

Arabic Handwritten Recognitions

Project Documentations



Prepare By:
Ayat AL Manaseer

Supervisor:
Eng. Ruba Alkhshieny

Table of content

Index of figures	3
Abstract.....	4
1. Introduction	4
1.1 problem statements.....	5
1.2 objective	5
1.3 contribution.....	5
2. Convolutional Neural Network CNN.....	5
2.1 convolution layer	7
2.2 pooling layer.....	7
2.3 fully connected layer.....	7
3. Background.....	8
4. The model	10
4.1 Arabic handwritten digit recognition.....	10
4.2 Arabic handwritten character recognition.....	10
4.3 Methods Of Arabic Handwritten Recognition.....	11
4.4 Arabic handwritten recognition advantages.....	12
4.5 The challenge of Arabic handwritten recognition	13
4.6 The use of Arabic handwritten recognition	15
5. Analysis of the model	16
5.1 the input and dataset.....	16
5.2 implementation	18
6. Result.....	27
7. Conclusion and future work.....	28
8. Acknowledge.....	29
9. References	30

Index of figures

Figure 1 Illustration of the convolution process on the input image of size $M \times M$ with a kernel of size $N \times N$ [3]. [4]	6
Figure 2 Arabic handwritten character	11
Figure 3 Online vs Offline Handwriting recognition system	12
Figure 4 Arabic letters in a different position [11].	14
Figure 5. Handwritten words with secondary markings circled: hamza, shadda, and madda [12].	14
Figure 6 the shape of the same words [12].	14
Figure 7 sample of the dataset	16
Figure 8 the methodology of Arabic handwritten recognitions	17
Figure 9 import the libraries	18
Figure 10 unzip of the dataset	19
Figure 11 loading the Arabic digits dataset	19
Figure 12 loading the Arabic characters dataset	20
Figure 13 sample of our dataset	20
Figure 14 the dataset after normalization	21
Figure 15 encoding outputs	21
Figure 16 merging the Arabic letters and numbers dataset	22
Figure 17 details about the results of some parameters	24
Figure 18 the model after 10 epochs	25
Figure 19 the check of overfitting at epochs 10	26
Figure 20 the check of overfitting as epochs 20	26
Figure 21 the final result of test model	27
Figure 22 the prediction of the new images	28

Abstract

As technology has advanced significantly, it has become necessary to recognize handwritten letters, words, and even phrases. particularly for commercial and educational organizations. Human errors that can happen during data entry are eliminated using optical character recognition (OCR) software. Due to the largely linear form of handwritten Arabic, particularly with its points, Arabic has gotten less attention in this field than in other languages.

Automated recognition of handwritten is one of the most interesting applications of machine learning. This project poses handwritten Arabic recognition for digits, and characters as a learning problem and provides an overview of the ML techniques that have been used to address this challenging task.

In this project, we have successfully built a Python deep-learning project on a handwritten recognition with two models one for digits, and characters. we have built and trained the Convolutional neural network which is very effective for image classification purposes. show the accuracy which presents 98.06%.

1. Introductions

Automated handwriting recognition has achieved significant real-world success in targeted applications such as address recognition on mail pieces for sorting automation and reading of courtesy and legal amounts on bank checks, using domain-dependent constraints to make the problem tractable. However, unconstrained handwritten text recognition is still an open problem that is attracting renewed interest as an active area of research due to the proliferation of smartphones and tablet devices where handwriting with a finger or stylus is likely to be a potentially convenient mode of input for these handheld devices [1].

Handwritten digit recognition is the ability of computers to recognize human handwritten digits. It is a hard task for the machine because handwritten digits are not perfect and can be made with many different flavors. Handwritten digit recognition is the solution to this problem which uses the image of a digit and recognizes the digit present in the image.

In this project, we will use the public datasets Arabic digits and Arabic letters for Arabic letters and Arabic digits respectively which are available at Kaggle kernel. Also, there are many related papers that introduced some solutions for that problem.

1.1 Problem statement

Rely on more papers we read about handwritten recognition just for digits or letters, so we will combine the dataset which contains 38 classes combining letters and digits together.

In this project, we want to build a model which can classify a new image into an Arabic letter or digits. We can use machine learning classification or deep learning algorithms like CNN which can successfully boil down images into a high representation of a highly accurate result.

1.2 Objective

The main goal of this project is to create and implement an Arabic handwriting system utilizing CNN that can be used for digits, and letters.

1.3 Contribution of our study

Building two Arabic handwritten recognition for digits, and characters using CNN.

2. Convolutional Neural Network CNN

A distinct strategy is used by the Convolutional Neural Network, which imitates how our eyes help us perceive our surroundings. When we perceive a picture, we instantly break it down into numerous smaller sub-images and examine each one separately.

The so-called convolution layer is where the job is done. To accomplish this, we create a filter that establishes the size of the partial images we will be examining and a step length that

establishes the number of pixels we proceed after each computation, or, more specifically, how near the partial images should be to one another. This action has significantly decreased the image's dimensionality.

The pooling layer comes next. From a purely computational standpoint, the same thing occurs here as in the convolution layer, with the exception that, depending on the application, we only take the average or maximum value from the result. This saves minute details in a few pixels that are essential to completing the task [2].

The fully-connected layer, which is familiar to us from conventional neural networks, comes last. We may use the densely meshed layers now that we have significantly shrunk the image's dimensions. In this instance, the many sub-images are linked once more in order to identify the links and complete the categorization [2].

In a convolutional neural network data and functions have additional structure. The input data $x_1, x_2, \dots, \text{and } x_n$ are images or sounds. Formally, the input to a convolutional layer is $M \times M \times C$ image where M is the height and width of the image, $M \times M$ is number of pixels in the image and C is number of channels per pixel. For gray scale image have one channel $C = 1$ but the RGB image have three channels $C = 3$ [3].

A CNN consists of a number of layers (convolutional layers, pooling layers, fully connected layers)

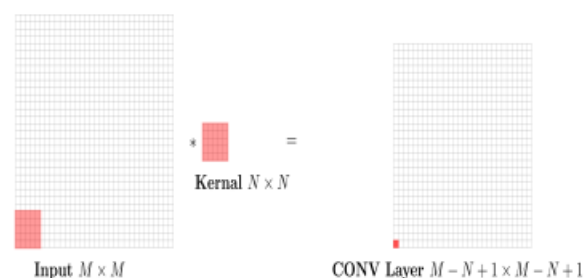


Figure 1 Illustration of the convolution process on the input image of size $M \times M$ with a kernel of size $N \times N$ [3]. [4]

For example, suppose we have the input is 32×32 and the kernel is 5×5 then the output of a convolution will be 28×28 .

