Arabic Character Recognition OCR

"This report is submitted as partial fulfillment of the requirements of the Computer Arabization course's practical exam"

Team members:

Mohammed Salah Esmaiel El-Ghrably

Mahmoud Essam Othman Eldsoqy

Omar Mohamed Ghazy Khater

Ayman Kamel Abdo Ail

Mohamed Ahmed Ibrahim AbouElsafa

Wasila Mohamed Ahmed

Supervised by:

Dr/ Marwa M. Fekry

Dr/ Ahmed Hamed

Ismailia, Egypt

July 2020

Definition

Project Overview

My project's aim is to program the computer to identify hand-written alphabets via matrix operations. Each alphabet image contains 32*32 pixels, and we create a matrix using these pixels. By multiply the matrix to several sample matrixes, the pixels are converted into a deep neural network. And finally, we Adam method so that the computer can predict the highest possibility of the alphabet written.

Keyword: deep learning, ANN, Feature Extraction, CNN, English, Machine Recognition, natural and physical sciences, image data, image processing

Domain Background

Character recognition is one of the most important research fields of image processing and pattern recognition. Character recognition is generally known as Optical Character Recognition (OCR). OCR is the process of electronic translation of handwritten images or typewritten text into machine editable text. It becomes very difficult if there are lots of paper-based information on companies and offices. Because they want to manage a huge volume of documents and records. Computers can work much faster and more efficiently than human. It is used to perform many of the tasks required for efficient document and content management. But computer knows only alphanumeric characters as ASCII code. So, computer cannot distinguish character or a word from a scanned image. In order to use the computer for document management, it is required to retrieve alphanumeric information from a scanned image. There are so many methods which are currently used for OCR and are based on different languages. The existing method like Artificial Neural Network (ANN) based on Arabic Handwritten character recognition needs the features to be extracted and also the performance level is low. So, a Convolutional Neural Network (CNN) based Arabic handwritten character recognition method is used. It's a deep machine learning method for which it doesn't want to extract the features and also a fast method for character recognition. My personal motivation is that I Have faced a problem, where I wanted to transform a paper

document into a digital one, but I had to type it character by character. I have been trying to find an easy way to solve the problem, simply the purpose of our project is

to recognize hand-written alphabets, so the computer can automatically identify the characters without any manual input.

The link to my data source is: here

Problem Statement

The main objective of this research is to find a new solution for handwritten text recognition of different fonts and styles by improving the design structure of the traditional Artificial Neural Network (ANN). ANNs have been successfully applied to pattern recognition, association and classification, forecast studies, and control applications, to name a few. The recognition results of such text or handwritten materials are then fed into Optical Character Recognition (OCR) as an electronic translation of images of handwritten, typewritten or printed text into machine-editable text. OCR is a field of research that is fully developed and has been quite useful in pattern recognition, Artificial Intelligence and machine Vision. Consequently, typewritten text recognition that is void of any distortions is now considered largely a solved problem. However, the direct use of OCR on handwritten characters remains a very difficult problem to resolve, yielding extremely low reading accuracy. handwritten document recognition is currently a difficult problem as different people have different handwriting styles. Scanning, segmentation and classification are the general processes that are being used to recognize handwritten documents. ANNs have proven to be excellent recognizers of printed characters and handwritten characters.

Metrics

Generating a confusion matrix, for summarizing the performance of a classification algorithm. Classification accuracy alone can be misleading if you have an unequal number of observations in each class or if you have more than two classes in your dataset. Calculating a confusion matrix can give you a better idea of what your classification model is getting right and what types of errors it is making.

It gives us insight not only into the errors being made by a classifier but more importantly the types of errors that are being made.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

