Universität des Saarlandes

Master Thesis

Speeding up Data Annotations for Handwritten Text Recognition

Author:

Moritz Wolf
moritz.wolf@dfki.de
2542750

Supervisors:

Prof. Dr. Dietrich Klakow

Dr. Jan Alexandersson

September 29, 2021

UNIVERSITÄT
DES
SAARLANDES

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Declaration

I hereby confirm that the thesis presented here is my own work, with all assistance acknowledged.

Saarbrücken, September 29, 2021

Abstract

Transcribing huge amounts of handwritten text is necessary to train supervised machine learning models due to the variability of handwritten text. This however is costly in both time and manpower. A common strategy to speed up annotation processes is to use human-in-the-loop methods and giving supporting suggestions to the annotator. In this thesis a combination of Transfer Learning and Active Learning with uncertainty sampling is proposed to incrementally train a well performing model that produces these suggestions after seeing only a few samples. We compare this strategy to using Active Learning with random sampling and find that uncertainty sampling leads to better suggestions and see a reduction in time humans spend in the annotation process by more than a half. We also find, that Active Learning converges at a better performance than conventional methods, that learn from all training data at once.

Contents

List of Figures

List of Tables

1. Introduction

Though there are more and more unsupervised approaches emerging, the great majority of modern machine learning applications are powered by supervised machine learning and require human feedback. Today, our intelligent devices are learning less from programmers who are hardcoding rules and more from examples and feedback given by humans, programmer or not. Be it an in-home virtual assistant or a machine translation system, they all rely on thousands, sometimes millions of annotated samples - the training data. Thus it is not surprising that one of the most widespread challenges for machine learning applications today is data scarcity. Having access to only a limited amount or no training data at all, makes the creation of usable supervised machine learning models impossible. In times of big data the availability of data in general is not always the problem anymore, but often there is an abundance of raw unlabeled data, that can not be used for machine learning until labeled by a human. This special case of data scarcity is called *annotation scarcity*. Common reasons why data is only scarcely annotated are, that the annotation process is too costly in time, manpower or money. This is typical for data where experts must do the labeling as in the medical domain, or when the task is especially time consuming as is often the case for semantic segmentation of images and handwritten text recognition.

One such example is annotating the Paquid dataset – a collection of over 12000 scans of handwritten patient records and interviews [8] (see Section 3.1). Similar data is used in current dementia research [18], but there is very little data available that is in traditionally machine-readable form. Due to the scale of the Paquid dataset the transcription of its contained text is sought after for training machine learning models. However, as long as they are in the form of images of handwritten text they are useless for such purposes. Since transcribing all 12000 relevant scans completely by hand is fairly inefficient and the resulting cost would be not viable either, the obvious solution is to (semi-) automatically transcribe the text contained in the images into digital text.

Offline Handwritten Text Recognition (HTR) systems transcribe text contained in scanned images into digital text. In contrast to the more well-known Optical Character Recognition, HTR does not focus on transcribing individual letters. Instead, it scans and processes images of entire lines or words and tries to decode this data in sequence. The input for HTR systems therefore usually has to be segmented separately in a previous step, i.e., depending on the system, either the words or lines of handwritten text have to

be isolated before being processed by the HTR model. The simplest method to extract the lines is to place a simple rectangle (bounding box) for each localized line.

A system able to effectively recognize handwritten words should be able to deal with the inherent variability of handwriting text. This variability not only comes from the different writing styles across different individuals, but also shows in text written by the same person at different times, with different writing materials (for examples see Figure 5).

For HTR systems, and machine learning in general, most of the time the solution is to just increase the amount of training data to account for all the different styles and variants. However humans have the same problem as well. Some hand writings may be easier to decipher than others and being unfamiliar with the text or even the language it is written in may amplify the problem further. Annotations for HTR therefore are typically costly on account of the duration of the process combined with the large amount of data needed.

To minimize this cost, we need to both minimize the amount of labeled data that is needed to train a machine learning model model and also minimize the time an annotator needs to label the data. Fortunately there exist methods to achieve exactly that: *Human-in-the-loop* machine learning techniques combine human and machine intelligence to both help machine learning models with training more efficiently and also assist humans in the annotation process. The cornerstones of human-in-the-loop machine learning are sampling data that is advantageous for training the model, labeling them by a human, using that data to train a model, and using that model to sample more data to annotate. Its most common method is to use Active Learning (AL), a incremental learning technique that decides which samples it should learn from next based on some criterion. Through incrementally adding to the training data, the model then gets better with each iteration until it can theoretically be used to predict the rest of the data fully automatically. The most common sampling criteria are either to just choose samples at random, or, more effectively, choose samples, for which the machine learning model is uncertain about the correct label.

We now established how humans can help a machine learning model to learn more efficiently, but maybe the model could help the human with labeling? To compute uncertainty, the model has to generate predictions for all unlabeled data anyway. We could then use these predictions as suggestions for the human annotator. That way, if at

least some of the predictions are correct, the annotators only have to amend the wrong predictions and therefore save time.

The topic of this thesis is to explore possible strategies how to annotate a dataset such as the Paquid dataset efficiently using human-in-the-loop machine learning. The proposed best strategy is using AL with uncertainty sampling. The model's predictions for the sampled items in each iteration are then used as suggestions in the annotation process.

There are three research questions that this thesis seeks to explore:

1. **Does an HTR model trained with Active Learning produce better suggestions in each iteration when using Uncertainty sampling than with Random sampling?**
   We want to of course use Active Learning in the most effective way, so we need to explore if uncertainty sampling is actually beneficial in an HTR context or if random sampling gives similar results.

2. **Do suggestions predicted by an HTR model trained with Active Learning with uncertainty sampling make the annotation process faster compared to using only a pretrained model?**
   Since uncertainty chooses samples where the model is the least confident, the predictions for these samples must be expected to be the least correct ones. Of interest is then, whether the suggestions even benefit the annotation process.

3. **Does an HTR model trained with Active Learning converge at a better performance than non-incremental learning methods?**
   Another interesting aspect is, whether the resulting models can be used not to just produce suggestions, but to automatically correctly predict the rest of the dataset. The AL models' performance should improve with each iteration until they converge at a certain point. The question would then be, how the performance of AL compares to training a model with convential methods, i.e. using all training data at once.

In the following chapters we will explore potential answers to these questions. First, to establish a mutual understanding of the topic and the prerequisites, we will discuss the related work, concerning annotation tools, HTR systems, and how Active Learning and Transfer Learning were previously used with HTR. After that the systems and data that was used for the experiments are introduced: In Chapter 4, the digitization- and

annotation tool HUMAN is introduced. The mentioned Paquid dataset, a prime example to show the challenges for HTR and human annotation alike, is presented in Chapter 3.1, followed by the HTR system Pylaia in Chapter 5. The conducted experiments to find answers to the research questions and their results are presented in Chapter 6 with their implications on the research questions being explored in Chapter 7 Finally we conclude the thesis with potential answers to the research questions and discuss the new questions that arose from the findings of the experiments for future work.

2. Related Work and Prerequisites

2.1. Handwritten Text Recognition

Handwritten Text Recognition was as a matter of fact one of the first application scenarios of neural networks. As early as 1998 Lecun et al. [17] proposed Convolutional Neural Networks (CNN) to recognize handwritten digits from the MNIST dataset. These early approaches largely depended on the use of Hidden Markov Models (HMM) to handle the sequential nature of handwritten text [6]. These are nowadays largely outperformed by using Recurrent Neural Networks (RNNs) not only in HTR but nearly all Natural Language Processing tasks. Subtypes of RNNs, such as Bidirectional Long Short-Term Memory (BLSTM) [12] or Multidimensional Long Short-Term Memory [14] have been widely adopted by the handwritten text recognition community. However there still are newer competitive approaches combining HMM with RNN [5, 3].

Formally, what these systems are trying to solve is the following optimization problem:

$$\hat{y} = \underset{y}{\operatorname{argmax}} P(y|x) \tag{1}$$

Where $\hat{y}$ is the output transcript, x is the input signal, represented as a sequence of frames in an image, and y is a sequence of textual symbols typically characters. The frames are typically a vertical column of pixels of a predefined kernel size, which are passed through multiple CNN layers. The resulting sequence is then processed by one or multiple RNN layers.

However, there is an inherent issue when trying to train HTR systems: How do we align the pixel frames in an image with the corresponding transcript of the text contained in

that image? This task is not as trivial as it may seem, because the input length of the symbols in the image may vary in length and the frame size can not be dynamically adapt to that. This problem is also well-known in Automatic Speech Recognition (AMR), where people's rates of speech varies and we have a continuous signal that has to be mapped to text. Connectionist Temporal Classification (CTC) [15] offers an approach to finding this alignment between input and output. The output of the RNN layers is a 2-dimensional CTC matrix containing all symbol probabilities for each frame.

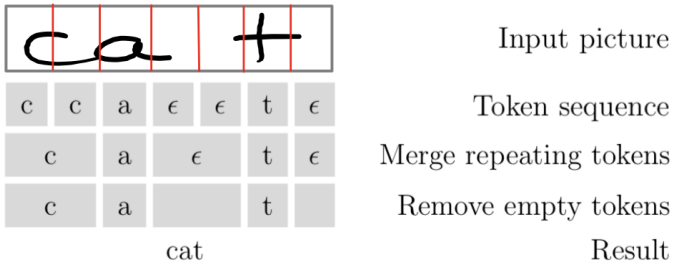


Figure 1: Example for GBF Decoding for CTC: The network makes predictions for each timestep (separated by red lines). Greedy Best First search computes the sequence of labels with highest probability in each timestep. Then repeating tokens are merged and empty tokens removed, resulting in the final output text.

A CTC decoding algorithm maps these symbol probabilities to the final text. The most standard decoder uses Greedy Best First search (GBF), choosing the token with highest probability in each timestep. In the resulting sequence repeating tokens are first merged and then any empty tokens removed. Figure 1 illustrates the full process in an example As a side note, due to the relation to AMR, frames are conventionally called timesteps in this context.

Beside GBF decoding, there are other approaches to calculate the final labeling from a CTC matrix. Graves [13] introduced the token passing alorithm based on state machines. Their algorithm searches for the most likely sequence of words in a dictionary by aligning them with the CTC matrix and scoring word-transitions with a Language Model.

Other decoding approaches such as CTCWordBeamSearch [28] incorporate a combination of beam search with a Language Model. Beam search is a heuristic search algorithm that explores a graph by iteratively expanding the most promising node in a set of candidates. In the case of HTR multiple candidates for the final labeling are calculated

from the probabilities in the CTC matrix. These candidates are called beams. In each iteration, each beam-labeling is extended by all possible labels and the beams from the previous iteration to form a tree as shown in Figure 2. To avoid exponential growth of the tree, only the beams with highest probability are kept after each iteration - the number of beams to keep is named "beam width". If two beam-labelings are equal, they get merged by first summing up the probabilities and then discarding one tree.

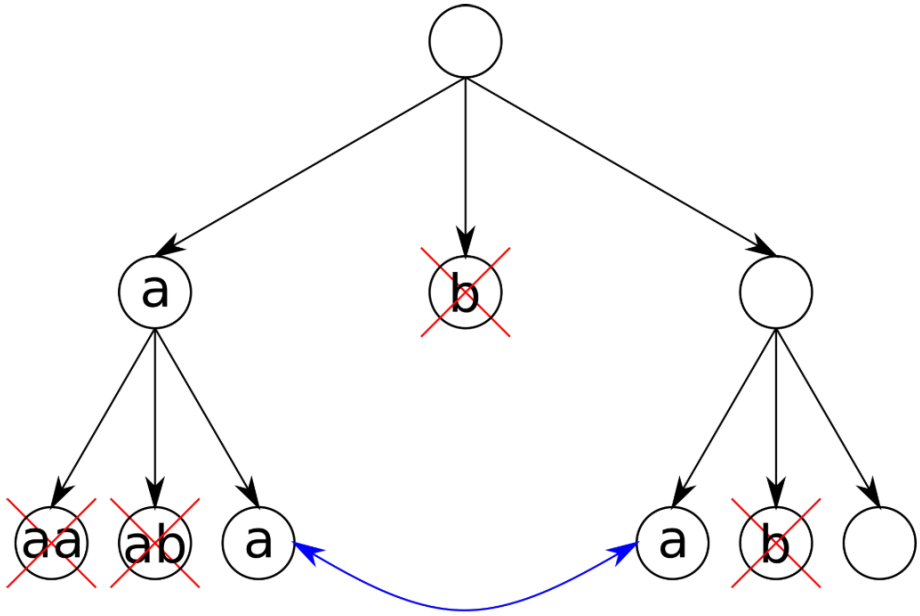


Figure 2: Example of iteratively extending beam-labelings (from top to bottom) to form a tree. Possible candidates are "a" and "b". The beam width is 2 meaning all except for the two best beams are removed (red cross) and equal labelings get merged (blue arrow). (Figure taken from Scheidl et al. [28])

Scheidl et al. [28]'s CTCWordBeamSearch introduces a mechanism to score the beams in each iteration with a word-level Language Model. It works by keeping track of the last decoded word in the beam and predicting possible words based on the current character sequence. A Bigram Language Model then computes scores based on the last word and all possible words and the scores are summed up. The best beams are then found by multiplying the beam probability with the Language Model score.

Examples for models that are based on the described CNN-RNN-CTC architecture are *SimpleHTR* [27], Bluche et al. [4]'s model and *Pylaia* [25].

Both the model by Bluche et al. [4] and *SimpleHTR* use computationally costly multidimensional LSTM layers. Puigcerver [25] found that using these multidimensional LSTM networks may not be necessary for a good outcome. His finding suggests, that "2D-LSTMs could be replaced by convolutional layers, at least at some extent in the lower layers, reducing the required computational resources with no (or little) loss on accuracy" [25]. A version of the framework *Pylaia*, that resulted out of these findings, is now used together with the annotation tool Transkribus [22] (see Section 2.2). A more detailed description of Pylaia follows in Chapter 5.

Nurseitov et al. [24] compared these 3 models for Russian and Kazakh handwriting recognition. One of the tests was to classify handwritten city names, a related task to recognizing animal names which was the focus of the experiments in this thesis. All models performed well on the task, with Bluche et al. [4]'s model and SimpleHTR slightly outperforming Puigcerver [25] due to overfitting.

2.2. Annotation tools

There are a plethora of annotation tools concerned with transcribing handwritten text or semantic segmentation. Some interesting examples are listed in this section.

Trankskribus [22] is an annotation tool for handwritten text and is mostly used for historical texts and manuscripts. The annotated data can then be uploaded to a server and used to train an HTR model with Pylaia (see Section 2.1 and Chapter 5). This model can then be used to directly annotate new data or give suggestions to human annotators.

Yimam et al. [35] used WebAnno [11] to annotate named entities. The annotators were shown predictions of a Named Entity Recognizer model with an f-score of 0.89. Showing these suggestions made the annotation process 20% faster than when not showing them.

Nova [2] uses a Human-in-the-Loop approach to train machine learning models while annotating. It allows to train and evaluate machine learning models, such as Support Vector Machines or neural networks directly from the interface. There are two strategies: (1) Session completion, where a model is trained on the first minutes of an annotated sessions to predict the remaining session, and (2) session transfer, where a model is trained on multiple sessions to predict completely unknown data.

Supervisely[1] is a commercial web platform for annotations of computer vision tasks. It incorporates a variety of models and model pipelines to generate suggestions to help annotators.

