COMP4102 - Project Report

Handwritten Text Recognition

Authors:

Hien Le (101044264)

&

Minh Nguyen (101039776)

Instructor:

Rosa Azami

Table of Contents

Abstract

A Handwritten Text Recognition system converts the text from scanned images into digital text. This is a very challenging problem as it is dealing with nearly infinite variations of handwriting styles. Every person has their own writing style that is very specific and unique. And worse, the problem can even get more complicated as the letters can touch each other, making it incredibly difficult to separate them. The system uses OpenCV to find and separate the letters, then uses a Convolutional Neural Network model to predict their labels, and produces a digital text as a result.

Introduction

Since the English dictionary is huge (approximately 470,00 words), the model cannot train on the dataset of words, but instead, to train on the dataset of 26 alphabetical letters. In order to do that, the system must be able to split the handwritten text into letters by using bounding boxes, or contours. It also needs to have a Machine Learning model that can classify a given image of a letter. A Convolutional Neural Network (CNN) is the most suitable model for such a task because it is more effective in detecting important features without any human supervision compared to its predecessors (e.g. a Feedforward Neural Network). As a result, the general routine to solve this problem is first to separate the scanned image of handwritten text into letters, then feed these separated letters to a trained neural network model and produce a digital text.

Background

The general idea for this problem is based on the paper, Handwriting Recognition by Deep Learning by Anil Matcha [1]. The paper uses deep learning to solve the same problem - recognizing handwritten text. However, the author proposes a very complex deep learning model: a Convolutional Neural Network (CNN) combined with a Recurrent Neural Network called Long Short-Term Memory (LSTM). The CNN model is used to classify the words, and the LSTM is used to combine the letters into a sentence.

In the project, we do not have such resources to train such a complex model (with millions of training parameters) on such a large dataset (approximately 300,000 images), so we consider a similar but simpler approach: to construct a simple machine learning model that recognizes letters (instead of the whole words) from the handwritten text.

Approach

The system uses OpenCV to separate the scanned image of handwritten text into letters. Specifically, the *findContours()* method from OpenCV is used to find the bounding boxes around each letter, and store them. The separated letters are then resized into a size of 28x28, and converted into binary images (black and white).

The dataset for this problem is called EMNIST dataset which contains 145,600 characters of 26 balanced classes. Each image from the dataset is a grayscale image and has the size of 28x28. The dataset is then split into training data and test data with the ratio of 8:2 (meaning 80% for training and 20% for testing).

The CNN model is constructed using Tensorflow and Keras. The input has the size of 28x28x1 (grayscale image with size of 28x28). The model has 8 layers and output a number from 0 to 27 which represents a letter (e.g. 1 = ‘A’, 2 = ‘B’,...). To prevent losing the training weights, the weights (checkpoints) are saved after every training epoch. The model is trained until it converges (the loss does not decrease anymore), then it is tested against the unseen test dataset in order to evaluate its performance.

Putting everything together, the system does the following:

- Train the CNN model, and save the weights (this only needs to be done once).
- Read the scanned image of handwritten text.
- Separate the text into letters.
- Feed each separated letters to the previously trained model
- The outputs from the model are the digital letters which are combined together to produce the digital as a final result.

Result

The CNN model achieves a very high accuracy in recognizing 26 alphabetical characters after 20 epochs. Specifically, it has an accuracy of 93.5% on the training dataset and 92% on the unseen test dataset. These results (accuracies) show that the model is a good fit which means that it has good skill on both the training dataset and the unseen test dataset. However, if we were to train more, the accuracy on the training dataset would eventually increase, but decrease for the test dataset. This problem is called overfitting. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data (e.i. the test dataset) [2].

The system's performance really depends on the input text. For example, it's able to recognize the input text correctly when the handwritten text is well-written and evenly spaced (Fig. 1).

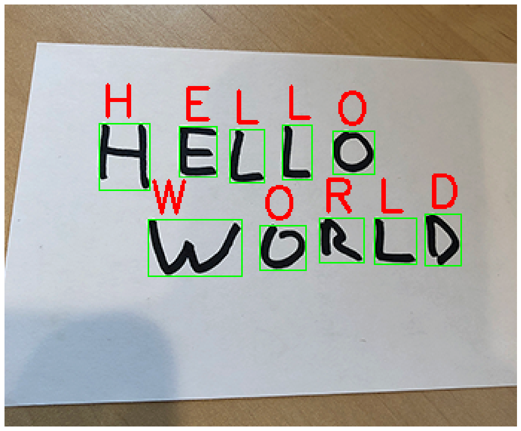

Figure 1. Example of a well-written text

On the other hand, if the handwritten text is not well-written, the system will predict incorrectly. In Figure 2, the system fails to recognize the following:

- The "O" in the word "TODAY" is recognized as a "D"
- The "E" in the word "TEAM" is recognized as a "G".
- "!" is not recognizable because it is not in the training dataset.
- The system incorrectly considers the dot "." on top of the letter "i" as a character.

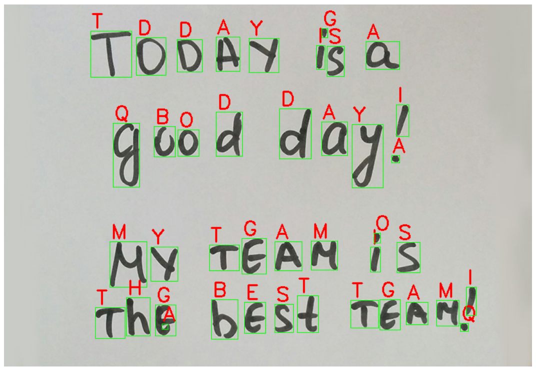

Figure 2. Example of a random text

List of Work

- *Collect the dataset*:

  The dataset must consist of images of the alphabets and numerical digits as well as their labels. Fortunately, there exists such a dataset called EMNIST [3] which contains 145,600 characters of 26 balanced classes.
- *Preprocess the images*:

  Including resizing the images so that they all have the same size 28x28, and converting the pixel to binary (meaning each pixel can either be black or white).
- *Separate text into characters*:

  Using OpenCV to find the contours (or bounding box) of each character. Then preprocess the separated characters like above: resize them to have the size of 28x28, convert them to black and white, and store them into a single array for later use.
- *Construct the neural network model*:

  A CNN (Convolutional Neural Network) model is created using Tensorflow to classify the characters. Since the input and dataset for this problem is quite simple (small size, binary images,...), the CNN model is not required to be complex.
- *Train the model*:

  The model is trained using Tensorflow. The weights (checkpoints) of the model are saved after every epoch for later use. The model converges after 20 epochs and achieves an accuracy of 93.5%.
- *Evaluate the model*:

  After finished training, the model is tested against the test data (20,800 test images) to see how the model works against the data it has never seen before. As a result, the model achieves an accuracy of 92%.
- *Test the whole system*:

  Testing the whole system against real-life input (images of handwritten text). The system is able to split a long text into characters which are recognized by the CNN model.

GitHub Page

https://github.com/quanghienle/handwritten-text-recognition

Reference

[1] Matcha, A. (2021) How to Easily Do Handwriting Recognition Using Deep Learning. https://nanonets.com/blog/handwritten-character-recognition/, Retrieved 15 Apr. 2021.

[3] Cohen, G., Afshar, S., Tapson, J., and Schaik, A. (2017) The EMNIST Dataset. https://www.nist.gov/itl/products-and-services/emnist-dataset, Retrieved 15 Apr. 2021.

[2] Brownlee, J. (2016) Overfitting and Underfitting With Machine Learning Algorithms. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/, Retrieved 15 Apr. 2021.