Handwritten Arabic Digits Recognition Using Neural Networks

Authors:

Khalid Al-Muqbil

Supervisor:

Dr. Nasser Alshammari

College of Computer and Information Sciences

Jouf University

April, 2019

Project in Brief

Project Title: Handwritten Arabic Digits Recognition Using Neural Networks

Organization: Jouf University

Undertaken By: Khalid Mohammed Al-Muqbil

Supervised By: Dr. Nasser Alshammari

Acknowledgment

I am highly indebted to Dr. Nasser Alshammari for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

My thanks and appreciations also go to our colleagues in developing the project and people who have willingly helped us out with their abilities.

Declaration

We hereby declare that this software, neither as a whole nor as a part has been copied out from any source. It is further declared that we have developed this software and accompanied report entirely on the basis of our personal efforts. If any part of this project is proved to be copied out from any source or found to be reproduction of some other. We will stand by the consequences. No portion of the work presented has been submitted in support of any application for any other degree or qualification of this or any other university or institute of learning.

NAME: .....

SIGNATURE: .....

NAME: .....

SIGNATURE: .....

Abstract

The task of recognizing handwritten digits is a well-established and important research area in machine and deep learning and many research efforts has been dedicate to it. However, recognizing handwritten Arabic digits is lacking compared to their English counterparts. In this study, we present a Convolutional Neural Network (CNN) model to classify handwritten Arabic digits. We evaluated the accuracy of the model using MADBase dataset, which is an Arabic dataset that resembles the famous MNIST dataset. MADBase consists of 60,000 training samples and 10,000 testing samples. The accuracy of our model reaches 99.35% which could be considered state-of-the-art accuracy amount currently published models that use MADBase.

Contents

List of Figures

List of Source Codes

Chapter 1

Introduction

Recognition in various fields is an important field in our time. It provides us a lot of time and effort, such as recognition of letters and numbers, faces recognition and pictures. The identification of handwritten numbers is important and helps to facilitate services and resolve some problems in our time. With the development of the devices contributed to a lot of the application of some of the theories proved and that has been addressed in a lot of dedicated endeavors in our time is prohibited, we have created a smart system capable of recognizing the Arabic numbers or (Indian) written by hand.

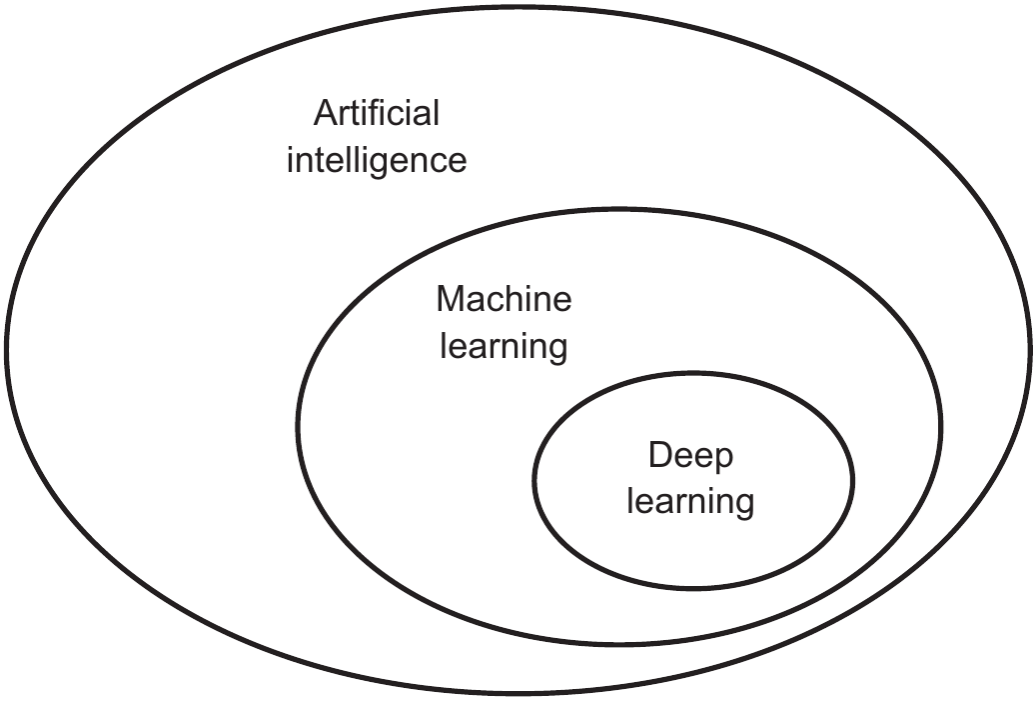

Figure 1.1: Where Deep learning fits in the general field of AI [7].

In recent years, Convolutional Neural Networks (CNN) have received increased attention as they are able to consistently outperform other approaches in virtually all fields of Machine Learning. Deep learning (DL) is a hierarchical structure network which simulates the human brain's structure to extract the data's features. Figure 1.1 shows how DL fits into the big picture of the Artificial Intelligence field. There are various DL architectures such as convolutional deep neural networks, deep belief networks, recurrent neural networks and stacked autoencoders. These algorithms allow computers and machines to model our world well enough to exhibit intelligence. Figure 1.2 shows a typical DL neural network structure.

Recognition is an area that covers various fields such as, face recognition, finger print recognition, character recognition, numerals recognition, etc. Handwritten Digit Recognition system (HDR) is an important compo-

nent in many applications; check verification, office automation, business, postal address reading and printed postal codes and data entry applications are few examples. The recognition of handwritten digits is a more difficult task due to the different handwriting styles of the writers. Deep learning techniques have achieved state-of-the-art performance in computer vision, speech recognition, and in natural language processing. This research focuses on recognizing Arabic Handwritten digits using a CNN model.