Now let's take the convolution layers in more details:

2.1 Convolution Layer

The 4x4x3 image's dimensions should be decreased in the first stage. We define a filter with a dimension of 2x2 for each color to achieve this. Additionally, we desire a step length of 1, meaning that the filter should advance exactly one pixel after each computation step. The image's details will be kept while the dimension is not drastically reduced. A 3x3 matrix will be produced by our convolutional layer if we replace a 4x4 matrix with a 2x2 matrix and move one column or one row in each step. The figure illustrates how the scalar product of the two 2x2 matrices is used to calculate the matrix's component values.

2.2 Pooling layer

The (Max) Pooling Layer takes the 3x3 matrix of the convolution layer as input and tries to reduce the dimensionality further and additionally take the important features in the image. We want to generate a 2x2 matrix as the output of this layer, so we divide the input into all possible 2x2 partial matrices and search for the highest value in these fields. This will be the value in the field of the output matrix. If we were to use the average pooling layer instead of a max-pooling layer, we would calculate the average of the four fields instead [4].

The pooling layer also filters out noise from the image, i.e. elements of the image that do not contribute to the classification. For example, whether the dog is standing in front of a house or in front of a forest is not important at first.

2.3 Fully connected layer

The fully-connected layer now does exactly what we intended to do with the whole image at the beginning. We create a neuron for each entry in the smaller 2x2 matrix and connect them to all neurons in the next layer. This gives us significantly fewer dimensions and requires fewer resources in training [4].

This layer then finally learns which parts of the image are needed to make the classification dog or non-dog. If we have images that are much larger than our 5x5x3 example, it is of course also possible to set the convolution layer and pooling layer several times in a row before going into the

fully-connected layer. This way you can reduce the dimensionality far enough to reduce the training effort [2].

3. Background

The issue of character recognition has received a lot of research in numerous tongues numerous strategies have been suggested for resolving this issue. Chinese, English, and other examples are given. as classifiers for deep learning Researchers have had a lot of success in recent years [5].

In many languages, CNN is commonly utilized in this industry. Arabic Words can be divided into segments to identify characters as stages before therapy. Nevertheless, some datasets have been produced. by using distinct characters. full understanding of the word Because Arabic characters are linked together, any categorization method has an additional hurdle. There is an absence of The researchers were required to use Arabic word datasets.

Das et al. [6]developed a set of 88 features representing samples of handwritten Arabic numerals. A multi-layer perceptron (MLP) classifier was developed with an input layer, a single hidden layer, and an output layer. The MLP was trained with the back propagation (BP) algorithm and used for classification of Arabic numerals using the CMATERDB 3.3.1 Arabic handwritten digit dataset [20]. Experimental results on the database of 3000 samples showed that the model achieved an average accuracy of 94.93%.

[7] presents the development of an Arabic online handwriting recognition system. To develop our system, we have chosen the neural network approach. It offers solutions for most of the difficulties linked to Arabic script recognition. they test the approach with our collected databases. This system shows a good result and it has a high accuracy(98.50% for characters, 96.90% for words).

El-Sawy et al. [3] A CNN was trained and tested on the AHCD database, which comprises 16,800 handwritten Arabic characters. They use dropout and normalization, and their model contains two convolutional layers. On the test data, the proposed CNN had a 94.9% accuracy rate.

A CNN was employed by Alaasam et al. [8] to identify ancient Arabic handwriting. Two convolutional layers, two fully connected layers, and a final fully connected layer made up the CNN. Three datasets—one with historical handwritten text images, one with synthetic text images, and one with printed text images—were used for the experiments. The combination of the handwritten text image dataset and the synthesized text image dataset yielded the best accuracy (85%).

Younis [9] created a CNN for Arabic character recognition in handwritten documents. Three convolutional layers and a fully linked layer made up the suggested CNN. Using the AHCD and AIA9K datasets, respectively, experimental results showed that the CNN was able to reach 94.7% and 94.8% accuracy.

The literature review on automatic handwriting recognition is summarized in Table 1. According to the prior literature, number recognition has received more attention than alphabetic character recognition. Additionally, only adult handwriting has been used to train the models for Arabic handwriting recognition. As far as we are aware, there hasn't been any research from above that combining the three on one module and the result will show in GUI. the module was just for digits or characters or just for text.

Table 1 Summary of handwriting recognition models using deep learning.

References	Year	Type	The dataset	Result
[6]	2010	Digits	CMATERDB	94.93%
[7]	2016	Character	Mars DB	98.5%
[3]	2017	Character	AHCD	94.9%
[8]	2017	Words	HAHPT	85%
[9]	2017	Character	AHCD	94.7%
		Character	AIA9K	94.8%

4. The model

4.1 Arabic handwritten digit recognition

In recent years, handwritten digit recognition has been an important field due to its applications in many fields. This work focuses on the recognition portion of handwritten Arabic numeral recognition which faces many challenges, including unlimited variation in human handwriting and large public databases. The paper presented a deep learning technique that can be effectively applied to recognize Arabic handwritten digits. LeNet-5, a convolutional neural network (CNN) that has been trained and tested (Arabic handwritten number images) contains 60,000 training and 10,000 test images. A comparison of the results is made, and it is finally shown that using CNN significantly improved across different machine learning classification algorithms. The convolutional neural network was trained and tested against (Images of Handwritten Numbers in Arabic) containing 60,000 training and 10,000 test images [10].

4.2 Arabic Handwritten Character Recognition

Large public datasets and the infinite variety in human handwriting provide difficulties for handwritten Arabic character recognition systems. In this study, we develop a deep learning architecture that is useful for identifying Arabic letters written by hand. A specific type of multi-layered feed-forward layer taught in supervised mode is a convolutional neural network (CNN). Our database of 16,800 handwritten Arabic characters was trained and evaluated by CNN. The performance of the CNN is improved in this study by the use of optimization techniques. Feature extractors and trainable classifiers are frequently used in tandem in common machine-learning techniques. Comparing CNN to other machine-learning classification techniques yields notable gains [3].

Arabic is a type of Semitic language used in countries in the Middle East as a mother tongue for millions of people [3]. In general, the Arabic alphabet consists of twenty-eight letters of the alphabet as illustrated in the figure below which consists of a sample of Arabic handwritten characters.

أ	ب	ت	ث	ج	ح	خ
alef	beh	teh	theh	jeem	hah	khah
د	ذ	ر	ز	س	ش	ص
dal	thal	reh	zain	seen	sheen	sad
ض	ط	ظ	ع	غ	ف	ق
dad	tah	zah	ain	ghain	feh	qaf
ك	ل	م	ن	ه	و	ي
kaf	lam	meem	noon	heh	waw	yeh

Figure 2 Arabic handwritten character

4.3 Methods Of Arabic Handwritten Recognition

In accordance with the time the identification takes place, there are two different forms of handwriting recognition.

