Handwriting Recognition Using CRNN in PyTorch

Cornelia Genz
Heinrich-Heine-Universität Düsseldorf
cogen100@uni-duesseldorf.de

Emily Bünker
Heinrich-Heine-Universität Düsseldorf
embue100@uni-duesseldorf.de

Abstract

This paper presents a deep learning model, which is trained to recognize handwritten names. We will discuss the individual elements that make up the model and describe how input data needs to be prepared for it. The performance of the model will be demonstrated by deploying it on a dataset of handwritten names. We are then going to evaluate the performance and explore ways of improving it.

1 Introduction

Handwriting recognition is a challenging task. We know this from our own experience when we try to decipher a note that we scribbled down hastily or when we struggle to read a doctor's prescription. If this task is already difficult for a person, would a computer stand a chance? It is definitely a task worth tackling since a lot can be gained from automatic handwriting recognition. It will save a lot of time and labor by replacing manual data entry and can make vast amounts of handwritten data available for further processing and analysis. In the past, many efforts have been made to develop handwriting recognition systems, and nowadays, there are many successful applications of handwriting recognition technologies. For example, it is used by public service institutions or insurance companies to read data from hand-filled forms, by postal services to sort letters, or by historians to analyze documents.

2 Theory and related work

2.1 Context, scope and delimitation of the topic

Two major forms of handwriting recognition can be distinguished: online and offline (Graves and Schmidhuber, 2008). In online handwriting recognition, information on the writing process is available to the system and will be analyzed, i.e. movement of a writing utensil, order and speed of writing. In offline recognition, the system is presented with a finished handwritten text and has no information on the writing process. Another distinction is whether the text to be recognized is written in block letters or in cursive (Kurén and Sundberg, 2023). Some systems are designed to recognize individual letters, others will recognize whole words or even whole lines of text (Dhiaf et al., 2023). Another question to consider is whether the system is to be created for a specific script and a specific language (Graves and Schmidhuber, 2008). Moreover, a system can be trained on input data created by one person only (personalization), or by several people (generalization) (Abdulkader et al, 2006).
In this paper, we will deal with offline recognition of complete hand-printed names written by different people in Latin script.

2.2 Background and related work

As to the implementation of handwriting recognition systems, there are also different approaches. In the past, features were manually extracted, and machine learning methods like Hidden Markov Models (HMM) were used to perform the recognition (Matcha, 2022). Most state-of-the-art systems use neural networks, the majority of which are based on convolutional and recurrent architectures (Barrere, 2018). The most recent approaches make use of transformers (Kurén and Sundberg, 2023; Wick et al., 2021, Dhiaf et al., 2023).

2.3 Convolutional Layer

A widely used architecture in handwriting recognition is the CRNN architecture, a combination of convolutional layers with a recurrent network (Barrere, 2018). Convolutional networks are especially suited for image data, which is treated as a 2D grid of pixels (Goodfellow et. al, 2016).

The purpose of the convolution operation is to extract a sequence of features from each image (Scheidl, 2018, Patel 2020). In the case of handwritten character recognition, those features could be vertical, horizontal and diagonal lines, curves and loops, that together as a pattern, characterize the specific letter. A convolutional layer can create "filters" that move across the image to be classified and identify such features present in the image. Each filter that is used on the image creates a so-called feature map. The individual feature maps are then flattened and joined to represent the image in a format that can be used as input for a classification neural network. Several convolution layers may be added, e.g. to detect higher-level features (features that consist of several of the first-layer features combined).

2.4 CTC Loss

RNNs are commonly used for sequence learning tasks such as handwriting recognition. The problem is that they need pre-segmented training data. This can be solved either by combining RNNs with HMMs or by using the CTC (Connectionist Temporal Classification) Loss. The former approach provides a segmented sequence during the training. However, this hybrid approach does not exploit the full potential of RNNs. With the CTC Loss function, it is possible to label unsegmented sequences (Graves et al., 2006). For this purpose, the images are divided vertically into time steps. The number can be arbitrarily chosen but should at least be as big as the maximum number of letters in the word. The output of the NN contains a matrix which maps each time-step and each character a probability as shown in Figure 1. The symbol "°" represents a space between two characters. Summing up the character-scores, the probability for each time-step should be 1. By multiplying the character scores for each time step, the score of a specific path can be computed. For the given example in Figure 1, a possible path is P=p(°)*p(C)*p(C)*p(C)*p(C)*p(E)*p(E)….

Repeated characters and blanks are removed (Schmidhuber 2009). The probabilities of all paths leading to the ground-truth text are summed up to calculate the loss. To train the NN and generate a high probability, the gradient of the loss is computed with respect to the parameters of the NN. Once the model has been trained, previously unseen input data can be processed by the NN. For decoding, the path with the highest probability is selected, blanks and duplicate letters are removed (Scheidl, 2018).