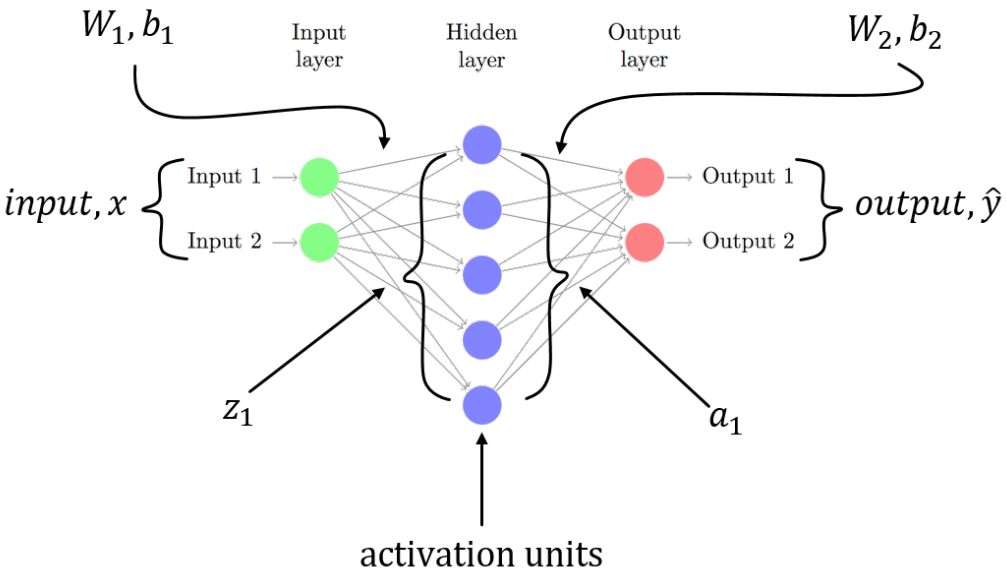

Figure 1.2: Typical neural network structure.

1.1 Aim

The aim of this project is to develop a neural network model that is capable of recognizing handwritten Arabic digits. The model should be trained on sufficiently sized database made by human volunteers and evaluated against other highly regarded model that perform the same task.

1.2 Objectives

To achieve the aim of this project, the following objectives are identified:

1. Conduct a literature review of current state-of-the-art approaches to recognize Arabic handwritten digits using neural networks based techniques.
2. Select or create an appropriate Arabic digits dataset.
3. Build a new CNN model to classify the Arabic digits.
4. Evaluate the accuracy of the model and compare it against other published models.

Chapter 2

Literature Review

Before we began implementing any classifiers, we wanted to investigate what material already existed in the domain of digits recognition. This dataset was created MADbase [10], and consists of 70,000 images of Arabic digits. These are divided into a training set of 60,000 images and a test set of 10,000. This seems to be the largest dataset for this task available in the literature. This makes it an ideal choice for training the network and fine-tuning parameters. Furthermore, as discussed in detail in the next section, previous results obtained from this dataset allow for comparison with the results presented in this manuscript. It is worth noting that this dataset is a modified version of an equivalent dataset called ADbase, which contains the same images with different image size. To create MADbase, ADbase images were resized and transformed from binary to grayscale to be equivalent to MNIST. While the MADbase dataset deals with digits, the Arabic Handwritten Character Dataset (AHCD) [5] includes 16,800 images of isolated characters divided in training set of 13,440 and a test set of 3,360 images. This seems to be the largest dataset available for this classification task. Regarding previous results, [11] presented a method for recognition of handwritten Arabic digits based on the extraction of Gabor-based features and Support Vector Machines (SVMs). The dataset used in this case contained 21,120 samples pro-

vided by 44 writers. The average classification accuracy rates obtained were of 99.85% and 97.94% using three scales and five orientations and four scales and six orientations respectively. [1] applied several classification methods to the MADbase dataset. Their best result was obtained with a Radial Basis Function Support Vector Machine (RBF SVM), with which a two-stage classification was performed. In the first stage, several customized features were extracted from a similar dataset by the researchers and then used as input for the RBF SVM. The classifier was tuned to maximize the classification accuracy, which had a final value of 99.48%. This value corresponds to the best parameter combination. [4] used a small dataset of 600 digit images to obtain a 99% recognition rate using a technique based on Loci characteristics. [15] proposed an approach for Arabic Digit recognition using neural networks and training through backpropagation. The dataset used in this case was also small, and the classification accuracy obtained was 96%. [16] obtained a test classification accuracy of 88% using a dataset of 3,510 digit images, by using a three-level classifier consisting of SVM, Fuzzy C Means and Unique Pixels. [14] presented two methods for enhancing the recognition of Arabic Handwritten Digits. The methods combine fuzzy logic pattern classification to counting the number of ends of the digit shapes to obtain a classification test accuracy of 95% for some fonts. [2] , using the ADbase dataset, obtained an 85.26% classification accuracy by using Dynamic Bayesian Networks (DBN).