1. online handwriting recognition

Online handwriting recognition entails the automatic conversion of text as it is written on a special digitizer or digital pad with a sensor that detects the movements of the pen tip and uses this dynamic data to assess the character and words as they are being typed.

2. Offline handwriting recognition

Offline handwriting recognition entails automatically converting a text image into letter codes that can be accessed by computers and text-processing software. The information gathered in this way is a still image of the handwriting. Also, Offline recognition is more difficult to do accurately without knowledge of pen pressure, stroke direction, etc. However, given the requirement for digitizing current historical and archive documents, it is still in great demand.

Figure below illustrate the two methods of arabic handwritten recognition



V7

Figure 3 Online vs Offline Handwriting recognition system

4.4 Handwritten Recognition Advantages

The many common applications of handwriting recognition make it valuable in a variety of sectors. Let's go over some advantages of using this technology.

❖ Improved Data Storage

The best possible data storage is made possible via handwriting recognition. Many documents, contracts, and personal records contain handwritten data that can be electronically converted using handwritten text recognition methods, such as original signatures or notes.

Compared to storing physical files, electronic data takes less physical space and financial resources. It saves money and eliminates the need to manually filter through paper documents to locate the information you need.

❖ Information Retrieval That Is Quicker

We can access data much more quickly from electronic storage than from physical copies thanks to handwriting recognition technology. By doing a file search and putting

the information we're looking for in the search box, we can locate stored electronic data rapidly.

This is comparable to search engine optimization techniques used by the IT sector. Finding information in the sea of content was made simple by indexing Internet resources.

❖ **Improved Client Services**

Handwriting analysis can assist streamline business operations and increase customer convenience and security.

For quicker access and more cost-effective storage, many companies can quickly digitize handwritten documents submitted by their clients. Additionally, organizations like banks, hospitals, and insurance businesses that deal with personal data can store their records safely in the cloud. When compared to maintaining hard copies, the scanned data requires adequate authorization to access, reducing the chance of security breaches.

4.5 The Challenge Of Arabic Handwritten Recognition

The demand for an Arabic Handwritten Optical Character Recognition (AHOCR) system is high because there are numerous applications involving handwritten papers, yet there are several challenges that must be overcome:

- There are 28 letters in the Arabic alphabet. Each can be used in one of two to four different ways, depending on where the letter is located inside the word or subword. The morphemes match the four positions: word beginning (sub), middle, end, and final. The letter (sub), alone, at the conclusion of the word. Figure 1 displays each letter's shape. The "components" of letters that do not exhibit their initials or middles are just their isolated forms, while the "intermediate" forms of such letters are their ultimate forms [11].

Name	Initial	Medial	Final	Separate	Pronunciation
alif*	ا	ا	ا	ا	see opposite
baa'	ب	ب	ب	ب	b
taa'	ت	ت	ت	ت	t
thaa'	ث	ث	ث	ث	th
jiim	ج	ج	ج	ج	j
Haa'	ح	ح	ح	ح	H
khaa'	خ	خ	خ	خ	kh
daal*	د	د	د	د	d
dhaal*	ذ	ذ	ذ	ذ	dh
raa'*	ر	ر	ر	ر	r
zaay*	ز	ز	ز	ز	z
siin	س	س	س	س	s
shiin	ش	ش	ش	ش	sh
Saad	ص	ص	ص	ص	S
Daad	ض	ض	ض	ض	D
Taa'	ط	ط	ط	ط	T
DHaa'	ظ	ظ	ظ	ظ	DH
:ain	ع	ع	ع	ع	:
ghain	غ	غ	غ	غ	gh
faa'	ف	ف	ف	ف	f
qaaf	ق	ق	ق	ق	g
kaaf	ك	ك	ك	ك	k
laam	ل	ل	ل	ل	l
miim	م	م	م	م	m
nuun	ن	ن	ن	ن	n
haa'	ه	ه	ه	ه	h
waaw	و	و	و	و	w
yaa'	ي	ي	ي	ي	y
on alif	آ	آ	آ	آ	

Figure 4 Arabic letters in a different position [11].

- Arabic employs a variety of external symbols, including dots (single, double, or triple), "hamza," and "maddah" as shown in figure 2 [12]

دَقَّار هِيشَر المَرْوَنَة أَ كَو دَة

Figure 5. Handwritten words with secondary markings circled: hamza, shadda, and madda [12].

- - As seen in Figure 3, the same character might differ significantly in length across several texts.

محمد محمد محمد

Figure 6 the shape of the same words [12].

- The similarity of the letters as seen in their major body (primary shape), where the curves are, and where the points (if any), like ع , ح (and خ) [13].

My solution will use a CNN which performs better with image classification problems as it can successfully boil down a given image into a highly abstracted representation through the layer filters which makes the prediction process more accurate.

4.6 The Use Of Arabic Handwritten Recognition

This technology has already begun to be adopted by numerous industries:

- Forms, tax receipts, and transaction history are digitalized in the insurance and banking industries using e-pdfs and signature verification.
- The retail sector keeps track of consumer transactions and bills.
- Healthcare facilities utilize electronic health records (EHR) and other digital data initiatives to prevent errors brought on by illegible prescriptions.
- HWR technologies are used by logistics organizations to scan bills of lading and sort packages by tag detection.

5. Analysis of the model

5.1 Datasets and input

For Arabic letters and Arabic digits, respectively, we'll use these datasets, Arabic Digits and Arabic Letters. Each dataset is a csv file containing the values of the image pixels and their appropriate label. More information about the datasets is provided below:

1. Arabic Numbers 60,000 training images and 10,000 test images make up the modified Arabic handwritten digits database known as MADBase, which is represented by the dataset. In MADBase, 700 authors contributed. Each digit (from 0 to 9) was written ten times by each author. The database was compiled from several organizations, including colleges of engineering and law, schools of medicine, the Open University (whose

students are of all ages), a high school, and a governmental organization, to guarantee that it included a variety of writing styles. MADBase is free to download.

2. The 16,800 characters in the Arabic Letters Dataset were written by 60 volunteers who ranged in age from 19 to 40. 90% of the participants wrote with their right hand. Each participant wrote the letters "alef" through "yeh" ten times. The photos were scanned at a 300 dpi resolution. Using Matlab 2016a, each block is automatically segmented to determine its coordinates. A training set (13,440 characters to 480 images per class) and a test set are created from the database (3,360 characters to 120 images per class). Training set and test set writers are distinct authors. To ensure that test set writers are not from a single university, the order in which they are included is randomized.

