Arabic Handwriting Recognition

Nour Nasser	19105525
Abdelrhman Hassan	19104406
Ramez Adel	19100730
Hussein Ehab	19100556
Mahmoud Ibrahim	19100242

Under supervision of Dr/Ghada Khoriba

Table of content

Problem Summary 3

Dataset 3

Methodology 4

Data Preparation 4

Data Preprocessing 5

Encoding Categorical Labels: 5

One-Hot Encoding: 5

Reshaping Input Images to 64x64x1: 5

Merging Letters and Digits Datasets: 6

Model Architecture 6

Training the Model 7

Model Evaluation 8

Improving the Model 8

Skills Gained 8

References: 9

Problem Summary

The automatic recognition of text on scanned images has enabled many applications, such as searching for words in large volumes of documents, automatic sorting of postal mail, and convenient editing of previously printed documents. The domain of handwriting in the Arabic script presents unique technical challenges and has been addressed more recently than in other domains. Many different methods have been proposed and applied to various types of images. This project aims to develop a system for recognizing Arabic handwritten images. The ability to automatically recognize text in scanned images has led to various applications such as searching for words in a large number of documents, sorting postal mail, and editing previously printed documents. Recognizing Arabic handwriting presents unique challenges and has been addressed more recently than other domains. There are many different methods that have been proposed and applied to various types of images. In this project, we will focus on recognizing handwritten Arabic letters and digits, which faces several challenges such as the unlimited variation in human handwriting and large public databases. We will use multiple deep learning models to classify the images into Arabic letters or digits. This project is of great importance because Arabic is a widely used language, and there is a need for an accurate and efficient system for recognizing Arabic handwritten text. The project will be implemented using deep learning to process the handwritten images.

Dataset

The dataset used in this project is sourced from Kaggle kernels and consists of two parts: Arabic Digits and Arabic Letters. Each dataset is provided as a CSV file and contains image pixel values along with their corresponding labels.

The Arabic Digits Dataset, known as MADBase, comprises 60,000 training images and 10,000 test images. It was created by 700 different writers who each wrote the digits 0 to 9 ten times. To capture diverse writing styles, the database was collected from various institutions, including Colleges of Engineering and Law, School of Medicine, the Open University (with students from different age groups), a high school, and a governmental institution. MADBase can be freely downloaded from a specified location.

The Arabic Letters Dataset consists of 16,800 characters written by 60 participants between the ages of 19 and 40. Around 90% of the participants are right-handed. Each participant wrote each of the characters from 'alef' to 'yeh' ten times. The images were scanned at a resolution of 300 dpi. MATLAB 2016a was used to automatically segment each character block and determine their coordinates. The dataset is divided into a training set, containing 13,440 characters with 480 images per class, and a test set, containing 3,360 characters with 120 images per class. The writers of the training and test sets are exclusive, and the selection of writers for the test set is randomized to ensure variability and avoid a single-institution bias.

Methodology

Arabic handwriting recognition is a challenging task that requires the use of advanced deep-learning techniques. One such technique is the use of convolutional neural networks (CNNs), which have proven to be highly effective for image recognition tasks. This methodology will outline the steps required to train a CNN for Arabic handwriting recognition.

Data Preparation

The first step in any machine learning project is to prepare the data. In this case, we must gather a sizeable Arabic handwriting sample dataset. The samples should cover various handwriting styles

from different writers and different writing instruments. The dataset should be labelled with the correct Arabic characters for each sample.

Data Preprocessing

Before training the CNN, we need to preprocess the data to make it suitable for the model. This includes resizing the images to a consistent size, converting them to grayscale, and normalizing the pixel values. To normalize the images, we apply rescaling by dividing each pixel value in the image by 255. This process transforms the pixel values to a range of [0, 1].

Encoding Categorical Labels:

The labels in the CSV files are categorical values, indicating a multi-class classification problem. Our output categories are defined as follows, Digits from 0 to 9 are assigned category numbers from 0 to 9 and Letters from 'alef' to 'yeh' are assigned category numbers from 10 to 37.

One-Hot Encoding:

To encode these categorical values, we utilize the technique of One-Hot Encoding with Keras. One-hot encoding converts each integer category into a binary matrix, where only one element in the array is '1', and the rest are '0'.

Reshaping Input Images to 64x64x1:

When working with TensorFlow as the backend, Keras Convolutional Neural Networks (CNNs) require a 4D array (referred to as a 4D tensor) as input. This 4D tensor should have a shape of (nb_samples, rows, columns, channels). Here, nb_samples represents the total number of images or samples, while rows, columns, and channels represent the dimensions of each image (rows and columns) and the number of channels (in our case, 1 for grayscale images).Thus, we reshape the

input images into a 4D tensor with a shape of (nb_samples, 64, 64, 1), as we are working with grayscale images of size 64x64 pixels.

Merging Letters and Digits Datasets:
To enable training our model on both the digits and letters datasets, we merge the two datasets together. This allows us to use the combined dataset for training purposes.