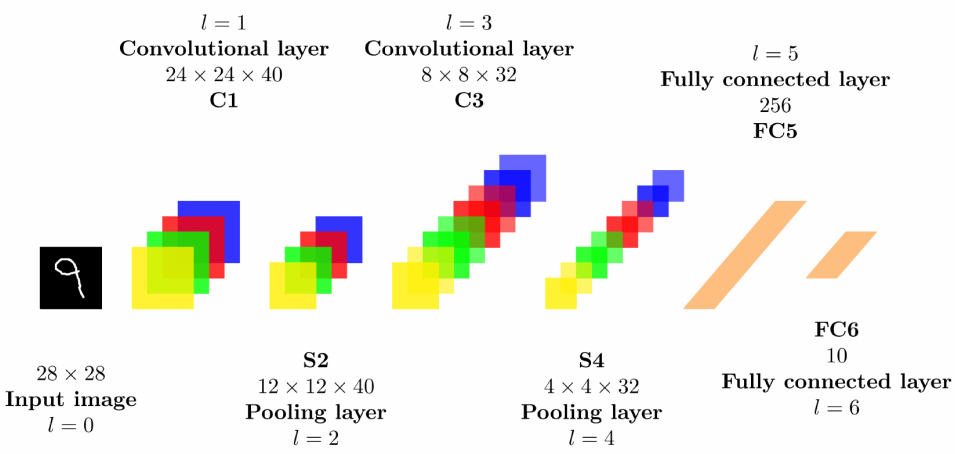

Figure 2.1: The CNN architecture used in [5].

In 2016, [5] presented a method to recognize, Arabic digits using a convolutional neural network. The result shows that the proposed method provides a recognition accuracy of overall than 99,15% for a MADBase handwritten database. Figure 2.1 shows the CNN model architecture the researchers used in their work. Figure 2.2 shows the results obtained for each digit.

Figure 2.2: The confusion matrix of the CNN model made by [5].

Chapter 3

Methodology

In this chapter, we will present the methods of this research project and describe the main stages to collect data, build the neural network, and validate the results and accuracy of the proposed application.

3.1 Tools and Libraries

Python [13] will be the main programming language used in this project due to the rich libraries and tools in its ecosystem. There are several libraries that are needed to accomplish the objectives of this research. We will be using the Keras [3] library to build the model. Keras is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. We will use Matplotlib [8] and SeaBorn library [18] for data visualization. Other libraries such as Numpy [17] and Pandas [12] will be used. The standard Pyhton library also has many useful modules that will allow us to perform the necessary steps for this project such as importing and analyzing data. To measure the accuracy of the model, we will follow the standard workflow for most classification tasks.

The dataset will be split into training and testing sets. The split ratio is 10%. which we shall more broadly address in it next section.

Keras will allow us constructing a Neural Network model, In Keras (Convolutional) Conv2D layers, these parameters are the first arguments passed to the layer: The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer. Each layer has weights that correspond to the layer the follows it. Conv2D (output_depth, (window_height, window_width)). A convolution works by sliding these windows of size 3×3 over the 3D input, feature map, stopping at every possible location, and extracting the 3D patch of surrounding features (shape (window_height, window_width, input_depth)). Each such 3D patch is then transformed (via a tensor product with the same studied weight matrix, called the convolution kernel) into a 1D vector of shape (output_depth,). All of these vectors are then spatially reassembled into a 3D output map of shape (height, width, output_depth). Every spatial location in the output feature map corresponds to the same location in the input feature map.

The Flattens layer: Flattens the input. Does not affect the batch size. If inputs are shaped (batch,) without a channel dimension, then flattening adds an extra channel dimension and output shapes are (batch, 1). after Convolution layers, we would need to 'unstack' all this multidimensional tensor into a very long 1D tensor. we can achieve this using Flatten. In short, which we used to convert a multidimensional tensor into a single 1D tensor.

The Max-Pooling operation. In the convent example, you may have noticed that the size of the feature maps is halved after every MaxPooling2D layer. For instance, before the first MaxPooling2D layers, the feature map is 26×26, but the MaxPooling operation halves it to 13×13. That is the role of max pooling: to aggressively down sample feature maps, much like strided convolutions. MaxPooling consists of extracting windows from the input feature maps and outputting the max value of each channel. It's conceptually

similar to convolution, except that instead of transforming local patches via a learned linear transformation (the convolution kernel), they're transformed via a hardcoded max tensor operation. A big difference from convolution is that MaxPooling is usually done with 2×2 windows and stride 2 Next, we define a pooling layer that takes the max called MaxPooling2D. It is configured with a pool size of 2×2. the feature map is 13×13, but the MaxPooling operation halves it to 6×6. Dropout layer: Regularizing neural networks is an important task to reduce overfitting. Dropout has been a widely-used regularization trick for neural networks.

In Convolutional neural networks CNN, dropout is usually applied to the fully connected layers. Meanwhile, the regularization effect of dropout in the Convolutional layers has not been thoroughly analyzed in the literature. which is indeed proved as a powerful generalization method. So we will explain how it works in short, that dropout in CNN regularizes the networks by adding noise to the output feature maps of each layer, yielding robustness to variations of images. Based on this observation, we propose a stochastic dropout whose drop ratio varies for each iteration. Furthermore, we propose a new regularization method which is inspired by behaviors of image filters. Rather than randomly drop the activation, we selectively drop the activation which have high values across the feature map or across the channels. Experimental results validate the regularization performance of selective max-drop and stochastic dropout is competitive to the dropout or spatial dropout. Dropout is a regularization technique, which aims to reduce the complexity of the model with the goal to prevent overfitting. Using dropout, you randomly deactivate certain units neurons in a layer with a certain probability p from a Bernoulli distribution, but this yet another hyperparameter to be tuned. The effect is that the network becomes less sensitive to the specific weights of neurons. This, in turn, results in a network that is capable of better generalization and is less likely to overfit the training data. therefore

not be the neural network will not be able to rely on particular activation in a given feed-forward pass during training. As a consequence, the neural network will learn different, redundant representations. the network cannot rely on the particular neurons and the combination (or interaction) of these to be present. Another nice side effect is that the training will be faster.

Batch normalization mitigates the effects of varied layer inputs. By normalizing the output of neurons, the activation function will only receive inputs close to zero. This ensures a non-vanishing gradient, solving the second problem. Batch normalization is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem. because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt them selves to a new distribution in every training step.

