Universität Zürich[UZH]

Bachelor thesis
for obtaining the academic degree
Bachelor of Arts
from the Philosophical Faculty of the University of Zurich

Improving Optical Character Recognition Output by Using a Bidirectional Long Short-Term Memory Model

Author: Gian Radler
Immatriculation-nr.: 19-747-708

Referent: PD Dr. Gerold Schneider

Institute of Computational Linguistics

Date of submission: 01.06.2022

Abstract

This thesis deals with the different types of models used in contemporary OCR post-processes, what the difficulties within these processes are and strives to overcome these problems. Specifically, with the help of a Bidirectional-LSTM model using the data set of the ICDAR-2017 competition. The results from the model are then directly comparable to models of other competitors of the aforementioned competition. This thesis tries to establish that this type of model is able to produce significant improvements to OCR post-processes by comparing the results to past competitors. It will also cover contemporary constraints on such models as well as attempt to find possible solutions for these problems.

Zusammenfassung

Diese Bachelorarbeit befasst sich mit den verschiedenen Arten von Modellen, die in modernen OCR post-processes angewendet werden, beleuchtet die Schwierigkeiten innerhalb dieser Prozesse und versucht, diese Problematiken zu überwinden. Konkret wird dies mit Hilfe eines Bidirectional-LSTM-Modells unter Verwendung des Datensatzes des ICDAR-2017-Wettbewerbs angegangen. Die Ergebnisse aus dem Modell sind dann direkt vergleichbar mit Modellen anderer Wettbewerber des oben genannten Wettbewerbs. Diese Arbeit versucht nachzuweisen, dass diese Art von Modell in der Lage ist, signifikante Verbesserungen der OCR-Nachbearbeitung zu erzielen, indem die Ergebnisse mit früheren Wettbewerbern verglichen werden. Es wird auch aktuelle Beschränkungen für solche Modelle behandeln und versuchen, mögliche Lösungen für diese Probleme zu finden.

Acknowledgement

I want to thank my sister Jill and a couple of other people such as Dominic Mailänder, Kaja Walter and especially León von Fournier for proof-reading the present text and giving constructive feedback. It is greatly appreciated.

Contents

List of Figures

List of Tables

List of Acronyms

ACL	Association for Computational Linguistics
AI	Artificial Intelligence
BNC	British National Corpus
BnF	National Library of France
CNN	Convolutional Neural Network
DL	Deep Learning
GPU	Graphical Processing Unit
LSTM	Long-short Term Memory
ML	Machine Learning
MT	Machine Translation
NLP	Natural Language Processing
NN	Neural Network
OCR	Optical Character Recognition
RNN	Recurrent Neural Network
UTF-8	Unicode Transformation Format (8-bit)

1 Introduction

1.1 Motivation

The subject of OCR, which is an acronym for Optical Character Recognition (hereafter every acronym that is relevant for this thesis can be found in list of acronyms one page prior to this), has fascinated me since the start of my studies back in 2019. There are multiple reasons I opted for this topic. Firstly, my personal interest in this subject was piqued during one of the introductory classes to computational linguistics. Up until the end of last year, I only learnt about OCR in theory and what it could do, but did not have the opportunity to work on something where OCR was applied. Last December, I had a window of opportunity to start working on post processes of OCR as part of a project from the University of Zurich. These aforementioned post processes were error detection and error correction, which will be the main focus of this thesis as well and will be focused on in chapter 2.2.

Secondly, OCR is well-developed but not yet perfect. There is a lot of room for improvement not only in the OCR-process itself, but also in the post-OCR processes such as automatic error correction via statistical or neural approaches. Over the course of the previously mentioned project, I adapted the Peter Norvig approach to this problem, which is of purely statistical nature. The results were good but could certainly be improved upon. So for this thesis I decided to take it one step further; as opposed to the previous attempt which relied on statistics, I wanted to build a system that was smarter and could attain even better results. For the purpose of achieving said goal a more profound approach was needed; one that involves neural networks.

Thirdly, I believe society as a whole could profit a lot if there was such a thing as perfect OCR or at least perfect post-processing. The handwriting of each and every person could be scanned and converted to text automatically with no use of a human annotator and texts could also be digitalized without any difficulties.

1.2 Prerequisites

In order for everybody who reads this thesis to understand what exactly is being worked on and fully wrap their head around it, it is necessary to lay out the appropriate foundation in regards to background knowledge as well as terminology, which is flung around everywhere. An understanding of the basic terminology is needed and thus the most important terms and concepts will be introduced: OCR, neural networks, more specifically RNNs & LSTMs and subsequently the current state of research in this area by briefly touching upon some literature, which will be elaborated on in chapter 2.2.

As mentioned before, OCR stands for Optical Character Recognition and belongs to the field of Computational Linguistics. OCR deals with the recognition of written text as the name implies, be that hand-written text or machine-written text. It is important to note that applying OCR on a text only makes sense if the input text is not digital-born, in other words, does not stem from a digital source. Because in that case a transcript for the input text should already be available on some machine with a very low error rate if at all, as OCR-ed texts tend to have a large error rate in comparison to the original no matter how good the OCR system is. As of yet, there exists no such thing as a perfect OCR system. This is due to a multitude of factors such as out-of-date-spelling, especially challenging hand-writing etc.

