Term Project: Handwriting Recognition

Furkan Demir 21802818
Muzaffer Köksal 21803125

I. Introduction

In the information age, availability and rapid access to information are more important than ever. This emphasis on speed and connectivity has also transformed the way we record and process information as well. Where university students once raced to record every bit of information into their leather bound notebooks before the end of class, nowadays recording information is as simple as taking picture with a single click of a button. While digital note taking tools have come a long way in their practicality and ubiquity, pen and paper which have been used for centuries has not yet become obsolete. People still use hand written notes in a wide variety of use cases. Furthermore, texts that predate computers such as historical documents, court filings etc... prove difficult to digitize as they need to be retyped word by word to do so. Over the last 3 decades[1], hand writing recognition technology has been researched to provide solutions to such problems. This project intends to provide an algorithm that will recognize hand writing and other non digitized texts and output it as a text file.

II. Use Cases

Handwriting recognition has a plethora of potential use cases. Some examples are;

-Banking processes where handwritten cheques need to be processed. Despite the ubiquity of credit cards and online banking, cheques are still somewhat popular in bank transactions.

-Indexing and querying of online libraries where thousands if not millions of pages are uploaded by users. For example, websites like CourseHero depend on user submitted documents that are often hand written such as homeworks and assignments.

-Plagiarism detection for non digitized sources and/or submissions. Plagiarism checkers such as Scribbr are often used in academic settings to check whether a body of work submitted by the student is actually his work. This, however does not work for submissions that are written by hand such as assignments that involve extensive mathematical calculations tend to be.

-Healthcare and Pharmaceutical Industries where doctors are infamous for having difficult-to-read handwriting.

-Digitization of old archives that predate computers such as Ottoman records.

III. Related Works

For typed texts OCR technology can nowadays achieve up to 99% accuracy. However, inconsistencies in handwriting styles, writing size, hastiness, print writing vs cursive writing etc... makes it more difficult to achieve such accuracy. Advancements in hand-writing recognition technology have generally been made with neural networks and advancements in such networks are therefore essential for hand writing recognition.

There are multiple companies that provide handwriting recognition services, one of the most notable being Google. Google provides OCR services with its Google Cloud Vision API which is one of the more popular tools in this area. Other vendors include Hanvon Technology and MyScript.

IV. Considerations

-Our handwriting algorithm will only detect Latin alphabet characters that are written in print writing(not cursive writing). In addition, only English words will be recognized from the output text.

-While pre-processing tools such as image dilation-erosion and edge detection will be used, we will assume that the image provided will be of sufficient resolution and the hand-writing will be somewhat readable.

V. Design Choices and Details

The handwriting recognition system will be derived from single character recognition and applied to the custom handwriting input. The designed algorithm can always be improved by doing pre-processing on the input data to make characters more readable by the system. The libraries that are examined in the further sections of the paper will be used for pre-processing, training and testing purposes.

A. Datasets

Two datasets were experimented with in this project. Handwritten A-Z was used originally and a CNN model using that dataset was developed. Handwritten A-Z is an MNIST like dataset that consists of more than 3700000 28x28 images[5] - While the accuracy of the tests was impressive, the Kaggle Nist dataset suffered from a lack of coverage. Handwritten A-Z only had capital letters in its data, meaning our program could not recognize lowercase letters. Therefore we moved away from this database.

Our current model uses EMNIST dataset with another CNN module derived in PyTorch. The EMNIST dataset is a set of handwritten character digits derived from the NIST Special Database 19. The images are converted to a 28x28 pixel image format and the dataset is structured to match MNIST dataset. It has 6 different splits: EMNIST ByClass: 814,255 characters.

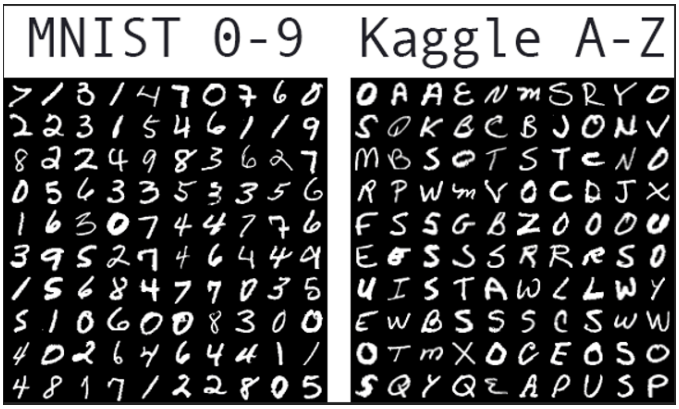

Fig. 1. Mnist and Kaggle datasets for alphanumeric characters.