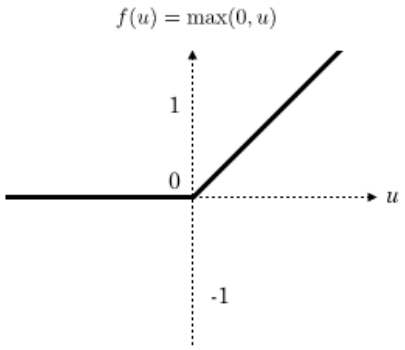

Figure 3.1: A ReLU activation function.

The Dense layer is a standard layer type that works for most cases. In a dense layer, all nodes in the previous layer connect to the nodes in the current layer. We have 128 nodes in each of our input layers. Activation is the activation function for the layer. An activation function allows models to take into account nonlinear relationships. The activation function we will be using is ReLU or Rectified Linear Activation as shown in Figure 3.1. Although it is two linear pieces, it has been proven to work well in neural networks. Because

we are attacking a classification problem, you'll end the network with a single unit The second layer needs a number of classes (a Dense layer of size 10) and a softmax activation function as shown in Figure 3.2. This unit will encode the probability that the network is looking at one class or the other.

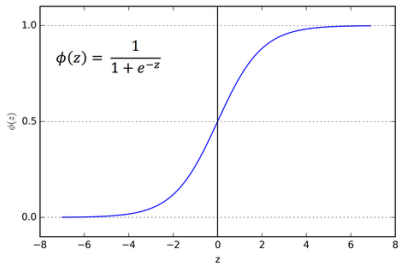

Figure 3.2: A ReLU activation function.

3.2 Dataset

The MADBase is a dataset developed by Sherif Abdelazeem, Ezzat Al-Sherif, for evaluating machine learning models on the handwritten digit classification problem. The dataset was constructed from a number of scanned document dataset available from The Electronics Engineering Department American University in Cairo. This is where the name for the dataset comes from, as the Modified ADBase or MADBase dataset. Each image is a 28 by 28 pixel square (784 pixels total). A standard spit of the dataset is used to evaluate and compare models, where 60,000 images are used to train a model and a separate set of 10,000 images are used to test it on as shown in Figure 3.3.

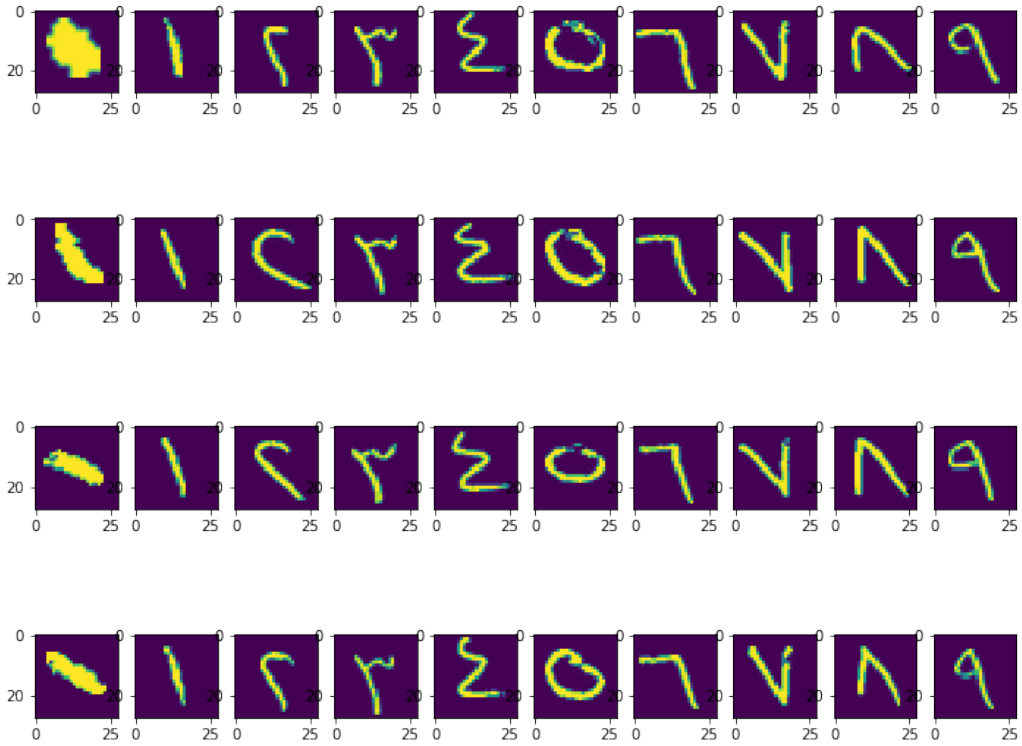

Figure 3.3: Sample of digits in MADBase.

Chapter 4

Implementation and Results

This chapter presents how we implemented the CNN model and how we built the interface and architecture of the application. Also, the chapter will presents the results of the accuracy of our model.

4.1 Graphical User Interface

There are several libraries that are needed to accomplish the objectives of this research. We will be using the Flask [6] library to build the back-end web service. Flask is web framework written in Python that allows the users to develop web application. Javascript [9] will be used in limited situations.

To make the application accessible from different platforms and from different devices, the application will use a web interface. This will make the application cross-platform application and will allow us to make the interface responsive the different screen sizes available.