The figure below present some sample of the dataset that we will used it during this model.

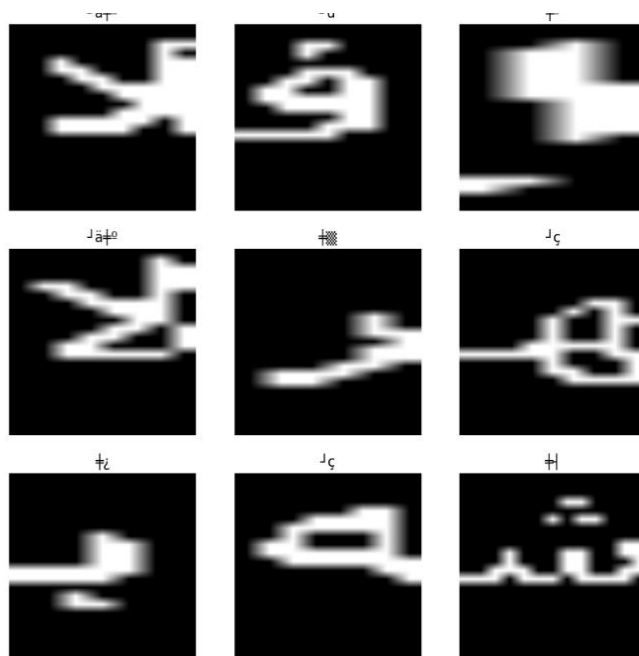


Figure 7 sample of the dataset

5.2 Implementation

We will follow the following workflow to build our model.

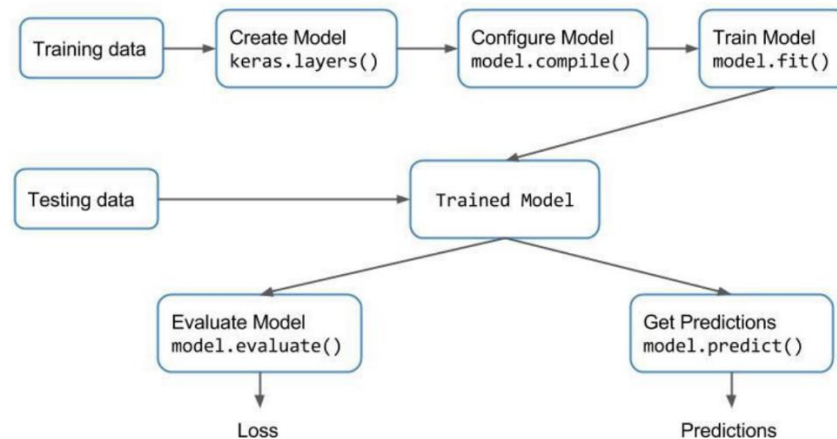


Figure 8 the methodology of Arabic handwritten recognitions

The interesting Python project requires you to have basic knowledge of Python programming, and deep learning with the Keras library. Keras is a powerful and easy-to-use free open-source Python library for developing and evaluating *deep learning* models [14]. It is part of the TensorFlow library and allows you to define and train neural network models in just a few lines of code.

Also, we need to create a new environment in Anaconda and called it the CNN project, and install pandas and the TensorFlow library. Also, we need to install a jupyter notes to implements our model.

Below are the steps to implement the handwritten digit and characters recognition project:

➤ Import the libraries and load the dataset

Machine learning is the most algorithm-intensive field in computer science. Gone are those days when people had to code all algorithms for machine learning. Thanks to Python and its libraries, modules, and frameworks.

Python machine-learning libraries have grown to become the most preferred language for machine-learning algorithm implementations. Learning Python is essential to master data science and machine learning.

So the first steps is import the importance dataset they we will used during build this model and the figure bellow present that.

```
#import Libraries
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.constraints import maxnorm
from keras.layers.convolutional import Conv2D
from keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPool2D, BatchNormalization
from keras.utils import np_utils
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
from tensorflow.keras.preprocessing import image
import keras
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import pandas as pd
#import seaborn
from sklearn.metrics import confusion_matrix, classification_report
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras import layers
```

Figure 9 import the libraries

Note that there are some of the library we need to install using the command pip install and the name of the library.

After that we need to loading the dataset which was in two zip file with name the first one was Arabic Handwritten Characters Dataset CSV-20230115T042302Z-001.zip and the other was Arabic Handwritten Digits Dataset CSV-20230115T042302Z-001.zip so first of all we need to un zip the files after that extract all the files in the folders which named Arabic dataset digits and Arabic Character Dataset and the figure bellow show that.

```

#The zipped file name is dataset.zip
!unzip "Arabic Handwritten Characters Dataset CSV-20230115T042302Z-001.zip" -d "datasetcharacter"
!unzip "Arabic Handwritten Digits Dataset CSV-20230115T042309Z-001.zip" -d "datasetdigits"

'unzip' is not recognized as an internal or external command,
operable program or batch file.
'unzip' is not recognized as an internal or external command,
operable program or batch file.

we need to extract all the files from the zip file to dataset character

import zipfile
with zipfile.ZipFile("Arabic Handwritten Characters Dataset CSV-20230115T042302Z-001.zip","r") as zip_ref:
    zip_ref.extractall("datasetcharacter")

import zipfile
with zipfile.ZipFile("Arabic Handwritten Digits Dataset CSV-20230115T042309Z-001.zip","r") as zip_ref:
    zip_ref.extractall("datasetdigits")

```

Figure 10 unzip of the dataset

So let's loading the dataset for characters and digits as mentioned below in details

1. Loading the Arabic digits dataset

There are 60000 training Arabic digit images of 64x64 pixels and 10000 testing Arabic digit images of 64x64 pixels. The figure below illustrate the loading of Arabic dataset as csv file and splitting to testing and training

loading the arabic numbers dataset

```

|: # Training digits images and Labels files
digits_training_images_file_path = "datasetdigits/Arabic Handwritten Digits Dataset CSV/training images.zip"
digits_training_labels_file_path = "datasetdigits/Arabic Handwritten Digits Dataset CSV/training labels.zip"
# Testing digits images and Labels files
digits_testing_images_file_path = "datasetdigits/Arabic Handwritten Digits Dataset CSV/testing images.zip"
digits_testing_labels_file_path = "datasetdigits/Arabic Handwritten Digits Dataset CSV/testing labels.zip"

# Loading dataset into dataframes
training_digits_images = pd.read_csv(digits_training_images_file_path, compression='zip', header=None)
training_digits_labels = pd.read_csv(digits_training_labels_file_path, compression='zip', header=None)
testing_digits_images = pd.read_csv(digits_testing_images_file_path, compression='zip', header=None)
testing_digits_labels = pd.read_csv(digits_testing_labels_file_path, compression='zip', header=None)

# print statistics about the dataset
print("There are %d training arabic digit images of 64x64 pixels." %training_digits_images.shape[0])
print("There are %d testing arabic digit images of 64x64 pixels." %testing_digits_images.shape[0])

There are 60000 training arabic digit images of 64x64 pixels.
There are 10000 testing arabic digit images of 64x64 pixels.

