FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Handwritten Text Recognition in Historical Documents

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Harald Scheidl, BSc
Matrikelnummer 00725084

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Robert Sablatnig
Mitwirkung: Dipl.-Ing. Dr.techn. Stefan Fiel

Wien, 24. Mai 2018

Harald Scheidl	Robert Sablatnig

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Handwritten Text Recognition in Historical Documents

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Harald Scheidl, BSc
Registration Number 00725084

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Robert Sablatnig
Assistance: Dipl.-Ing. Dr.techn. Stefan Fiel

Vienna, 24th May, 2018

Harald Scheidl	Robert Sablatnig

Erklärung zur Verfassung der Arbeit

Harald Scheidl, BSc
Schüttaustraße 60, 1220 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 24. Mai 2018

Harald Scheidl

Danksagung

Ich möchte mich an dieser Stelle bei meinem Betreuer Stefan Fiel bedanken, welcher immer kompetent und zeitnah meine Fragen beantwortet hat. Die Möglichkeit zur Inanspruchname eines Studienabschluss Stipendiums hat geholfen, mich am Studienende voll und ganz auf diese Arbeit konzentrieren zu können. Schließlich möchte ich mich noch bei Helene für das Korrekturlesen bedanken.

Acknowledgements

I like to thank my advisor Stefan Fiel for his competent and fast responses to all my questions which arose during the work. Additionally, I am grateful that I received a scholarship for the last months of my studies which made it possible to fully concentrate on this thesis. Finally, I want to thank Helene for proof-reading this work.

Kurzfassung

Digitalisierte historische Dokumente liegen in Form von gescannten Bildern vor. Dokumente, zu welchen der Text digital vorliegt, erleichtern jedoch die Arbeit für Historiker. Unter anderem vereinfacht sich dadurch die Textsuche. Handschrifterkennungssysteme sind in der Lage die Transkription automatisch durchzuführen. Im Wesentlichen existieren zwei Methoden zur Handschrifterkennung: Hidden Markov Models und neuronale Netze. Die im Zuge dieser Arbeit implementierte Handschrifterkennung basiert auf neuronalen Netzen. Die Eingabebilder werden mit verschiedenen Methoden verbessert, wodurch die Problemstellung für den Klassifikator vereinfacht wird. Einerseits wird der Bildkontrast normalisiert, andererseits werden die Bilder zufällig verändert um die Datensatzgröße künstlich zu erhöhen, wodurch der Klassifikator besser auf neuen Daten generalisiert. Ein weiterer, optionaler Vorverarbeitungsschritt richtet kursive Handschrift auf. Der Klassifikator selbst ist an die Problemstellung angepasst. Convolutional Neural Networks (CNNs) lernen Faltungsmatrizen zum Extrahieren relevanter Bildmerkmale. Handschrift hat eine sequentielle Gestalt, es kann somit hilfreich sein, Informationen links und rechts der aktuell betrachteten Position miteinzubeziehen. Genau dies geschieht mittels Recurrent Neural Networks (RNNs), welche für jede Position eine Wahrscheinlichkeitsverteilung über die möglichen Buchstaben liefern. Dieses Berechnungsergebnis des RNNs wird mit der Connectionist Temporal Classification Operation decodiert. Schließlich werden etwaige Rechtschreibfehler noch mittels Textkorrektur ausgebessert. Zur Evaluierung werden fünf öffentlich zugängliche Datensätze verwendet, wobei drei davon als historisch zu betrachten sind (9. Jahrhundert, 15. bis 19. Jahrhundert sowie die Zeit um das Jahr 1800). Die Ergebnisse werden mit öffentlich verfügbaren Ergebnissen anderer Methoden verglichen. Die Vorverarbeitungsschritte werden optional deaktiviert, um deren Einfluss zu analysieren. Die Ausgabesequenz des RNNs ist codiert, zum Decodieren existieren verschiedene Algorithmen, welche optional um ein Sprachmodell erweitert werden können. Sowohl die Decodierungsalgorithmen als auch die Sprachmodelle werden evaluiert. Schließlich wird noch die Effizienz der Textkorrektur ermittelt. Diese Arbeit liefert vier Beiträge zur Handschrifterkennung in historischen Dokumenten: erstens die Analyse verschiedener Parameter im implementierten System, zweitens eine Methode zur Segmentierung von Wörtern durch Decodieren des RNN Ergebnisses, drittens eine auf CNNs basierende Alternative zu RNNs und viertens einen Decodierungsalgorithmus welcher Wörterbuch und Sprachmodell integriert, gleichzeitig aber beliebige Zeichen zwischen Wörtern zulässt.

Abstract

After digitalization historical documents are available in the form of scanned images. However, having documents in the form of digital text simplifies the work of historians, as it e.g. makes it possible to search for text. Handwritten Text Recognition (HTR) is an automatic way to transcribe documents by a computer. There are two main approaches for HTR, namely hidden Markov models and Artificial Neural Networks (ANNs). The proposed HTR system is based on ANNs. Preprocessing methods enhance the input images and therefore simplify the problem for the classifier. These methods include contrast normalization as well as data augmentation to increase the size of the dataset. Optionally, the handwritten text is set upright by a deslanting algorithm. The classifier has Convolutional Neural Network (CNN) layers to extract features from the input image and Recurrent Neural Network (RNN) layers to propagate information through the image. The RNN outputs a matrix which contains a probability distribution over the characters at each image position. Decoding this matrix yields the final text and is done by the connectionist temporal classification operation. A final text postprocessing accounts for spelling mistakes in the decoded text. Five publicly accessible datasets are used for evaluation and experiments. Three of these five datasets can be regarded as historical, they are from the 9^{th} century, from the 15^{th} until the 19^{th} century and from around the year 1800. The accuracy of the proposed system is compared to the results of other authors who published their results. Preprocessing methods are optionally disabled to analyze their influence. Different decoding algorithms and language models are evaluated. Finally, the text postprocessing method is analyzed. The four main contributions of this thesis are: (1) analysis of different parameters and different architectures of the proposed system, (2) a word segmentation approach using a decoding method for the RNN output, (3) a CNN based replacement of the RNN layers and (4) a decoding algorithm which integrates a dictionary and language model while still allowing arbitrary non-word characters between words.

Contents

CHAPTER 1

Introduction

Handwritten Text Recognition (HTR) is the task of transcribing handwritten text into digital text. It is divided into online and offline recognition [PF09]. Online recognition is performed while the text to be recognized is written (e.g. by a pressure sensitivity device), therefore geometric and temporal information is available. Offline recognition, on the other hand, is performed after the text has been written. The text is captured (e.g. by a scanner) and the resulting images are processed. Online recognition is regarded as the easier problem [PF09]. Challenges regarding HTR include the cursive nature of handwriting, the variety of each character in size and shape and large vocabularies [SDN16]. Currently, huge amounts of historical handwritten documents are published by online libraries [SRTV14]. Transcribing and indexing makes these documents easier to access [SRTV14]. An illustration of HTR in the domain of historical documents is given in Figure 1.1 which shows one text-line from the Ratsprotokolle dataset and its transcription.

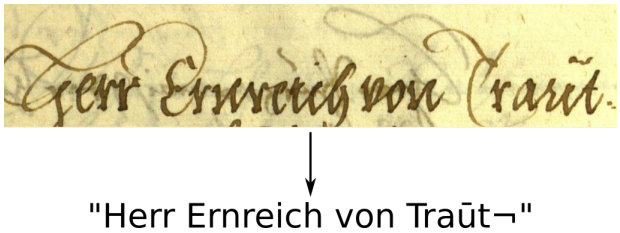

Figure 1.1: Illustration of HTR: a digital text is outputted by the HTR system for an image containing text.

1.1 Scope of Discussion

This thesis focuses on the classifier, its parameters, preprocessing methods for the input and a text-postprocessing method for the output. Only offline HTR is discussed because images of handwritten text serve as input. The investigated classifier builds on Artificial Neural Networks (ANNs). Hidden Markov Models (HMMs) are also applied to HTR tasks. However, the published results of HTR contests (see Sánchez et al. [SRTV14] or Sánchez et al. [SRTV16]) show that ANNs outperform HMMs, therefore ANNs have been chosen for this thesis. Document analysis methods to detect the text on a page or to segment a page into lines are not discussed. The only exception is word-segmentation, which is presented because the datasets are annotated on line-level. Therefore, one possibility is to segment text-lines into words and classify each word on its own, while another one is to avoid segmentation and feed complete lines into the classifier.

1.2 Methodological Approach

The proposed system makes use of ANNs, an illustration is given in Figure 1.2. Multiple Convolutional Neural Network (CNN) layers are trained to extract relevant features from the input image. These layers output a 1D or 2D feature map (or sequence) which is handed over to the Recurrent Neural Network (RNN) layers. The RNN propagates information through the sequence. Afterwards, the output of the RNN is mapped onto a matrix which contains a score for each character per sequence element. As the ANN is trained using a specific coding scheme, a decoding algorithm must be applied to the RNN output to get the final text. Training and decoding from this matrix is done by the Connectionist Temporal Classification (CTC) operation. Decoding can take advantage of a Language Model (LM).

There are two reasons for preprocessing: making the problem easier for the classifier and (on-the-fly) data augmentation. Contrast normalization, removing the slant from the text and word segmentation are discussed as methods which simplify the problem. Data augmentation is implemented by random transformations to the dataset images. The decoded text can contain spelling mistakes, therefore a text-postprocessing method is applied to account for them.

1.3 Contribution

There are four main contributions of this thesis:

- Analysis of multiple ANN architectures and evaluation of their parameters: a HTR system is implemented and experiments are conducted to analyze the influence of preprocessing and postprocessing methods and of different decoding strategies.
- Dilated Convolution: a pure-convolutional replacement for the RNN layers is proposed, which is based on dilated convolutions. This makes it possible to

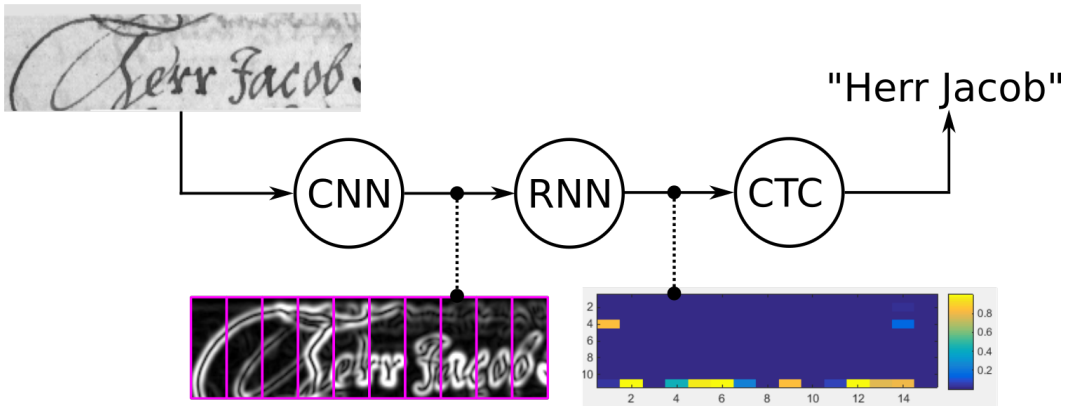

Figure 1.2: Overview of the proposed HTR system: the input image is first fed into the CNN layers to extract features. Afterwards, the output is regarded as a sequence (1D in this case) of features and serves as input to the RNN layers. The RNN outputs a matrix containing character probabilities for each time-step of the sequence. This matrix is decoded into the final text by a CTC decoding algorithm.

implement a HTR system using deep-learning frameworks which do not feature RNNs.

- Word segmentation: an implicit word segmentation based on a decoding algorithm for the RNN output is proposed and evaluated. This method outperforms other approaches which do not take advantage of learning in the domain of historical documents.
- CTC word beam search decoding: an algorithm which integrates a dictionary and a LM but still allows arbitrary non-word characters like punctuation marks between words.

1.4 Structure of Thesis

Chapter 2 starts by introducing the five datasets used to evaluate the proposed HTR system. Afterwards, ANNs with a strong focus on topics which are relevant for HTR are discussed. Two types of ANNs play a particular role in this domain, namely CNNs and RNNs, therefore these are presented in more detail. A short introduction to HMMs is given because these models are widely used in domains concerned with sequence recognition. For both the ANN and HMM approach, related work is discussed. Afterwards, Chapter 3 describes the parts of the proposed HTR system. First, preprocessing methods are discussed. Then, the classifier itself is described. This includes the architecture of the ANN and the decoding algorithms which map the RNN output onto the outputted text. Finally, a text-postprocessing method is discussed which accounts for spelling mistakes. Chapter 4 presents the results of the HTR system on the five presented datasets. Also the conducted experiments are discussed. Finally, Chapter 5 concludes the thesis.

CHAPTER 2

State of the art

This chapter first introduces the five datasets used to evaluate the proposed HTR system. Afterwards, the foundations of ANNs are presented. The focus lies on the parts relevant for HTR, namely CNNs to extract features and RNNs to propagate information through the image. Loss calculation and decoding are done by the CTC operation. Finally, two successful HTR approaches are presented: the first method builds on HMMs, while the second one uses ANNs.

2.1 Datasets and Evaluation Metric

Five datasets are used to evaluate the proposed HTR system. To get an idea of how the handwritten text looks like, one sample per dataset is shown in Figure 2.1. This section first gives a comparison of the dataset statistics. Afterwards each dataset is presented in detail. Finally, the evaluation method for those datasets is described.

2.1.1 Datasets

In Table 2.1 a comparison of dataset statistics is given. The number of characters is between 48 (SaintGall) and 93 (Bentham). A similar character distribution can be found for all datasets, an example for SaintGall is shown in Figure 2.2. It is written in Latin, the occurrence frequency of the character “i” is around 10% while it is 0.01% for the character “X”.

Unique words are calculated by splitting a text into words and then only count the unique ones. IAM has the highest number of unique words, while CVL has the lowest number. Out-Of-Vocabulary (OOV) words are words contained in the test-set but not in the training or validation-set. The performance of a word-level LM is influenced by the percentage of unknown words, therefore the number of OOV words gives a hint if using such a LM makes sense.

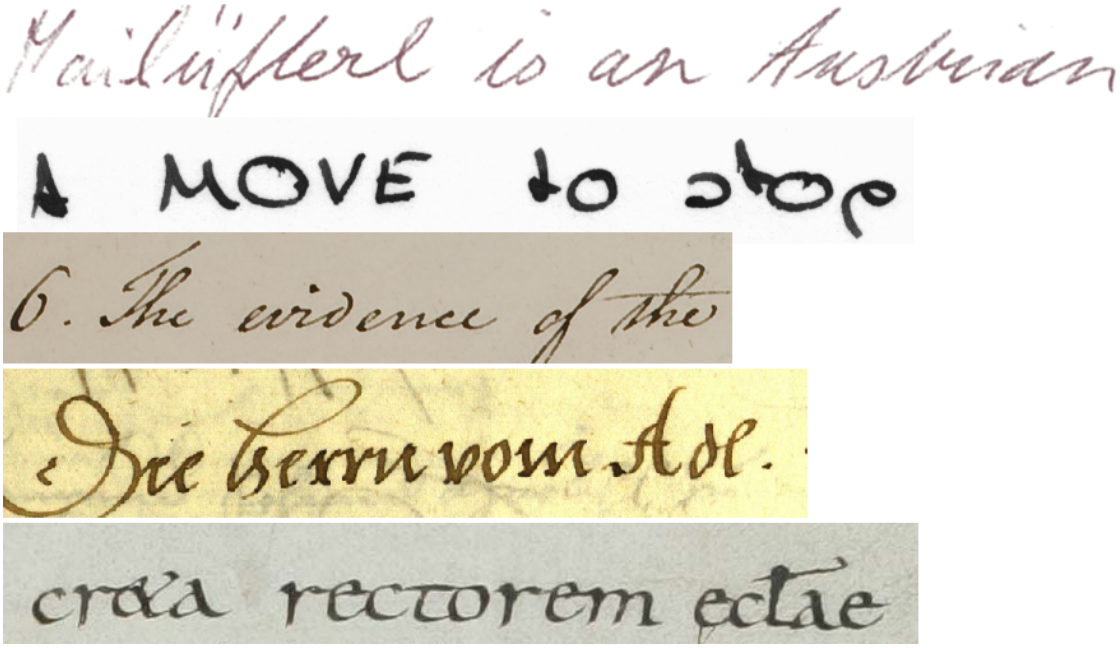

Figure 2.1: One sample per dataset is shown. From top to bottom: CVL, IAM, Bentham, Ratsprotokolle and SaintGall.

Dataset	#chars	#unique words			OOV	#lines		
		train	valid.	test		train	valid.	test
CVL	54	383	275	314	1.91%	11438	652	1350
IAM	79	11242	2842	3707	37.87%	10244	1144	1965
Bentham	93	8274	2649	1911	17.63%	9198	1415	860
Ratsprotokolle	90	6752	1471	1597	32.06%	8366	1014	1138
SaintGall	48	4795	895	1293	55.45%	1052	143	215

Table 2.1: Dataset statistics.

Because the classifier introduced later in this thesis uses line-images as input, the number of lines of a dataset corresponds to the number of samples for training, validation and testing. All datasets have a total number of lines of around 10000, the only exception is SaintGall which has a little more than 1000 lines of text.

Some notes on the datasets:

- CVL [KFDS13]: this dataset is mainly dedicated for writer retrieval and word spotting tasks. It contains handwriting from 7 texts written by 311 writers. The texts are in English and German language. There is only annotation for words but not for punctuation marks. Data is annotated on word-level.
- IAM [MB02]: contains English text. The version of the dataset described by Marti

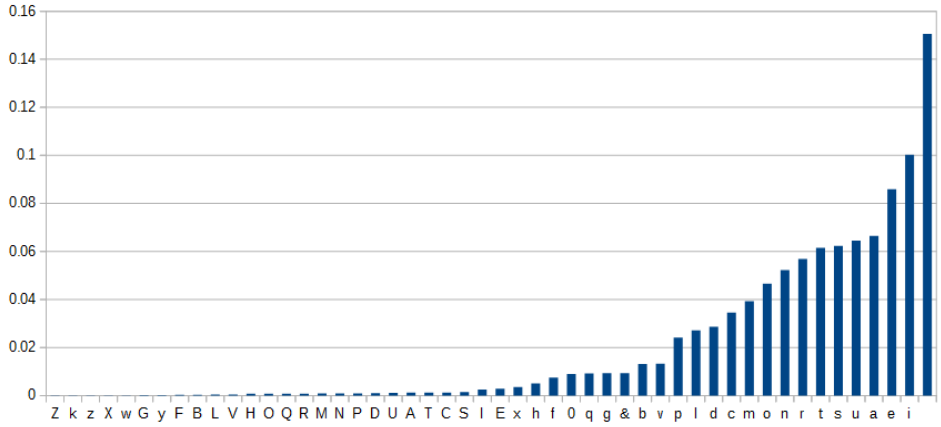

Figure 2.2: Sorted character frequency in SaintGall dataset. The most frequent characters are " ", "i", "e", "a" and "u" while the least frequent characters are "Z", "k", "z", "X" and "w".

and Bunke [MB02] contains 1066 forms filled out by approximately 400 writers. It has since been extended, the IAM website [MB] lists 1539 forms and 657 writers. Data is annotated on word-level and on line-level.

- Bentham [SRTV14]: written by Jeremy Bentham (1748-1832) and his secretarial staff. Contains 433 pages. Mostly written in English, some parts are Greek and French. Data is annotated on line-level and partly on word-level.
- Ratsprotokolle [SRTV16]: texts from council meetings in Bozen from 1470 to 1805. An unknown number of writers with high variability in writing style created this dataset. It is written in Early Modern German. Data is annotated on line-level.
- SaintGall [FFFB11]: this dataset contains 60 pages with 24 lines per page. It is written in Latin by a single person at the end of the 9^{th} century. It describes the life of Saint Gall. Data is annotated on line-level.

2.1.2 Evaluation Metric

The most common error measures for HTR are Character Error Rate (CER) and Word Error Rate (WER) [Blu15]. CER is calculated by counting the number of edit operations to transfer the recognized text into the ground truth text, divided by the length of the ground truth text (see Equation 2.1) [Blu15]. The numerator essentially is the Levenshtein edit-distance and the nominator normalizes this edit-distance [Blu15]. For WER, the text is split into a sequence of words. This word sequence can then be used to calculate the WER just as a character sequence is used to calculate the CER. Both CER and WER can take on values greater than 100% if the number of edit operations exceeds the ground truth length [Blu15]: if "abc" is recognized but the ground truth is "xy", the CER has a value of $3/2 = 150\%$.

$$CER = \frac{\#insertions + \#deletions + \#substitutions}{length(GT)} \tag{2.1}$$

As an example the CER and WER of the recognized text “Hxllo World” with the ground truth text “Hello World” is calculated. The character edit-distance is 1, the length of the ground truth text is 11, therefore $CER = 1/11$. For WER the unique words are written to a table with unique identifiers yielding w_1=“Hxllo”, w_2=“Hello”, w_3=“World”. Then, each word is replaced by its identifier from the table, so the two strings become “$w_1 w_3$” and “$w_2 w_3$”. The word edit-distance is 1, the ground truth length is 2, therefore WER=1/2.

2.2 Artificial Neural Networks

This section introduces ANNs with a strong focus on topics which are necessary to understand their application in the context of HTR. The foundations of ANNs are presented. Building on this knowledge, two special types of ANNs, namely CNNs and RNNs are discussed.

2.2.1 Basics

First, the basic building block of an ANN, the neuron, is introduced. Afterwards the combination of neurons to form neural networks with multiple layers and the training of such networks is presented.

Perceptron

First ideas for ANNs date back to the 1940s [GBC16]. Linear models of the form $y(x, w) = H(\sum_i w_i \cdot x_i)$ are used for binary classification tasks, where x represents the input vector, w the parameter vector, i the indexing variable of the vectors and H the Heaviside step function [GBC16]. Both models presented in the following use linear functions. The McCulloch-Pitts Neuron has excitatory and inhibitory inputs with fixed (manually set) parameters. Rosenblatt introduces the perceptron which is able to learn the parameters by using training samples for which the target categories are known [GBC16]. An illustration of a perceptron is shown in Figure 2.3. As both models build on linear functions they are not able to approximate all possible functions. One of the cases where linear models fail is the XOR function: there is no straight line which can be drawn to separate the two classes as can be seen in Figure 2.3 [GBC16].