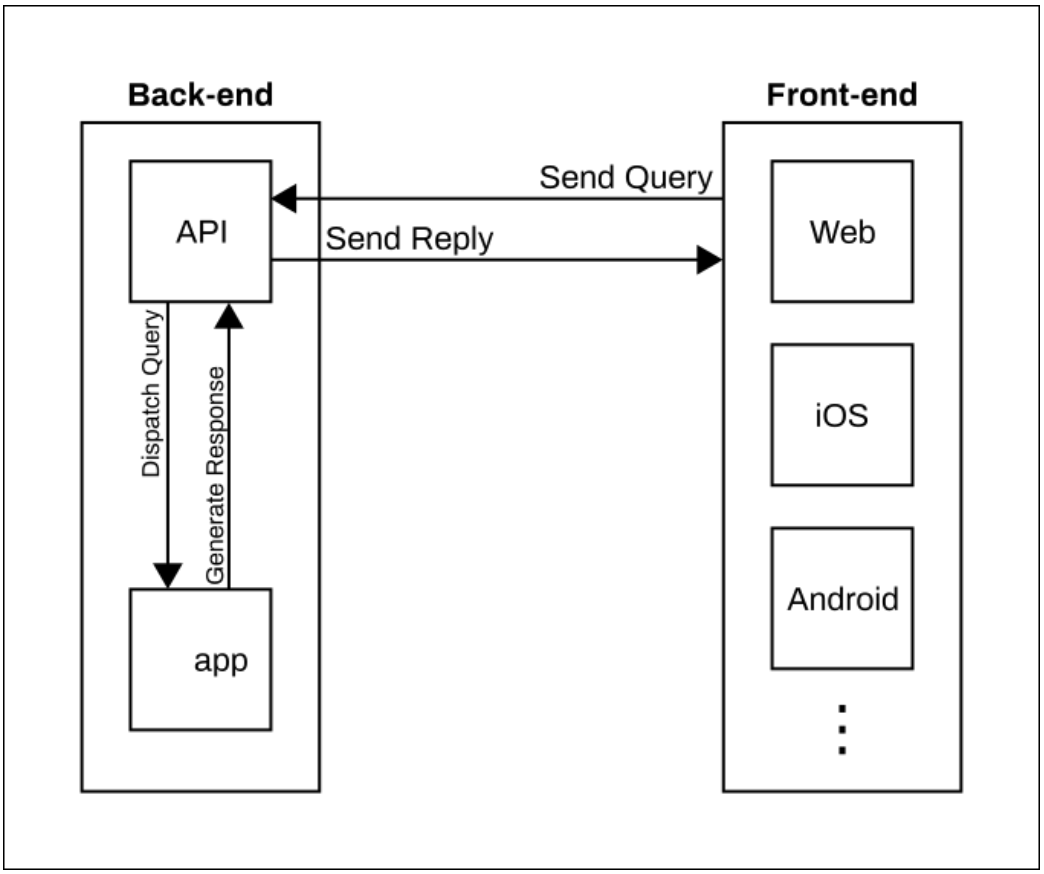

Figure 4.1: The overall architecture of the application.

To make the application accessible for any device and operating system, we will use web interface. Figure 4.1 shows the overall architecture of the application. It is divided into two parts, the front-end and the back-end. The front-end will be the actual web interface as shown in Figure 4.2. This interface currently is developed using HTML, Javascript and CSS. In the future other options are available such as developing native applications for Android and/or iOS. This is possible because we are relying on an Application Programming Interface (API) to perform the necessary functions of the application.

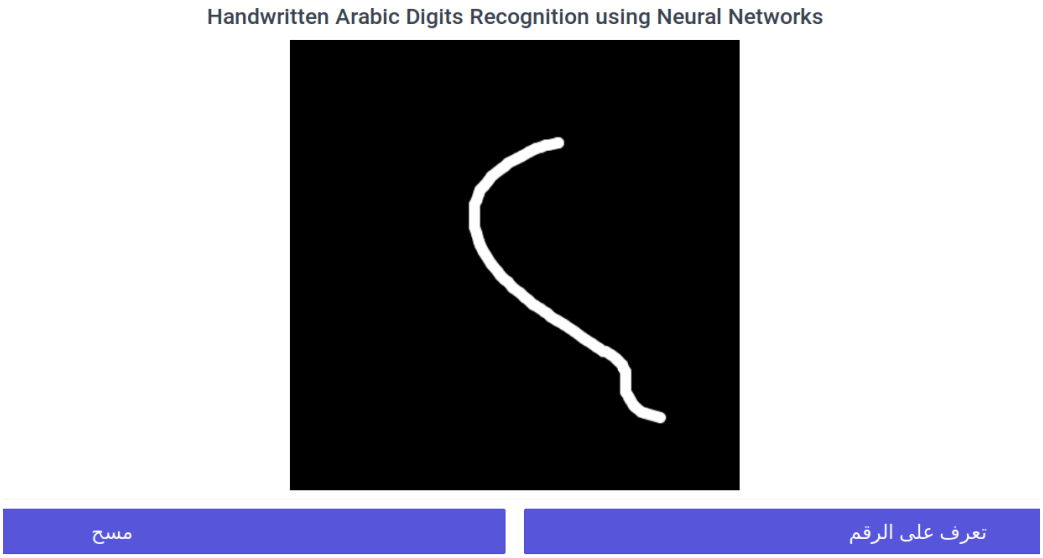

Figure 4.2: The web interface of the application.

Figure 4.2 shows the web interface of the application and how it presents the predictions of the model. The interface features a blank space for the user to enter the digits they want to be recognized and the application will convert the image into the appropriate format for our model. The model will generate several predictions and present them to the user as shown in Figure 4.3.