	Actual = Yes	Actual = No
Predicted = Yes	TP	FP
Predicted = No	FN	TN

Analysis

Data Exploration

The Alef-yeh Handwritten Data contains capitalized handwritten alphabet images in size of 32x32 pixels. Each alphabet in the image is centered at 20x20 pixel box. There are 16798 images in total, or approximately 598 images for each of the alphabet, in the data file. The dataset contains 2 .csv files with information necessary to make a prediction. The images are taken from NIST (NIST).

```
print("shape:",train_data.shape)
print("shape:",train_label.shape)
print("shape:",test_data.shape)
print("culoms count:",len(test_data.iloc[1]))

train_label.rename(columns={'1':'label'}, inplace=True)
test_label.rename(columns={'1':'label'}, inplace=True)
#y = train_label['label']


train_label
```

```
shape: (13439, 1024)
shape: (13439, 1)
shape: (3359, 1024)
culoms count: 1024
```

	label
0	0
1	0
2	0
3	0
4	0
...	...
13434	27
13435	27
13436	27
13437	27
13438	27

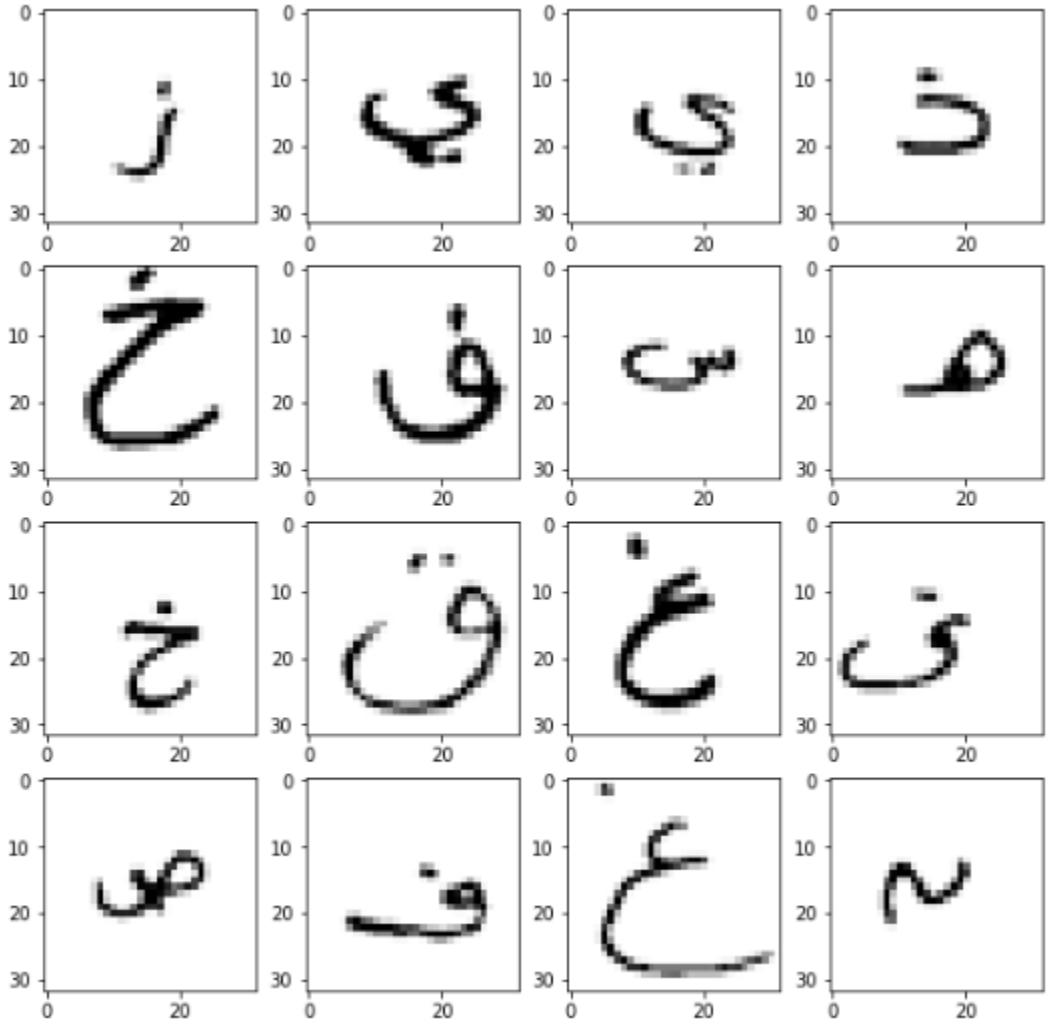
Exploratory Visualization

Writers of the training set and test set are exclusive. Ordering of including writers to test set are randomized to make sure that writers of the test set are not from a single institution (to ensure variability of the test set).

```
from sklearn.utils import shuffle

train_data_shuffle = shuffle(train_data)

plt.figure(figsize = (10,10))
row, colums = 4, 4
for i in range(16):
    plt.subplot(colums,row, i+1)
    plt.imshow(train_data_shuffle.iloc[i].values.reshape(32,32).squeeze().T,interpolation='nearest',cmap='Greys')
plt.show()
```



Algorithms and Techniques

This classifier is a Convolution Neural Network, which is the state-of- art algorithm for most image processing tasks, including classification. It needs a large amount of training data compared to other approaches, fortunately, the dataset is big enough the algorithm outputs an assigned probability for each class.