```

Figure 11 loading the Arabic digits dataset

2. Loading the Arabic characters dataset

There are 13440 training Arabic digit images of 64x64 pixels and 3360 testing Arabic digit images of 64x64 pixels in csv file and the dataset also here was in zip file so we used the parameters compression assign to zip

```
[7]: # Training letters images and Labels files
training_letters_images_file_path = "datasetcharacter/arabic Handwritten Characters Dataset CSV/training_images.zip"
letters_training_labels_file_path = "datasetcharacter/arabic Handwritten Characters Dataset CSV/training_labels.zip"

# Testing letters images and Labels files
letters_testing_images_file_path = "datasetcharacter/arabic Handwritten Characters Dataset CSV/testing_images.zip"
letters_testing_labels_file_path = "datasetcharacter/arabic Handwritten Characters Dataset CSV/testing_labels.zip"

# Loading dataset into datframes
training_letters_images = pd.read_csv(letters_training_images_file_path, compression='zip', header=None)
training_letters_labels = pd.read_csv(letters_training_labels_file_path, compression='zip', header=None)
testing_letters_images = pd.read_csv(letters_testing_images_file_path, compression='zip', header=None)
testing_letters_labels = pd.read_csv(letters_testing_labels_file_path, compression='zip', header=None)

# print statistics about the dataset
print("There are %d training arabic letter images of 64x64 pixels." % training_letters_images.shape[0])
print("There are %d testing arabic letter images of 64x64 pixels." % testing_letters_images.shape[0])
training_letters_images.head()
```

There are 13448 training arabic letter images of 64x64 pixels.
There are 3360 training arabic letter images of 64x64 pixels.

```
Out[7]:
```

Figure 12 loading the Arabic characters dataset

So, let's visualize some images to understand the inputs we will deal with in this model which are Arabic characters and numbers as the figure below show that , but first we need to convert the csv input to image

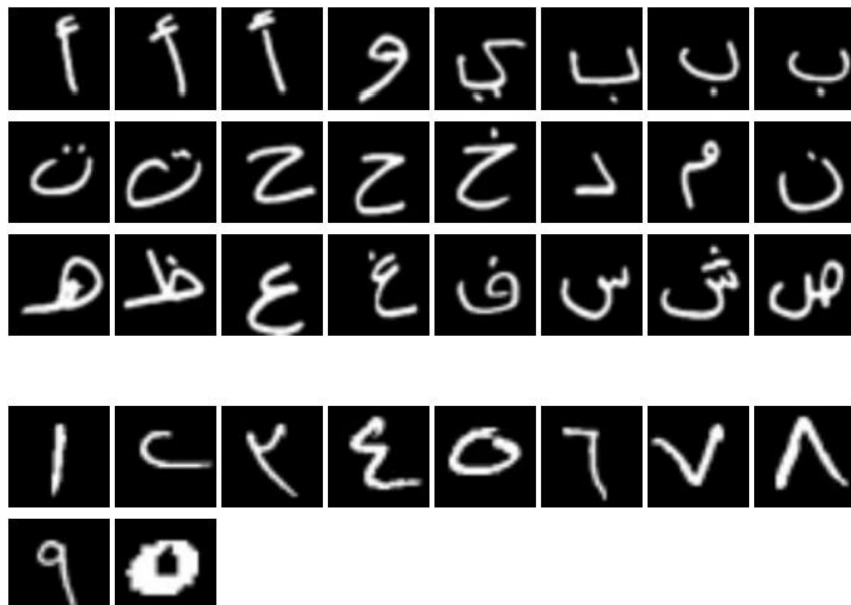


Figure 13 sample of our dataset

Now we loading the dataset successfully lets preparing the dataset to build the model.

➤ Preprocessing The Dataset

Image Normalization

We rescale the images by dividing every pixel in the image by 255 to make them into range $[0, 1]$ as the figure below show that.

```

Training images of digits after scaling
(60000, 4096)

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]], dtype=float32)

```

Figure 14 the dataset after normalization

➤ Encoding Categorical Labels

From the labels csv files we can see that labels are categorical values and it is a multi-class classification problem. Our outputs are in the form of Digits from 0 to 9 have categories numbers from 0 to 9 Letters from 'alef' to 'yeh' have categories numbers from 10 to 37 Here we will encode these categories values using One Hot Encoding with keras. One-hot encoding transforms integer to a binary matrix where the array contains only one '1' and the rest elements are '0'.

Figure 15 illustrate the output of our encoding

```

: print(training_digits_labels_encoded)

[[1. 0. 0. ... 0. 0. 0.]
 [0. 1. 0. ... 0. 0. 0.]
 [0. 0. 1. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]

```

Figure 15 encoding outputs

➤ Reshaping Input Images to 64x64x1

When using TensorFlow as backend, Keras CNNs require a 4D array (which we'll also refer to as a 4D tensor) as input, with shape (nb_samples,rows,columns,channels) where nb_samples corresponds to the total number of images (or samples), and rows, columns, and channels correspond to the number of rows, columns, and channels for each image, respectively.

So we will reshape the input images to a 4D tensor with shape (nb_samples, 64, 64 ,1) as we use grayscale images of 64x64 pixels.

And now, after the dataset are ready we will merging the dataset merging the letters and numbers together. Figure 16 show that the dataset combining together with 73440 for training and

13360 for testing with 38 classes which are 0-9 for numbers and 10-37 for letters as mentioned below.

```
:
training_data_images = np.concatenate((training_digits_images_scaled, training_letters_images_scaled), axis=0)
training_data_labels = np.concatenate((training_digits_labels_encoded, training_letters_labels_encoded), axis=0)
print("Total Training images are {} images of shape".format(training_data_images.shape[0]))
print(training_data_images.shape, training_data_labels.shape)

testing_data_images = np.concatenate((testing_digits_images_scaled, testing_letters_images_scaled), axis=0)
testing_data_labels = np.concatenate((testing_digits_labels_encoded, testing_letters_labels_encoded), axis=0)
print("Total Testing images are {} images of shape".format(testing_data_images.shape[0]))
print(testing_data_images.shape, testing_data_labels.shape)

Total Training images are 73440 images of shape
(73440, 64, 64, 1) (73440, 38)
Total Testing images are 13360 images of shape
(13360, 64, 64, 1) (13360, 38)
```

Figure 16 merging the Arabic letters and numbers dataset

➤ Create And Compile The Model

After loading the dataset successfully, now we will build the model using the CNN and to be note that Machine learning classification techniques like SVM, Decision Tree, etc. are available. However, in this case, deep learning neural networks will be used, which frequently yield superior results than ML algorithms, particularly when dealing with intricate patterns like those found in photographs. Therefore, in order to recognize the photos with good performance, we will develop a CNN model.