Neural Networks are the latest breakthrough in computer science as these NNs are able to improve themselves given the correct architecture and enough training material for a specific task. For the sake of brevity it is assumed that everyone has heard of the terms artificial intelligence (AI) or Deep Learning (DL) once before. AI is one of the most prominent results of what was made possible with the creation of NNs. NN is a generic term that covers a whole variety of different sub types ranging from Convolutional Neural Networks to RNNs or even LSTMs which all employ certain kinds of algorithms and structures so that they mirror the functions of neurons in our brain, thus the name *neural* network. The most important type of NN for this thesis is the Long short-term memory or in short LSTM. A LSTM is a kind of Recurrent Neural Network, which means that opposed to, for instance, a standard feed-forward neural network it is able process not only a single input data at a time but whole sequences of data such as whole sentences word by word or character by character by relaying the information of the previous data points to the subsequent data points. I will elaborate on the structure of a typical LSTM-cell in chapter 3.2.3

1.3 Research Questions

The research questions that shall be answered in this thesis, are:

1. Can the output of an OCR system be further improved upon without building a new OCR system? If so: Is the output significantly better?
2. Is this achievable by creating a neural network model that deals with post-OCR processes such as error correction? In this specific instance with a bidirectional-LSTM?

Concrete approach: I will write a script that concerns itself with the aforementioned questions and thus with the post-OCR processes error detection and correction via a specific type of NN called LSTM. A model will be constructed with the LSTM as base and answer the questions in relation to said LSTM model and respective results.

1.4 Thesis Structure

In this first introductory chapter, the motivation for taking this topic as the basis of this thesis as well as the necessary foundation, which is required to fully understand the subsequent chapters, were laid out. The double-layered research question, which is the core aspect of this thesis, is introduced in this chapter as well and further expanded upon in later chapters.

Chapter 2 introduces the status quo of post-OCR processes in combination with the literature read for this thesis.

Chapter 3 deals with the methodology and elaborates on the structure of the LSTM cell.

Chapter 4 reports the results of the LSTM model and lays out the evaluation process as well as highlights its difficulties.

Chapter 5 discusses the aforementioned results and specifically compares it with results from the other papers.

Chapter 6 concludes this whole paper and ties everything together in one last condensed text with the experiences learnt during the writing of this thesis.

2 Literature Review

2.1 State of the Art

OCR has been on a constant rise during the last 30 years. This led to a constant improvement, but still, as of now, there is no perfectly functioning OCR system that has no problem with any possible type of text. On the one hand, you have historical texts, which may have decayed to certain extent, making it rather hard to properly recognize each character that appears within. On the other hand, you have all kinds of fonts or hand-writings that make it also nigh impossible for humans to properly recognize certain words or characters. To make it easier for the system to get a reasonable result, OCR referred to in this thesis is commonly split into two parts: The (building of a) technical OCR-system (leaving out most of the technical part which deals with building such a system for this section as one could write a whole paper about this topic) and the OCR post-correction.

However, a rough understanding of the OCR system itself is needed in order to assess the current state of the art more accurately. It will also further enhance our understanding of why these OCR post-processes carry such importance in improving the overall end product of OCR-ed texts. As the main focus for this thesis is not the OCR system itself, only one paper from the bibliography has dealt extensively with the building of such a system, namely Sahu and Sukhwani. The others have only very briefly touched upon this topic if they even did so at all. Thus the following information will be mostly taken from Sahu and Sukhwani. In recent years, OCR has undergone rather dramatic changes, because of a multitude of reasons. The most note-worthy of which are the availability influx in GPUs in addition to DL-based algorithms that are used for pattern recognition. In their project, Sahu and Sukhwani opted to go with a convolutional neural network (CNN) combined with a special type of LSTM called "LSTM with peephole connections"[1] for the building of the system. The two most common types of NNs used for building a OCR-engine are exactly these two previously suggested ones: The CNN and the RNN, which is

[1] Sahu and Sukhwani [2015]

the class of NN the LSTM belongs to. As said by the authors, their system would be counted as one of the top systems within the field. Sahu and Sukhwani [2015] have stated in their results section that "the accuracy of the proposed model is 75.87% which is the highest in this research area"[2]. There are a plethora of OCR-engines available, such as ABBYY FineReader, Transkribus and Luratech, but all engines are far from perfect. Considering this fact, it makes the result of their proposed system all the more surprising. After having gained a superficial understanding of the technical part of an OCR-engine, in the following section we will elaborate on the OCR post-processes.

2.2 Further Literature

For this following review section we will go through the papers within the bibliography in a chronological order. These reviews focus mostly on two specific aspects of the post-OCR dealings: Firstly, within the OCR post-process we have the error detection part. This part of OCR concerns itself, as the name suggests, with finding any errors the system might have made during the analysis/recognition procedure of the engine. Secondly, there is the error correction part, which is far more difficult than the error detection part as it requires a rather broad spectrum of techniques to overcome it.

As we briefly touched upon in chapter 1.1, there are essentially two approaches one could go for when building an automated OCR post-correction system. The first possibility would be the statistical approach whilst the second one is the DL-approach. The most prominent statistical approach I have found, and coincidentally also worked with, is Peter Norvigs attempt[3], which is recurring across a multitude of websites. In this project, very strongly summarized, he employed a probabilistic and context-based model that corrects words based on the least amount of changes needed to get a correct word from a wrong one. This was accomplished by changing, adding or subtracting letters within the word to get possible "correction candidates", which were then ranked according to their respective probability of appearance within the corpus. Within my own adaptation and implementation of Norvigs code I managed to reach an astonishing 73% for accuracy and 76% for recall on a German corpus from the 2000s.

In the 2017 ALTA shared task competition[4], these post-processes were the key as-

[2]Sahu and Sukhwani [2015]

[3]paraphrasing of the code he used

[4]Wang [2017]

pects of the competition. At this point in time, OCR post-correction was not a new problem but rather well established. The ALTA shared task of 2017 was one of the first that dealt with OCR in a manner that was not purely statistical but also contained neural network approaches, meaning that ML started to get utilized within the newer systems. However, the 2017 shared task proved to have one rather strong limitation for its participants; the task was only graded on the overall final result of both parts together. The used metric for grading all the participating teams was the f-score, which is calculated by $2 * \frac{accuracy*recall}{accuracy+recall}$. Because these two parts were so significantly different, it would only make sense to split the grading up as well.