HUMAN [33] is another open source web-based annotation tool that uses modules to cover a variety of annotation tasks on both textual and image data. Chains of these modules are controlled by a deterministic state machine, making it a flexible and fast to set up tool. The server-client structure and its modularity allow it to be easily used in a human-in-the-loop system. As this is the tool that was used for experiments in this thesis, it is described in more detail in Chapter 4

2.3. Human-in-the-loop machine learning for HTR

The large cost for annotations raises the question, if and how it is possible to combine human and machine intelligence in the most efficient way to reduce this cost. *Human-in-the-loop machine learning* represents one solution to this question. Its most common techniques are to use Active Learning (AL) together with annotations. Machine learning models almost always get more accurate with more labeled data. Active learning is the process of deciding which data is used in the learning process. The most common technique to find the most beneficial data for training a model is to let the machine learning model itself decide which samples it should learn from next.

Figure 3 shows the Human-in-the-loop process in principle. Through incrementally adding to the training data, the model gets better in each iteration until it can theoretically be used to predict the rest of the data fully automatically. Therefore the general term for this type of learning is "incremental learning".

[1] `https://supervise.ly` (accessed: 15.12.2020)

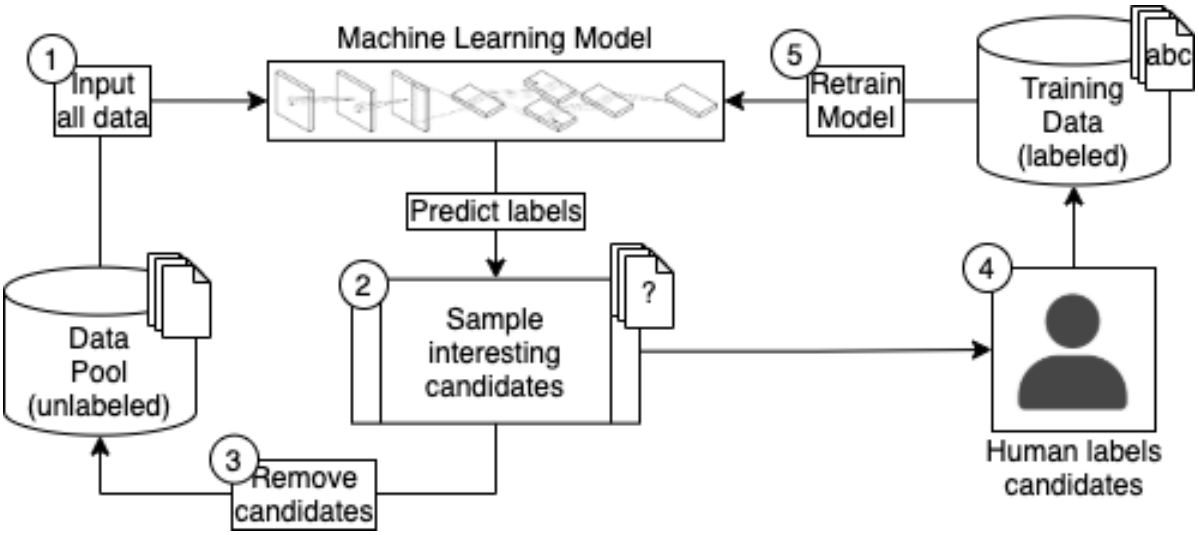


Figure 3: The Human-in-the-loop paradigm, employing Active Learning and human annotations to label data. (1) Predict labels for all unlabeled data in the data pool. (2) Choose interesting samples. (3) Remove the chosen samples from the data pool. (4) Human annotator labels the candidates. (6) The labeled data gets appended to the training data. (5) The model retrains with this data. Repeat from (1).

There are two dominant methods to choose the interesting candidates for training - random sampling and uncertainty sampling. Random sampling, as the name suggests, just chooses random samples from the pool of unlabeled data. Uncertainty sampling, on the other hand, chooses unlabeled items that are near the decision boundary in the current model. For a binary classification task, an uncertain item will have close to a 50% probability of belonging to either label. These items are most likely to be wrongly classified, so they are the most likely to result in a label that differs from the predicted label. The idea is, that after adding the labeled uncertain items to the training data and continuing the training of the model, the decision boundary will be moved and the model can distinguish better between the two labels. There are many ways to calculate uncertainty. The three most common approaches according to Settles [29] are:

1. **Least confidence sampling**
   The least confidence is the difference between the most confident prediction and 100% confidence. This is the easiest method to compute as we can just choose samples for which the most probable label prediction had the lowest probability compared with all other samples.

2. **Margin sampling**
   The difference between the two most confident predictions - for the label that the model predicted, how much more confident was it than for the next-most-confident label.

3. **Entropy-based sampling**
   Entropy, also known as surprisal, is a metric in information theory, that represents the amount of information needed to "encode" a distribution. As such, it is also often thought of as a measure of uncertainty in machine learning. Entropy is defined as
   $$H(x) = \frac{-\sum_{i=1}^{n} P(x_i) log(P(x_i))}{log(n)}$$
   where n is the number of possible labels and $P(x_i)$ is the prediction probability of label i. Thus in contrast to the other two sampling methods, entropy-based sampling takes all probabilities for a prediction into account.

That leaves us with only one more question: How do we start the process? In the first iteration, we do not yet have a model which could produce any predictions, so we can not find uncertain items or show suggestions to the annotator. A common and recently very popular technique to solve this is Transfer Learning.

Transfer Learning (TL) is the process of taking a machine learning model that was built for one specific task and adapting it to another task. This is also known as *domain adaptation* or *pretraining* models. Popular pretrained models that are commonly used for TL include BERT[10] for NLP tasks or ImageNet[9] for Computer Vision tasks, revolutionizing their respective fields over the last years. The way this typically works is that the last layer or last few layers of a base model are refitted to output new target label for the desired task. This is useful, when there is not enough data available to achieve a high accuracy for a specific task, but there is enough data for a related task. As a simple example, let's assume that we are doing semantic segmentation on images and want to identify cats and dogs and only have a small dataset containing images labeled "cat"and "dog". Let's also assume we have an additional much larger corpus of labeled images of zoo animals. We could then train a model on the larger zoo dataset which can then classify "lion", "wolf", "elephant", and so on really well and then continue training, (also known as "retrain" or "fine tune"), the last layer with our very small corpus to output the labels "cat" and "dog". However when the output labels are the same in both tasks, as is in HTR system which recognise the same characters, it is common to

fine tune the complete model with samples from the target domain.

There are no large corpus pretrained models available out-of-the-box for HTR as they are for Computer Vision or general NLP. It is however possible to train an HTR model on a larger dataset of handwritten text ourselves. This base model can then produce predictions for the Paquid dataset or any handwritten text, but these models typically do not generalize well for new samples because of the mentioned inherent variability of the data. Thus fine tuning with for example Active Learning is necessary to get a model that produces actually helpful suggestions.

Following are some examples where Transfer Learning and human-in-the-loop methods were used for HTR applications or similar problems.

There are two examples where Transfer Learning was successfully used for HTR: Aradillas et al. [1] tried to improve HTR for small training data sizes by applying TL. They trained a HTR model with the CNN-LSTM-CTC combination, as described in Section 2.1, using the IAM Dataset [21], one of the largest freely available labeled datasets of handwritten text. They then retrained the model on the Washington Dataset, a comparably small dataset of 565 text lines. They tried retraining with all combinations of the last layers of the model, from just the last layer to all layers from beginning to end. The results showed, that retraining nearly all layers yielded the best prediction performance on the smaller Washington dataset and a massive boost in performance, 5% Character Error Rate (CER) with TL compared to 40% CER by learning only from the Washington dataset. Cascianelli et al. [7] also tried learning on different larger scale corpora, one of which was the IAM dataset, with the Pylaia HTR-system. Interestingly they did not fine tune the model on the target corpus but tried adapting the model directly and transcribe ancient italian manuscripts. The results of this were expectedly fairly unsatisfactory, with CER of 70%.

Active Learning was occasionally used for HTR in the past: Romero et al. [26]'s approach is one such example. They derive the final labels for handwriting through a probabilistic finite-state automaton and compute the derivational entropy of all possible paths. Their AL algorithm then chooses samples with the highest derivational entropy, obtaining slightly a better WER than when using random sampling.

Malhotra et al. [20] presented an approach for AL and CTC in the related field of Automatic Speech Recognition. They used the path probability of the decoded CTC sequence as a uncertainty measure and selected the samples with the least confidence,

normalized for length of the sequence. They also tried sampling randomly and found uncertainty sampling consistently outperforming random sampling. Finally using AL to not only benefit the model performance, but also support humans in the annotation process by showing suggestions of the current model was proposed by Yang et al. [34] for segmentation of biomedical images. However they did unfortunately not compare the annotation times with or without these suggestions.

3. Data

To answer the research questions, parts of the previously mentioned Paquid dataset were annotated. This corpus serves as a typical real-world example to show the challenges both human annotators and HTR systems face. In this chapter, first the dataset itself will be introduced and the challenges in annotating it will be described, followed by the preprocessing steps that had to be performed before the experiments.

3.1. Corpus

The Paquid[8] cohort are 3777 individuals aged 65 years or older from France. A subset of 2792 subjects was studied from 1988 until 2004 on the prevalence and the correlates of clinically diagnosed dementia.

The study's resulting dataset that I had access to consists of 12581 PDF and TIFF files for 2615 test subjects, containing scans of filled in patient interview forms written on paper. The subjects were interviewed at most seven times, depending on if and when they dropped out of the experiment. This results in there being at least one and at most seven relevant files per subject. They can contain a general anamnesis, a second anamnesis specifically for dementia, and a series of cognitive tests used to diagnose dementia, but may be missing any of these. These paper transcripts then were scanned or in some cases saved via fax. The tests include Trail Making tests, drawing a complex figure, and recounting words and characters in a specific order. There is one more test which is of special interest for dementia research and the focus for this thesis: the Isaac's Set Test.

The Isaac's Set Test is a test on verbal fluency, where patients are asked to recite the highest number of words they can in 4 semantics categories (colours, animals, fruits and cities) in one minute. All words are noted down by hand by an interviewer into a prepared form. For each category, after 15 and 30 seconds a "/" character is added in the transcript and after 60 seconds the end of the test is marked with "//". This thesis concentrated on annotating the transcripts of the category "animal names" since it seemed most promising for future dementia research [18]. Keeping the data in one specific domain also simplifies the annotation process and keeps potential Language Models simple.

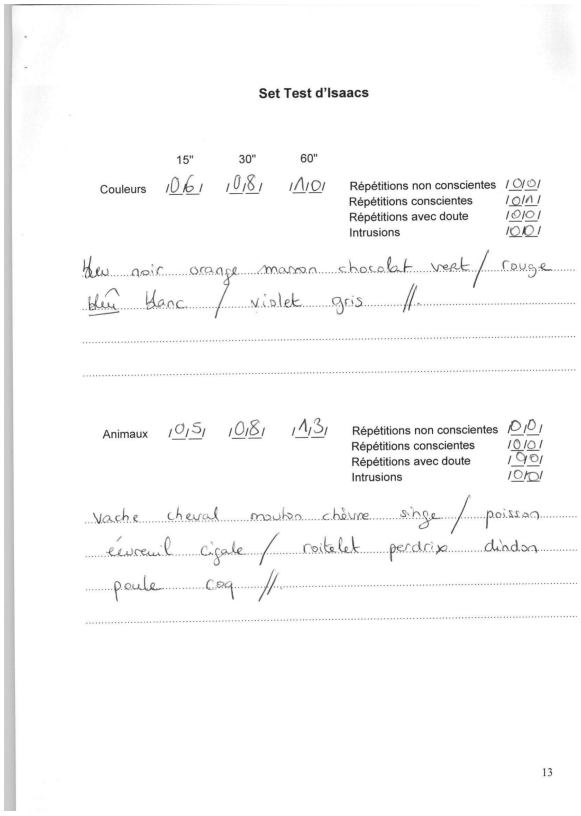


Figure 4: An example page of the Isaac's Set Test with the categories "colors" and "animals"

An example of how a page containing such a transcript is typically structured can be seen in Figure 4. The structure of the page containing the Isaac's Set Test itself is always similar, but the pages location within a document varies greatly. This is because the structure of the document was changed over the several years of the study, leading to 7 differently structured document types. However the document types are also not consistent in themselves, as apparently often errors in the scanning process occurred, with duplicate pages, and missing pages or whole sets of pages being commonplace.

The fast nature of the task additionally led to a lot of noise being introduced into the transcripts. Since the interviewers had to write down text during the test, they had to compromise in regards to readability. Abbreviations and spelling errors are as commonplace as artefacts like struckthrough, underlined and boxed text. Combined with the fact that there was an unknown number of interviewers, this results in a variety of different writing styles.

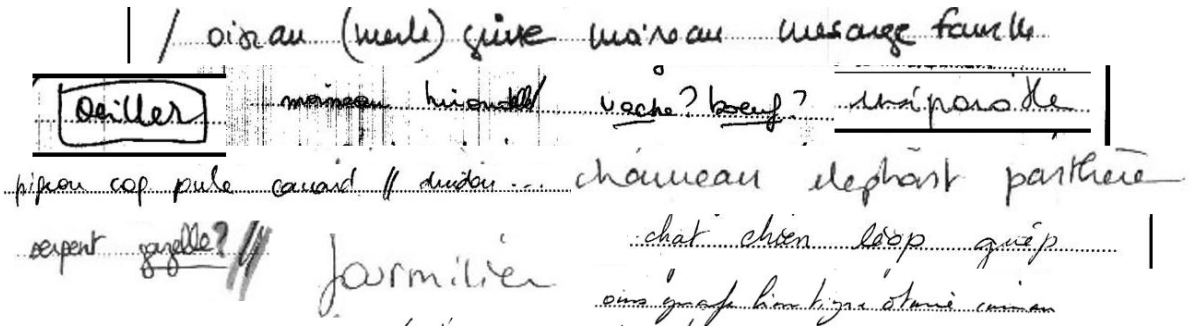


Figure 5: Text excerpts from the Paquid dataset.

This huge variability is demonstrated in Figure 5. It is a collage of excerpt lines and words coming from different classes and authors in the dataset and presents a variety of different artifacts and writing styles.

3.2. Preprocessing

The variability in document types made it non-trivial to find the correct page. However it was possible to naively guess page numbers for each document type which was correct for most documents. Extraction of that page from the document could be done by simple bash scripts using ImageMagick's [31] convert function. Sifting through the extracted pages and filtering out all unwanted pages was completed in a few days of work. Of the approximately 10000 pages that were extracted in this manner, 2000 were selected in total for annotations in the experiments. This number was chosen to be able to complete the annotations in the time frame of this thesis.

For the later described Baseline Experiment (see Section 6.1) for 300 of these 2000 pages followed an initial paragraph and line segmentation. In a similar manner as the page segmentation, for each document type, an area for the paragraph was discerned which would be correct for most pages. The area of the paragraph was then split into four parts horizontally to get the approximate areas of the lines. The line areas were then extracted again using ImageMagick. All resulting segmentations were then used and further processed in the Baseline Experiment.

This semiautomatic preprocessing took around a week of manual labor in total.

3.3. Conclusion

The Paquid dataset is a prime example to show the difficulties with processing medical texts and handwritten text in general. In this chapter we have seen the vast variability in the writing styles and also between different document types. The extracted pages of this dataset will be used as the unlabeled data for the experiments in Chapter 6.

4. A Tool for Digitization and Annotation: HUMAN

The annotation tool that was used for experiments in this thesis is the Hierarchical Universal Modular ANnotator (HUMAN) [33]. It is an open source web-based annotation tool that uses modules to cover a variety of annotation tasks on both textual and image data.