Multilayer Perceptron

In the 1980s the main idea to improve ANNs was to combine simple units (neurons) to complex neural networks [GBC16]. Those neural networks are able to approximate

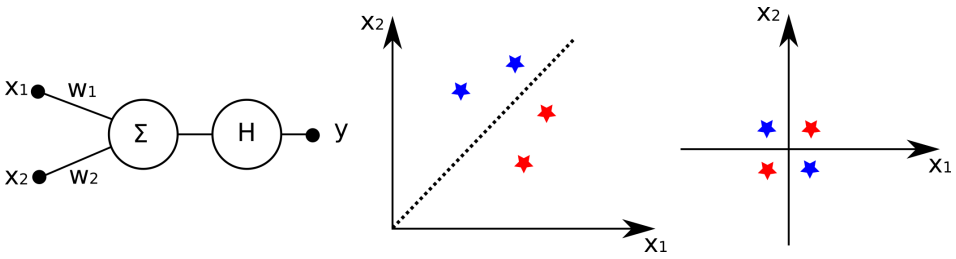

Figure 2.3: Left: network diagram of the perceptron for two inputs. Middle: a linear classifier learns to separate two classes (red, blue) by a straight line (dotted line). Right: the XOR problem, for which no (straight) separating line can be found.

arbitrary functions and therefore can solve the XOR problem [GBC16]. Multiple connected layers of neurons are used, which is the reason for the name Multilayer Perceptron (MLP). The first layer of neurons is called input layer, the last layer output layer, and the layers in between are called hidden layers. There are different ways to count the layers, this thesis sticks with the counting method described by Bishop [Bis06] by taking the number of hidden layers plus 1.

A single neuron, as shown in Figure 2.4 on the left, first applies a linear combination of its inputs to calculate an activation $a = \sum_i w_i \cdot x_i$. Afterwards a nonlinear activation function g is applied to the activation $y = g(a)$ [Bis06]. Instead of calculating the activation for each neuron separately, it can be advantageous to combine all input neurons to a vector x and all weights to a matrix W and then calculate the activation of the layer by $a = W \cdot x$. It is possible to visualize such an ANN with a computational graph, an example can be seen in Figure 2.4 on the right. This graph specifies how input, hidden and output layers are connected by functions.

The importance of the nonlinearity can be seen when multiplying out the parameters of multiple layers of purely linear neurons. See the computation graph in Figure 2.4. The output of the hidden layer is $h = W_1 \cdot x$ which is the input for the next layer $y = W_2 \cdot h$. Combining both equations shows that the effect of multiple linear layers has no more representational power than a single linear layer: $y = W_2 \cdot h = W_2 \cdot W_1 \cdot x = (W_2 \cdot W_1) \cdot x = W \cdot x$.

Training

The first step to train a neural network is to define a cost (or loss) function. The type of cost function depends on the task: for regression tasks usually the sum-of-squares error function is used, while for classification tasks the cross-entropy error function is used [Bis06]. As HTR is a classification problem, a cost function for a (binary) classification model is derived in the following. The training-set consists of samples indexed by the variable n. A sample contains input data x_n and a target class $t_n \in \{0, 1\}$. The output of the ANN can be interpreted as the probability of seeing class 1 given x and w [Bis06]. The binary distribution over the classes is given by Equation 2.3. Taking the negative

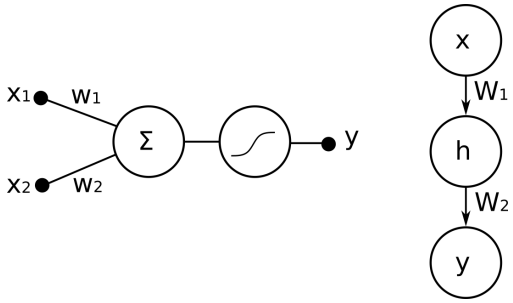

Figure 2.4: Left: network diagram of a single neuron as used in the MLP. The Heavyside step function from the perceptron is replaced by a nonlinearity. Right: Computational graph. The circles represent input x, hidden state h and output y and are vector-valued. The mapping from one layer to the next layer is achieved by multiplying with the matrices W_1 and W_2.

log-likelihood and summing over all training samples yields the cost function as shown in Equation 2.4 [Bis06].

$$y_n = y(x_n, w) \tag{2.2}$$

$$p(t_n|x_n, w) = y_n^{t_n} + (1 - y_n)^{1-t_n} \tag{2.3}$$

$$E(w) = -\sum_n t_n \cdot ln(y_n) + (1 - t_n) \cdot ln(1 - y_n) \tag{2.4}$$

Training the model means searching in parameter space for a (local) minimum of the cost function as can be seen in Figure 2.5 [Bis06]. This is done by starting with an initial (random) set of parameters and iteratively moving in parameter space downhill the cost function [Bis06]. The gradient of the cost function points in the direction of greatest increase of the cost function, therefore a step in the opposite direction decreases its value. This yields the gradient descent formula $w' = w - \eta \cdot \nabla_w E(w)$ [Bis06]. The step size is controlled by the learning rate η. More advanced algorithms exist which adjust the learning rate depending on local properties of the gradient [MH17]. Error back-propagation is an efficient way to calculate the partial derivatives $\frac{\partial E(w)}{\partial w_i}$ and hence the gradient $\nabla_w E(w)$ [LBOM12].

The cost function presented in Equation 2.4 takes all training samples as input to perform one single step of training. This is called batch training [GBC16]. Imagine doubling the training-set size by simply duplicating each sample. Then, the time for calculating the cost function also doubles, while the gradient is the same (besides a constant factor) [Bis06]. Such redundant calculations can be avoided by evaluating the cost function only for random subsets of the training-set, which is known as mini-batch training [Bis06]. Online training is the extreme case of just taking one sample per training step [Bis06].

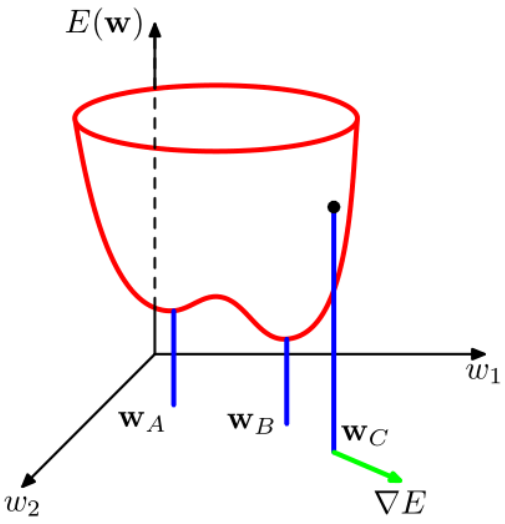

Figure 2.5: Illustration of a cost function E over two parameters w_1 and w_2. The parameter vectors w_a and w_b are two local minima of the cost function. For w_c the gradient is shown. Following in the opposite direction of the gradient guides the training algorithm to a local minimum. Image taken from Bishop [Bis06].

Capacity

Machine learning differs from pure optimization that not only the training error should be minimized, but also the error on the test-set. Performing well on this previously unseen test-set is called generalization [GBC16]. Another important term in this context is overfitting. The cost function with respect to the training-set gets minimized while training. When evaluating the cost function on a validation-set, one can see that this function is U-shaped (see Figure 2.6): its value first decreases similar to the training cost function, until it reaches a minimum, afterwards it increases again [GBC16]. The reason for this rise of the validation cost is overfitting: the model learns properties which are only useful for the training-set, but which do not generalize onto the validation-set [GBC16]. In the next section different techniques are presented to avoid overfitting.

The capacity of a model is the ability to fit a wide range of functions [GBC16]. It should be big enough to learn the relevant patterns in the data, but small enough to avoid overfitting by learning patterns that do not help when applying the model to the test-set. Only models with a large enough capacity are able to overfit the training data [GBC16]. The Vapnik-Chervonenkis (VC) dimension from statistical learning theory is a theoretical measurement for the capacity of a binary classifier [VLL94]. The key insights of this theory are that the gap between training and test error increases as the model capacity increases, but decreases as the number of training samples increases [GBC16].

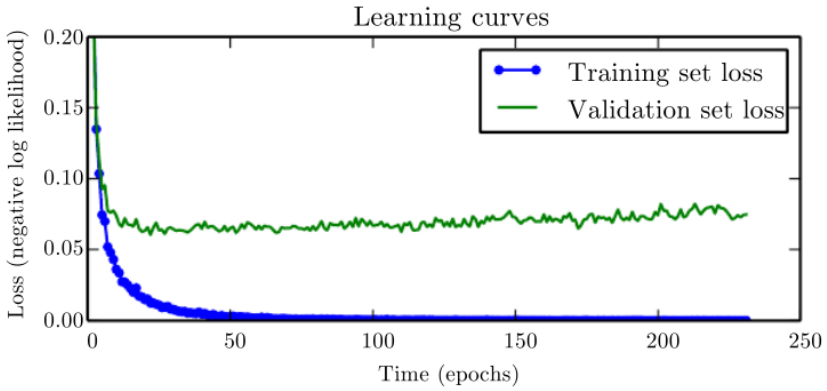

Figure 2.6: The cost function evaluated for the training and validation-set. If the model has enough capacity, the training error keeps decreasing, while the validation error stagnates after some time and then starts increasing again. Image taken from Goodfellow et al. [GBC16].

Regularization

Goodfellow et al. defines regularization as "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error" [GBC16]. Early Stopping is a technique which terminates the training process at a local minimum of the validation cost function [GBC16]. This is done by calculating the validation cost after each epoch of training. Training gets stopped if there is no more improvement of the validation cost for a predefined number of epochs [GBC16]. Early Stopping works by limiting the search volume in parameter space [Bis06]. This is similar to L^2 regularization, which constraints the norm of the parameters $||w||_2$ and therefore also constraints the search space [GBC16].

Generalization gets better when training uses more data [GBC16]. Dataset augmentation is a way to increase the size of the training-set by creating fake data from the original training data [GBC16]. Simple transformations like shifting, rotating or scaling the original data can improve the generalization error, even when the model itself is shift invariant like it is the case for CNNs [GBC16]. LeCun et al. [LBBH98] report a drop of the test error rate from 0.95% to 0.8% by distorting the original images and adding them to the training data. Examples for data augmentation applied on a handwritten word can be seen in Figure 2.7. Regularization can also be defined via the structure of the neural network [Bis06]. CNNs are an example for such a restriction which limit the input of neurons to a small region called receptive field [Bis06]. More details on this type of neural network can be found in the next section.

2.2.2 Convolutional Neural Networks

The traditional image classification approach is to extract local features from the image, which then serve as input to a classifier [LBBH98]. An ANN can be used as such a

Figure 2.7: Data augmentation increases the size of the training-set by applying transformations to the original images. Left: original image. Middle: sheared image. Right: resized image.

classifier [LBBH98]. It is also possible to directly input an image into an ANN and let it learn the features itself. The problems with this approach are [LBBH98]:

- Large images require many parameters to be learned. Think of an input image with size 100×100 pixels and a fully connected first layer with 100 hidden neurons. This gives 10^6 weights to be learned in the first layer. A large number of parameters increases the capacity of the model and therefore also increases the number of samples needed to train the model.
- Invariance to translation or size variance is difficult to achieve. It is possible to learn this invariance by showing training samples at different locations and sizes. This results in similar (redundant) weight patterns at different locations, such that features can be detected wherever they appear in the image. Again, showing the same images at different positions and sizes increases the size of the training-set.
- The topology of images is ignored. Images have a strong local 2D structure, but in a fully connected first layer the ordering of the input neurons (image pixels) does not matter.

CNNs are able to tackle these problems and are tremendously successful in practical applications [GBC16]. CNNs work with data that is organized in a grid-like structure, such as images which have a 2D structure or plain audio data which have a 1D structure [GBC16]. The key difference between MLPs and CNNs is that in CNNs at least one layer uses a convolution operation instead of the usual matrix multiplication [GBC16].

A diagram of a CNN layer can be seen in Figure 2.8 on the left. It consists of three parts which are afterwards discussed in more detail:

- Convolution: the activation of the neuron is calculated by the convolution of an input and a kernel.
- Nonlinearity: a nonlinear function is applied to the activation of the neuron.
- Pooling: this operation replaces the output at a certain location by a summary statistic of nearby outputs.

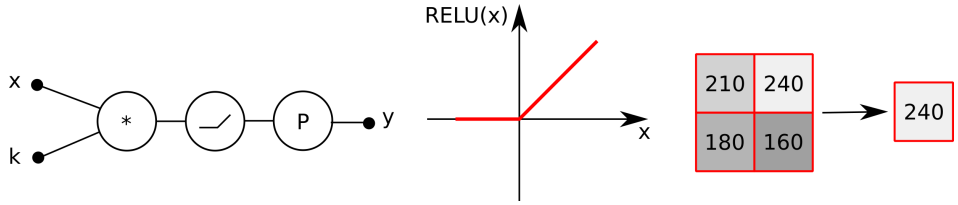

Figure 2.8: Left: network diagram of a convolutional layer. An image patch x is convolved by a kernel k. As a nonlinearity the RELU function is applied. Finally, pooling generates a summary statistic for small regions. Middle: plot of the RELU function. Right: max-pooling of a 2×2 region.

Convolution is an operation on two functions and outputs a third function. The operation is commutative, associative and distributive. For computer vision tasks a discrete version of convolution is used. For 1D data the formula to calculate the convolution is shown in Equation 2.5. In the context of CNNs, the first argument x is called input, the second argument k kernel and the output y is called feature map. The time-indexes are denoted by t and τ. Because convolution is commutative, the arguments to index x and k can be swapped in a way that is convenient for implementations.

$$y(t) = x(t) * k(t) = \sum_{\tau=-\infty}^{\infty} x(\tau) \cdot k(t-\tau) = \sum_{\tau=-\infty}^{\infty} x(t-\tau) \cdot k(\tau) \tag{2.5}$$

For the 2D structure of images a 2D version of discrete convolution is needed, which is shown in Equation 2.6. Indexing is now achieved by i and m in the first dimension and j and n in the second dimension. Only the version used in implementation with swapped arguments is given.

$$y(i,j) = x(i,j) * k(i,j) = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(i-m, j-n) \cdot k(m,n) \tag{2.6}$$

A Rectified Linear Unit (RELU) is used as a nonlinear activation function in CNNs [GBC16]. It is defined as $y = g(a) = max(0, a)$ on the activation a of the neuron. The RELU essentially zeros out the negative part of its argument while it keeps the positive part untouched. A plot is shown in Figure 2.8 in the middle.

Pooling [GBC16] is added on top of the nonlinearity: it replaces the output at a certain location by a summary statistic of nearby outputs. An example for pooling is max-pooling on a 2×2 neighborhood which is illustrated in Figure 2.8 on the right: the feature map is divided in 2×2 squares and for each of them, only the maximum value is kept. This operation reduces the height and width of the feature map by a factor of 2.

Training of CNNs is similar to training MLPs. A key difference between those two ANN types is that a CNN layer shares the weights of its convolution kernel: the kernel is shifted through the image, therefore the same weights are applied to multiple image positions. The backpropagation algorithm to calculate the gradient is slightly modified to account for this weight sharing [LBBH98]. This is achieved by first calculating all partial derivatives as in a fully connected layer, and then summing up all partial derivatives belonging to the same kernel weight. The kernel weights learned for visual tasks are sensitive for edges, an example for such kernels is shown in Figure 2.9.

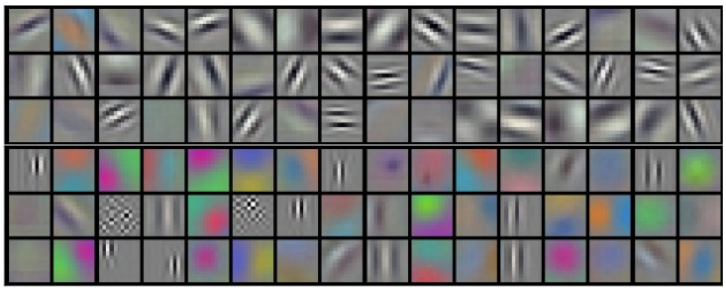

Figure 2.9: Convolutional filter kernels of the first network layer learned for an image classification task. Image taken from Krizhevsky et al. [KSH12].

2.2.3 Recurrent Neural Networks

A RNN is a special kind of ANN which is used to model sequential data such as handwritten text or speech [GSK+17]. Sequential data can be of arbitrary length along the time axis, where time axis in the context of speech is clear, while in the context of handwritten text it means the axis along which the writing happens. This is from left to right for e.g. English text and the other way round for e.g. Arabic text. The intuition of using RNNs is that relevant information can be scattered around multiple time-steps [GBC16]. An example is shown in Figure 2.10, where it is easy to identify the character "n" when looking at the whole image, but it is impossible to tell if it is an "n" or "u" when looking at the character on its own.

Additionally to the input and output sequence x_t and y_t, RNNs also have an internal state sequence h_t, which memorizes past events. The internal state h_t is a function of the current input x_t and the last internal state h_{t-1}, that means $h_t = f(h_{t-1}, x_t)$ [GBC16]. The detailed formulas for calculating the activation a_t, the hidden state h_t, the output o_t and the normalized output y_t are shown in Equations 2.7 to 2.10 [GBC16].

$$a_t = W \cdot h_{t-1} + U \cdot x_t \tag{2.7}$$

$$h_t = tanh(a_t) \tag{2.8}$$

$$o_t = V \cdot h_t \tag{2.9}$$

$$y_t = softmax(o_t) \tag{2.10}$$

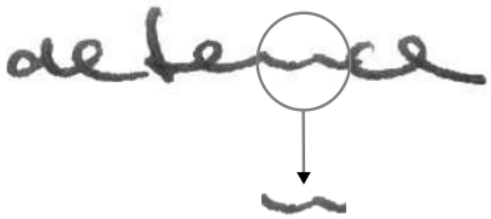

Figure 2.10: The importance of temporal context in HTR. When looking only at the marked character it is difficult to tell if it is an “n” or “u”. Using context from left and right makes it easy to tell that is is the letter “n”. Image taken from Graves et al. [GLF+09].

The computational graph of a RNN has cycles which represent the connections between neighboring time-steps. Unfolding (i.e. replicating) the graph of a RNN to the length of the input sequence results in a conventional ANN. This ANN is trained by standard methods, i.e. first calculating the gradient of the error function and then updating the parameters with an algorithm like gradient descent. The gradient calculation is called Backpropagation Through Time (BPTT) when applied to an unfolded graph [GBC16]. Similar as in the case of CNNs, the parameters (U and V) are shared between different time-steps. An example for a computational graph of a RNN and its unfolded graph can be seen in Figure 2.11.

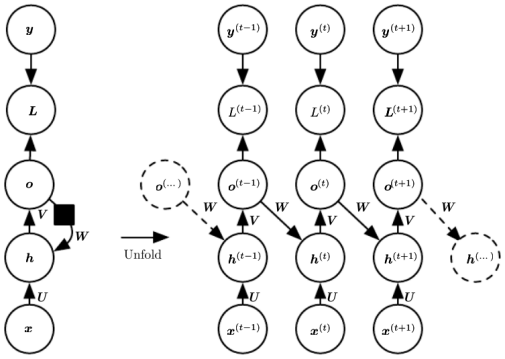

Figure 2.11: Left: Computational graph of a RNN. The recurrent input is o_{t-1}, the RNN is therefore slightly different from the presented equations which use h_{t-1} instead. Right: the same RNN but unfolded in time. Image taken from Goodfellow et al. [GBC16].

When used for practical purposes RNNs have the problem of vanishing gradients [Hoc91]. This can be seen when looking at an unfolded graph: long-term memory is achieved by applying the same function over and over again in the form $h_t = f(f(f(...), x_{t-1}), x_t)$. Depending on the function f this either vanishes or explodes the backpropagated error signal. Goodfellow et al. [GBC16] give a detailed explanation by decomposing the function f, with the relevant part being the matrix W, into its eigenvalues and eigenvectors.

Raising the (diagonal) eigenvalue matrix to the power of k, diagonal elements $|d_{ii}|$ smaller than 1 tend to 0, whereas values larger than 1 tend to ∞ when increasing k.

Long Short-Term Memory

Hochreiter [Hoc91] discusses the vanishing gradient problem of RNNs. To avoid them he introduces Long Short-Term Memory Recurrent Neural Networks (LSTM) [HS97]. A LSTM block learns when to remember and when to forget past events which is achieved by introducing gating neurons [GBC16].

A LSTM block is shown in Figure 2.12. There are two recurrences, an outer and an inner one [GBC16]. The outer recurrence is realized by using the output y_t from time-step t as input for time-step $t + 1$. The inner recurrence is the linear self-loop of the state cell. This inner loop is called Constant Error Carousel (CEC) [HS97]. Gates control the information flow in a LSTM block: the input gate controls how much the input data effects the state cell and the output gate controls how much the internal state effects the output [GSC99]. A problem with the original LSTM formulation is that the state cell accumulates more and more information over time, therefore the internal state has to be manually reset [GSC99]. To solve this problem Gers et al. [GSC99] introduces a forget gate to reset or fade out the internal state. Peephole connections are added to a LSTM block such that the cell can take direct control of the gating neurons in the next time-step [GSK+17].

The vanishing gradient problem is solved by the CEC. When the error signal flows backward in time and no input or error is present (gates are closed), the signal remains constant, neither growing nor decaying [GSC99]. While a vanilla RNN is able to bridge time-lags of 5-6 time-steps between input event and target event, LSTM is able to bridge 1000 time-steps [GSC99]. This and the stable error propagation make LSTMs a popular implementation of RNNs which are widely used in practice when working with sequences [GBC16].

Multidimensional LSTM

RNNs process 1D sequences and therefore have one recurrent connection. Graves et al. [GFS07] introduce Multidimensional RNNs (MDRNN) by using n recurrent connections to process nD sequences. In the forward pass the hidden layer gets input from the data and from the n recurrent connections one time-step back as shown in Figure 2.13. A suitable ordering of the data points is needed such that the RNN already processed points from which it receives its inputs at a certain time-step. One possible ordering for 2D data is shown in Figure 2.13. Gradient calculation is done by an extension of the BPTT for n dimensions. Also 1D LSTMs can be extended to Multidimensional LSTMs (MDLSTM) by introducing n self-loops with n forget gates.