This was improved upon in a subsequent shared task competition called the "ICDAR2017 Competition on Post-OCR Text Correction" by Chiron et al.. This competition was held in the same year as the ALTA shared task, because of which one could assume that they would be very similarly structured; however, this is not true. The common denominator of these two competitions is the focus on the two OCR post-processes error detection and correction. But they could not have been carried out more differently: While the data set for the ALTA shared task ultimately originated from a crowd-sourcing attempt [Wang, 2017], the data set for the ICDAR2017-competition was built as part of the AMÉLIOCR project that dealt with OCR post-correction [Chiron et al., 2017]. It not only contains English sentences but also French phrases. Its total amount of characters sums up to 12 million characters equally distributed between French and English sentences. This data set will be of further relevance in chapter 3.1, because I used this data set as well for this thesis as it is easily accessible and well-explained in this paper. The gold standard for this data set was created by various external as well as internal projects from the National Library of France (BnF). Generally speaking, this should lead to a higher quality gold standard than for the crowd-sourcing project. Not only the data set was different but also the tasks, which were split for this competition, with separate grading for each task: The f-score was used for the error detection task while the metric used for the error correction task was "a weighted sum of the Levenshtein distances between the correction candidates and the corresponding token in the Ground Truth"[5].These Levenshtein distances were previously alluded to in the paragraph mentioning Peter Norvig and the changing, adding or subtracting of characters in words. This, combined with a counter for these operations, is exactly Levenshtein distances are. Within the competition were a plethora of submissions ranging from statistical approaches all the way to completely neural attempts. It should be mentioned that the overall highest score for the first task was 73% and for the second task only an improvement of 44%. However, these two scores were

[5][Chiron et al., 2017]

not achieved by the same team. The most commonly used approaches were variants of LSTMs, probabilistic approaches as well as sequence to sequence models.

Next we have the "IDCAR 2019 Competition on Post-OCR Text Correction"[6]. This competition was a direct successor to the 2017 shared task and focused on correcting the deficiencies of its predecessor. It split the two parts of post-OCR processes into two different subsequent tasks, which should be completely separately graded. Meaning that one could not compete in the second task if they did not have a working system for the first task. This made a lot of sense in the grand scheme of things and also helped to focus on each task individually. The competition also improved upon the scheme, which was used to host the previous shared task. The volume of available data was almost twice as big, for a total of 22 million OCR-ed characters, and it consisted of ten different European languages, whereas its predecessor only had two languages. The metrics, by which the results were corrected, were also changed. For instance, in the second task it was now required for each error to have a ranked list of correction candidates. Apart from these few note-worthy changes the general concept of the competition remained the same. There were again a lot of differing systems being developed but only five systems were submitted for the competition in the end. Out of which the team using a pretrained BERT language model was the clear winner. They employed BERT for a character correction which was context-based. They had achieved the best score for every language in both tasks, except for Finnish and French in the second task, by a large margin.

The final paper examined for this thesis is "OCR Error Correction Using BiLSTM" written by Kayabas et al. in 2021. the authors from said paper argue with similar points we have seen in the previous papers. They give a very brief summary on LSTMs as well as its bidirectional counterpart. They reinforce the statement we have seen in preceding paper such as the one by Wang that there are essentially only two types of approaches one could take: Firstly, the dictionary-/statistics-based approach and secondly, the neural approach we have come to know rather well over the course of this thesis. Additionally, they emphasize the common use of LSTMs we have seen elsewhere as well. However, one topic they stress a lot that does not appear in other papers to such an extent, if at all, is hyperparameters. Hyperparameters are parameters that have a direct influence on the learning process of a NN and thus play a very important role in the creation of their model to which I can only wholeheartedly agree. Tuning of hyperparameters is a very influential part of model-creation, it can potentially make or break the model as a whole. This aspect is very time-consuming. Their building of the model itself with the different

[6]Rigaud et al. [2019]

noise ratio levels is very fascinating and informative, because it is of no relevance to this thesis, it will be left out. The last thing to note, however, is that they report their results as an improvement of a significant 97% - 98% depending on which data the model was used on. Nowhere does it state relative to what model or system this significant improvement is or whether the improvement is measured relative to the total potential increase of the OCR-ed input. Most likely it was compared to a standard LSTM approach, but we cannot find out for sure.

3 Methodology

3.1 Dataset

The data, which will be used for training as well as for the evaluation of the system, will be the exact same one that was used in the "IDCAR 2017 Shared Task Competition"[1]. This data set was created within the AMÉLIOCR project steer-headed by the BnF and British Library (BL). This is done for a multitude of reasons: Firstly, because the data was already made available by the organizers, it's convenient to access it and secondly, because there are concrete values I can compare the model with, as there were a multitude of teams with varying scores each in the categories of precision, recall, f-score and similarity based on Levenshtein distances. The data itself does not need to be adjusted on a major level, however, the total volume of data available is about 12 million OCR-ed characters equally split into English and French. For this thesis only the English part of the data set will be used.

Due to degradation of the original input document into the OCR-engine, it is possible that certain sentences and their corresponding gold standard cannot be fully aligned in a reasonable manner. The English texts have an average error rate of 2.4% across the board. The provided data set was split in the usual 20-80 ratio, where 80% of the data is used for training as well as testing purposes, while the rest is to be used for evaluating the model. For my model I used the data from the training part only for training and testing and completely neglected the evaluation data for all but actual evaluation.

