB R2021a will be used to perform most of the image processing unless other software is required. Section 4 of this paper will present a detailed description of this process. Deep network designer, a tool from MATLAB, is used to build a CNN from scratch, configure its layers, adjust the training options, and then train it on the dataset.

The remainder of the paper is organized as follows. Section 2 presents the methodology used in the paper. Section 3 presents the collection and preparation of the dataset used. The experiments and results are given section 4. The conclusion of the work is presented in section 5.

2. METHOD

The proposed OCR system is implemented using deep learning because of its widely spread applications as mentioned above. The main advantage of deep learning is that it automatically extracts the features without any human supervision. More specifically, the CNNs were used, which are a relatively new method [14], [15].

CNNs do not require human intervention to extract features from the data. Instead, features are learned directly from the data during the training process, reducing the need for manual preprocessing. In fact, this feature is for deep learning techniques in general. CNNs learn hierarchical representations of data, starting from simple features (like dots and lines) in the initial layers to more complex features (like shapes and objects) in deeper layers. This allows CNNs to effectively capture complex patterns in data like images and videos. Figure 1 presents the different layers involved in building a CNN.

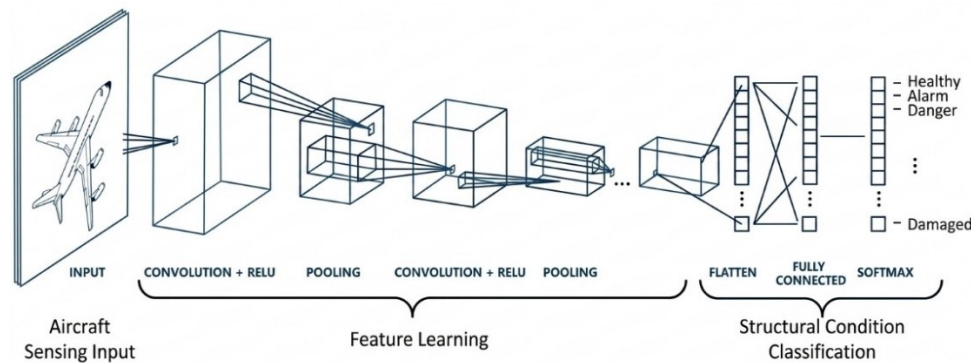


Figure 1. Hierarchical structure of CNNs

CNNs also reuse the same filters across the entire image, which cuts down significantly on the number of parameters that need to be learned compared to traditional neural networks. Thanks to their pooling layers, they can recognize patterns no matter where they appear in the image a feature that's especially useful for tasks like object detection and image classification. Pooling also shrinks the size of the data representations, which lowers computational complexity and boosts efficiency. Another key advantage is parallelization: because of how CNNs are structured, they can process multiple pixels at once, leading to faster training compared to methods that handle data sequentially. This becomes a major advantage when working with large datasets [16].

Despite their many features, CNNs also face some challenges, such as the need for large datasets. Achieving high performance in CNNs requires massive datasets, but collecting and labelling these datasets is often costly and time-consuming, especially in scenarios where the data is scarce or expensive to obtain. A method to overcome this challenge is data augmentation, which is the process of artificially increasing the size and diversity of the training dataset by subjecting it to some transformations such as rotating images by small angles, randomly cropping portions of images, flipping images along the horizontal axis, and changing image brightness and contrast. This reduces the risk of overfitting by preventing the model from memorizing specific data patterns. The quality of the data is also important, not just the quantity, as CNNs are sensitive to small variations in input data, which can lead to significant changes in output and affect the accuracy if the images are blurry or noisy. This also leads to overfitting, where the model learns specific details or noise in the training data, resulting in poor performance on unseen or test data, especially when trained on small or narrow datasets. Dropout is a regularization technique that reduces overfitting by "dropping" (disabling) a random subset of neurons with a certain probability during each training iteration. This forces the network to learn more robust and generalized features rather than relying on specific neurons [17]. After the data collection phase, another challenge arises, which is the high computational cost. Training CNN models requires significant computational power due to the large number of operations involved, particularly when dealing with large datasets and deep architectures. This can lead to long training times and the need for specialized hardware, such as GPUs. The advent of advanced computing technologies such as cloud computing and parallel computing provides effective ways to overcome this challenge. Another solution is to use transfer learning techniques to improve performance using pre-trained models [18]. There are also some less significant challenges such as that CNNs are often considered "black boxes", making it difficult to interpret how they arrive at specific decisions. This lack of transparency can be a significant drawback in applications where understanding the model's reasoning is crucial [19]. It is also worth noting that it is sometimes a good idea to integrate CNNs with recurrent neural networks (RNNs) to improve handling of sequential data such as connected text [20].