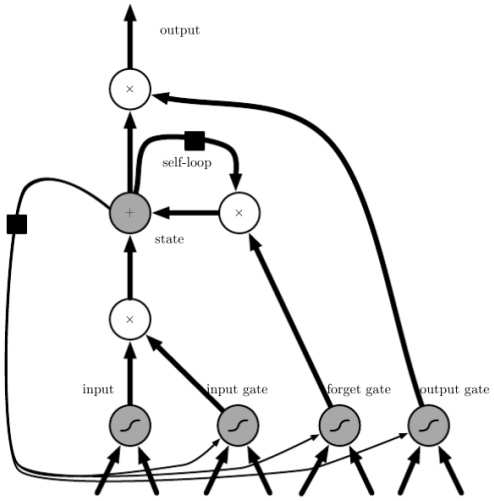

Figure 2.12: LSTM block consisting of gating neurons and an internal cell. The optional connections from the cell to the gating neurons are called peephole connections. Image taken from Goodfellow et al. [GBC16].

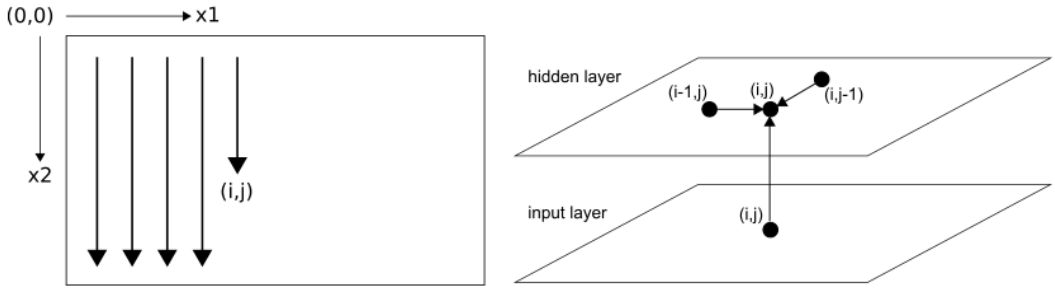

Figure 2.13: Left: a possible ordering for data-points for the 2D case of MDRNN. Right: data-flow in the forward pass. Image taken from Graves et al. [GFS07].

Dilated Convolution

Dilated Convolution (DC) is a modification to standard CNN. However, it is used in tasks which usually use RNNs, therefore the discussion about DC is done in this section [ODZ+16]. Oord et al. describe DC as "convolutions with holes" [ODZ+16]. The kernel k is not applied to each element of the input x as in vanilla convolution, but only to each n^{th} element. The formula for 1D convolution is modified as shown in Equation 2.11 to account for the dilation width n [YK16].

$$y(t) = x(t) *_n k(t) = \sum_{\tau=-\infty}^{\infty} x(t - n \cdot \tau) \cdot k(\tau) \tag{2.11}$$

As shown by Strubell et al. [SVBM17] multiple layers of DC with exponentially increasing dilation width can be stacked to form an Iterated Dilated Convolutional Network (IDCN). IDCN is used to propagate information through a long sequence with only a few DC layers. Given the kernel size w, the width of the receptive field r of a CNN increases linearly in the number of layers l, as can be seen in Equation 2.12. For IDCN the width increases exponentially as shown in Equation 2.13.

$$r_{CNN} = l \cdot (w - 1) + 1 \tag{2.12}$$

$$r_{IDCN} = (2^l - 1) \cdot (w - 1) + 1 \tag{2.13}$$

An example for a IDCN is given in Figure 2.14. There are 3 layers and a kernel width of 3, which yields a receptive field width of 15.

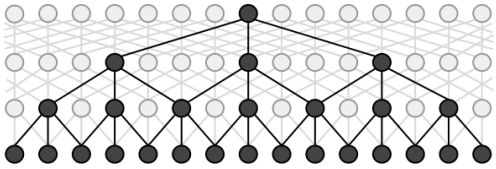

Figure 2.14: Dilated convolutional layers. Each layer increases the number of holes in the filter, while the kernel without holes has a constant size of 3. Image taken from Strubell et al. [SVBM17].

Oord et al. [ODZ+16] use IDCN for speech recognition and generation. They use a special variant called causal DC, which only makes use of the context from one side of the sequence. Yu and Koltun [YK16] use this network type for segmentation tasks and exploit the multi-scale context aggregation feature of IDCNs.

Connectionist Temporal Classification

Graves et al. [GFGS06] introduce the CTC loss and decoding operation. With a CTC output layer it is possible to train an ANN from pairs of images and labels, there is no need to specify at which location a character appears in the image. CTC also handles decoding by finding the most probable labeling from the framewise label probabilities, optionally using a LM.

The following discussion builds on Graves [Gra12] and an illustration can be found in Figure 2.16. The output y of a RNN is a matrix of shape $C \times T$, where C is the number of characters plus 1 and T is the sequence length. For each time-step, the entries for the characters form a probability distribution because of the applied softmax function (see Section 2.2.3). CTC needs one additional character (therefore the RNN outputs one extra character), the so called *blank* label. The meaning of this blank is that no character is recognized at this frame. It is needed to separate repeated characters (like in pi*zz*a)

and to prevent the RNN to predict the same character for a long time. This special label should not be confused with a whitespace character.

The probability of seeing character k at time-step t is denoted by y_k^t. A path $\pi = (\pi_1, \pi_2, ..., \pi_T)$ is a way through the matrix, going from the first to the last frame and selecting one character π_t per frame with character probability $y_{\pi_t}^t$. A path essentially encodes a labeling. The mapping from a labeling to a path can be modeled by a Finite State Machine (FSM) as shown in Figure 2.15. As can be seen, for each labeling multiple paths exist, therefore the encoding represents a one-to-many mapping. Calculating the probability of a path is done by multiplying the individual character probabilities on the path $p(\pi) = \prod_t y_{\pi_t}^t$. The collapsing function $\mathcal{B}(\pi)$ maps a path to a labeling by first removing duplicate characters and then removing the blank symbols. This inverts the operation carried out by the FSM and is a many-to-one mapping. As an example, $\mathcal{B}$("a a a - b") with "-" representing the blank gets mapped to "ab". Finally, to get the probability of a given labeling l, one has to sum over all paths yielding this labeling, i.e. $p(l|x) = \sum_{\mathcal{B}(\pi)=l} p(\pi|x)$.

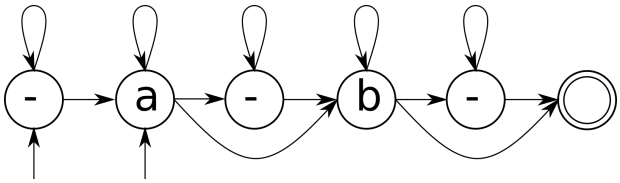

Figure 2.15: This FSM shows how to create valid paths π for the labeling l ="ab". It is created by putting a blank label at the beginning and at the ending of the labeling and in between each character. Further, transitions are inserted between consecutive characters and between consecutive characters and blanks. Examples for valid paths are π ="- a - - b" and π ="a a a b b" among others.

Decoding is done by finding the most probable labeling $l^* = argmax_l(p(l|x))$ given the output x of the RNN. Best path decoding [Gra12] is a simple approximation to find the most probable path π^*, which is the concatenation of the labels with highest score y_k^t for each frame and afterwards apply $\mathcal{B}$ to this path to get the most probable labeling $l^* = \mathcal{B}(\pi^*) = \mathcal{B}(argmax_\pi(p(\pi|x)))$. An illustration of this algorithm is given in Figure 2.16. Prefix search decoding [Gra12] is a best-first search. It expands the most promising labeling by all possible labels until the most probable labeling has been found. A heuristic to split the RNN output into smaller parts may be needed to make this algorithm feasible [Gra12]. Beam search decoding [HS16] calculates the probability of multiple labeling candidates. In each time-step the candidates are extended by all possible labels. This gives new candidates for the next time-step. To keep the algorithm feasible, only the most promising labelings are kept. It is possible to integrate a LM into the algorithm, e.g. a character-level LM. Such a LM uses character bigram probabilities to score pairs of consecutive characters. Pseudo-code and more details on the character-level LM can be found in Chapter 3. Token passing [Gra12] is another CTC decoding algorithm which can be applied when the words to be recognized are known in advance [Gra12]. This

algorithm works on word-level (the former ones work on character-level) and finds the most probable word sequence. Possible words are listed in a dictionary and bigram probabilities are assigned to consecutive words. Pseudo-code and more details on the word-level LM can be found in Chapter 3.

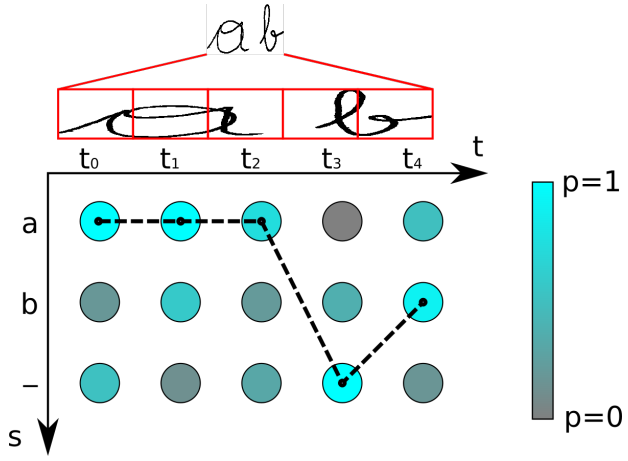

Figure 2.16: Best path decoding picks the most probable label for each time-step. This gives the path "a a a - b" in this example (dashed line). The final labeling is "ab" and is obtained by applying the collapsing function $\mathcal{B}$ to the path.

A comparison between best path decoding and beam search decoding for a given RNN output is shown in Figure 2.17 and is explained in the following. Best path decoding picks the most probable label at each time-step, this yields the path "- -" and therefore the result is "". Beam search decoding calculates the probabilities for possible labelings. The labeling "" can only be produced by the path "- -", the probability of this path is $0.6 \cdot 0.6 = 0.36$. The probability of seeing "a" is the sum over all paths ("a a", "a -", "- a") yielding "a". This gives a probability of $0.4 \cdot 0.4 + 0.4 \cdot 0.6 + 0.6 \cdot 0.4 = 0.64$. Beam search decoding is therefore able to find the correct answer in this example, while best path decoding fails.

Besides decoding, CTC is also used to train the ANN as described by Graves et al. [GFGS06]. The CTC loss function outputs the error signal for the RNN, which is then propagated backwards by BPTT to calculate the gradients. The loss for a single training sample is defined by $E(x, l) = -ln(p(l|x))$ and $\frac{\partial E(x,l)}{\partial y_k^t}$ is then passed backward to frame t and character k of the RNN. The probability $p(l|x)$ was already defined, the only difficulty for calculating it is to find all paths π yielding the labeling l, that means, all paths for which $l = \mathcal{B}(\pi)$ holds. Graves et al. [GFGS06] show a feasible dynamic programming approach for this problem.

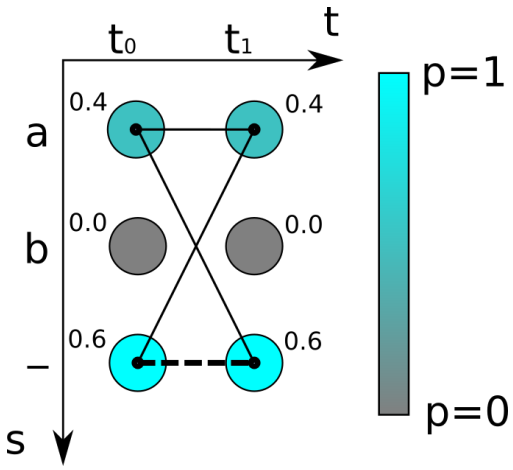

Figure 2.17: The number next to each circle gives the probability of seeing the label at a given time-step. The dashed line is the path yielding labeling "". The thin lines are the paths yielding "a". Explanation see text.

2.3 Handwritten Text Recognition

The results published for HTR competitions (e.g. see Sánchez et al. [SRTV14] or Sánchez et al. [SRTV16]) show that there are two approaches which achieve state-of-the-art performance. The first one uses HMMs, which have hidden states representing possible characters. A decoding algorithm is able to find the most probable state sequence given the input and outputs the recognized text. The second approach uses ANN. Multiple architectures exist, however the discussed publications (see Section 2.3.2) show that the use of RNNs and CTC can be seen as the least common denominator.

2.3.1 Hidden Markov Models and Hybrid Approaches

HMMs have been developed in the 1960s and have since been successfully applied to sequential data such as speech or handwritten text [Bis06]. First the foundations of HMM are discussed, afterwards practical implementations for the task of HTR are shown.

The following text is based on Bishop [Bis06]. A HMM consists of observable variables $x_t \in X$ and internal (hidden) variables $z_t \in Z$ (see Figure 2.18 on the left). Hidden variables z_t are discrete and can take on one out of k possible states. The transition probability $p(z_{t+1}|z_t)$ from one state to another is defined by a matrix A (see Figure 2.18 on the right). The probability of starting at a specific state is defined by the vector π. Finally, the probability distribution $p(x_t|z_t)$ of observing x_t given the hidden state z_t is controlled by the parameter Φ. In practical implementations $p(x_t|z_t)$ can be learned by an ANN and Φ then represents the parameters of this ANN. This combination of HMM and ANN is called hybrid approach.

The joint probability is given by Equation 2.14. The parameters of a HMM are

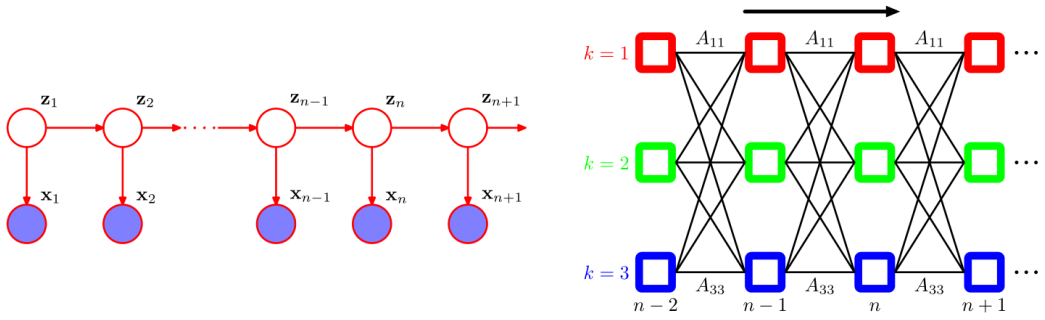

Figure 2.18: Left: Illustration of a HMM with z_i being the internal (hidden) states and x_i being the observable features. Right: Plot of the hidden states and the transition probabilities over time (state/time grid). Images taken from Bishop [Bis06].

$\Theta = \{\pi, A, \Phi\}$. Training a HMM means finding parameters Θ which maximize Equation 2.15. This is done in an iterative fashion by using the exception maximization algorithm.

$$p(X, Z|\Theta) = p(z_1|\pi) \cdot \prod_{t=2}^{T} p(z_t|z_{t-1}, A) \cdot \prod_{t=1}^{T} p(x_t|z_t, \Phi) \tag{2.14}$$

$$p(X|\Theta) = \sum_{Z} p(X, Z|\Theta) \tag{2.15}$$

A HMM model is a generative model, that means it can be used to generate new data. An example from Bishop [Bis06] to generate new data gives a good intuition of how HMMs work: the trajectories of writing the digit "2" are learned. The possible observations are lines with different orientation. The hidden states represent the different parts of the digit "2". Sampling from this trained model means starting with an initial state z_1 according to $p(z_1|\pi)$, then selecting a line segment x_1 according to $p(x_1|z_1)$, then moving to the next hidden state z_2 according to $p(z_2|z1, A)$ and so on. Some generated images are shown in Figure 2.19.

Unfolding the hidden states over time is shown in Figure 2.18 on the right. In the context of HTR, the hidden variables represent the possible characters and are therefore of interest [PF09]. The most probable path through the lattice of hidden states has to be determined. Exponentially many paths exist with respect to the number of time-steps, therefore an efficient algorithm to find the most probable path is needed. The Viterbi algorithm is able to find a solution in linear time [PF09]. It is also possible to integrate a LM into a HMM. The formula to find the most probable labeling l^* is shown in Equation 2.16. While the HMM yields the value of $p(x|l)$, the probability of seeing a specific labeling $p(l)$ can be modeled by a statistical LM, e.g. by modeling the probability of consecutive words with bigrams [PF09].

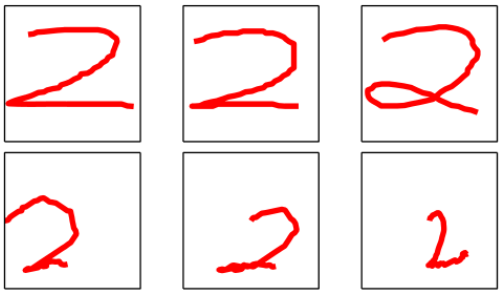

Figure 2.19: HMMs are generative models. Six instances of the digit "2" are created by randomly taking paths trough the state/time grid according to the transition probabilities. Image taken from Bishop [Bis06].

$$l^* = argmax_l(p(l|x)) = argmax_l \frac{p(l) \cdot p(x|l)}{p(x)} \tag{2.16}$$

As shown by Plötz and Fink [PF09], when using HMMs for HTR, a separate HMM is created for each possible character. Multiple character HMMs are used in parallel to account for all possible characters at this location. A series of such parallel character models is used to recognize a complete line of text. An illustration for such a HMM to recognize a text line can be seen in Figure 2.20 [PF09].

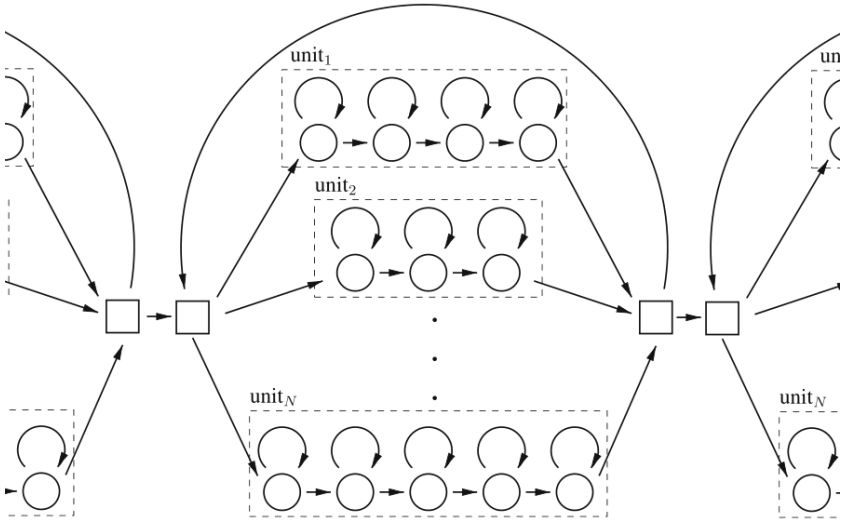

Figure 2.20: In the domain of HTR each character is modeled by a separate HMM. Each end-state of a character HMM is connected to all the begin-states of character HMMs. Decoding a HMM is done by searching the most probable path through the state/time grid. Image taken from Plötz and Fink [PF09] with modifications.

A popular HTR method before the advent of the CTC was the hybrid HMM and ANN approach [ECGZ11]. Instead of using a Gaussian mixture model for the emission probabilities of a HMM, an ANN is used. After the preprocessing steps Boquera et al. [ECGZ11] extract a sequence of feature vectors for each text line. This is achieved by applying a sliding windows to the image and compute gray-value statistics for each possible position. The emission probabilities and a LM are then used by the Viterbi algorithm to find the most likely labeling for the input image. Boquera et al. [ECGZ11] report that character HMMs with 8 states yield the best WER on the IAM dataset.

2.3.2 Convolutional and Recurrent Neural Networks

Figure 2.21 shows the main components of an end-to-end trainable ANN for HTR as proposed by Shi et al. [SBY16]. The input to the ANN is a gray-value image containing text. A stack of convolutional layers maps the input image onto feature maps. The output of the final layer of the CNN can be regarded as a sequence of length T with F features. Information is then propagated along this sequence with a stack of RNNs. The RNNs map the sequence to another sequence of same length T, assigning probabilities to each of the C different classes. Finding the most probable labeling in this $C \times T$ sized matrix is called decoding and is done with a CTC output layer. While training, the CTC loss is used to calculate a loss value for a training batch which is then backpropagated to the output layer of the RNN. The integration of a LM can be done directly in the CTC decoding algorithm or as a postprocessing step which modifies the predicted labeling.

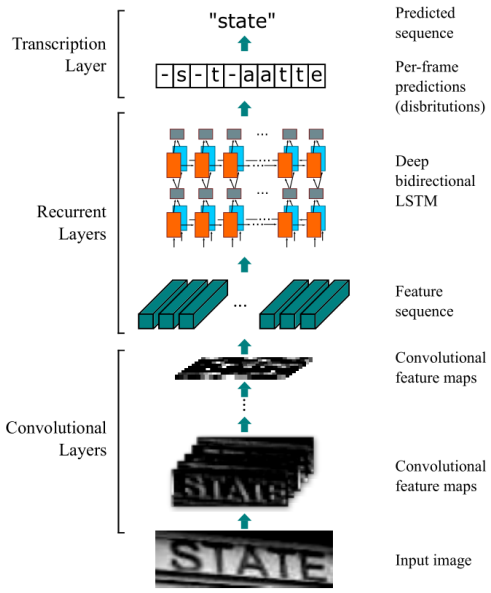

Figure 2.21: The network architecture called CRNN as proposed by Shi et al. [SBY16]. It uses CNN layers, followed by LSTM layers and a final CTC layer. Image taken from Shi et al. [SBY16].