Fig. 2. EMNIST, extended MNIST dataset for alphanumeric characters.

62 unbalanced classes. EMNIST ByMerge: 814,255 characters. 47 unbalanced classes. EMNIST Balanced: 131,600 characters. 47 balanced classes. EMNIST Letters: 145,600 characters. 26 balanced classes. EMNIST Digits: 280,000 characters. 10 balanced classes. EMNIST MNIST: 70,000 characters. 10 balanced classes[12]. We are currently using EMNIST Balanced dataset.

B. Pre-Processing

For the HANDWRITTEN A-Z project, we first apply a series of pre-processing steps to sharpen and clean the image. The provided image is first converted into a binary image and the closing of binary image is obtained. This step ensures that the gaps that may occur due to inconsistencies in writing are closed and ensures that no additional letters are detected from such gaps(Fig. 5.). After the closing, we obtain use contours to detect each letter and separate them for our model to recognize.

For the EMNIST project, we transform the image to grayscale and resize it into 28x28. Then we convert it into a shape that a PyTorch model can accept, we do ToTensor[9] transformation and feed the data to trained model.

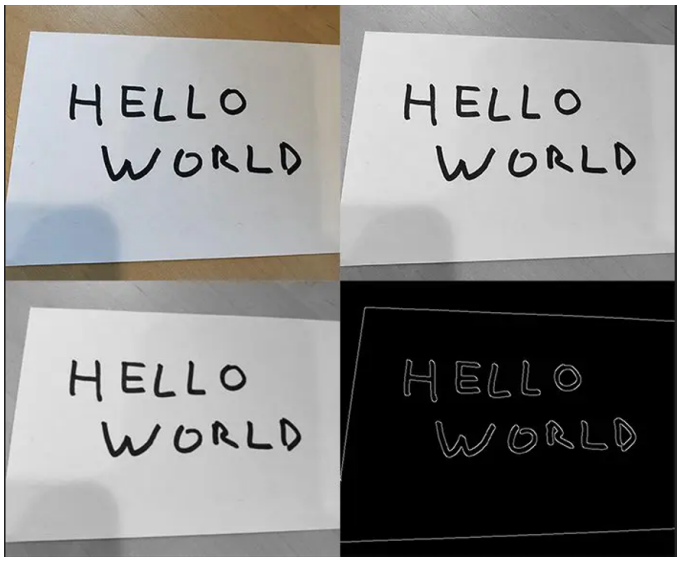

Fig. 3. Pre-Processing the input images.

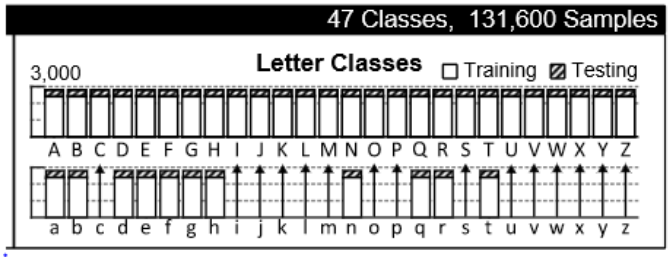

Fig. 4. EMNIST dataset labels.

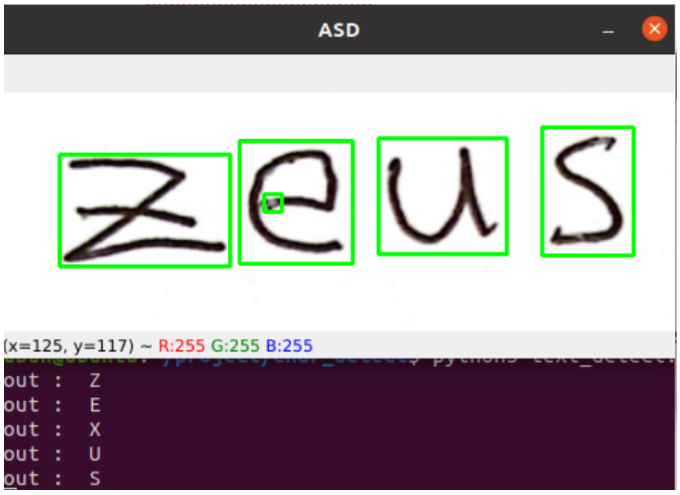

Fig. 5. False result without pre-processing.