This modularity makes it easy to adapt to new annotation scenarios especially when several dependencies exist between annotations or when a single annotation task does not capture the problem. Both of this is the case when annotating data for HTR, as segmentation annotations and word annotations for the segmented lines are needed.

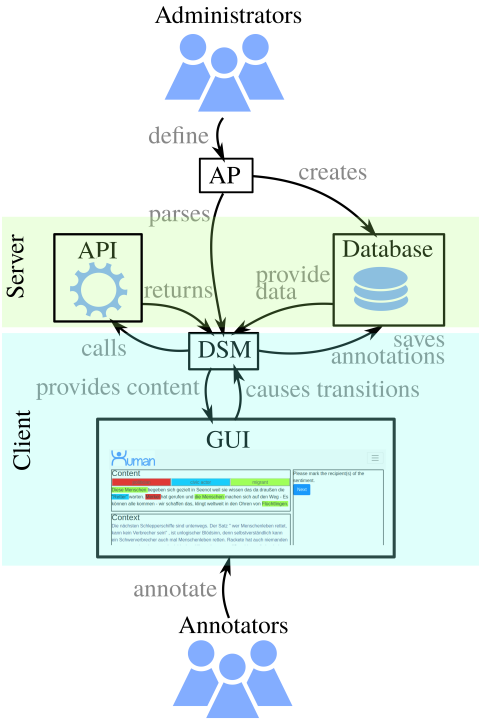


Figure 6: Architecture of HUMAN (from Wolf et al. [33]).

As a web-based tool its architecture follows a basic client-server model (see Figure 6). In this model, clients and servers exchange messages in a request–response pattern, where the client sends a request to which the server responds.

The client displays annotation modules in a Graphical User Interface (GUI). It is controlled by a Deterministic State Machine (DSM) which manages the order and transitions of the different modules. Through the DSM the client can request new content from the server or send finished annotations

to the server when an annotation instance is completed. The annotators interact with the DSM and modules through the GUI to solve their annotation tasks.

During the setup of the system, administrators design an annotation protocol, which is a JSON-style definition of the desired modules and their transition order. It thereby represents a simplified version of the DSM. The real DSM and database are then automatically created based on this protocol. This makes the tool very flexible and fast to set up.

The server consists mostly of two parts: Firstly, a database, which is used to choose new annotation instances to the client and saving finished annotations which are sent from the client. The second part is a customizable API, which can be used to further process annotations or send suggestions to the client to be shown to the annotator. For this, in the annotation protocol, an API function name can be defined which must correspond to an arbitrary function of the same name which defined in the API. This function will then be called at the transition into that annotation module. The arguments of the call, like images or annotations are predefined in the various annotation modules. The API can be used for iterative learning: We could define a function to train models with new annotated data from the database or automatically start classifications which are then shown in the client's GUI.

Because of its flexibility, the customizable API, which can be used for iterative learning, and the fact that, as one of the creators of HUMAN I was already very familiar with the codebase, it was decided to use this annotation tool for my experiments. However some modules and functionalities had to be created or adapted to be able to efficiently annotate handwritten text.

4.1. New modules and functionalities

HTR annotations consist of three tasks: Paragraph segmentation, line segmentation and line annotation. Note that these are dependant on each other, as we want to segment the lines inside the paragraphs and add word annotations for the content of each line. The segmentations are realised with bounding boxes. A corresponding *Bounding Box* module was already implemented in HUMAN, but the transition to other modules was unreliable and the underlying functionality had to be rewritten.

When loading the module an image is displayed in the background and, if available, predicted Bounding Boxes are drawn in the foreground (see Figure 8). A click inside of a box activates the box and adds anchor point in the corners and edges of the box as in Figure 7. The box can then be resized via drag and drop on these points. The whole box can be moved via drag and drop anywhere inside the box. When clicking outside of any bounding box, any activated box will be deactivated again and when there is no activated one a new box will be added at the clicked point. When finished a click on "Continue" can save all bounding boxes and then transitions to the next module.

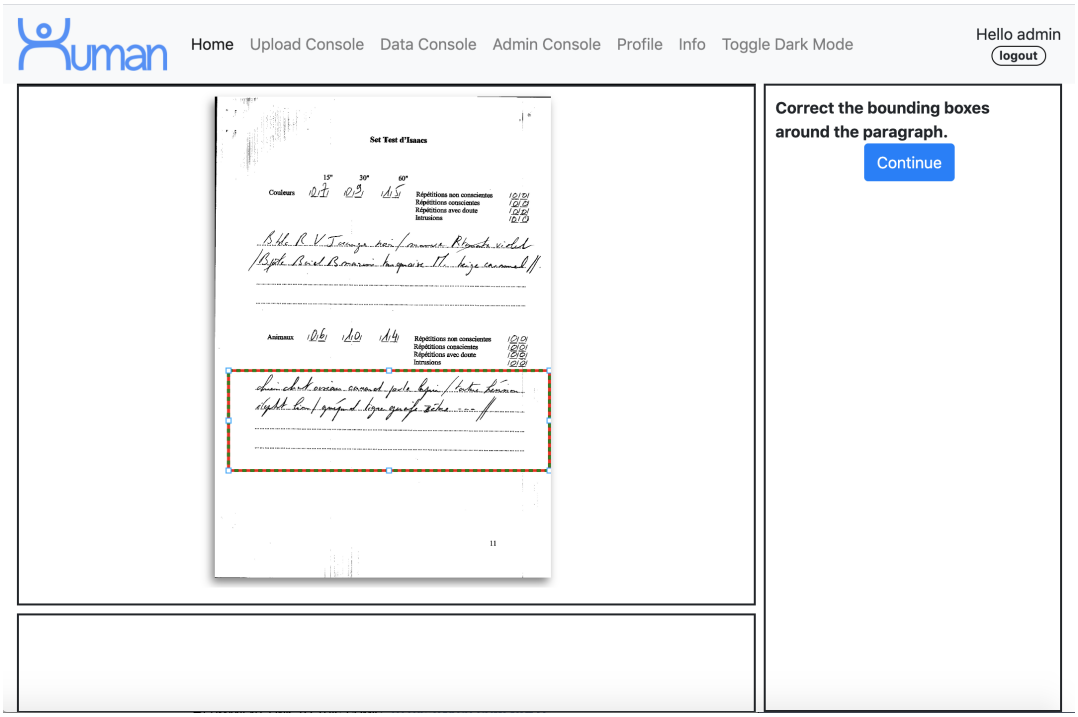


Figure 7: Example screenshots of the *Bounding Box* module for paragraph segmentation.

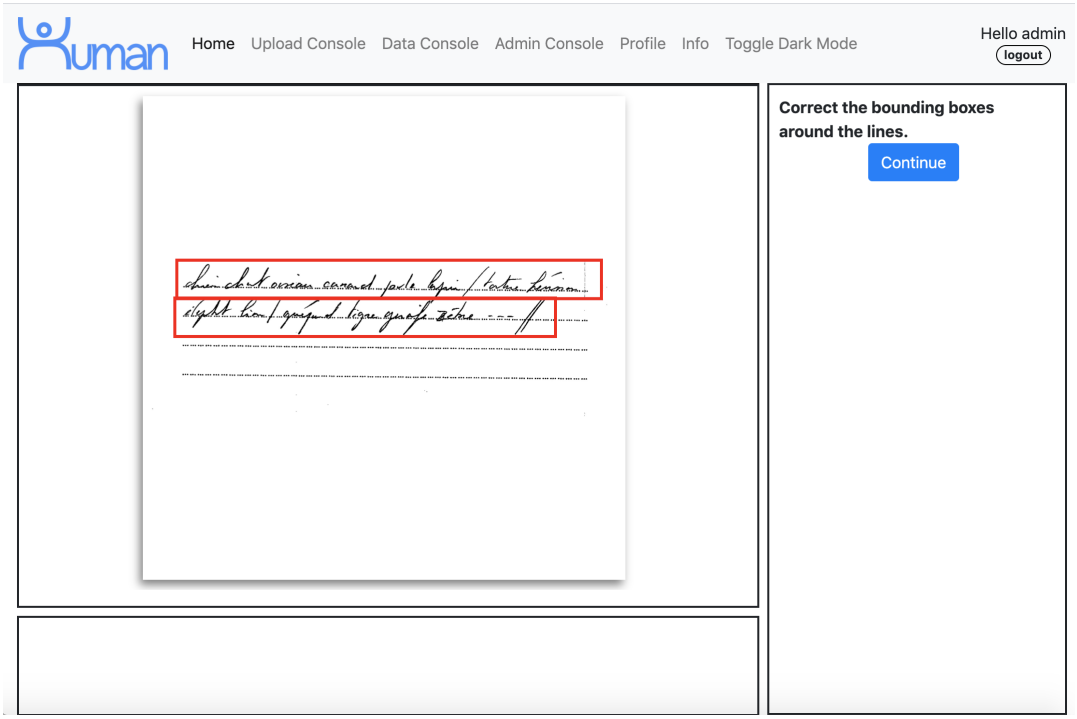


Figure 8: Example screenshots of the *Bounding Box* module for line segmentation

Bounding Box Labeling module To add labels to the lines' bounding boxes another module was needed, the *Bounding Box Label* module. For an example screenshot see Figure 9. It needs a bounding box as input either from a *Bounding Box* module or from the server API. Optionally the server can provide an ordered list of labels as suggestions. These suggestions are then displayed as labels on top of the bounding box. These labels can be individually marked by clicking on them or pressing *Enter* to move forward or *Shift+Enter* to move backward. The text inside the labels can be changed via an input field on the top right. To make it more convenient to find label texts, an autocomplete functionality was added with suggestions containing the input text shown in a list below the input field. The suggestions are taken from a predefined list of possible words. A suggestion can be selected by clicking or cycled with *Tab*- or arrow-keys. *Enter* transfers the selected item to the label and continues to the next label.

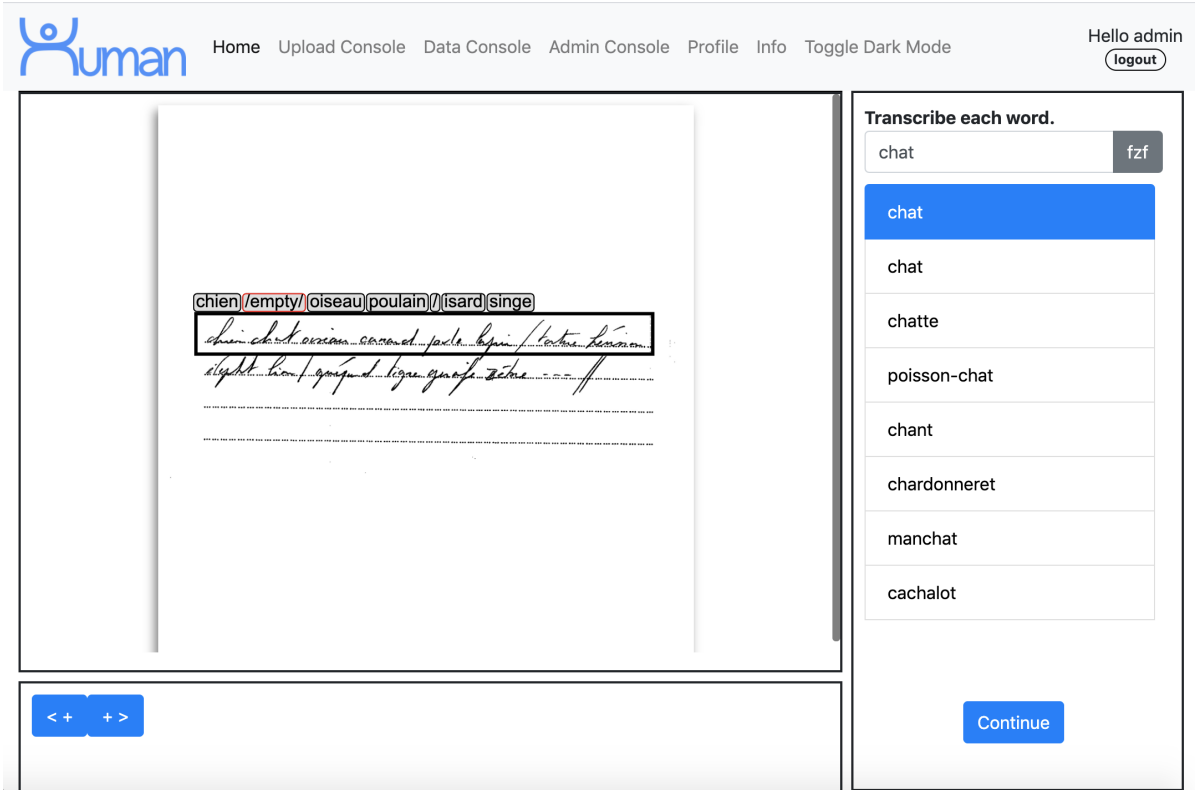


Figure 9: Example screenshot of the *Bounding Box Label* module

There are two autocomplete methods that can be toggled with a button beside the input box: A naive algorithm and fuzzy search. The naive algorithm only matches words that start with exactly the same string as the input. Fuzzy search, also known as approximate string matching, on the other hand, finds strings that match a pattern approximately rather than exactly, as the naive algorithm used. This means it tries to match words that **contain** the input somewhere but still keeps the order. For example "ptmus" matches "hippopotamus" and "mll" matches "mandrill" and "millepede". This proved advantageous for annotating handwritten text, as some words were illegible. Fuzzy search still works with only some characters or the parts of the word one could read and still gives reasonable suggestions. The fuzzy search algorithm used in HUMAN is based on FuzzyMatchv1 from the fzf project[2]. Its implementation into HUMAN was completely provided by my colleague Max Depenbrock.

The buttons "< +" and "+ >" add a new label box to the left or right of the currently activated box respectively. These labels have the initial text "/empty/" to mark them

[2]https://github.com/junegunn/fzf

as empty. Similarly, if an annotator removes the text in the input field the text for the label will be set to "/empty/". To make it easier to track when a prediction was corrected for each label both the prediction and the final annotation in pairs are saved. When a box gets added via the buttons, the prediction is set to "/empty/", to signify adding the label. This makes it easy to track when a label was added or removed.

New functionalities To find out how the quality of the predictions impacts the time needed for the actual annotations, this time naturally has to be measured. I therefore implemented timers that measure the time spent in each module and save them together with the annotations in the database. API functions to start training and classifications which could be shown to the annotators were implemented but could unfortunately not be used in practice. The annotation tool and HTR systems had to reside in different server structures which could not easily interact with each other, so training, classifications and filling the database with new unlabeled data and predictions had to be done via manually started scripts. All new modules and functionalities were uploaded to the open-source project and are freely available on Github[3].

[3] `https://github.com/uds-lsv/human`

4.2. Module combinations

In this section, we discuss how the previously introduced modules can play together for three different annotation tasks that can be used for handwritten text transcriptions.