The data is arranged into a multitude of files numbering somewhere from 800-900 total. Additionally, the size of each file ranges from 3 or 4 KB to somewhere between 3 - 4 MB. This made it very difficult to train and allocate enough capacity for this process. Thus I truncated the data by either removing the files that were too large or splitting files into a number of smaller files, because if nothing was done, the resulting arrays would be too large by multiple degrees of magnitude. From my experience the optimal file size is ranged from 4 - 10 KB. This small file size allows the model

[1] Chiron et al. [2017]

to better work through its input sentences as each file was specifically designed for containing one input sequence and one target sequence marked with [OCR_aligned] or [GS_aligned] respectively. GS being the acronym for Gold Standard. From all these files containing one input and target sequence, I generated two new files called 'ocr_aligned.txt' and 'gs_aligned.txt' respectively. For the new files I attached to each sequence from the original data set file a '\t' in the beginning of the sequence and appended a '\n' to the end of every sequence. Each of these files contained the content of one of the given data files in a single line. In other words I concatenated every input and target sequence within these two files so that later on I could extract the list of sequences for either the input or target sequences from these files without actually having to loop over all the files again. The script was written in such a way that if these aforementioned files were not available, python would create them anew by looping over all the data set files again. If the files were available that step would simply be skipped.

One last important part for the data set is still left to clarify - misalignment. How does the data set deal with misalignment? How is a token defined in the first place? The answer to the latter is surprisingly simple: everything that is separated by a space is counted as one separate token. That was how it was done in Chiron et al.'s paper so I will do the same. Otherwise it will take up a lot more time especially in combination with the misalignment fillers '#', which display misalignment within the gold standard, and '@', which fulfils the function of a padding symbol within the Ground Truth. The @ symbols were basically negligible for the whole training and testing phase but the '#' posed to be a problem. Because of the rather frequent misalignment within the sequences especially towards the beginning of the strings, they often only consisted of '#' repeated for 20 - 40 times depending on the sequence. These symbols were only in the Ground Truth, which made it very difficult for the program to ignore it and not take it into consideration for the training process.

The attempt with this data set will be more of a quantitative approach rather than a qualitative one, because in order to train a model there is a certain quantity needed for the model to be trained appropriately, as it sadly is not as viable to just use qualitative data. Another factor is that this data set contains so much noise that it could not be labeled as qualitative. With a total of 6 million OCR-ed English characters at our disposal for training and evaluation this should suffice even though the data is not the best. In the end I decided to use 1 million characters of the data set out of memory and running time constraints. Even within the gold standard there are mistakes or inaccuracies, which make it more difficult for the training and testing process. These mistakes can seem insignificant, but they can potentially harm the learning system.

3.2 Scripts

3.2.1 General Information

The overarching goal of this thesis was to correct OCR-ed strings by feeding them into a self-made model and retrieving its output strings. This aforementioned model was created with the coding language Python by employing a specific type of neural network called LSTM, enabling us to build a model capable of learning. This model is able to process the output of the OCR-system character by character. This sequential procedure is standard for post-OCR processes to get corrected strings, the only difference within all these possibilities is the approach itself which can vary from statistical approaches all the way to machine learning ones. As post-OCR problems are usually approached in this manner, it is definitely the method best suited for this attempt as well. Before elaborating on how the model was built exactly, the preparation done on the data set prior to the building of the model is necessary. This following sections of the methodology will be split into two parts: Firstly, the construction of the data including the manual as well as automatic preparation in the form of python scripts will be laid out in Subsection 3.2.2. Secondly, the main script written for accomplishing the goal of correcting input strings will be explained thoroughly in Subsection 3.2.3 especially in regards to the model which will be illustrated.

3.2.2 Data Preparation

The data preparation can be split into two parts: The manual work done on the data and the scripts written to process said data automatically. Because it is more sensible to go through these processes chronologically, the manual aspect of data preparation will be looked at first.

The general structure of the data was already explained in chapter 3.1, thus it will not be reiterated but expanded upon. The available data was split into two equal parts, namely, French and English. In this thesis the French part will be left out completely, instead we will focus on the English monograph part of the data set as opposed to the periodical part. This has one predominant reason: The smallest files contained within the periodical section are approximately 20 KB large, whereas there are over 550 files in the training data set of the monographs in combination with the evaluation files that range from 4 - 10 KB in size, which is a lot more suitable for my needs due to memory and run time constraints. Combined, a total of approximately

1 million characters are contained within these files. All the suitable files within the training and evaluation section of the data set were assembled into one folder for later processing. Because the other directories became redundant, they were deleted.

Subsequently, in an effort to extract all the sentences from the files, an automated process was needed as the extraction of the sentences from over 550 files manually would have been very arduous and time-costly. The script took out sentences according to the start token [OCR_aligned] or [GS_aligned] respectively and condensed the sentences from each file into two separate files according to whether the sentence was an input or target sentence. The sentences were written onto one line each and were stripped of their starting token and enclosed by a '\t' in the beginning and '\n' at the end. These resulting files were then used as direct input for the main script.

3.2.3 Model Structure

As outlined above, the script integrated an LSTM neural network to accommodate the sequential nature of the input data. But prior to processing the input data within the model, it had to be collected and transformed into a format which made it feasible for the model to process its input data. The data was extracted from the two previously created text files called 'gs_aligned.txt' and 'ocr_aligned.txt' that were alluded to in the paragraph above. With these two files as the needed exposition, the preparation of the input data for the model started. First of all, a list of appearing characters was needed for each of the files; this was accomplished by creating a set out of the files. Subsequently, each sentence from the files had to be taken out and these were saved into two variables called 'ocr_sent_list' and 'gs_sent_list', depending on which file the sentences stemmed from. This was done with the help of a regular expression that searched for the previously set start token '\t' enclosing wildcards in the middle and at the last position the sentence-final '\n'.