C. Algorithm

We have used 2 different CNN models to 2 different datasets. We will examine both of them and compare results in this section. First CNN model worked on Handwritten A-Z dataset and Kaggle Nist dataset.

For the first CNN model, in order to implement the character recognition, first the datasets will be loaded with helper functions from TenserFlow and Keras [6].

For Kaggle A-Z dataset, the program will loop over for each row that has 785 column data, 784 column for x grayscale pixel represantion and 1 column for label [7]. Then, it will parse and cast the rows as a 784-dimensional array that will be used in reconstruction of each image and create an array of images and labels seperately.

For Mnist dataset, Keras has its own helper function for preparing the data. Again, two arrays that corresponds to images and labels will be gathered with the help of the helper function. Then the two image and two label arrays will be combined.

Later, for training purposes; Keras and TenserFlow are used [8]. We took models package from Keras and create our CNN model.

Firstly, a Sequantial model is created and used with the following Keras layers:

- Conv2D layer with parameters:
  - filters: 32
  - activation layer relu
  - kernel size: 3x3
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Conv2D layer with parameters:
  - filters: 64
  - activation layer relu
  - kernel size: 3x3
  - padding: same
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Conv2D layer with parameters:
  - filters: 128
  - activation layer relu
  - kernel size: 3x3
  - padding: valid
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Dense layer for activation Relu

Additionally, the datasets will be divided into 80-20 percent that is 80% will be the training set and 20% will be the test set. Furthermore, several pre-processing techniques such as dilation, erosion, de-skewing, etc. may be used to improve the results of the algorithm. Input images can be seperated to foreground and background to capture the alphanumeric letters more accurately.

Second CNN model which was developed by PyTorch. We first import the EMNIST dataset via datasets package of torchvision [10] and create our datasets. Then, load these data to dataloaders for training, validation and testing purposes. Split of training, validation and test sets are 70-10-20 respectively.

Then, we create our model with PyTorch layers in the following order:

- Conv2D layer with parameters:
  - in_channels: 1 (since there is no color to work)
  - out_channels: 64
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 64 input
- Relu layer
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Conv2D layer with parameters:
  - in_channels: 64 (since there is no color to work)
  - out_channels: 128
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 128 input
- Relu layer
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Dropout layer with 0.3 input
- Conv2D layer with parameters:
  - in_channels: 128 (since there is no color to work)
  - out_channels: 256
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 256 input
- Relu layer

- Conv2D layer with parameters:
  - in_channels: 256 (since there is no color to work)
  - out_channels: 256
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 256 input
- Relu layer
- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2
- Conv2D layer with parameters:
  - in_channels: 256 (since there is no color to work)
  - out_channels: 512
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 512 input
- Relu layer
- Conv2D layer with parameters:
  - in_channels: 512 (since there is no color to work)
  - out_channels: 512
  - kernel size: 3x3
  - stride: 1
  - padding: 1
- BatchNorm2d with 512 input
- Relu layer

- MaxPool layer with parameters:
  - kernel size: 2x2
  - stride: 2

Linear Layers

- Linear Layer with 512 to 256 dimensions
- Relu Layer
- Linear Layer with 256 to 47 dimensions(since we have 47 classes)

This model is derived from various different tries and finally decided on. It takes a bit time to train but it results in very promisin result which we will discuss in further parts.

We have used both SGD and Adam optimizer on the PyTorch CNN model. SGD is proven to be much more usefull, especially in training loss and validation loss values. Thus, we have concluded on SGD optimizer in our second model with learning rate = 0.001 and momentum = 0.9

D. Parameters and Fine Tuning

After the models were ready, we tried them with different hyper parameters to achieve the best result. In the first CNN model which is derived from Keras, these are the hyper parameters:

- batch-size = 1
- epoch = 11
- Early stopping with the following parameters:
  - min_delta = 0
  - patience = 2
  - verbose = 0
- Reduced Learning rate with the following parameters:
  - factor = 0.2
  - patience = 1
  - min_lr = 0.0001

Early stopping is stopping the training when a monitored metric has stopped improving [11]. Reduced Learning Rate is reducing the learning rate when a monitored metric has stopped improving [13]. Also, we have used the Adam optimizer to our model. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments [14].