The architecture of CNNs consists of a sequence of layers that work together to process inputs and produce outputs. The input layer first receives the raw image (the main type of input data) provided to the model. This layer simply passes the image to the next layer without any computations, but it must be adjusted to accommodate the image properties (such as height, width, and number of channels) as these properties must be uniform. Then, the image is fed into the convolution layer, which performs a convolution operation between the image and one or more filters (also called kernels) to result in a feature map containing the automatically extracted features. The input images can have one or more channels, and the filters have the same number of channels (each filter has a channel for each channel in the input image). The image and the filter, meaning each channel in the image with its corresponding channel in the filter, are considered. Both the image and the filter are treated as two-dimensional matrices, where the image matrix represents pixel values, and the filter matrix contains weight values. The filter matrix is typically much smaller in height and

width. In the convolution operation, the filter slides over the image one stride at a time from the beginning of the image (the top-left corner) to the end of the row, then starts again from the beginning of the row one stride below the previous position, and so on until it has gone through the entire image as shown in Figure 2.

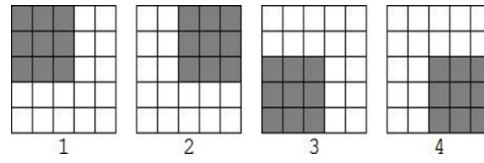


Figure 2. The positions of a 3×3 filter on a 5×5 image with a 2×2 stride

This requires precision in calculating the dimensions of both the filter and the stride so that they fit the input image dimensions. Assuming that the dimensions of the input image are $I_h \times I_w$, the dimensions of the filter are $F_h \times F_w$, and the dimensions of the stride are $S_h \times S_w$, the feature map will be a $M_h \times M_w$ matrix as shown in (1) and (2).

$$M_h = \frac{I_h - F_h}{S_h} + 1 \quad (1)$$

$$M_w = \frac{I_w - F_w}{S_w} + 1 \quad (2)$$

These two amounts must be natural numbers. Otherwise, padding the input image may be needed (change its dimensions by adding zeros around it). Additionally, each stride dimension must not be greater than its corresponding filter dimension to ensure that all parts of the input image are passed through. Each entry in the mentioned matrix corresponds (in order) to one of the positions through which the filter passes. At each position the filter passes through during its striding, an entry wise product is performed between the filter and the corresponding patch of the image. The entry wise product operation between two matrices of identical dimensions is performed as (3).

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{23}b_{23} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix} \quad (3)$$

Then, all the values resulting from the entry wise product operations are summed with a different value for each filter, called the bias, to produce the corresponding entry of the feature map. So, if X is the input image, W is the filter, and b is the bias, then every entry of the feature map Y is computed as shown in (4).

$$Y_{mn} = \sum_{i,j,k} \left((W_{i,j,k})(x_{m+i,n+j,k}) \right) + b \quad (4)$$

The output of the convolution layer is a set of feature maps, one for each filter. These feature maps highlight the different learned features from the input image. On the other hand, the filters are learnable parameters (like weights in artificial neural networks or ANNs). The multiple filters learn different patterns (detect different features). The convolution layer is followed by an activation function, which is a mathematical function applied on the output of the convolution layer to introduce non-linearity into the model, enabling it to learn more complex patterns. One common activation function used in CNNs is the rectified linear unit (ReLU), which reduces the computational load by acting as shown in (5).

$$ReLU(x) = \max(0, x) \quad (5)$$

Another important one is the sigmoid (σ), which is used in the context of binary classification and probabilistic interpretations tasks, where it transforms its input into an output in range of 0 to 1, based on (6).

$$\sigma(x) = \frac{1}{1+e^{-x}} \quad (6)$$

Of course, the inputs to these functions are the feature maps from the previous convolution layers, which are matrices. So, these functions are applied to each entry of the input matrix. The output is then passed to the pooling layer, which reduces the size of the data to decrease the complexity of the model and increase its efficiency and resistance to small variations in the input images. Max pooling, which selects the maximum

value from a specific region (called a window), is commonly used. The concept of stride is also present here with the only difference being that the window replaces the filter as shown in Figure 3.

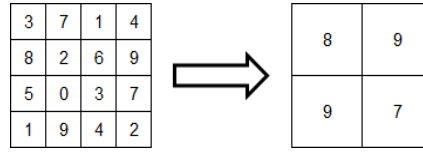


Figure 3. Max pooling with a 2×2 window and a 2×2 stride

A sequence of convolution layer, activation function, and pooling layer is usually repeated more than once in CNNs, immediately after the input layer. But instead of the raw images, the input of each subsequent convolution layer is the output of its previous layer, which will be one or more feature maps. Additionally, multiple feature maps are treated as channels of a single input image when fed into a convolution layer. After them comes the role of the flattening layer to convert the multi-dimensional input matrices into vectors. This layer may be combined with the next layer, which is the fully connected layer. The fully connected layer processes a flattened vector for classification or decision-making based on the extracted features. So, it is utilized as a classifier in classification models. Inside this layer, every neuron is connected to every neuron in the previous layer as shown in Figure 4.

