Computer Vision: Final Assignment
Optical Character Recognition

Dogan Aykas

29 May 2020

1 Introduction

The chosen topic for this individual assignment is Optical Character Recognition. Frequently referred with its abbreviation OCR, the main idea is to translate a text that is written or printed into a machine encoded text [2]. The main idea of this individual assignment is to create a neural network that is trained to identify a letter or a digit, and translate any input images given to the code into its machine encoded text and return the translations as a txt file.

2 Related Work

OCR is a popular and widely used specialization in the world of artificial intelligence and computer vision. There are numerous amounts of resources that explain introductory deep learning through creating vanilla neural networks from an OCR perspective. The MNIST dataset, which consists of images of handwritten digits and their respective labels, is the most frequently used resource for these applications. OCR has been used in practical fields, including user-friendly applications that convert printed or handwritten files into its machine encoded form, that is readable by any computer. Main challenges in OCR today are mostly related to the accuracy of applications' translations. Historical documents which are physically damaged is the main problem of OCRs currently since their hindered condition also transfer while being transferred via scanning to an electronic environment [1].

3 Methodology

The idea behind the code has been inspired from the GitHub project "Optical-Character-Recognition" [3], which is a basic OCR implementation in python. This implementation uses a dataset consisting of 10,000 train images of uppercase letters and digits and about 3,000 to test them. The predicting power of the model trained in the original code is fairly poor due to the manual implementation of a neural network and the relatively small amount of dataset it uses during its training.

The code that I have re-implemented for this individual assignment starts with using a new dataset, EMNIST, which is frequently referred as extended MNIST. The EMNIST dataset has the original MNIST handwritten digits, plus uppercase and lowercase letters. The dataset includes data consisting of 62 classes, in which 26 of them are the uppercase letters, 26 of them being lowercase letters, and 10 of them being digits.

After the dataset being uploaded to the code and the data being preprocessed, the part corresponding to creating the neural network and training it have been reimplemented. For this, keras library is used. The model's performance is depicted with a confusion matrix after this step.

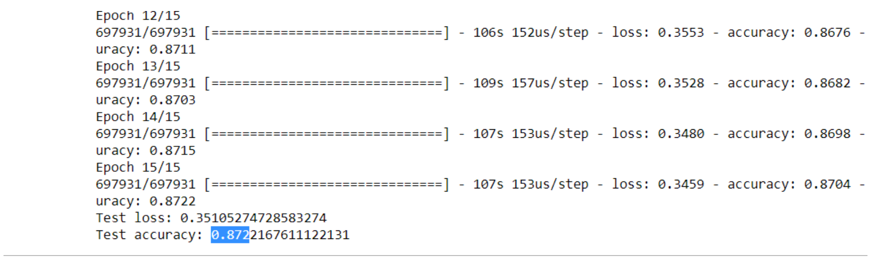
The second part of the code concerns with taking input images (e.g. handwritten scans) and perform OCR on these. In order to do so, the input images are loaded, the letters or digits present in them are individually found using morphology, and then classified with the trained model. The translations of the images are saved as a text file named as "translations.txt".

4 Results

The EMNIST dataset is in csv format, which contains 697,931 training examples and 116,322 test ones. Input images originally have the shape of 28*28, and they consist of their corresponding pixel values. For both the training and test inputs, the source csv file contains the pixel values as flattened arrays of size 784, which correspond to the inputs' original shape that is 28*28. During the reshaping of these EMNIST input files, the images are deformed, meaning that they appear as oriented 90 degrees in the counterclockwise direction and are inverted vertically. The necessary pre-processing steps to convert the to their correct representation are needed in order the input sample images to be classified correctly. For the labels of both training and test data are identified with their ASCII decimal values, converted to categorical arrays that will be appropriate for the CNN model to process.

The CNN model is created using Keras library. It uses the sequential model as its base. This sequential model has two convolutional layers that both have a kernel size of (3,3) and have filter sizes of 32 and 64, respectively. After the second layer, the resulting arrays first applied a dropout layer with a rate of 0.25, and then are flattened. Next, the model is followed by a dense layer, consisting of 128 nodes, a dropout with a rate of 0.5 is applied and the goes through the final dense layer that has 62 nodes, equal to the number of classes in the EMNIST dataset.

Throughout the training of the model, the following hyperparameters gave the best result in terms of the accuracy: learning rate of 0.01, batch size of 128, and 10 epochs. “Relu” is chosen as the approximation function and “Adam” has been chosen for the optimizer. Implementation-wise, the code is run under “Tensorflow-Gpu” for computational convenience. The final test accuracy is 0.872. Below is the results and confusion matrix of the model.



```
Epoch 12/15
697931/697931 [==============================] - 106s 152us/step - loss: 0.3553 - accuracy: 0.8676 -
uracy: 0.8711
Epoch 13/15
697931/697931 [==============================] - 109s 157us/step - loss: 0.3528 - accuracy: 0.8682 -
uracy: 0.8703
Epoch 14/15
697931/697931 [==============================] - 107s 153us/step - loss: 0.3480 - accuracy: 0.8698 -
uracy: 0.8715
Epoch 15/15
697931/697931 [==============================] - 107s 153us/step - loss: 0.3459 - accuracy: 0.8704 -
uracy: 0.8722
Test loss: 0.35105274728583274
Test accuracy: 0.8722167611122131
```

Figure 1: Model results

Looking at the confusion matrix, it is clear that the model struggles to identify some certain types of characters that resemble each other. The most apparent example for that is the characters "O" and "0" and "1” and "l". This is explainable in the sense that the model cannot differentiate word structures whether it is logical to assume a letter over digit, therefore have a hard time differentiating between those.

