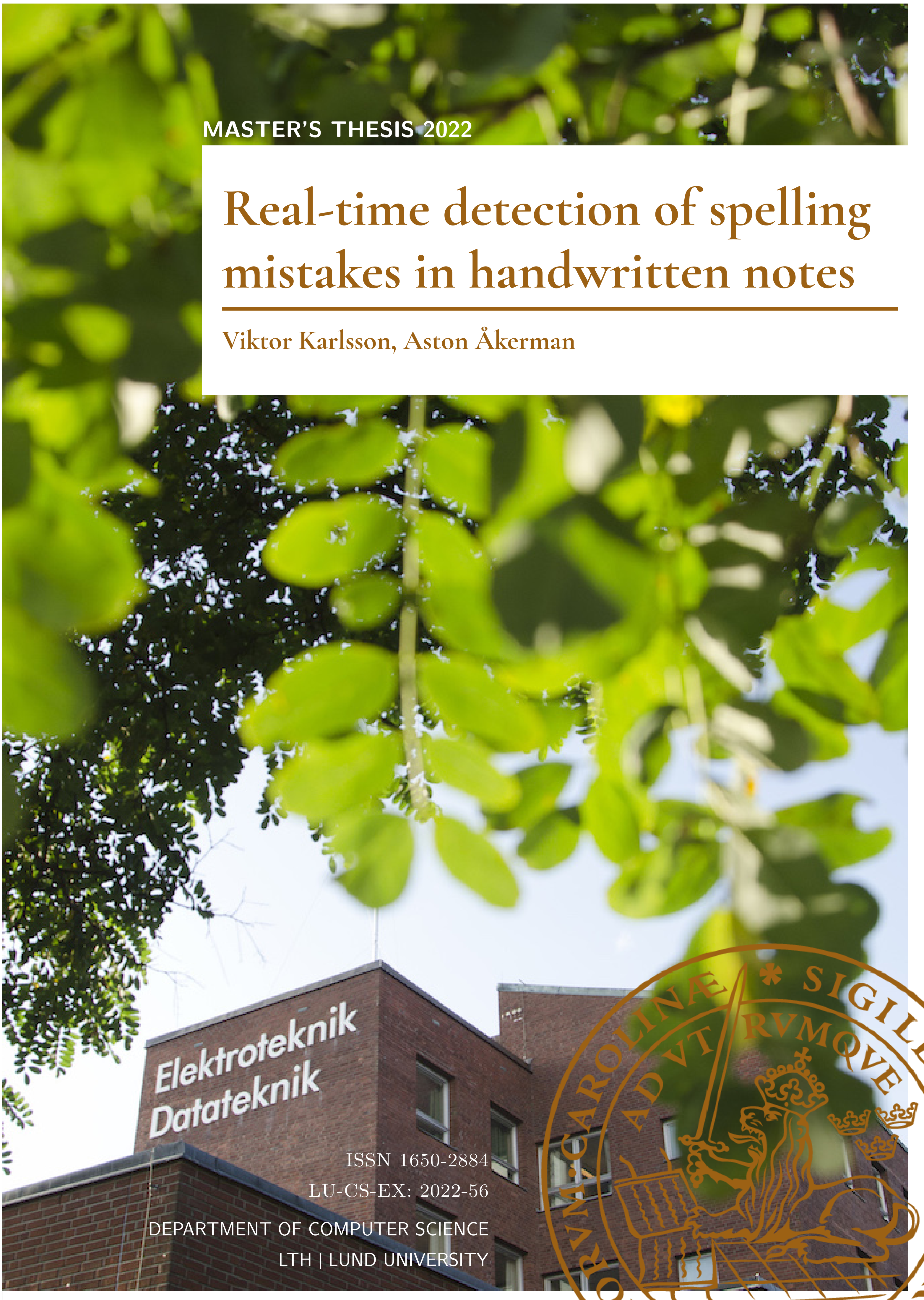


MASTER'S THESIS 2022

Real-time detection of spelling mistakes in handwritten notes

Viktor Karlsson, Aston Åkerman

ISSN 1650-2884
LU-CS-EX: 2022-56

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

LU-CS-EX: 2022-56

Real-time detection of spelling mistakes in handwritten notes

Realtidsdetektering av stavfel i handskrivna anteckningar

Viktor Karlsson, Aston Åkerman

Real-time detection of spelling mistakes in handwritten notes

Viktor Karlsson
vi2041ka-s@student.lth.se

Aston Åkerman
as1028ak-s@student.lth.se

October 18, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Marcos Rubinstein, marcos.rubinstein@heig-vd.ch

Examiner: Marcus Klang, marcus.klang@cs.lth.se

Abstract

The art of handwriting has been relevant for thousands of years and continues to be so in modern times. Technology that can recognize handwritten text in images could be utilized by smart-glasses, as a supportive tool for visually impaired or as a tool to facilitate learning how to write. With this thesis, we present **Orthographer**, an end-to-end handwritten text recognition system, able to detect spelling mistakes in handwritten text in real time. The system shows promising capabilities, detecting $\sim$ **69%** of the spelling mistakes in our test set and running at $\sim$ **16.3**ms per word. The code and models will be made publicly available at the URL: `https://github.com/viktorkar/orthographer`.

Keywords: Optical Character Recognition, Handwritten Text Recognition, Handwritten Text Detection, Machine Learning, Image Processing, Natural Language Processing, System Development

Acknowledgements

We would like to thank Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud (HEIG-VD) for providing us with the opportunity to realize this project. Particularly, we want to recognize our supervisors Pierre Nugues at Lund University and Marcos Rubinstein at HEIG-VD, who contributed with vital suggestions, feedback and assistance during the developmental process, in addition to input on the writing of the thesis.

Contents

Chapter 1

Introduction

1.1 Background

To be literate means to know how to read and write. It is one of the most important skills in modern society and has been used in societies for more than 5000 years, one early example being the bureaucracy of keeping track of taxation and accounting in the Mesopotamian society (Robson, 1992). In today's society, handwriting is still present in our daily lives, be it for note taking in academic classes and business meetings, writing down ideas or brainstorming. Even though everything is becoming more and more digitized, handwriting continues to be the preferred way to capture ideas for many.

At the same time, computers, tablets and phones have made it incredibly easy to preserve, arrange, and disseminate text. Text in digital format has the advantage of easily being analysed to detect spelling errors, grammatical errors and to extract specific information. In order for handwritten text to be able to benefit from the same tools, it is essential to be able to detect, recognize and digitize it. This concept is referred to as handwritten text recognition (HTR) and is a problem of high interest within research in the field of automatic document analysis. Due to the significant differences in handwriting between individuals and handwritten characters' imprecise nature, the recognition problem has proved hard to model and is an area where state-of-the-art solutions are continuously improving (Sueiras, 2021).

Research around the HTR problem is split into two different areas, *online handwriting recognition* and *offline handwriting recognition*. Online handwriting recognition uses the continuous (x, y) positions of a pen to recognize the text, which for example is the case when writing on a tablet. Offline handwriting recognition consists of recognizing text in a given image. The latter is seen as the more difficult of the two as less information is available and images can easily include noise or be distorted (Sueiras, 2021). The focus of this thesis is solely on the offline handwriting recognition problem. Figure 1.1 shows an example of the difference between the two types of data.

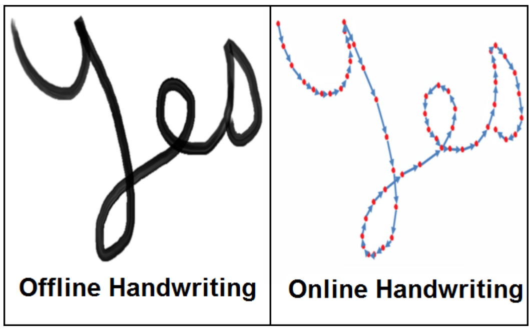


Figure 1.1: Example of offline and online handwriting data (Rüfenacht, 2020).

1.2 Problem Definition

This thesis aims to explore the possibility of utilizing and adapting state-of-the-art text recognition techniques and deep learning models to develop a system for the purpose of detecting spelling errors in handwritten notes in real time. The system covers the problems of text detection, text recognition and spell checking, all of which need to fulfill the limitation of being able to run in real time. A potential future use-case for the research is for the software, or parts of the software developed, to be used in augmented reality (AR) smart-glasses or from a smart-phone, which emphasizes a goal of decreasing the amount of computational capacity that the system requires. The trade-off between achieving state-of-the-art handwritten text recognition, in addition to supporting real-time computation on common central processing units (CPUs), is central to the project.

1.3 Contribution

Previous research is focused on exploring and evolving the technology of handwritten text recognition. In this project, we experimented with the techniques found in modern research and explored whether a real-time spelling correction system could be developed using these techniques. The contributions of this project are summarized as follows:

- We propose a modular system able to detect spelling errors in handwritten notes in real time. The different modules can be worked on and improved independently, facilitating future research projects where such a system can be used in augmented reality, as an educational tool for learning to write, or as a supportive tool for the visually impaired.
- The system can run on machines without a lot of computational resources.
- The code and models used will be publicly available at the URL: `https://github.com/viktorkar/orthographer`.

1.3.1 Division of responsibility

This research was to a high degree conducted collectively, with most decisions and challenges being discussed and solved with input from both Aston and Viktor. Additionally, pair programming or code reviews have regularly been utilized as development methods.

However, responsibilities were divided between different areas. Viktor Karlsson was primarily responsible for the project and code structure, the text detection module and post processing module. Aston Åkerman was responsible for the text recognition module and the spell correction module, as well as being responsible for the processing of the datasets.

1.4 Limitations

When developing the system, we made various assumptions about the data and configuration which put some limitations on how the program can be utilized. These limitations are:

- The handwritten notes are assumed to only contain text, written left to right and top to bottom.
- When evaluating the system, we only use the characters $[a\text{–}z0-9]$.
- The camera and paper are assumed to be perfectly still.
- All experiments and evaluations are performed on handwritten text in English.
- The system needs to be able to run on a laptop running an Intel Core i7-9750H CPU @ 2.60GHz and a Geforce GTX 1660 Ti GPU with 16GB of RAM.

1.5 Thesis Outline

Chapter 1 introduced the thesis problem and gave some motivation to our work.

Chapter 2 describes the models and datasets used in our experiments.

Chapter 3 explains our approach to the problem and the different modules built for our system.

Chapter 4 describes the experiments carried out in the report.

Chapter 5 includes evaluation results from the different modules and the entire system.

Chapter 6 analyses the results and discusses our findings.

Chapter 7 includes our conclusions and proposes future work.

Chapter 2
Terminology

In this chapter, we concisely introduce the technical concepts needed to understand the main concepts of the report. We will not be thorough with the detailing, but further information can be easily gathered using the name of the concept.

- **Bounding box:** A solution for how object detection models outputs results is to output coordinates for the 4 corners of a box that covers each detected object. (Gron, 2017)
- **Convolutional Neural Network (CNN):** A category of artificial neural networks, that are using stacked layers of convolutional operations, which takes into account surrounding data points. This aspect makes the models particularly suitable for domains where the prediction of one data point depends on the surrounding points. (Gron, 2017)
- **Transformer:** A category of artificial neural networks, that uses the technique self-attention to weigh the importance of different parts when inputting whole sequences of sequential data, (i.e. an entire image consisting of multiple pixel values). (Gron, 2017)
- **Objectness score:** Concept presented by Choi et al. (2019), which uses the idea that an object detection neural network model will predict objects with varying precision depending on the angle and distance. This objectness score is used to determine the exact class among all of the detected classes.
- **Class probabilities:** For classification in machine learning, models can output the probabilities of an input belonging to each class. (Gron, 2017)
- **Precision:** A measurement that answers the question "Out of all predictions predicted 'x', how many were actually 'x'?". (Gron, 2017)
- **Recall:** A measurement that answers the question "Out of all actual 'x', how many were predicted as 'x'?". (Gron, 2017)

- **F1-score:** The harmonic mean of precision and recall. A value of 1.0 indicates a perfect precision and recall. (Gron, 2017)
- **Mean average precision (mAP):** Metric useful for measuring the precision of object detection models. As these models often output bounding boxes, some type of threshold is usually set to determine if the output is deemed correct or not (eg. 50% overlap). We could then use mAP to for example measure the average precision when the threshold is greater than 0.5 (annotated mAP@0.5). (Gron, 2017)
- **ICDAR:** Stands for "International Conference on Document Analysis and Recognition", and is a conference series for the document analysis community. Some datasets presented at ICDAR will be mentioned in the report.
- **Pre-trained model:** If we train a machine learning model that has already been trained on a dataset, we say that the model is a ”pre-trained model”.
- **Fine-tuning:** Instead of initializing the machine learning model weights using randomized values, weights are reused from the same model trained on a similar dataset, where to an extent learnt behaviour carries over to the new dataset. (Gron, 2017)
- **Transfer learning:** Applying the learnt knowledge of a pre-trained model for a new task. (Gron, 2017)
- **Confidence score:** A probability of how confident the model is of its prediction. (Gron, 2017)
- **Explicit post processing:** Post processing of predictions are done separately from the prediction of the model. For example using a dictionary to post process words that do not exist in the language. (Li et al., 2021)
- **Implicit post processing:** Post processing of predictions are done by the model itself. For example using a language model that has learnt the spelling and grammar conventions of a language. (Li et al., 2021)
- **Encoding:** To transform categorical variables (such as a specific shape in an image) into numerical variables, also called features. Is used in machine learning to feed information from for example an input image into neural nodes. (Gron, 2017)
- **Decoding:** To transform numerical variables, also called features, into a legible form. For example, combining the outputs of neural nodes and deciphering the combination into a prediction. (Gron, 2017)
- **Gradient Descent:** Multidimensional optimization problem of iteratively leaping towards the direction of a minima. (Gron, 2017)
- **Regularization:** Inclusion of penalty term to loss function in order to penalize extreme values, for the purpose of thwarting overfitting. (Gron, 2017)
- **Early stopping:** Regularization method of stopping the training early by manually looking at the loss and accuracy of the training and validation data. (Gron, 2017)

- **Data augmentation:** Technique of altering or modifying existing training data in order to create more training data. (Gron, 2017)
- **Generalization:** The ability for a machine learning model to correctly predict on unseen data from the same distribution. (Gron, 2017)

Chapter 3
Technical Background

In this chapter, we introduce prominent deep learning models and datasets related to the fields covered by the project. These models and datasets are later used in Chapters 5 and 6. The system we developed uses a pipeline consisting of multiple modules explained in detail in Sect. 4.1. The modules related to this chapter are the text detection module and the text recognition module. The sections pertaining to models in this chapter are structured in a sequential order to follow this system pipeline.

3.1 Text Detection Models

The task of determining text regions in images and labeling detected regions with bounding boxes is called *text detection* and is seen as the fundamental step of text recognition (Wang et al., 2020b). In the last years, scene text recognition research has progressively veered from straight horizontal text towards more difficult text such as multi-oriented and arbitrary-shaped. Handwritten text is by nature difficult to automatically recognize as the text is imprecise and substantial differences in writing habits between individuals (Sueiras, 2021).

3.1.1 EAST: An Efficient and Accurate Scene Text Detector

EAST: An Efficient and Accurate Scene Text Detector is a prominent text detection model created by Zhou et al. (2017), and is capable of real-time detection. EAST consists of a simple 2-step pipeline that, given an image, can predict word positions or positions of entire text lines of any orientation and quadrilateral shape. It works by first passing the image through a fully convolutional network (FCN) model, seen in Figure 3.1, that directly produces the word or text line predictions.

The predictions are then sent through a non-maximum suppression to produce the final results. The simplicity of the pipeline makes EAST very efficient and fast compared to many other models, which is of high value for our system where computational resources are limited. In their original paper, Zhou et al. (2017) achieved an F-score of 0.7820, running at 13.2fps at 720p resolution on the ICDAR 2015 dataset.

As indicated by its name, EAST is developed specifically for scene text detection and the pretrained model available for download is trained on the ICDAR2013 and ICDAR2015 datasets. As can be seen in Figure 3.2, this type of data is very dissimilar to our use case, where detection of handwritten text in a document is the focus. Therefore, re-training or fine-tuning the model on a dataset more similar to our use case would probably yield better results for our system than simply using the pretrained model out of the box.

For the experiments carried out in this report, we used the EAST model implemented in the computer vision library OpenCV.