Convolutional, nonlinear, pooling (downscaling), and fully connected layers are applied to the input image by CNNs to produce the output. A single class or a probability of classes that best describe the image may be the result, as we previously stated. The challenging thing now is figuring out what each of these levels performs. So let's move on to the most crucial.

Convolutional layer is the top hidden layer. The layer comprises 16 feature maps, each measuring 3 by 3, with a Relu activation function. This layer serves as the input and anticipates photos with the aforementioned structure.

The second layer, batch normalization, fixes the IID assumption being broken by feature distributions that differ between training and test data. It assists us in two ways: it speeds up learning and increases general accuracy.

The MaxPooling layer is the third layer. In order to let the model to make assumptions about the features and prevent overfitting, the input is down-sampled using the MaxPooling layer. The quantity of parameters to learn is also decreased, which cuts down on training time.

A Regularization layer employing dropout is the following layer. To avoid overfitting, it is set up to randomly omit 20% of the layer's neurons.

A second hidden layer was created using 32 3x3 feature maps and a relu activation mechanism to extract additional features from the image.

Other concealed layers with 64 and 128 feature maps of equal size and a relu activation function are used to extract intricate patterns from the image that will subsequently be used to characterize the numbers and characters. Additional levels of MaxPooling, Batch Normalization, Regularization, and GlobalAveragePooling2D.

With 38 neurons (the number of output classes), the output layer is the last layer, and it utilizes the softmax activation function because there are several classes. Each neuron will provide the class's probability.

Because the problem is a multi-class classification problem, I chose categorical crossentropy as a loss function. Accuracy was another parameter I used to gauge how well our neural network was working.

Take note of the hyperparameters the model used:

- 20% of the layer nodes dropped off.
- Epochs: 10; after that, the model will be gradually fitted over a further 20 epochs.
- Batch size: 20 since it is a sufficient number and divides by both the total size of the training dataset and the total size of the testing dataset.
- Optimizer: Following the section on refinement, we shall find that Adam is the optimal optimizer to employ.
- Activation Layer: Following the section on refinement, we shall see that the optimal activation to utilize is relu.
- Kernel initializer: The best kernel initializer will be displayed after the refinement step.

We will adjust the kernel initializer, optimizer, and activation function parameters to observe how they affect our model design. The following potential values for these parameters will be used while running GridSearch:

- RMSprop, Adam, Adagrad, and Nadam are the optimizers.
- ['normal', 'uniform'] kernel initializer
- ['relu', 'linear', 'tanh'] activation

The figure below the result of some of them

```
{'optimizer': 'RMSprop', 'kernel_initializer': 'normal', 'activation': 'relu'}
Train on 73440 samples, validate on 13360 samples
Epoch 1/5
73440/73440 [=====] - 84s 1ms/step - loss: 0.3349 - acc: 0.9089 - val_loss: 2.4386 - val_acc: 0.5037
Epoch 2/5
73440/73440 [=====] - 82s 1ms/step - loss: 0.1106 - acc: 0.9670 - val_loss: 0.1605 - val_acc: 0.9507
Epoch 3/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.0865 - acc: 0.9738 - val_loss: 0.1138 - val_acc: 0.9665
Epoch 4/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.0788 - acc: 0.9772 - val_loss: 2.5557 - val_acc: 0.5233
Epoch 5/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.0710 - acc: 0.9788 - val_loss: 0.1519 - val_acc: 0.9559
=====
{'optimizer': 'RMSprop', 'kernel_initializer': 'uniform', 'activation': 'relu'}
Train on 73440 samples, validate on 13360 samples
Epoch 1/5
73440/73440 [=====] - 83s 1ms/step - loss: 0.3012 - acc: 0.9181 - val_loss: 0.3428 - val_acc: 0.8807
Epoch 2/5
73440/73440 [=====] - 82s 1ms/step - loss: 0.1075 - acc: 0.9672 - val_loss: 0.1243 - val_acc: 0.9611
Epoch 3/5
73440/73440 [=====] - 82s 1ms/step - loss: 0.0875 - acc: 0.9734 - val_loss: 0.0698 - val_acc: 0.9793
Epoch 4/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.0793 - acc: 0.9761 - val_loss: 7.7491 - val_acc: 0.3073
Epoch 5/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.0727 - acc: 0.9780 - val_loss: 3.6094 - val_acc: 0.5719
=====
{'optimizer': 'RMSprop', 'kernel_initializer': 'normal', 'activation': 'linear'}
Train on 73440 samples, validate on 13360 samples
Epoch 1/5
73440/73440 [=====] - 81s 1ms/step - loss: 0.6358 - acc: 0.8246 - val_loss: 0.5407 - val_acc: 0.8210
Epoch 2/5
73440/73440 [=====] - 80s 1ms/step - loss: 0.2895 - acc: 0.9135 - val_loss: 0.3693 - val_acc: 0.8883
Epoch 3/5
73440/73440 [=====] - 80s 1ms/step - loss: 0.2099 - acc: 0.9368 - val_loss: 0.1959 - val_acc: 0.9409
Epoch 4/5
73440/73440 [=====] - 80s 1ms/step - loss: 0.1700 - acc: 0.9478 - val_loss: 2.2486 - val_acc: 0.7231
Epoch 5/5
73440/73440 [=====] - 80s 1ms/step - loss: 0.1494 - acc: 0.9535 - val_loss: 1.5786 - val_acc: 0.6550
=====
{'optimizer': 'Adam', 'kernel_initializer': 'normal', 'activation': 'relu'}
Train on 73440 samples, validate on 13360 samples
Epoch 1/5
73440/73440 [=====] - 90s 1ms/step - loss: 0.3496 - acc: 0.9075 - val_loss: 6.2671 - val_acc: 0.2249
Epoch 2/5
73440/73440 [=====] - 89s 1ms/step - loss: 0.1066 - acc: 0.9679 - val_loss: 0.2868 - val_acc: 0.9138
Epoch 3/5
73440/73440 [=====] - 89s 1ms/step - loss: 0.0800 - acc: 0.9755 - val_loss: 0.1065 - val_acc: 0.9688
Epoch 4/5
73440/73440 [=====] - 89s 1ms/step - loss: 0.0692 - acc: 0.9782 - val_loss: 0.0776 - val_acc: 0.9775
Epoch 5/5
73440/73440 [=====] - 89s 1ms/step - loss: 0.0581 - acc: 0.9820 - val_loss: 0.2886 - val_acc: 0.9206
=====
{'optimizer': 'Adam', 'kernel_initializer': 'uniform', 'activation': 'relu'}
Train on 73440 samples, validate on 13360 samples
Epoch 1/5
73440/73440 [=====] - 92s 1ms/step - loss: 0.3284 - acc: 0.9116 - val_loss: 1.2440 - val_acc: 0.7041
Epoch 2/5
73440/73440 [=====] - 90s 1ms/step - loss: 0.1081 - acc: 0.9666 - val_loss: 1.4073 - val_acc: 0.7209
Epoch 3/5
73440/73440 [=====] - 90s 1ms/step - loss: 0.0826 - acc: 0.9748 - val_loss: 0.5529 - val_acc: 0.8179
Epoch 4/5
73440/73440 [=====] - 90s 1ms/step - loss: 0.0669 - acc: 0.9794 - val_loss: 0.0744 - val_acc: 0.9786
Epoch 5/5
73440/73440 [=====] - 90s 1ms/step - loss: 0.0618 - acc: 0.9812 - val_loss: 0.0576 - val_acc: 0.9831
=====
```