The following parameters can be tuned to optimize the classifier:

1. Training parameters
    - Training length (number of epochs)
    - Batch size (How many images to look at once during a single training step)
    - Learning rate (how fast to learn, this can be dynamic)
2. Neural network architecture
    - Number of layers
    - Layer type (convolutional, fully connected, or pooling)

Benchmark

As a kaggle data set, my benchmark model showed that the results were promising with a 98% classification accuracy rate on testing images. I plan to work on improving the performance of handwritten character recognition. https://www.kaggle.com/bbloggsbott/understanding-convolutional-neural-network

Methodology

Data Preprocessing

In order to achieve maximum efficiency, we need to preprocess the data before we feed it into the neural network. first, I split the data into training and test data then scale it; it's important to scale the data not to have a dominate variables as we know from the Euclidean distance concept.

Once we got explore the dataset we preprocess it by:

1. Step 1: Import Libraries. First step is usually importing the libraries that will be needed in the program. ...
2. Step 2: Import the Dataset. ...
3. Step 3: Taking care of Missing Data in Dataset. ...
4. Step 4: Encoding categorical data. ...
5. Step 5: Splitting the Dataset into Training set and Test Set. ...
6. Step 6: Feature Scaling.

```
train_data = train_data.reshape(train_data.shape[0], 32, 32, 1).astype('float32')
test_data = test_data.reshape(test_data.shape[0], 32, 32, 1).astype('float32')

train_label = np_utils.to_categorical(train_label)
test_label = np_utils.to_categorical(test_label)
```

In the code above, we first use the shuffle() method to randomize the data (the original one is sorted from Alef-yeh, and it is not machine-friendly). Then, we extract the label, the number indicating which letter is this row representing, from the raw data. keras.utils.to_categorical() is used to convert the number, into the one-hot array.

Implementation

1. Import libraries

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        import os
        from sklearn.preprocessing import MinMaxScaler
        import tensorflow as tf
        from keras.models import Sequential
        from keras.layers import Dense
        from keras.layers import Dropout
        from keras.layers import Flatten
        from keras.layers.convolutional import Conv2D
        from keras.layers.convolutional import MaxPooling2D
        from keras import backend as K
        from keras.utils import np_utils
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import confusion_matrix

        Using TensorFlow backend.
```

2. Importing Dataset

```
train_data = pd.read_csv('./dataset/csvTrainImages 13440x1024.csv').astype('float32')
train_label = pd.read_csv('./dataset/csvTrainLabel 13440x1.csv').astype('int32')-1
test_data = pd.read_csv('./dataset/csvTestImages 3360x1024.csv').astype('float32')
test_label = pd.read_csv('./dataset/csvTestLabel 3360x1.csv').astype('int32')-1


print ("tamam") #don't worry it means okay in my mother tounge
```

```
tamam
```

Well, The Dataset contains handwritten alphabet images (Alef - yeh) in size of 32x32 pixels. Each alphabet in the image is centered at 20x20 pixel

box. There are 16798 images in total, or approximately 598 images for each of the alphabet, in the data file.

3. Data Exploration

- Data Image visualization
- changing labels to alphabets
- I'm filtering the data frame by label frequencies or characters now, using the famous groupby() method .