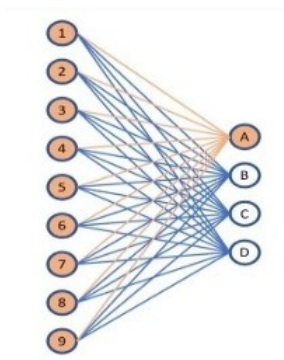


Figure 4. The fully connected layer

If m is the number of neurons of its previous layer, n is the number of its neurons, $X1 \times m$ is its input matrix, $Wm \times n$ is its weight matrix (where each row contains the weights that connect one neuron from the previous layer to all neurons of the current layer in order), $B1 \times n$ is its bias matrix, and $Y1 \times n$ is its output matrix as shown in (7).

$$Y = XW + B \quad (7)$$

The number of neurons in a fully connected layer depends on the task (classification or regression). In classification models, this number must be equal to the number of classes in the model, so that its output is a set of values, one for each class. The softmax function then converts these values into probabilities (values between 0 and 1, such that their sum equals 1) representing the likelihood of each class given the input. If x_n is the n th original value (of the fully connected layer) as shown in (8).

$$\text{Softmax}(x_n) = \frac{e^{x_n}}{\sum_i e^{x_i}} \quad (8)$$

The only task left for the last layer, the output layer, is to determine the most likely class according to the probabilities computed by the previous function (the prediction for the given input image). Everything mentioned so far is about what is called forward propagation, where data propagates from the input layer to the output layer for prediction. But for training purposes, another type of data propagation called backpropagation is needed. Backpropagation is used to update the weights and biases based on the difference between predicted values and actual values. During training, the input image is first passed through forward propagation, which outputs a predicted value for the input. Then the loss is calculated using a loss function to

measure the difference between the predicted value and the actual value. The loss function depends on the task, but a common version for classification tasks is the cross-entropy loss. If p is the predicted value (probability) of the true class of an input image, then the cross-entropy loss L is calculated as shown in (9).

$$L = -\sum_{i=1}^C y_i \log(p_i) \quad (9)$$

where C is the number of classes (28 classes), y_i is 1 for correct classes and 0 otherwise, and p_i is the model's predicted probability for class i . After the loss is calculated, the gradients are calculated in each layer using the differentiation to measure the impact of changing the weights on the loss, and then distribute these changes to the previous layers, which is called backpropagation. Finally, an optimization algorithm like stochastic gradient descent (SGD) or Adam is used to update the weights based on the gradients calculated during backpropagation. If ω_0 is the old weight, ω_n is the new weight, α is a factor called the learning rate, and L is the loss, then the update formula using SGD is as shown in (10).

$$\omega_n = \omega_0 - \alpha \left(\frac{\partial L}{\partial \omega_0} \right) \quad (10)$$

This process is repeated with each batch (set of samples) of the training data, gradually improving the performance of the model through multiple iterations by minimizing the loss. Instead of passing the entire training dataset through the network at once (which is called an epoch), the data is divided into smaller groups (batches). The weights are updated after processing each batch based on the average loss of the samples in that batch. This approach speeds up training and reduces memory usage. Additionally, performance is evaluated on a dataset (called a validation dataset) separate from the training data. This can be done several times during or after training and helps monitor the generalization ability and prevent overfitting.

3. DATASET DESCRIPTION

The data used is carefully collected to achieve randomness and balance from two datasets, "Arabic handwritten alphabets, words, and paragraphs" (AHAWP) [21] and "Arabic handwritten character database" (AHCDB) [22]. In the first dataset, the samples are divided by the users (writers), but it does not cover the 28 basic Arabic letters (which the model aims to recognize). It covers only 18 of them with about 820 images from 82 users, and a total of 14,740 images. On the other hand, the second dataset covers all 28 letters with 750 images for each letter, for a total of 21,000 images, but it is less diverse as it seems to be made by fewer users. So, to add some diversity to our collection, and to make it include all 28 letters, for each letter, 250 images were selected belonging to 50 users from AHAWP and another 250 from AHCDB, making a collection of 500 images per letter for the first 18 letters and 500 images from the AHCDB for the other 10 letters. The total number of images was 14,000 images.