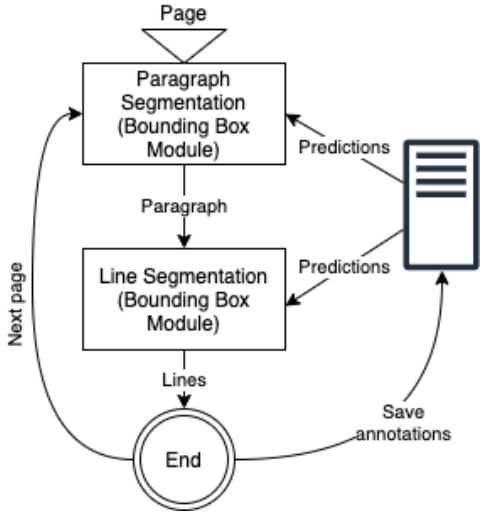


Figure 10: Simplified state machine for paragraph and line segmentation

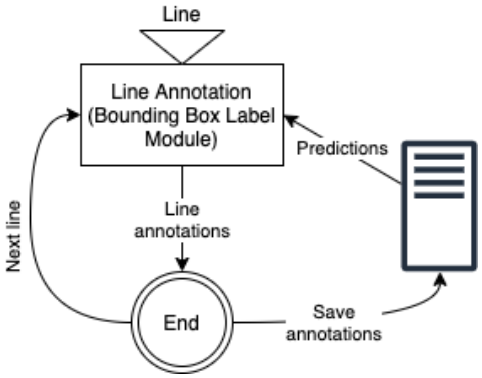


Figure 11: Simplified state machine for line annotations

Paragraph and line segmentation For this task the server would first choose an unlabeled page to show to the annotator. This page would be displayed in the *Bounding Box Module* with a prediction for its location being provided by the server (see Figure 7). The annotator can then correct the bounding box and move to the next module, another *Bounding Box Module*. In the transition, the page image is cropped to the annotated paragraph bounding box and predictions for line segmentations inside of this paragraph are computed by the server and displayed in the module. These again can be corrected by the annotator (see Figure 8). When finished, the segmented paragraphs and lines are sent to the server and saved in the database. The server then provides a new page until none are left. A Graph of the simplified state machine for this task can be seen in Figure 10.

Line annotation For line annotation (see Figure 11), the server chooses an unlabeled line from its database. A previously segmented paragraph together with a bounding box for the line and optional labels for the line are sent by the server and displayed in

the *Bounding Box Label Module* (see Figure 9). After the labels for this single line are added or corrected by the annotator, the line labels are sent to the server and saved in the database.

Segmentation and annotation We can also append the line annotation task to the paragraph and line segmentation task. For this last task the annotator first does paragraph and line segmentation as in the **paragraph and line segmentation task**. The resulting paragraph and lines however then serve as input for the **line annotation task**. For this we iterate over the segmented lines, compute predictions for each and one by one display them in the *Bounding Box Label Module*. After all lines are labeled, the segmentations and line annotations are saved on the server. For the state machine refer to Figure 12

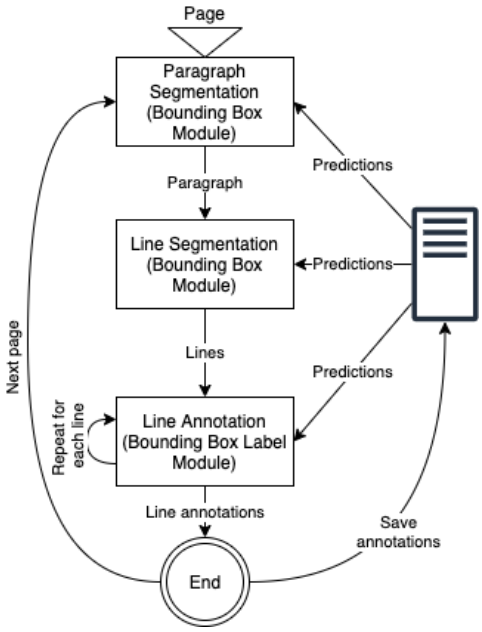


Figure 12: Simplified state machine for paragraph and line segmentation and line annotations

4.3. Conclusion

In this chapter HUMAN, a tool for digitization and annotation, was presented. It consists of modules that can be chained together via a deterministic state machine for flexibility and allows easy use in a human-in-the-loop machine learning context. To use HUMAN for HTR annotations and digitization in general, two new modules were implemented and additional functionalities for timings were implemented. Three setups were shown which use these modules and were later used in the experiments in Chapter 6.

5. The Handwritten Text Recognition System Pylaia

In my experiments I used the HTR system *Pylaia*. In the following I first present the architecture and overall functionality of the system. Then the necessary modifications to make Pylaia suitable for Active Learning are described.

5.1. Architecture

A visualisation of the architecture can be seen in Figure 13.

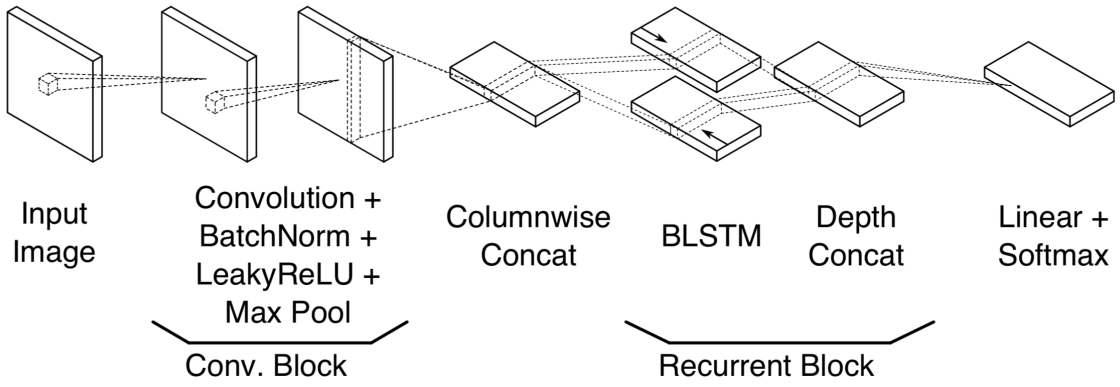


Figure 13: Architecture of Pylaia (from Puigcerver [25]). It consists of 5 Convolutional Blocks which process pixel frames in an input image reducing dimensionality in each block. These are fed into 5 Recurrent Blocks which process the frame sequences into a CTC output matrix.

The Pylaia model consists of 5 convolutional Blocks followed by 5 recurrent blocks. In our configuration, a convolutional block contains a two-dimensional convolutional layer with a kernel size of 3×3 pixels, with both horizontal and vertical stride of 1 pixel. The activation function for Batch normalization [16] is used after the convolutional layer in order to normalize the inputs to the Leaky Rectifier Linear Units [19] (LeakyReLU) activation function. Finally, the output is fed to a Maximum Pooling layer (Maxpool) witch reduces the dimensionality of the input images.

The recurrent blocks are formed by bidi-rectional 1D-LSTM layers, that process the input image columnwise in left-to- right and right-to-left order. The output of the two directions is concatenated depth-wise. Finally each column is mapped to an output label, so the output is a CTC matrix containing a label for each image frame.

To finally decode the output CTC matrix, Pylaia uses Greedy Best First Search as described in Chapter 2.

5.2. Modifications

To make Pylaia suitable for Active Learning with uncertainty sampling, some adjustments had to be made to its decoding step. First, since the Greedy Best First Decoder did not output the proabilities I computed the log probabilities and added them to the output.

However, there is a problem with using the probabilities of CTC as is: The length of a sequence influences the probability. The longer the sequence, inevitably the lower the probability will be. To actually compare the path probabilities with each other to compute uncertainty we have to normalize them for length.

As the model's output is CTC as described above converts the feature sequence X of a picture into N character sequences C. Then the length normalized probability is:

$$P^{LN}(\boldsymbol{C}_i|\boldsymbol{X}) = P(\boldsymbol{C}_i|\boldsymbol{X})^{1/length(\boldsymbol{C}_i)} \quad \forall i = 1, 2, ..., \boldsymbol{N} \tag{2}$$

Note, that when when we use logarithmic probabilities in the CTC matrix, the formula is more efficient:

$$ln\, P^{LN}(\boldsymbol{C}_i|\boldsymbol{X}) = \frac{ln\, P(\boldsymbol{C}_i|\boldsymbol{X})}{length(\boldsymbol{C}_i)} \quad \forall i = 1, 2, ..., \boldsymbol{N} \tag{3}$$

In a small preliminary experiment I found, that beam search combined with a Language Model could improve the final labeling accuracy considerably. Therefore an additional Decoder was implemented, which uses the previously described CTCWordBeamSearch by Scheidl et al. [28]. Both this Beamsearch-Decoder and the GBF-Decoder were used in later experiments.

5.3. Conclusion

In this Chapter the architecture of the Pylaia HTR system was introduced. Then the necessary modifications were presented, that were implemented into this system to enable use for AL. This system then was used in experiments to find answers to our research questions

6. Experiments

To answer the research questions defined in Chapter 1 four experiments were performed.

The first question was, whether a model trained with Active Learning produces better suggestions in each iteration when using uncertainty sampling than with random sampling. For this, we annotated samples of the paquid dataset chosen by uncertainty sampling following the human-in-the-loop paradigm. in an experiment we call *uncertainty sampling experiment* (see Section 6.2). Then to compare this, the *random sampling experiment* (see Section 6.3) followed a similar approach but the samples were chosen randomly. In this experiment there were no human annotations, as the samples were already labeled in the uncertainty sampling experiment.

However to train a model in the first place, we need a labeled validation dataset and to be able to see if the model generalizes well for unseen data, we need a labeled test dataset. Therefore a prerequisite was to first annotate a part of the Paquid data for this purpose. A baseline model, which was trained on other handwritten text than Paquid was trained. This model then was used to produce suggestions for annotating the validation and test sets in the *baseline experiment* (see Section 6.1). This annotation process is then used to answer the second research question - whether suggestions predicted with uncertainty sampling make the annotation process faster than when using an adapted model as a baseline.

For the final question of where the AL models converge compared to non-incremental learning methods, two models were trained with all labeled data from the uncertainty sampling experiment at once.

In the following we introduce the used metrics and the baseline model and then discuss the annotation process with the baseline model. Afterwards the annotations with the uncertainty sampling method are described, followed by the random sampling experiment. Finally we show where the models converged.

Metrics Common metrics for an HTR system's performance are Word Error Rate (WER) and Character Error Rate (CER). WER and CER are derived from the Levenshtein distance. It is defined as:

$$\mathrm{WER} = \frac{S + D + I}{N} \tag{4}$$

where S is the number of substitutions, D is the number of deletions and I the number of insertions that are needed to transform a list of words into another. N is the number of words in the target string. Note that conventionally deletion and insertion refer to the way to get from the ground truth to the prediction. For an example, if the ground truth is "cool cat" and the prediction is "cat", then the corresponding edit-operation is a deletion. Normally punctuation symbols are excluded for WER but for this thesis they are tokenized and thus count as words.

CER works analogous to WER but instead, the operations to transform a *character* string into another one are used, with N being the number of characters.

As an example, to transform the string "Batts are cool ." into "Cats are cool .", on a word level only one substitution ("Batts" $\rightarrow$ "Cats") is needed so the WER would be $1/4 = 0.25$. On a character level however we need one substitution ("B" $\rightarrow$ "C") and one deletion (one "t") resulting in a CER of $2/15 \approx 0.13$.

For predictions for annotations additionally to WER and CER, accuracy and the time needed to complete the annotation were measured.

Baseline Model The Baseline model was trained with the Pylaia HTR-system on the IAM-Dataset[21] with the configuration and splits taken from the *IAM-HTR* example[4]. The IAM-dataset is composed of 1539 scanned text pages, which are handwritten by 657 different writers and segmented into paragraphs and lines. The 10042 lines were split into training, validation and test-sets of 6161, 966 and 2915 lines, respectively.

The optimizer used in training was RMSProp with an initial learning rate of 0.0003. To prevent overfitting, early stopping after 20 epochs of not improving Validation-CER was used. It trained for 119 epochs in total with the model at 99 epochs giving the best results on both validation and test sets. The model achieved a WER of 0.267 and a CER of 0.077 on the IAM test dataset.

[4]https://github.com/carmocca/PyLaia-examples/tree/0a9a2d0934ec1266350761da5b4b054cb4605193/iam-htr

6.1. Baseline experiment

In this experiment the baseline model was used to predict the transcriptions of line images from the Paquid Dataset. The motivation for this experiment was threefold: *i*) to use the resulting annotations as the validation and test datasets in later experiments. *ii*) to make the annotators used to the tool and *iii*) to get a baseline for how much time is needed at most for annotations to calculate the amount of data that could be annotated during this thesis.

6.1.1. Design

Data During preprocessing of Paquid for 300 pages, the paragraphs and lines were segmented semi-automatically by guessing which segmentation would be correct for most documents of a certain type. These 300 were presented via HUMAN with the **segmentation and annotation** setup to the annotators (see Section 6.2.1). With this setup, first the paragraph and line segmentations of a document are corrected, then the individual lines' text is transcribed one by one.

For the label predictions, the images of the lines were extracted and enhanced with imgtxtenh [32] a tool to clean noisy scanned images. The baseline model then transcribed these enhanced images and the predictions were used as suggestions in HUMAN. This approach soon showed a major drawback: The predictions were nearly unusable when the automatic segmentation did not capture the lines well. Because of this it was decided that the segmentations should be done in a separate step before the text annotations in the next experiment.

Annotation There were three annotators including the author (in the following graphs and tables named User 1). The other two annotators were research assistants employed at DFKI. Every document was annotated only once, to save time. Since documents were not split equally between the annotators, each user saw a different amount of documents. The amount of documents, lines and words each user worked on can be seen in Table 1.

6.1.2. Results

Model performance The accuracy of the model predictions was 0.048 with a WER of 1.016 and a CER of 0.725. The main reason for this was a large underproduction of characters. This is most obvious when looking at the number of edit operations: While there were 14555 substitutions, there were 8863 deletions and 2257 insertions. A high number of substitutions is expected when the model's accuracy is low, but the high number of deletions especially compared to insertions shows this underproduction. This may sound counter-intuitiv at first, but as a reminder, deletion refers to how we get from the ground truth to the prediction. The underproduction however was mostly due to the initial line segmentations often capturing empty lines or spaces between lines. When a line was segmented correctly, actually the opposite was the case — then the model recognized more characters than actually were written in that line.

User	Documents	Lines	Words
1	164	528	4263
2	64	204	1512
3	69	202	1678
All	297	934	7453

Table 1: Amount of annotated documents and their contained lines and words for each user. User 1 is the author.

Annotation Times The completion of this task took 18:40 hours overall. Of this time the segmentations took 2:47 hours and the text annotations 15:53 hours in total.

Figure 14 shows the times each user took for annotations. The absolute times have to be taken with a grain of salt however, because times varied significantly with the difficulty of the task. This is why the figure shows the mean over batches of 15 annotations, to normalize for this effect. The individual times per line were also capped at 10 minutes to eliminate the cases where annotators left the annotation tool open while doing something else. We can see signs that the annotators learned how to read the handwritten text and use the annotation tool efficiently in the trendlines which decrease for all users be it at different rates. This effect is sometimes also known as "priming". However we also see that the times did not converge yet so it is possible that the priming could further decrease times with following annotations.