```
print("shape:",train_data.shape)
print("shape:",train_label.shape)
print("shape:",test_data.shape)
print("culoms count:",len(test_data.iloc[1]))

train_label.rename(columns={'1':'label'}, inplace=True)
test_label.rename(columns={'1':'label'}, inplace=True)
#y = train_label['label']


train_label
```

```
shape: (13439, 1024)
shape: (13439, 1)
shape: (3359, 1024)
culoms count: 1024
```

	label
0	0
1	0
2	0
3	0
4	0
...	...
13434	27
13435	27
13436	27
13437	27
13438	27

```
train_data = pd.read_csv('./dataset/csvTrainImages 13440x1024.csv').astype('float32')
train_label = pd.read_csv('./dataset/csvTrainLabel 13440x1.csv').astype('int32')-1
test_data = pd.read_csv('./dataset/csvTestImages 3360x1024.csv').astype('float32')
test_label = pd.read_csv('./dataset/csvTestLabel 3360x1.csv').astype('int32')-1


print ("tamam") #don't worry it means okay in my mother tounge
```

```
tamam
```

Until we got a conclusion

conclusion from the labels amount

and here we come to a conclusion which is:-

```
print("train alef count:", label_size['alef'])
print("test alef count:", label2_size['alef'])
```

```
train alef count: 479
test alef count: 119
```

each alphabit has 598 sample 479 for the training, and 119 for the testing.

4. Data preprocessing

- Building and Training the model
- First, we build the model. Then, we use the fit() method to train the network.
- Optimizer for the CNN: Adam, with 18 epoch

```
cls = Sequential()
cls.add(Conv2D(32, (5, 5), input_shape=(32, 32, 1), activation='relu'))
cls.add(MaxPooling2D(pool_size=(2, 2)))
cls.add(Dropout(0.3))

cls.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
cls.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='relu'))
cls.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
cls.add(Dropout(0.25))


cls.add(Flatten())
cls.add(Dense(units = 256, activation = 'relu'))
cls.add(Dense(units = 256, activation = "relu"))
cls.add(Dropout(0.5))
cls.add(Dense(28, activation='softmax'))

cls.compile(loss='categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
history = cls.fit(train_data, train_label , validation_data=(test_data, test_label), epochs=18, batch_size=200, verbose=2)
```

```
Train on 13439 samples, validate on 3359 samples
Epoch 1/18
 - 24s - loss: 2.6033 - accuracy: 0.2191 - val_loss: 137.7240 - val_accuracy: 0.5210
Epoch 2/18
 - 24s - loss: 1.4336 - accuracy: 0.5309 - val_loss: 76.5933 - val_accuracy: 0.7333
Epoch 3/18
 - 23s - loss: 0.9046 - accuracy: 0.6890 - val_loss: 53.4358 - val_accuracy: 0.8264
Epoch 4/18
 - 23s - loss: 0.6548 - accuracy: 0.7826 - val_loss: 53.2164 - val_accuracy: 0.8622
Epoch 5/18
 - 23s - loss: 0.5323 - accuracy: 0.8179 - val_loss: 44.8008 - val_accuracy: 0.8890
Epoch 6/18
 - 23s - loss: 0.4472 - accuracy: 0.8495 - val_loss: 39.5074 - val_accuracy: 0.9029
Epoch 7/18
 - 24s - loss: 0.3744 - accuracy: 0.8723 - val_loss: 46.9660 - val_accuracy: 0.8994
Epoch 8/18
 - 23s - loss: 0.3209 - accuracy: 0.8901 - val_loss: 36.8874 - val_accuracy: 0.9250
Epoch 9/18
 - 23s - loss: 0.2786 - accuracy: 0.9034 - val_loss: 40.2008 - val_accuracy: 0.9214
Epoch 10/18
 - 23s - loss: 0.2474 - accuracy: 0.9145 - val_loss: 36.9877 - val_accuracy: 0.9268
Epoch 11/18
 - 23s - loss: 0.2110 - accuracy: 0.9277 - val_loss: 44.7105 - val_accuracy: 0.9250
Epoch 12/18
 - 23s - loss: 0.1998 - accuracy: 0.9311 - val_loss: 43.8246 - val_accuracy: 0.9277
Epoch 13/18
 - 24s - loss: 0.1915 - accuracy: 0.9345 - val_loss: 32.6508 - val_accuracy: 0.9402
Epoch 14/18
 - 23s - loss: 0.1625 - accuracy: 0.9455 - val_loss: 45.6569 - val_accuracy: 0.9244
Epoch 15/18
 - 23s - loss: 0.1478 - accuracy: 0.9485 - val_loss: 39.1671 - val_accuracy: 0.9411
Epoch 16/18
 - 23s - loss: 0.1395 - accuracy: 0.9525 - val_loss: 38.1199 - val_accuracy: 0.9428
Epoch 17/18
 - 23s - loss: 0.1335 - accuracy: 0.9542 - val_loss: 37.1741 - val_accuracy: 0.9384
Epoch 18/18
 - 23s - loss: 0.1239 - accuracy: 0.9568 - val_loss: 38.5297 - val_accuracy: 0.9449
```

Refinement

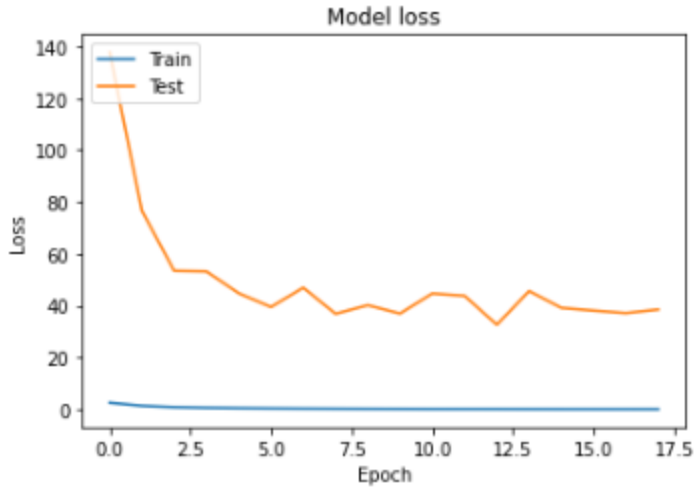
As a kaggle data set, my benchmark model showed that the results were promising with a 94% classification accuracy rate on testing images. I plan to work on improving the performance of handwritten character recognition. but Increasing the number of CNN layers and use convolutional, using max pooling and flatten, using relu and softmax activation function and Optimizing the CNN using Adam, all together did improve the accuracy.