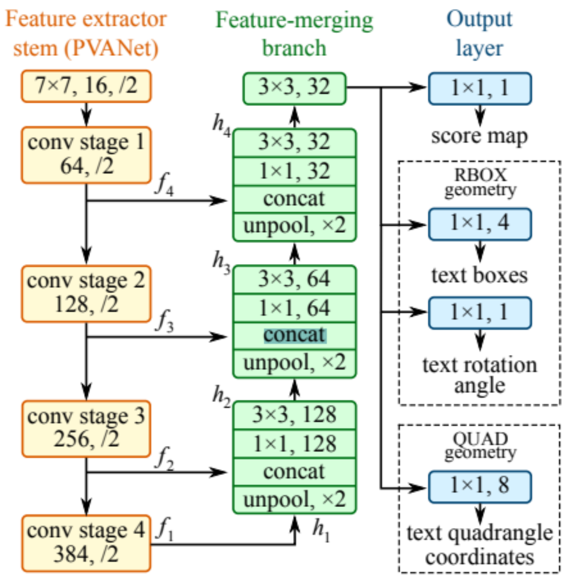


Figure 3.1: EAST FCN architecture

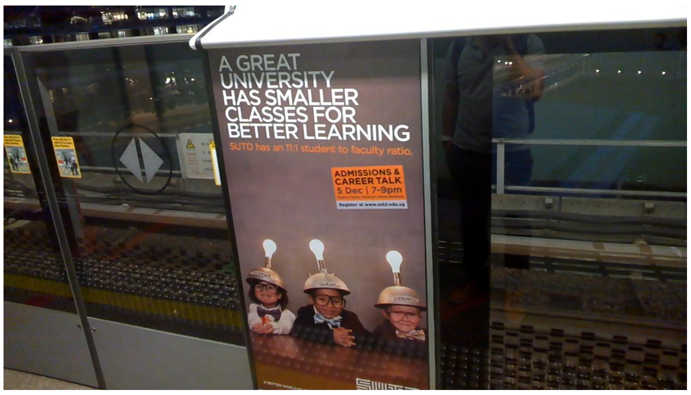

A GREAT UNIVERSITY HAS SMALLER CLASSES FOR BETTER LEARNING
SUTD has an 11:1 student to faculty ratio.
ADMISSIONS & CAREER TALK
5 Dec | 7-9pm

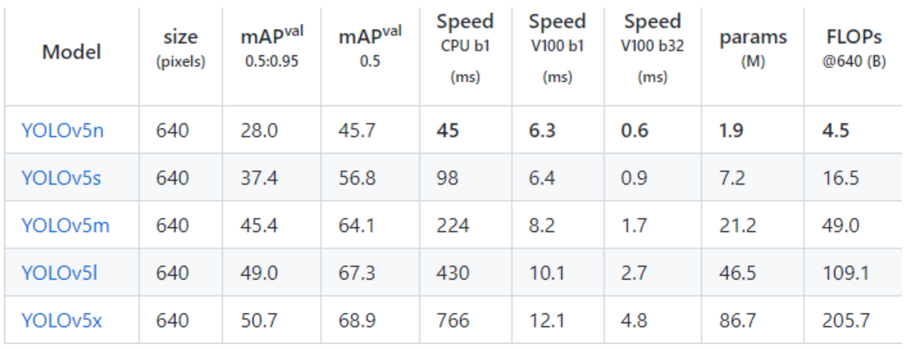
Figure 3.2: Example image from the ICDAR datasets.

3.1.2 YOLO: You Only Look Once

Another promising model for our text detection and segmentation module is the YOLO: You Only Look Once model. The model was first introduced in 2015 by Redmon et al. (2016) and was seen as a milestone in object detection research due to its capability of detecting objects in real time while maintaining a high accuracy (Chamidu, 2020). Since then, new and improved versions of the model have been published with YOLOv5 being one of the most recent ones and the one of most interest to our system.

As can be seen in Figure 3.3, multiple pretrained YOLOv5 model variants are available with varying performance and number of parameters. This is valuable for our system where the trade-off between inference time and accuracy needs to be considered. The models are trained on the COCO dataset (Lin et al., 2014) which is a large-scale object detection, segmentation, and captioning dataset. By using the pretrained weights and training the model on a dataset for handwritten text detection, we can apply the YOLOv5 model for the task of handwritten text detection.

Looking at the implementation details of YOLOv5, an overview of the network architecture can be seen in Figure 3.4. Starting of with the model backbone, cross stage partial networks (CSPNet) are incorporated into the open source neural network framework Darknet creating CSPDarknet. CSPNet was introduced by Wang et al. (2020a) and has shown to greatly reduce the amount of computations in deeper networks, enabling state-of-the-art methods such as ResNet, ResNeXt, and DenseNet to be light-weighted for devices with limited GPU and CPU resources like mobile phones. Secondly, YOLOv5 applies path aggregation network (PANet) (Liu et al., 2018) as its neck. The model neck generates feature pyramids which help the model generalize well on objects of different scales, for example small and large text in our system. Finally, the head generates the final output, consisting of bounding boxes, class probabilities and objectness score.



Model	size (pixels)	mAP^{val} 0.5:0.95	mAP^{val} 0.5	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7

Figure 3.3: Performance of YOLOv5 variants.

3.2 Text Recognition Models

The task of predicting actual text contents of detected text regions is called *text recognition*. This section includes related research and models for the project, regarding text recognition.

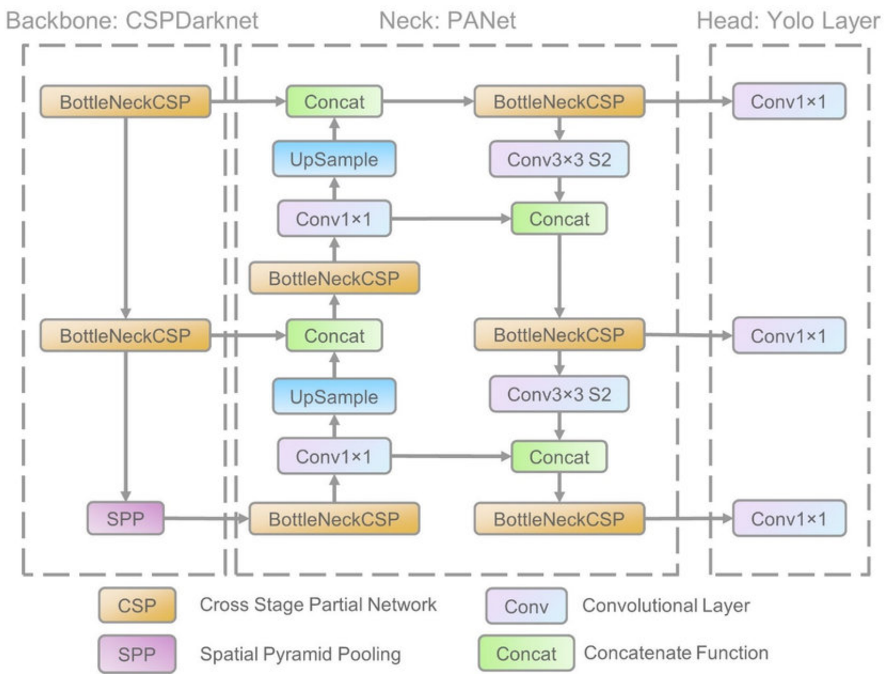


Figure 3.4: YOLOv5 network architecture with its three parts: (1) CSPDarknet, (2) PANet, and (3) Yolo Layer (Xu et al., 2021)

.

3.2.1 Transformer model: TrOCR

Transformer models have been highly successful in solving natural language processing (NLP) tasks and have recently found more and more success in the field of text recognition as well (Dosovitskiy et al., 2021). In a recent paper, Li et al. (2021) introduced the first end-to-end transformer based text recognition OCR model using pretrained computer vision (CV) and NLP models. The model is called TrOCR and has after its publication also been made available in the HuggingFace Transformer package. The model achieved state-of-the-art performance on both handwritten and printed text and is available in three different variants with varying number of parameters.

In contrast to previous transformer based architectures within the field, TrOCR is not reliant on a convolutional neural network as the backbone for image understanding. Instead, TrOCR uses a vision transformer (ViT) style (Dosovitskiy et al., 2021) model structure for the encoder. The encoder transforms the input image to a sequence of square image blocks that can be sequentially processed enabling it to choose what parts of the image to pay attention to. Also, unlike features extracted from convolutional neural networks (CNN), there are no image-specific inductive biases introduced Li et al. (2021).

The encoder is initialized using DeiT (Touvron et al., 2021) and BEiT (Bao et al., 2022). For the decoder, TrOCR uses an original transformer model initialized using the RoBERTa Zhuang et al. (2021) models. Because some encoder-decoder attention layers are absent in the RoBERTa models, some layers are also randomly initialized. An overview of the architecture is shown in Figure 3.5.

As is also shown in Figure 3.5, TrOCR takes lines of text as input but can also handle images containing single words. The model performs much better when multiple words are

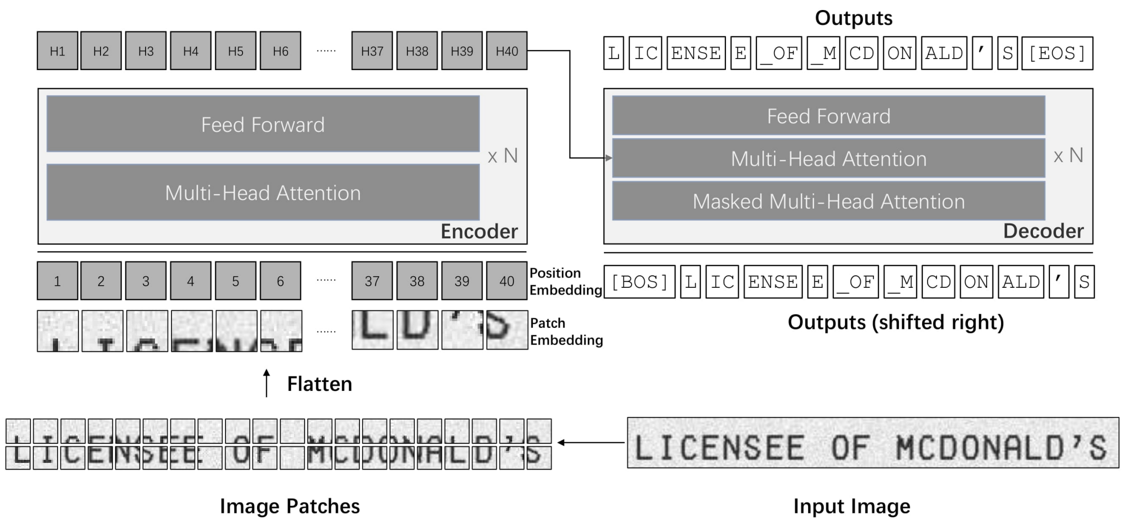


Figure 3.5: TrOCR architecture. Image taken from the original paper by Li et al. (2021).

present as it then can make use of its acquired language modeling ability. This can be seen in the results displayed in Section 5.

Since the language modeling ability is dependent on how the decoder is initialized, TrOCR can easily be extended to support multi-language by using a pretrained model with support for multiple languages on the decoder side.

3.2.2 Unified four-stage STR framework

In a research paper, Baek et al. (2019) make an analysis of how current research results are compared in the field of scene text recognition (STR). Furthermore, they present novel STR models that they create by combining modules from prior research. Additionally, the authors present a framework for training and evaluating scene text recognition models without explicit post processing using language models. They also publish pretrained models, using prior research. The framework consists of a pipeline of four stages, which are summarized as:

Transformation stage: This is an optional stage that performs pre-processing of cropped bounding boxes of detected text. The boxes are normalized and transformed using a thin-plate spline (TPS) transformation, which is a variant of the Spatial Transformer Network, in order to improve text recognition. The model automatically detects reference points in the input image, which are subsequently used to perform a smooth spline interpolation of the input image. The model was created by Shi et al. (2016). See Fig. 3.6.

Feature extraction: This stage computes features of the transformed image, using a convolutional neural network. This can be seen as an encoding stage, where an image is encoded using features found during training. Three existing well-known CNN architectures are explored in this stage by the authors, with the one showing best evaluation results being ResNet. ResNet was first proposed by He et al. (2016). Other architectures implemented in the framework for feature extraction are VGG (Simonyan and Zisserman, 2015) and RCNN (Lee and Osindero, 2016).

Sequence modeling: This is an optional stage which takes features as input from the prior stage. These features can lack contextual information, and a bidirectional long short-term memory model (BiLSTM) can be used to create a sequence giving contextual information about the features (Shi et al., 2015). Contextual information can improve text recognition performance as information from surrounding characters can give information, especially with characters that generally have higher or lower start/end points compared to average character start/end points such as *l*, *h*, *t* and *g*, *q*, *p*. In cases of thick characters being split up into multiple boxes, this property of contextual information can also improve performance. Each column from a feature extraction is used as a frame in the BiLSTM.

Prediction stage: In this stage, each frame from the sequence modeling stage is added up, or if not used, the features extracted during the second stage are added up. This can be seen as a decoding stage, where the extracted features are decoded into digital text.

There are two approaches implemented for this stage in the framework, which both allow for end-to-end training of the pipeline.

1. The first is connectionist temporal classification (CTC) (Graves et al., 2006). CTC is an output function for sequence-based problems where time is a variable. An important feature of CTC is that the length of the input sequence and the output sequence do not have to match. This means that the size of the input image does not have to match with the length of the outputted text prediction. In the case of handwritten text recognition, the edit distance between the prediction and the target strings can be used as a loss function.
2. The Attn approach works by automatically *soft-searching* the input image for relevant sections, which it gives more emphasize to, in comparison to sections deemed less relevant (Bahdanau et al., 2015). Attn uses an alignment model, which gives scoring depending on how well an element of the sequence aligns with the target sequence. This sequence of elements can either be the BiLSTM hidden state, or, if not used, the raw output from the feature extraction stage.

The best module combination by Baek et al. (2019) is named STAR-Net, which uses both of the two optional stages (transformation and sequence modeling). For feature extraction, Baek et al. (2019) use ResNet and for the prediction stage, they use CTC.

Early development versions of our system used pretrained models trained by Mu (2020), which were trained in a fork of the public repository of the research by Baek et al. (2019). These models were trained on scene text datasets, but are still included in Chapter 6 as an initial baseline to compare the improvement of finetuned models. The fork was created in order to integrate the work by Baek et al. (2019) with the computer vision library OpenCV.

AttentionHTR

Kass and Vats (2022) finetuned pretrained models by Baek et al. (2019) and also trained mod-

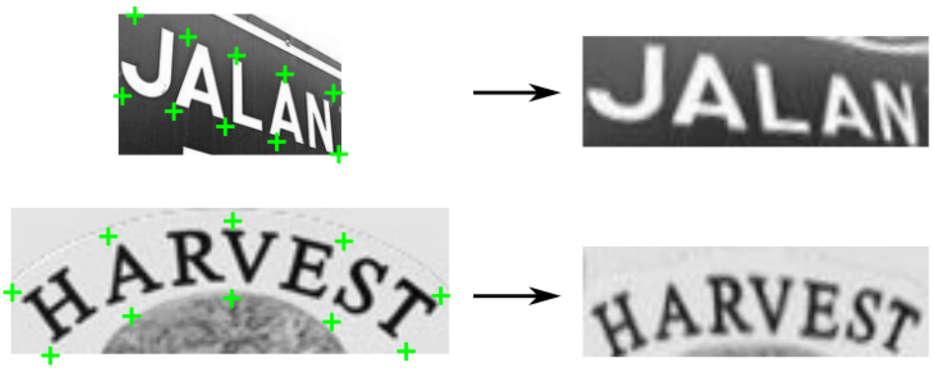


Figure 3.6: Thin-Plate-Spline transformation, samples taken from Shi et al. (2016)

els from scratch using the framework also by Baek et al. (2019) for handwritten offline text, with optimization of learning rate and choice of gradient descent optimization algorithm, in order to achieve good performance with the lower availability of data for offline HTR. There are more data available for STR, which Baek et al. (2019) optimized the training for. In fact, Kass and Vats (2022) argue that using transfer learning from STR to HTR is a viable solution, due to the scarcity of available data for HTR. The authors propose that future work is to experiment with different regularization techniques than the current of early stopping, as well as introducing data augmentation to the training, which they predict will lead to improved performance. Kass and Vats (2022) achieved better results for HTR by switching the prediction stage from CTC to Attn and they named their model AttentionHTR.

3.3 Language Models

In order to improve the results of a text recognition model, a usual approach is to incorporate a language model to post process or as a building block of the text recognition model. Language models predict words given their context. These models are now trained on gigantic corpora and can correct minor prediction errors as they encapsulate semantic knowledge from the corpora.

However, in the context of developing a system to detect spelling mistakes, there are inherent problems that can make this approach unfit for the context. The purpose of using a language model for post processing is to correct errors in the text recognition and transform it into a suitable grammatically and semantically correct likely prediction.

3.3.1 BERT: Bidirectional Encoder Representations from Transformers

The transformer-based language model BERT was created and published in 2018 by Jacob Devlin and other researchers from Google (Devlin et al., 2019) and has since become the baseline for language models (Rogers et al., 2020).