For the second CNN model we derived from PyTorch, we have used the following hyper parameters:

- batch-size = 256
- epoch = 10
- learning rate = 0.001
- dropout = 0.3
- momentum = 0.9

These parameters were tested in different variety of values to achieve the best result. The epoch number is limited to 10 to prevent over fitting.

E. Results

Our Keras CNN model with Handwritten A-Z provided limited success in handwritten text recognition. The validation accuracy of the model approximated to 73% at the 2nd epoch and increasing the number of epochs did not improve the accuracy of the model by a meaningful margin beyond that point(Fig. 4.). Fig. 5. shows the contours received after the pre-processing steps mentioned above and the result given by the model.

Our Pytorch CNN model that utilizes the EMNIST dataset with Pytorch provided better accuracy than the previous iteration. After 8 epochs, our validation accuracy became 89.49%(Fig. 5.). Increasing the number of epochs further did not the accuracy and at the 10th epoch, it decreased. Therefore the model was not trained beyond that point.

After the testing of the second CNN model with EMNIST dataset, we reported a confusion matrix in (Figure 10.). The confusion matrix includes 47x47 which is equivalent to having a spot for each 47 classes with 47 labels. Also the F1

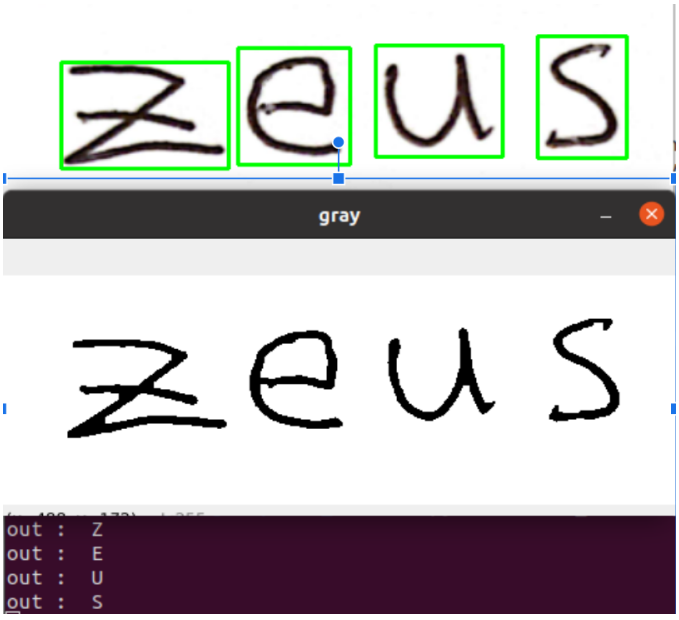

Fig. 6. Accuracy rate of first model.

Fig. 7. Accuracy rate of first model.

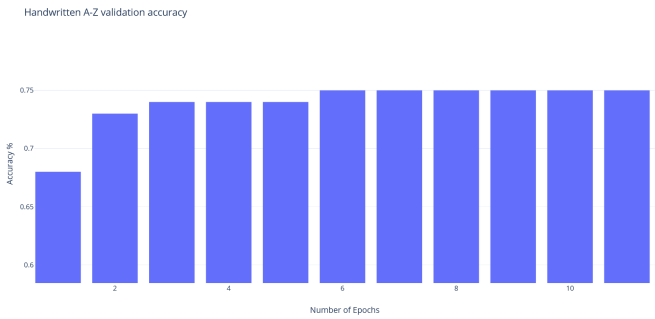

Fig. 8. Accuracy rate of first model.

score of the Pytorch CNN model is: F1 : 0.894

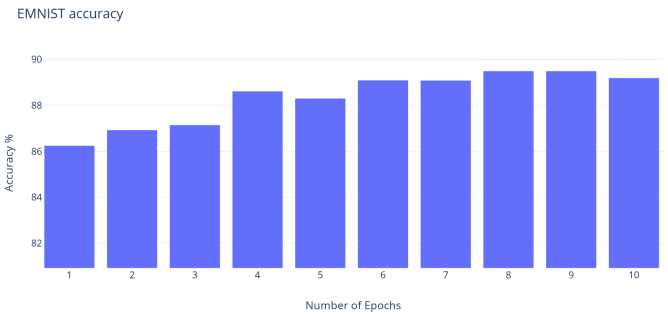

Fig. 9. Accuracy rate of second model.