```
scores = cls.evaluate(test_data, test_label, verbose=0)
print("CNN Score:",scores[1])
```

```
CNN Score: 0.9449240565299988
```

booom, we got a 94.4% acc - let's visualize it

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.show()
```



Results

Model Evaluation and Validation

The final architecture is chosen because they performed the best results among the tried combination (the architecture in jupyter notebook)

The complete description of the final model: -

- The kernel size of the first two convolution layers is 5 * 5.
- The first two convolutional layers.
- learned 32 filters and they use a relu activation function.
- The max pool layer has a pool size of 2 * 2.
- The dropout rate is 0.25.
- The kernel size of the third and the fourth convolution layers is 3 * 3.
- The third and the fourth convolution layers learned 64 filters, using a relu activation function.

- The max pool layer has a pool size of 2 * 2, strides = (2, 2) - The dropout rate is 0.3.
- The first two dense layers have 128 units and use a relu activation function.
- The last dense layers have28 units and use a softmax activation function.

This model can detect Latin character in 32*32 photos

- The model generalizes well to unseen data it's predicted the label perfectly.
- The model didn't affect with small changes in the data, or to outliers because of scaling of the data between values 0 to 1.
- We can trust in the model because of it us a high accuracy after fitting the Neural Network.

Justification

After I train the Network, I have an accuracy of 94.4% which is greater than the accuracy presented in the Benchmark model (94%).

When I create the first model I get bad accuracy and try several times with different parameters, so I increased the epochs from 10 t0 18 also I increased the layer to make it deeper so that's enough to have a good model. I changed the optimizer from the descent to Adam to get more high accuracy

This architecture solves the proposed problem

Conclusion

Free-Form Visualization

```
In [1]: from keras.models import load_model
        new_model_2 = load_model('my_model.h5')
        new_model_2.summary()
```

```
Using TensorFlow backend.
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               589952
_________________________________________________________________
dense_2 (Dense)              (None, 26)                3354
=================================================================
Total params: 594,138
Trainable params: 594,138
Non-trainable params: 0
_________________________________________________________________
```

Reflection

The process used for this project can be summarized using the following steps:

1. An initial problem and relevant, public dataset were found
2. The dataset was downloaded and preprocessing
3. The benchmark model was created for the model
4. The classifier was trained using the data (multiple time, until a good set of parameters were found)

Improvement

It will be more useful to classify words or even paragraphs rather than characters (e.g. classify email spams)