Devlin and his colleagues were influenced by the research of Vaswani et al. (2017), who implemented the transformer model used in BERT. The transformer architecture uses layers of multi-head self-attention mechanism connected position-wise to a fully connected feed-forward network. Residual connections created by He et al. (2016) are used around both the multi-head self-attention mechanism and for the fully connected layer. This prevents the vanishing gradient problem, as the backpropagation of the gradient does not travel through the non-linear activation function of the layers, but instead through the residual connection.

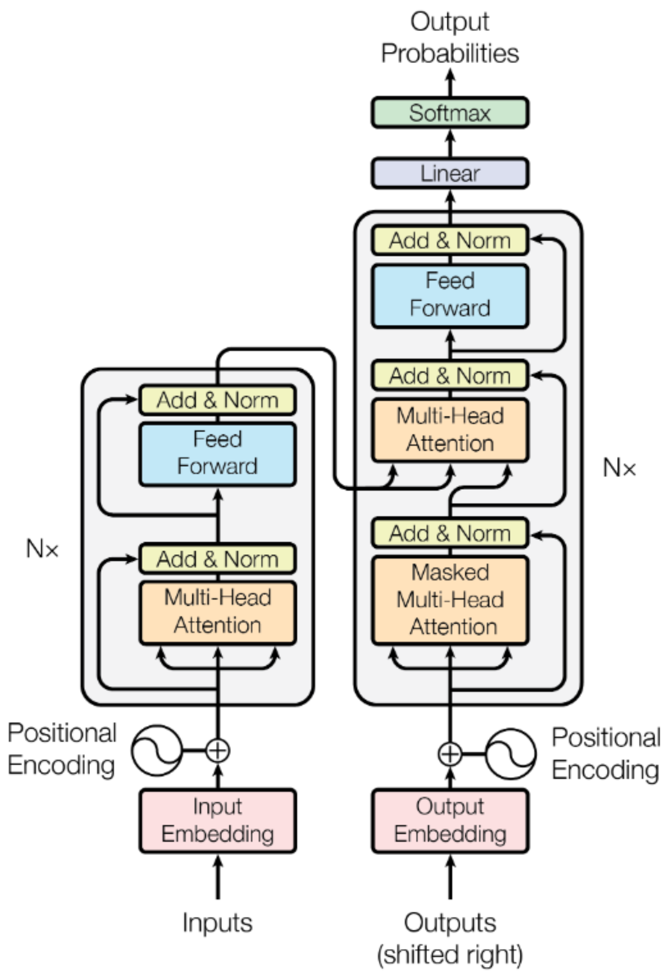


Figure 3.7: Architecture of a transformer model, image from Vaswani et al. (2017)

The architecture of BERT is a multi-layer bidirectional transformer with an encoder but no decoder. The training of BERT is divided into two stages, the first one being pretraining and the second being finetuning. During pretraining, unsupervised learning is conducted to train the model for the two tasks of masked language model and next sentence prediction, while the finetuning uses labelled data for other tasks. This facilitates the generality of BERT, which can be used for many different NLP problems. However, new tasks also require the finetuning stage, which may require the adding of a decoder to the structure. Anyway, the architectural difference between BERT models finetuned for different tasks is negligible.

Due to BERT's popularity and the generality of the architecture, there exists many different trained versions of BERT for different purposes and languages. Neuspell Jayanthi et al. (2020) is an example of this, which trained many transformer-based language models for the

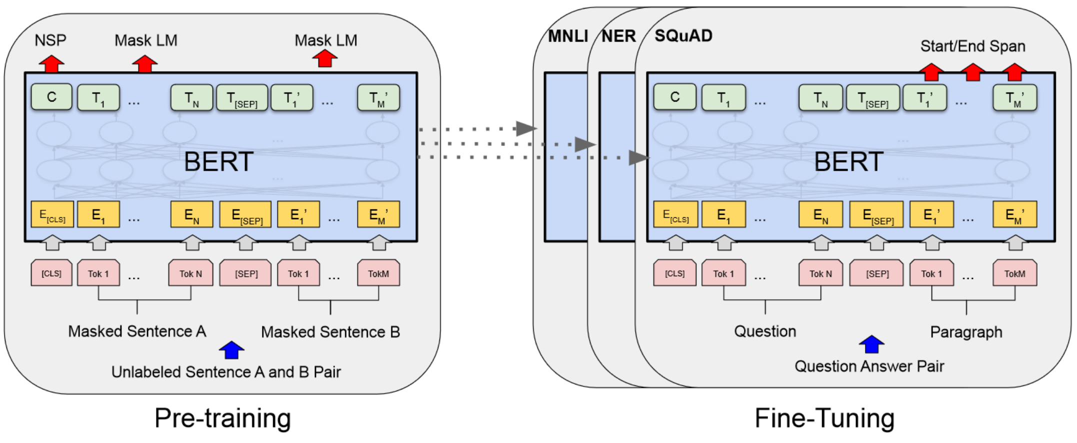


Figure 3.8: The two general training procedures for transformer language models like BERT, image from Devlin et al. (2019)

task of spelling correction. One of these models is called BertChecker, which is specifically a finetuned BERT model. BertChecker is further discussed in Section 4.1.

3.4 Datasets

This section describes the databases and datasets we used to carry out the experiments in this thesis.

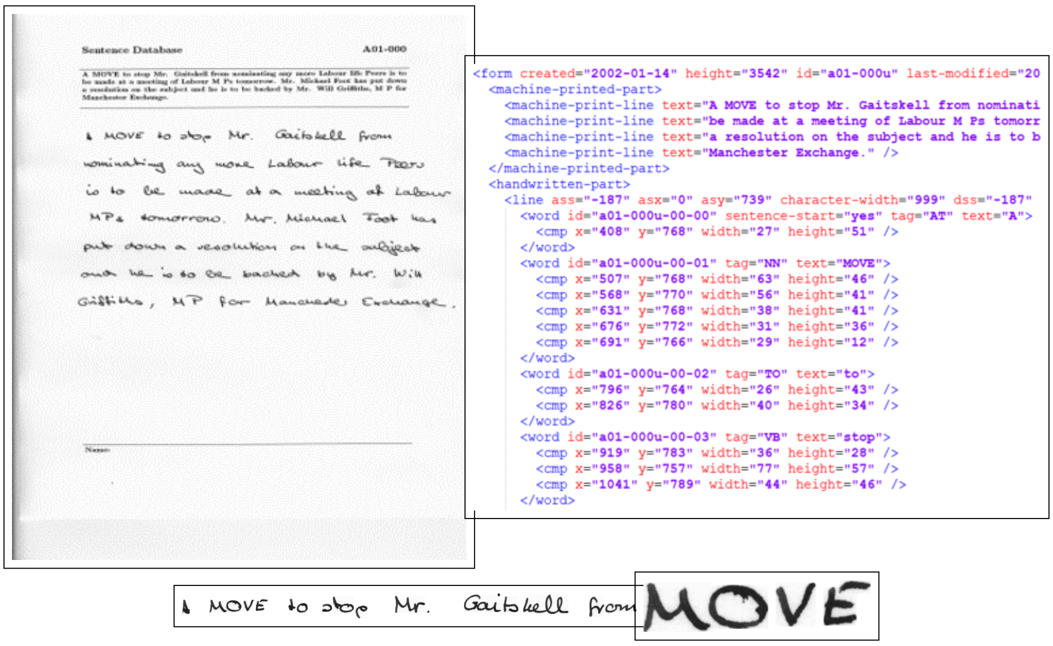
3.4.1 IAM Database

The IAM database was introduced by Marti and Bunke (2002) and contains 1539 pages of scanned handwritten English text contributed from 657 different writers. Little information is found about the writers of the data, which is important to assess diversity, and to achieve satisfactory generality in models trained with the dataset. The texts however, are based on the Lancaster-Oslo-Bergen Corpus (Johansson et al., 1986) and in total there are 13,353 isolated text lines and 115,320 isolated words. The database provides PNG images with 256 gray levels of forms, lines and words.

An example image from each dataset can be seen in Figure 3.9 together with the provided metadata stored in an XML file. The XML file contains among other information the position and transcription of each word.

The IAM database has been widely used in the literature and to be able to compare results, two different standard partitions are available. The first one is the official partition created by the IAM authors which creates the splits per writer in a training set, two validation sets and one test set. The other partition is named Aachen or RWTH and has become the most widely used for handwritten text recognition (Sueiras, 2021).

The IAM database also contains some data that were removed before our experiments. The XML file contains information about segmentation errors on the line level. In total, there are 2009 lines marked as incorrectly segmented out of the 13,353. There are also 33 crossed out words, identified and discussed by Sueiras (2021), annotated # in the XML file.



```
<form created="2002-01-14" height="3542" id="a01-000u" last-modified="20
  <machine-printed-part>
    <machine-print-line text="A MOVE to stop Mr. Gaitskell from nominati
    <machine-print-line text="be made at a meeting of Labour M Ps tomorr
    <machine-print-line text="a resolution on the subject and he is to b
    <machine-print-line text="Manchester Exchange." />
  </machine-printed-part>
  <handwritten-part>
    <line ass="-187" asx="0" asy="739" character-width="999" dss="-187"
      <word id="a01-000u-00-00" sentence-start="yes" tag="AT" text="A">
        <cmp x="408" y="768" width="27" height="51" />
      </word>
      <word id="a01-000u-00-01" tag="NN" text="MOVE">
        <cmp x="507" y="768" width="63" height="46" />
        <cmp x="568" y="770" width="56" height="41" />
        <cmp x="631" y="768" width="38" height="41" />
        <cmp x="676" y="772" width="31" height="36" />
        <cmp x="691" y="766" width="29" height="12" />
      </word>
      <word id="a01-000u-00-02" tag="TO" text="to">
        <cmp x="796" y="764" width="26" height="43" />
        <cmp x="826" y="780" width="40" height="34" />
      </word>
      <word id="a01-000u-00-03" tag="VB" text="stop">
        <cmp x="919" y="783" width="36" height="28" />
        <cmp x="958" y="757" width="77" height="57" />
        <cmp x="1041" y="789" width="44" height="46" />
      </word>
```

Figure 3.9: Samples from IAM forms, lines and words datasets together with the XML metadata.

Additionally, segmented images containing solely punctuation symbols [., :; & + −?!"() ∗ /′] were also removed from the dataset before our experiments and characters were converted to lower case. The resulting partition is shown in Table 3.1 and it is the partition we used for the experiments and benchmarks carried out in this report.

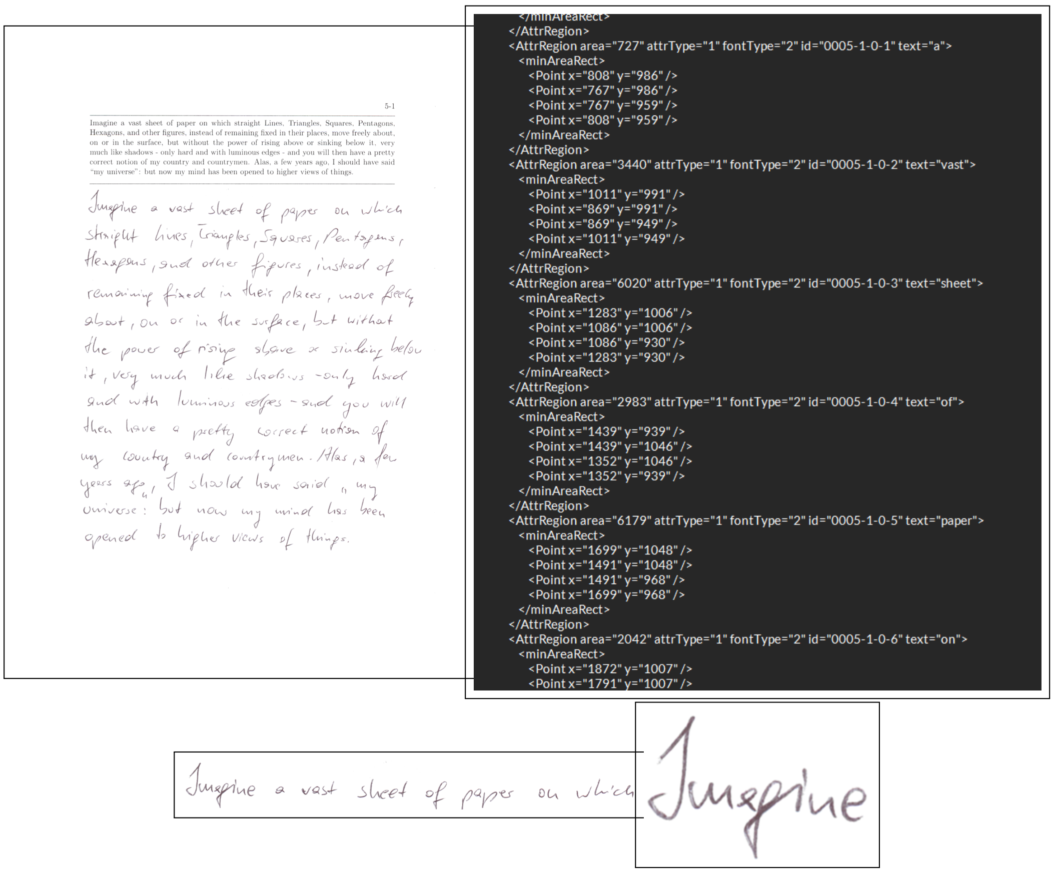
Partition	# Pages	# Lines	# Words
train	747	5695	42,493
validation	116	840	6453
test	336	2338	17,751

Table 3.1: The IAM split used for the experiments and benchmarks carried out in the report.

3.4.2 CVL Database

Kleber et al. (2013) created the CVL database for the purpose of writer identification and keyword spotting, and thereby not explicitly for text recognition. However, the database consists of annotated offline handwritten text documents in the same format as the IAM database, which makes the dataset potentially viable for training text recognition models. For the purpose of text recognition, the CVL database has a weakness compared to the IAM database as it is less diverse in regards to numbers of writers and texts.

The handwriting was performed by 310 writers, of which 283 wrote 5 texts, and the remaining 27 writers wrote 7 texts. All texts are of the same 7 samples, 6 of which were in English, the last one being in German. In total, the database consists of 101,069 words, nevertheless of these, the number of unique words is only 292. However, the number of forms



5-1

Imagine a vast sheet of paper on which straight Lines, Triangles, Squares, Pentagons, Hexagons, and other figures, instead of remaining fixed in their places, move freely about, on or in the surface, but without the power of rising above or sinking below it, very much like shadows - only hard and with luminous edges - and you will then have a pretty correct notion of my country and countrymen. Alas, a few years ago, I should have said "my universe": but now my mind has been opened to higher views of things.

Imagine a vast sheet of paper on which
straight lines, Triangles, Squares, Pentagons,
Hexagons, and other figures, instead of
remaining fixed in their places, move freely
about, on or in the surface, but without
the power of rising above or sinking below
it, very much like shadows - only hard
and with luminous edges - and you will
then have a pretty correct notion of
my country and countrymen. Alas, a few
years ago, I should have said „my
universe": but now my mind has been
opened to higher views of things.

```
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="727" attrType="1" fontType="2" id="0005-1-0-1" text="a">
    <minAreaRect>
      <Point x="808" y="986" />
      <Point x="767" y="986" />
      <Point x="767" y="959" />
      <Point x="808" y="959" />
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="3440" attrType="1" fontType="2" id="0005-1-0-2" text="vast">
    <minAreaRect>
      <Point x="1011" y="991" />
      <Point x="869" y="991" />
      <Point x="869" y="949" />
      <Point x="1011" y="949" />
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="6020" attrType="1" fontType="2" id="0005-1-0-3" text="sheet">
    <minAreaRect>
      <Point x="1283" y="1006" />
      <Point x="1086" y="1006" />
      <Point x="1086" y="930" />
      <Point x="1283" y="930" />
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="2983" attrType="1" fontType="2" id="0005-1-0-4" text="of">
    <minAreaRect>
      <Point x="1439" y="939" />
      <Point x="1439" y="1046" />
      <Point x="1352" y="1046" />
      <Point x="1352" y="939" />
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="6179" attrType="1" fontType="2" id="0005-1-0-5" text="paper">
    <minAreaRect>
      <Point x="1699" y="1048" />
      <Point x="1491" y="1048" />
      <Point x="1491" y="968" />
      <Point x="1699" y="968" />
    </minAreaRect>
  </AttrRegion>
  <AttrRegion area="2042" attrType="1" fontType="2" id="0005-1-0-6" text="on">
    <minAreaRect>
      <Point x="1872" y="1007" />
      <Point x="1791" y="1007" />
```

Imagine a vast sheet of paper on which

Imagine

Figure 3.10: Samples from CVL forms, lines and words datasets together with the XML metadata.

written by any individual writer is distributed more evenly than in the IAM database, which can be seen in Figure 3.11.