Different modification to the described exemplary ANN architecture exist. Graves [Gra12] extracts nine (handcrafted) geometrical features from a sliding window which moves over the image. He uses a bidirectional LSTM containing 100 hidden cells. The feature sequence is processed forward and backward by two separate RNNs and the resulting sequences are then combined. A statistical LM and a dictionary containing 20000 words are used. Decoding is done with the best path algorithm to measure the CER and with the token passing algorithm for the WER.

The end-to-end trainable ANN presented by Shi et al. [SBY16] contains 7 convolutional layers and 2 layers of bidirectional LSTM units with 256 hidden cells. The input images only contain single words. Lexicon based transcription uses a dictionary to limit the possible labelings to a set of words. When e.g. using the Hunspell spell-checking dictionary which contains 50000 words, finding the most probable word means calculating the probability for each of those 50000 words, which is computationally expensive. The authors instead calculate an approximative labeling with best path decoding. Then all words close (with respect to the edit distance) to this labeling are potential candidates for which the probability is calculated. The final labeling is the word with the highest score. It should be noted that the paper describes the ANN in the context of scene text recognition, but the same model was used for transcribing medieval handwritten documents in the ICFHR 2016 competition [SRTV16].

Voigtlaender et al. [VDN16] use MDLSTM to propagate information through the input image in both dimensions. A so called building block is used which contains a convolutional layer, a MDLSTM and an average pooling layer. Different architectures created by stacking multiple building blocks are examined. They conclude that the depth of the networks plays an important role. As a preprocessing step, the input images are deslanted. It is shown that the type of parameter initialization effects the speed of convergence. Using the Glorot function outperforms a normal distribution for initialization.

The previously discussed methods work on line-level, which require preprocessing of the document pages to get the desired line-segmentation. Bluche [Blu16] proposes an attention-based approach which takes text paragraphs as input. The ANN contains MDLSTM layers as described in the paper of Voigtlaender et al. [VDN16]. However, instead of collapsing the 2D output sequence of the MDLSTM by averaging over the vertical axis, a weighting scheme is applied. The weighting scheme is learned and is able to pay attention to different lines. To account for the ground truth text given on paragraph-level, the CTC loss function is modified accordingly. The results are competitive with state-of-the-art approaches, while avoiding explicit line-segmentation as a preprocessing method. Figure 2.22 shows the results of recognizing text on paragraph-level, the implicit line-segmentation is color-coded.

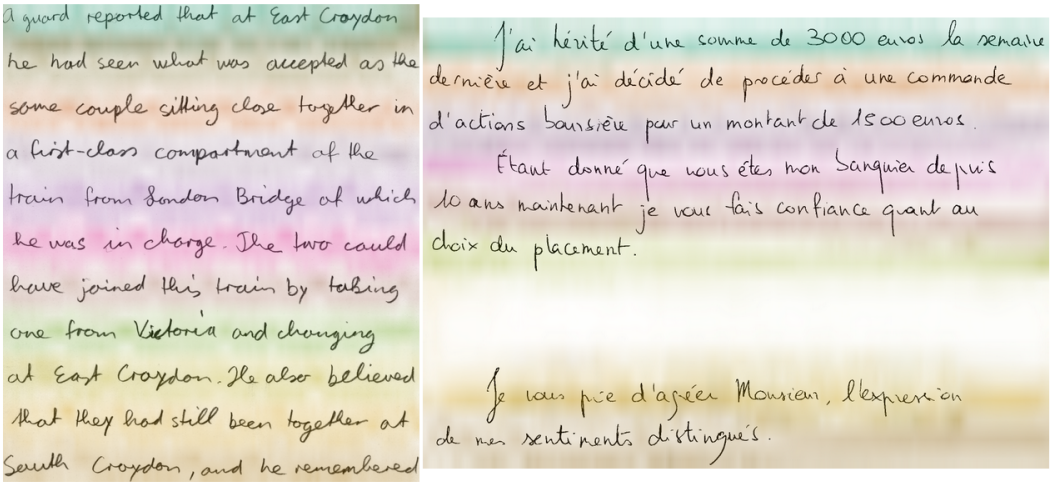

Figure 2.22: Paragraph-level text recognition. Implicit line-segmentation is color-coded. Image taken from Bluche [Blu16].

2.4 Summary

Five HTR datasets (CVL, IAM, Bentham, Ratsprotokolle and SaintGall) which are used to evaluate the proposed HTR system are presented. Besides the datasets, error measures are needed to evaluate the system. Two common choices are CER and WER, which are both calculated by normalizing the edit distance between recognized text and ground truth text, the first one on character-level, the second one on word-level. The foundations of ANNs are presented. For HTR two types of ANNs play an important role: CNNs to extract relevant features and RNNs to propagate information through the image. CTC enables training from image and labeling pairs. The usage of the CTC operation is twofold: it serves as a loss function and as a decoding function. HMMs and ANNs are the best-performing approaches according to HTR competitions.

CHAPTER 3

Methodology

The methods used for the proposed HTR system are presented in this chapter. Preprocessing includes deslanting to set the handwritten text upright and random modifications to the original images to enlarge the size of the dataset. Word segmentation can be applied to the line-images to keep the classifier small by only inputting word-images. Four published word segmentation approaches are presented and a new approach using best path decoding is proposed. An ANN is used as a classifier. To propagate information through the feature sequence different RNN types are integrated into the ANN. IDCN has so far only been applied to speech recognition, an architecture using this network layer for HTR is presented. Two state-of-the-art CTC decoding algorithms are discussed which extract the final labeling from the RNN output. Further, a new algorithm is proposed which combines the advantages of the former two. Character-level or word-level LMs can improve the recognition performance by introducing knowledge about the language in the decoding step. Finally, a text-postprocessing method is presented which corrects spelling mistakes in the final text.

3.1 Preprocessing

Even tough the HTR system presented in this chapter is end-to-end trainable, preprocessing steps can help to increase the accuracy of the system.

3.1.1 Deslanting

Vinciarelli and Luettin define slant as follows: “The slant is the angle between the vertical direction and the direction of the strokes that, in an ideal model of handwriting, are supposed to be vertical” [VL01]. An illustration of slant and slope is shown in Figure 3.1, but only slant will be discussed in the following. Deslanting is the process of transforming the handwritten text so that the slant angle gets minimized.

Figure 3.1: Slant and slope angle of handwritten cursive text. Image taken from Vinciarelli and Luettin [VL01].

The deslanting technique proposed by Vinciarelli and Luettin [VL01] builds on the hypotheses that an image containing text is deslanted if the number of image columns with continuous strokes is maximal. Algorithm 3.1 shows how this deslanting approach can be implemented. It takes a list of possible shear factors A which can be regarded as the search space of the algorithm. The input image I is mapped to a monochrome image I_{bw} by applying a threshold (e.g. by using Otsu's method [Ots79]). Afterwards a shear transform for each of the possible shear factors $\alpha \in A$ is applied to the image I_{bw}. For each sheared image I_α and each of its columns i, the distance Δy_α between the first and last foreground (black) pixel and the number of foreground pixels h_α is calculated. A continuous stroke is indicated by the equality of Δy_α and h_α. The value of h_α^2 is then added to the sum $S(\alpha)$, the reason for the square is that short strokes can be due to noise and are therefore lower weighted. The algorithm outputs the input image sheared by the factor α for which the sum $S(\alpha)$ has its maximum. An example for an input image and the deslanted output image is shown in Figure 3.2.

Algorithm 3.1: Deslanting Algorithm

Data: image I, list of shear factors A
Result: deslanted image

$I_{bw} = threshold(I)$;
$S = \{\}$;
for $\alpha \in A$ **do**
    $I_\alpha = shear(I_{bw}, \alpha)$;
    for $i \in colums(I_\alpha)$ **do**
        h_α = number of foreground pixels in column i of I_α;
        Δy_α = distance between first and last foreground pixel in column i of I_α;
        if $h_\alpha == \Delta y_\alpha$ **then**
            $S(\alpha)$ += h_α^2;
        end
    end
end
$\hat{\alpha} = argmax_\alpha(S(\alpha))$;
return $shear(I, \hat{\alpha})$;

Figure 3.2: Left: original image. Right: result of deslanting algorithm.

3.1.2 Word Segmentation

The first part introduces four explicit word segmentation approaches which use classic image processing techniques not involving ANNs. In the second part an implicit segmentation based on best path decoding is proposed.

Explicit Word Segmentation

Manmatha and Srimal [MS99] propose a scale-space approach which segments a text-line into individual words. This method has a simple implementation and is therefore chosen for the evaluation in Chapter 4.

A line-image consists of individual or connected characters. The characters representing a single word should be merged together. A way to achieve this is by transformation the image I into a blob-representation I_{blob} by convolving it with an appropriate filter kernel F (see Equation 3.3). Manmatha and Srimal use an anisotropic filter kernel F which is derived from a 2D Gaussian function G by summing up the second-order derivatives G_{xx} and G_{yy} (see Equations 3.1 and 3.2). The parameters of the kernel are its scale σ_x in horizontal direction and its scale σ_y in vertical direction. In Figure 3.3 two kernels are plotted for parameter ratios $\eta = \sigma_x/\sigma_y$ of 3 and 9. The ratio η should be set to the typical ratio between width and height of words. Manmatha and Srimal suggest values between 3 and 5. Of course the parameters can be optimized by doing a search in parameter space on a training-set (for which the properties regarding the text-style must be similar to the test-set).

$$G(x, y, \sigma_x, \sigma_y) = \frac{1}{2 \cdot \pi \cdot \sigma_x \cdot \sigma_y} \cdot e^{-(\frac{x^2}{2 \cdot \sigma_x^2} + \frac{y^2}{2 \cdot \sigma_y^2})} \tag{3.1}$$

$$F(x, y, \sigma_x, \sigma_y) = G_{xx} + G_{yy} \tag{3.2}$$

$$I_{blob} = I * F \tag{3.3}$$

Figure 3.4 shows an illustration of Manmatha and Srimal's method. First a threshold is applied to the input image. Afterwards it is convolved by the presented filter kernel. Another threshold is applied to get a binary blob image, in which each blob represents a word. A list containing the coordinates of the blob-bounding-boxes is one possibility to represent the result. Heuristics can be applied to filter out blobs which are too small or lie too much off-center. The authors test their method on 30 randomly selected pages

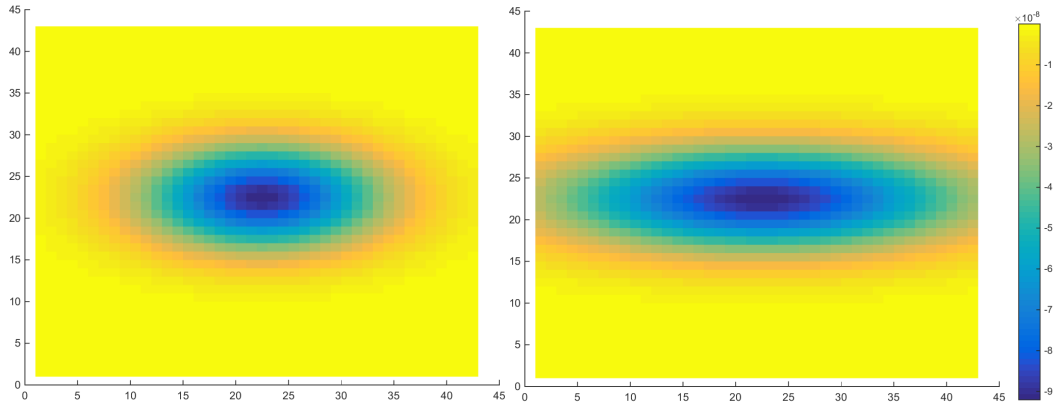

Figure 3.3: Plots of filter kernels as used by Manmatha and Srimal. Left: $\eta = 3$. Right: $\eta = 9$.

from the George Washington corpus which contains 6400 pages. 87.6% of the words are correctly segmented [MS99]. A word is said to be incorrectly segmented if it has separated bounding boxes or if more than 50% of a character in a word is not detected.

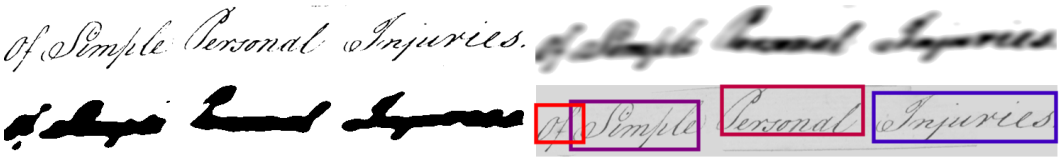

Figure 3.4: Word segmentation as proposed by Manmatha and Srimal. Top left: input image after applying threshold. Top right: image filtered by anisotropic kernel. Bottom left: blob image. Bottom right: resulting word regions.

Another word segmentation approach is proposed by Marti and Bunke [MB01]. It first calculates a binary image from the input image which yields a set of connected components. For each component c_i the distance $d(c_i, c_j)$ to all other components is calculated as follows: a line is drawn between the two centers of gravity. The length of the line not contained in the convex hull of one of the two components is then the desired distance, see Figure 3.5. A complete graph can now be created by using the components c_i as the vertices and connecting all vertices with each other by edges. The weight of each edge is the computed distance $d(c_i, c_j)$ between the two components c_i and c_j. Next, the Minimum Spanning Tree (MST) is calculated for this complete graph. A threshold is applied to the MST edge weights to decide if component pairs belong to the same word. The threshold value is calculated from a heuristic formula and must be tuned on a training-set. For evaluation the authors use a subset of the IAM dataset [MB01]. As the dataset is only annotated on line-level (at the time Marti and Bunke wrote their paper), the segmentation results are manually checked for errors. The segmentation error is 4.44%, whereof 2.36% are due to over-segmentation and 2.08% are

due to under-segmentation [MB01].

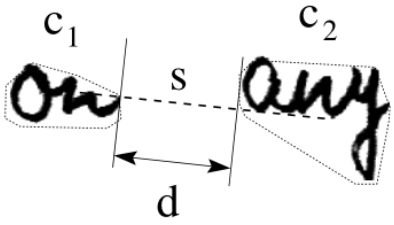

Figure 3.5: Distance calculation as used by Marti and Bunke: for each pair of connected components a line s connecting the centers of gravity is drawn. The distance d is the length of the line outside the convex hulls of the two components c_1 and c_2. Image taken from Marti and Bunke [MB01].

The distance between pairs of components is also used by Papavassiliou et al. [PSKC10]. The components are sorted by their x-coordinate. The foreground pixels of consecutive components are regarded as two different classes of a classification problem. A Support Vector Machine (SVM) searches a plane which separates those two classes. The distance is the margin between the two classes as calculated by the SVM. All distances are inserted into a histogram. The desired threshold lies between the two maxima of this histogram. The distances between pairs of components are then classified by this threshold (as in the approach of Marti and Bunke). Evaluation is done on the ICDAR07 dataset: 20 pages are used for training and 80 for testing. A segmentation accuracy of 93% is achieved [PSKC10].

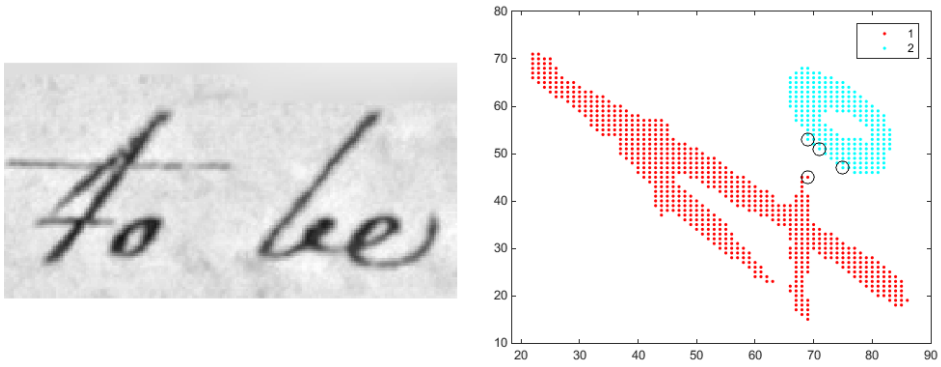

Figure 3.6: Left: input image. Right: (rotated) result of SVM. In this example, the components “t” and “o” are regarded. The pixel coordinates are assigned to one of two classes, a SVM then calculates a separating plane. The support vectors are marked by black circles.

The last approach for word segmentation described here explores different distance measures and their combination with heuristic formulas. It is proposed by Seni and Cohen [SC94] in the context of handwritten postal address recognition. The different

distance measures are shown in Figure 3.7. The first one is the bounding box measure. The distance is the horizontal distance between two boxes or zero if the boxes overlap. The second method is the Euclidean distance, which is the minimal distance between two points of consecutive components. The third and final method is the horizontal distance between two components at a given vertical position. Heuristic formulas further combine these measures to more complex ones. Again a threshold is used to classify distances to decide if two components belong to the same word. Evaluation is done on 3000 text-lines from postal address blocks. The accuracy is measured by counting how often the gap-ordering is correct, i.e. how often inter-word gaps are assigned larger values than intra-word gaps. The value of this measure is 88% for the bounding box method and 89% for the Euclidean method [SC94].

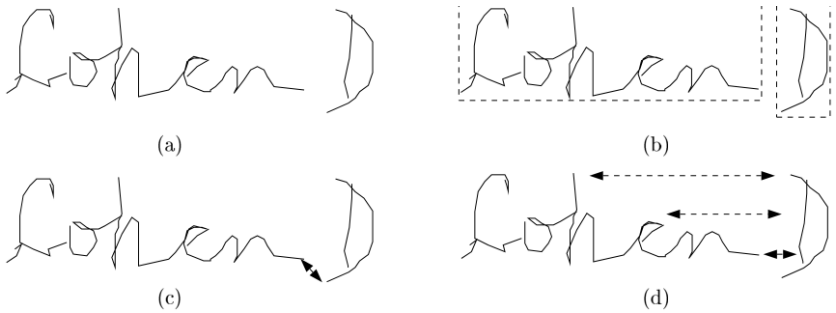

Figure 3.7: The distance measures used by Seni and Cohen [SC94]. Top left: Original image. Top right: bounding box distance. Bottom left: minimal Euclidean distance between two connected components. Bottom right: horizontal distance at different vertical positions. Image taken from Seni and Cohen [SC94].

Implicit Word Segmentation

For the proposed approach a HTR system based on ANNs trained with the CTC loss is required. The CTC best path decoding algorithm is modified to output the best path without collapsing it. Word boundaries are encoded by (multiply) whitespace characters in the best path. Four bounding boxes (see Figure 3.8) can be used to segment this path. The first bounding box (blue) simply splits the path at whitespace characters and regards the remaining subpaths as words. The second (green) and third (black) bounding box remove the blanks on the left or on the right of the subpaths. Finally, the fourth (red) one removes the blanks on both sides.

An ANN scales the input image down by a fixed factor, e.g. by the factor 8 for the HTR system presented in this chapter. It is therefore possible to calculate the position of a word in the input image by multiplying the position on the best path by the mentioned scaling factor. The remaining question is if recognized characters on the best path lie near their position in the input image. For example, the RNN could align all characters to the left, regardless of their real position, because the CTC loss works in a segmentation-free manner. This question will be investigated in Chapter 4.

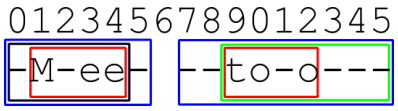

Figure 3.8: Four possible bounding boxes (red, green, black, blue) to define word boundaries on the best path. The 16 time-steps are indexed above the path.

3.2 Language Model

A LM can be used to guide the CTC decoding algorithms by incorporating information about the language structure. The language is modeled on character-level or on word-level, but the basic ideas are the same for both. For simplicity only a word-level LM is discussed. A LM is able to predict upcoming words given previous words and it is also able to assign probabilities to given sequences of words [JM14]. For example, a well trained LM should assign a higher probability to the sentence "There are ..." than to "Their are ...". A LM can be queried to give the probability $P(w|h)$ that a word sequence (history) h is followed by the word w. Such a model is trained from a text by counting how often w follows h. The probability of a sequence is then $P(h) = P(w_1) \cdot P(w_2|w_1) \cdot P(w_3|w_1, w_2) \cdot ... \cdot P(w_n|w_1, w_2, ..., w_{n-1})$ [JM14].

Of course it is not feasible to learn all possible word sequences, therefore an approximation called N-gram is used. Instead of using the complete history, only a few words from the past are used to predict the next word. N-grams with N=2 are called bigrams. Bigrams only take the last word into account, that means they approximate $P(w_n|h)$ by $P(w_n|w_{n-1})$. The probability of a sequence is then given by $P(h) = \prod_{n=2}^{|h|} P(w_n|w_{n-1})$. Another special case is the unigram LM, which does not consider the history at all but only the relative frequency of a word in the training text, i.e. $P(h) = \prod_{n=1}^{|h|} P(w_n)$.

If a word is contained in the test text but not in the training text, it is called OOV word (see Chapter 2). In this case a zero probability is assigned to the sequence, even if only one OOV word occurs. To overcome this problem *smoothing* can be applied to the N-gram distribution, more details can be found in Jurafsky and Martin [JM14].

3.3 Prefix Tree

A prefix tree or trie (from re*trie*val) is a basic tool in the domain of string processing [Bra08]. It is used in the proposed CTC decoder to constrain the search space and to query words which contain a given prefix. Figure 3.9 shows a prefix tree containing 5 words. It is a tree-datastructure and therefore consists of edges and nodes. Each edge is labeled by a character and points to the next node. A node has a word-flag which indicates if this node represents a word. A word is encoded in the tree by a path starting at the root node and following edges labeled with the corresponding characters of the word. Querying characters which follow a given prefix is easy: the node corresponding to the prefix is identified and the outgoing edge labels determine the characters which can

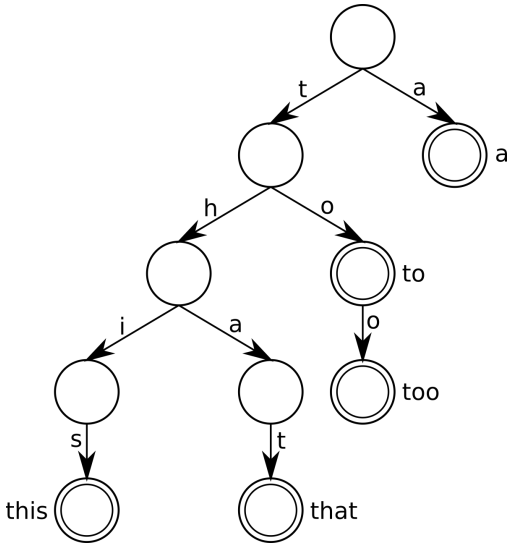

Figure 3.9: Prefix tree containing the words “a”, “to”, “too”, “this” and “that”. Double circles indicate that the word-flag is set.

follow the prefix. It is also possible to identify all words which contain a given prefix: starting from the corresponding node of the prefix, all descendant nodes are collected which have the word-flag set. As an example, the characters and words following the prefix “th” in Figure 3.9 are determined. The node is found by starting at the root node and following the edges “t” and “h”. Possible following characters are the outgoing edges “i” and “a” of this node. Words starting with the given prefix are “this” and “that”.