Since the images came from different sources image preprocessing stage was required. First the images were separated into folders, each named after the Arabic letter that the folder contains its images. The images differ in format between the two datasets, being PNG formatted in AHAWP, and Bitmap formatted in HACDB. So, the second step after collecting the images was to unify their format, a program called IrfanView was used to convert all the images in the collection to PNG format numbered from 000 to 499 for each letter. In addition, the images of AHAWP are grayscale and are changed into binary images using MATLAB. Next, the empty white spaces around the images are cropped to make them smaller in size. Figure 5 shows the steps involved in building the dataset.

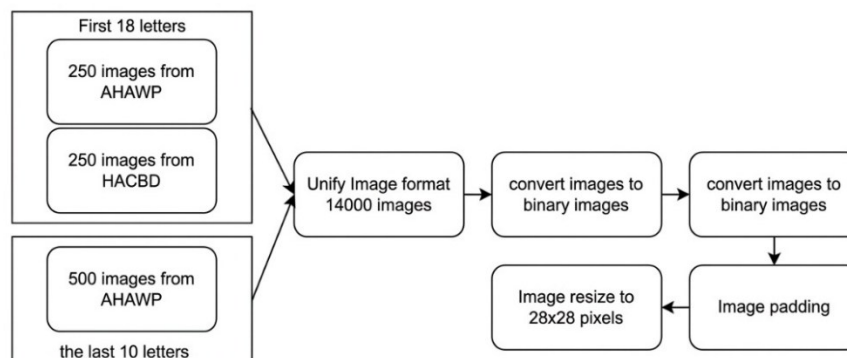


Figure 5. Steps involved in building the dataset

It is preferable (but not required) that the input images are square. Therefore, padding each image with empty and balanced white spaces either from the right and left (if the height is greater than the width) or from the top and bottom (if the width is greater than the height) while leaving the square images as they are without change. Finally, the images were downsized to just 28×28 pixels to reduce the computational power needed to train the model in a reasonable time. Additionally, this type of image does not require high resolution, as the largest image size before downsizing was 348×348 (an exaggerated size for just letters).

4. IMPLEMENTATION AND RESULTS

The CNN was built and trained from scratch, for this purpose, the deep network designer was used, an application in MATLAB R2021a. This tool offers an interactive environment for designing, analyzing, and training deep learning models on an easy-to-use graphical interface. The dataset was divided into two sets, 80% of the data was used for training and 20% was used for testing. The division of the data was done equally among all classes.

The network built consists of an input layer that receives single-channel images of 28×28 size (according to the collection images), followed by a convolution layer with two 5×5 filters and a 2×2 stride, followed by a ReLU activation function, and then a max pooling layer with a 2×2 window. Then again, a convolution layer but with four 5×5 filters and the same 2×2 stride, followed by a ReLU activation function and then a max pooling layer with the same 2×2 window. Then comes the fully connected layer with 28 neurons (Arabic letters), followed by a softmax layer and finally the output layer. Figure 6 shows the CNN network used to classify the dataset and Table 1 shows the details of each layer used.

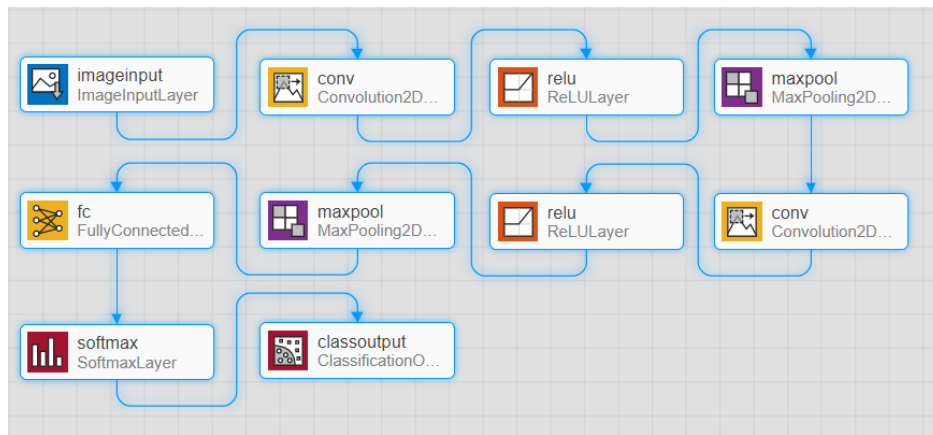


Figure 6. The CNN used to classify the dataset

Table 1. Details the layer coefficients

Layer type	Output size	Parameters	Activation
Input	$28 \times 28 \times 1$	-	-
Conv2D-1	$26 \times 26 \times 32$	5×5 kernel, 32 filters	ReLU
MaxPooling-1	$13 \times 13 \times 32$	2×2 pool size	-
Conv2D-2	$11 \times 11 \times 64$	5×5 kernel, 64 filters	ReLU
MaxPooling-2	$5 \times 5 \times 64$	2×2 pool size	-
Flatten	1,600	-	-
Fully connected	128	128 units	ReLU
Output	28	28 units	Softmax