3.4.3 IMGUR5k handwriting dataset

IMGUR5k handwriting dataset is a new dataset created by Krishnan et al. (2021), which features challenging image data of handwritten text written by many different authors. Compared to CVL and IAM, the dataset features more authors and includes much more varied background and handwriting. In total, the dataset consists of 8903 images by approximately 5305 authors. Together, the images have 135,375 words. The annotations of the images include diverse special characters, which in our experiments have been normalized into characters of the set **[a–z0 – 9]**, for example **ü** → **u**.

The annotations of the images do not only include positions of the bounding boxes, but also the angles of the boxes, in order to support highly challenging data such as a line of text written in the shape of a circle.

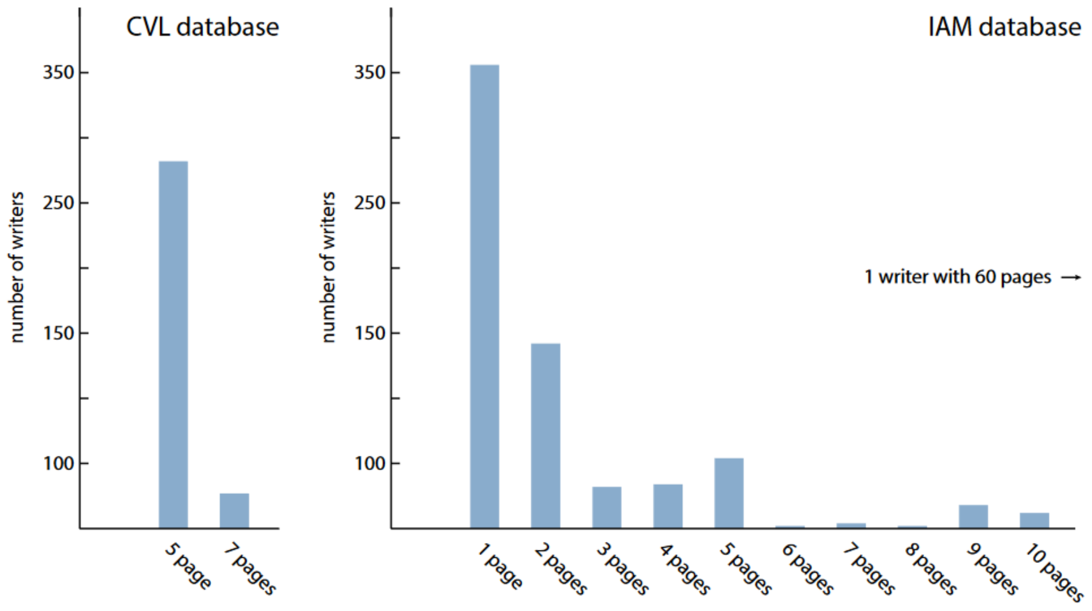


Figure 3.11: Distribution of writers in CVL and IAM databases, image taken from Kleber et al. (2013)

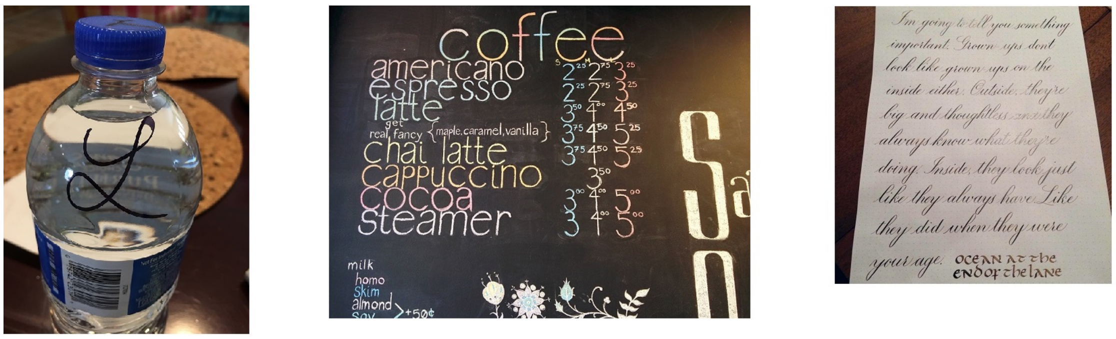
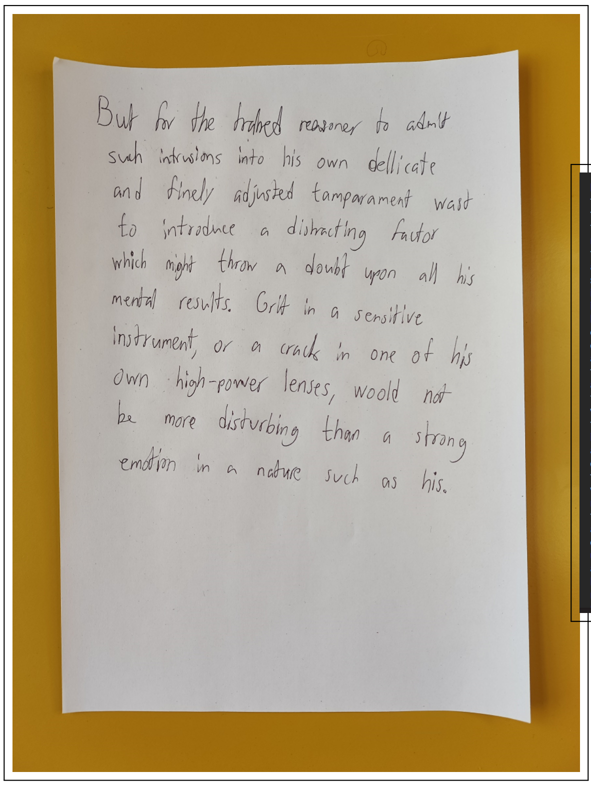

coffee
americano
espresso
latte
real fancy {maple, caramel, vanilla}
chai latte
cappuccino
cocoa
steamer
milk
homo
skim
almond
soy
+50¢
I'm going to tell you something important. Grown ups don't look like grown ups on the inside either. Outside, they're big and thoughtless and they always know what they're doing. Inside, they look just like they always have. Like they did when they were your age.
OCEAN AT THE END OF THE LANE

Figure 3.12: Samples from IMGUR5k handwriting dataset, Krishnan et al. (2021)

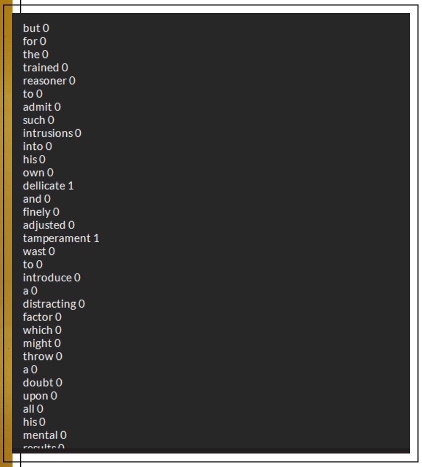
3.4.4 Custom dataset

In order to evaluate the developed system for the intended task, we gathered a small custom handwritten text dataset containing artificial spelling errors. The dataset consists of 12 pages, 833 words where of 75 are misspelled words. For each page, we created a target file containing a word on each row together with its class: *{0=correctly spelled, 1=incorrectly spelled}*. A sample from the dataset can be viewed in Figure 3.13, with the input on the left and targets on the right.

As of now, the dataset lacks sufficient diversity, with all writers being male (Beech and Mackintosh, 2005) computer science students, which is seen as a weakness of the dataset.



But for the trained reasoner to admit
such intrusions into his own dellicate
and finely adjusted tamparament wast
to introduce a distracting factor
which might throw a doubt upon all his
mental results. Grit in a sensitive
instrument, or a crack in one of his
own high-power lenses, woold not
be more disturbing than a strong
emotion in a nature such as his.



```
but 0
for 0
the 0
trained 0
reasoner 0
to 0
admit 0
such 0
intrusions 0
into 0
his 0
own 0
dellicate 1
and 0
finely 0
adjusted 0
tamperament 1
wast 0
to 0
introduce 0
a 0
distracting 0
factor 0
which 0
might 0
throw 0
a 0
doubt 0
upon 0
all 0
his 0
mental 0
```

Figure 3.13: Example image from the custom created dataset together with part of its transcript.

Chapter 4
Approach

In this chapter, we detail our approach to the system development. This includes a description of the system architecture and the motivation behind its components. Furthermore, approaches to pre and post processing techniques to improve results within the system are also discussed.

4.1 System Architecture

We divide the system architecture into multiple components, where each component can be worked on in parallel and evaluated independently. Figure 4.1 shows an overview of the system architecture, which consists of modules for video capturing, text detection, text recognition, post processing and spell correction. Below, we discuss the motivation behind each module.

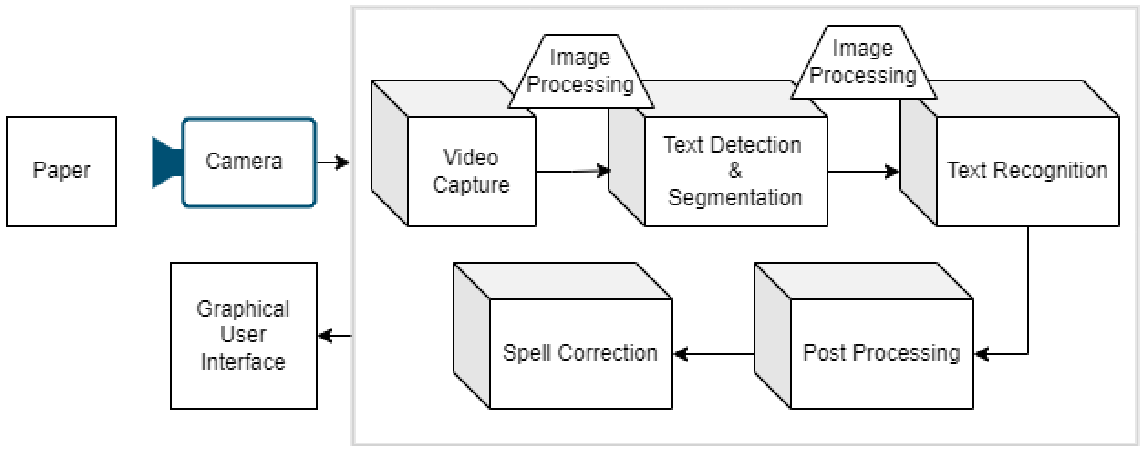


Figure 4.1: System architecture.

4.1.1 Video Capture Module

The first step in the system pipeline is the capture of live video from a specified video source. Since the output and feedback from the system are displayed in the captured frames, we want the capturing and displaying of frames to be run independently of the rest of the modules to avoid large delays between displayed frames. To solve this, we utilize multithreading, which enables us to capture and display frames in a separate thread while the heavy computational tasks are run in the background. Python provides various libraries for image capture and our system utilizes the open source library OpenCV (Open Source Computer Vision Library) as it is easy to use and popular.

4.1.2 Text Detection and Segmentation Module

The text detection and segmentation module is responsible for detecting and segmenting text in a given input image. Looking at existing text detection models, we require a model fast enough to run in real time on consumer grade CPUs with state-of-the-art performance on handwritten text detection. For this purpose, we decided to test and evaluate the two models *EAST* and *YOLO* discussed in Section **??**. These models also perform word predictions compared to text line predictions which is deemed more suitable as the bounding box for each word then can be utilized throughout the system pipeline. We can then more easily make sure that potential spell corrections end up at the correct positions in the image. It also simplifies the text recognition step as we can more easily segment and pass each word through the text recognition model. One downside with this method however, is that semantic relationships are lost as each word is processed separately. This is further discussed in Section 4.1.3.

4.1.3 Text Recognition Module

The text recognition module is responsible for performing the handwritten text recognition (HTR) task. Given a list of all detected and cropped out words in the frame, the module outputs the recognized word in each crop. For the system to be able to perform spell correction, the HTR model needs to have adequate performance in predicting both incorrectly and correctly spelled words, which can limit which HTR models are suitable for the system. Analogously to the aspects of the text detection module, inference time is an important factor that is considered. In this section, we discuss our approach to the HTR problem and how we deal with the requirements of our system.

For the aspect of performing adequately on both correctly and incorrectly spelled words, different neural network architectures are compared in Sect. 6. More specifically, we compared the models:

- Three different models trained for scene text recognition.
- TrOCR discussed in section 3.2.1.
- AttentionHTR discussed in section 3.2.2.
- Our own model trained similarly as AttentionHTR based on the Unified four-stage STR framework described in 3.2.2.

Every model except TrOCR processes each cropped out word independently and doesn't take any context into consideration. This means that potential spelling errors won't be automatically corrected because the model figured out the word from its context. Along with the recognized string prediction, a confidence score is returned, indicating how confident the model is in its prediction.

For TrOCR, we process the text line by line instead of word by word. Because TrOCR can utilize learnt language knowledge, it performs much better when it can utilize the context of each word. We wanted to examine how this would affect the number of misspelled words detected compared to the other models. How we went from individual cropped out words to text lines is discussed in 4.1.4.

Utilizing Previous Predictions

Since the system runs in near real time on a live video feed, the same words will be predicted over and over again. In a perfect scenario, we would only run the recognition model on new words written by the user or when an existing word was modified. This would greatly affect inference time and reduce the number of unnecessary computations.

To approach this, the system stores the center positions (x, y) of the currently detected words together with the consecutive predictions made at each point. To allow for some leeway and reduce the sensitivity of this approach, (x, y) is scaled down to $x = [0, 40]$, $y = [0, 60]$. These previous predictions are then utilized in two ways.

Firstly, a limit of 5 is set as the maximum number of consecutive predictions at each position. This means that unless the size of the bounding box changes or the paper is moved, there will be at most 5 predictions at each position. Secondly, the prediction with the highest confidence score at each position is always returned. This reduces flickering between different predictions between frames and makes the system more stable.

4.1.4 Post Processing Module

The post processing module is used to improve the output from the text recognition module. We experimented with three different post processing techniques which are explained in this section.

Improving the Predictions with TrOCR

As discussed in Section 3.3, a very common way to improve the text recognition accuracy is to incorporate a language model and introduce the context into the recognition process. In our scenario, we want to maintain as many of the accurately identified spelling errors as possible while reducing the number of errors produced by the text recognition model. To accomplish this, we experimented with also running the cropped out words through the TrOCR model. As discussed in Section 3.2.1, TrOCR can take lines of text as input and, with its incorporated language model, make use of the context in its predictions. However, due to the fact that our text detection model performs word segmentation rather than line segmentation, certain additional steps are required to construct the text lines that TrOCR can take advantage of.

Firstly, for simplicity we make the assumption that the text is written left to right, top to bottom and the cropped out words are not inherently sorted in this manner. This means that the detected bounding boxes first need to be sorted to be able to correctly construct the text lines with words appearing in the correct order. To achieve this, the system uses the algorithm explained below. The results of the algorithm can be seen in Figure 4.2.

1. Calculate the average box height h_{mean} of the detected bounding boxes.
2. Find the box i with the lowest y_{center} coordinate.
3. Find all boxes k, where $(y^{(k)}_{center} - y^{(i)}_{center}) \leq h_{mean}$.
4. Sort the found boxes by x_{min} value, low to high.
5. Append the boxes to the results and remove from the original collection.
6. Repeat step 2-5 until the original collection is empty.

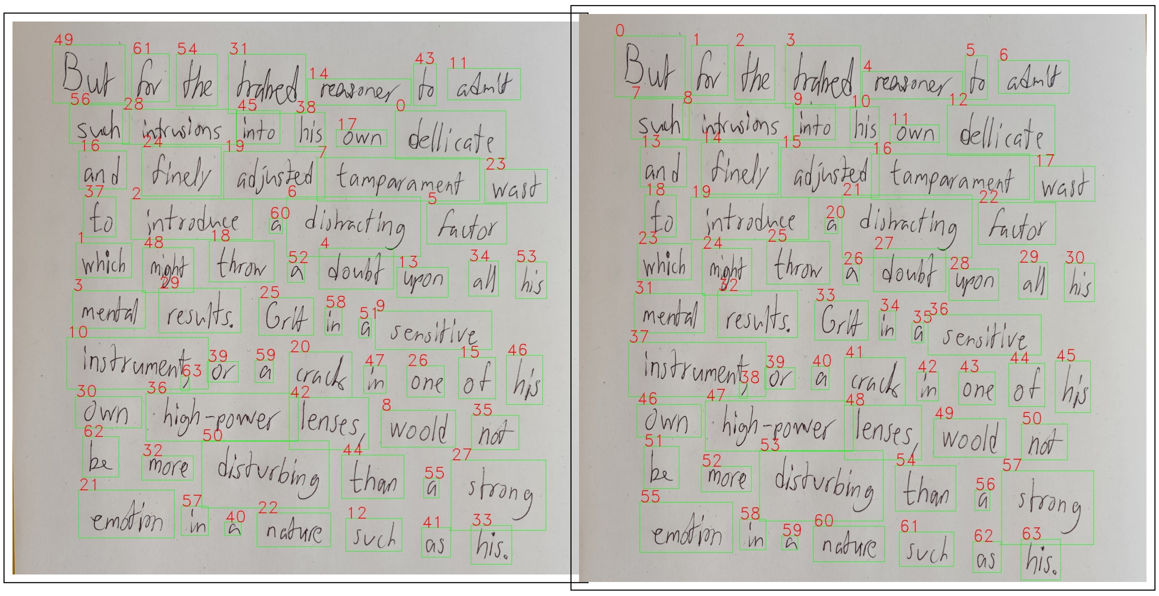


Figure 4.2: Result of the sorting of bounding boxes.