Model Architecture

The next step is to design the CNN architecture. A typical CNN for image recognition consists of several convolutional layers, followed by pooling layers, and then one or more fully connected layers. The initial hidden layer in our model is a convolutional layer. It consists of 16 feature maps with a size of 3×3 and uses the rectified linear unit (ReLU) activation function. This layer serves as the input layer and expects images with the specified structure mentioned earlier. The following layer is Batch Normalization, which addresses the issue of varying feature distributions across the training and test data, breaking the independent and identically distributed (IID) assumption. By normalizing the inputs, it helps in two ways: facilitating faster learning and improving overall accuracy. Next, we have the MaxPooling layer, which performs down-sampling on the input. This allows the model to make assumptions about the features and helps reduce overfitting. Additionally, it reduces the number of parameters to learn, resulting in shorter training time. Subsequently, there is a Regularization layer that employs dropout. It is configured to randomly deactivate 20% of the neurons in the layer, reducing overfitting by introducing randomness during training. Another hidden layer follows, featuring 32 maps with a size of 3×3 and the ReLU activation function. This layer aims to capture additional features from the image. Further hidden layers are included, each with 64 and 128 feature maps of size 3×3 and the ReLU activation

function. These layers are designed to capture more complex patterns from the images, which will later help in identifying the digits and letters. The model also incorporates additional MaxPooling, Batch Normalization, Regularization, and GlobalAveragePooling2D layers.

Finally, we have the output layer, consisting of 10 neurons, corresponding to the number of output classes. Given that we are dealing with multiple classes, the softmax activation function is used. Each neuron in this layer provides the probability of the corresponding class.

Training the Model

With the data and the model architecture ready, we can train CNN. We will tune the parameters optimizer, kernel initializer and activation. We used a function to loop through different models with different parameters to find the best parameter values. We tried different optimizers such as, Adam, Nadam, Adagrad, and RMSprop, we tried setting the kernel initializer to both uniform and normal and finally we set the activation function to relu, linear, and tanh. Based on this we found that the best parameters for the model were, Adam optimizer with the kernel set to uniform and the activation function set to relu. The following figure shows the metrics of the best-found parameters.

```
================================================================================
{'optimizer': 'Adam', 'kernel_initializer': 'uniform', 'activation': 'relu'}
Epoch 1/5
3672/3672 [==============================] - 129s 35ms/step - loss: 0.3203 - accuracy: 0.9145 - val_loss: 0.2568 - val
_accuracy: 0.9202
Epoch 2/5
3672/3672 [==============================] - 120s 33ms/step - loss: 0.1046 - accuracy: 0.9676 - val_loss: 0.2302 - val
_accuracy: 0.9330
Epoch 3/5
3672/3672 [==============================] - 120s 33ms/step - loss: 0.0805 - accuracy: 0.9751 - val_loss: 2.5301 - val
_accuracy: 0.4805
Epoch 4/5
3672/3672 [==============================] - 114s 31ms/step - loss: 0.0688 - accuracy: 0.9788 - val_loss: 0.0672 - val
_accuracy: 0.9790
Epoch 5/5
3672/3672 [==============================] - 115s 31ms/step - loss: 0.0607 - accuracy: 0.9814 - val_loss: 0.3271 - val
_accuracy: 0.8977
================================================================================
```

Model Evaluation

After training the model, we evaluate its performance on the test set. We measure the model's performance using metrics such as accuracy, precision, recall. We may also use techniques such as confusion matrices and visualizations to understand how the model makes predictions. the accuracy was train accuracy: 98.74%, test accuracy of 98.286% after training on 10 epochs only. We also compared our model to a simple vanilla cnn model and our model outperformed it as the baseline accuracy was **32.37%**and our test accuracy was **98.82%**

Improving the Model

If the model's performance is not satisfactory, we may need to make changes to the model architecture, data preprocessing, or hyperparameters. We can experiment with different architectures, optimization algorithms, and learning rates to improve the model's performance. We tried to train our model with more epochs to improve our accuracy. However, our test accuracy improved from **98.286% to 98.82%**

Skills Gained

Many skills have been improved as a result of working on this project, aside from the basic skills such as working in a group, time management, and splitting tasks. We were introduced to working with TensorFlow, Keras and different libraries, which are essential for developing skills in deep learning. Also, implementing a CNN model, working with neural networks, and understanding how it works by practicing it in the project added a great experience and improved our skills in working with it.

References:

Altwaijry, N., & Al-Turaiki, I. (2021). Arabic handwriting recognition system using convolutional neural network. *Neural Computing and Applications*, *33*(7), 2249–2261. https://doi.org/10.1007/s00521-020-05070-8

Durayhim, A. B., Al-Ajlan, A., Al-Turaiki, I., & Altwaijry, N. (2023). Towards Accurate Children's Arabic Handwriting Recognition via Deep Learning. *Applied Sciences, 13*(3), 1692. https://doi.org/10.3390/app13031692