As previously mentioned, the dataset was divided into two subsets, with 20% (2,800 images) allocated for testing and the remaining 11,200 images used for training. The learning rate was set to 0.01, and the batch size was defined as 14, meaning the training process consisted of 800 iterations per epoch. During experimentation, it was observed that the CNN's performance did not improve beyond the third epoch; in fact, training the network for additional epochs led to overfitting. Specifically, when extended to a fourth epoch, the testing accuracy began to decline, indicating the onset of overfitting. Figure 6 illustrates the system's accuracy and loss curves. It can be observed that accuracy increased sharply after the first epoch (around iteration 800), continued to improve gradually during the second epoch, and showed minimal enhancement in the third epoch, confirming that the model had effectively converged. In the lower section of Figure 6, the loss function is presented. The loss function indicates how far off the model's predictions are

from the true labels. The loss function is shown in (8), In the first epoch, the models predicts all classes nearly equally, so $p_i = \frac{1}{28} \approx 0.036$ and when this value is substituted in equation 8 $L = -\log\left(\frac{1}{28}\right) \approx 3.33$. This is shown in Figure 7. As the training proceeds the loss stabilizes around 0.2-0.3.

The confusion matrix for the results is shown in Figure 8. In the confusion matrix it is visible that letters with dot changes such as ط-ظ-ع-غ, د-ذ-ر-ز, ب-ت-ث-ي, and ج-ح-خ have higher confusion since they are very similar. To further analyze the performance of the CNN, the recall, precision, and F1-score are calculated for each character. The obtained results are shown in Table 2. From Table 2, one can see that the average F1-score is 90.9% which is higher than the overall accuracy of 89.2%. This is due to the fact that there are fundamental differences between the two quantities. The F1-score being slightly higher than the accuracy suggests that the model has learned characteristics of most Arabic letters reasonably well, with only few challenging cases such as ي and ج-ح-خ group.