The following step was the most vital to the whole input data transformation. The optimal input format for this specific model is a one-hot-encoded array. To achieve such formatted input, for starters, a numpy array fitted to the input dimensions and filled with 0s was needed. For this specific instance, the numpy.zeros function was used in combination with the appropriate dimensions, which were the following: the first parameter given to the function was the length of the sentence list, the second was the longest sentence measured by amount of characters and finally the number of characters appearing within these sentences was given as the last parameter. Following this, a character dictionary was set up with each character as key and the

corresponding index it has within the list of all characters as value. In the subsequent step, a loop over the sentences within the list and another loop over each character within the sentence was performed. It used the sentence index, the character index and lastly the index of the particular character the loop was at to set a value of the array to 1.0 instead of 0.0. After the loop had finished, the one-hot-encoded array was finished as well; this process was completed for the input as well as for the output sentences. A reverse-engineered character dictionary with the character index as key and the actual character as value were additionally instantiated for later use.

The arrays created in the previous step were then split into a training and a evaluation part for the input array as well as output array. The training - evaluation split is fixed at 80 - 20 for this data. Additionally to the four newly sliced arrays, two sentence lists containing only the evaluation data sentences were instantiated. Prior to building the actual model two other actions were taken: Firstly, the averaged similarity between the evaluation sentences were calculated as well as the percentage of '#'s within the sentences of the gold standard. The necessary hyperparameters also needed to be set up. After a lot of testing and tinkering the optimal values found for each of the variables were as follows: 256 for the latent dimension size, 18 for the batch size and ultimately 20 for the amount of epochs.

Finally, the building of the model was at hand after all the necessary preparation had been finished. The framework chosen for the model was the tensorflow library containing keras, mostly because I was acquainted with it and the documentation being very thorough and understandable. The model was setup to be sequential meaning that each layer only had one input and output. The first layer to be defined was the input layer which was done in a similar fashion to how it had been done for the one-hot-encoding. The next layer and most important one of the model was the LSTM layer. The structure of a typical LSTM-cell is found in Figure 3.1. The picture was taken from Wikipedia as it is a compact and well-understandable depiction of an LSTM-cell. The LSTM has three inputs at each time step within the sequence. The first input was synonymous with the actual one-hot-encoded array value for

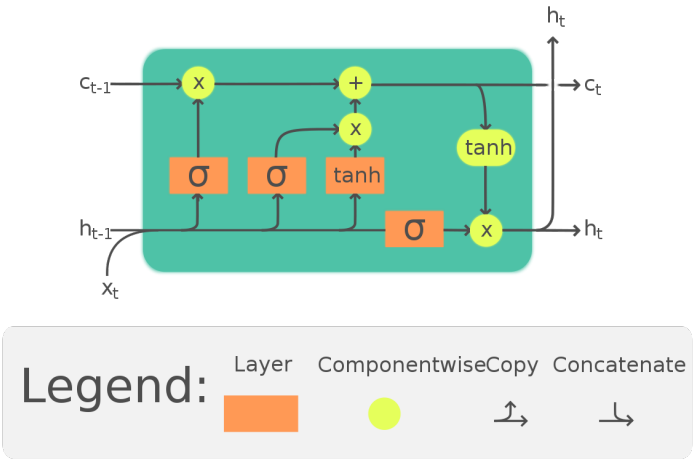

Figure 3.1: LSTM-Cell (Source: Wikipedia)

the character at time step t in our model called x_t. The second input information is the hidden layer information from the previous time step h_{t-1} . The last information given to the LSTM-cell was the accumulated context information form the preceding time steps labeled C_{t-1}. Various calculations and mathematical operations were then performed with these three inputs, for instance, the sigmoid activation function was applied to the concatenation of h_{t-1} and x_t. This was repeated until every calculation shown in Figure 3.1 was performed and two strands of information were left: h_t which functioned as input to the next layer as well as output of the LSTM system for the initially fed character x_t as seen by the arrow on the far right that goes from bottom to top. C_t was the other information strand left and it was fed to the next iteration of the LSTM-cell to act as context information for the next time step.

Additionally to the usual LSTM layer, a bidirectional layer was stacked on top which increased the overall output accuracy by 0.5% (e.g. from 98.75% to 99.22%). This bidirectional layer made the input of the LSTM layer double as there was now the normal string fed into the LSTM as well as the reversed string. This was of high significance, because the LSTM was not limited to the context information C_{t-1} at time step t but also had access to future context information. This increase is large when keeping in mind that the base similarity of the evaluation data is at 96.61%. The final layer within the model was a dense layer that condensed the outputs from the bidirectional-LSTM layer to the amount of target characters and due to the softmax activation function, probabilities will be assigned to each of the characters. To start training the model, the model first had to be compiled with the appropriate parameters which in this case, after a lot testing and trying different things, were: rmsprop for the optimizer, categorical-crossentropy for the loss function and accuracy for the metrics. Afterwards it had to be fitted to the data by feeding it the input and target array. Following the training process only the evaluation was left. This was achieved by a self-written sequence-decoding function which translated the output array back to the English language in combination with a loop that iterated over the evaluation data. The evaluation process will be further clarified in chapter 4.1.