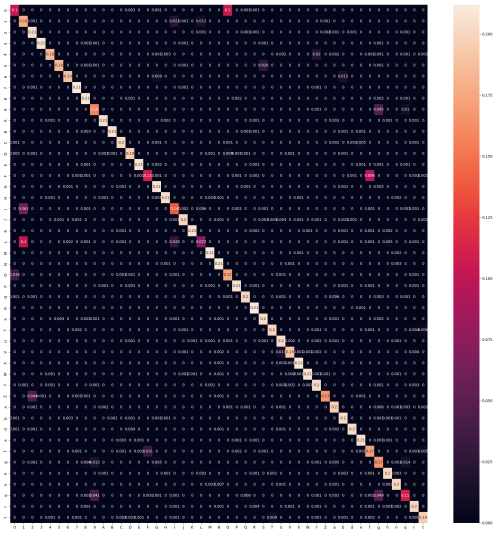

Fig. 10. Confusion Matrix of the PyTorch CNN Model

REFERENCES

[1] *Bortolozzi, F, et al.* Recent advances in handwriting recognition, Paraná/Brazil,Rua Imaculada Conceição, 2005, pp. 1, 2 ,3.

[2] *MNIST database - Wikipedia* [Online]. Available: https://en.wikipedia.org/wiki/MNIST_database. [Accessed: 27-Mar.-2022].

[3] *MNIST handwritten digit database, Yann LeCun, Corinna Cortes* [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 27-Mar.-2022].

[4] *NIST Special Database 19 — NIST* [Online]. Available: https://www.nist.gov/srd/nist-special-database-19. [Accessed: 27-Mar.-2022].

[5] *A-Z Handwritten Alphabets in .csv format — Kaggle* [Online]. Available: https://www.kaggle.com/sachinpatel21/az-handwritten-alphabets-in-csv-format. [Accessed: 27-Mar.-2022].

[6] K. Team, *Keras documentation: About keras,* Keras. [Online]. Available: https://keras.io/about/. [Accessed: 27-Mar-2022].

[7] *OCR with Keras, tensorflow, and Deep Learning,* PyImageSearch, 17-Apr-2021. [Online]. Available: https://pyimagesearch.com/2020/08/17/ocr-with-keras-tensorflow-and-deep-learning/. [Accessed: 27-Mar-2022].

[8] *Fine-tuning resnet with keras, tensorflow, and Deep Learning,* PyImageSearch, 17-Apr-2021. [Online]. Available: https://pyimagesearch.com/2020/04/27/fine-tuning-resnet-with-keras-tensorflow-and-deep-learning/. [Accessed: 27-Mar-2022].

[9] *Totensor¶*, ToTensor - Torchvision 0.12 documentation. [Online]. Available: https://pytorch.org/vision/stable/generated/torchvision.transforms.ToTensor.html. [Accessed: 10-May-2022].

[10] *Datasets¶*, Datasets - Torchvision 0.12 documentation. [Online]. Available: https://pytorch.org/vision/stable/datasets.html. [Accessed: 10-May-2022].

[11] K. Team, *Keras Documentation: Earlystopping*, Keras. [Online]. Available: https://keras.io/api/callbacks/early_stopping/. [Accessed: 10-May-2022].

[12] Patricia.flanagan@nist.gov, 2019. The EMNIST dataset. NIST.

Available at: https://www.nist.gov/itl/products-and-services/emnist-dataset [Accessed May 10, 2022].

[13] K. Team, *Keras Documentation: Reducelronplateau*, Keras. [Online]. Available: https://keras.io/api/callbacks/reduce_lr_on_plateau/. [Accessed: 10-May-2022].

[14] J. Brownlee, *Gentle introduction to the adam optimization algorithm for deep learning*, Machine Learning Mastery, 12-Jan-2021. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/: :text=Adam%20is%20a%20replacement%20optimization,sparse%20gradients%20on%20noisy%20problems. [Accessed: 10-May-2022].