Figure 17 details about the results of some parameters

And after compiling the model and from the obtained results we can see that best parameters are:

- Optimizer: Adam
- Kernel_initializer: uniform
- Activation: relu

➤ Evaluate the model

To save memory and speed up the training, we will train the model with batch size=20. The model will be trained initially on 10 epochs to see the accuracy we will achieve, and then the number of epochs will be increased to be trained on in order to raise the accuracy.

Here are the outcomes following 10 epochs.

```

Train on 73440 samples, validate on 13360 samples
Epoch 1/10
73440/73440 [=====] - 94s 1ms/step - loss: 0.3189 - acc: 0.9153 - val_loss: 10.5339 - val_acc: 0.1437

Epoch 00001: val_loss improved from inf to 10.53390, saving model to weights.hdf5
Epoch 2/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.1049 - acc: 0.9681 - val_loss: 0.3283 - val_acc: 0.9012

Epoch 00002: val_loss improved from 10.53390 to 0.32826, saving model to weights.hdf5
Epoch 3/10
73440/73440 [=====] - 90s 1ms/step - loss: 0.0797 - acc: 0.9757 - val_loss: 0.6523 - val_acc: 0.7985

Epoch 00003: val_loss did not improve from 0.32826
Epoch 4/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0702 - acc: 0.9782 - val_loss: 0.2071 - val_acc: 0.9394

Epoch 00004: val_loss improved from 0.32826 to 0.20706, saving model to weights.hdf5
Epoch 5/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0592 - acc: 0.9811 - val_loss: 0.0752 - val_acc: 0.9769

Epoch 00005: val_loss improved from 0.20706 to 0.07522, saving model to weights.hdf5
Epoch 6/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0545 - acc: 0.9833 - val_loss: 0.9768 - val_acc: 0.6818

Epoch 00006: val_loss did not improve from 0.07522
Epoch 7/10
73440/73440 [=====] - 88s 1ms/step - loss: 0.0497 - acc: 0.9845 - val_loss: 0.1118 - val_acc: 0.9671

Epoch 00007: val_loss did not improve from 0.07522
Epoch 8/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0473 - acc: 0.9854 - val_loss: 0.0576 - val_acc: 0.9829

Epoch 00008: val_loss improved from 0.07522 to 0.05756, saving model to weights.hdf5
Epoch 9/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0453 - acc: 0.9863 - val_loss: 0.0000 - val_acc: 0.9767

Epoch 00009: val_loss did not improve from 0.05756
Epoch 10/10
73440/73440 [=====] - 89s 1ms/step - loss: 0.0432 - acc: 0.9862 - val_loss: 0.2793 - val_acc: 0.9097

Epoch 00010: val_loss did not improve from 0.05756

```

Figure 18 the model after 10 epochs

The model after compiling 10 epochs didn't present any overfitting according to the plot which show in figure 19 that we are still safe from overfitting as both training accuracy and validation accuracy are increasing with increasing in epochs. Figure 20 illustrate clearly the overfitting when we increasing the epochs to 20 which means we didn't need to increasing the epochs and stop in 10 epochs.

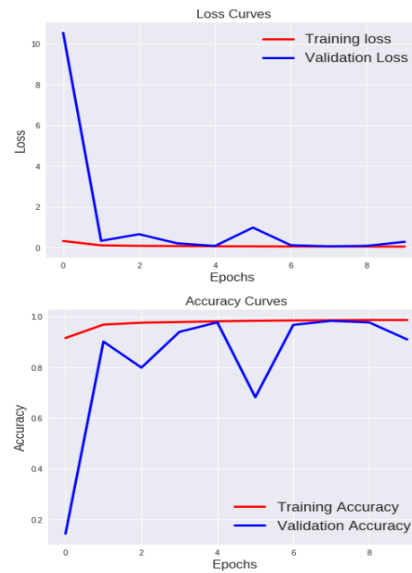


Figure 19 the check of overfitting at epochs 10

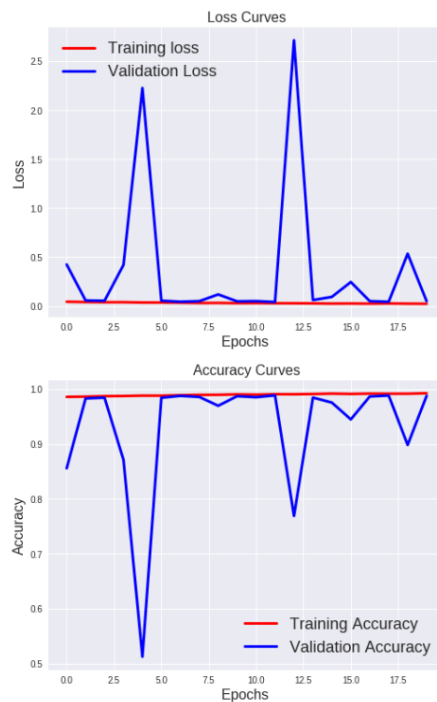


Figure 20 the check of overfitting as epochs 20

So, we will stop fitting the model at that point we have already got very satisfied score on the testing dataset (we got test accuracy at 98.06%).