4 Results

4.1 Evaluation

This chapter solely focuses on the evaluation process and reports the results. The discussion of the results follows in Chapter 5. There were multiple modus operandi to choose from for the evaluation process of the model such as the "manual" approach or the fully automated approach. The automatic evaluation of the model with its built-in function evaluate() was very convenient but it seemed to lack precision. This was due to the special structure of the data set with the misalignment fillers '#' and the padding symbols '@'. The built-in function was convenient but it lacked the necessary flexibility for its output, because it was limited to the chosen metric which was previously defined. The most important thing was getting an output string which was corrected and not only a number that is the result of comparing two arrays. Because even if you saw the number it would not tell you how the output string was structured, where exactly it failed, where it excelled or if it even properly worked. With the intention of better accommodating these symbols within the evaluation process, the manual approach was chosen whereas, the automated evaluation was only taken as a reference.

The only possibility for validating the true capabilities of the model was by letting the evaluation part of the system reverse engineer the predicted array into a string. This was done with an evaluation loop and the self-written sequence_decoder function. In a first step, a loop was written to iterate over the evaluation sentence pairs. The target and thus gold standard sentence was in its most basic format: a normal string. The input data, however, was in its array format. The array remained unchanged since it was sliced of the initial combined array. The array of each sentence was fed into sequence_decoder function which in return predicted a target array. This target array was then reverse engineered back into a string by iterating over each sentence and furthermore by iterating over the most probable character indices for each character position in the original string. These most probable characters indices were turned back into strings as a result of extracting its value from the previously reverse-engineered character index dictionary by inserting the most probable

index as key. The resulting string was sent back for comparison with the ground truth. A final loop was written to iterate over each character pair in the decoded sentence and the gold standard. The final valid score was calculated as follows: In case the character from the target sentence was the filler symbol '#' neither of the two characters counted towards the total. If they matched, the total count as well as the correct count increased by one. However, if they did not match, only the total count was incremented by one. The result was then calculated as a percentage by $\text{correct}_{\text{count}}/\text{total}_{\text{count}} * 100$.

4.2 Overview Results

Following the previously described evaluation process and feeding it the evaluation data yielded the following results: It is to be noted, however, that all the results surmised in the subsequent table were trained and evaluated on data from the training and evaluation section of the data set containing English monographs and no periodicals. There would be nothing to gain from including results of the evaluation on English periodicals since the amount of data from the monographs would overshadow the amount of processable data from the periodicals due to mainly memory constraints by a few-fold. The results of the model were averaged over ten separate evaluation runs. The only values which were not averaged were those that remained constant throughout the evaluation runs, due to the fact that they were not influenced by the output of the model. This is perfectly illustrated by the "2.21%" in the bottom row of table 4.1.

	Raw Input and Target Data (constant)	Predicted Input and Target Data (averaged over 10 runs)	Automated Evaluation (averaged over 10 runs)
Percentage of Correctly Matched Characters (including '#')	96.61%	98.75%	99.36%
Percentage of Correctly Matched Characters (excluding '#')	98.79%	99.19%	(99.36%)
Percentage of '#' within Target Data (constant)	2.21%	2.21%	2.21%* assumed value

Table 4.1: Overview of the most important scores from the Bidirectional-LSTM model

However, some immediate clarifications for the table might be needed. In the first column, the three categories, where a score will be calculated for each category, are mentioned. The first category mentioned is the "Percentage of Correctly Matched

Characters (including '#')". As the name implies, in this row the similarity score for each of the sentence pairs will be calculated, hits containing '#' have been included in the calculation of the score. The subsequent row calculates a similarity score as well for each sentence pair, but this time the counts containing a '#' have not been counted towards the overall total. The last category calculates the average percentage of '#'s within the ground truth, thus it is constant. In the top row, the type of evaluation is listed.

The target data remains constant across the whole table, whether it is for comparison with the raw input, the predicted input or the automated evaluation. Raw input in this specific instance means the OCR-ed text before it has been processed in any way besides being extracted from the file. The percentages between these sentences and the target phrases remained constant over all iterations of model-building and evaluating said model, because they are not affected by the model in the slightest. Thus, this column is marked as constant. The similarity for the first column yielded a score of 96.61%, a score of 98.79% for the second category and the constant 2.21% for the last row. The next column is the predicted input column, which is in simplified terms the output of the model. In this column, the predicted string and target sentence are compared. The subsequent values were averaged over ten different evaluation runs. These following scores are the ones of utmost importance for this thesis: For the first category, the model yielded a score of 98.75%, while it resulted in an even higher score for the subsequent row. For this category, the output of the model managed to reach 99.19%. The score of the ultimate row remained the same for the model output as it has no influence on it. The last column focuses on the built-in evaluation function of the keras model. Because the counting of '#' towards the overall accuracy cannot be changed, the scores are the same for both rows even though they should differ. The score in the penultimate row has been put into parentheses, because it most likely did not mechanically exclude the '#' characters while calculating the total accuracy score. Thus, the valid score of the built-in evaluation function belongs to the first row with an astounding result of 99.36%. The value 2.21% is marked with an asterisk, because this value has to be assumed. Since the value is the same for the other two columns, it should be as well for this column as the gold standard should not be influenced by the model. All the aforementioned scores are similar based on the number of differing characters except for the score for the automated evaluation, which is of the metric accuracy.

5 Discussion

5.1 Result Elaboration

This section will compare results reported in chapter 4.2 with the results of the ICDAR-2017 competition, while the subsequent section will deal with the relation to previous studies as well the models limitations. It will also reconsider the research questions asked in the very beginning of the paper.

First and foremost, the results of the previous chapter 4.2 should be reconsidered. In an effort to facilitate this process, the table will be included again in this section. The table remains unchanged apart from one additional row to the bottom.