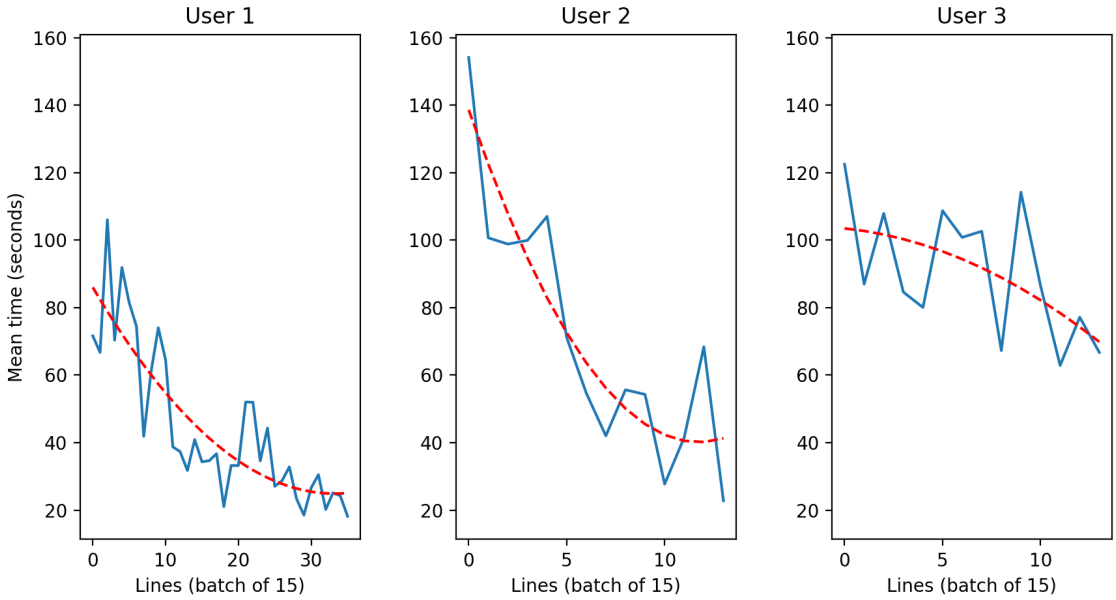


Figure 14: Time needed to label one line normalized over batches of 15 lines in the baseline experiment. Red dotted line is a trend line. User 1 is the author.

6.2. Uncertainty sampling experiment

In the second experiment we follow the human-in-the-loop paradigm to train a HTR model which assists a human in the annotation process. The model produces incrementally better by choosing which samples the human will annotate next via uncertainty sampling. This experiment will be called "uncertainty sampling experiment" in the following.

6.2.1. Design

Figure 15 shows the process of the uncertainty sampling experiment. It closely follows the human-in-the-loop process outlined in Chapter 2.

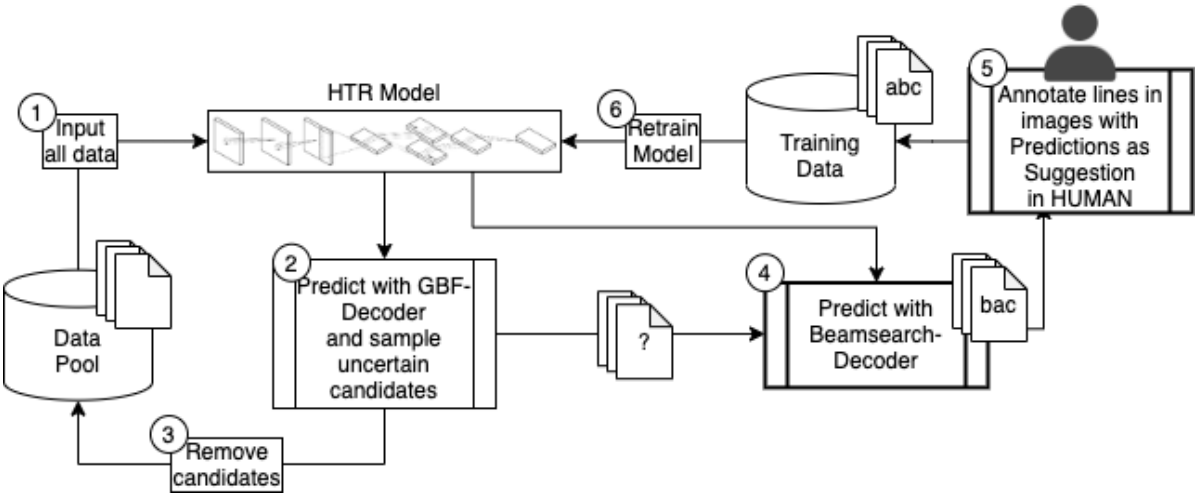


Figure 15: Outline of the uncertainty sampling experiment: (1) Predict labels for all unlabeled data in the data pool. (2) Model produces predictions with GBF Decoder. Choose 300 candidates via uncertainty sampling. (3) Remove the chosen candidates from data pool. (4) Model produces predictions with Beamsearch Decoder. (5) Human annotator corrects the predictions in HUMAN. (6) The corrected data gets appended to the training data. The model retrains with this data. Repeat from (1) until data pool is empty.

One key difference to the previously described method is, that the predictions from the model are shown to the annotator as suggestions. The annotators were the same as in the previous experiment. The other difference is that in this experiment, we have two sets of predictions for two different uses. The first set follows the human-in-the-loop process and are predictions on all unlabeled data in the data pool with the fast GBF-Decoder. These predictions are used to sample uncertain candidates. These uncertain candidates are then predicted again, this time with the Beamsearch-Decoder which produced better results but is considerably slower. Speed was one reason for this approach, but the main reason was, that the model was trained and validated with the GBF Decoded. The reasoning was, that the uncertainty sampling results would be more meaningful this way, without any influences of a language model. As the uncertainty sampling method, Least Confidence sampling was used, meaning that we chose 300 samples, for which the prediction had the lowest probability. The baseline model was used as a starting point for the first iteration, incrementally fine tuning it on Paquid data.

The pre-processed data described in Chapter 3.1 had to be further processed before they could be used for AL. More precisely, there were still 1700 pages intended as

the data pool, for which there were no paragraph and line segmentations yet. For the paragraph and line segmentations, the *paragraph and line segmentation* setup in HUMAN were used. These segmentations took around 21 hours for the three annotators. Afterwards, the line images were again extracted with ImageMagick and enhanced with imgtxtxenh [32]. Afterwards the resulting line images were split into data pool and validation- and test datasets. The samples from the data pool are simultaneously the training data when labeled and will be referred to as such in the following. The amount of pages and lines in each dataset split can be seen in Table 2

Split	Pages	Lines
Data pool/Training	1693	5168
Validation	100	312
Test	197	613

Table 2: Amount of pages and lines in each dataset split.

6.2.2. Results

Training progress We can see the training progress in Figure 16. The impact of each new batch of training data clearly shows in the drop of validation CER. In later iterations however, this impact gets smaller and smaller, which also is reflected in the fact that around epoch 450, early stopping would stop training after one iteration because the validation CER would not drop further. Noteworthy is also, that we see signs of overfitting in every iteration in validation loss slowly increasing in each iteration cycle. This however must not necessarily be a bad thing, as we see both validation CER and test WER and CER fall despite this fact. The loss's slope also becomes less steep with larger training data sizes in later iterations.

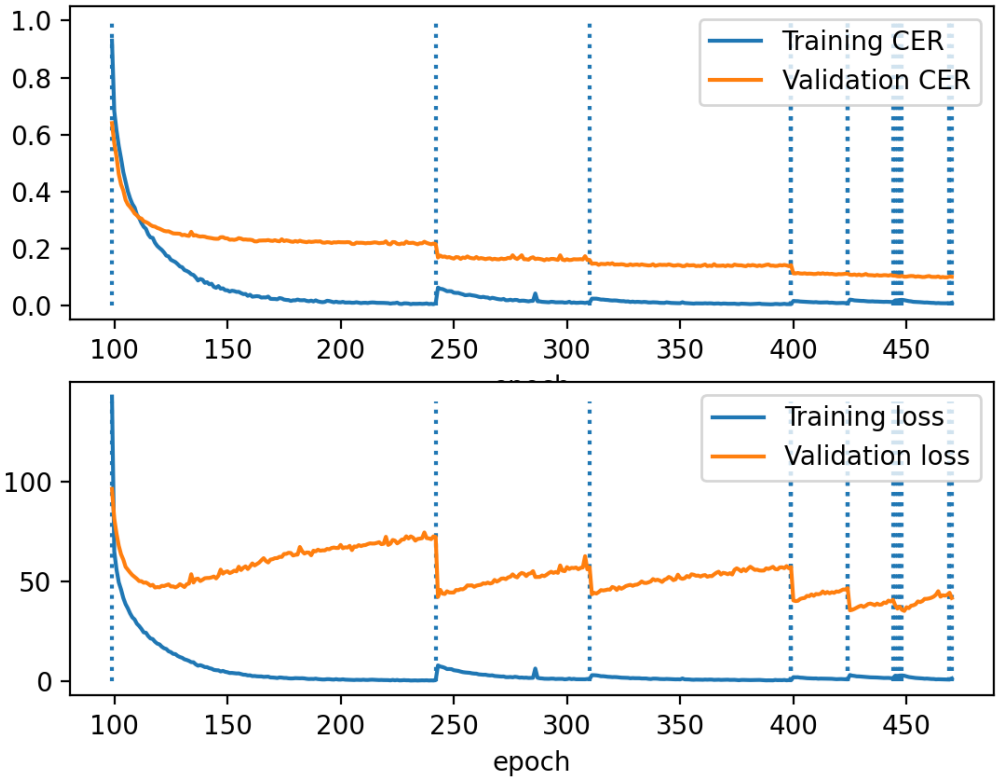


Figure 16: HTR Model's training and validation loss and CER during training in the uncertainty sampling experiment. Dotted lines show the epochs where models were used for predictions in each iteration. The thicker line around epoch 450 shows multiple iterations.

Performance on test data In this experiment, the Beamsearch-Decoder consistently outperformed the GBF-Decoder in WER. Figure 17 shows the WER on the test dataset in each iteration and illustrates well the advantages of the Beamsearch-Decoder.

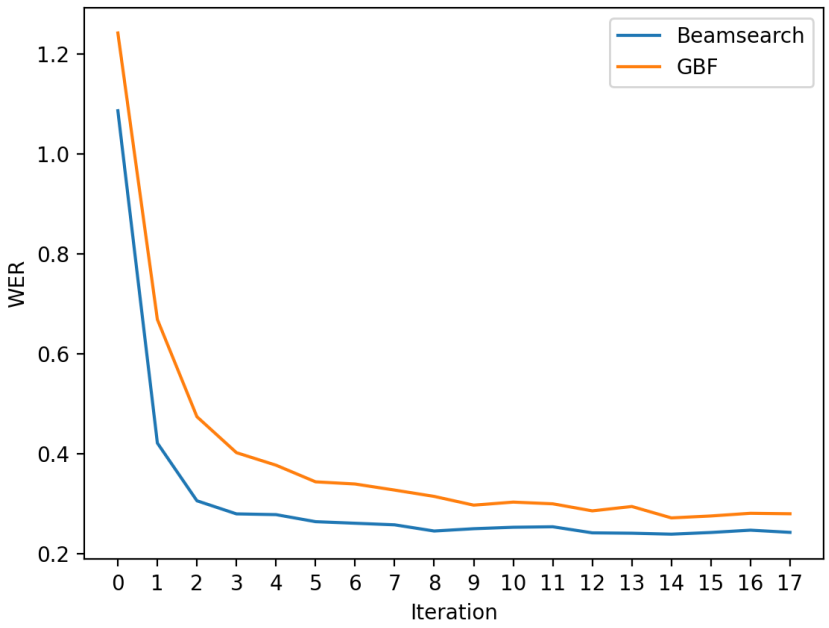


Figure 17: Comparison between GBF and Beamsearch decoders on test data in the uncertainty sampling experiment.

Figure 18 shows the WER in each iteration in context with the WER edit-operations. We see the test WER drop sharply in the beginning and relatively fast even out at around 27%. The noticeable discrepancy between insertions and deletions mostly comes from inconsistent placement of parenthesis in the test data. If the suggestion is "(chat)" but the ground truth is "(chat)" then there is one substitutions operation from "(chat)" to "chat" and two insertion operations for the parenthesis. This is why we do not see this effect for Test CER in Figure 19.

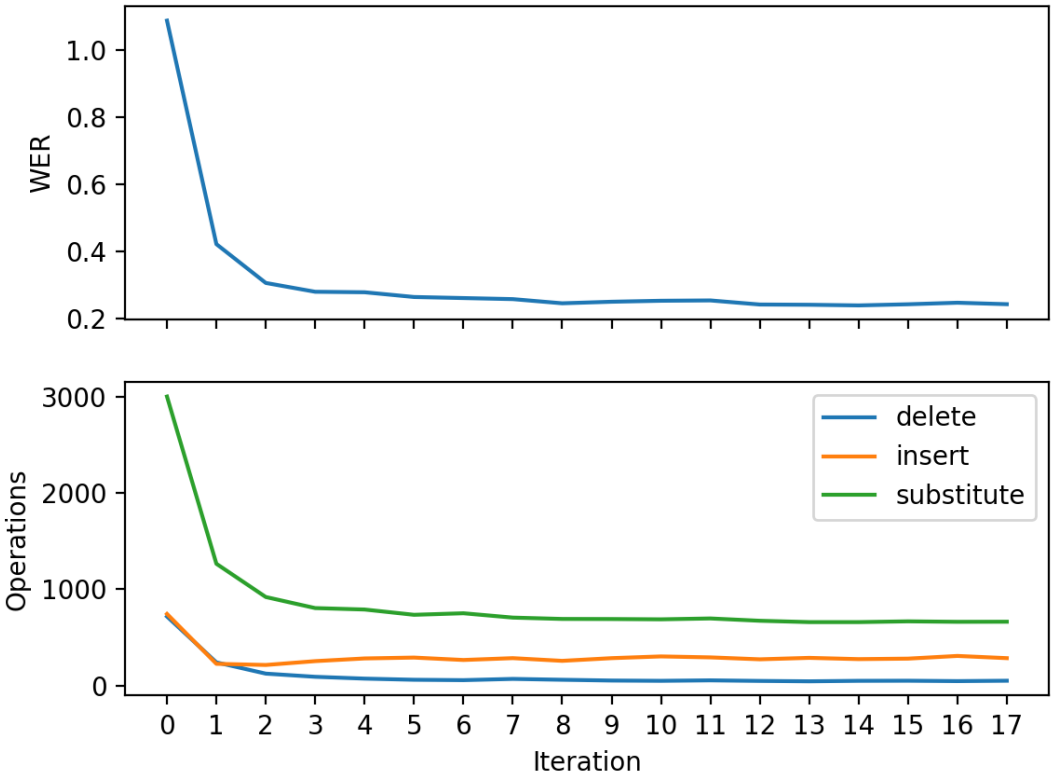


Figure 18: WER and the WER edit-operations on testdata for each iteration in the uncertainty sampling experiment.