During the recognition of the sample input images, the sample input images are preprocessed where they are denoised, converted to grayscale. The individual characters are found in from the processed images and are separated and reshaped as individual images in order to be recognized by the CNN model trained before. After the recognition is complete, the code returns the machine encoded translation of the texts fed into the code as inputs. Figure 2 is a representation of a sample image and how the individual characters are detected.

Finally, the characters in the input images are classified with the trained CNN model. After the classification, the labels of the input images are taken and transformed into a readable string format. The code returns a txt file that includes the text representation of the input images in respective order. The characters in the same line are returned as a single string, followed by the strings that are beneath it.

5 Discussion

The translations of the input images are not perfect as it can be seen above. The CNN model has difficulties distinguishing similar characters like "O" and "0". During the character extraction process, words that have unaligned characters (e.g. combinations of uppercase and lowercase letters) cannot be seen as a single line, therefore words not recognized

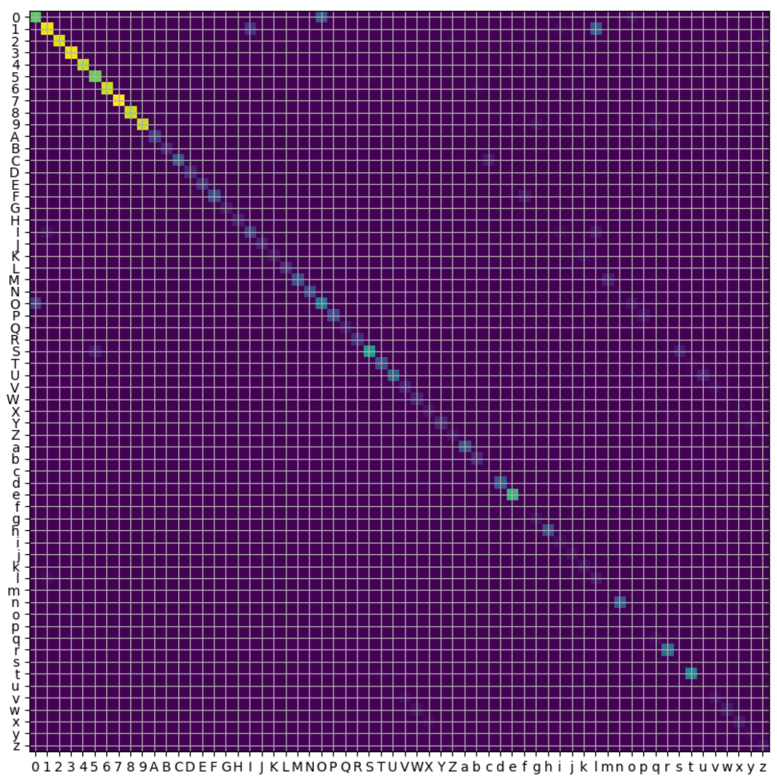


Figure 2: Confusion Matrix

accurately. Furthermore, letter extraction function sometimes recognize false letters which are essentially void, thus can not classifies characters that do not exist, causing the code to output the wrong results. In addition, the model has a hard time extracting characters from low resolution captchas and cannot output the correct machine encoded text formats.

Computation-wise, the training of the model takes a long amount of time that is around 15 minutes, which can take much longer if the code is run under CPU. Furthermore, the code does not have the functionality to detect spaces, which is a problem if the user has the purpose to convert long texts as their original form.

6 Conclusion

OCR is a great topic for implementing the knowledge we learn from our curriculum. It is connected to deep learning as well as computer vision and enabled me to include a variety

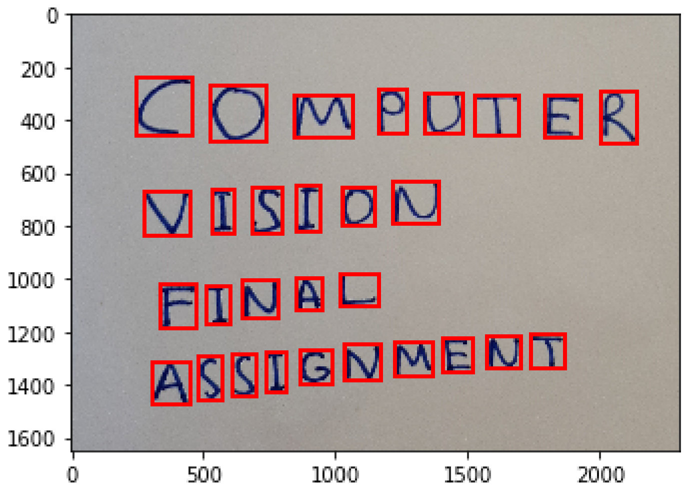
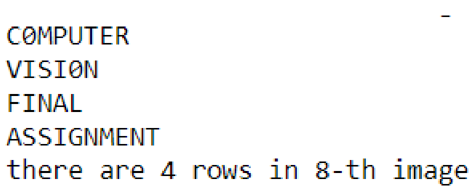


Figure 3: Character Detection



```
C0MPUTER
VISI0N
FINAL
ASSIGNMENT
there are 4 rows in 8-th image
```

Figure 4: Translation of Figure 2

of topics as one. Obviously, big commercial applications like Adobe has very sophisticated and user-friendly implementations for OCR, compared to those, this implementation is trivial. However, the main framework of the code included is essentially same as other implementations. Using more powerful computers for the training of the model will give better results in terms of accuracy, providing the final translations to be better.

References

[1] Mehmet Kaya Karez Abdulwahhab Hamad. A detailed analysis of optical character recognition technology. 2016.

[2] Wikipedia. Optical character recognition, 2020.

[3] @ziliHarvey. Optical-character-recognition, 2018.