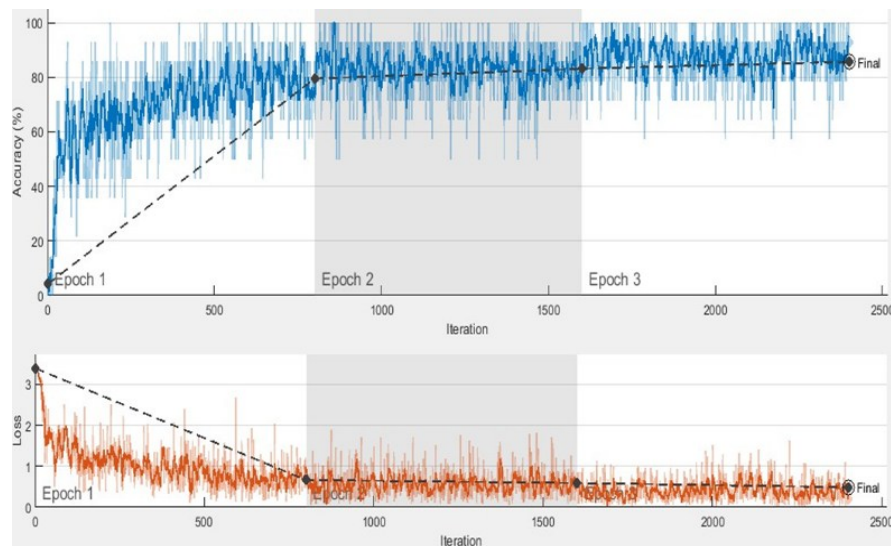


Figure 7. The graphs of both accuracy and loss

	ا	ب	ت	ث	ج	ح	خ	د	ذ	ر	ز	س	ش	ص	ض	ط	ظ	ع	غ	ف	ق	ك	ل	م	ن	هـ	و	ي
ا	95	1	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0
ب	2	90	4	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
ت	1	3	91	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
ث	0	2	4	89		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5
ج	0	0	0	0	88	5	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3
ح	0	0	0	0	6	87	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2
خ	0	0	0	0	4	6	86	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4
د	1	0	0	0	0	0	0	93	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
ذ	0	0	0	0	0	0	0	3	90	1	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
ر	1	0	0	0	0	0	0	2	1	92	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0
ز	0	0	0	0	0	0	0	1	2	2	92	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0
س	0	0	0	0	0	0	0	0	0	0	0	91	5	1	0	0	0	0	0	0	0	0	0	0	0	0	0	3
ش	0	0	0	0	0	0	0	0	0	0	0	6	90	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2
ص	0	0	0	0	0	0	0	0	0	0	0	2	3	90	2	0	0	0	0	0	0	0	0	0	0	0	0	3
ض	0	0	0	0	0	0	0	0	0	0	0	1	2	3	91	0	0	0	0	0	0	0	0	0	0	0	0	3
ط	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	93	3	1	0	0	0	0	0	0	0	0	0	3
ظ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	92	2	0	0	0	0	0	0	0	0	0	4
ع	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	91	3	0	0	0	0	0	0	0	0	3
غ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	4	92	0	0	0	0	0	0	0	0	3
ف	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	91	4	2	0	0	0	0	0	3
ق	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	92	2	0	0	0	0	0	3
ك	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	3	91	1	0	0	0	0	3
ل	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	92	3	1	0	2	
م	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	92	3	1	0	2	
ن	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	91	3	0	3
هـ	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2	2	92	2	2	
و	1	0	0	0	0	0	0	1	2	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	91	1
ي	2	3	2	4	2	0	0	0	0	0	0	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	84
Total Pred	102	99	101	98	100	98	95	99	95	98	98	102	101	96	93	96	98	99	95	96	99	97	97	99	97	96	97	100

Figure 8. The confusion matrix for the Arabic letters

Table 2. Classification performance for each Arabic letter

Class (Letter)	Precision (%)	Recall (%)	F1-score (%)	Class (Letter)	Precision (%)	Recall (%)	F1-score (%)
ا (Alif)	93.1	95	94	ض (Dad)	91	91	91
ب (Ba)	90.9	90	90.5	ط (Tah)	93	93	93
ت (Ta)	90.1	91	90.5	ظ (Zah)	93.9	92	92.9
ث (Tha)	90.8	89	89.9	ع (Ain)	91.9	91	91.5
ج (Jeem)	88	88	88	غ (Gain)	92	92	92
ح (Ha)	88.8	87	87.9	ف (Fa)	91	91	91
خ (Kha)	90.5	86	88.2	ق (Qaf)	91.1	92	91.5
د (Dal)	93.9	93	93.5	ك (Kaf)	92.9	91	91.9
ذ (Thal)	94.7	90	92.3	ل (Lam)	90.2	92	91.1
ر (Ra)	93.9	92	92.9	م (Meem)	92	92	92
ز (Zay)	93.9	92	92.9	ن (Noon)	92.9	91	91.9
س (Seen)	91.9	91	91.5	ه (Ha)	91.1	92	91.5
ش (Sheen)	91.8	90	90.9	و (Waw)	92.9	91	91.9
ص (Sad)	92.8	90	91.4	ي (Yaa)	84	84	84

Table 3 shows a performance comparison of different learning methods compared with the CNN. The tables show that the performance of the CNN network gave better results in comparison with other systems. The proposed model achieved an accuracy of 89.2%, which proves that this type of network (CNN networks) is more suitable for the purposes of recognizing all types of images, including images of letters and characters as shown in Figure 6.

Table 3. Performance comparison of different classifiers

Dataset	Model / approach	Reported accuracy (%)
HACDB	Support vector machine [23]	86.1
HACDB	K-nearest neighbors [24]	85.47
HAAWP	Support vector machine with handcrafted features [25]	87.2
HACDB and AHAWP	Histogram of oriented gradients (HOG) + multilayer perceptron (MLP) [26]	88.2
HACDB and AHAWP	Support vector machine [25]	87.5
HACDB and AHAWP	Proposed CNN [25]	89.2

5. CONCLUSION

This paper developed and tested a CNN for handwritten Arabic letter recognition, achieving an accuracy of 89.2% on a carefully prepared dataset. The dataset used in this paper is composed of selected images from the HACDB and AHAWP datasets. A total of 14,000 images were collected and were used for testing and training CNN. The results confirm the suitability of CNNs for this task, while also highlighting challenges such as overfitting and limited dataset diversity. Future work could build on this foundation by applying deeper architecture, data augmentation, and transfer learning to improve accuracy, as well as integrating linguistic models for handling connected script and ambiguous characters.

FUNDING INFORMATION

Authors state no funding involved.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Anas Quteishat	✓	✓	✓	✓	✓	✓		✓	✓	✓		✓	✓	
Ashraf Al Sharah		✓				✓			✓	✓	✓			
Ahmed Qtaishat	✓		✓	✓		✓	✓		✓		✓			
Tareq Ahmad Alawneh	✓		✓	✓		✓	✓		✓		✓			

C : Conceptualization

M : Methodology

So : Software

Va : Validation

Fo : Formal analysis

I : Investigation

R : Resources

D : Data Curation

O : Writing - Original Draft

E : Writing - Review & Editing

Vi : Visualization

Su : Supervision

P : Project administration

Fu : Funding acquisition

CONFLICT OF INTEREST STATEMENT

Authors declare that there is no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are openly available in Kaggle and Mendeley Data at <https://www.kaggle.com/datasets/mloey1/ahcd1> and <https://data.mendeley.com/datasets/2h76672znt/2>, respectively.