After the sorting, the cropped out words are concatenated into lines of text. To avoid text lines that are too long for TrOCR to manage, a maximum of 10 words are concatenated per line. To handle crops of different sizes, all crops are padded bottom and top to match the height of the tallest crop. The result is displayed in Figure 4.3.

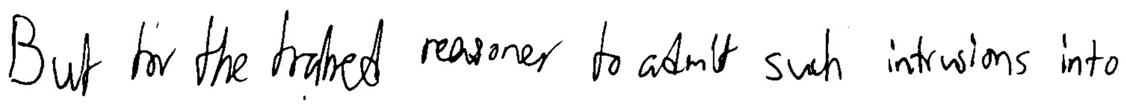


Figure 4.3: Concatenated words to a text line.

The concatenated text lines are then passed to TrOCR which outputs its prediction of the text. By then comparing the output with the predictions from the recognition module, we try to reduce the number of errors produced from the system while maintaining as many correctly identified spelling errors as possible. This is accomplished by looking at the confidence score for each of the recognition module's predictions. If the confidence score is greater than a preset threshold (0.2), the prediction is used. Else, the edit distance between the two predictions is calculated and if it is below 3, TrOCR's prediction is used. If both the confidence score is low and edit distance high, the text recognition's prediction will still be used.

Improving the Predictions with a Language Model

Another more straight forward approach is to pass the complete output from the text recognition module through a language model that can correct misspelled words. Similarly to when using TrOCR, the detected words first need to be sorted left to right, top to bottom, so that the entire text can be constructed with the words appearing in the correct order.

As the language model, we used the BertChecker class available in the Neuspell (Jayanthi et al., 2020) toolkit. BertChecker uses a BERT (Devlin et al., 2019) model specifically trained for context sensitive spelling correction in English. Given a sentence, the class will try to replace incorrectly spelled words with a suitable word given the context. We therefore pass the complete output from the text recognition module to the BertChecker class and receive a complete transcript without any spelling errors. The transcript is then used in the same way as when using TrOCR. The BertChecker is trained specifically for correction in English but this approach could be applied to other languages if the language model was trained for multilingual correction.

Improving the Predictions with Lexicon

The final and most simple approach is using a lexicon to replace incorrect words with low confidence. For this approach, we simply load a lexicon into the system of the language the user has currently selected. We then iterate over the outputs from the text recognition module and if an output has a confidence value below the set threshold (0.2), we try to replace the output with the word in the lexicon with the lowest edit distance. To find the word, we utilize Symspell which is further discussed in the next section.

4.1.5 Spell Correction Module

The spell correction module is implemented to detect spelling mistakes in the recognized text and to generate spelling suggestions for the found errors. In this section, we describe two different approaches to solving this problem, more specifically the approach of using a dictionary and a deep learning language model.

SymSpell

SymSpell is an algorithm which evolved from the classical approach famously implemented by Norvig (SeekStorm, 2021), but originally created by Kernighan et al. (1990). SymSpell achieves faster time complexity by solely depending on delete operations to cover the range of words reachable. Other approaches usually require transpose, insert and replace operations which are expensive, but in Symspell, these are transformed into deletes of the dictionary term. SymSpell uses a corpus to generate a Python dictionary, which is a lexicon of all terms found in the corpus pointing towards the term frequency. For this research we use the python implementation Symspellpy v.6.7.

The correction module iterates through the output from the post processing module and finds the word with the closest edit distance to each output. If the edit distance is not equal to 0, the output is not in the loaded lexicon and it is determined to be a spelling mistake. Outputs containing numbers are skipped as they are not present in the lexicon but shouldn't be considered as spelling errors.

To find a suggestion from the dictionary, the word with the shortest edit distance is selected. If more than one word is found with this edit distance in the dictionary, the one with the highest term frequency in the corpus is selected as suggestion.

NeuSpell

Another approach is to use a language model to detect spelling errors. Compared to the dictionary approach described above, a language model is also able to detect words that doesn't fit the context. This means the model is capable of finding syntactic mistakes, where a word would correctly exist in an dictionary but is used incorrectly. To explore this approach, we utilized the BertChecker model by Jayanthi et al. (2020) in a similar way as was done in the post processing module. First, the entire output from the post processing module was passed as input to BertChecker. We then compared the output of BertChecker with its input. If a word was replaced by another, it is marked as "incorrectly spelled" by the spell correction module.

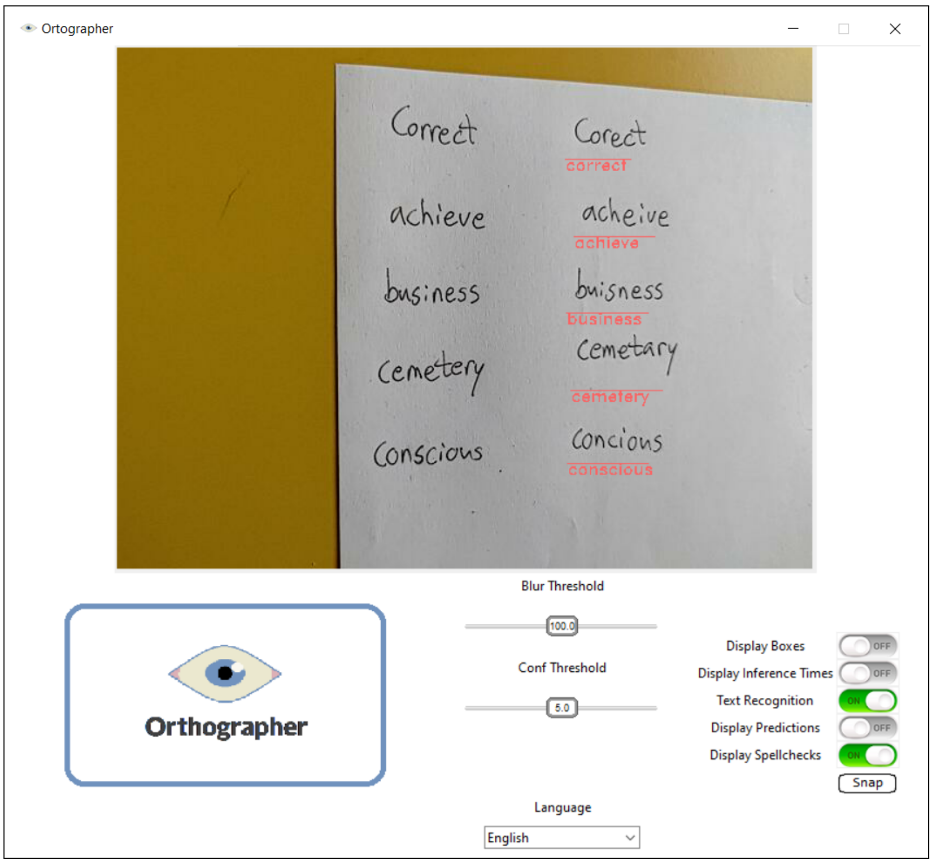
4.1.6 Graphical User Interface

To be able to display the results from the system, we developed a simple graphical user interface (GUI) using the Tkinter framework. The GUI is shown in Figure 4.4 and includes functionality to view what the different modules of the system produce. It gives the user the ability to toggle the display of inference times, bounding boxes of detected words, recognized words, spelling errors as well as spell suggestions.

Furthermore, the user can choose to turn off text recognition in order to only run text detection, freeing up memory and processing resources. To be able to adapt the system for the setup, we also implemented sliders to manually be able to vary the threshold value for blur detection, explained later on in Section 4.2.1, and the confidence threshold value used by the text detection module.

4.2 Image pre processing

Working with live image data, the concept "garbage in, garbage out" is highly relevant. Captured frames can easily be distorted, be blurry, lack illumination, or include unwanted data (noise). To counter this, multiple pre processing techniques and tools are available to increase the quality of the data fed through the system and to make sure it adheres to the standards that the different modules expect. In this section, we describe the pre processing techniques used in our system and how they affect our input data.



Blur Threshold 100.0
Conf Threshold 5.0
Display Boxes OFF
Display Inference Times OFF
Text Recognition ON
Display Predictions OFF
Display Spellchecks ON
Language English

Figure 4.4: The systems graphical user interface.

4.2.1 Blur Detection

When capturing frames of someone writing a handwritten note, some motion blur is easily introduced by the person moving the paper or by simply moving the hand while writing. To make sure that the image fed forward from the video module to the text detection module is sharp, we apply a blur detection step. There are multiple ways to perform blur detection and our system employs the easy and straightforward scheme proposed by Bansal et al. (2016). The proposed algorithm runs the Laplacian operator over the gray scaled image and then calculates the variance of the response. If the variance is below a predefined threshold, the image is deemed blurry. The reason this method works is due to the characteristics of the Laplacian operator. The Laplacian in two dimensions is defined by

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2},$$

where x and y are the Cartesian coordinates of the xy plane. The Laplacian measures the 2nd derivative of an image which means it is useful for detecting rapid intensity change, hence its frequent use for edge detection in image processing. Looking at the variance of the response, a high value would suggest that there are a high density of edges and non-edges in the image. A low value would instead indicate that there are few edges in the image and we consider the image blurry.

The downside of this method is that the predefined threshold value has to be set manually and is highly dependent on the domain, light, and camera setup. As can be seen in Figure 4.4, we included a slider for this threshold value to facilitate the tweaking the value while running the program.

4.2.2 Thresholding

With varied illumination and uncertain background and text colors, some type of thresholding can be applied to help distinguish between background and foreground pixels. For our system, we wanted to set the background pixel values to 255 (white paper background), and foreground pixel values to 0 (black text). There exist different thresholding methods to accomplish this and in Figure 4.5, we can see the results after applying global thresholding, adaptive Gaussian thresholding and Otsu's binarization.

When applying thresholding to the entire image, we found that adaptive Gaussian thresholding yielded the best results for our system. Otsu's binarization is too easily influenced by pixels outside the paper and simple thresholding requires manual adjustment of the threshold value depending on illumination in different regions. Adaptive thresholding works great because it automatically determines the threshold for each pixel based on the region around it. This means different regions have different threshold values which gives better results when illumination is varying.

4.2.3 Image Alignment using Principal Component Analysis

Compared to digital text, handwritten text isn't necessarily always perfectly aligned. The paper might be twisted or the words might be written slightly rotated. To gain the most performance out of our text recognition model, we want to make sure that each word in the paper is aligned correctly. In the paper *Automatic Image Alignment Using Principal Component Analysis*, Ur Rehman and Lee (2018) propose an algorithm to automatically align images using principal component analysis (PCA). In their paper, they perform the alignment on MRI scans of the brain and on the EMNIST dataset (Cohen et al., 2017) containing handwritten digits. Below, we explain how the algorithm is implemented in our system and display some of its results. The underlying math definitions for each step are thoroughly discussed in the original paper and therefore omitted in our description. We summarize it here:

1. Given a binary image of a cropped out word, we first construct a matrix $\boldsymbol{X}$ containing N rows of the positions $[col, row]$ of all pixels with the value 0 (text pixels).

2. We compute and subtract the mean for the respective columns. This ensures that the first principal component expresses the direction of largest variance.

3. We calculate the covariance matrix $\boldsymbol{K_{XX}}$ of $\boldsymbol{X}$.

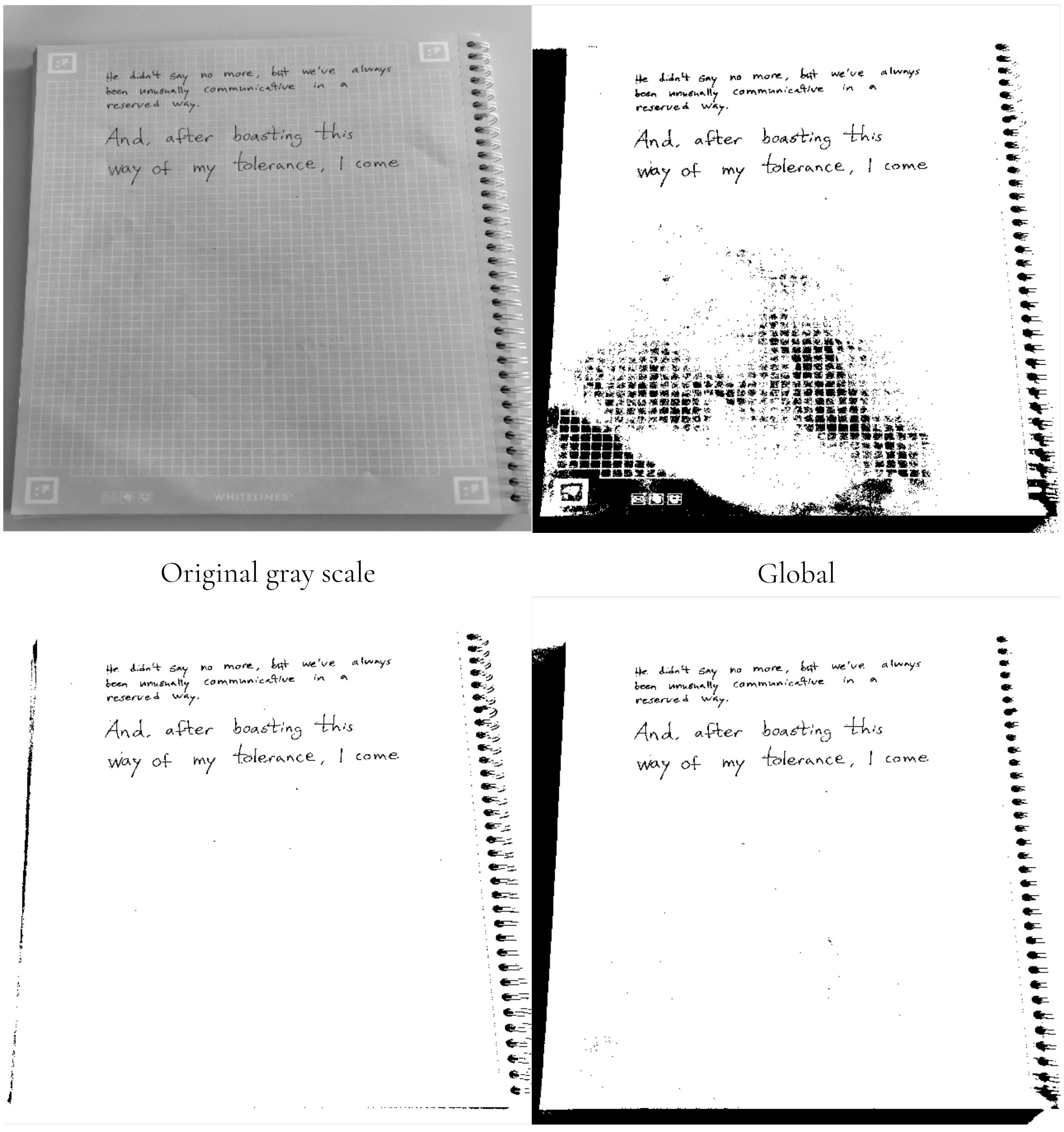
4. We compute the eigenvectors $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ and eigenvalues λ_1, λ_2 of the covariance matrix $\boldsymbol{K_{XX}}$. Using $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, we create the matrix $\boldsymbol{A}$ with the eigenvector corresponding to the largest eigenvalue in the first column and the other in the second column.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} e_1 = [a_{11}, a_{21}]^T, e_2 = [a_{12}, a_{22}]^T$$

5. The rotation angle can now be calculated. We suppose the rotation is less or equal to $90°$ which is ensured by the second line. In the third line below, we determine if the image should be rotated left or right by looking at the direction of the eigenvectors.

$$\theta = \arccos(\frac{tr(A)}{2})$$
$$\theta = \begin{cases} 180° - \theta, & \text{if } \theta > 90° \\ \theta, & \text{otherwise} \end{cases}$$
$$\theta = -\text{sign}(a_{11} \times a_{22})\theta$$

Results from running the algorithm are displayed in Figure 4.6. As can be noted in the figure, the algorithm performs much worse when the word characteristics are tall and short.