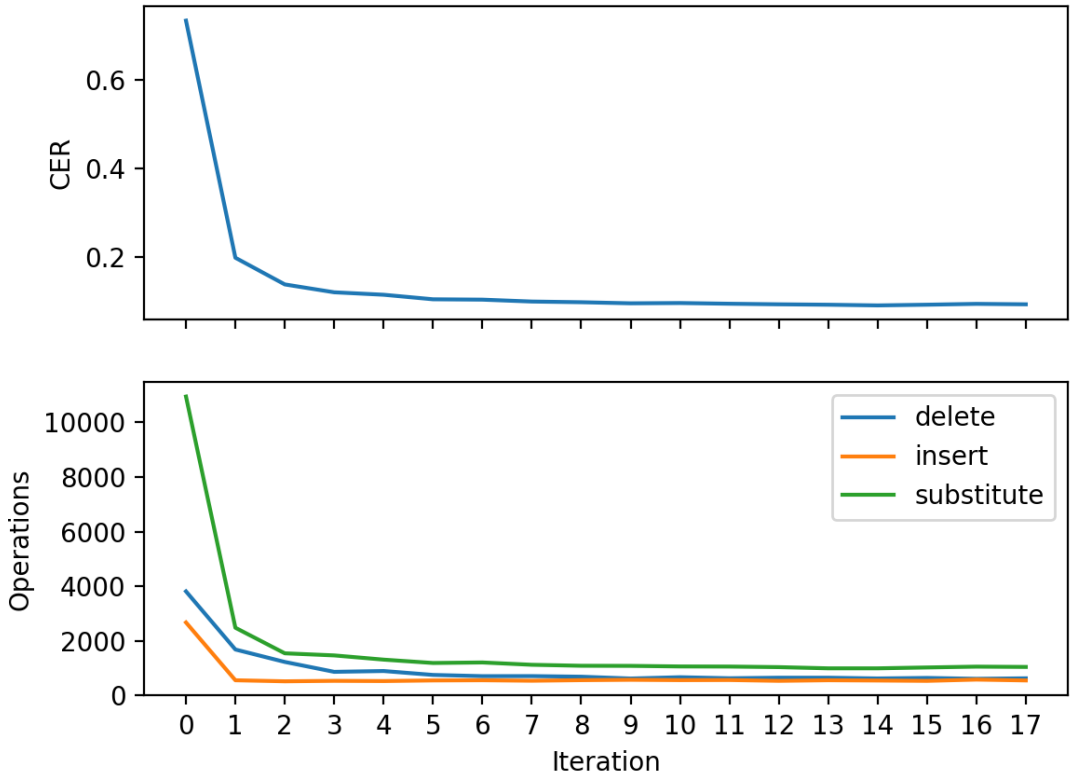


Figure 19: CER and the CER edit-operations on testdata for each iteration in the uncertatinty sampling experiment.

Predictions for annotations Next we take a look at the performance of the predictions that were used as suggestions in the annotation process. In Figure 20 we see the accuracy and WER of the suggestions.

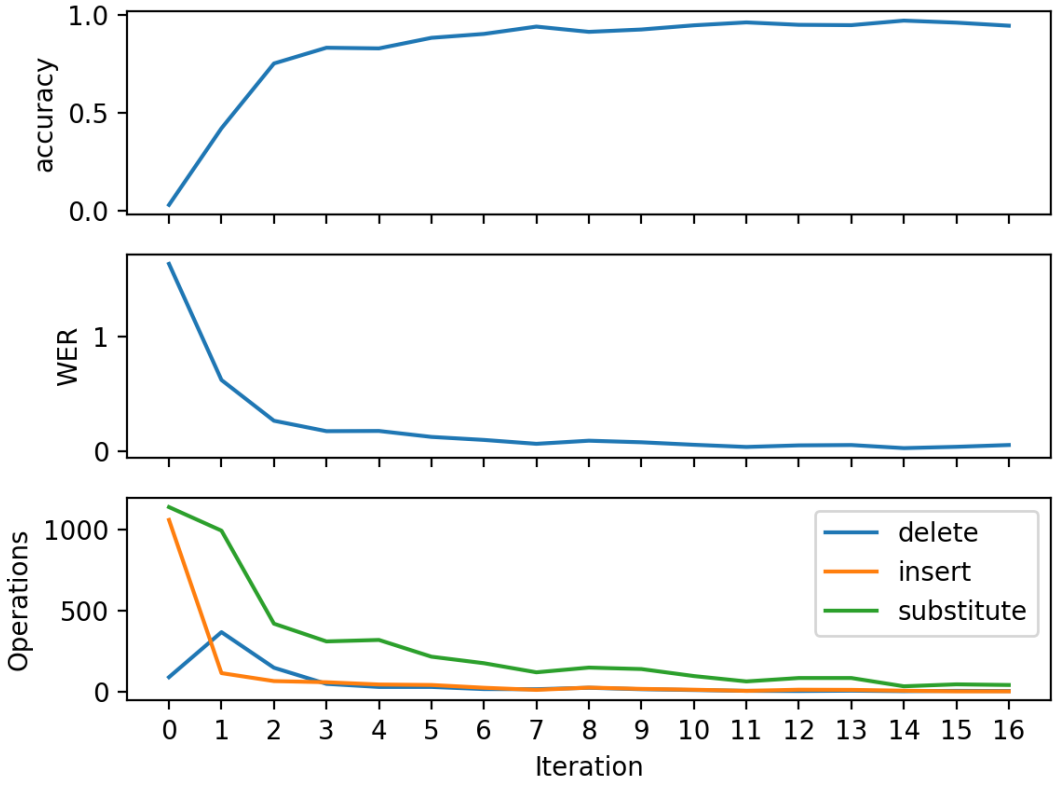


Figure 20: Accuracy, WER and WER edit-operations for predictions for annotations in the uncertainty sampling experiment.

We can see a steady drop in WER in the beginning after which it slowly converges at around 0.5%. We see that the accuracy is closely correlated and rises as WER drops peaking at around 95% after 13 iterations after which it converges. We see here, that WER is a strong indicator for how high the accuracy and correspondingly how high the annotation effort is. Noteworthy is the extremely high WER of 106% in the first iteration. If we take a look at the edit-operations, we see that this is not only due to the high number of substitutions - which is expected - but the nearly equally high number of insertions. Equally noteworthy is that in the second iteration there are nearly no insertions anymore but we see a rise in deletions, after which both the number insertions and deletions becomes negligibly small. If we take a look at example predictions and annotations (see Figures 21 and 22) for iteration 0 and 1, we see why this is the case. In iteration 0, the model massively overproduces words - in the example there are only two words in the ground truth, but eight in the prediction. However in iteration 1 we

see that the model now slightly underproduces because valid words get merged with numbers in between. We can also see the model starting to recognise mostly correct substrings of words, however they are often slightly misaligned.

Prediction	ibis	ouistiti	uf / rat	uf	mite	Hermine
Annotation	/	poule				

Figure 21: Example predictions and annotations for iteration 0

Prediction	...	isard9pinson	atele9tourterelle	isard	
Annotation	...	lapin	cheval	vache	taureau

Figure 22: Example predictions and annotations for iteration 1. Similar substrings in prediction and annotation are colored the same

A mutual feeling of the annotators was, that these first two iterations seemed to be the most difficult to decipher of all iterations. This could be, caused by the low accuracy, by annotators not being proficient in reading the different handwritings yet and can be counted as evidence, that uncertainty sampling chose lines that were difficult to decode for the HTR model as well as for humans too.

Annotation durations This perceived difficulty of the first iterations can also be seen in the mean times annotators needed for each iteration (see Figure 23)

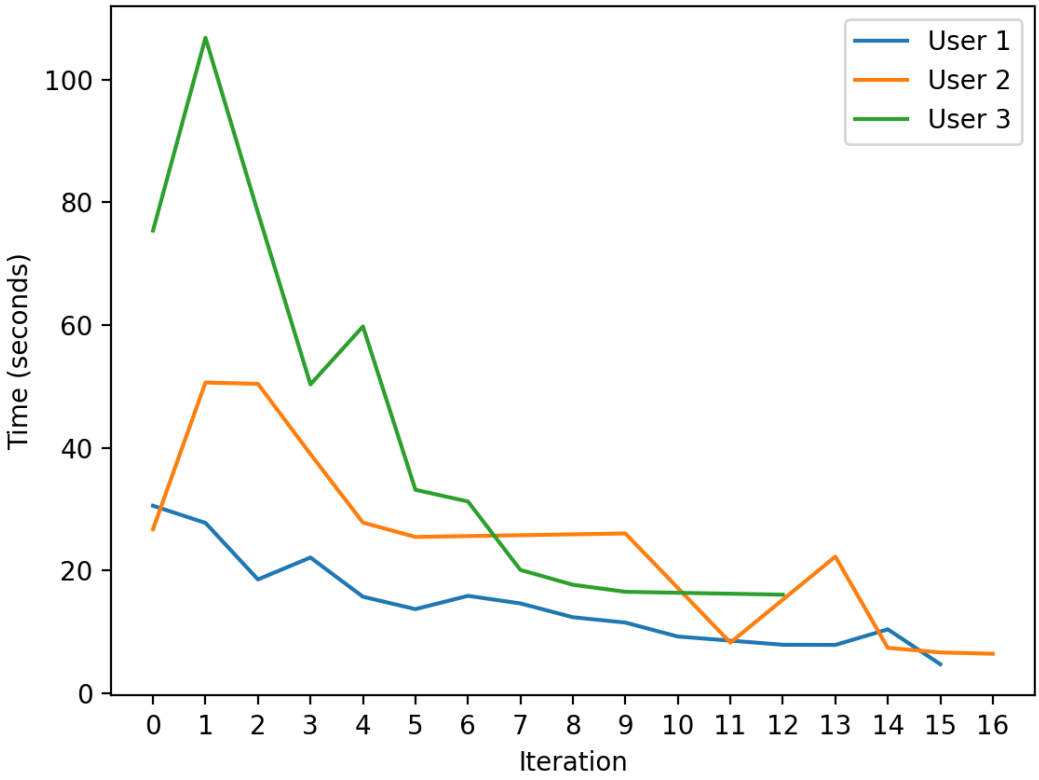


Figure 23: Mean time per line in seconds for each user. User 1 is the author.

The times are highest in the first few iterations and fall rapidly as the accuracy rises. The relation between time and accuracy can be seen in Figure 24. We see the times decrease relatively steadily over the course of the experiment, while the accuracy rises massively in the beginning and then quickly evens out. This could be a sign, that the annotators were still learning how to best perform the annotation task and makes it unfortunately very hard to say how better suggestions had a benefit on the annotation process.

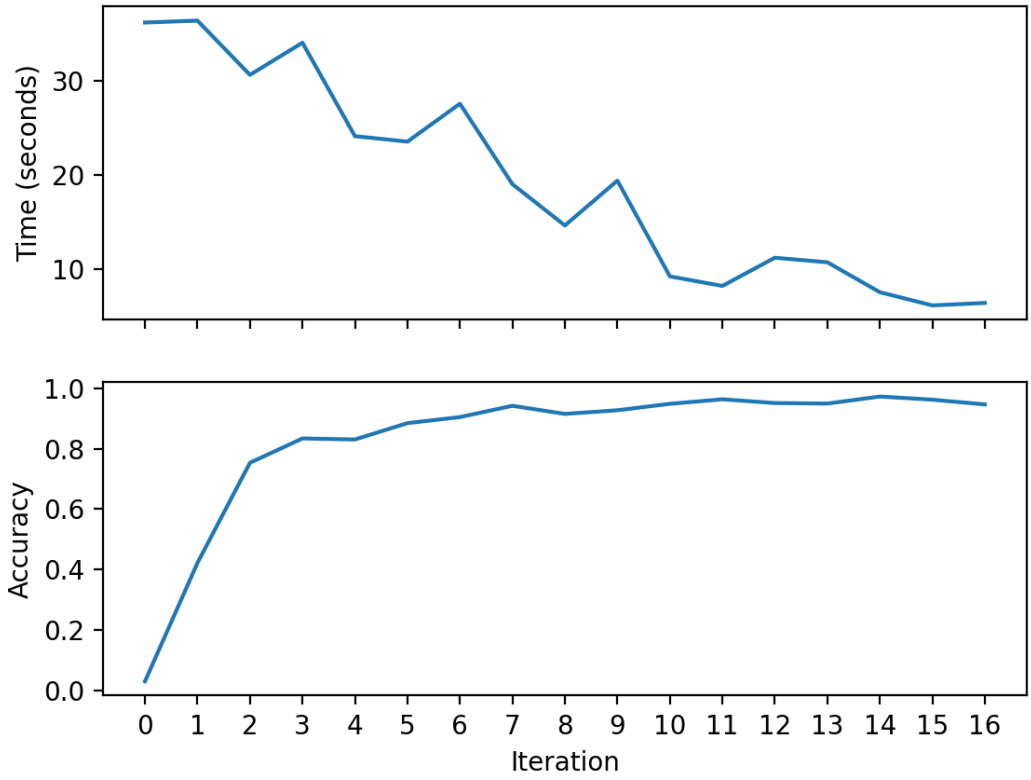


Figure 24: Accuracy of suggestions and mean time of all users per iteration

6.3. Random sampling experiment

To compare how Active Learning with uncertainty sampling behaves in comparison to random sampling, a third experiment was conducted where the annotations were simulated.

6.3.1. Design

The architecture for the random sampling experiment is essentially the same as described in Figure 15 for the uncertainty sampling experiment. There are however two differences: In step (2) the candidates are sampled randomly instead of finding uncertain ones. Secondly, since for comparison the datapool had to be the same as in the uncertainty

sampling experiment and the data was already annotated, the annotation process was simulated and only the WER and CER on the new sampled lines in each iteration were calculated.

6.3.2. Results

This experiment was conducted three times in total to see how random sampling performs on average. Figure 25 shows the WER of all random sampling models on the test set using the GBF-Decoder in each iteration. We can see, that there is little variation between the models overall, so for the rest of the evaluation we only consider the mean over the values in each iteration.

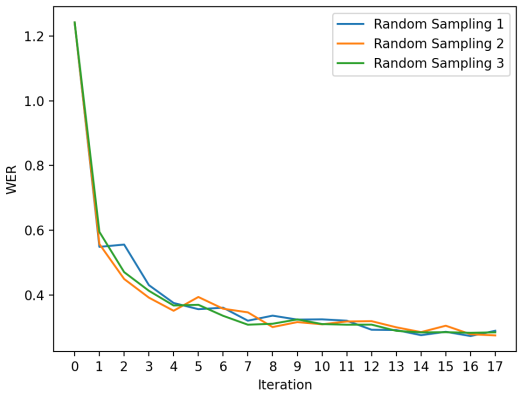
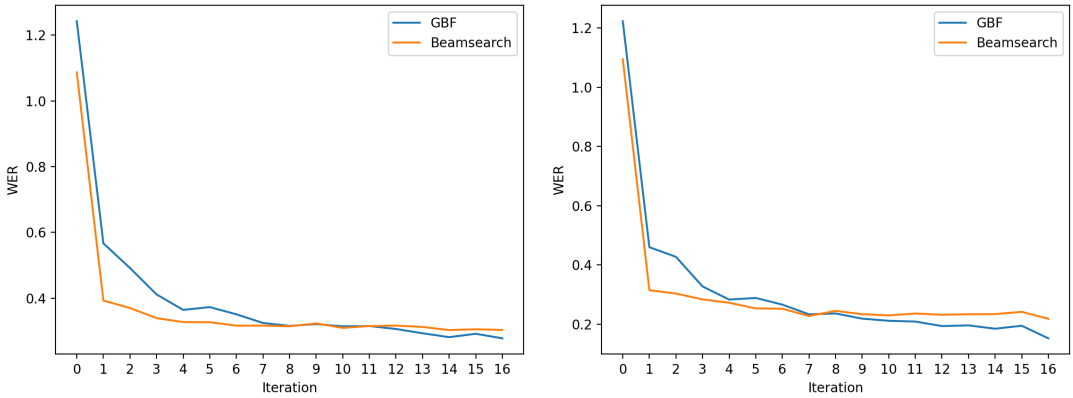


Figure 25: WER on test data for all Random Sampling experiments using the GBF-Decoder.

In Figure 26 we compare the mean WER of the Random Sampling models between GBF-Decoder and Beamsearch-Decoder.