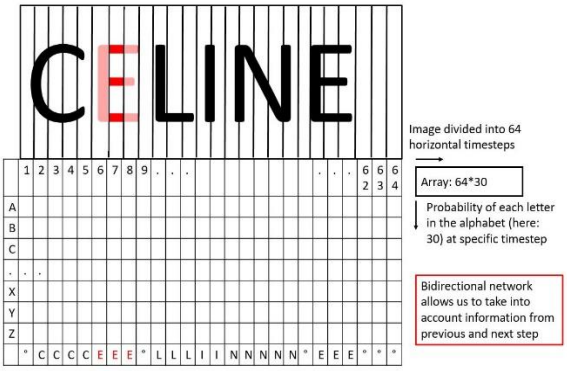
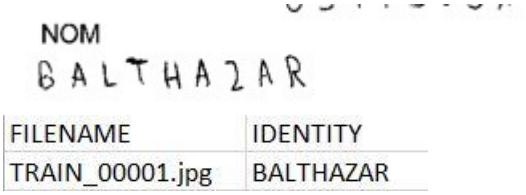


Figure 1: Output from neural network

3 Data and Resources

The dataset used in this project can be found on the kaggle website (Landlord, 2020). There are about 414,000 images in total, each containing a handwritten name and each written by an individual person. They were collected through charity projects. The dataset is already divided into train, validation and test sets. For each of these sets, there is a csv file that associates the image filenames with the corresponding person names (strings). Figure 2 shows an exemplary image and its corresponding label.



FILENAME	IDENTITY
TRAIN_00001.jpg	BALTHAZAR

Figure 2: Image of handwritten name, and corresponding content of CSV file

The preprocessing of the image files included reshaping them to a size of 256 x 64 pixels by cropping or padding them and converting the values for white and black pixels to 0s and 1s.

The code for this project can be found in our Github repository (Genz and Bünker, 2023). The Python version 3.10.12 and the following libraries were used: torch, pandas, numpy, cv2, pillow, pathlib, os, argparse, and torchmetrics.

4 Model architecture and training

In our project, we basically reproduced the neural network from the kaggle notebook "Handwriting Recognition using CRNN in Keras" (Nadar, 2020), but instead of Keras, we implemented it with PyTorch. The model is made up of three convolutional layers, whose purpose is to find features and characteristic patterns in the image data. The convolutional layers are each followed by a normalization layer and the output is activated by an activation function (ReLU). The number of dimensions is then reduced by a pooling layer. Apart from that, pooling reduces overfitting and provides generalization, i.e. it allows the model to still identify features, even if they are slightly distorted, (e.g. differences due to individual handwriting). Dropout layers are introduced to prevent overfitting. Once the data has passed through the three levels of convolution, the output is run through a linear layer and then fed to two bidirectional LSTM layers. These allow to take into account the context of the data being analyzed (what has gone before and what is to come). The output is then sent through a final linear layer to change the number of features to the number of possible characters in the alphabet (plus one for the CTC blank: '°'). Softmax is used as activation function.

The CTC loss function was used to calculate the loss. For the training of the model, i.e. the minimization of the loss, we used the Adam optimizer with a learning rate of 0.0001.
Since creating our very first deep learning model from scratch and processing real-world data with it proved to be enough of a challenge, we contented ourselves with recreating the model from the kaggle notebook and did not try and experiment with different hyperparameters.

5 Results and Discussion

The model was trained with varying numbers of data points and epochs. Subsequently, the performance of the model was manually assessed by generating and looking at predictions for the training set, i.e., data that the model had already seen. At 15,000 data points and 100 epochs, the model yielded only random strings without resemblance to the names, at 50,000 data points and 100 epochs, a couple of names became recognizable. At 150,000 data points and 100 epochs, the model was able to recognize a few names 100% correctly, and many with only minor mistakes.

In order to evaluate the performance of the model, predictions were generated for a selection of data from the test set, i.e. data that the model had not seen before. The results can be found in the file "Evaluation of predictions from test set.xlsx" in the Github repository (Genz and Bünker, 2023).

There are two main metrics for the numerical evaluation of the model's performance. On the one hand, there is classic accuracy, i.e. the percentage of correctly recognized full names. However, in that case, the performance might not be correctly reflected, as names that have been recognized with minor mistakes would not be taken into account. For that reason, a character-based approach is a better option. The commonly used metric is the character error rate (CER). "It is computed as the Levenshtein distance, which is the sum of the character substitutions (S_c), insertions (I_c) and deletions (D_c) that are needed to transform one string into the other, divided by the total number of characters in the ground truth (N_c): (Matcha, 2022):

$$CER = \frac{S_c + I_c + D_c}{N_c} \quad (1)$$

With the most well-trained version of our model, we were able to reach an accuracy of 39% on the test set, and a CER of 0.3096.

5.1 Evaluation

Whether this performance is considered good or bad depends on many factors: the expectations of the user, the purpose of the application, the quality of the used dataset, the performance of comparable models, etc. It is clear that our model makes too many mistakes to be used for automatic data entry, however, it also clearly demonstrates the ability to learn how to recognize handwritten text, which we consider a success.