He didn't say no more, but we've always been unusually communicative in a reserved way.
And, after boasting this way of my tolerance, I come
WHITELINES

Original gray scale

Global

Adaptive Gaussian

Otsu's binarization

Figure 4.5: Three different thresholding methods.

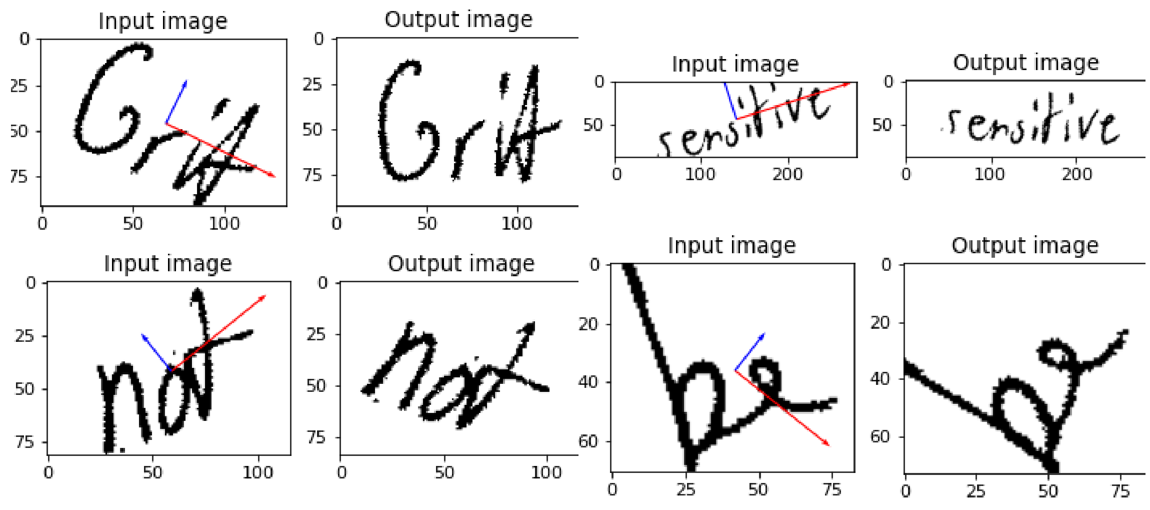


Figure 4.6: Results of running the PCA alignment with the calculated principal components drawn in the input image.

Chapter 5

Experiments

5.1 Finetuning YOLO for Handwritten Text Detection

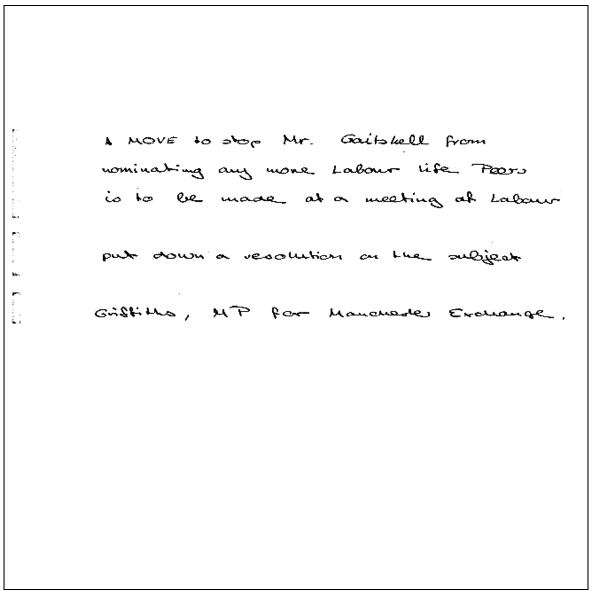
Since the available pretrained YOLOv5 models are trained on the COCO dataset which doesn't contain text, the models can't be used out of the box in our system for handwritten text detection. To fine-tune the YOLO model for handwritten text detection, we implemented a Google Colab notebook where we trained it on the IAM Forms dataset displayed in Figure 3.9.

To match the input shapes that YOLO expects, the data first had to be processed. For the input images, we stripped away all digital text, performed a simple thresholding and resized them to the shape 640×640. As discussed in Section 3.4.1, the IAM data contains some lines with "bad segmentation". To avoid training the model on bad data, we therefore stripped away these lines and removed the associated labels. Labels of special characters were also removed as we wanted the model to solely detect words.

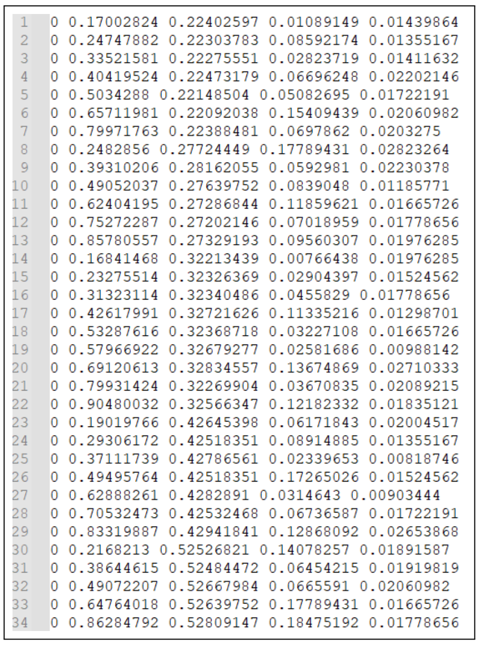
To construct the labels, we then extracted the x_{min}, x_{max}, y_{min}, y_{max} coordinates for every remaining word in the IAM XML sheet and then calculated the normalized x_{center}, y_{center}, $width$, and $height$ for each word. An example from the processed data is displayed in Figure 5.1 and as can be noted, YOLO also expects a class label among the targets which is represented by the 0 in the first column. We see that the input image has had some lines removed due to bad segmentation and now consists of 34 word and 34 labels.

From the processed data, we then created the Aachen train/val/test split discussed in Section 3.4.1. To track and visualize the training process, the machine learning platform Weights & Biases (Biewald, 2020) was utilized. Looking at the plots from the training process in Figure 5.2, we can see that the YOLOv5s model successfully learns to identify handwritten text. The mean average precision score (mAP_0.5) reaches 97% and there are no signs of overfitting looking at the validation loss.

We performed the finetuning on the nano, small and large version of the YOLO model



A MOVE to stop Mr. Gaitskell from
nominating any more Labour life Peers
is to be made at a meeting of Labour
put down a resolution on the subject
Griffiths, MP for Manchester Exchange.



```
0 0.17002824 0.22402597 0.01089149 0.01439864
0 0.24747882 0.22303783 0.08592174 0.01355167
0 0.33521581 0.22275551 0.02823719 0.01411632
0 0.40419524 0.22473179 0.06696248 0.02202146
0 0.5034288 0.22148504 0.05082695 0.01722191
0 0.65711981 0.22092038 0.15409439 0.02060982
0 0.79971763 0.22388481 0.0697862 0.0203275
0 0.2482856 0.27724449 0.17789431 0.02823264
0 0.39310206 0.28162055 0.0592981 0.02230378
0 0.49052037 0.27639752 0.0839048 0.01185771
0 0.62404195 0.27286844 0.11859621 0.01665726
0 0.75272287 0.27202146 0.07018959 0.01778656
0 0.85780557 0.27329193 0.09560307 0.01976285
0 0.16841468 0.32213439 0.00766438 0.01976285
0 0.23275514 0.32326369 0.02904397 0.01524562
0 0.31323114 0.32340486 0.0455829 0.01778656
0 0.42617991 0.32721626 0.11335216 0.01298701
0 0.53287616 0.32368718 0.03227108 0.01665726
0 0.57966922 0.32679277 0.02581686 0.00988142
0 0.69120613 0.32834557 0.13674869 0.02710333
0 0.79931424 0.32269904 0.03670835 0.02089215
0 0.90480032 0.32566347 0.12182332 0.01835121
0 0.19019766 0.42645398 0.06171843 0.02004517
0 0.29306172 0.42518351 0.08914885 0.01355167
0 0.37111739 0.42786561 0.02339653 0.00818746
0 0.49495764 0.42518351 0.17265026 0.01524562
0 0.62888261 0.4282891 0.0314643 0.00903444
0 0.70532473 0.42532468 0.06736587 0.01722191
0 0.83319887 0.42941841 0.12868092 0.02653868
0 0.2168213 0.52526821 0.14078257 0.01891587
0 0.38644615 0.52484472 0.06454215 0.01919819
0 0.49072207 0.52667984 0.0665591 0.02060982
0 0.64764018 0.52639752 0.17789431 0.01665726
0 0.86284792 0.52809147 0.18475192 0.01778656
```

Figure 5.1: Processed IAM Forms dataset used as training data for YOLO. Input image on the left, labels on the right.

and ran all training for 100 epochs. The models are evaluated and compared later in Chapter 6.

5.2 Finetuning Scene Text Recognition models for Handwritten Text Recognition

The Scene Text Recognition (STR) framework created by Baek et al. (2019) has implemented procedures to finetune pretrained models. Additionally, the authors have made pretrained STR models publicly available. An issue with training models for handwritten text recognition (HTR) is the scarcity of available annotated data, while there are much more resources available for STR. For this reason, we conducted an experiment with finetuning a pretrained STR model for HTR using this framework.

Initial experiments showed promising results, where we finetuned the STAR-Net model on the IAM words dataset. Better results than our initial experiments were achieved by Kass and Vats (2022), who swapped the prediction stage of the STAR model from CTC to the attention-based alternative also implemented by Baek et al. (2019). Additionally, they trained two models with different approaches. They trained one model for the purpose of achieving a high accuracy for the IAM dataset, by changing the training procedure to first finetuning the STR model to the IMGUR5k dataset, and then finetuning it again to the IAM dataset. Their other approach was to combine the two datasets into one, in order to train a general model.

The IMGUR5k dataset includes a lot of special characters, some in UTF-16 encoding, such as the $\sqrt{\ }$ character, in addition to a lot of diacritic characters (with for examples apostrophes or umlauts). We opted to translate these to their ASCII counterpart using the

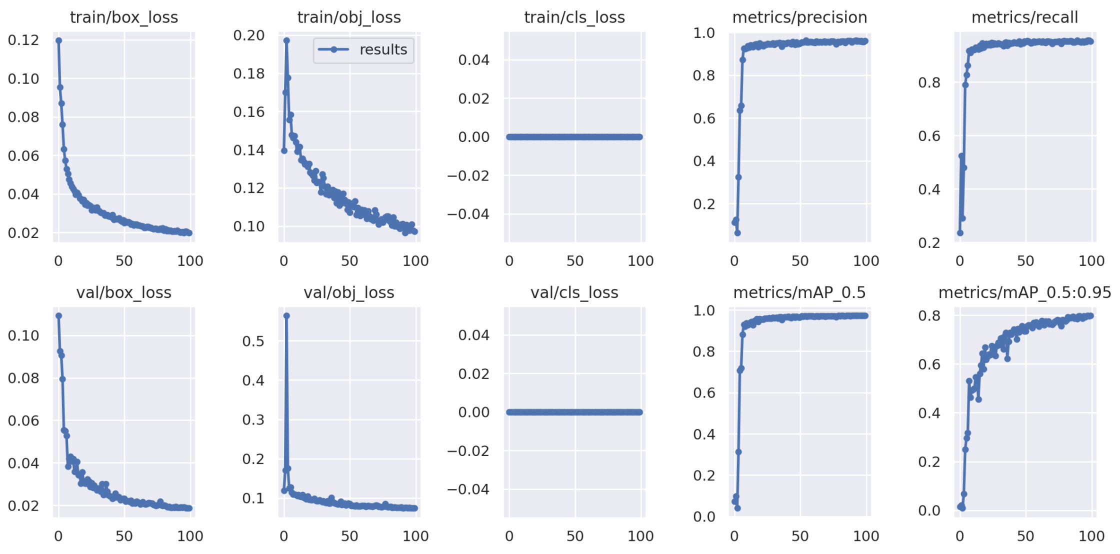


Figure 5.2: Visualization plots from the YOLOv5s training.

characters that the pretrained models were trained for, which is **[a−z0 − 9]** by using the python library unidecode. For most cases this leads to suitable translations such as é → e and å → a, but some less prevalent characters in the dataset such as ∘ and $\sqrt{\quad}$ have less suitable translations, namely ∘ → deg and $\sqrt{\quad}$ → sqrt. This was a decision made to retain the most amount of samples as possible, as data scarcity is an aforementioned difficulty when it comes to HTR model training.

As the other handwritten text datasets we use have cropped out words of the dataset available, the same was done for IMGUR5k by us, in order to facilitate re-usability of code. Using the angle from the bounding box annotations, the cropped words where then rotated using this angle from the annotation files.

We combined and used the IAM, CVL and IMGUR5k databases, specifically using the cropped words from each respective database to finetune the pretrained STAR-net model by Baek et al. (2019). Thus we follow the second approach by Kass and Vats (2022) in order to train a HTR model for the purpose of general recognition, with expectations of not reaching state-of-the-art results on the IAM Aachen split test set. The cropped images are naturally of varying sizes, and the STR framework by Baek et al. (2019) resizes all inputs to **100x32**. This works well for bigger images, but heavily distorts very small images such as images of a single letter or punctuation marks. For this reason, we opted to instead pad input images of smaller size to **100x32**, while we resize larger images to **100x32**. Two examples of the data processing can be seen in Figure 5.4.

In order to be used with the STR framework developed by Baek et al. (2019), the dataset was split into the Aachen split discussed in Section 3.4.1 and then converted into Lightning-Memory Mapped Database (lmdb) format. An lmdb dataset is suited for multi-processed environments, which was used to speed up training.

During limited manual optimization of stage configuration and regularisation, we later deactivated the transformation stage as observations showed that it didn't necessarily translate seamlessly from STR to HTR, as can be seen in Figure 5.3. From this change, we observed positive change in inference time, but not necessarily an increase or decrease in recog-

Before transformation
After transformation

Figure 5.3: Samples of handwritten text before and after TPS transformation stage.

nition performance. Due to this change, the model architecture no longer corresponds to the architecture of STAR-net. For this reason, this model is subsequently in the thesis referenced as OrthoNet. We also conducted minor experiments with adding minor dropout $\{p = 0.1, p = 0.05\}$ to all layers of ResNet. The main consequences of this was that training sessions being able to run for longer before overtraining, which we observed as a positive impact.

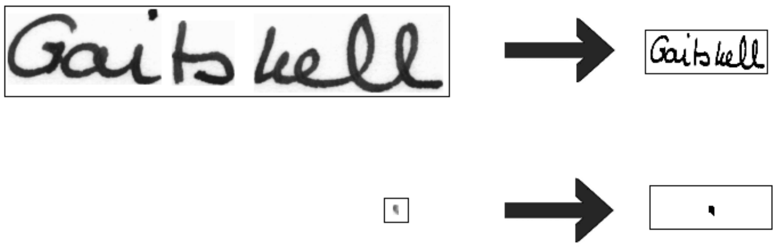


Figure 5.4: Processing of IAM Words for OrthoNet finetuning

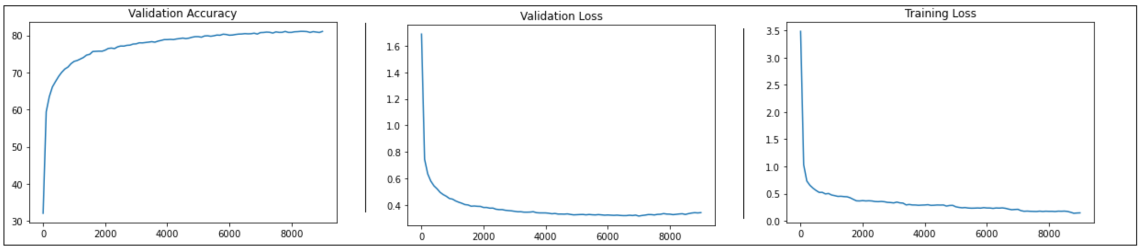


Figure 5.5: Visualization plots from the OrthoNet training.

Chapter 6
Results

Since the system uses a pipeline containing multiple modules, it's important to be able to evaluate each module separately to find and determine what's working and what's not. This allows us to more easily find weaknesses in the system and also lets us see how different changes affect the different parts of the system.

To accomplish this, we constructed two different Jupyter Notebooks, one for the text detection module, one for the text recognition module. To benchmark the entire system we also wrote a python script. In the sections below, we discuss how we constructed the respective benchmark procedures. All benchmarks were run on on the same machine, running an Intel Core i7-9750H CPU @ 2.60GHz and a Geforce GTX 1660 Ti GPU. All inference times displayed in the results are the calculated average inference times.

6.1 Text Detection Evaluation

To evaluate and compare the performance of different text detection solutions, we ran the models on the test set from our processed IAM Forms dataset displayed in Figure 5.1 and described in Section 5.1.