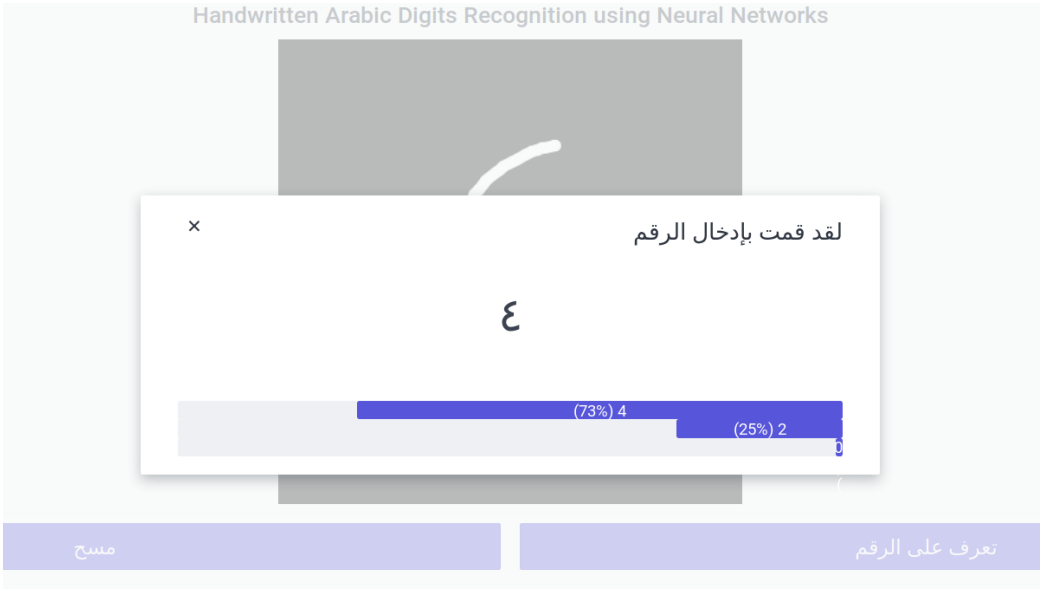

Figure 4.3: The predictions produced by the application.

4.2 Results and Discussion

Figure 4.4 shows the detailed layers of our CNN model and for the full parameters of our model consult Listing 11.

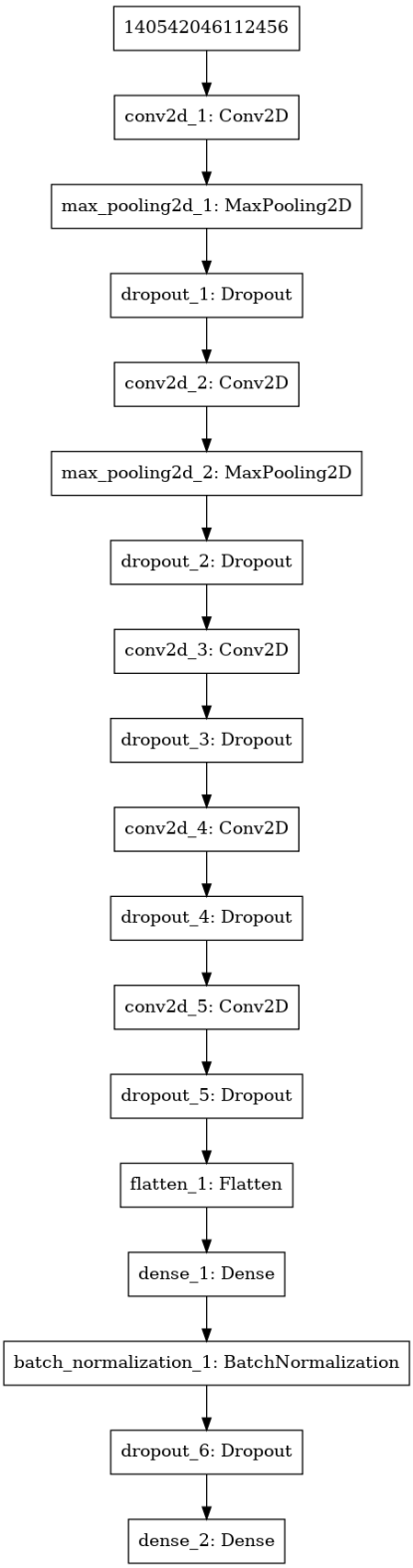

Figure 4.4: The CNN model architecture.

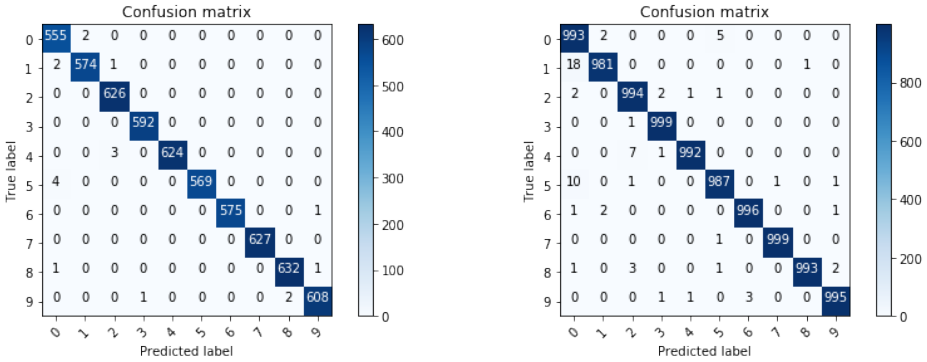

Figure 4.5: The confusion matrices of the model.

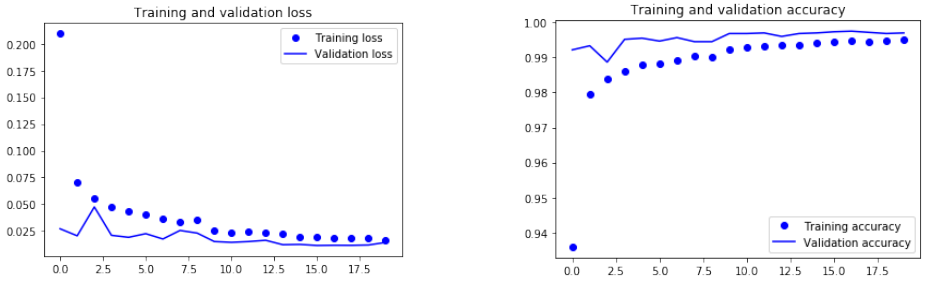

Figure 4.6: The training history accuracy and loss.