5.2 Challenges and ideas for improvement

The most basic handwriting recognition systems are trained on the task of recognizing one single letter or digit. This is a classification task with a finite number of classes. In contrast to that, the recognition of handwritten names is a sequence-to-sequence task. It is more challenging because the number of potential ground-truth words is unlimited. The model must find out by itself how many letters a word has and where in the image they are. This increases the probability of making mistakes.

And the words to be recognized are not restricted to a specific lexicon, such as a dictionary of a natural language or a list of possible names. Therefore, such external resources cannot be used to correct mistakes in the recognized text.

The quality of the training data also plays a role. Some images contain noise, others might have been cropped too much in the preprocessing step. Some people used lowercase letters to write their names instead of uppercase or had very untidy handwriting.

In general, shorter names whose letters are written neatly and clearly and do not overlap have a higher chance of being recognized correctly.

Figure 3: Data points with difficulties due to noise and unconventional handwriting

In order to improve the results, the dataset could be cleaned from bad data points. A lexicon of names could be added to correct minor mistakes in recognized names in a post-processing step. Furthermore, the validation set could be used to optimize hyperparameters, such as the number of epochs, the learning rate, the number of time steps, etc. And finally, the model could be trained on more training data (available) with enough computing power (not always available).

References

Abdulkader, Ahmad; Revow, Michael; Haluptzock, Patrick. 2006. "Personalization of an online handwriting recognition system." *Tenth International Workshop on Frontiers in Handwriting Recognition.* (https://www.academia.edu/2022643/Personalization_of_an_online_handwriting_recognition_system , retrieved on September 11, 2023)

Barrere, Killian; Lemaitre, Aurélie; Coüasnon, Bertrand. 2018. *Results of a PyTorch implementation of an Handwritten Text Recognition Framework.* (https://perso.eleves.ens-rennes.fr/people/killian.barrere/, retrieved on September 10, 2023)

Dhiaf, Marwa; Cheikh Rouhou, Ahmed; Kessentini, Yousri; Ben Salem, Sinda. 2023. „MSdocTr-Lite: A lite transformer for full page multi-script handwriting recognition". *Pattern Recognition Letters, Volume 169.* (https://doi.org/10.1016/j.patrec.2023.03.020, retrieved on September 10, 2023)

Genz, Cornelia; Bünker, Emily. 2023. *handwriting-recognition.* (https://github.com/ConnyGenz/handwriting-recognition.git, retrieved on October 2, 2023)

Goodfellow, Ian; Bengio, Yoshua, Courville, Aaron. 2016. *Deep Learning*. MIT Press. (http://www.deeplearningbook.org, retrieved on September 10, 2023)

Graves, Alex; Fernández, Santiago; Gomez, Faustino; Schmidhuber, Jürgen. 2006. "Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks." *ICML '06: Proceedings of the 23rd international conference on Machine learning.* (https://www.cs.toronto.edu/~graves/icml_2006.pdf, retrieved on September 14, 2023)

Graves, Alex and Schmidhuber, Jürgen. 2008. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks." *Advances in Neural Information Processing Systems 21.* (https://papers.nips.cc/paper_files/paper/2008/file/66368270ffd51418ec58bd793f2d9b1b-Paper.pdf, retrieved on September 10, 2023)

Hannun, Awni. 2017. "Sequence Modeling with CTC". *Distill*. (https://distill.pub/2017/ctc, retrieved on September 14, 2023)

Kurén, Jonathan and Sundberg, Martin. 2023. Handwritten Text Recognition Using a Vision Transformer. (https://uu.diva-portal.org/smash/get/diva2:1737799/FULLTEXT01.pdf, retrieved on September 10, 2023)

Landlord (kaggle user). 2020. *Handwriting Recognition. Transcriptions of 400,000 handwritten names* (kaggle dataset). (https://www.kaggle.com/datasets/landlord/handwriting-recognition, retrieved on September 11, 2023)

Nadar, Jebastin. 2020. *Handwriting Recognition using CRNN in Keras*. (https://www.kaggle.com/code/samfc10/handwriting-recognition-using-crnn-in-keras, retrieved on September 11, 2023)

Matcha, Anil Chandra Naidu. 2022. "How to easily do Handwriting Recognition using Machine Learning." *Nanonets Blog*. (https://nanonets.com/blog/handwritten-character-recognition/, retrieved on September 11, 2023)

Patel, Dhaval. 2020. Simple explanation of convolutional neural network | Deep Learning Tutorial 23 (Tensorflow & Python) (video). (https://www.youtube.com/watch?v=zfiSAzpy9NM, retrieved on September 14, 2023)

Scheidl, Harald. 2018. "An Intuitive Explanation of Connectionist Temporal Classification". *Towards Data Science*. (https://towardsdatascience.com/intuitively-understanding-connectionist-temporal-classification-3797e43a86c, retrieved on September 14, 2023)

Wick, C.; Zöllner, J.; Grüning, T. 2021. „Transformer for Handwritten Text Recognition Using Bidirectional Post-decoding". *Document Analysis and Recognition – ICDAR 2021*. Springer. (https://doi.org/10.1007/978-3-030-86334-0_8, retrieved on September, 10, 2023)