As metrics, we calculated the precision, recall and inference time per word. To determine whether a prediction is correct or not, we calculated the intersection-over-union (IOU) value which measures the extent of overlap between two bounding boxes. The IOU value is between 0 and 1.0, where a higher value means a greater region of overlap. We decided to use a threshold where we recognized a prediction as correct if $IOU > 0.5$ with one of the target boxes. Figure 6.1 displays an example of how IOU is calculated, while Table 6.1 shows the results from the text detection benchmark.

Looking at the results, we can tell that all finetuned YOLOv5 models clearly outperform the EAST model, both in regards to inference time and accuracy. Between the YOLOv5 variants, the difference is minimal with YOLOv5n having a slightly lower inference time and YOLOv5l a slightly higher recall and precision.

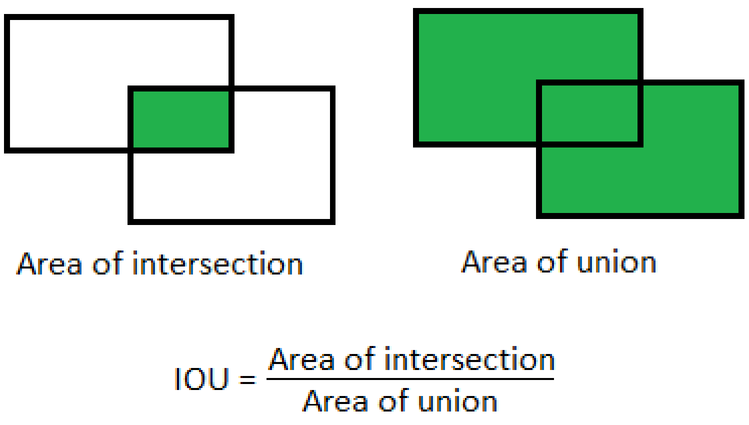


Figure 6.1: Formula to calculate IOU.

Model	Precision %	Recall %	Inference Time (ms)
EAST	67.7	56.6	3.966
YOLOv5n	95.9	94.4	**0.814**
YOLOv5s	96.0	95.4	0.918
YOLOv5l	**96.2**	**96.1**	1.075

Table 6.1: Text detection benchmark results. Inference time is calculated per word.

6.2 Text Recognition Evaluation

To evaluate and compare the performance of different text recognition models, we ran the models on the test set of the IAM words dataset described in Section 3.4.1. Since the TrOCR model is able to perform predictions on text lines, we also ran the different TrOCR variants on the test set of the IAM lines dataset to see how the performance changed.

As metrics, we computed the average character error rate (CER), the average word error rate (WER), and inference time per word. CER and WER are calculated as:

$$CER = \frac{(S + D + I)}{N_1},$$
$$WER = \frac{(S + D + I)}{N_2},$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, N_1 is the number of characters in the reference and N_2 the number of words in the sequence. The CER works at the character level and can be seen as the percentage of characters that were incorrectly predicted. WER works at word level and can be seen as the percentage of words that were incorrectly predicted. A score of 0 is a perfect score.

Tables 6.2 and 6.3 show the results from the text recognition benchmarks. Looking at the results, the AttentionHTR-IAM model outperforms the other models in regards to CER and WER on the IAM words dataset. OrthoNet performs slightly worse than AttentionHTR-

IAM but has more than **50%** lower inference time. Comparing the results with that of Table 6.3, we can also note that the TrOCR variants perform very poorly on single words but achieve top results in regards to CER and WER when given lines of text containing multiple words.

Model	**CER %**	**WER %**	**Inference Time (ms)**
DenseNet_CTC	52.72	86.71	**5.69**
ResNet_CTC	35.00	68.66	55.33
CRNN_VGG_BiLSTM_CTC	34.38	63.68	17.58
TrOCR_small	22.59	35.65	62.22
TrOCR_base	26.89	42.60	131.84
TrOCR_large	15.03	26.73	232.56
OrthoNet	6.24	18.72	15.30
AttentionHTR-IAM	**5.08**	**14.56**	37.44
AttentionHTR-General	6.13	16.03	37.39

Table 6.2: Text recognition benchmark results on IAM words. Columns specify the model used, character error rate, word error rate and inference time per word.

Model	**CER %**	**WER %**	**Inference Time (ms)**
TrOCR_small	6.42	15.52	**185.89**
TrOCR_base	4.78	12.35	337.86
TrOCR_large	**3.44**	**10.25**	405.83

Table 6.3: Text recognition benchmark results on IAM lines. Columns specify the model used, character error rate, word error rate and inference time per line.

6.3 System Evaluation

Finally, we constructed a way to measure the performance of the entire system. Since the true purpose of the system is to find spelling errors, we needed a handwritten dataset containing spelling errors so we could measure the percentage of errors that the system finds, as well as how often the system incorrectly finds spelling errors. To accomplish this, we created our own benchmark dataset discussed in Section 3.4.4. We then ran the dataset through the system pipeline and measured the F1-score, recall and precision of the detected spelling mistakes. We also calculated the character error rate on the output from the post processing module as well as the each module's total inference time per word.

To calculate the spelling mistake metrics, we constructed two different sets M_{true} and M_{pred} for each sample in the dataset. M_{true} contained all true spelling mistakes and M_{pred} contained all spelling mistakes that the system found. Recall, precision and F1-score was then calculated as:

$$Recall = \frac{M_{pred} \cup M_{true}}{M_{true}}$$

Model	Source	Explicit LM	Character-based	IAM CER %
Transformer	Li et al. (2021)	-	-	2.9
CNN + BLSTM + CTC	Bluche and Messina (2017)	X	-	3.2
MDLSTM + CTC	Voigtlaender et al. (2016)	X	-	3.5
MDLSTM + MLP/HMM	Castro et al. (2018)	X	-	3.6
CNN + BLSTM + Attn	Kass and Vats (2022)	-	X	4.3
MDLSTM + CTC	Bluche (2015)	X	-	4.4
CNN + LSTM + CTC	Puigcerver (2017)	X	-	4.4
MDLSTM + Attn	Bluche (2016)	X	-	4.4
Transformer	Kang et al. (2022)	-	-	4.6
LSTM + HMM	Doetsch et al. (2014)	X	-	4.7
LSTM + HMM	Voigtlaender et al. (2015)	X	-	4.8
CNN + LSTM + Attn	Michael et al. (2019)	X	X	4.8
CNN + CTC	Yousef et al. (2020)	-	X	4.9
CNN + LSTM + Attn	Coquenet et al. (2022)	-	X	4.9
LSTM + HMM	Pham et al. (2014)	X	-	5.1
MDLSTM + CTC	Gui et al. (2018)	X	-	5.1
CNN + BLSTM + Attn + CTC	Dutta et al. (2018)	-	-	5.1
CNN + BLSTM	Kang et al. (2021)	-	-	5.7
CNN + BGRU + GRU + Attn	Huang et al. (2020)	X	-	5.7
CNN + CTC	Bluche et al. (2017)	-	-	6.1
CNN + BLSTM + CTC	Ours	-	X	6.2
Transformer	Singh and Karayev (2021)	-	X	6.5
MDLSTM + CTC	Kang et al. (2018)	X	-	6.6
CNN + BGRU + GRU	Chowdhury and Vig (2018)	-	-	6.8
CNN + BLSTM + LSTM	Kozielski et al. (2013)	-	-	8.1
GMM/HMM	Sueiras et al. (2018)	X	-	8.2
CNN + LSTM + Attn	Krishnan et al. (2018)	-	-	8.8
CNN + LSTM + CTC	España-Boquera et al. (2011)	-	-	9.7
MLP/HMM	Chen et al. (2017)	X	-	9.8
MDLSTM + CTC	Dreuw et al. (2011)	X	-	11.1
MLP/HMM	Poulos and Valle (2021)	X	-	12.4
CNN + BLSTM + GRU + Attn	Louradour and Kermorvant (2013)	-	X	16.6
MDLSTM + CTC	Ogiela and Jain (2012)	-	-	17.0

Table 6.4: Reported state of the art text recognition on the test sets of IAM Aachen split.

$$Precision = \frac{M_{pred} \cup M_{true}}{M_{pred}}$$

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

We ran the benchmark on different configurations of the system to be able to compare models within each module. All benchmarks are based on the following default configuration:

- Text Detection: YOLOv5s
- Text Recognition: OrthoNet
- Post processing: None
- Spell Correction: Lexicon (Symspell)

In Table 6.5 we see how the different text detection models compare to each other. We can see that the YOLO models clearly leads to better system performance compared to the EAST model. Between the YOLO models, the difference is minor with the small YOLO model yielding the best results.

In Table 6.6, the different text recognition models are compared. Note that TrOCR runs on lines created from concatenated words, as discussed in section 4.1.4. The table shows that the OrthoNet model yields the best results on our system in all columns, followed by the AttentionHTR models. TrOCR fails to find spelling mistakes and shows a very poor F1 score.

In Table 6.7, we compare the different techniques used in the post processing module. Compared to using no post processing, we see that all techniques slightly decreases recall while increasing precision and F1 score. TrOCR is the slowest, requiring 25.7ms per word, while using a lexicon (Symspell) is the fastest, running at 0.044ms per word.

Finally, in Table 6.8, we see how the techniques for spell correction compares. The results show that using a Lexicon yields better results and runs significantly faster than using a language model.

Text Detection	CER %	F1 %	Recall %	Precision %	Inf (ms)
EAST	25.16	10.05	21.85	6.81	6.51
YOLOv5n	6.65	52.71	67.31	47.45	**1.94**
YOLOv5s	**6.21**	**55.64**	**69.72**	**50.21**	1.98
YOLOv5l	6.74	52.58	66.59	47.74	2.84

Table 6.5: System benchmark results with different text detection models.

Text Recognition	CER %	F1 %	Recall %	Precision %	Inf (ms)
AttentionHTR-General	6.97	52.61	67.04	46.44	37.75
AttentionHTR-IAM	7.34	52.41	66.98	46.29	37.15
OrthoNet	**6.21**	**55.64**	**69.72**	**50.21**	**14.03**
TrOCR_small	10.64	9.91	6.25	25.0	25.70
TrOCR_base	12.23	22.86	17.63	38.03	51.18
TrOCR_large	12.74	17.18	12.0	27.50	67.11

Table 6.6: System benchmark results with different text recognition models.

Post Processing	CER %	F1 %	Recall %	Precision %	Inf (ms)
TrOCR_small	**5.89**	57.12	68.68	52.45	25.70
Lexicon (Symspell)	5.92	**60.26**	68.68	**56.91**	**0.044**
Language model (BertChecker)	6.22	57.58	68.68	53.24	4.87

Table 6.7: System benchmark results with different post processing techniques.

6.3.1 Total Inference Time

When measuring the end-to-end inference time of the system with the best solution of each module according to the benchmarks configuration, we get an average inference time of

Spell Correction		**F1 %**	**Recall %**	**Precision %**	**Inf (ms)**
Lexicon	(Symspell)	**55.64**	**69.72**	**50.21**	**0.061**
Language Model	(BertChecker)	50.47	67.64	44.19	4.80

Table 6.8: System benchmark results with different spell correction techniques.

16.3156ms per word. The measurement is taken based on 5 different runs of the system benchmark.

Chapter 7

Discussion

In this chapter, we discuss the results presented in Chapter 6 and the insights gained during the project. We also bring up limitations of the developed system and discuss some future work.

7.1 Result Findings

7.1.1 Text Detection Module

As handwritten text detection is not an active research area, there are no clear state-of-the-art research to compare the finetuned YOLO models with. However, as expected, the finetuned YOLO models perform significantly better than Zhou et al. (2017), presented in Table 6.1, both with regards to accuracy and inference time. Kass and Vats (2022) argue that finetuning from scene text recognition to handwritten text recognition is a valid approach due to the scarcity of annotated handwritten document data, and we argue the same for handwritten text detection as the same databases can be used.

The YOLO models reach a 96% recall and precision score and only take 1ms to run per word. Therefore, we do not view the text detection module as a bottleneck for the system or as the most critical module on which to spend efforts. However, we believe there could be possible improvements if further research is conducted on finetuning YOLO object detection for handwritten text detection. We predict that training the models with more data, for example with the IMGUR5k and CVL datasets, or by using data augmentation would lead to improvements in performance.

7.1.2 Text Recognition Module

As shown in Table 6.4, many different architectures and solutions are currently relevant in text recognition. However, there are significant difficulties in comparing research results

(Poulos and Valle, 2021). Of existing research projects, some are dependent on domain-specific dictionaries or use explicit language models to post process raw predictions. Furthermore, recent promising results use a transformer architecture for text recognition. Such an architecture integrates an implicit language model as decoder unit and has a bias to produce predictions which conforms to grammatical and spelling conventions. Information that is missing in Table 6.4 is whether an implicit language model is integrated in the model, which is the case for the transformer architectures in the table. Another important aspect to take into account is whether the models are case-sensitive and whether special characters are filtered out from the test set. Comparably to the general model trained by Kass and Vats (2022), our model is also trained for general purposes, and not specifically trained to score as high as possible on the IAM dataset. For this reason, we expected to not score higher than the AttentionHTR-IAM model.

There is also a significant number of errors in the IAM dataset, which becomes an important detail for comparing results, especially as the CER metric in state of the art is reaching perfection. Some papers report results with different initial conditions, i.e. the input is already segmented in lines or not, faulty segmentations are filtered out, etc., which makes the comparison difficult. We believe this could be an explanation as to why the benchmarked results in Table 6.2 of the models by Li et al. (2021) and Kass and Vats (2022) are worse than the results they present in their respective papers. This should be taken into account when evaluating the results we present.

7.1.3 General System Evaluation

Looking at the results from the system evaluation shown in Tables 6.5, 6.6, 6.7 and 6.8, we can see how different models compare to each other on our created dataset. As discussed in Section **??**, the dataset lacks diversity and is too small to make any general conclusions about the system. However, we can use it to compare with the results acquired on the other benchmarks and to get an intuition of how the system performs when images are passed through the entire pipeline.

Starting with the results in Table 6.5, we see that the finetuned YOLO models lead to significantly better system performance compared to the EAST model. This is expected and supported by the results acquired from the text detection evaluation. If we also compare the CER with the results from the text recognition evaluation, the YOLO models lead to an error rate very close to OrthoNet's CER on IAM. This suggests that YOLO successfully detects close to all words in the system dataset, or that the handwriting in the dataset is easier for OrthoNet to recognize than the handwriting in the IAM dataset.

In Table 6.6, we see that models like TrOCR isn't suited for tasks like detection of spelling mistakes. TrOCR's inherent language model causes predictions to adhere to norms of the learnt language, meaning that misspelled words often are automatically corrected from the context. Consequently, TrOCR achieves a lower F1-score and recall than the character-based models such as OrthoNet and AttentionHTR. In Table 6.7 we see how the different post processing techniques affect the final result. The results show that the post processing module increases the precision score while preserving close to the same recall. This indicates that predictions with low confidence from the recognition module usually contain errors that the post processing module corrects. However, CER is not affected as positively as we expected. This suggests that the post processing module fails to predict the correct word most of the time.

Finally, Table 6.8 indicate that to determine what is incorrectly spelled and not, it's better to use a lexicon than a language model for our system. This result is not necessarily true because of how our dataset was created. Creating the dataset, we used a dictionary created by the SymSpell library to decide whether a word is misspelled or not. This results in an unfair comparison, as SymSpell has been used to decide the targets of the dataset. The language model might also deem words as misspelled based on the context. This means that if words from the post processing appear in the incorrect order, it might flag words as misspelled even though each word is independently correctly spelled. This might explain that we see a lower precision score in the results when using the language model.

To further improve the recall of the system, we would have to improve the recognition module or change the post processing approach. With our approach, we rely on the recognition module to detect as many spelling mistakes as possible and then use the post processing module to try to filter out the true spelling mistakes. Therefore, the processing module will never increase the recall of the system. This logic is supported by the results in Table 6.7.

Looking at the achieved end-to-end inference time when using the top performing solutions from each module, the entire system only takes 16.3156ms to run per word. This is very close to the inference time we get by adding the inference time of each module from the tables: 1.94 + 14.03 + 0.044 + 0.061ms = 16.075ms, indicating that very little time is needed for passing the data between the modules. This is seen as a great accomplishment as it allows the system to run in real time without needing a lot of computational resources.

Chapter 8

Conclusion

In this thesis, we presented an end-to-end system for real-time detection of spelling mistakes in handwritten notes. Given a live video feed, the system is able to detect and recognize handwritten text and mark words that contain spelling errors.