Overall, the results of the model were very positive, especially in regard to the results of the ICDAR-2017 competition. The team which employed a BiLSTM approach within the competition did not manage to achieve significant results of any kind. Their results for task 2 were non-exploitable for any of the text categories, for instance, English monographs [Chiron et al., 2017]. The team which achieved the best result for task 2 of said competition was coincidentally from the Computational Linguistics department of the University of Zurich as well. They managed to reach an improvement rate of 43% in the category of English monographs. This percentage is calculated as the overall improvement of the output in regards to its input. For instance, the results from our model for the similarity including the '#' symbol were reported as 98.75% while the baseline similarity was only 96.61%. This meant that the total improvement the model could achieve was 3.39%. Now we take this percentage as the 100% improvement rate and calculate the increase of the model relative to that percentage. The difference between the baseline similarity and the model is calculated as follows: the similarity achieved by our model, namely, 98.75% - the similarity of the baseline, that is to say, 96.61%. This yields a result of 2.14%. Now we calculate the relative improvement rate as shown: 2.14 / 3.39 * 100 which yields a result of 63.1%. The same score can be computed for the automated evaluation column as well in an identical manner: As the valid result is in the first row, the values for calculation should be taken from there as well, which leads us to

	Raw Input and Target Data (constant)	Predicted Input and Target Data (averaged over 10 runs)	Automated Evaluation (averaged over 10 runs)
Percentage of Correctly Matched Characters (including '#')	96.61%	98.75%	99.36%
Percentage of Correctly Matched Characters (excluding '#')	98.79%	99.19%	(99.36%)
Percentage of '#' within Target Data (constant)	2.21%	2.21%	2.21%* assumed value
Relative Improvement Rate in Percentage	-	63.1% and 33.1%	81.1%

Table 5.1: Overview of the most important scores from the Bidirectional-LSTM model

the following computation: 3.39 for the total possible improvement and an actual improvement of 2.75% (99.36 - 96.61). Resulting in a relative improvement rate of 81.1% (2.75 / 3.39 * 100). Both of these results would be far better than the result obtained by my fellow computational linguists. However, as mentioned in chapter 2.2, the '#' are ignored for the final calculation of the score. That means even if the model is capable of correctly predicting '#', it will not count towards the final score. All things considered, if the GT had been less noisy and better aligned, the overall result could have been very different.

As stated above, the score that was calculated in the preceding paragraph is not valid as a final result, thus the correct score needs to be computed by repeating the same steps as previously described: For the second score of 99.19% we get the following result: possible improvement is 1.21% while the actual improvement is 0.4%. 0.4 / 1.21 * 100 equals to 33.1%. This result is slightly inferior to that of my fellow computational linguist, but still better than any other team participating in the IDCAR-2017 competition that submitted a model for the English monograph section. This result is very positive especially in regards to the team which participated in the competition with the same basic idea of using a bidirectional-LSTM model. The difference is rather astounding considering that this group did not manage to achieve any significant results at all. An overall improvement of the input was expected, but the degree of improvement was unclear in the beginning. The reason for this expectation was that with a solidly built statistical model an improvement could have been assured no matter how slight. Even more so for a standard and well-established approach such as the LSTM model. This led to the assumption that

if an enhanced version of the LSTM was used and properly executed there should be a guaranteed improvement of the input. One surprising result, however, is the stark difference between the manual evaluation and the built-in evaluation function. An improvement of 81.1% should not be possible, especially when considering the fact that it was most likely carried out with including the '#' characters. There was no iteration of the manual evaluation reaching something above 99.25% with the '#' excluded let alone the evaluation containing '#'s. The percentage for the latter case never exceeded 98.84% which makes this case even more peculiar. The only possible explanation is that it calculated the accuracy in a different manner as opposed to my similarity approach, where it left out words which were initially wrong but remained unchanged. The similarity method did make no such distinction.

5.2 Assessment and Limitations

In the following paragraphs, the research question introduced into this thesis in chapter 1.3 is reconsidered. It shows the limitations this model and thus this thesis has and relates it to the literature read in preparation for this paper. Without further ado, the assessment concerning the validity of the answers to the research question shall be done. The nature of the posed research question is two-fold: Firstly,

> Can the output of an OCR system be further improved upon without building a new OCR system? If so: Is the output significantly better?

and secondly,

> Is this achievable by creating a neural network model that deals with post-OCR processes such as error correction? In this specific instance with a bidirectional-LSTM?

Considering the first question whether or not the output can be improved, it is definitely a yes. However, this is does depend heavily on the approach used and how well the model is built. As can be seen, for instance, with the Norvig approach of statistical nature which is able to achieve correction in the same order of magnitude as simple neural-based attempts. Another good illustration for this would be the dichotomy between the two BiLSTM attempts made. One attempt was made as part of the ICDAR-2017 competition that did not manage to achieve any improvement on the given data, whereas the attempt from Kayabas et al. managed to achieve resounding success even amongst other top-tier models. In the first case the output was definitely not significantly better while in the latter the authors claimed that the improvement rate reached tremendous 97-98%. In the case of my model, this

question can definitely be answered with yes as well. A minimum improvement rate of 33% from the input is definitely a significant amount. The evaluation set is comprised of approximately 190'000 characters, which means that an absolute increase of 0.4% means a total of 760 characters were changed out of roughly 2'300 possible characters.

The second research question was already mostly answered within the preceding paragraph. Nonetheless, as seen in the previously given example, this goal is certainly achievable. Most approaches that deal with OCR post-processes do so in a neural manner. This can be seen in both papers from the IDCAR competitions as well as the most recently cited one, which focused on the BiLSTM model. The model created in the making of this thesis and the results it yielded definitely support that argument as well.

The limitations in this thesis clearly lie within the data set as well as the built model. These two major limitations go hand-in-hand, because the data set was used to build the model and the model, on the other hand, was used to evaluate the data set. The major issue within the data set is two-fold: Firstly, the OCR-ed strings were in such bad shape that it could not be made out what was supposed to be written originally at a given point. This resulted in difficulties in the alignment process, because of certain missing parts, and thus could not be reasonably aligned. And secondly, the faulty corrections due to mistakes in the ground truth. This then resulted in the model learning to replace certain input characters with the filler '#', which was wasted potential to say the least.