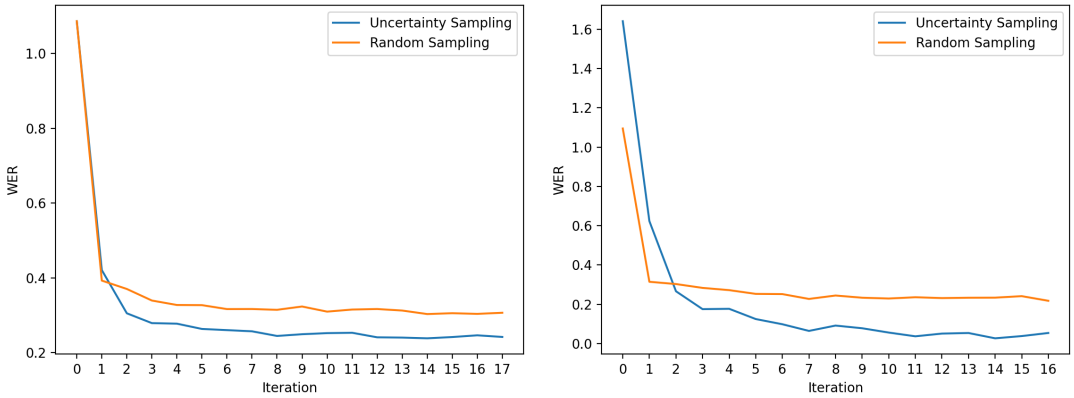
(a) Test (b) Suggestions

Figure 26: Mean WER per Iteration between Decoders using Random Sampling.

We see that the Beamsearch-Decoder performs well for the first few epochs, but eventually, starting from iteration 7, it gets outperformed by the GBF-Decoder. The effect is

apparent on both the predictions on the test set and in the produced suggestions. This is in contrast to uncertainty sampling, were the Beamsearch-Decoder always outperformed GBF.

In Figure 27 we compare the mean WER of the Random Sampling models with the uncertainty sampling experiment. We see that the WER of the suggestions was a lot better with Random sampling for the first two epochs. This could show, uncertainty sampling chose mostly difficult lines in the beginning, as on test data, random sampling performs only slightly better than uncertainty sampling. Both methods level off fast after around four iterations, however the uncertainty sampling converges at around 27% on test while random sampling converges to 35%. The WER for suggestions is similar - random sampling levels off at around 30% while uncertainty sampling converges to 15%. In this comparison both methods use the Beamsearch-Decoder.



(a) Test (b) Suggestions

Figure 27: Comparison of mean WER per iteration between random sampling and uncertainty sampling using the Beamsearch Decoder.

However, if we compare the WER of uncertainty sampling and random sampling when both use the GBF-Decoder on the test data 28, we find that they perform nearly the same throughout.

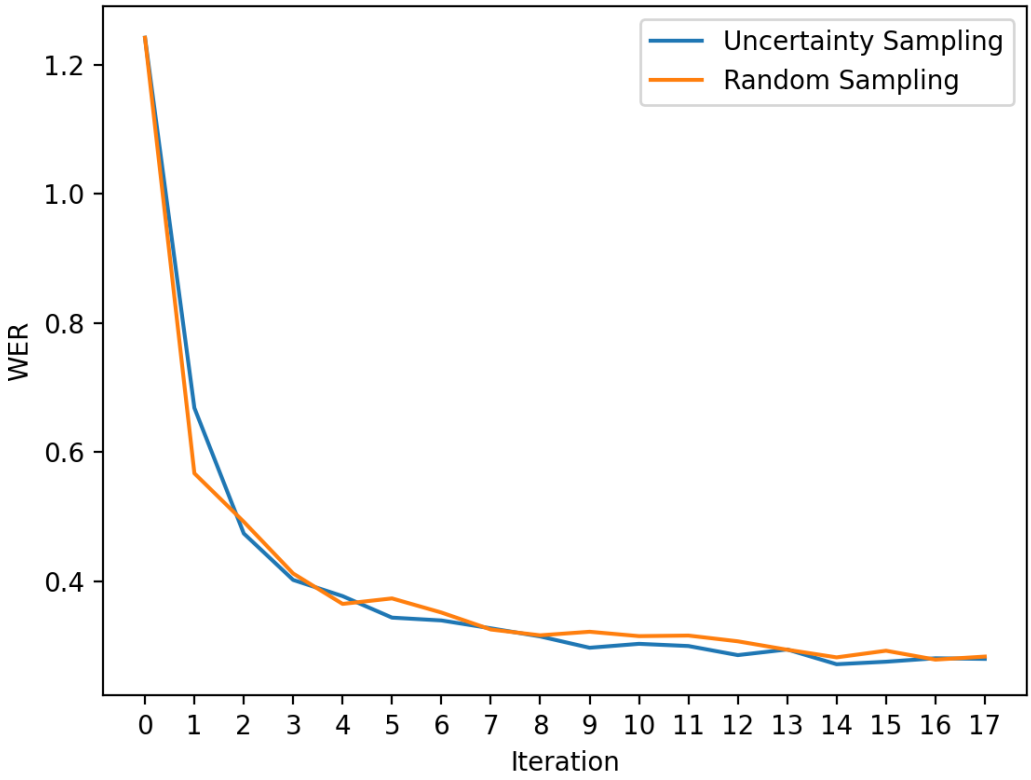


Figure 28: Mean WER on test data of Random Sampling experiments and WER of Uncertainty Sampling, both using the GBF-Decoder.

6.4. Model Convergence

In the final experiment we explore how incremental learning compares to learning conventionally with a fixed training set. For this two more model were trained with all training data - one using Transfer Learning and one training from scratch. By comparing these two models we can also explore the effect of TL on model performance. The model settings again were the same as in all experiments. The baseline model trained on the IAM-dataset again served as the base model for Transfer Learning.

6.4.1. Results

The model trained from scratch with all training data at once, trained for 86 epochs and achieved a WER of 27% on the test data. The transfer model which was pretrained on the IAM-dataset also continued training with all training data at once and achieved a WER of 26%. The massive improvement that fine tuning brought can be seen if we compare this with the WER of 108% the Baseline model which was not fine tuned at all. The Error Rates of all best models of each method tested in the experiments can be seen in Figure 3

	WER	**CER**
Baseline	1.086	0.734
From scratch	0.276	0.127
Transfer Learning	0.265	0.119
Random sampling best	0.275†	0.094†
Random sampling mean	0.278†	0.102†
Uncertainty sampling	**0.239**	**0.091**

Table 3: Comparison of the performances of the best model on test data for each method. †: achieved with GBF-Decoder

We find, that the best model iteration of training with AL with uncertainty sampling outperformed all other methods. Random sampling converged at a WER similar to when learning from scratch, but the CER was slightly better than with all non-incremental methods and more comparable to uncertainty sampling.

7. Discussion

With the results of the experiments we can return to the research questions and consider what evidence we have to answer them.

Question 1: Does an HTR model trained with Active Learning produce better suggestions in each iteration when using Uncertainty sampling than with Random sampling? In our experiments Active Learning with uncertainty sampling consistently produced better suggestions than random sampling when using the Beamsearch-Decoder. However their performance with the GBF-Decoder was near identical. This raises the question, why beamsearch performed better in one case but not the other. We can speculate that the reason for this has to do with the change in entropy in the predictions caused by uncertainty sampling. Either, there is a higher entropy when using uncertainty sampling and therefore beam search has more chance to deviate and choose other paths than GBF. This however would be very unexpected, as it would mean that uncertainty sampling had the opposite effect than what was intended and made the model more uncertain. Or, the second possibility, there is a higher prediction entropy when using random sampling. This could lead to beam search choosing predominately wrong paths as the language model scores would have a lot more influence.

To note is also, that in the context of these experiments AL is not used in the proper sense. AL is designed to sample only a few lines that are helpful for the model from a very big data pool, but in the experiment we exhaustively sampled from a relatively small predefined set of lines until it was annotated completely. This exhaustive sampling in all likelihood did not result in the best possible models for the task but had a practical reason: Segmenting all Paquid data or training a segmentation model would have been too time consuming and was not possible within the constraints of this thesis. We can conclude, that uncertainty sampling outperformed random sampling and could probably even perform better when sampling from a bigger data pool.

Question 2: Do suggestions predicted by an HTR model trained with Active Learning with uncertainty sampling make the annotation process faster compared to using only a pretrained model? We found, that the times needed for annotation fell rapidly for each annotator after the first few iterations when using AL, so the annotation process got indeed faster. Also compared with the annotations with suggestions from the baseline model, there was a considerable speedup from around 30, 40 and 80 seconds to 10, 15 and 20 seconds for each user respectively.

However we have to take these values with a grain of salt, for multiple reasons: First, for the experiment with the baseline model, the annotation task was slightly different, as complete page was first segmented into multiple lines, which then all got annotated

in one sweep. For the uncertainty sampling experiment these steps had to be separated. Also the times in the experiment with the baseline model did not converge. This means that times in the later experiment could still go down because the annotators just got more proficient to read the hand-writings. Thirdly not all annotators did equally many annotations in each iteration - sometimes one annotator even did none for a few iterations, especially in the end when the annotation process was really fast.

Question 3: Does an HTR model trained with AL converge at a better performance than non-incremental learning methods? The final experiment showed that the best models from both AL variants performed better than the models trained with non-incremental methods. However the best model trained with uncertainty sampling performed slightly better than when random sampling. Also the TL model outperformed the model trained from scratch again confirming previous findings that Transfer Learning benefits an HTR system's performance.

8. Conclusion and Future Work

The Active Learning strategy in general proved a complete success. Both incremental strategies produced better suggestions and generalized better on unseen data than when training non-incrementally from scratch or with transfer learning. Also uncertainty sampling performed nearly equally as random sampling when using the GBF-Decoder. However uncertainty sampling was a lot better when using beam search. This to my knowledge has not been observed before and should be further explored in future research, for example by testing the entropy for the prediction probabilities in each method. Also it is possible that uncertainty sampling would perform better on a larger training pool. This was not possible in the thesis because of the time consuming line segmentation step, but could be explored with the rest of the Paquid dataset. Nevertheless we saw the times needed for annotations more than half for each annotator when supported with suggestions produced with Active Learning. It is however possible, that this speedup could come from external factors, such as the annotators getting more proficient in their task so the results are inconclusive. This could be further explored by eliminating most of the external factors by letting the same annotators label more samples from the Paquid dataset.

We could also confirm previous observations from Aradillas et al. [1] and Cascianelli et al. [7] that a TL model could better generalize over the test data than when training from scratch.

A known problem with confidence based sampling in general is sampling bias. Malhotra et al. [20] used Least Confidence sampling for ASR and observed "that the application of uncertainty sampling to ASR leads to generation of samples that have significant speaker bias in the labeled set, i.e. some of the speaker groups are sampled considerably more than the others". Similarly, in the case of HTR when multiple handwritten styles are observed, it could be the case, that previously unseen writers are predominantly sampled. Because of time constraints in the thesis the properties of the data chosen by uncertainty sampling were not thoroughly explored. Therefore we can not know if there is any such bias for example for a specific writer, document type or the noisiness of the document. This however would be interesting to investigate in the future. If it turns out, that there is such a problem, there are multiple approaches to mitigate sampling bias. Malhotra et al. [20]'s solution was, to implement a speaker recognition algorithm to keep the speaker groups diverse. Another idea is proposed in Munro [23]. First a certain amount of uncertain data is sampled, and then clustered by K-means or a similar clustering algorithm. Then from each cluster a small amount of centroids, outliers and random items are sampled. This adds a certain amount of noise into the sampling process and might be a valid strategy to combat sampling bias.

During annotations for this thesis, there came up many ideas to make the labeling process more efficient and comfortable for the annotators. The most prominent was to introduce other input methods and modalities than mouse and keyboard into the annotation tool. Such a multimodal annotation tool could for example incorporate automatic speech recognition methods so instead of typing a word a person could speak it instead. This could even be combined with an eye tracking unit in place of a mouse. Annotators could then look at a certain word, and read it out loud. Linking the word's location in the picture and the recognized utterance would then result in the annotation, enabling a truly hands-free annotation process. Graphic tablets were used for some of the annotations in this thesis, but they proved only slightly useful for the segmentation tasks. The annotation tool could however be further optimised for their use. Having these various modality options, additionally to mouse and keyboard, could further improve endurance and health of the annotators and with that the quality of the annotations.

Regarding HTR-systems, with Transformer models replacing LSTM in almost every area

of NLP, this will probably be the future for HTR as well. There are first models using an end-to-end approach with the Transformer architecture already [27] - it would be interesting to find out if AL is still a valid option for this new generation of models.

Automatic line segmentations were out of scope for this thesis but they are nevertheless a key part of the HTR process. Parallel to the thesis experiments with layoutparser [30], a toolkit for deep learning based document image analysis. The experiments results were promising, but not sufficiently precise for HTR during the time of the thesis, so it could not be used yet. For this model, again similar human-in-the-loop methods as in this thesis could be used to gradually improve the segmentations. Another interesting approach would also be to have a pipelined system where segmentation and HTR models are trained in conjunction. There again active learning could be employed.

One of the bottlenecks in the experiments was the rather slow implementation of the beam search decoder, however CTCWordBeamSearch [28], got an update near the end of this thesis after which a speedup of around an order of magnitude was measured. With this it may be feasible to experiment with using CTCWordBeamSearch not only for predictions, but also directly in the training process. Truly incorporating beam search into the end-to-end system could considerably improve the performance with human-in-the-loop methods.

In conclusion we could show that Active Learning is a valid strategy to speed up the annotation process considerably, with Uncertainty sampling outperforming random sampling. The best models resulting from Active Learning also generalized better over unseen data than models trained conventionally on all data at once. This makes Active Learning a valid strategy to train models not only to help with annotations, but to fully automatically predict unseen data without human input.

Acknowledgments

I'm extremely grateful to my supervisors Dietrich Klakow and Jan Alexandersson, without whoms discussions this thesis would not have been possible. I would also like to pay my special regards to the annotators, Eva and Max, for their hard work deciphering even the weirdest french animal names. I'm also grateful to DFKI GmbH and the LSV group at Saarland University, for their provision of resources. Finally I'd like to thank my friends and family for their continued support throughout the creation of this thesis.

References

[1] J. C. Aradillas, J. J. Murillo-Fuentes, and P. M. Olmos. "Boosting handwriting text recognition in small databases with transfer learning". In: *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR* (2018), pp. 429–434. ISSN: 21676453. DOI: 10.1109/ICFHR-2018.2018.00081. arXiv: 1804.01527.

[2] T. Baur, A. Heimerl, F. Lingenfelser, J. Wagner, M. F. Valstar, B. Schuller, and E. André. "eXplainable Cooperative Machine Learning with NOVA". In: *KI - Künstliche Intelligenz* 34.2 (June 2020), pp. 143–164. ISSN: 0933-1875. DOI: 10.1007/s13218-020-00632-3. URL: https://doi.org/10.1007/s13218-020-00632-3%20http://link.springer.com/10.1007/s13218-020-00632-3.

[3] A.-L. Bianne-Bernard, F. Menasri, R. Al-Hajj Mohamad, C. Mokbel, C. Kermorvant, and L. Likforman-Sulem. "Dynamic and Contextual Information in HMM Modeling for Handwritten Word Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.10 (2011), pp. 2066–2080. DOI: 10.1109/TPAMI.2011.22.

[4] T. Bluche and R. Messina. "Gated Convolutional Recurrent Neural Networks for Multilingual Handwriting Recognition". In: *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR* 1 (2017), pp. 646–651. ISSN: 15205363. DOI: 10.1109/ICDAR.2017.111.