6. Result

We are attempting to resolve the problem of Arabic handwritten picture recognition, as we described in the problem description section. Arabic handwriting poses special technical difficulties and has just lately been studied compared to other disciplines. Many alternative approaches have been suggested and used on a variety of image formats, but in this case we'll use CNN to approach the issue from a different angle.

The challenges we face here are numerous, starting with the necessity of accounting for the numerous variations in human handwriting using a large variations dataset, followed by the preprocessing of images of various sizes by reshaping them, and finally, image normalization so that all the values are nearly of equal importance. Choosing a suitable model to be able to gather the data is the next major problem.

We will train the model using batch size =32 to reduce used memory and make the training more quick. We will train the model first on 10 epoch to see the accuracy that we will obtain then we will increase the number of epochs to be trained to improve the accuracy.

The figure below showed the final which provide a justification result with 98.06%

Test the Model

```
21]: # Final evaluation of the model
metrics = model.evaluate(testing_data_images, testing_data_labels, verbose=1)
print("Test Accuracy: {}".format(metrics[1]))
print("Test Loss: {}".format(metrics[0]))


418/418 [=====] - 14s 31ms/step - loss: 0.0636 - accuracy: 0.9806
Test Accuracy: 0.9806137681007385
Test Loss: 0.0635581985116005
```

Figure 21 the final result of test model

After that we build a Demo to see how the model works on examples of the testing data as assign Digits from 0 to 9 were encoded to categorical labels from 0 to 9 Letters from 'alef'='ا' to 'yeh'='ي' were encoded to categorical labels from 10 to 37 and we predict example as mentioned below in figure 21

```
J: sample_index = 5
show_example(sample_index)
convert_values_to_image(testing_digits_images.loc[sample_index], True)

1/1 [=====] - 0s 357ms/step
The following image has the written character '5' but the model predicted it as '5'
```



```
J: sample_index = 11414
show_example(sample_index)
convert_values_to_image(testing_letters_images.loc[sample_index - testing_digits_images.shape[0]], True)

1/1 [=====] - 0s 28ms/step
The following image has the written character 'ا' but the model predicted it as 'ا'
```




Figure 22 the prediction of the new images

7. Conclusion and future work

In this project , I built a CNN model which can classify the Arabic handwritten images into digits and letters. We tested the model on more than 13000 image with all possible classes and got very high accuracy 98.8% which is much better score.

The final model captured almost all the dataset, so if want to improve it we should include more datasets which have variations in the handwritten style although we think we already include much variations in our dataset.

As another improvement we think we can enhancement the model to text recognitions also we can build a GUI app in android to predicted the new images and it's accuracy.

8. Acknowledge

All thanks to the wonderful engineer, Ruba Al-Khushieny, for everything she gave us, and she was the main reason for me learning Python, writing the code and solving programming problems easily.

I also thank Al-Hussein Technical University for this wonderful initiative, and we thank all those in charge of it and for this most wonderful training program

In the end, I thank all my colleagues in this session, in which we spent the most beautiful days together, with all love for them

Ayat

9. References

References

- [1] U. Porwal, Z. Shi and S. Setlur, "Machine learning in handwritten Arabic text recognition," *In Handbook of Statistics* , pp. 443-469, 2013.
- [2] N. Lang, "using-convolutional-neural-network-for-image-classification," 5 Dec 2021. [Online]. Available: <https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4>.
- [3] A. El-Sawy, M. Loey and H. El-Bakry, "Arabic handwritten characters recognition using convolutional neural network," *WSEAS Transactions on Computer Research*, pp. 11-19, 2017.
- [4] J. Le, "the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images," 7 oct 2018. [Online]. Available: <https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d>.
- [5] M. N. AlJarrah, M. Z. Mo'ath and R. Duwairi, "Arabic handwritten characters recognition using convolutional neural network," in *12th International Conference on Information and Communication Systems (ICICS) IEEE.*, 2021.
- [6] N. Das, F. AMollah, S. Saha and S. S. Haque, "Handwritten arabic numeral recognition using a multi layer perceptron," *arXiv preprint arXiv*, 2010.
- [7] U. N. NETWORK, "ARABIC ONLINE HANDWRITING RECOGNITION.," *International Journal of Artificial Intelligence and Applications (IJAAI)*, 2016.
- [8] R. Alaasam, B. Kurar, M. Kassis and El-Sana, J., "Experiment study on utilizing convolutional neural networks to recognize historical Arabic handwritten text.," in *In1st International Workshop on Arabic script analysis and recognition (ASAR).IEEE*, 2017.
- [9] Younis, K. S., "Arabic hand-written character recognition based on deep convolutional neural networks.," *Jordanian Journal of Computers and Information Technology*, 2017.
- [10] M. Loey, A. El-Sawy and H. El-Bakry, "Deep learning autoencoder approach for handwritten arabic digits recognition," in *International Conference on Advanced Intelligent Systems and Informatics*, 2017.
- [11] L. M. Lorigo and V. Govindaraju, "Offline Arabic handwriting recognition: a survey.," in *IEEE transactions on pattern analysis and machine intelligence*, 2006.

- [12] H. M. Balaha, H. A. Ali, E. K. Youssef, A. E. Elsayed, R. A. Samak, M. S. Abdelhaleem and M. M. Mohammed, "Recognizing arabic handwritten characters using deep learning and genetic algorithms," in *Multimedia Tools and Applications*, 2021.
- [13] M. A. Al-Jubour, "Offline Arabic Handwritten Isolated Character Recognition System Using Support vector Machine and Neural Network.," *Journal of Theoretical & Applied Information Technology*, p. 95(10)., 2017.
- [14] J. Brownlee, "tutorial-first-neural-network-python-keras," 18 june 2022. [Online]. Available: <https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>.
- [15] S. Abdleazeem and E. El-Sherif, "Arabic handwritten digit recognition.," *International Journal of Document Analysis and Recognition (IJ DAR)*, pp. 127-141, 2008.
- [16] unknown, "MNIST_database," [Online]. Available: https://en.wikipedia.org/wiki/MNIST_database.
- [17] "top-python-libraries-for-machine-learning," 3 oct 2022. [Online]. Available: <https://www.upgrad.com/blog/top-python-libraries-for-machine-learning/>.
- [18] unknown, "scikit-learn," 2020. [Online]. Available: <https://scikit-learn.org/stable/>.
- [19] unknown, "top-python-libraries-for-machine-learning," 3 oct 2022. [Online]. Available: <https://www.upgrad.com/blog/top-python-libraries-for-machine-learning/>.
- [20] "Elsayed, A. E., Samak, R. A., Abdelhaleem, M. S., ... & Mohammed, M. M.. Recognizing arabic handwritten characters using deep learning and genetic algorithms," in *Multimedia Tools and Applications*, 2021.