Aoe et al. [AMS92] show an efficient implementation of this datastructure. Finding the node for a prefix with length L needs $\mathcal{O}(L)$ time in their implementation. The time to find all words containing a given prefix depends on the number of nodes of the tree. An upper bound for the number of nodes is the number of words W times the maximum number of characters M of a word, therefore the time to find all words is $\mathcal{O}(W \cdot M)$.

3.4 Classifier

An ANN consisting of CNN and RNN layers is used as a classifier. Different ANN architectures are presented which mainly differ in the used RNN type. Three CTC decoding algorithms with integrated LM are discussed.

3.4.1 Architecture

The architecture of the proposed ANN is mainly inspired by Shi et al. [SBY16] and Voigtlaender et al. [VDN16]. Three different types of RNNs are tested. In Table 3.1 the architecture used for MDLSTM is shown, while in Table 3.2 the architecture for LSTM and ICDN is shown. The CNN layers and the mapping onto the (vector-valued) 1D

Type	Description	Output size
Input	gray-value line-image	$800 \times 64 \times 1$
Conv+Pool	kernel 5×5, pool 2×2	$400 \times 32 \times 64$
Conv	kernel 5×5	$400 \times 32 \times 128$
Conv+Pool+BN	kernel 3×3, pool 2×2	$200 \times 16 \times 128$
Conv	kernel 3×3	$200 \times 16 \times 256$
Conv	kernel 3×3	$200 \times 16 \times 256$
Conv+BN	kernel 3×3	$200 \times 16 \times 512$
Conv+Pool	kernel 3×3, pool 2×2	$100 \times 8 \times 512$
MDLSTM	bidir., 512 hidden cells	$100 \times 8 \times 512$
Mean	avg. along vert. dim.	$100 \times 1 \times 512$
Collapse	remove dimension	100×512
Project	project onto C classes	$100 \times C$
CTC	decode or loss	≤ 100

Table 3.1: Architecture for the ANN with MDLSTM. Abbreviations: average (avg), bidirectional (bidir), vertical (vert), dimension (dim), batch normalization (BN), convolutional layer (Conv).

RNN input sequence are identical for LSTM and IDCN. 7 CNN layers map the input image of size 800×64 onto a sequence of length 100 with 512 features per element. For MDLSTM the CNN layers are modified such that a 2D sequence is created with a width of 100, a height of 8 and 512 features per element. The CTC layer needs a 1D sequence (however, each sequence element itself is a vector containing the corresponding character probabilities), therefore the output of the MDLSTM is averaged along the height dimension. A linear projection is applied to map the output of the RNN onto the character distribution for each time-step (distribution over all characters contained in the regarded dataset plus one extra character to represent the blank). Finally, the CTC decoding layer outputs a sequence (text) which is at most 100 characters long.

This thesis evaluates the purely convolutional RNN type in the domain of HTR for the first time. The IDCN architecture shown by Oord et al. [ODZ+16] uses a residual block as illustrated in Figure 3.10. The input of one layer is fed through a DC and the result of two different nonlinearities is then multiplied. Skip connections add the result of this layer to an output matrix, while the residual part is fed into the next layer. The IDCN architecture used in the proposed HTR system is illustrated in Figure 3.11. The basic element is a DC layer with a given dilation width. Three layers of increasing dilation width (1, 2 and 4) are stacked and form a so called *block*. The output of each layer is fed to the next layer as well as to an intermediate output. Multiple blocks can be stacked, to avoid overfitting the kernel parameters are shared layer-wise. All the intermediate outputs are concatenated to form one (large) output matrix. For each time-step the features are projected onto the classes which serve as input for the CTC layer.

Type	Description	Output size
Input	gray-value line-image	$800 \times 64 \times 1$
Conv+Pool	kernel 5×5, pool 2×2	$400 \times 32 \times 64$
Conv+Pool	kernel 5×5, pool 1×2	$400 \times 16 \times 128$
Conv+Pool+BN	kernel 3×3, pool 2×2	$200 \times 8 \times 128$
Conv	kernel 3×3	$200 \times 8 \times 256$
Conv+Pool	kernel 3×3, pool 2×2	$100 \times 4 \times 256$
Conv+Pool+BN	kernel 3×3, pool 1×2	$100 \times 2 \times 512$
Conv+Pool	kernel 3×3, pool 1×2	$100 \times 1 \times 512$
Collapse	remove dimension	100×512
Opt. 1: LSTM	bidir., 2 stacked layers, 512 hidden cells	100×1024
Opt. 2: IDCN	2 layers of IDCN with rates 1, 2 and 4	100×3072
Project	project onto C classes	$100 \times C$
CTC	decode or loss	≤ 100

Table 3.2: Architecture for the ANN with LSTM or IDCN. Either LSTM (option 1) or IDCN (option 2) is used.

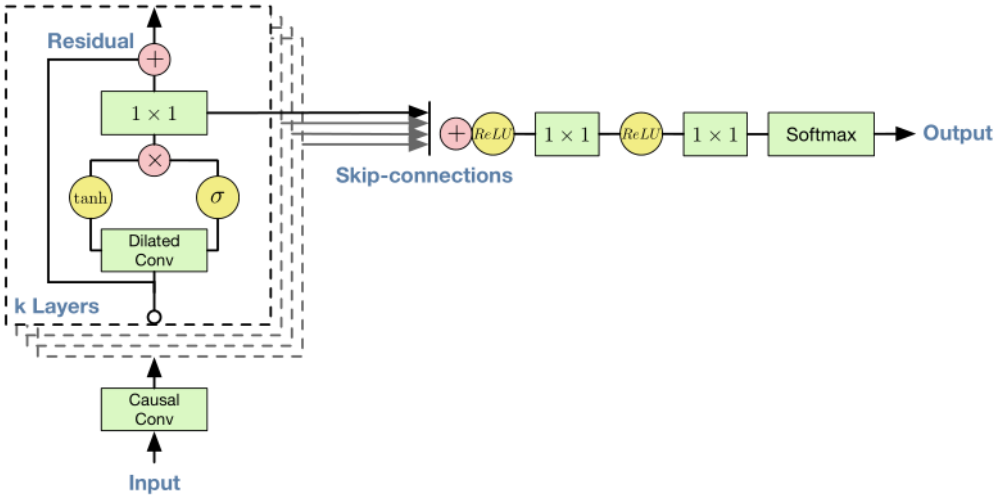

Figure 3.10: The residual block as proposed by Oord et al. [ODZ+16]. Image taken from Oord et al. [ODZ+16].

3.4.2 Training

The ANN is trained with the RMSProp algorithm which uses an adaptive learning rate [MH17]. RMSProp divides the learning rate η by an exponentially decaying sum of squared gradients avg, the parameter update is shown in Equation 3.4. In its original formulation, the suggested value for the learning rate is 0.001 and 0.9 for the decaying factor γ, however Mukkamala and Hein [MH17] give a more detailed explanation on parameter selection. An advantage over other training algorithms such as gradient descent

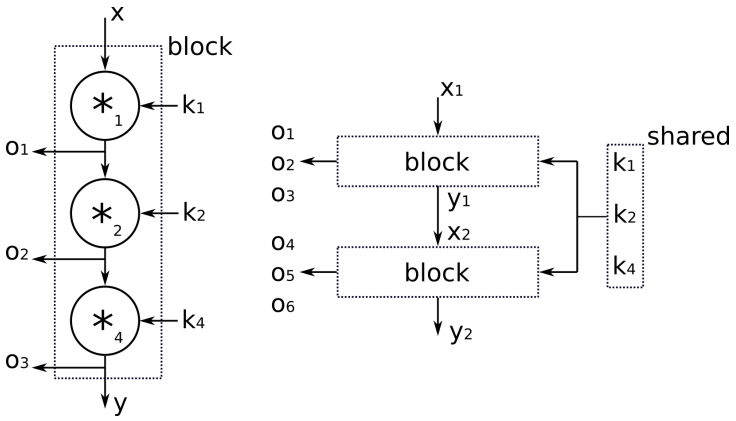

Figure 3.11: IDCN architecture of the proposed HTR system. x is the input and y the output of a block, o the intermediate output of a layer and k the parameters of a convolution kernel.

is that the learning rate is adapted automatically, only an initial value has to be set [SBY16].

$$w' = w - \frac{\eta}{\sqrt{avg + \epsilon}} \cdot \nabla_w E(w) \tag{3.4}$$

$$avg' = 0.9 \cdot avg + 0.1 \cdot (\nabla_w E(w))^2 \tag{3.5}$$

As stated by Goodfellow et al. [GBC16], training a deep ANN can be problematic because the parameter update is done under the assumption that all other layers are unchanged. This of course is not true and the effect increases the more layers are added. Batch normalization tackles this problem by normalizing a batch of input activations (for the input or any hidden layer) by subtracting the mean and dividing by the standard deviation.

While training, the error of the training-set keeps decreasing, while the error of the validation-set starts to increase again at a certain point. This is due to overfitting as described in Section 2.2.1. The early stopping algorithm as shown by Goodfellow et al. [GBC16] is used to prevent overfitting (see Algorithm 3.2). The patience parameter p denotes the number of training epochs the algorithm continues after the epoch with the best result achieved so far on the validation-set. If no better result can be found after p trials training terminates.

Algorithm 3.2: Early Stopping

Data: patience p
Result: optimal parameters w^* of trained ANN

```
i = 0;
v* = ∞;
while i < p do
    train the ANN;
    update w;
    v = ValidationSetError(w);
    if v < v* then
        v* = v;
        w* = w;
        i = 0;
    end
    else
        i = i + 1;
    end
end
return w*
```

3.4.3 CTC Decoding with Language Model

The algorithm presented first is token passing and works on word-level. It looks for the most probable sequence of dictionary words and is guided by a word-level LM. The second algorithm is Vanilla Beam Search (VBS) decoding which keeps track of the most promising labelings while iteratively going through all time-steps. A character-level LM helps to consider only labelings with a likely character sequence. Finally, a new algorithm called Word Beam Search (WBS) decoding is proposed, which combines the advantages of the former two.

CTC Token Passing

This algorithm works on word-level. It takes a dictionary containing all words to consider and a word-bigram LM as input. The output word sequence is constrained to the dictionary-words.

The token passing algorithm is first described by Young et al. [YRT89] in the domain of HMMs and the following discussion is based on this publication. A single word is modeled as a state machine. Each state represents a character of the word. The state machine can either stay in a state or continue to the next state. Going from state i to state j is penalized by p_{ij} and a state at time-step t is penalized by $d_j(t)$. Aligning a word model to a feature sequence means finding the best path through the state machine. The cost for a possible alignment is $S(i_0, i_1, ..., i_T) = \sum_i (p_i + d_i(t))$. Finding the best

path through the states can be formulated as a token passing algorithm:

- Calculate the cost (score) for each start state and put the value into a token.
- Move all tokens one time-step ahead, incrementing the cost value by p_{ij} and $d_j(t)$.
- Clear the old tokens and then go over all states which hold a (new) token and take the best one for each state. This token represents the minimal cost path to this state.
- If more than one word should be considered multiple word models are put in parallel. The information in the token has to be extended to save the sequence (history) of words the path goes through. The transition from one word to another can be penalized by a word-bigram LM.

For CTC decoding, the original algorithm has to be changed to account for the blank label [Gra12] as shown in Algorithm 3.3. An illustration is given in Figure 3.12. The states of a word model are denoted by s in the pseudo-code, where $s = 0$ is the start state and $s = -1$ is the final state of a word. The extended word w' is the original word w extended by a blank label at the beginning, at the end and in between all characters. For example, "Hello" gets extended to "-H-e-l-l-o-". A token for segment s of word w at time-step t is denoted by $tok(w, s, t)$ and holds the score and the word-history. Time and words are 1-indexed, e.g. $w(1)$ denotes the first character of a word. The algorithm has a time complexity of $\mathcal{O}(T \cdot W^2)$, where T denotes the length of the output sequence of the RNN and W the number of words contained in the dictionary [Gra12].

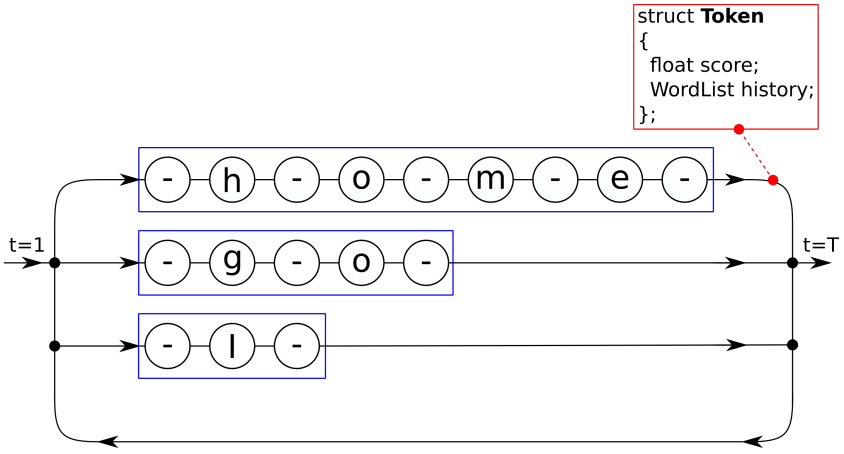

Figure 3.12: Three word models (blue) are put in parallel. Information flow is implemented by a set of tokens (red, only one shown) which are passed through the states and between the words.

Algorithm 3.3: CTC Token Passing

Data: RNN output matrix mat, dictionary W and word-bigram LM $p(w|\hat{w})$
Result: most probable word sequence

$tok(w, s, 1) = (0, \{\}) \quad \forall w, \forall s$;
for $w \in W$ **do**
  $tok(w, 1, 1) = Token(mat(blank, 1), \{w\})$;
  $tok(w, 2, 1) = Token(mat(w(1), 1), \{w\})$;
  if $|w| == 1$ **then**
    $tok(w, -1, 1) = tok(w, 2, 1)$;
  end
end
for $t = 2...T$ **do**
  for $w \in W$ **do**
    $w^* = argmax_{\hat{w}}(tok(\hat{w}, -1, t-1).score \cdot p(w|\hat{w}))$;
    $tok(w, 0, t).score = tok(w^*, -1, t-1).score \cdot p(w|w^*)$;
    $tok(w, 0, t).history = tok(w^*, -1, t-1).history + w$;
    $w' = extend(w)$;
    for $s = 1...length(w')$ **do**
      $P = \{tok(w, s, t-1), tok(w, s-1, t-1)\}$;
      if $w'(s)$!= $blank \quad and \quad s > 2 \quad and \quad w'(s-2)$!= $w'(s)$ **then**
        $P = P \cup tok(w, s-2, t-1)$;
      end
      $tok(w, s, t) = bestToken(P)$;
      $tok(w, s, t).score \mathrel{*}= mat(w'(s), t)$;
    end
    $tok(w, -1, t) = bestToken(\{tok(w, |w'|, t), tok(w, |w'|-1, t)\})$;
  end
end
$w^* = argmax_{\hat{w}}(tok(\hat{w}, -1, T).score)$;
return $w^*.history$;

CTC Vanilla Beam Search

The algorithm described here is a simplified version of the one published by Hwang and Sung [HS16]. The depth pruning is removed because the number of time-steps is limited for offline HTR. At each time-step a labeling can be extended by a new character from the set of possible characters. This creates a tree of labelings which grows exponentially in time. To keep the algorithm feasible only the best labelings at each time-step are kept and serve as input to the next time-step. A character is encoded by a path which contains one or more instances of the character and is optionally followed by one or more blanks. Repeated characters in a labeling are encoded by a path which separates the repeated characters by at least one blank. It is easy to see that different paths can encode

the same labeling, e.g. $\mathcal{B}$(“a -”)=$\mathcal{B}$(“a a”)=“a”. Therefore, the probabilities of all paths yielding the same labeling must be summed up to get the probability of that labeling. A sketch outlining the functioning of the algorithm is shown in Figure 3.13.

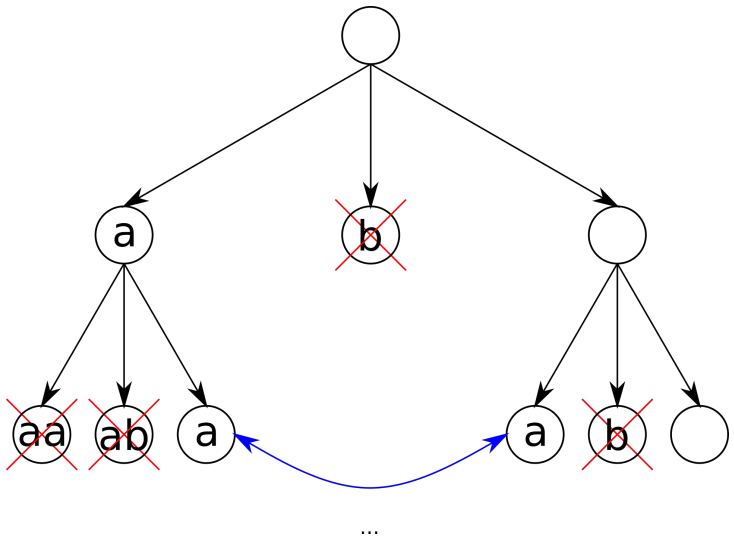

Figure 3.13: Iterative construction of beams (from top to bottom). Starting from the most recent time-step, each beam is copied to the next time-step. Further, each beam gets extended by all possible characters (“a”, “b”) and these new beams are also added to the next time-step. The tree of labelings grows exponentially in time. To keep beam search decoding feasible, beams with low score get removed (red). Beams representing the same labeling are merged together (blue).

From what was just discussed a simple beam search decoder can be constructed. This algorithm and the one from the next section share the basic structure, therefore pseudo-code is shown in Algorithm 3.4 for both of them and the differences are discussed in the text. The set B holds the beams of the current time-step, and P holds the probabilities for the beams. P_b is the probability that the paths of a beam end with a blank, P_{nb} that they end with a non-blank, and P_{txt} is the probability assigned by the LM. P_{tot} is an abbreviation for $P_b + P_{nb}$. The algorithm iterates from $t = 1$ through $t = T$ and creates a tree of beam-labelings as shown in Figure 3.13. An empty beam is denoted by $\varnothing$ and the last character of a beam is indexed by -1. The best beams are calculated by the function $bestBeams(B, BW)$ by sorting the beams B with regard to $P_{tot} \cdot P_{txt}$ and only keep the BW best ones (BW denotes the beam width). For each of the beams, the probability of seeing the beam-labeling at the current time-step is calculated. Separately book-keeping for paths ending with a blank and paths ending with a non-blank accounts for the CTC coding scheme. Each beam b is extended by every possible character c in the alphabet C. The function $nextChars(b)$ ignores its parameter b in the case of VBS and always returns the complete alphabet C. A character-bigram LM $p(c|b)^\gamma$ is applied inside of the function $scoreBeam(LM, b, c)$ which calculates the probability of seeing the

last character in b and the new character c next to each other. Its influence is controlled by the parameter γ. The beams are normalized with respect to their length: Graves and Jaitly [GJ14] do this only once for the beams B after the last iteration of the outermost loop. The $completeBeams(B)$ function is not applied in VBS, i.e. in this case it is the identity function and simply returns its argument B. To avoid numerical problems the algorithm can make use of log-probability. The time complexity is not given by Hwang and Sung [HS16], however it can be derived from the pseudo-code. At each time-step, the beams are sorted according to their score. In the previous time-step, each of the BW beams is extended by C characters, therefore $BW \cdot C$ beams have to be sorted which accounts for $\mathcal{O}(BW \cdot C \cdot log(BW \cdot C))$. As this sorting happens for each of the T time-steps, the overall time-complexity is $\mathcal{O}(T \cdot BW \cdot C \cdot log(BW \cdot C))$. The two inner loops are ignored because they only account for $\mathcal{O}(BW \cdot C)$.

CTC Word Beam Search

The motivation to propose the WBS decoding algorithm is twofold:

- VBS works on character-level and does not constrain its beams (text candidates) to dictionary words.
- The running time of token passing depends quadratically on the dictionary size [GLF+09], which is not feasible for large dictionaries as shown in Section 4.2.5. Further, the algorithm does not handle non-word character strings. Punctuation-marks and large numbers occur in the datasets, however, putting all possible combinations of these into the dictionary would enlarge it unnecessarily.

WBS decoding is a modification of VBS and has the following properties:

- Words are constrained to dictionary words.
- Any number of non-word characters is allowed between words.
- A word-level bigram LM can optionally be integrated.
- Better running time than token passing (regarding time-complexity and real running time on a computer).

The dictionary words are added to a prefix tree which is used to query possible next characters and words given a word prefix. Each beam of WBS is in one of two states. If a beam gets extended by a word-character (typically “a”, “b”, ...), then the beam is in the word-state, otherwise it is in the non-word-state. Figure 3.14 shows the two beam-states and the transitions. A beam is extended by a set of characters which depends on the beam-state. If the beam is in the non-word-state, the beam-labeling can be extended by all possible non-word-characters (typically “ ”, “.”, ...). Furthermore, it can be extended by each character which occurs as the first character of a word. These

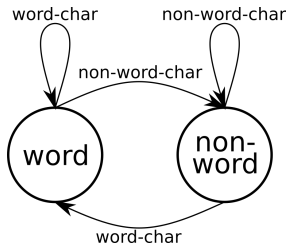

Figure 3.14: A beam can be in one of two states.

characters are retrieved from the edges which leave the root node of the prefix tree (see Section 3.3). If the beam is in word-state, the prefix tree is queried for a list of possible next characters. Figure 3.15 shows an example of a beam currently in the word-state. The last word-characters form the prefix "th", the corresponding node in the prefix tree is found by following the edges "t" and "h". The outgoing edges of this node determine the next characters, which are "i" and "a" in this example. In addition, if the current prefix represents a complete word (e.g. "to"), then the next characters also include all non-word-characters.