Figure 4.5 shows the results of evaluating the performance of our model on MADBase with confusion matrices of the validation and testing sets. Listing 16 and Listing 15 shows the actual code used to produce these figures

We used 20 epochs (as shown in Listing 6) and Figure 4.6 shows the progress of training during these epochs. Our model reached an accuracy of **99.35%** on the testing dataset without using any data augmentation techniques. This is to make the comparison with the work of [5] as fair as possible since they did not use data augmentation. However, with data augmentation our model reached an accuracy of **99.75%**. The settings we used for doing data augmentation is listed in Listing 10.

Chapter 5

Conclusion

Our model was able to achieve competitive scores to other published models on the same dataset. The other models' accuracy scored 99.15% accuracy without data augmentation. Using the same parameters, our model scored 99.35% accuracy without data augmentation. With data augmentation, our model reached 99.75% accuracy. These results gives us confidence in the effectiveness of our CNN model. However, there were some concerns regarding the quality of the data in MADBase itself especially the data for the number zero. Yet, when we fixed all of our model parameters and settings and made them as close as possible to the parameters of the best published model we found [5], our model on average scored 0.20% points higher than it.

For future work, we would like to test our model on other datasets and compare it against other DL architectures and models. As mentioned earlier, there were some concerns regarding MADBase and one possible future work is publishing an improved version of MADBase or creating a whole new dataset to help improve the research efforts in the Arabic handwritten recognition field.

Appendix

This appendix lists all the code used in this project.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools
import seaborn as sns

x_train = pd.read_csv("MADBase/x_train.csv", header=None)
y_train = pd.read_csv("MADBase/y_train.csv", header=None,
 ↪  names=['label'])
x_test = pd.read_csv("MADBase/x_test.csv", header=None)
y_test = pd.read_csv("MADBase/y_test.csv", header=None,
 ↪  names=['label'])

print("x_train shape", x_train.shape)
print("y_train shape", y_train.shape)
print("x_test shape", x_test.shape)
print("y_test shape", y_test.shape)
```

Listing 1: Imported libraries and reading the dataset.

```
x_train.head()
y_train.head()
sns.countplot(y_train['label'])
sns.countplot(y_test['label'])
```

Listing 2: Exploring the dataset.

```
def rot_digit(digit):
    return np.fliplr(np.rot90(digit, axes=(1,0)))
```

Listing 3: A helper function to fix the rotation of the digits.

```
x_train = x_train.astype('float32').values
y_train = y_train.astype('float32').values
x_test  = x_test.astype('float32').values
y_test  = y_test.astype('float32').values

plt.figure(figsize=(12,10))
x, y = 10, 10
for i in range(100):
    plt.subplot(y, x, i+1)
    plt.imshow(rot_digit(x_train[i].reshape((28,28))),
    ↪  interpolation='nearest')
plt.show()
```

Listing 4: Preparing the data and plotting the first 100 digits.

```
x_train = x_train/255.0
x_test = x_test/255.0

print('x_train shape:', x_train.shape)
print(x_train.shape[0], 'train samples')
print(x_test.shape[0], 'test samples')
```

Listing 5: Normalizing the features' values.

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D,
↪ MaxPool2D
from keras.layers.normalization import BatchNormalization
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau
from sklearn.model_selection import train_test_split
batch_size = 32
num_classes = 10
epochs = 20
input_shape = (28, 28, 1)

X_train = x_train.reshape(x_train.shape[0], 28, 28,1)
X_test = x_test.reshape(x_test.shape[0], 28, 28,1)
```

Listing 6: Importing Keras and setting some parameters and reshaping the data.

```
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
X_train, X_val, Y_train, Y_val = train_test_split(X_train,
↪ y_train, test_size = 0.1, random_state=42)
```

Listing 7: Splitting the training data to traning and validation sets.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),activation='relu',
kernel_initializer='lecun_normal', input_shape=input_shape))
model.add(MaxPool2D((2, 2)))
model.add(Dropout(0.20))
model.add(Conv2D(64, (3, 3), activation='relu',
 ↪  padding='same', kernel_initializer='lecun_normal'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu',
 ↪  padding='same', kernel_initializer='lecun_normal'))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu',
 ↪  padding='same', kernel_initializer='lecun_normal'))
model.add(Dropout(0.25))
model.add(Conv2D(128, (3, 3), activation='relu',
 ↪  padding='same', kernel_initializer='lecun_normal'))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax'))

model.compile(loss=keras.losses.categorical_crossentropy,
              optimizer=keras.optimizers.Adam(),
              metrics=['accuracy'])
```

Listing 8: The CNN model.

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=2,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.0001)
```

Listing 9: Reducing the learning rate when there is not meaningful learning.

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=15,
        zoom_range = 0.1,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=False,
        vertical_flip=False)
```

Listing 10: Data augmentaion parameters.

```
model.summary()

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
dropout_1 (Dropout)          (None, 13, 13, 32)        0
conv2d_2 (Conv2D)            (None, 13, 13, 64)        18496
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
dropout_2 (Dropout)          (None, 6, 6, 64)          0
conv2d_3 (Conv2D)            (None, 6, 6, 64)          36928
dropout_3 (Dropout)          (None, 6, 6, 64)          0
conv2d_4 (Conv2D)            (None, 6, 6, 128)         73856
dropout_4 (Dropout)          (None, 6, 6, 128)         0
conv2d_5 (Conv2D)            (None, 6, 6, 128)         147584
dropout_5 (Dropout)          (None, 6, 6, 128)         0
flatten_1 (Flatten)          (None, 4608)              0
dense_1 (Dense)              (None, 128)               589952
batch_normalization_1 (Batch (None, 128)               512
dropout_6 (Dropout)          (None, 128)               0
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 868,938
Trainable params: 868,682
Non-trainable params: 256
```

Listing 11: The model parameters.