The system consists of a pipeline with modules for video capture, text detection, text recognition, post processing and spelling correction. Our experiments show that transfer learning can successfully be utilized for data-efficient training, both for handwritten text detection and for handwritten text recognition. Furthermore, it suggests that implicit word level language models are not suitable for handwritten text recognition where spelling mistakes should be conserved, while character-based text recognition models conserve more spelling mistakes. The results show that the post processing module has a positive impact on the system by effectively correcting mistakes made by the recognition module. Running the created test set through the system pipeline, the system detects $\sim 69\%$ of the spelling errors, achieves a character error rate of $\sim 6\%$, and runs at ~ 16.3ms per word.

For future work, the system is viewed as a framework where each module can be worked on, improved and evaluated independently. The modules can easily be adjusted to use other model architectures, making it easy to incorporate other models into the system. The code and models are therefore available at this GitHub repository:
`https://github.com/viktorkar/orthographer`. Suggested improvements include making the detection module adaptable to a wider range of notes, improving the recognition module to be able to recognize more characters and decreasing the error rate. Additionally, there are multiple parameters to optimize in the system, as well as in the training of OrthoNet. Potential future use cases for the system could be as a tool in smart-glasses, as a supportive tool for the visually impaired or as a tool to facilitate learning how to write.

8.1 Future Work

Based on the limitations set on the system, several suggestions for future work can be discussed.

Firstly, since the handwritten notes are assumed to only contain text, written left to right and top to bottom, the system will not perform as well on notes containing more than text, or with text not following this assumed structure. It's likely that figures and tables would be detected as text by the detection module and consequently marked as incorrectly spelled after being passed through the pipeline. Therefore, future work should include making the system adaptable to a wider range of notes. For this, an approach similar to Singh and Karayev (2021) might be of interest, where their system is able to distinguish between different regions of text, math expressions, tables, and figures. Unfortunately, these authors did not publish their dataset.

Secondly, since the system is only able to recognize the characters $[a{-}z0-9]$, the number of languages for which the application can be used are restricted. This limitation also leads to bad performance on words containing special characters. For example, "I'm" would be recognized as "im", and because "im" is not a real word in the English dictionary, it is flagged as a spelling mistake if a dictionary approach is used for spell correction. Future work thereby should include improving the text recognition module to be able to recognize special characters and work with other languages.

Thirdly, our approach to make use of previous predictions requires that the camera and paper are perfectly still so that the position of each detected word remains stationary. For the system to work with, for example smart glasses, in the future, this technique would have to be adjusted. One possible solution could for example be to first apply an object tracking model that can detect the outlines of either the paper in the image or bounding boxes of text and keep track of objects between frames.

References

Baek, J., Kim, G., Lee, J., Park, S., Han, D., Yun, S., Oh, S. J., and Lee, H. (2019). What is wrong with scene text recognition model comparisons? dataset and model analysis. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 4714–4722.

Bahdanau, D., Cho, K., and Bengio, Y. (2015). Neural machine translation by jointly learning to align and translate. 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Bansal, R., Raj, G., and Choudhury, T. (2016). Blur image detection using laplacian operator and open-cv. In *2016 International Conference System Modeling Advancement in Research Trends (SMART)*, pages 63–67.

Bao, H., Dong, L., Piao, S., and Wei, F. (2022). BEit: BERT pre-training of image transformers. In *International Conference on Learning Representations*.

Beech, J. and Mackintosh, I. (2005). Do differences in sex hormones affect handwriting style? evidence from digit ratio and sex role identity as determinants of the sex of handwriting? *Personality and Individual Differences*, 39.

Biewald, L. (2020). Experiment tracking with weights and biases. Software available from wandb.com.

Bluche, T. (2015). *Deep Neural Networks for Large Vocabulary Handwritten Text Recognition*. PhD thesis, Université Paris-Sud.

Bluche, T. (2016). Joint line segmentation and transcription for end-to-end handwritten paragraph recognition. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 838–846, Red Hook, NY, USA. Curran Associates Inc.

Bluche, T., Louradour, J., and Messina, R. (2017). Scan, attend and read: End-to-end handwritten paragraph recognition with mdlstm attention. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 1050–1055.

Bluche, T. and Messina, R. (2017). Gated convolutional recurrent neural networks for multilingual handwriting recognition. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 646–651.

Castro, D., L. D. Bezerra, B., and Valença, M. (2018). Boosting the deep multidimensional long-short-term memory network for handwritten recognition systems. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 127–132.

Chamidu, S. (2020). Yolo v4 or yolo v5 or pp-yolo? `https://towardsdatascience.com/yolo-v4-or-yolo-v5-or-pp-yolo-dad8e40f7109`. Accessed 2022-03-02.

Chen, Z., Wu, Y., Yin, F., and Liu, C.-L. (2017). Simultaneous script identification and handwriting recognition via multi-task learning of recurrent neural networks. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01.

Choi, H., Kang, M., Kwon, Y., and Yoon, S. (2019). An objectness score for accurate and fast detection during navigation. *CoRR*, abs/1909.05626.

Chowdhury, A. and Vig, L. (2018). An efficient end-to-end neural model for handwritten text recognition.

Cohen, G., Afshar, S., Tapson, J., and van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2921–2926.

Coquenet, D., Chatelain, C., and Paquet, T. (2022). End-to-end handwritten paragraph text recognition using a vertical attention network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Doetsch, P., Kozielski, M., and Ney, H. (2014). Fast and robust training of recurrent neural networks for offline handwriting recognition. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 279–284.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Dreuw, P., Doetsch, P., Plahl, C., and Ney, H. (2011). Hierarchical hybrid mlp/hmm or rather mlp features for a discriminatively trained gaussian hmm: A comparison for offline handwriting recognition. In *2011 18th IEEE International Conference on Image Processing*, pages 3541–3544.

Dutta, K., Krishnan, P., Mathew, M., and Jawahar, C. (2018). Improving cnn-rnn hybrid networks for handwriting recognition. In *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 80–85.

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

Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. (2006). Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural 'networks.

Gron, A. (2017). *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition.

Gui, L., Liang, X., Chang, X., and G Hauptmann, A. (2018). Adaptive context-aware reinforced agent for handwritten text recognition. In Shum, P. H., Hubert, and Hospedales, T., editors, *29th British Machine Vision Conference, BMVC 2018*. British Machine Vision Association and Society for Pattern Recognition. British Machine Vision Conference 2018, BMVC 2018 ; Conference date: 03-09-2018 Through 06-09-2018.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

Huang, X., Lisheng, Q., Yu, W., Li, J., and Ma, Y. (2020). End-to-end sequence labeling via convolutional recurrent neural network with a connectionist temporal classification layer. *International Journal of Computational Intelligence Systems*, 13.

Jayanthi, S. M., Pruthi, D., and Neubig, G. (2020). NeuSpell: A neural spelling correction toolkit. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 158–164, Online. Association for Computational Linguistics.

Johansson, S., Atwell, E., Garside, R., and Leech, G. (1986). The tagged lob corpus : user's manual.

Kang, L., Riba, P., Rusiñol, M., Fornés, A., and Villegas, M. (2022). Pay attention to what you read: Non-recurrent handwritten text-line recognition. *Pattern Recognition*, 129:108766.

Kang, L., Riba, P., Villegas, M., Fornés, A., and Rusiñol, M. (2021). Candidate fusion: Integrating language modelling into a sequence-to-sequence handwritten word recognition architecture. *Pattern Recognition*, 112:107790.

Kang, L., Toledo, J. I., Riba, P., Villegas, M., Fornés, A., and Rusiñol, M. (2018). Convolve, attend and spell: An attention-based sequence-to-sequence model for handwritten word recognition. In Brox, T., Bruhn, A., and Fritz, M., editors, *Pattern Recognition - 40th German Conference, GCPR 2018, Stuttgart, Germany, October 9-12, 2018, Proceedings*, volume 11269 of *Lecture Notes in Computer Science*, pages 459–472. Springer.

Kass, D. and Vats, E. (2022). Attentionhtr: Handwritten text recognition based on attention encoder-decoder networks. In Uchida, S., Barney, E., and Eglin, V., editors, *Document Analysis Systems*, pages 507–522, Cham. Springer International Publishing.

Kernighan, M. D., Church, K. W., and Gale, W. A. (1990). A spelling correction program based on a noisy channel model. In *Proceedings of the 13th Conference on Computational Linguistics - Volume 2*, COLING '90, page 205–210, USA. Association for Computational Linguistics.

Kleber, F., Fiel, S., Diem, M., and Sablatnig, R. (2013). Cvl-database: An off-line database for writer retrieval, writer identification and word spotting. In *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, pages 560–564.

Kozielski, M., Rybach, D., Hahn, S., Schlüter, R., and Ney, H. (2013). Open vocabulary handwriting recognition using combined word-level and character-level language models. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8257–8261.

Krishnan, P., Dutta, K., and Jawahar, C. (2018). Word spotting and recognition using deep embedding. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 1–6.

Krishnan, P., Kovvuri, R., Pang, G., Vassilev, B., and Hassner, T. (2021). Textstylebrush: Transfer of text aesthetics from a single example.

Lee, C. and Osindero, S. (2016). Recursive recurrent nets with attention modeling for ocr in the wild. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2231–2239, Los Alamitos, CA, USA. IEEE Computer Society.

Li, M., Lv, T., Cui, L., Lu, Y., Florencio, D., Zhang, C., Li, Z., and Wei, F. (2021). Trocr: Transformer-based optical character recognition with pre-trained models.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. (2014). Microsoft coco: Common objects in context. volume 8693.

Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8759–8768.

Louradour, J. and Kermorvant, C. (2013). Curriculum learning for handwritten text line recognition. *Proceedings - 11th IAPR International Workshop on Document Analysis Systems, DAS 2014*.

Marti, U. and Bunke, H. (2002). The iam-database: an english sentence database for offline handwriting recognition.

Michael, J., Labahn, R., Grüning, T., and Zöllner, J. (2019). Evaluating sequence-to-sequence models for handwritten text recognition. *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1286–1293.

Mu, Z. (2020). Provide ocr model for opencv dnn.

Ogiela, M. R. and Jain, L. C. (2012). Computational intelligence paradigms in advanced pattern classification.

Pham, V., Bluche, T., Kermorvant, C., and Louradour, J. (2014). Dropout improves recurrent neural networks for handwriting recognition. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 285–290.

Poulos, J. and Valle, R. (2021). Character-based handwritten text transcription with attention networks. *Neural Comput & Applic*, 33.

Puigcerver, J. (2017). Are multidimensional recurrent layers really necessary for handwritten text recognition? In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 01, pages 67–72.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, Los Alamitos, CA, USA. IEEE Computer Society.

Robson, K. (1992). Accounting numbers as "inscription": Action at a distance and the development of accounting. *Accounting, Organizations and Society*, 17(7):685–708.

Rogers, A., Kovaleva, O., and Rumshisky, A. (2020). A primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8:842–866.

Rüfenacht, M. (2020). Handwritten text recognition (htr) in 2020.

SeekStorm (2021). Symspell.

Shi, B., Bai, X., and Yao, C. (2015). An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PP.

Shi, B., Wang, X., Lyu, P., Yao, C., and Bai, X. (2016). Robust scene text recognition with automatic rectification. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4168–4176.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556.

Singh, S. S. and Karayev, S. (2021). Full page handwriting recognition via image to sequence extraction. In *Document Analysis and Recognition – ICDAR 2021*, pages 55–69. Springer International Publishing.

Sueiras, J. (2021). Continuous offline handwriting recognition using deep learning models.

Sueiras, J., Ruiz, V., Sanchez, A., and Velez, J. F. (2018). Offline continuous handwriting recognition using sequence to sequence neural networks. *Neurocomput.*, 289(C):119–128.

Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jegou, H. (2021). Training data-efficient image transformers and distillation through attention. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10347–10357. PMLR.

Ur Rehman, H. Z. and Lee, S. (2018). Automatic image alignment using principal component analysis. *IEEE Access*, 6:72063–72072.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

Voigtlaender, P., Doetsch, P., and Ney, H. (2016). Handwriting recognition with large multi-dimensional long short-term memory recurrent neural networks. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 228–233.

Voigtlaender, P., Doetsch, P., Wiesler, S., Schlüter, R., and Ney, H. (2015). Sequence-discriminative training of recurrent neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2100–2104.

Wang, C.-Y., Liao, H.-y., Wu, Y.-H., Chen, P.-Y., Hsieh, J.-W., and Yeh, I.-H. (2020a). Cspnet: A new backbone that can enhance learning capability of cnn. pages 1571–1580.

Wang, Y., Xie, H., Zha, Z., Xing, M., Fu, Z., and Zhang, Y. (2020b). Contournet: Taking a further step toward accurate arbitrary-shaped scene text detection. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11750–11759.

Xu, R., Lin, H., Lu, K., Cao, L., and Liu, Y. (2021). A forest fire detection system based on ensemble learning. *Forests*, 12:217.

Yousef, M., Hussain, K. F., and Mohammed, U. S. (2020). Accurate, data-efficient, unconstrained text recognition with convolutional neural networks. *Pattern Recognition*, 108:107482.

Zhou, X., Yao, C., Wen, H., Wang, Y., Zhou, S., He, W., and Liang, J. (2017). East: An efficient and accurate scene text detector. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2642–2651, Los Alamitos, CA, USA. IEEE Computer Society.

Zhuang, L., Wayne, L., Ya, S., and Jun, Z. (2021). A robustly optimized BERT pre-training approach with post-training. In *Proceedings of the 20th Chinese National Conference on Computational Linguistics*, pages 1218–1227, Huhhot, China. Chinese Information Processing Society of China.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-08-25

EXAMENSARBETE Real-time detection of spelling mistakes in handwritten notes
STUDENTER Aston Åkerman, Viktor Karlsson
HANDLEDARE Pierre Nugues (LTH), Marcos Rubinstein (HEIG-VD)
EXAMINATOR Marcus Klang (LTH)

Real-time detection of spelling mistakes in handwritten notes

POPULÄRVETENSKAPLIG SAMMANFATTNING **Aston Åkerman, Viktor Karlsson**

The art of handwriting has been relevant for thousands of years and continues to be so in modern times. Technology that can recognize handwritten text in images could be utilized by smart-glasses, as a supportive tool for visually impaired or as a tool to facilitate learning how to write. With our thesis, we present **Orthographer**, an end-to-end handwritten text recognition system, able to detect spelling mistakes in handwritten text in real time.

What happens when you make a spelling mistake in a word processing software? The misspelled word gets marked and a suggestion is presented for what you probably had intended to write. What if there was a way to receive the same response when writing on a normal piece of paper with a normal pen? Could augmented reality glasses perhaps be utilized for this purpose? Recently, influential tech companies such as Google, Meta and NVIDIA has started increasingly presenting prototypes of augmented reality projects. Smart glasses capable of recognizing spelling mistakes in handwritten notes is one of the many futuristic opportunities that the technology of machine learning and augmented reality presents.

To even begin thinking about making such a system, it is first essential that we are able to automatically find and read handwritten text in live video. In this study, we present a system called ”Orthographer”, able to do just that. The system captures live video from a connected camera, finds where there is text in the video frames and then tries to understand what is written. If a misspelled word is found, the system displays a red mark below the words and gives a suggestion of the correct spelling.

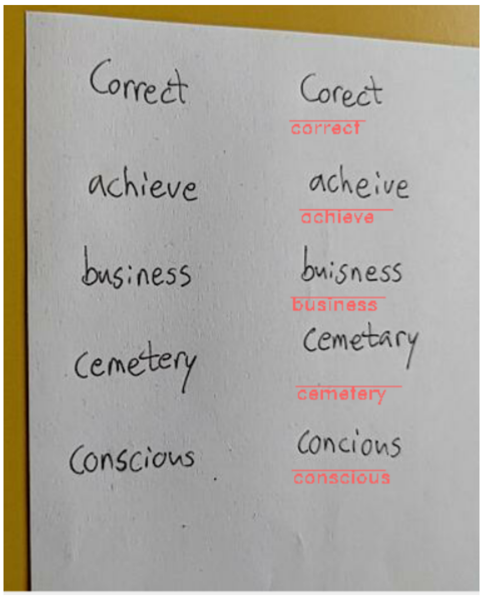


Figure 1: Orthographer detecting spelling mistakes.

The system is built around a modular architecture, where each module can be worked on, improved and evaluated independently. This facilitates future research and project improvements, as each module easily can be changed when better performing machine learning models are developed. Working in fields advancing with razor speed, this is especially valuable for facilitating others to continue researching, developing and experimenting towards a system that could be used in smart glasses or run from a normal smart phone.

Looking at the performance of the system, we can say that it achieves promising results but that there still is room for improvement. On our own test data, Orthographer is able to find $\sim 69\%$ of all spelling mistakes with a $\sim 6\%$ character error rate. Most mistakes stems from the system failing to understand our handwriting so either this is the part where there is the most room for improvement, or we simply need to improve our handwriting. I mean, the system has looked at hundreds of thousands of images of handwritten words, it probably knows how to properly write better than anyone else so who is really to blame for the mistakes?