[5] T. Bluche, H. Ney, and C. Kermorvant. "Tandem HMM with convolutional neural network for handwritten word recognition". In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE, May 2013, pp. 2390–2394. ISBN: 978-1-4799-0356-6. DOI: 10.1109/ICASSP.2013.6638083. URL: http://ieeexplore.ieee.org/document/6638083/.

[6] H. Bunke, M. Roth, and E. Schukat-Talamazzini. "Off-line cursive handwriting recognition using hidden markov models". In: *Pattern Recognition* 28.9 (Sept. 1995), pp. 1399–1413. ISSN: 00313203. DOI: 10.1016/0031-3203(95)00013-P. URL: https://linkinghub.elsevier.com/retrieve/pii/003132039500013P.

[7] S. Cascianelli, M. Cornia, L. Baraldi, M. L. Piazzi, R. Schiuma, and R. Cucchiara. "Learning to Read L ' Infinito : Handwritten Text Recognition with Synthetic Training Data". In: *To appear in: Proceedings of the 19th International Conference on Computer Analysis of Images and Patterns*. 2021, pp. 1–10.

[8] J. F. Dartigues, M. Gagnon, P. Michel, L. Letenneur, D. Commenges, P. Barberger-Gateau, S. Auriacombe, B. Rigal, R. Bedry, A. Alperovitch, J. M. Orgogozo, P. Henry, P. Loiseau, and R. Salamon. "The Paquid research program on the epidemiology of dementia. Methods and first results". In: *Revue Neurologique* 147.3 (1991), pp. 225–230. ISSN: 00353787.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, Kai Li, and Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June. IEEE, June 2009, pp. 248–255. ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPR.2009.5206848. URL: https://ieeexplore.ieee.org/document/5206848/.

[10] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of deep bidirectional transformers for language understanding". In: *NAACL HLT 2019 - 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies - Proceedings of the Conference* 1.Mlm (2019), pp. 4171–4186. arXiv: 1810.04805.

[11] R. Eckart de Castilho, É. Mùjdricza-Maydt, S. M. Yimam, S. Hartmann, I. Gurevych, A. Frank, and C. Biemann. "A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures". In: *Proceedings of the workshop on Language Technology Resources and Tools for Digital Humanities (LT4DH) at COLING 2016* (2016), pp. 76–84.

[12] A. Graves, M. Liwicki, S. Fernandez, R. Bertolami, H. Bunke, and J. Schmidhuber. "A Novel Connectionist System for Unconstrained Handwriting Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 31.5 (May 2009), pp. 855–868. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2008.137. URL: http://ieeexplore.ieee.org/document/4531750/.

[13] A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Vol. 385. Studies in Computational Intelligence. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012. ISBN: 978-3-642-24796-5. DOI: 10.1007/978-3-642-24797-2. URL: http://link.springer.com/10.1007/978-3-642-24797-2.

[14] A. Graves and J. Schmidhuber. "Offline handwriting recognition with multidimensional recurrent neural networks". In: *Advances in Neural Information Processing Systems 21 - Proceedings of the 2008 Conference*. 2009, pp. 545–552. ISBN: 9781605609492.

[15] A. Hannun. "Sequence Modeling with CTC". In: *Distill* 2.11 (Nov. 2017). ISSN: 2476-0757. DOI: 10.23915/distill.00008. URL: https://distill.pub/2017/ctc.

[16] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *32nd International Conference on Machine Learning, ICML 2015* 1 (2015), pp. 448–456. arXiv: 1502.03167.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. ISSN: 00189219. DOI: 10.1109/5.726791. arXiv: 1102.0183. URL: http://ieeexplore.ieee.org/document/726791/.

[18] H. Lindsay, P. Müller, I. Kröger, N. Linz, A. König, R. Zeghari, F. R. Verhey, and I. H. Ramakers. "Multilingual Learning for Mild Cognitive Impairment Screening from a Clinical Speech Task Multilingual Learning for Mild Cognitive Impairment Screening from a Clinical Speech Task". In: *Proceedings of Recent Advances in Natural Language Processing*. 2021, pp. 835–843. ISBN: 9789544520724. DOI: 10.26615/978-954-452-072-4_096.

[19] A. L. Maas, A. Y. Hannun, and A. Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *in ICML Workshop on Deep Learning for Audio, Speech and Language Processing* 28 (2013).

[20] K. Malhotra, S. Bansal, and S. Ganapathy. "Active Learning Methods for Low Resource End-to-End Speech Recognition". In: *Interspeech 2019*. ISCA: ISCA, Sept. 2019, pp. 2215–2219. DOI: 10.21437/Interspeech.2019-2316. URL: http://www.isca-speech.org/archive/Interspeech%7B%5C_%7D2019/abstracts/2316.html.

[21] U.-V. Marti and H. Bunke. "The IAM-database: an English sentence database for offline handwriting recognition". In: *International Journal on Document Analysis and Recognition* 5.1 (2002), pp. 39–46.

[22] G. Muehlberger, L. Seaward, M. Terras, S. Ares Oliveira, V. Bosch, M. Bryan, S. Colutto, H. Déjean, M. Diem, S. Fiel, B. Gatos, A. Greinoecker, T. Grüning, G. Hackl, V. Haukkovaara, G. Heyer, L. Hirvonen, T. Hodel, M. Jokinen, P. Kahle, M. Kallio, F. Kaplan, F. Kleber, R. Labahn, E. M. Lang, S. Laube, G. Leifert, G. Louloudis, R. McNicholl, J. L. Meunier, J. Michael, E. Mühlbauer, N. Philipp, I. Pratikakis, J. Puigcerver Pérez, H. Putz, G. Retsinas, V. Romero, R. Sablatnig,

J. A. Sánchez, P. Schofield, G. Sfikas, C. Sieber, N. Stamatopoulos, T. Strauß, T. Terbul, A. H. Toselli, B. Ulreich, M. Villegas, E. Vidal, J. Walcher, M. Weidemann, H. Wurster, and K. Zagoris. "Transforming scholarship in the archives through handwritten text recognition: Transkribus as a case study". In: *Journal of Documentation* 75.5 (2019), pp. 954–976. ISSN: 00220418. DOI: 10.1108/JD-07-2018-0114.

[23] R. Munro. *Human-in-the-Loop Machine Learning - Active learning and annotation for human-centered AI (Version 11)*. MEAP Editi. Manning Publications, 2021.

[24] D. Nurseitov, K. Bostanbekov, M. Kanatov, A. Alimova, A. Abdallah, and G. Abdimanap. "Classification of handwritten names of cities and handwritten text recognition using various deep learning models". In: *Advances in Science, Technology and Engineering Systems* 5.5 (2020), pp. 934–943. ISSN: 24156698. DOI: 10.25046/AJ0505114.

[25] J. Puigcerver. "Are Multidimensional Recurrent Layers Really Necessary for Handwritten Text Recognition?" In: *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Vol. 21. 4. IEEE, Nov. 2017, pp. 67–72. ISBN: 978-1-5386-3586-5. DOI: 10.1109/ICDAR.2017.20. URL: http://ieeexplore.ieee.org/document/8269951/.

[26] V. Romero, J. A. Sanchez, and A. H. Toselli. "Active Learning in Handwritten Text Recognition using the Derivational Entropy". In: *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. IEEE, Aug. 2018, pp. 291–296. ISBN: 978-1-5386-5875-8. DOI: 10.1109/ICFHR-2018.2018.00058. URL: https://ieeexplore.ieee.org/document/8583776/.

[27] H. Scheidl. "Handwritten Text Recognition in Historical Documents". PhD thesis. Technische Universität Wien, 2018. ISBN: 9781450325882. URL: http://www.ub.tuwien.ac.athttp//www.ub.tuwien.ac.at/eng.

[28] H. Scheidl, S. Fiel, and R. Sablatnig. "Word beam search: A connectionist temporal classification decoding algorithm". In: *Proceedings of International Conference on Frontiers in Handwriting Recognition, ICFHR* 2018-Augus (2018), pp. 253–258. ISSN: 21676453. DOI: 10.1109/ICFHR-2018.2018.00052.

[29] B. Settles. *Active learning literature survey*. Tech. rep. University of Wisconsin-Madison Department of Computer Sciences, 2010.

[30] Z. Shen, R. Zhang, M. Dell, B. C. G. Lee, J. Carlson, and W. Li. "LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis". In: 1.Dl (2021), pp. 1–16. ISSN: 16113349. DOI: 10.1007/978-3-030-86549-8_9. arXiv: 2103.15348. URL: http://arxiv.org/abs/2103.15348.

[31] The ImageMagick Development Team. *ImageMagick*. Version 7.0.10. Jan. 4, 2021. URL: https://imagemagick.org.

[32] M. Villegas, V. Romero, and J. A. Sánchez. "On the Modification of Binarization Algorithms to Retain Grayscale Information for Handwritten Text Recognition". In: *En Pattern Recognition and Image Analysis: 7th Iberian Conference, IbPRIA 2015*. Santiago de Compostela, Spain: Springer International Publishing, 2015, pp. 208–215. DOI: 10.1007/978-3-319-19390-8_24. URL: http://link.springer.com/10.1007/978-3-319-19390-8%7B%5C_%7D24.

[33] M. Wolf, D. Ruiter, A. G. D'Sa, L. Reiners, J. Alexandersson, and D. Klakow. "HUMAN: Hierarchical Universal Modular ANnotator". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2020, pp. 55–61. DOI: 10.18653/v1/2020.emnlp-demos.8. arXiv: 2010.01080. URL: http://arxiv.org/abs/2010.01080%20https://www.aclweb.org/anthology/2020.emnlp-demos.8.

[34] L. Yang, Y. Zhang, J. Chen, S. Zhang, and D. Z. Chen. "Suggestive Annotation: A Deep Active Learning Framework for Biomedical Image Segmentation". In: *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*. Ed. by M. Descoteaux, L. Maier-Hein, A. Franz, P. Jannin, D. L. Collins, and S. Duchesne. Cham: Springer International Publishing, 2017, pp. 399–407. ISBN: 978-3-319-66179-7.

[35] S. M. Yimam, C. Biemann, R. Eckart de Castilho, and I. Gurevych. "Automatic Annotation Suggestions and Custom Annotation Layers in WebAnno". In: *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Vol. 1. 1. Stroudsburg, PA, USA: Association for Computational Linguistics, 2014, pp. 91–96. DOI: 10.3115/v1/P14-5016. URL: http://aclweb.org/anthology/P14-5016.

A. Appendix

A.1. Baseline Model

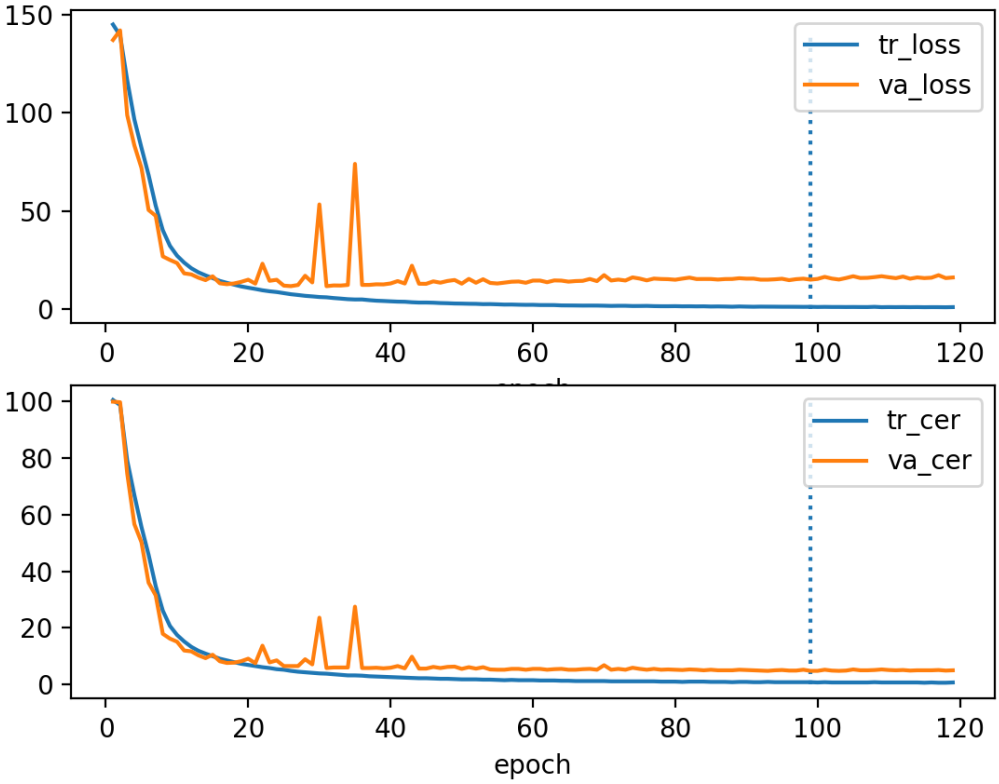


Figure 29: Baseline model training- and validation-loss and training- and validation-CER. The dotted line shows the best epoch 99.

A.2. Active Learning model performance

Iteration	Word accuracy	Correct	Total words	WER	CER
0	0.031	69	2235	1.641	1.198
1	0.422	1021	2418	0.624	0.320
2	0.753	1825	2423	0.266	0.116
3	0.834	2038	2445	0.175	0.083
4	0.830	1887	2273	0.177	0.084
5	0.884	2083	2356	0.125	0.055
6	0.904	2047	2264	0.099	0.052
7	0.941	2203	2340	0.065	0.030
8	0.915	2038	2228	0.092	0.036
9	0.927	2112	2279	0.078	0.032
10	0.948	2098	2213	0.056	0.021
11	0.963	2054	2133	0.037	0.017
12	0.951	1981	2084	0.051	0.021
13	0.949	1892	1994	0.054	0.021
14	0.972	1741	1791	0.027	0.013
15	0.962	1457	1515	0.038	0.019
16	0.946	930	983	0.054	0.029

Table 4: Model performance of AL with uncertainty sampling for suggestions

Iteration	Greedy Best First		Beamsearch	
	WER	CER	WER	CER
0	1.242	1.162	1.086	0.734
1	0.668	0.230	0.421	0.199
2	0.474	0.163	0.305	0.139
3	0.402	0.138	0.279	0.121
4	0.377	0.128	0.278	0.115
5	0.343	0.117	0.264	0.105
6	0.339	0.118	0.260	0.104
7	0.327	0.109	0.257	0.100
8	0.314	0.108	0.245	0.099
9	0.297	0.103	0.250	0.096
10	0.303	0.104	0.252	0.097
11	0.299	0.101	0.253	0.095
12	0.285	0.101	0.241	0.094
13	0.294	0.101	0.241	0.093
14	0.271	0.096	0.239	0.091
15	0.275	0.098	0.242	0.093
16	0.280	0.099	0.247	0.095
17	0.279	0.098	0.242	0.094

Table 5: Model performance of AL with uncertainty sampling on test-data with GBF- and Beamsearch-Decoder

A.3. Best model performances

	WER			CER		
	delete	insert	substitute	delete	insert	substitute
Baseline	717	744	3006	3812	2674	10944
From scratch	81	244	814	975	477	1572
Transfer Learning	70	**239**	781	828	**526**	1489
Random sampling best	196	224	816	1184	586	1348
Uncertainty sampling	**48**	274	**659**	**625**	549	**995**

Table 6: Comparison of edit-operations of CER and WER on test data with baseline, from scratch and Transfer Learning model and uncertainty and random sampling using their respective best models.