Optionally, a word-level LM can be integrated. A bigram LM is assumed in the following text. The more words a beam contains, the more often it gets scored by the LM. To account for this, the score gets normalized by the number of words. Four possible LM scoring-modes exist and names are assigned to them which are used throughout the thesis:

- Words: only a dictionary but no LM is used.
- N-grams: each time a beam makes a transition from the word-state to the non-word-state, the beam-labeling gets scored by the LM.
- N-grams + Forecast: each time a beam is extended by a word-character, all possible next words are queried from the prefix tree. Figure 3.15 shows an example for the prefix "th" which can be extended to the words "this" and "that". All beam-extensions by possible next words are scored by the LM and the scores are summed up. This scoring scheme can be regarded as a LM forecast.
- N-grams + Forecast + Sample: in the worst case, all words of the dictionary have to be taken into account for the forecast. To limit the number of possible next words, these are randomly sampled before calculating the LM score. The sum of the scores must be corrected to account for the sampling process.

Algorithm 3.4 shows the pseudo-code for WBS decoding. The basic structure of the algorithm is discussed in the previous section. Instead of extending each beam by every character from the alphabet, the set of possible next characters is calculated from the current beam-state and beam-labeling (as already explained). When extending a beam,

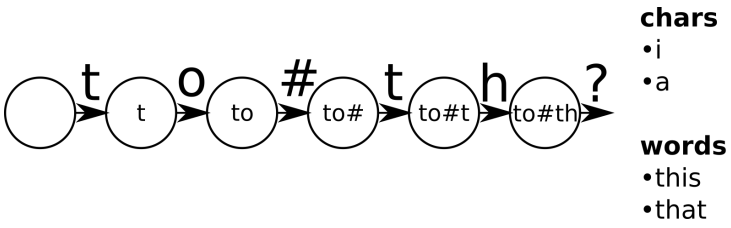

Figure 3.15: A beam currently in the word-state. The "#" character represents a non-word character. The current prefix "th" can be extended by "i' and "a" and can be extended to form the words "this" and "that". The prefix tree from Figure 3.9 is assumed in this example.

the LM calculates a score P_{txt} depending on the scoring-mode. In each iteration the LM score is normalized by taking P_{txt} to the power of $1/numWords(b)$, where $numWords(b)$ is the number of words contained in the beam b. After the algorithm finished its iteration through time, the beam-labelings get completed if necessary: if a beam-labeling ends with a prefix not representing a complete word, the prefix tree is queried to give a list of possible words which contain the prefix. Two different ways to implement the completion exist: either the beam-labeling is extended by the most likely word (according to the LM), or the beam-labeling is only completed if the list of possible words contains exactly one entry.

The time-complexity depends on the scoring-mode used. If no LM is used, the only difference to VBS is to query the next possible characters. This takes $\mathcal{O}(M \cdot C)$ time, where M is the maximum length of a word and C is the number of unique characters. The overall running time therefore is $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + M))$. If a LM is used, then a lookup in a unigram and/or bigram table is performed when extending a beam. Searching such a table takes $\mathcal{O}(log(W))$. This sums to the overall running time of $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + M + log(W)))$. In the case of LM forecasting, the next words have to be searched which takes $\mathcal{O}(M \cdot W)$. The LM is queried S times, where S is the size of the word sample. If no sampling is used, then $S = W$. The running time sums to $\mathcal{O}(T \cdot BW \cdot C \cdot (log(BW \cdot C) + S \cdot log(W) + W \cdot M))$.

Algorithm 3.4: Word Beam Search

Data: RNN output matrix mat, BW and LM
Result: most probable labeling

$B = \{\varnothing\}$;
$P_b(\varnothing, 0) = 1$;
for $t = 1...T$ **do**
  $bestBeams = bestBeams(B, BW)$;
  $B = \{\}$;
  for $b \in bestBeams$ **do**
    if b != $\varnothing$ **then**
      $P_{nb}(b, t) \mathrel{+}= P_{nb}(b, t-1) \cdot mat(b(-1), t)$;
    end
    $P_b(b, t) \mathrel{+}= P_{tot}(b, t-1) \cdot mat(blank, t)$;
    $B = B \cup b$;
    $nextChars = nextChars(b)$;
    for $c \in nextChars$ **do**
      $b' = b + c$;
      $P_{txt}(b') = scoreBeam(LM, b, c)$;
      if $b(t) == c$ **then**
        $P_{nb}(b', t) \mathrel{+}= mat(c, t) \cdot P_b(b, t-1)$;
      else
        $P_{nb}(b', t) \mathrel{+}= mat(c, t) \cdot P_{tot}(b, t-1)$;
      end
      $B = B \cup b'$;
    end
  end
end
$B = completeBeams(B)$;
return $bestBeams(B, 1)$;

3.5 Postprocessing

The text recognized by the classifier can be further processed to correct spelling mistakes. This postprocessing has no access to the optical model (RNN output) and therefore works solely on text-level. Tong and Evans [TE96] propose a text correction system which uses a dictionary. The dictionary is created from the training text and holds a list of unique words and their occurrence frequency. First the algorithm (see Algorithm 3.5) splits the recognized text into a list of words. For each word not contained in the dictionary a set of suggestions is created. This is done by edit operations which are applied to the original word: characters are inserted, deleted and substituted. As an example a subset of the suggestions for the misspelled word “Hxllo” are: “Hcllo”, “Hdllo” and “Hello”. The probability of executing a given edit operation for a character is learned

while training. To give an example, substituting the characters “a” and “o” happens more often than substituting “x” and “o” because the former ones look more similar. The overall probability $score(s.text)$ of a suggestion s is the product of the edit operation probability $s.prob$ and the word frequency $P(s.text)$. Edit operations yielding non-words (like “Hdllo” in the example above) are removed in this step because these words have zero probability. The best scored suggestion for each misspelled word is selected and the words are concatenated to form the corrected text line.

The text correction is trained in parallel to the classifier. The probability of an edit operation (e.g. substituting “o” for “a”) is increased each time this edit operation is able to correct a word.

Algorithm 3.5: Text correction

Data: recognised text line, dictionary with word frequency $P(w)$
Result: corrected text line

```
words = split(line);
for w ∈ words do
    corr = w;
    if P(w) == 0 then
        suggestions = editOps(w);
        score = {};
        for s ∈ suggestions do
            if P(s.text) · s.prob > 0 then
                score(s.text) = P(s.text) · s.prob;
            end
        end
        corr = argmax_s(score(s));
    end
    corrWords.append(corr);
end
return concatenate(corrWords);
```

3.6 Summary

This chapter presented the relevant parts of the proposed HTR system. An algorithm is shown which removes the slant angle of handwritten text by searching the sheared image with maximum continuous vertical lines in it. Either complete line-images or word-images can serve as input for the classifier. In the former case, no segmentation is needed for the task of HTR. However, the HTR system can be used for word-segmentation by adapting the best path decoding algorithm. In the latter case, the classifier can be kept small. Four word segmentation approaches are presented which make use of different distance measures to classify gaps between word candidates. Regarding the classifier itself, three

architectures are proposed which make use of different RNN types. Multiple decoding algorithms exist which optionally can integrate a LM. The presented text-postprocessing method searches for the most likely correction of a misspelled word and also takes the character confusion probabilities of the HTR system into account, i.e. it learns the weaknesses of the system.

CHAPTER 4

Experiments and Results

The proposed HTR system is evaluated with the five datasets presented in Chapter 2. Experiments are conducted to analyze the influence of preprocessing methods, different RNN types, CTC decoding algorithms and text-postprocessing. Afterwards, an overview of the results and a comparison to other systems is given.

4.1 Setup

For all experiments except word segmentation, CER and WER are used to measure the accuracy of the HTR system. If results are shown in a table, CER and WER are usually given in the format CER/WER. The hardware used for the evaluation is shown in Table 4.1 which especially concerns the running times presented in this chapter. TensorFlow is used as a deep-learning framework while OpenCV, OpenCL and Matlab are used for image processing. The datasets are put in LMDB databases in a CRNN-compatible fashion.

For IAM, Bentham and Ratsprotokolle the official split of the dataset into training, validation and test-set is available and therefore used for the evaluation. For CVL and SaintGall no official split (for HTR tasks) exists. Therefore those datasets are randomly split into subsets for training, validation and testing. The number of pages per subsets

Parameter	**Value**
CPU	Intel Core i5-4690, 4 cores, 3.5GHz
RAM	16GB
GPU	Nvidia GTX 980, 2048 cores, 1126MHz, 4GB
OS	Ubuntu 16.04

Table 4.1: Hardware.

Dataset	Training-set	Validation-set	Test-set
CVL	1364	80	160
SaintGall	45	6	9

Table 4.2: Split of CVL and SaintGall dataset into the subsets for training, validation and testing. The table shows the number of randomly chosen pages per subset.

is fixed, see Table 4.2. For each dataset 3 different random assignments are evaluated and the mean of the individual results is taken as the final result. In the following, the random assignments are referenced by appending "0", "1" or "2" to the dataset name, e.g. "CVL0" denotes the first random assignment for the CVL dataset.

RMSProp is used as an optimizer with a learning rate of 0.001. Also other optimizers have been tested and especially Adam and Adadelta show good performance too. The batch size is set to 25. The larger the batch size, the smoother the loss curves, which makes optimization easier. On the other hand, training noise (due to a small batch size) can help to escape from local minima of the loss function. The batch size is also limited by the GPU memory. Figure 4.1 shows the plot of the loss functions for batch sizes 5 and 25, the difference regarding the smoothness can easily be seen.

4.2 Experiments

The first experiments analyze optional data preprocessing. Afterwards, the influence of different RNN types is examined. The hyperparameter selection for IDCN is presented to give an idea which parts mainly influence the result. The decoding algorithms and the text-postprocessing are able to improve the accuracy on noisy data by incorporating language information and are tested on two datasets with different language properties.

4.2.1 Image Preprocessing and Data Augmentation

All dataset images are converted to gray-value images to decrease the size of the database. Keeping the input size of the ANN constant simplifies the implementation of the classifier, however, images from a dataset vary in size. The simplest method as used by Shi et al. [SBY16] is to scale the input image to the desired input size which yields distorted images. Instead of resizing, the image can be downsized (horizontally and vertically by the same scaling factor) if needed and then copied into an image of the desired input size. The parts not covered by the copied image are filled with a background color. This background color is calculated by taking the gray-value with the highest bin in the image histogram.

Data augmentation happens on-the-fly when the classifier asks for a new training sample and therefore does not increase the database size. As already described, the training image is copied to the input image to avoid distortion. A simple type of data augmentation is

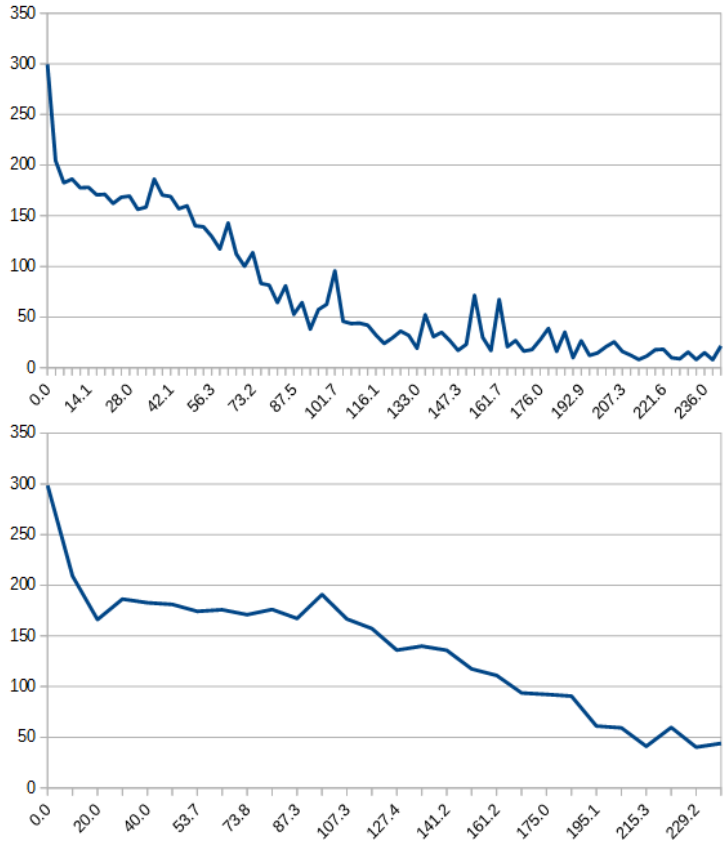

Figure 4.1: Plot of loss functions for the first 4 minutes of training on the SaintGall dataset. Horizontal axis represents the time in seconds, vertical axis represents the CTC loss value. Top: batch size is 5. Bottom: batch size is 25.

to copy the image to a random position. Another modification to the training image is doing random resizing (by a factor between 3/4 and 4/3, independent for both axis).

Table 4.3 shows the results when applying different preprocessing methods to the Bentham dataset. The first row shows a baseline result for the discussion with contrast normalization, no stretching and randomization of position and size. It can be seen that stretching the images to the full input size decreases the recognition performance the most. Shi et al. [SBY16] use this technique for words, for which size does not differ as much as for complete text-lines. Therefore stretching can not be recommended when working on line-level. Further it can be seen that even a simple preprocessing method like contrast normalization improves the CER by more than 1% from 6.76% to 5.72%. This operation subtracts the mean and divides by the standard deviation of the gray-value distribution for each image. Data augmentation decreases the CER by around 0.5% and the WER by more than 1%.

Setting	Result
Baseline	5.72/16.74
No contrast normalization	6.76/20.08
Stretch	8.17/22.18
No randomization	6.20/17.95

Table 4.3: Different preprocessing settings and the resulting CER/WER.

	No deslanting	CPU	GPU
Absolute	9.2ms	21.7ms	12.3ms
Relative	100%	236%	134%

Table 4.4: Average time to read, process and write one sample from the Bentham dataset while creating the database. The running times are shown when deslanting is disabled and when deslanting is done on either the CPU or the GPU.

4.2.2 Deslanting

Deslanting is a computational expensive algorithm as multiple sheared versions of the original image have to be calculated. The implementation has a search space of 9 different shear factors in the range from -1 to 1. The algorithm is implemented with OpenCV for the CPU and with OpenCL for the GPU. As can be seen in Table 4.4 the running time per sample is reduced from $21.7ms$ to $12.3ms$ when moving the computation from the CPU to the GPU. This performance improvement is achieved by parallelizing the loops of the algorithm and doing the summation in a final reduction step. The algorithm is optionally applied when creating a LMDB database.

The accuracy of the HTR system is given for the Bentham and Ratsprotokolle dataset, see Table 4.5. For Bentham the results get improved by deslanting the images, while for Ratsprotokolle the results get worse. Figure 4.2 gives an idea why deslanting only helps with Bentham. Bentham is written in a cursive fashion throughout the dataset. In the original images, one time-frame is often occupied by two characters, but after deslanting it is (approximately) only occupied by one character at most. This makes the recognition problem easier, because the RNN has to disentangle less characters by propagating information through the sequence. Deslanting the Ratsprotokolle dataset does not help because even though the text is cursive, single time-frames still often contain more than one character at a time. Another explanation for the decreasing accuracy is that the variability of the input data is decreased by the deslanting algorithm. Therefore deslanting can be seen as an antagonist to data augmentation. There is no general answer if deslanting helps for a given dataset. It has to be tested for each dataset separately.

Dataset	Original	Deslanted
Bentham	5.72/16.74	5.41/16.39
Ratsprotokolle	7.99/27.61	8.28/28.33

Table 4.5: Comparison (CER/WER) of original and deslanted datasets. Evaluation is done with MDLSTM and best path decoding.

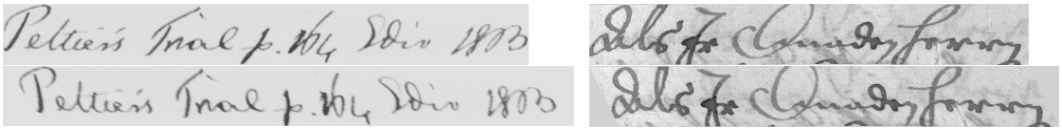

Figure 4.2: Top: original images. Bottom: deslanted images. Left: Bentham dataset. Right: Ratsprotokolle dataset.

4.2.3 Word Segmentation

First a qualitative comparison between the four explicit methods and the implicit method is given. Afterwards Manmatha and Srimal's method and the implicit method are compared from a quantitative point of view.

Qualitative Results

The explicit word segmentation methods described in Section 3.1.2 all exploit the distance of neighboring word candidates to distinguish between inter-word-gaps and intra-word-gaps. The approach described by Manmatha and Srimal [MS99] uses an anisotropic filter kernel to create a blob image. The blob image greatly depends on the height/width ratio of the filter kernel which must be set manually or must be tuned on a training-set. The optimal parameters vary between different documents, writers and even pages. Figure 4.3 shows the results for two different parameter settings. The image either gets split into too few or too many words. This is an issue that can be seen for many of the tested line-images because the inter-word distance greatly differs between different word instances. However, for the correctly segmented words the calculated bounding boxes tightly surround the word.

Marti and Bunke's [MB01] approach builds a graph from the blob image. A MST is calculated and the edge weights of this tree get classified. Figure 4.4 shows the MST and the result for a sample from the Bentham dataset. The problem of over- and undersegmentation still exists as with Manmatha and Srimal's approach. When looking at the resulting segmentation the problem of splitting "gentleman" into "g" and "entleman" is typical for this approach: even though it is clear to a human viewer that this split is incorrect, it is not obvious to an algorithm which only knows about the distance between two blobs and the distance distribution of all tree edges. The other methods (Papavassiliou et al. [PSKC10], Seni and Cohen [SC94]) differ in the way they calculate the distance between blobs, but the segmentation problems are similar.

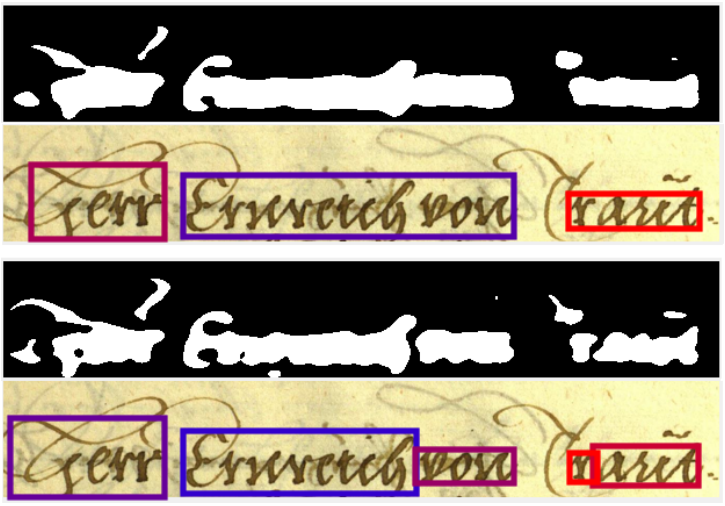

Figure 4.3: Two samples from the Ratsprotokolle dataset which are incorrectly segmented with Manmatha and Srimal's method with different parameters. Top: filter kernel with size 21×3. Bottom: filter kernel with size 11×3.

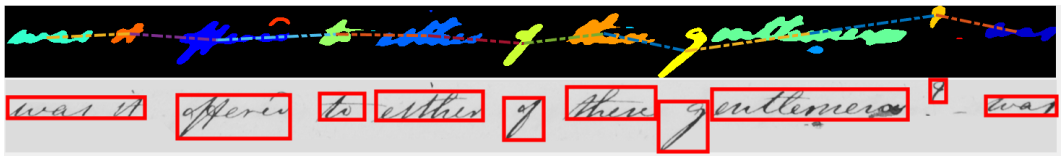

Figure 4.4: Method described by Marti and Bunke applied to a sample from Bentham dataset. Top: different blobs have different colors and represent vertices of a graph. Small blobs are ignored. Line segments represent edges and form a MST connecting the vertices. The edge weights get classified. Bottom: resulting word boundaries. The word "gentleman" is split into "g" and "entleman", while "was it" was recognized as a single word. The question sign "?" is classified as a word on its own.

All mentioned methods classify inter-word gaps only with help of a distance measure. This works if words are separated by large gaps, while characters of a single word are tightly coupled. Historical datasets such as the Ratsprotokolle dataset with its cursive writing style are hard to segment if only relying on this single feature. Another problem is that if a single word is incorrectly segmented, the assignment of the ground truth word labels to the segmented word images is not possible any more. It can be concluded that these word segmentation methods which do not take advantage of learning are not well suited for HTR in historical documents.

Instead of segmenting the words as a preprocessing step, it is possible to let the classifier segment the words implicitly by learning a whitespace character. Complete line-images containing multiple words are used as input to the classifier. This of course increases the input size, training time and memory consumption. The HTR system uses an image of

size 128×32 for single word recognition, while the size is increased to 800×64 for text-line recognition. This increases the number of input pixels by a factor of 12.5. Additionally to learning the characters of words, a whitespace (inter-word) character must be learned. One question is how the whitespace character interferes with the blank label (which has a special meaning for CTC): Figure 4.5 shows the input image and the probability of the whitespace and blank label for each time-step. It can be seen that the whitespace learned is in fact a word separation label, i.e. it is only learned in between two words. The white area after the last word in the image gets encoded by blanks. Even between words the whitespace labels only appear as sharp spikes, the rest is filled by blanks. The blank and whitespace label therefore do not interfere in a negative way. This means that the ANN is able to implicitly segment words.