REFERENCES

- [1] A. A. I. Alasadi, M. S. A. Smerat, K. Govindu, G. Sivakumar, and V. Sahasranamam, "Mathematically modified deep learning model assisted handwritten digit recognition for intelligent document processing systems," *Journal of Machine and Computing*, vol. 5, no. 4, pp. 2661–2671, Oct. 2025, doi: 10.53759/7669/jmc202505204.
- [2] R. Najam and S. Faizullah, "Analysis of recent deep learning techniques for Arabic handwritten-text OCR and post-OCR correction," *Applied Sciences*, vol. 13, no. 13, Jun. 2023, doi: 10.3390/app13137568.
- [3] T. T. H. Nguyen, A. Jatowt, M. Coustaty, and A. Doucet, "Survey of post-OCR processing approaches," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–37, Jul. 2022, doi: 10.1145/3453476.
- [4] S. Faizullah, M. S. Ayub, S. Hussain, and M. A. Khan, "A survey of OCR in Arabic language: applications, techniques, and challenges," *Applied Sciences*, vol. 13, no. 7, Apr. 2023, doi: 10.3390/app13074584.
- [5] C. A. Schmidt, K. H. F. -Wellman, and P. D. Neuffer, "From OCR and ECAR to energy: perspectives on the design and interpretation of bioenergetics studies," *Journal of Biological Chemistry*, vol. 297, no. 4, Oct. 2021, doi: 10.1016/j.jbc.2021.101140.
- [6] A. A. -Khadrah *et al.*, "Enhancing intrusion detection systems using machine learning and advanced feature selection methods," *Electronics*, vol. 15, no. 9, Apr. 2026, doi: 10.3390/electronics15091860.
- [7] M. S. Kasem, M. Mahmoud, and H.-S. Kang, "Advancements and challenges in Arabic optical character recognition: a comprehensive survey," *ACM Computing Surveys*, vol. 58, no. 4, pp. 1–37, Mar. 2026, doi: 10.1145/3768150.
- [8] H. Al-Yousefi and S. S. Udpal, "Recognition of Arabic characters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 8, pp. 853–857, 1992, doi: 10.1109/34.149585.
- [9] M. Li *et al.*, "TrOCR: transformer-based optical character recognition with pre-trained models," *The Thirty-Seventh AAAI Conference on Artificial Intelligence (AAAI-23)*, vol. 37, no. 11, pp. 13094–13102, Jun. 2023, doi: 10.1609/aaai.v37i11.26538.
- [10] A. Mortadi *et al.*, "ALNASIKH: an Arabic OCR system based on transformers," in *2023 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, Sep. 2023, pp. 74–81, doi: 10.1109/MIUCC58832.2023.10278370.
- [11] A. Mostafa *et al.*, "OCFormer: a transformer-based model for Arabic handwritten text recognition," in *2021 International Mobile, Intelligent, and Ubiquitous Computing Conference (MIUCC)*, May 2021, pp. 182–186, doi: 10.1109/MIUCC52538.2021.9447608.
- [12] F. Valizadeh, F. Ghasemian, and E. Shabaninia, "Comparative analysis of large language models for OCR post-processing in Persian: from ParsBERT to GPT," in *2025 29th International Computer Conference, Computer Society of Iran (CSICC)*, Feb. 2025, pp. 1–6, doi: 10.1109/CSICC65765.2025.10967472.
- [13] Y. Bengio, Y. LeCun, C. Nohl, and C. Burges, "LeRec: a NN/HMM hybrid for on-line handwriting recognition," *Neural Computation*, vol. 7, no. 6, pp. 1289–1303, Nov. 1995, doi: 10.1162/neco.1995.7.6.1289.
- [14] M. A. -Jaber, N. Makhoul, M. A. -Jaber, and R. Beale, "Cost prediction for product development using hybrid deep learning model: a meta-heuristic model," *Multimedia Tools and Applications*, vol. 84, no. 27, pp. 33307–33330, Dec. 2024, doi: 10.1007/s11042-024-20437-y.
- [15] L. Alzubaidi *et al.*, "Review of deep learning: concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, no. 1, Mar. 2021, doi: 10.1186/s40537-021-00444-8.
- [16] T. Kattenborn, J. Leitloff, F. Schiefer, and S. Hinz, "Review on convolutional neural networks (CNN) in vegetation remote sensing," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 173, pp. 24–49, Mar. 2021, doi: 10.1016/j.isprsjprs.2020.12.010.
- [17] J. Gupta, S. Pathak, and G. Kumar, "Deep learning (CNN) and transfer learning: a review," *Journal of Physics: Conference Series*, vol. 2273, no. 1, May 2022, doi: 10.1088/1742-6596/2273/1/012029.
- [18] K. Shaheed *et al.*, "DS-CNN: a pre-trained Xception model based on depth-wise separable convolutional neural network for finger vein recognition," *Expert Systems with Applications*, vol. 191, Apr. 2022, doi: 10.1016/j.eswa.2021.116288.
- [19] Y. Zhang *et al.*, "Carrying out CNN channel pruning in a white box," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 10, pp. 7946–7955, Oct. 2023, doi: 10.1109/TNNLS.2022.3147269.
- [20] H. S. Gill and B. S. Khehra, "An integrated approach using CNN-RNN-LSTM for classification of fruit images," *Materials Today: Proceedings*, vol. 51, pp. 591–595, 2022, doi: 10.1016/j.matpr.2021.06.016.
- [21] M. Khan, "Arabic handwritten alphabets, words and paragraphs per user (AHAWP) dataset," *Mendeley Data*, V1, 2021, doi: 10.17632/2h76672znt.1.
- [22] A. Lawgali, M. Angelova, and A. Bouridane, "HACDB: handwritten Arabic characters database for automatic character recognition," in *2013 4th European Workshop on Visual Information Processing, EUVIP 2013*, 2013, pp. 255–259.
- [23] M. N. AlJarrah, M. M. Zyout, and R. Duwairi, "Arabic handwritten characters recognition using convolutional neural network," in *2021 12th International Conference on Information and Communication Systems (ICICS)*, May 2021, pp. 182–188, doi: 10.1109/ICICS52457.2021.9464596.
- [24] M. Loey, A. El-Sawy, and H. EL-Bakry, "Deep learning autoencoder approach for handwritten arabic digits recognition," 2017, *arXiv:1706.06720*.
- [25] A. Lawgali, "A survey on Arabic character recognition," *International Journal of Signal Processing, Image Processing and Pattern Recognition*, vol. 8, no. 2, pp. 401–426, Feb. 2015, doi: 10.14257/ijsp.2015.8.2.37.
- [26] A. Mezghani, R. Maalej, M. Elleuch, and M. Kherallah, "Recent advances of ML and DL approaches for Arabic handwriting recognition: a review," *International Journal of Hybrid Intelligent Systems*, vol. 19, no. 1-2, pp. 61–78, Jul. 2023, doi: 10.3233/HIS-230005.