```
history = model.fit(X_train, Y_train,
                    batch_size=batch_size,
                    epochs=epochs, verbose=1,
                    ↪  validation_data=(X_val, Y_val))
```

Listing 12: Training the data without augmentation.

```
datagen.fit(X_train)
history_aug = model.fit_generator(
datagen.flow(X_train,Y_train, batch_size=batch_size),
             epochs = epochs, validation_data = (X_val,Y_val),
             verbose = 1, steps_per_epoch=X_train.shape[0] //
             ↪  batch_size,
             callbacks=[learning_rate_reduction],)
```

Listing 13: Training the data with augmentation.

```
final_loss, final_acc = model.evaluate(X_val, Y_val,
↪  verbose=0)
print("Final loss: {0:.6f}, final accuracy:
↪  {1:.6f}".format(final_loss, final_acc))

final_loss, final_acc = model.evaluate(X_test, y_test,
↪  verbose=0)
print("Final loss: {0:.6f}, final accuracy:
↪  {1:.6f}".format(final_loss, final_acc))
```

Listing 14: Pinting the results on the validation and test sets.

```
# Predict the values from the validation dataset
Y_pred_val = model.predict(X_val)

# Convert predictions classes to one hot vectors
Y_pred_val_classes = np.argmax(Y_pred_val, axis = 1)

# Convert validation observations to one hot vectors
Y_true_val = np.argmax(Y_val, axis = 1)

# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true_val,
↪  Y_pred_val_classes)

# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

Listing 15: Validation Confusion Matrix.

```
# Predict the values from the validation dataset
Y_pred_test = model.predict(X_test)

# Convert predictions classes to one hot vectors
Y_pred_test_classes = np.argmax(Y_pred_test, axis = 1)

# Convert validation observations to one hot vectors
Y_true_test = np.argmax(y_test, axis = 1)

# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true_test,
↪  Y_pred_test_classes)

# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

Listing 16: Testing Confusion Matrix.

```
h = history_aug
#h = history

print(h.history.keys())
accuracy = h.history['acc']
val_accuracy = h.history['val_acc']
loss = h.history['loss']
val_loss = h.history['val_loss']
epochs = range(len(accuracy))
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation
↪  accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
```

Listing 17: The training history.

References

[1] Sherif Abdleazeem and Ezzat El-Sherif. Arabic handwritten digit recognition. *International Journal of Document Analysis and Recognition (IJDAR)*, 11(3):127–141, 2008.

[2] Jawad H AlKhateeb and Marwan Alseid. Dbn-based learning for arabic handwritten digit recognition using dct features. In *2014 6th international conference on Computer Science and Information Technology (CSIT)*, pages 222–226. IEEE, 2014.

[3] François Chollet et al. Keras. `https://keras.io`, 2015.

[4] Ouafae El Melhaoui, Mohammed Maroc, Mohamed El Hitmy, and Fairouz Lekhal. Arabic numerals recognition based on an improved version of the loci characteristic. 2011.

[5] Ahmed El-Sawy, EL-Bakry Hazem, and Mohamed Loey. Cnn for handwritten arabic digits recognition based on lenet-5. In *International Conference on Advanced Intelligent Systems and Informatics*, pages 566–575. Springer, 2016.

[6] Flask. Flask the microframework for python, 2018. `http://flask.pocoo.org`.

[7] Chollet Francois. Deep learning with python, 2017.

[8] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering*, 9(3):90–95, 2007.

[9] Javascript. Javascript, 2018. `https://www.javascript.com`.

[10] MADBase. Madbase. the arabic handwritten digits databases, by sherif abdelazeem, ezzat el-sherif,. http://datacenter.aucegypt.edu/shazeem/.

[11] Sabri A Mahmoud. Arabic (indian) handwritten digits recognition using gabor-based features. In *2008 International Conference on Innovations in Information Technology*, pages 683–687. IEEE, 2008.

[12] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[13] Python. Python programming language, 2018. `https://www.python.org`.

[14] Majdi Salameh. Arabic digits recognition using statistical analysis for end/conjunction points and fuzzy logic for pattern recognition techniques. *World of Computer Science & Information Technology Journal*, 4(4), 2014.

[15] P Pandi Selvi and T Meyyappan. Recognition of arabic numerals with grouping and ungrouping using back propagation neural network. In *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 322–327. IEEE, 2013.

[16] Maen Takruri, Rami Al-Hmouz, and Ahmed Al-Hmouz. A three-level classifier: fuzzy c means, support vector machine and unique pixels for arabic handwritten digits. In *2014 World Symposium on Computer Applications & Research (WSCAR)*, pages 1–5. IEEE, 2014.

[17] S. van der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation. *Computing in Science Engineering*, 13(2):22–30, March 2011.

[18] Michael Waskom, Olga Botvinnik, Paul Hobson, John B. Cole, Yaroslav Halchenko, Stephan Hoyer, Alistair Miles, Tom Augspurger, Tal Yarkoni, Tobias Megies, Luis Pedro Coelho, Daniel Wehner, cynddl, Erik Ziegler, diego0020, Yury V. Zaytsev, Travis Hoppe, Skipper Seabold, Phillip Cloud, Miikka Koskinen, Kyle Meyer, Adel Qalieh, and Dan Allan. seaborn: v0.5.0 (november 2014), November 2014.