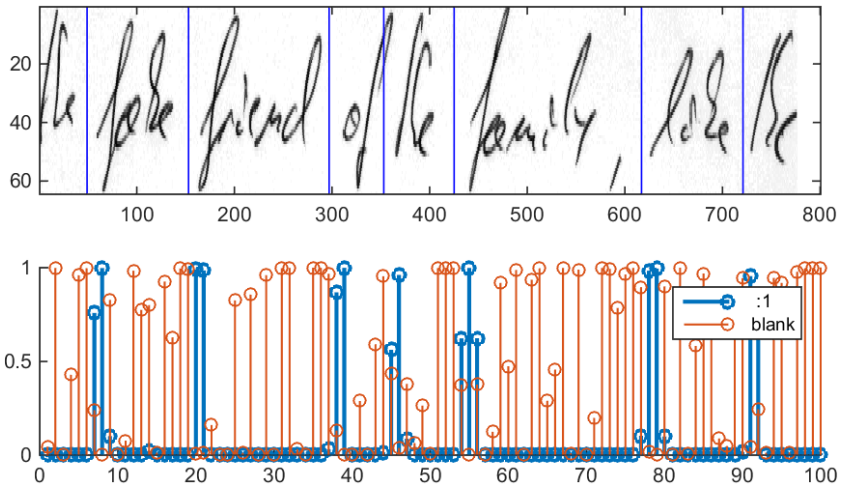

Figure 4.5: Top: word segmentation derived from the learned whitespace label. Bottom: distribution of the whitespace label (blue) and the blank label (orange) over time in the RNN output.

Quantitative Results

A detailed quantitative comparison is only given for the method of Manmatha and Srimal [MS99] and the proposed implicit segmentation of the ANN. The error measures are defined as follows: the first one is the number of lines which are segmented into the correct number of words and the second one is the average absolute distance between ground truth word beginning (or end) and recognized word beginning (or end). Of course these distances can only be calculated if the number of segmented words is correct, as otherwise the mapping between ground truth and recognized words is ambiguous. The distances measured in pixels are summed up and divided by the number of words. Only the horizontal begin and end positions are taken into account because the ANN does not output a vertical segmentation.

Bentham is used as an evaluation dataset as it contains historical handwritten text (in

Method	Correct word-count	Avg. error left	Avg. error right
Manmatha 9×3	34%	18px	21px
Manmatha 11×3	47%	18px	21px
Manmatha 15×3	47%	25px	26px
MDLSTM “blue”	91%	15px	48px
MDLSTM “green”	91%	18px	48px
MDLSTM “black”	91%	15px	31px
MDLSTM “red”	91%	18px	31px
IDCN “blue”	88%	10px	56px
IDCN “green”	88%	12px	56px
IDCN “black”	88%	10px	38px
IDCN “red”	88%	12px	38px

Table 4.6: Comparison of methods for word segmentation. The colors refer to the bounding boxes presented in Section 3.1.2. Evaluation is done on a random sample of size 100 from the Bentham dataset. Average line-width is $1506px$ and average word-width is $172px$.

contrast to CVL and IAM) and is partly annotated on word-level (only CVL and IAM are also annotated on word-level). A random set of 100 line-images is used. These line-images contain 872 words in total. The average width of a line is $1506px$ while it is $172px$ for a word. Manmatha and Srimal's method is evaluated for three kernel sizes: 9×3, 11×3 and 15×3, where the first number is the width and the second one is the height. Further heuristics are needed to account for small blobs in the image: blobs with an area less than $400px$ are ignored. Also blobs placed too much off-center (i.e. at the bottom or the top) are ignored. For the proposed implicit segmentation the four bounding boxes (blue, green, black an red, see Section 3.1.2) calculated for the best path through the RNN output are investigated.

Results are shown in Table 4.6. It can be seen that Manmatha and Srimal's method only succeeds to segment 47% of the lines into the correct number of words. But for the lines segmented into the correct number of words the segmentation positions have an average error of only around $20px$ on both sides. This is around 10% of the average word width. The ANN segmentation is able to correctly predict the number of words for 91% of the lines. Two types of RNNs are tested, namely MDLSTM and IDCN. MDLSTM achieves slightly better results. The segmentation error depends on the bounding box used. Best results are achieved for the black one, which trims the blank labels only on the right side of each subpath. Using an untrimmed path as a baseline, trimming on the left side increases the error by 20%, while on the right side it decreases the error by up to 45%.

It can be concluded that the implicit segmentation learned by the ANN outperforms a classic image processing approach. As already stated, the CTC loss function trains the ANN in a segmentation-free manner, even though a trained ANN is able to achieve good segmentation results. The problem with Manmatha and Srimal's method is that it is not

Dataset	MDLSTM	LSTM	IDCN
Bentham Validation, $p = 5$	6.01/18.23	6.53/18.44	6.17/18.31
Bentham Test, $p = 5$	6.17/18.41	6.60/19.02	6.05/17.71
Bentham Validation, $p = 10$	5.72/17.05	5.95/17.39	6.19/18.47
Bentham Test, $p = 10$	5.91/16.74	6.02/17.25	6.12/18.03

Table 4.7: Comparison (CER/WER) of the RNN types on the Bentham dataset. The first experiment is executed with (early stopping) patience $p = 5$ while the second experiment is executed with patience $p = 10$.

able to predict the correct number of words for more than 50% of the lines. However, if the number of words is correctly predicted, the segmentation error is approximately as good as the one from the ANN approach. When creating a training database, the number of words per line is known in advance. It is therefore possible to pass the correct number of words to the segmentation algorithm and automatically tune the parameters for each line until the recognized number equals the correct number. When applying this to the Bentham dataset, the correct word-count can be increased to 66%, but this increases the runtime of the algorithm too as a grid search on the parameters must be performed for each sample.

4.2.4 RNN Types

Three types of RNNs are tested: LSTM, MDLSTM and IDCN. As this is the first use of IDCN in the domain of HTR, different variations are evaluated and discussed.

Comparison

The results for LSTM, MDLSTM and IDCN on the Bentham dataset are compared in Table 4.7. The model is trained with two different patience settings for the early stopping algorithm. Best path decoding is used for all experiments to avoid the influence of a LM. MDLSTM shows the lowest CER for the validation-set, but IDCN performs best on the test-set with patience 5. For patience 10, MDLSTM also outperforms the other RNNs on the test-set. Interestingly, IDCN performs better on the test-sets than on the validation-sets. Further, the lower of the two patience settings achieves better results for IDCN. This suggests that this RNN type is more sensitive to overfitting than (MD)LSTM. From the results on the validation-set it seems promising to use MDLSTM if enough data is available to avoid overfitting. IDCN has the advantage that a complete HTR system can be built using only CNN layers. This makes it possible to use deep learning frameworks which do not feature RNNs.

When comparing the output of (MD)LSTM and IDCN layers for the same image, it can be seen that there are differences. If a written character fills multiple time-steps in the image, this can be encoded by either repeating this label or by concatenating (multiple) blank labels. This coding scheme is described in more detail in Section 2.2.3. (MD)LSTM

	MDLSTM	LSTM	IDCN
Absolute	8546	9094	6458
Relative	132%	141%	100%

Table 4.8: Comparison of how RNNs encode labelings. The Bentham test-set is used. The number of blank labels between the first and last non-blank label on the best path are counted for the complete test-set.

encodes a label by a peak followed by blanks, while IDCN encodes a label by repeating the label. To give an example, (MD)LSTM encodes "a" as "a - - -" while IDCN encodes it as "a a a a". This is not a strict rule but a tendency towards one of these behaviors. An example is shown in Figure 4.6. To prove this observation, the blank labels between the first and last non-blank label on the best path are counted. All other blanks are ignored because they are used to encode empty image space. Results are shown in Table 4.8. As can be seen, LSTM uses 41% more blanks than IDCN.

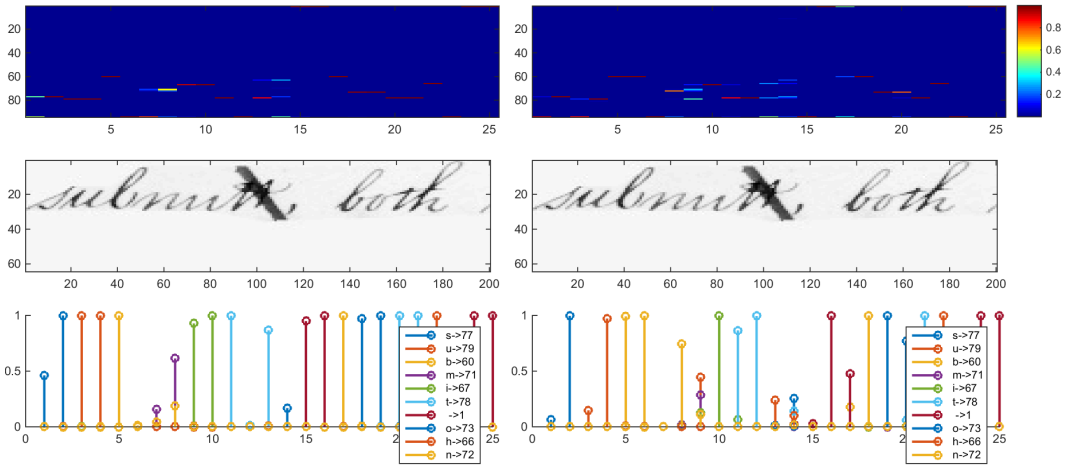

Figure 4.6: Difference of how MDLSTM and IDCN encode labels. Left: IDCN with 6 repeated labels. Right: MDLSTM with 4 repeated labels. Top: output of RNN, horizontal axis is time axis, vertical axis is class label axis. Probability of seeing a label at a given time-step is color-coded. Middle: input image. Bottom: probability of seeing the labels "s", "u", "b", "m", "i", "t", "o" "h" and " " over time.

Experiments with IDCN

The evaluation of the following experiments in done on word images from the IAM dataset and the accuracy is given by the number of correctly classified words divided by the total number of words. As a starting point the RNN is replaced by a fully-connected ANN (FCN) which gives an accuracy of 37%. Six layers of the residual block as used in WaveNet give an accuracy of 53%. Concatenating the results from each layer into one final feature map gives better results than the summation of those as used in WaveNet

[ODZ+16]. Also using 1×1 convolutions to project the output to the classes increases the accuracy and decreases the number of parameters when compared to a FCN. To speed-up training the building block is simplified to only have one DC and one non-linearity without changing the accuracy. Parameter sharing between the layers is a way to avoid overfitting and increases the accuracy to 55%. Separate parameters can be learned for each feature. This further increases the accuracy to 67%. Setting the dilation factor to 1, 2 and 4 for each block and using 2 blocks in total gives the best result of 75% accuracy. Changing the kernel size (without holes) from 3 to 5 does not change the accuracy which suggests that enough temporal context is already available. Batch normalization is an important ingredient to achieve convergence when training the model.

4.2.5 CTC Decoding Algorithms

The differences between the algorithms are first analyzed using one sample from the IAM dataset. Afterwards, the algorithms are evaluated on the CVL and SaintGall dataset to give a quantitative comparison.

Qualitative Results

In Figure 4.7 the output of a RNN is shown for a given input. It has 100 time-steps and 80 classes, with the 80^{th} class being the blank label. The other 79 non-blank classes are (grouped and ordered): “!"#&’()*+,-./”, “0123456789”, “:;?”, “ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz”. These classes correspond to the complete set of characters contained in the IAM dataset. The classifier is not fully trained and the label predictions are inaccurate, which makes it easier to see the different behavior of the CTC decoding algorithms on noisy data. The probability of seeing a label at a given time is color-coded and sums to 1 for each time-step. The probability of the labels “o”, “f” and blank is shown over time. It can be seen that most of the time the blank label has a high probability, while the other labels appear as sharply located spikes which often only occupy one time step. This is the reason why best path decoding works well even tough it is just an approximation: one single path often is much more likely than all other paths. Another observation is that the spikes coincide with the locations of the corresponding characters in the image. Therefore CTC can be used to segment a text (as already discussed) into words and characters, despite the fact that CTC is a segmentation-free approach. Figure 4.8 shows the result of segmenting the text into characters using best path decoding.

Decoding the presented RNN output with best path decoding, both beam search decoders and token passing yields different results. The ground truth text is “the fake friend of the family, like the”. VBS, WBS and token passing use a LM for decoding which is built from the ground truth word sequence. Results are shown in Table 4.9. The error measure is the edit distance, i.e. the number of insert, delete and substitute operations to transform the recognized text into the ground truth text. Both beam search decoders and token passing achieve an edit distance of 3 and therefore outperform the other algorithms. The

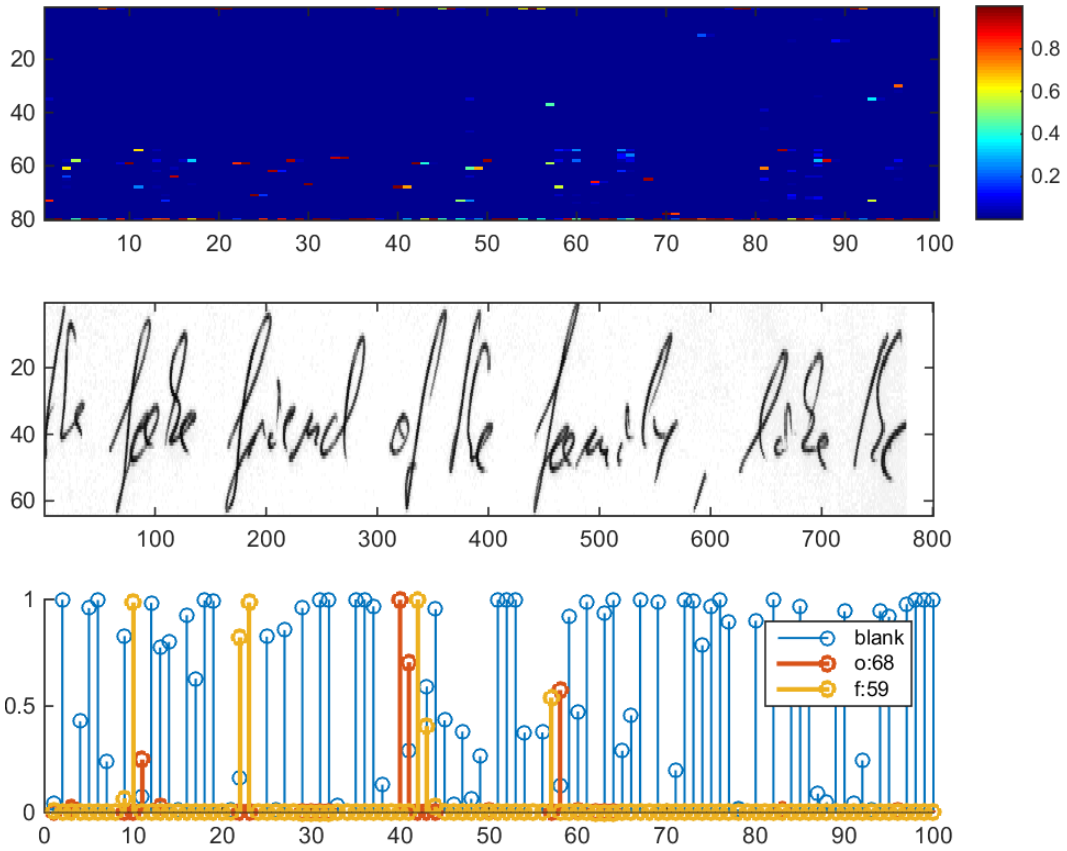

Figure 4.7: Top: output of RNN layer. The horizontal axis corresponds to time while the vertical axis corresponds to the labels with the last label representing the blank label. Middle: the input image. Bottom: probability of the labels "o", "f" and the blank label over time.

Figure 4.8: Segmentation into characters using best path decoding. The ANN is not yet fully trained and therefore makes some mistakes.

weighting parameter γ for the LM in VBS is crucial as can be seen when the LM is disabled ($\gamma = 0$) or when the LM has the same influence as the optical model ($\gamma = 1$).

Quantitative Results

Tables 4.10 and 4.11 show the quantitative results for the decoding algorithms on the CVL and SaintGall dataset (only using random assignment 0 in both cases). These two datasets are chosen because CVL has the lowest OOV word rate of 1.91% while

Type	Text	Edit distance
Ground Truth	“the fake friend of the family, like the”	0
Best Path	“the fak friend of the fomly hae tC”	9
VBS $\gamma = 0$	“the fak friend of the fomcly hae tC”	9
VBS $\gamma = 0.1$	“the fake friend of the family lie th”	3
VBS $\gamma = 1$	“the f friend of ly l”	19
WBS: *Words* mode	“the fake friend of the family fake the”	3
Token Passing	“the fake friend of the family fake the”	3

Table 4.9: Different CTC decoding algorithms are applied to the same RNN output. The influence of the LM in the VBS algorithm is governed by the γ parameter as shown in Section 3.4.3.

Algorithm	No postprocessing	Postprocessing
Best Path	2.20/6.97	1.79/3.88
Prefix Search	2.15/6.60	1.74/3.62
VBS	1.95/5.69	1.62/3.14
WBS: *Words* mode	1.15/1.78	1.15/1.78

Table 4.10: Results (CER/WER) for the presented decoding algorithms with and without text-postprocessing. Evaluation is done on the CVL0 dataset, the RNN type is IDCN.

Algorithm	No postprocessing	Postprocessing
Best Path	4.27/21.66	6.19/32.32
Prefix Search	4.10/20.77	6.04/31.84
VBS	3.95/20.23	6.03/31.54
WBS: *Words* mode	13.12/63.22	13.12/63.22

Table 4.11: Results (CER/WER) for the presented decoding algorithms with and without text-postprocessing. Evaluation is done on the SaintGall0 dataset, the RNN type is IDCN.

SaintGall has the highest rate of 55.45%. This rate influences the accuracy of LMs and text-postprocessing (which will be discussed in the next section). All algorithms except token passing are tested: for a fair comparison, the same text corpus is used to train the LMs of the decoders, however, it is not feasible to use token passing because of the size of the resulting dictionary. To give an example, the running time of token passing increases from $570ms$ to $2120ms$ per sample when increasing the number of dictionary words from 1280 to 4610. However, the mentioned implementation already uses an approximation as described by Graves et al. [Gra12] to reduce the time complexity to $\mathcal{O}(T \cdot W \cdot log(W))$. ICDN is used as a RNN in the ANN. The following list compares best path decoding, prefix search decoding, VBS and WBS:

γ	CER	WER	CER+WER
1.0	5.0	30.1	35.1
0.5	3.8	23.8	27.6
0.25	3.3	21.1	24.4
0.175	3.3	20.9	24.2
0.125	3.2	20.5	23.7
0.1	3.2	20.4	23.6
0.075	3.2	20.8	24
0.05	3.2	20.6	23.8
0.0	3.2	21.2	24.4

Table 4.12: Tuning the hyperparameter γ of VBS for the SaintGall dataset. The sum of CER and WER is optimized. Results for tuning BW between 50 and 100 are not shown. Best parameters are $\gamma = 0.1$ and $BW = 50$.

- Best path decoding: it is an approximation and therefore is in last place for CVL and in second-but-last place for SaintGall.
- Prefix search decoding: it performs better than best path decoding but worse than VBS which is guided by a character-level LM.
- VBS: the results show that a character-level LM is robust against a large number of OOV words as it outperforms the other decoders on the SaintGall dataset.
- WBS: if most of the words to be decoded are contained in the dictionary, as it is in the case of CVL, then this algorithm outperforms all others. However, for SaintGall with its large number of OOV words the algorithm performs worst.

VBS has two hyperparameters: the beam width BW and the influence of the language model γ. Those hyperparameters must be tuned on the validation-set. This is shown in Table 4.12 for the SaintGall dataset. For this dataset the best hyperparameters are $\gamma = 0.1$ and $BW = 50$. The parameter BW must be small enough to yield good running time (which grows according to $\mathcal{O}(BW \cdot log(BW))$ with respect to BW) but big enough to get high accuracy. For the other datasets (CVL, IAM, Bentham and Ratsprotokolle) the influence of the LM must be decreased to $\gamma = 0.01$ to get optimal results.

The running times for recognizing one sample by the ANN are shown in Table 4.13. It is mainly influenced by the CTC decoding algorithm in use. TensorFlow's default implementations are used for best path and prefix search decoding. The others are implemented in C++ and are added to TensorFlow as custom operations. Time measurement is done for single-threaded implementations, splitting the batch elements into 4 parts and decoding these in parallel decreases the running time by 20%. For the beam search decoding algorithms it is crucial to make the inner loop fast by using data structures with low access time to the list of labelings and by avoiding memory allocations when

Type	Time per sample
Best Path (TF)	30ms
Prefix Search (TF)	60ms
VBS (own)	290ms
WBS: *Words* mode (own)	200ms
WBS: *N-grams* mode (own)	300ms

Table 4.13: Time to recognize a sample from the Bentham dataset for different CTC decoding algorithms. BW is set to 50 for both beam search decoders. The text in brackets specifies if the TensorFlow (TF) implementation or an own implementation is used.

concatenating the current labeling b and the new label c. Replacing a tree based search (average time complexity $\mathcal{O}(log(N))$ with N denoting the number of elements) by a hash based search ($\mathcal{O}(1)$) decreases the running time by 25% for this setup.

Each algorithm has its advantages and disadvantages. If it is important to recognize the text very fast, but accuracy is not that important, then best path decoding is well suited. This algorithm can also be used when the characters are recognized with high confidence (sharp label peaks in RNN output), because then the probability distribution over paths is sharply peaked at the best path. Prefix search decoding always outperforms best path decoding, but is a bit slower (factor 2 for this HTR system). If a character-level LM is available, beam search can be used to avoid unlikely labelings. The recommended use case for both WBS and token passing is when a dictionary with low OOV word rate is available. However, while the practical usage of token passing is limited by its high running time, WBS can be used even for large dictionaries.