BIOGRAPHIES OF AUTHORS

Dr. Anas Quteishat    received his B.Sc. in Electronic Engineering from Princess Sumaya University for Technology, Jordan in 2003. In 2005 he finished his M.Sc. in Electronic System Design from the University of Science Malaysia. He got his Ph.D. in 2009 in Computational Intelligence from the University of Science Malaysia. His research interests include artificial neural networks, pattern recognition, intelligent systems, and embedded systems. He has published many research papers in international journals. Currently, he is an associate professor at Al Balqaa Applied University. He can be contacted at email: anas.quteishat@bau.edu.jo.



Dr. Ashraf Al Sharah    has completed his Ph.D. from Tennessee State University, United States. He was a research associate at cyber vis research lab. He served as an assistant professor in the Department of Computer Engineering at Al-Ahliyya Amman University. He is serving now as an assistant professor in Department of Electrical Engineering in Al-Balqa Applied University. His research interest include wireless security, IoT, smart attack, AI, and game theory. He can be contacted at email: aalsharah@bau.edu.jo.



Ahmed Qtaishat    finished his master's degree from University Utara Malaysia, in intelligent systems in 2007. He joined Sohar University in 2011. From 2012 until 2017, he worked as a coordinator in the general foundation program. His research area focuses on artificial intelligence, machine learning, and genetic algorithms. He has published research articles in different national and international journals and conferences. He can be contacted at email: aqtaishat@su.edu.om.



Dr. Tareq Ahmad Alawneh    has received his B.S. and M.S. degrees in Computer Engineering from Jordan University of Science and Technology and his Ph.D. degree in Computer Engineering from the University of Hertfordshire, United Kingdom, in 2021. He is currently an assistant professor in the Department of Electrical Engineering at Al-Balqa Applied University, Jordan. His research interests include cache partitioning algorithms, low-power and high-performance architectures, DRAM systems, multi-core processors, and embedded systems. He can be contacted at email: tareq.alawneh@bau.edu.jo.