In a future study, one major improvement could be to enhance the data set by not including any noise at all or at least the bare minimum. The filler character '#' alone amounted to approximately 2-2.5% of all letters available within the data, which is, given the 1 million characters I used, a stunning amount of 20'000 - 25'000 characters that were only '#'. Whether this is done on a manual or automated basis is up to future researchers. If perfect training data was reality, one could explore potential systems which are exceptionally frail in normal condition towards data containing a lot of noise. Another potential route would be to build a model that is specifically constructed to cope with filler characters in a special way such as, for instance, ignoring them altogether during the learning process.

6 Conclusion

Through this thesis I was able to profit and learn a lot about neural networks as well as how to build such a system from scratch. In the beginning of this project I was wondering how I would even start working on it, but this was a needless worry. As soon as I started reading through the literature regarding this subject, it began to elicit questions within me related to this topic. The thirst for answers to these questions was gradually quenched whilst immersing myself into this topic. Especially two of these questions are no strangers to this thesis. They became the research questions and the drive that kept me going.I was pondering whether or not I would be capable of creating such a model myself especially after working through the two IDCAR competition papers. The IDCAR competition of 2017 proved to be the best possible paper for this thesis as I could compare my system directly against other systems within the same category.

This thesis describes the model I built using the data set from the ICDAR-2017 competition. The model was built from scratch up using the tensorflow documentation as a guiding light and it was steadily improved by adapting hyperparameters and changing the size of the input files to the most practical size. The tasks of the model were not split into two, like it had been done in previous research, but only the end result of the decoded string was measured by means of similarity to the ground truth. During this process the answers to the research questions became clearer and clearer. The results of my model were surprising with highest valid score of 33.1% relative improvement rate of the input data, that was only second to another computational linguists team in the IDCAR-2017 competition. Another surprise were the differences between the manual and automated evaluation, which contrasted strongly. One of the most important takeaway messages, however, was that it did not necessarily depend on what kind of model you were using, but rather whether or not it was well-built as was neatly illustrated with the stark contrast between the BiLSTM model built as a part of the IDCAR-2017 competition, the BiLSTM model constructed by Kayabas et al. and the model built by myself. The first being not viable at all, the second one supposedly belonging to the best models created and finally my own that would have been a solid contender within the IDCAR-2017 com-

petition. For these reasons, the research question could be answered by affirming both constituents of it to be true.

Constraints imposed upon the system came mostly from the data set. Future research into OCR post-processes could profit a lot from data sets containing less noise as too much noise deteriorates the proper functioning of the model. Because the model learns to write filler characters, this results in wasted potential of the system. This noise can be in the form of an incorrect gold standard or alignment which cannot be properly done. The fault for this can, for instance, lie within the output of the OCR system, which makes alignment not very feasible. Another direction of possible research would be to build a model which is specifically built to circumvent training on filler characters and ignores them for the learning process.

In conclusion, I can say that I learnt a lot over the course of this project, especially during the process of building the model. Creating such a system from scratch and not only modifying an already existing system helped immensely towards my understanding of neural networks in general and not only of this specific kind of NN. This thorough understanding was by far the single most valuable experience I have gained during this whole ordeal.

Glossary

accuracy A basic score for evaluating automatic **annotation tools** such as **parsers** or **part-of-speech taggers**. It is equal to the number of **tokens** correctly tagged, divided by the total number of tokens. [...]. (See **precision and recall**.)

References

G. Chiron, A. Doucet, M. Coustaty, and J.-P. Moreux. ICDAR2017 Competition on Post-OCR Text Correction. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, pages 1423–1428, Kyoto, France, Nov. 2017. IEEE. doi: 10.1109/icdar.2017.232. URL `https://hal.archives-ouvertes.fr/hal-03025499`.

A. Kayabas, A. E. Topcu, and Kiliç. Ocr error correction using bilstm. In *2021 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–5, 2021. doi: 10.1109/ICECET52533.2021.9698712.

C. Rigaud, A. Doucet, M. Coustaty, and J.-P. Moreux. Icdar 2019 competition on post-ocr text correction. In *Proceedings of the 15th International Conference on Document Analysis and Recognition (2019)*, 2019.

D. K. Sahu and M. Sukhwani. Sequence to sequence learning for optical character recognition. *CoRR*, abs/1511.04176, 2015. URL `http://arxiv.org/abs/1511.04176`.

Y. Wang. SuperOCR for ALTA 2017 shared task. In *Proceedings of the Australasian Language Technology Association Workshop 2017*, pages 124–128, Brisbane, Australia, Dec. 2017. URL `https://aclanthology.org/U17-1016`.

CV

Personal Information

Gian Radler
Ferdinand-Hodler-Strasse 22
8049 Zürich
gian.radler@uzh.ch

Education

Since 2019	Bachelor Studies in Computational Linguistics and Language Technology at the University of Zurich

Professional and part-time activities

2020-2021	Tutorials PfL and ICL II
2021-2022	OCR-project at UZH

A Miscellaneous

This appendix contains a list of scripts, files and directories I used for my work.

- Scripts
  - `norvig.py (for reference)`
  - `sentence_alignment.py`
  - `bilstm.py`
- Text Files
  - `readme.txt`
  - `gs_aligned.txt`
  - `ocr_aligned.txt`
- Directories
  - `...\Bachelor_Thesis\ICDAR2017_datasetPostOCR_v1\ICDAR2017_datasetPostOCR_Training_10M_v1.2\eng_monograph`