4.2.6 Postprocessing

Text-postprocessing is evaluated using the same experimental setup as used for the decoding algorithms. The results are shown in Table 4.10 for CVL and in Table 4.11 for SaintGall. CVL has a low number of OOV words (1.91%, see Table 2.1), therefore the postprocessing method is able to enhance the CER and WER for all decoding algorithms except WBS. As WBS constraints its recognized words to dictionary words, there are no spelling mistakes left to be corrected in the text. This suggests that there is no profit in using WBS and text-postprocessing together. The same applies for token passing as it also constrains its output. The result of best path decoding is improved from 2.20% to 1.79% for the CER and from 6.97% to 3.88% for the WER. This shows that best path decoding in combination with postprocessing yields a fast and accurate recognition system. Also the result of beam search decoding is improved, as the character-level LM is orthogonal to the postprocessing method which works on word-level.

SaintGall is a dataset with many OOV words (55.45%), therefore text-postprocessing is not able to improve the accuracy. If this method encounters an OOV word (which is

Ranking	Insert	Delete	Substitute
1	r	e	l → t
2	d	r	b → t
3	c	n	s → r
4	e	o	n → u
5	s	s	i → e

Table 4.14: Ranking of most frequent characters in insert, delete and substitute operation for Bentham dataset.

unknown but correct), it replaces it by a word from the dictionary and thereby introduces a new error in the final text. As can be seen in Table 4.11, using text-postprocessing even worsens the results, e.g. from 4.27% to 6.19% for CER and from 21.66% to 32.32% for WER when using best path decoding. Again, the result of WBS is not changed if postprocessed.

An experiment using the Bentham dataset and the MDLSTM architecture is conducted to analyze the learned probability distributions. The implemented method only corrects words (i.e. no punctuation signs or numbers). The Bentham dataset contains 93 labels whereof 58 of them are characters forming words. Only these characters are used in postprocessing. When training has finished, the learned probabilities for the edit operations can be analyzed. Figure 4.9 shows the probabilities for insert and delete operations for each character, Figure 4.10 shows the probabilities for substituting one character by another and Table 4.14 shows the characters with highest frequency for all three types of edit operations. Insert and delete operations use total probabilities, therefore the most frequent characters in English text also tend to have high scores for the edit operations. Future work can try to condition these two operations on the surrounding characters. The substitution operation is already conditioned on the recognized character.

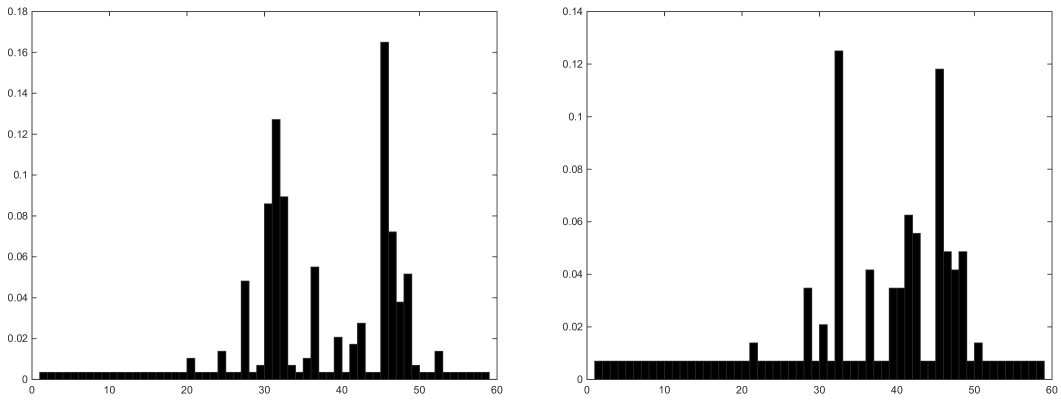

Figure 4.9: Left: frequency of insertions for each character from Bentham dataset. Right: the same for deletions.

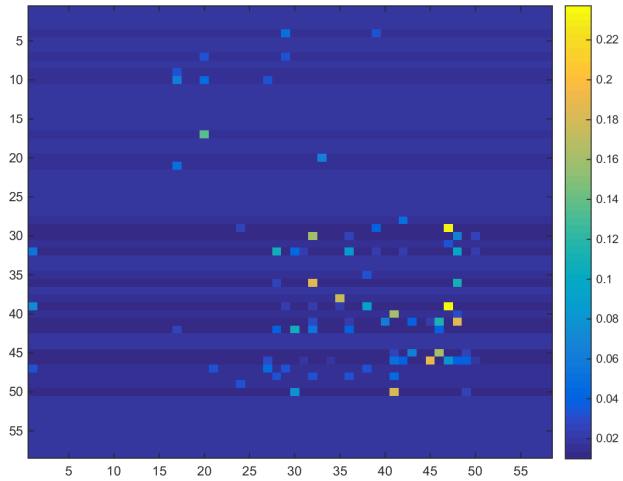

Figure 4.10: Frequency of substituting one character by another (confusion matrix) for Bentham dataset.

Setting	Result
No postproc.	6.20/17.95
Postproc.	5.99/15.32
Postproc. $P_{op} = 1$	6.11/15.64

Table 4.15: Postprocessing: disabled, enabled and enabled but with uniform operation distribution.

The probability distributions for edit operations are sharply peaked around a few characters. To show that the learned distributions are an important ingredient of the algorithm, three experiments are applied on the Bentham dataset. The first one has postprocessing disabled. The second one has postprocessing enabled. Finally, the third one has postprocessing enabled, but uniform distributions are used instead of the learned distributions. Best path decoding is used as a CTC decoding algorithm for all experiments. Results are shown in Table 4.15. It can be seen that just taking the word probability into account decreases the CER from 6.20% to 6.11%. Additionally using the edit operation probability further decreases the CER to 5.99%.

4.3 Main Results

Only a subset of all possible hyperparameter combinations is evaluated. Line-images are fed into the classifier and no explicit word segmentation takes place. Contrast normalization and data augmentation are enabled. The MDLSTM architecture is chosen because it performs best as shown by the conducted experiments. All presented decoding algorithms are evaluated, however postprocessing is just applied to prefix search and VBS. The LM weighting parameter γ of VBS is set to 0.01. The Bentham and Ratsprotokolle

datasets are tested in two versions: the original ones and the deslanted ones. To avoid overfitting training is stopped by the early stopping algorithm with patience 5 for IAM and CVL and patience 10 for all others. The concatenated ground-truth text of training and validation-set is used to train the LM of VBS and WBS. The text from the test-set is used to train the LM of token passing because this algorithm has a high running time regarding the dictionary size and therefore is only feasible for the small dictionary created from the test-set. Further, the test-set is also used for WBS such that a comparison between both algorithms is possible. All other settings can be found in Section 4.1.

The results of the evaluation are shown in Table 4.16. As already shown for best path decoding, deslanting improves the results for Bentham but worsens them for Ratsprotokolle. The same holds true for all other algorithms except token passing, for which the trend is the other way round. The accuracy of WBS depends on the quality of the dictionary. When training the LM with the test-set, WBS always outperforms the other algorithms regarding the CER. This is the setup which can be compared to token passing, because of the same LM source. *Words* mode (i.e. only using a dictionary) is used for WBS while token passing additionally uses N-gram probabilities, which explains why token passing outperforms WBS regarding WER 4 out of 7 times. When training the LM of WBS with the concatenated text of training and validation-set, this algorithm performs worst for all datasets except CVL. As already analyzed, a dictionary with a low OOV word rate must be available when using WBS and the same holds true for token passing too. Adding a large number of words (e.g. from a word-list of the corresponding language) to the dictionary decreases the OOV rate and therefore increases the decoding accuracy. The text-postprocessing improves the accuracy for CVL, Bentham (deslanted) and Ratsprotokolle (original and deslanted). Again, the high OOV word rate explains why postprocessing fails for SaintGall.

Dataset	VBSP	VBS	WBSTe	WBSTr	PrefixP	Prefix	Path	Token
CVL	1.54/2.68	1.76/4.53	**1.07**/1.45	1.11/1.68	1.59/2.80	1.85/4.86	1.94/5.22	1.35/**1.32**
IAM	8.95/25.01	8.63/28.30	**4.86**/**10.15**	9.77/27.20	8.98/25.24	8.71/28.80	8.94/29.47	10.04/11.72
Bentham	5.55/14.37	5.55/16.18	**4.15**/**8.00**	6.16/16.88	5.60/14.37	5.55/16.26	5.72/16.74	8.24/9.34
BenthamD	5.23/13.45	5.32/16.10	**3.73**/**7.50**	5.80/16.48	5.21/13.49	5.31/16.22	5.41/16.39	8.48/9.82
Ratsprotokolle	7.31/24.24	7.08/25.32	**4.00**/8.31	8.41/29.94	7.30/24.04	7.19/25.75	7.35/26.29	7.73/**7.88**
RatsprotokolleD	7.78/25.55	7.80/28.38	**4.11**/8.76	8.88/31.33	7.92/26.12	8.11/29.83	8.27/30.34	7.32/**7.51**
SaintGall	5.09/27.79	2.81/15.05	**2.01**/2.15	12.30/62.17	5.01/27.38	2.78/14.59	2.82/14.96	3.00/**1.63**

Table 4.16: This table gives an overview of the results for the datasets. Results are given in the format CER/WER. Abbreviations: VBSP: VBS with character LM and postprocessing, VBS: VBS with character LM, WBSTe: WBS with word LM trained from test-set, WBSTr: WBS with word LM trained from training-set and validation-set, PrefixP: prefix search with postprocessing, Prefix: prefix search, Path: best path decoding, Token: token passing with word LM trained from test-set. The BenthamD and RatsprotokolleD are the deslanted versions of these two datasets.

Method	Result
Own method	8.95/25.01
Graves	18.2/25.9

Table 4.17: Comparison of published results for IAM dataset.

Method	Result
CITlab	5.0/14.6
Own method	5.23/13.45
LIMSI	5.5/15.0

Table 4.18: Comparison of published results for Bentham dataset.

Method	Result
RWTH	4.79/20.94
BYU	5.05/21.08
PRHLT-char-lm	5.15/20.64
A2IA	5.43/22.13
PRHLT-no-lm	5.91/24.11
Own method	7.31/24.24
LITIS	7.32/26.04
ParisTech	18.53/46.59

Table 4.19: Comparison of published results (truncated to 2 decimal places) for Ratsprotokolle dataset.

4.4 Comparison to other HTR systems

HTR results from other authors are published for IAM, Bentham and Ratsprotokolle. The results achieved under VBS with postprocessing enabled are used for the comparison. For Bentham the result of the deslanted dataset is taken because it performs better than the original dataset. The comparison is given in Tables 4.17, 4.18 and 4.19. Table rows are sorted by CER. The results from other authors are taken from Sánchez et al. [SRTV14], Sánchez et al. [SRTV16] and Graves [Gra12].

For IAM the WER is about the same as reported by Graves [Gra12]. Interestingly the CER of Graves is much worse. This can be due to the decoder in use: Graves applies token passing to the RNN output, which achieves good results for the WER but not for the CER. A similar behavior regarding CER and WER can also be seen in Table 4.16 for the proposed HTR system. For Bentham, the CER is in between the other two methods, but the WER is better than those methods. The reason is that postprocessing improves the WER for VBS from 16.10% to 13.45%. Bentham has a low OOV word rate of 17.63% which enables the postprocessing to correct many spelling mistakes. Finally, for the Ratsprotokolle dataset CER and WER are on the 6^{th} out of 8 places. These

results show that the proposed HTR system is able to achieve state-of-the-art results.

4.5 Summary

This chapter presented the results for the proposed HTR system. Multiple experiments are conducted to analyze the influence of different parts of the system. Contrast normalization increases the accuracy, the same applies for data augmentation via random image transformations. When removing the slant angle of the text, the results are improved for one of the two evaluated datasets. Segmenting a text-line into words shows poor performance for the tested preprocessing methods. Using the trained HTR system instead, it is possible to segment most of the words correctly. MDLSTM outperforms LSTM and IDCN. However, even if IDCN shows a slightly worse CER than MDLSTM, it makes it possible to build a purely-convolutional ANN for HTR. Choosing the right decoding algorithm for an already trained ANN can improve the accuracy by incorporating information about the language. Text-postprocessing corrects spelling mistakes and learns the character confusion probabilities of the HTR system. It is able to improve the results if the number of OOV words is low. A comparison to published results from other authors shows that the proposed system is able to achieve state-of-the-art performance.

CHAPTER 5

Conclusion

This thesis has four main contributions. First, a purely convolutional replacement for the RNN is proposed. This architecture uses IDCN and has the potential to make HTR available on more systems and frameworks which do not support RNNs but only CNNs. Next, a method to segment a text-line into words using best path decoding is suggested. This method outperforms other methods which rely on classic image processing without exploiting learning techniques. Further, a decoding algorithm is proposed which uses a dictionary and a LM to constrain the recognized words while allowing arbitrary non-word characters between them to account for numbers or punctuation marks. Finally, different settings, architectures, preprocessing and postprocessing methods are analyzed regarding their influence on the recognition accuracy.

5.1 Summary

Five datasets are used to evaluate the proposed system. Those datasets include two with modern handwriting, one from around the year 1800, one from the late middle-ages and one from around the year 900. Results from other authors are available for three datasets and are used to compare the proposed HTR system to other systems.

The preprocessing methods can be divided into those simplifying the problem for the classifier and into those increasing the size of the dataset. Contrast normalization increases the recognition performance for all datasets, for the Bentham dataset the CER is improved by 1%. Deslanting enhances the results for Bentham (CER decreased by 0.3%) but not for Ratsprotokolle (CER increased by 0.9%). To further improve the accuracy the size of the datasets can be artificially increased by random image transformations. This decreases the CER by 0.5% for Bentham. The IAM dataset is fully annotated on word-level, all other datasets only contain annotation on line-level. This suggests two solutions to tackle the problem: either a line-image is segmented into words or the complete text-line is fed into the classifier which additionally has to

learn a word-separation label (i.e. whitespace character). Experiments show that the analyzed word segmentation methods perform poor on historical datasets: segmentation methods are compared by counting the number of lines for which the segmentation yields the correct number of words. The explicit method is able to achieve 47% on this task, while the implicit method achieves 91%. As a by-product of these experiments a word segmentation method using best path decoding is developed.

The evaluated ANN architectures mainly differ in the type of RNN used. LSTM propagates information through a 1D sequence which is aligned with the horizontal axis of the image. MDLSTM extends LSTM by using a 2D sequence, therefore information is also propagated along the vertical axis. A new method using only convolutional operations is suggested in the form of the IDCN and achieves state-of-the-art performance. MDLSTM outperforms the other two RNN types and is therefore used to compare the results to those of other authors. Evaluated on the Bentham dataset, MDLSTM performs 0.1% better than LSTM and LSTM again performs 0.1% better than IDCN regarding CER. The ANN is trained using the CTC loss and outputs the recognized text in a specific coding scheme. Multiple decoding algorithms are evaluated. Using best path decoding as a baseline, VBS decreases the CER from 7.35% to 7.08% while token passing decreases the WER from 26.29% to 7.88% on the Ratsprotokolle dataset. The integration of LMs especially helps decoding noisy data by incorporating information about language structure.

Finally, the recognized text can be checked for spelling mistakes. A text-postprocessing method tries to find the most likely word given a misspelled word. This method is trained on the ANN output and is therefore able to learn common mistakes of the ANN such as confusing the characters “a” and “o”. It is able to decrease the CER by 0.2% on the CVL dataset.

5.2 Future Work

Further preprocessing methods to simplify the task for the classifier will be tested. One promising method is to remove the slope of the text, which results in text lying approximately in horizontal direction. This also enables tighter cropping of the text-lines which yields larger text in the input images. More variability will be added to the data by using further data augmentation methods. Deslanting decreased the recognition performance on the Ratsprotokolle dataset. It will be tried to first deslant the images and then add a random slant again to augment the data. This might improve the results for the mentioned dataset in case the decrease in recognition performance was due to removing data variability.

Regarding the classifier, the suggested IDCN will be further improved by doing hyper-parameter search on parameters such as dilation width and number of layers. The concatenation of all intermediate outputs of the IDCN layers uses a lot of memory. Further experiments will be conducted to identify intermediate outputs which can be

ignored or downsampled without affecting the accuracy. The downsampling scheme of an image-pyramid is a possible starting point for these experiments.

The text-postprocessing uses edit operations to produce suggestions which have an edit distance of 1 to the original word. This will be extended to recursively produce suggestions with an edit distance of 2 or more. The number of suggestions grows exponentially with the number of recursions, however the frequency of confusing certain characters differs, therefore a large number of suggestions can safely be ignored. It might also improve the results to condition the insert and delete operation on the surrounding characters.

Bibliography

[AMS92] J.I. Aoe, K. Morimoto, and T. Sato. An efficient implementation of trie structures. *Software: Practice and Experience*, 22(9):695–721, 1992.

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[Blu15] T. Bluche. *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition*. PhD thesis, Université Paris Sud-Paris XI, 2015.

[Blu16] T. Bluche. Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. In *Advances in Neural Information Processing Systems*, pages 838–846, 2016.

[Bra08] P. Brass. *Advanced data structures*. Cambridge University Press Cambridge, 2008.

[ECGZ11] S. Espana-Boquera, M. Castro-Bleda, J. Gorbe-Moya, and F. Zamora-Martinez. Improving offline handwritten text recognition with hybrid HMM/ANN models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(4):767–779, 2011.

[FFFB11] A. Fischer, V. Frinken, A. Fornés, and H. Bunke. Transcription alignment of Latin manuscripts using Hidden Markov models. In *Proceedings of the 2011 Workshop on Historical Document Imaging and Processing*, pages 29–36. ACM, 2011.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[GFGS06] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist Temporal Classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 369–376. ACM, 2006.

[GFS07] A. Graves, S. Fernández, and J. Schmidhuber. Multi-dimensional Recurrent Neural Networks. In *International conference on Artificial Neural Networks*, pages 549–558, 2007.

[GJ14] A. Graves and N. Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1764–1772, 2014.

[GLF+09] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5):855–868, 2009.

[Gra12] A. Graves. *Supervised sequence labelling with recurrent neural networks*. Springer, 2012.

[GSC99] F. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. *Neural Computation*, 12(10):850–855, 1999.

[GSK+17] K. Greff, R. Srivastava, J. Koutník, B. Steunebrink, and J. Schmidhuber. LSTM: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, pages 2222–2232, 2017.

[Hoc91] S. Hochreiter. Untersuchungen zu dynamischen neuronalen Netzen. Master's thesis, Technische Universität München, 1991.

[HS97] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[HS16] K. Hwang and W. Sung. Character-level incremental speech recognition with recurrent neural networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5335–5339. IEEE, 2016.

[JM14] D. Jurafsky and J. Martin. *Speech and Language Processing*. Pearson London, 2014.

[KFDS13] F. Kleber, S. Fiel, M. Diem, and R. Sablatnig. CVL-Database: An off-line database for writer retrieval, writer identification and word spotting. In *12th International Conference on Document Analysis and Recognition*, pages 560–564. IEEE, 2013.

[KSH12] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[LBBH98] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LBOM12] Y. LeCun, L. Bottou, G. Orr, and K. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

[MB] U. Marti and H. Bunke. IAM. `http://www.fki.inf.unibe.ch/databases/iam-handwriting-database`. Accessed: 2017-11-08.

[MB01] U. Marti and H. Bunke. Text line segmentation and word recognition in a system for general writer independent handwriting recognition. In *Sixth International Conference on Document Analysis and Recognition*, pages 159–163. IEEE, 2001.

[MB02] U. Marti and H. Bunke. The IAM-database: an English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, 2002.

[MH17] M. Mukkamala and M. Hein. Variants of RMSProp and Adagrad with Logarithmic Regret Bounds. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2545–2553, 2017.

[MS99] R. Manmatha and N. Srimal. Scale space technique for word segmentation in handwritten documents. *Lecture Notes in Computer Science*, pages 22–33, 1999.

[ODZ+16] A. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[Ots79] N. Otsu. A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66, 1979.

[PF09] T. Plötz and G. Fink. Markov models for offline handwriting recognition: a survey. *International Journal on Document Analysis and Recognition*, 12(4):269–298, 2009.

[PSKC10] V. Papavassiliou, T. Stafylakis, V. Katsouros, and G. Carayannis. Handwritten document image segmentation into text lines and words. *Pattern Recognition*, 43(1):369–377, 2010.

[SBY16] B. Shi, X. Bai, and C. Yao. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and its Application to Scene Text Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):2298–2304, 2016.

[SC94] G. Seni and E. Cohen. External word segmentation of off-line handwritten text lines. *Pattern Recognition*, 27(1):41–52, 1994.

[SDN16] D. Suryani, P. Doetsch, and H. Ney. On the Benefits of Convolutional Neural Network Combinations in Offline Handwriting Recognition. In *15th International Conference on Frontiers in Handwriting Recognition*, pages 193–198. IEEE, 2016.

[SRTV14] J. Sánchez, V. Romero, A. Toselli, and E. Vidal. ICFHR2014 competition on handwritten text recognition on transcriptorium datasets . In *14th International Conference on Frontiers in Handwriting Recognition*, pages 785–790. IEEE, 2014.

[SRTV16] J. Sánchez, V. Romero, A. Toselli, and E. Vidal. ICFHR2016 Competition on Handwritten Text Recognition on the READ Dataset. In *15th International Conference on Frontiers in Handwriting Recognition*, pages 630–635. IEEE, 2016.

[SVBM17] E. Strubell, P. Verga, D. Belanger, and A. McCallum. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. In *Conference on Empirical Methods in Natural Language Processing*, 2017.

[TE96] X. Tong and D. Evans. A statistical approach to automatic OCR error correction in context. In *Proceedings of the fourth Workshop on Very Large Corpora*, pages 88–100, 1996.

[VDN16] P. Voigtlaender, P. Doetsch, and H. Ney. Handwriting recognition with large multidimensional long short-term memory recurrent neural networks. In *15th International Conference on Frontiers in Handwriting Recognition*, pages 228–233. IEEE, 2016.

[VL01] A. Vinciarelli and J. Luettin. A new normalization technique for cursive handwritten words. *Pattern Recognition Letters*, 22(9):1043–1050, 2001.

[VLL94] V. Vapnik, E. Levin, and Y. LeCun. Measuring the VC-dimension of a learning machine. *Neural Computation*, 6(5):851–876, 1994.

[YK16] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016.

[YRT89] S. Young, N. Russell, and J. Thornton. *Token passing: a simple conceptual model for connected speech recognition systems*. Cambridge University Engineering Department Cambridge, 1989.